DOCTOR OF SCIENCES

Challenging High Dimensionality in Evolutionary Optimization using Cooperative Co-evolutionary Algorithms

Blanchard, Julien

Award date:
2021

Awarding institution:
University of Namur

UNIVERSITÉ DE NAMUR

FACULTÉ DES SCIENCES

DÉPARTEMENT DE MATHÉMATIQUE

Challenging High Dimensionality in Evolutionary Optimization using Cooperative Co-evolutionary Algorithms

Thèse présentée par
Julien Blanchard
pour l'obtention du grade
de Docteur en Sciences

Composition du Jury :

Charlotte BEAUTHIER
Timoteo CARLETTI (Promoteur)
Mohammed EL-ABD
Alexandre MAYER
Annick SARTENAER (Présidente du Jury)
Daniel TUYTTENS

Juin 2021

Cover design: © Presses universitaires de Namur

Rue Grandgagnage 19
B-5000 Namur (Belgique)

Printed in Belgium.

ISBN: 978-2-39029-459-9
Registration of copyright: D/2021/1881/14

Algorithmes co-évolutionnaires coopératifs : le défi des problèmes de grande dimension au sein des processus d'optimisation évolutionnaire

par Julien Blanchard

Résumé : La grande dimensionnalité de certains problèmes d'optimisation a un impact négatif sur la capacité des algorithmes évolutionnaires à les optimiser efficacement. En effet, la complexité de ces problèmes croit exponentiellement lorsque la dimension augmente. Les algorithmes co-évolutionnaires coopératifs dépassent cette limitation en profitant des avantages de la stratégie « diviser pour mieux régner ». Ils divisent les problèmes de grande dimension en plusieurs problèmes plus petits et plus simples pouvant être optimisés avec un algorithme évolutionnaire standard. L'objectif de ce travail est d'étudier ces algorithmes co-évolutionnaires coopératifs et de développer de nouveaux outils permettant de résoudre des problèmes d'optimisation possédant des caractéristiques couramment rencontrées en ingénierie et en sciences. Une attention particulière est accordée aux problèmes contraints, aux problèmes imbriqués et aux problèmes coûteux en ressources informatiques. Les principales nouveautés des algorithmes développés dans cette thèse concernent la décomposition des problèmes de grande dimension et la coopération entre les différents sous-problèmes obtenus. Des simulations numériques sont réalisées afin de mettre en avant les avantages des outils développés.

Challenging High Dimensionality in Evolutionary Optimization using Cooperative Co-evolutionary Algorithms

by Julien Blanchard

Abstract: In evolutionary optimization, high dimensionality has a negative impact on the algorithms performance since the complexity of handled problems grows exponentially with dimensionality. Cooperative co-evolutionary algorithms overstep this limitation by enjoying the benefits of the divide-and-conquer strategy. They divide large problems into some simpler and smaller subproblems that can be optimized with a standard evolutionary algorithm. The aim of this thesis is to study these cooperative co-evolutionary algorithms and to develop new tools to allow them to tackle optimization problems with common features appearing in engineering and sciences. Particular attention is paid to constrained problems, overlapping problems and computationally expensive problems. The main innovations of the newly proposed algorithms focus on the decomposition of the large-scale problems and the cooperation between the obtained subproblems. Numerical experiments are conducted to put forward the benefits of the developed tools.

Thèse de doctorat en Sciences Mathématiques (Ph.D. thesis in Mathematics)
Date : 29/06/2021
Département de Mathématique
Promoteur (Advisor) : Timoteo CARLETTI

Remerciements

Cette thèse est le résultat d'un travail long de plusieurs années. La réussite d'un projet d'une telle envergure n'aurait pas été possible sans les contributions, directes ou indirectes, de nombreuses personnes. Il est donc temps pour moi de les remercier.

Je tiens avant tout à remercier mon promoteur, Timoteo Carletti, pour la confiance qu'il m'a offerte en acceptant d'encadrer ma thèse. Merci pour ta bienveillance et ta très grande disponibilité. Merci pour tes nombreuses relectures attentives de cette thèse ainsi que des différents articles rédigés dans le cadre de celle-ci. Merci aussi pour l'indépendance que tu m'as accordée, tout en continuant à me prodiguer de précieux conseils lorsque le besoin s'en faisait sentir. Merci encore pour tout ce que tu as pu m'apporter tout au long de la réalisation de ce travail.

J'ai également eu la chance de réaliser cette thèse en collaboration avec le centre de recherche Cenaero. À ce titre, en plus de l'encadrement offert par mon promoteur, j'ai été supervisé par Charlotte Beauthier, ingénieure de recherche à Cenaero. Je tiens à la remercier de m'avoir accompagné dans cette folle aventure de poursuivre mon mémoire, pour lequel elle m'avait déjà supervisé, en une thèse de doctorat. Merci pour tes diverses relectures et ta grande disponibilité. Merci aussi pour ton important soutien lors des moments stressants, comme mes présentations en conférence, mais également pour les moments moins stressants partagés autour d'un verre ou d'un excellent repas. Je remercie également toute l'équipe Minamo pour leur chaleureux accueil lors de mes divers passages à Cenaero en tant que stagiaire, employé et doctorant.

J'aimerais remercier Alexandre Mayer et Annick Sartenaer d'avoir accepté de faire partie de mon comité d'accompagnement et de mon jury. Merci pour vos précieuses remarques et idées partagées lors des réunions d'avancement. Merci aussi pour votre relecture très attentive de ce manuscrit. Annick, merci d'avoir accepté de présider le jury et surtout merci de m'avoir transmis cette passion "des maths et du numérique" lors des cours d'*analyse numérique* et d'*algèbre linéaire numérique*. Si je me suis dirigé vers ce sujet de mémoire qui s'est ensuite transformé en sujet de thèse, c'est en grande partie grâce à toi. Thank also to Daniel Tuyttens and Mohammed El-Abd for being members of my jury. Your comments during the private defense had been very helpful to improve the quality of this manuscript.

Merci au département de mathématique de l'Université de Namur, qui, via ma fonction d'assistant, a financé la plupart de ce travail de thèse. Merci aussi à Anne Lemaître de m'avoir octroyé la bourse d'aide au doyen pour quelques mois, ce qui a été d'une importance majeure dans la mise en place de ce projet.

Je souhaite également remercier les membres du département, qui contribuent toutes et tous, à la merveilleuse ambiance qui y règne. Merci à François, pour les bons moments partagés au bureau 133. Merci aussi d'avoir supporté la vie de bureau, parfois très mouvementée, d'un assistant. Merci à Loïc, cela a également été un plaisir de partager quelques instants avec toi au bureau, j'aspire à en partager d'autres prochainement. Merci à toute l'équipe des assistants, des plus anciens aux petits nouveaux, merci à Charlotte, Anne-Sophie, Jonathan, Delphine, Martin G., Marie M., Manon, Marie P., Morgane, Marvyn, Nicolas, Candy, Christian, Martin M. et Judicaël. En particulier, un tout grand merci à Eve-Aline, je n'aurais pas pu compter sur une meilleure Marraine, et à Pauline, pour tous les moments partagés depuis que l'on s'est lancé dans la folle aventure des mathématiques il y a plus de dix ans. Merci à Anthony, Jean-François et Riccardo, c'est toujours un plaisir de faire un petit tour en aéronef avec vous. Merci à Arnaud, François et tous les autres accros de la carte déjà cités précédemment. Merci à Alexis pour le grand suspens que tu installes avant chaque carte jouée. Merci à Joanna, la plus fidèle parmi les fidèles de l'Arsenal. Merci aussi aux deux plus beaux "coll. did." que sont Jérémy et Ambi. Merci à tout le staff académique et enseignant, j'ai eu la chance de travailler avec grand nombre d'entre vous pour les cours et/ou la recherche. Merci à Juan pour ta disponibilité et le soutien technique, ô combien précieux. Merci à Pascale et Alice, nos deux secrétaires, toujours là pour nous apporter leur aide précieuse.

Enfin, je tiens à remercier tous les miens, sans qui je ne serais la personne que je suis aujourd'hui. Un immense merci à mes parents : merci de m'avoir donné les moyens d'arriver où j'en suis aujourd'hui, merci pour les valeurs que vous m'avez transmises, merci pour le soutien inconditionnel que vous me portez et merci pour toutes les choses pour lesquelles il n'y a pas de mots. Un merci tout particulier à ma soeur Aurore et à mon beau-frère Mathieu, qui même s'ils n'ont jamais trop compris ce que je cherchais, ont toujours été là pour moi. Merci aussi à ma filleule Odile, tu fais de moi un parrain heureux et comblé. Merci à Bon-Papa, qui veille sur nous tous depuis là-haut, tu as toujours été très fier de nous et je suis sûr que tu aurais été fier de la nouvelle étape que je franchis aujourd'hui. Merci aussi à Fabienne, Vincent, Antoine, Clara et Édouard, merci pour tous les moments passés au sein de votre formidable famille. Et évidemment, un énorme et gigantesque merci à Charline, de me supporter au quotidien, mais également pour l'immense bonheur que tu m'apportes chaque jour.

À tous, merci du fond du coeur.

Julien

Contents

List of Acronyms

CC-EA	Cooperative Co-Evolutionary Algorithm (p.2)
CC-GA	Cooperative Co-evolutionary Genetic Algorithm (p.18)
CEC	Congress on Evolutionary Computation (p.4)
CPSO	Cooperative Particle Swarm Optimizer (p.54)
CVT	Centroidal Voronoi Tessellations (p.8)
DC	Divide-and-Conquer (p.17)
DSM	Design Structure Matrix (p.29)
DG	Differential Grouping (p.25)
DG2	a more advanced version of Differential Grouping (p.28)
DM-HDMR	Decomposition Method based on High Dimensional Model Representation (p.24)
EA	Evolutionary Algorithm (p.1)
EDG	Enhanced Differential Grouping (p.36)
ERDG	Efficient Recursive Differential Grouping (p.36)
FDG	Fast Differential Grouping (p.36)
FE	Function Evaluation (p.44)
FEA	Factored Evolutionary Algorithm (p.55)
FII	Fast Interdependency Identification (p.32)
FPS	Fitness Proportional Selection (p.9)
gDG	graph-based Differential Grouping (p.28)
HD-CC-EA	Hybrid Decomposition-based Cooperative Co-Evolutionary Algorithm (p.43)

HEB	High dimensional, Expensive and Black-box (p.79)
IDG	Improved Differential Grouping (p.28)
ISM	Interaction Structure Matrix (p.29)
JADE	an adaptive version of Differential Evolution (p.87)
LCVT	Latinized Centroidal Voronoi Tessellations (p.8)
LHS	Latin Hypercube Sampling (p.8)
LSC	Large-Scale Constrained (p.39)
LSGO	Large-Scale Global Optimization (p.2)
MLCC	Multilevel Cooperative Co-evolution (p.23)
OCC-EA	Overlapped Cooperative Co-Evolutionary Algorithm (p.60)
ORDG	Overlapped Recursive Differential Grouping (p.58)
RBF	Radial Basis Function (p.73)
RD-CC-EA	Random Decomposition-based Cooperative Co-Evolutionary Algorithm (p.43)
RDG	Recursive Differential Grouping (p.33)
RDG2	a more advanced version of Recursive Differential Grouping (p.35)
RDG3	an even more advanced version of Recursive Differential Grouping (p.57)
RDG3-CC-EA	Cooperative Co-Evolutionary Algorithm relying on the RDG3 decomposition (p.62)
SACC	Surrogate-Assisted Cooperative Co-evolutionary (p.79)
SACC-EAM	Surrogate-Assisted Cooperative Co-Evolutionary Algorithm of Minamo (p.80)
SACC-EAM-II	another Surrogate-Assisted Cooperative Co-Evolutionary Algorithm of Minamo (p.92)
SACC-JADE	Surrogate-Assisted Cooperative Co-Evolutionary Algorithm based on JADE optimizer (p.87)
SA-EAM	Surrogate-Assisted Evolutionary Algorithm of Minamo (p.83)
SAO	Surrogate-assisted optimization (p.69)
SPE	Sampling Points Evaluation (p.29)
SSC	Subsubcomponent (p.38)
TRDG	Three-level Recursive Differential Grouping (p.36)
xDG	extended Differential Grouping (p.28)

Introduction

Context

In everyday life, people can enjoy the benefits of mathematical optimization tools. They are used to handle a wide range of applied problems in various areas including artificial intelligence, economics, engineering, finance, manufacturing, medicine, physics, supply chain, transportation, etc. In a car, most of the engine components are designed following an optimization procedure. For example, the waste heat recovery can be optimized using specific tools and can be valued in the form of power electricity (Rosset et al. (2018)). In medicine, using radiotherapy to treat cancerous cells also requires efficient optimization tools to maximize the radiation exposure to cancer tissue while sparring the adjacent organs at risk (Pugachev et al. (2001)). In a wider context, numerical optimization has attracted considerable attention in the last decades. This has contributed to the growth of various optimization branches. In particular, the topic studied in the present work is a research area which gathers optimization and computer science. It has been named in the literature as *Evolutionary Computing* (Eiben and Smith (2015)).

Evolutionary computing, as stated by its name, is a particular kind of computing inspired by the process of natural evolution (Darwin (1859)). The latter can be simply presented as follows. A population of individuals in a given environment that has limited resources strives for survival and reproduction. In this context, the fitness of an individual measures its proper adaptation to this environment and determines its ability to survive and reproduce. In evolutionary computing, the same principles are extended to problem solving through trial-and-error testing. To solve a problem, we start with a set of candidate solutions playing the role of individuals. Their quality/fitness, i.e., how well they solve the problem, determines the chance to be kept in the candidate solutions set and also the chance to be used to build new candidate solutions. From the second half of the twentieth century, various algorithms were developed in the field of evolutionary computing. They are called *Evolutionary Algorithms* (EAs) and

cover different methods such as Genetic Algorithms (Goldberg (1989)), Evolutionary Programming (Yao et al. (1999)), Evolution Strategies (Beyer and Schwefel (2002)) and Differential Evolution (Price et al. (2005)).

Such EAs are effective tools to solve many real-world optimization problems, notably because they do not require the use of derivatives of the problem under scrutiny. Such information may often be impractical to obtain or unavailable, for example in the context of black-box optimization problems. Moreover, in recent years, studied problems in engineering and sciences become increasingly complex and involve an increasingly large number of decision variables. This is a source of concern for EAs since their performance declines when the dimensionality of the search space increases. In this context, new approaches have been proposed to tackle Large-Scale Global Optimization (LSGO) problems. They can be classified in two categories: non-decomposition and decomposition methods (Jian et al. (2020)). EAs based on non-decomposition methods consider all the decision variables as a whole. They rely on new search operators, multi-population strategies, self-adapting control parameters or embedding local search strategies to enhance the exploration capabilities when dealing with LSGO problems. EAs based on decomposition methods rely on the "divide-and-conquer strategy". They divide the initial LSGO problem into smaller ones which focus on smaller groups of decision variables and are therefore easier to be solved. The first EA following this approach has been introduced by Potter and De Jong (1994) and is called *Cooperative Co-Evolutionary Algorithm* (CC-EA). Since then and to this day, the CC-EAs have been further studied and improved in various EA paradigms. This compelling approach is the one considered in this thesis to further challenge high dimensionality in evolutionary optimization.

Outline

The central part of the manuscript is organized as follows. Chapters 1, 2 and 5 provide some useful background to better handle the following content. Chapters 3, 4 and 6 present our proposed CC strategies to challenge high dimensionality in evolutionary computation.

Chapter 1. This first chapter provides the general description of a genetic algorithm. It will be the EA optimizer used in most numerical experiments presented in this thesis. The general scheme of such a genetic algorithm relies on a population of candidate solutions, called individuals, that is improved in an iterative process. During this process, there are different ways to represent, select, reproduce and mutate the individuals. The most common ones are described. Finally, this chapter ends with the presentation of some approaches that allow genetic algorithms to deal with constrained optimization problems.

Chapter 2. This chapter proposes a state of the art of cooperative co-evolutionary algorithms. It introduces the general CC framework relying on the three following mainstays: decomposition, optimization and combination. Then, the two main types of CC-EAs encountered in the literature are presented. The first one

relies on multiple random decompositions, the other on a unique decomposition based on interaction detection. Such a detection deserves considerable attention and will be considered later in the thesis.

Chapter 3. Following the state of the art, our contribution on CC-EAs is developed in this chapter. It firstly consists in a hybrid decomposition-based CC-EA that further splits large subcomponents of interacting variables, even if they are non-separable. Secondly, this new CC framework is extended in order to tackle large-scale constrained optimization problems. The performance of the newly proposed approach is assessed on a constrained benchmark set specially designed for this purpose.

Chapter 4. Standard CC strategies may encounter some difficulties in tackling with problem instances composed of several components that interact with each other. In order to optimize such overlapping problems efficiently, we introduce a CC framework that performs a decomposition with overlapped variables between subcomponents. This requires developing a new decomposition strategy and adapting the exchange of information between subcomponents during the optimization. Simulation experiments illustrate the performance of the new approach compared with a standard CC one. An extended analysis of the overlapped approach on Rosenbrock function is also proposed.

Chapter 5. Optimization problems flowing from numerical simulations may be time-consuming to evaluate. In this context, it is inconceivable to solve such expensive optimization problems with EAs since they often require a high number of function evaluations. This chapter presents the baselines of surrogate-assisted optimization. This is an effective way out to tackle such problems in a reasonable amount of time. In this framework, an efficient computational model of the fitness landscape is built and is used for most of the function evaluations instead of the expensive function. This model is updated in an iterative scheme by evaluating the expensive function at some appropriate points.

Chapter 6. The CC approach introduced in Chapter 2 has been developed to tackle LSGO problems with EAs. Although this approach is effective, it is inappropriate to deal with expensive problems since it requires a relative large number of function evaluations. Moreover, the surrogate-assisted approach presented in Chapter 5 efficiently deals with expensive problems but suffers from the curse of dimensionality when large-scale problems are considered. This chapter introduces a new framework that combines these two approaches in order to optimize large-scale and expensive optimization problems.

Finally, this manuscript is closed with a conclusion that summarizes the work presented in this thesis and with some interesting perspectives for future researches.

Contributions

This thesis brings several contributions to the field of large-scale global optimization with evolutionary algorithms. They are also presented in the following publications and correspond to the results discussed in Chapters 3, 4 and 6. I fully contributed to the results of these publications by conducting the research work and trying to present it in the clearest possible way (in writing the proceedings papers but also in presenting them in the corresponding conferences).

- J. Blanchard, C. Beauthier, and T. Carletti. A cooperative co-evolutionary algorithm for solving large-scale constrained problems with interaction detection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 697–704, New York, NY, USA, 2017. ACM.
- J. Blanchard, C. Beauthier, and T. Carletti. A surrogate-assisted cooperative co-evolutionary algorithm for solving high dimensional, expensive and black box optimization problems. In *EngOpt 2018 Proceedings of the 6th International Conference on Engineering Optimization*, pages 41–52, Heidelberg, Germany, 2019a. Springer.
- J. Blanchard, C. Beauthier, and T. Carletti. A surrogate-assisted cooperative co-evolutionary algorithm using recursive differential grouping as decomposition strategy. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 689–696, Piscataway, New Jersey, USA, 2019b. IEEE.
- J. Blanchard, C. Beauthier, and T. Carletti. Investigating overlapped strategies to solve overlapping problems in a cooperative co-evolutionary framework. In *Optimization and Learning*, Cham, Switzerland, 2021. Springer International Publishing (accepted).

Moreover, the present research work relies on many implementations and numerical simulations carried out using the High Performance Computing ressources available at UNamur. I conducted all these implementations in the Minamo platform, the multi-disciplinary optimization platform developed at the applied research center Cenaero[(1)]. In particular, I implemented all the CC strategies presented in this thesis and I inserted them around the genetic algorithm and the surrogate-assisted optimization scheme already implemented in the platform. Finally, through this manuscript, I believe to provide an interesting and relevant document with a valuable bibliography to any researcher who would like to venture in cooperative co-evolutionary algorithms.

Acknowledgement

The present research benefited from computational resources made available on the Tier-1 supercomputer of the Fédération Wallonie-Bruxelles, infrastructure funded by the Walloon Region under the grant agreement n°1117545.

[(1)] http://www.cenaero.be

Chapter 1

Genetic algorithms

Genetic algorithms are meta-heuristic[(1)] optimization algorithms inspired by Darwin's theory of evolution (Sivanandam and Deepa (2008)). They were born in the 1960s in the USA. In this period, with the breakthrough of computer sciences, considerable attention has been applied to implement automated problem solving techniques inspired by natural selection and genetics. One of these implementations was carried out by John Holland, his colleagues and students, including David Goldberg, and was called genetic algorithm (Holland (1973)). Since then, many research efforts have been focused on genetic algorithms or, in a wider sense, on evolutionary computing. They are very powerful tools to tackle a wide class of problems. Their features offer a good exploration of the search space but also the exploitation of promising areas, making them interesting, successful and popular global optimizers.

This chapter introduces the description of the problem at hand. Then, it gives the general scheme and common features of genetic algorithms, as described in multiple textbooks (Goldberg (1989); Coley (1999); Eiben and Smith (2015)). It also presents different strategies to deal with constrained optimization problems (Coello (2002), Singh et al. (2016)). Finally, the genetic algorithm used to obtain the results presented in this thesis is briefly introduced.

1.1 Problem description

This section presents the overall description of the optimization problems considered in this thesis. In a general way, optimization problems occurring in engineering and design are represented with a black-box approach. It means that the analytic form of the objective function is unknown. The only available information is its value at given points. The goal of the optimization is to find the inputs that minimize

(1) See the box on page 6 for further discussion on the terms heuristic and meta-heuristic.

Heuristic or meta-heuristic ?

In the literature, genetic algorithms, and more generally nature-inspired algorithms are sometimes referred to as heuristic algorithms, other times as meta-heuristic algorithms. Basically, heuristic algorithms are alternatives to conventional optimization techniques. They are developed to solve specific problems in sciences and engineering with better computational performance at the expense of lower accuracy (Gavrilas (2010)).

The term 'meta-heuristic' was introduced in Glover (1986) to describe high level heuristics. In this context, meta-heuristic algorithms combine basic heuristic methods in order to efficiently explore the search space without requiring any special knowledge on the optimization problems to be solved. According to Blum and Roli (2003), this class of algorithms includes, amongst others, ant colony optimization, genetic algorithms, simulated annealing and tabu search.

However, the discussion on the classification of these algorithms (heuristic vs meta-heuristic) is still open. Some people may consider some characteristics of an algorithm to make it a 'high level' heuristic, some others not. Another argument often put forward to discern a meta-heuristic is its capacity of learning from its own experience during the optimization (one may think to a genetic algorithm whose mutation rate is self-adapted according to the improvement during the last iterations for example). On the contrary, a basic heuristic always repeats the same process during the whole optimization. It is often the case of local search methods.

To conclude, although some clarifications have been made, the limit between heuristic and meta-heuristic remains unclear. From my personal point of view, I would prefer referred the genetic algorithm to a meta-heuristic because it may be much more sophisticated than a basic local search and can learn from itself during the evolutionary process.

(or maximize) the objective function $f(\vec{x})$ under the constraint that the vector of inputs $\vec{x} = [x_1, x_2, \ldots, x_n]$ remains in a specified domain $\mathscr{D}$. In other words, the problem to be solved is given by:

$$\min / \max_{\vec{x} \in \mathscr{D}} f(\vec{x}). \tag{1.1.1}$$

In particular, the work presented in this thesis focuses on continuous optimization problems since we consider that each input x_i takes real values. Genetic algorithms, amongst others, can be an effective approach to solve such optimization problems covering a wide range of problems occurring in many real-world applications.

1.2 General scheme

Although there are multiple variants of genetic algorithms, the underlying structure behind these variants is the same. Genetic algorithms start from a population of guesses for the solutions, called individuals. This population is then evolved following Darwin's theory of evolution: the individuals which are the best adapted to their environment, i.e., the best fit, survive and reproduce, the others disappear. This is referred to as natural selection/survival of the fittest.

In our framework, an individual is composed by one chromosome that is a vector of genes. Depending on the context, each variable of the problem at hand is encoded in one or several genes. The whole chromosome represents a potential solution of this problem (see Section 1.3 for further details). Therefore, the initial population is a set of potential solutions that will be improved in an iterative process described in Figure 1.1. Each individual of the population is evaluated with the fitness function F. It is a function that describes the quality of the individuals: the higher the fitness value, the better the quality of the individual. In most cases, the fitness function is equal to the objective function. Therefore, good solutions have high fitness values for maximization problems and low fitness values for minimization problems.

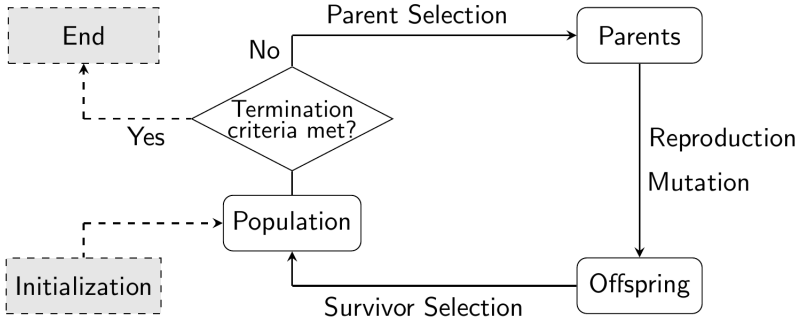


Figure 1.1 – General scheme of a genetic algorithm.

The first step of the iterative process is to select pairs of parents in the population. They are randomly selected following their fitness value: the higher the fitness value of an individual, the higher the probability to be selected as parent. Then, reproduction and mutation operators are applied to create offspring individuals. For this step, crossover operators act on pair of parents to create new individuals whose genetic information is a combination of the genetic material of their parents. After that, some individuals are mutated with a small probability. It means that their genetic information is slightly changed. The obtained individuals constitute the offspring population. Finally, some of the better individuals among the previous population and the offspring population are chosen to compose the population of the next iteration. In the context of genetic algorithm, an iteration is called a generation. The process is repeated until the termination criteria is met. Details about all operators described above are given in Section 1.4.

1.3 Representation

There are different ways to choose the genetic representation of an individual. One of the simplest and earliest representation is the binary coding (Goldberg (1989)). In this case, each gene is represented by a bit that can take the value 0 or 1. Therefore, a chromosome is a sequence of 0's and 1's that can stand for a potential solution of the problem at hand. For example, a chromosome of size 8 can represent an integer

between 0 and 255 as illustrated in Figure 1.2. Another commonly used representation is the reflected binary code, also known as Gray code (Rowe et al. (2004)). Unlike the standard binary code, it offers a representation such that consecutive integers always have Hamming distance one. This property presents diverse advantages, notably with regard to the mutation operator.

1	0	1	1	0	1	0	1

$$\mathbf{1} \times 2^7 + \mathbf{0} \times 2^6 + \mathbf{1} \times 2^5 + \mathbf{1} \times 2^4 + \mathbf{0} \times 2^3 + \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0 = 181$$

Figure 1.2 – A binary chromosome and its decimal representation.

When dealing with variable values that come from a continuous distribution, it may be more appropriate to use a real-valued representation (Wright (1991)). In this case, each gene is a real number. This kind of representation could be suitable to work with physical quantities once one cannot discretize the search space. For example, considering the design of an aircraft propeller, each gene could represent the length of the propeller blades, the tilt angle of a specific component or even parameters of a complex fluid dynamics system.

1.4 Operators

Initialization

A genetic algorithm starts from an initial population. Most of the time, its individuals are randomly generated. For a binary coding of individuals, the generation of the initial population simply consists in choosing the value (0 or 1) of each gene of each individual randomly. For a real-valued representation of individuals, the i-th gene of the individual x can be randomly set as follows:

$$x_i = L_i + \alpha(U_i - L_i), \tag{1.4.1}$$

where α is a real random number uniformly chosen in the interval $[0,1]$, L_i and U_i are, respectively, the lower and upper bounds of the domain of the i-th gene value. Some advanced sampling techniques such as Latin Hypercube Sampling (LHS), Centroidal Voronoi Tessellations (CVT) or Latinized Centroidal Voronoi Tessellations (LCVT) could also be used to generate the initial population. These techniques will be further investigated in Section 5.2.

Parent selection

The selection of individuals that will produce the offspring is a key step. On the one hand, individuals with high fitness values[(2)] must have an important probabi-

[(2)] In this section, without the loss of generality, we set ourselves in the context of maximization problems. Therefore, we consider that good solutions have high fitness values.

lity to be selected in order to spread their good genetic material to the next generation. On the other hand, some individuals with low fitness values could sometimes be selected in order to avoid premature convergence (Leung et al. (1997)). Moreover, it may happen that they offer some genetic material that could be valuable to reach promising areas. Traditional procedures for the selection of parents are the Fitness Proportional Selection (FPS), the ranking selection and the tournament selection (Eiben and Smith (2015)).

The fitness proportional selection, first introduced in Holland (1975), assigns to the i-th individual of the population the probability $P(i)$ to be selected according to the following equation:

$$P(i) = \frac{F_i}{\sum_{j=1}^{\mu} F_j}, \qquad \forall i = 1, \ldots, \mu, \tag{1.4.2}$$

where μ is the total number of individuals in the population. The probability $P(i)$ is the ratio between the fitness of the i-th individual F_i and the sum of the fitness of the μ individuals of the population. It requires that all the fitness values must be positive. An example of these probabilities is shown in Table 1.1 (columns 4 and 5) for a population of 4 individuals. In this table, two different fitnesses are considered, which leads to different probabilities. Moreover, the FPS operator is also known under the name of *roulette wheel operator*. Indeed, a roulette wheel slot can be associated to each individual whose opening angle is proportional to its fitness value. Then, to select an individual, you just have to spin the wheel and choose the individual corresponding to the slot where the ball will land. From an algorithmic point of view, the fitness proportion of each individual can be matched to a line segment in the interval $[0, 1]$. To select an individual, a random number is generated between 0 and 1 and the corresponding individual is selected.

Individual	Fitness 1	Fitness 2	Prob. with FPS for Fitness 1	Prob. with FPS for Fitness 2	Prob. with Ranking Selection $s = 1.5$
A	1	21	0.05	0.21	0.13
B	3	23	0.14	0.22	0.21
C	7	27	0.33	0.27	0.29
D	10	30	0.48	0.30	0.37

Table 1.1 – Example of selection probabilities computation for the fitness proportional selection and the ranking selection.

This mechanism has been widely studied. However, it suffers from several drawbacks. Firstly, the computed probabilities are highly dependent on the variance of the fitness. If the variance is large, an individual can have a fitness value considerably larger than the others. In this case, it will be too frequently selected and it will lead to a premature convergence of the algorithm towards a local optimum. On the contrary, if the variance is too small, the selection operator acts nearly such as a random selection. Furthermore, if the fitness values are shifted, it changes the selection probabilities

while the order between individuals remains the same. This behavior is illustrated in Table 1.1 (columns 3 and 5), fitness 2 is a +20 shift of fitness 1 (columns 2 and 4).

The ranking selection can be viewed as an alternative selection scheme to avoid inconveniences of fitness proportional fitting, i.e., discarding the absolute values for the fitness and taking into account only the relative values (Baker (1987)). Individuals are sorted from 0 to $\mu - 1$ according to their fitness value and the selection probabilities are computed on the basis of their rank i, the worst individual being at position 0, the best one at position $\mu - 1$. There are several ways to assign the probabilities from the ranks. The linear ranking is commonly used and is computed as follows:

$$P(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}, \tag{1.4.3}$$

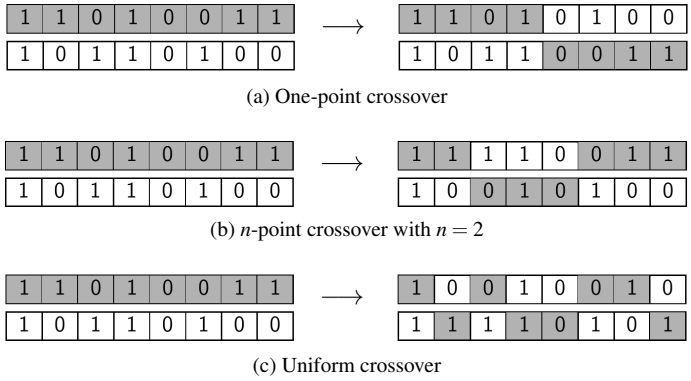
where $s \in]1,2]$. It is a parameter controlling the distribution of the probabilities. Choosing $s = 1$ would be equivalent to randomly choosing an individual in the population without any selection pressure. Moreover, setting $s = 2$ will prevent the worst individual to be selected (see Table 1.1 column 6).

The tournament selection is the most widely used selection scheme. It consists in choosing randomly k individuals in the population and selecting the best one based on the fitness values. The tournament size k enables controlling the selection pressure (Miller and Goldberg (1995)). The larger the size, the greater the probability that individuals with high fitness values will be selected. On the contrary, if the size is small, low fitness individuals have a quite large probability to be selected. When the best individual of the tournament is selected for reproduction, the term deterministic tournament is used. Another variant, called stochastic tournament, consists in choosing the best individuals with a probability $p \in]\frac{1}{2},1]$, the second best with a probability $p(1-p)$, the third one with a probability $p(1-p)^2$ and so on. This new parameter p is another tool to manage the selection pressure.

Reproduction and mutation

Reproduction and mutation operators that we can name variation operators create the diversity within the population and facilitate novelty. The reproduction allows one to produce from two individuals, called parents, one or two new individuals called children. It is carried out with a crossover. It merges the genetic information of the two parents to create the children.

For a binary coding of individuals, the most common operators are the one-point crossover, the n-point crossover and the uniform crossover (Coley (1999)). The one-point crossover is the simplest one. Parents chromosomes are cut at a random point and the tails are exchanged to create two children. The n-point crossover is a generalization of the previous one. Parents chromosomes are cut at n random points and the children are created by swapping one in two pieces of the chromosomes. Finally, the uniform crossover consists in exchanging each gene of the parents with 0.5 probability. These binary crossovers are illustrated in Figure 1.3.



(a) One-point crossover

(b) n-point crossover with $n = 2$

(c) Uniform crossover

Figure 1.3 – Binary crossovers, parents are on the left, children on the right.

For a real-valued representation of individuals, there are three kinds of crossover operators: the discrete crossover, the (whole) arithmetic crossover and the blend crossover (Eiben and Smith (2015)). The discrete crossover consists in applying the above presented rules for binary representation of individuals to the real-valued representation as shown in Figure 1.4a. However, the drawback of this very simple methodology is that the crossover does not produce any new values in the population. Indeed, each gene of a child is a gene of one of its two parents with equal likelihood. For the arithmetic crossover, the i-th gene of the two children $z_{1,i}$ and $z_{2,i}$ is a linear combination of the corresponding genes of the parents x_i and y_i:

$$z_{1,i} = \alpha_i x_i + (1-\alpha_i) y_i, \qquad \forall\, i = 1, \dots n, \tag{1.4.4}$$

$$z_{2,i} = (1-\alpha_i) x_i + \alpha_i y_i, \qquad \forall\, i = 1, \dots n, \tag{1.4.5}$$

where α_i is a real number between 0 and 1. There are several ways to choose the parameters α_i. The simplest one is to fix $\alpha_i = \alpha$, $\forall\, i = 1, \dots n$. In this case, α could be randomly chosen in the interval $[0,1]$ for each mating of parents or could be fixed to a predetermined value. In particular, if α is set to 0.5, the two children are identical as shown in Figure 1.4b. For this reason, with that kind of crossover, it is common to produce only one child starting from two parents. Such a crossover operator is represented in Figure 1.5a for two real-valued parents x and y. If α_i is fixed to the same value for the first and the second gene, the offspring lies a line segment between the two parents. In particular, if $\alpha = 0.5$, it lies in the middle of its two parents. Another way to deal with α_i parameters is to pick up a new random number between 0 and 1 for each gene as illustrated in Figure 1.4c. In this case, for two real-valued parents, the child lies in a rectangle whose vertices are determined by the position of the two parents as shown in Figure 1.5b. Finally, the blend crossover allows one to produce offspring outside of the n-dimensional box determined by the two parents (Eshelman and Schaffer (1993)). Indeed, with that kind of crossover, supposing that for the i-th gene $x_i < y_i$, the value of z_i lies in the interval $[x_i - \gamma d_i, y_i + \gamma d_i]$ where d_i represents the distance between x_i and y_i. To set the i-th gene of a child that satisfies

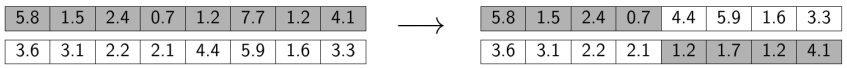
the above condition, a random number u is picked up in the interval $[0,1]$. Then α_i and z_i are computed as follows:

$$\alpha_i = (1-2\gamma)u - \gamma \quad (1.4.6)$$
$$z_i = (1-\alpha_i)x_i + \alpha_i y_i. \quad (1.4.7)$$

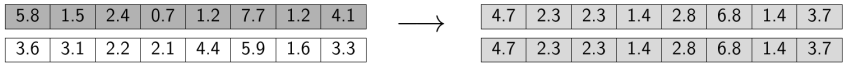
This crossover is presented for two real-valued parents in Figure 1.5c.

In a natural world, the mutation of some genes happens very rarely. However, the mutation is needed in the context of genetic algorithms. Indeed, although crossover operators effectively explore the search space, they may sometimes loose some potentially useful genetic material. The mutation operator protects against such a loss of information.

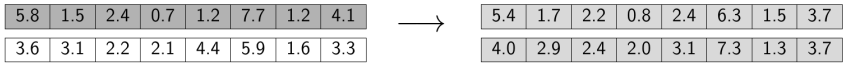
For a binary coding of individuals, the mutation operator is very simple. It allows each bit to flip from 0 to 1 or from 1 to 0 with a small probability p. An example of such a mutation is proposed in Figure 1.6, the 3 and 8-th bits have been flipped. Generally, the parameter p is fixed in such a way that on average, one bit per offspring is flipped. However, the optimal value of p is problem-dependent. A higher value of p



(a) Discrete crossover (one-point)

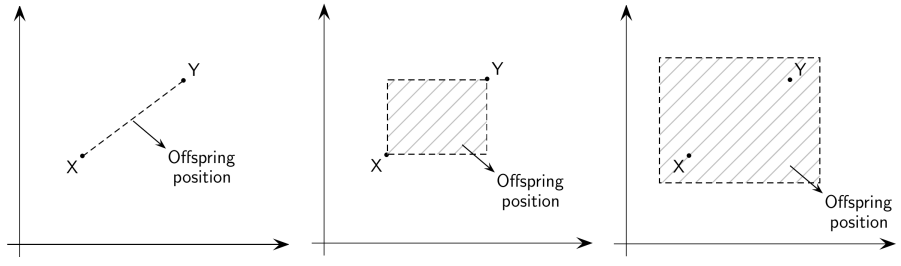


(b) Arithmetic crossover with $\alpha_i = 0.5$, $\forall\, i = 1,\ldots 8$



(c) Arithmetic crossover with different α_i values taken from this array
[0.8 0.9 0.1 0.9 0.6 0.2 0.3 0.5]

Figure 1.4 – Crossovers for real-valued representation, parents are on the left, children on the right.



(a) Arithmetic crossover $\alpha_1 = \alpha_2$ (b) Arithmetic crossover $\alpha_1 \neq \alpha_2$ (c) Blend crossover

Figure 1.5 – Crossovers for two real-valued parents.

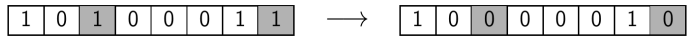


Figure 1.6 – Mutation for binary representation.

could be used to incite more exploration while a smaller one is more appropriate to boost the exploitation.

For a real-valued representation, each individual has a small probability to be mutated. In particular, if the latter is mutated, the mutation operator changes the value of each gene. Generally, the introduced perturbation is nonuniform. It is designed to, usually but not always, produce small changes of the gene value. Furthermore, it is also possible to perform large perturbations but with smaller probabilities. Such a pertubation can be built using a normal distribution. Given a gene x_i, its value after mutation is obtained as follows:

$$x_i' = x_i + \alpha(U_i - L_i), \tag{1.4.8}$$

where α is a real number randomly chosen following a normal distribution with mean zero and user-specified standard deviation σ, U_i and L_i are, respectively, the upper and lower bounds of the domain of the i-th gene value. The normal distribution is appropriate in this framework since approximately two thirds of α values will lie in the interval $[-\sigma, \sigma]$.

Survivor selection

The survivor selection, also called replacement step, carries out the composition of the population of the next generation. It is built from the population of parents and offspring. The two main approaches for the survivor selection are age-based selection and fitness-based selection (Eiben and Smith (2015)). The general idea of age-based selection is that each individual will persist in the population for a fixed number of generations. The simplest age-based selection strategy is to set this number equal to one. In this case, at each generation, the offspring population replaces the population of parents. That kind of strategy is often combined with elitism. It guarantees that the best individuals will be kept in the population for the next generation. For fitness-based selection, a very simple strategy allowing to keep a constant population size is presented. The populations of μ parents and λ children are merged and ranked according to their fitness. The best μ are kept for the next generation. This is called the $(\mu + \lambda)$ selection.

Termination

In an ideal study case, if the optimal value of the fitness function is known, the stopping criterion will be the discovery of this optimal value. However, genetic algorithms are stochastic and there is no guarantee to reach the optimum. Furthermore, the optimum value is most of the time unknown. Therefore, other stopping criteria are necessary. Usually, the algorithm is stopped when one of the below conditions is verified:

- the maximum allowed CPU time is consumed;
- the maximum number of generations/fitness evaluations is reached;
- there is no improvement of the fitness value for the last k generations/fitness evaluations.

1.5 Dealing with constraints

Genetic algorithms features presented in the previous sections allow one to solve a wide range of unconstrained optimization problems. However, when dealing with real-world applications, it is common to work with constrained optimization problems. The latter can be formulated as follows:

$$\min_{\vec{x}\in\mathscr{D}} \quad f(\vec{x}) \tag{1.5.1}$$

$$\text{subject to} \quad g_i(\vec{x}) \leq 0, \quad i = 1,\ldots,p, \tag{1.5.2}$$

$$h_j(\vec{x}) = 0, \quad j = 1,\ldots,q, \tag{1.5.3}$$

where x is the vector of solutions $\vec{x} = [x_1, x_2, \ldots, x_n]$, $\mathscr{D}$ is the search space domain, $f(\vec{x})$ is the objective function, $g_i(\vec{x})$ $(i = 1,\ldots,p)$ are inequality constraints and $h_j(\vec{x})$ $(j = 1,\ldots,q)$ are equality constraints. In what follows, only inequality constraints are considered since equality constraints can be transformed in inequalities with the following equation:

$$|h_j(\vec{x}) - \varepsilon| \leq 0, \tag{1.5.4}$$

where ε is a very small value describing the tolerance allowed. In order to solve constrained optimization problems with genetic algorithms, multiple constraint-handling techniques have been studied. According to Coello (2002), they can be classified in 5 categories: penalty functions, special representations and operators, repair algorithms, separation of objectives and constraints, and hybrid methods. In this work, the presented methodologies are restricted to several kinds of penalty functions and to the superiority of feasible points, which is a particular case of separation of objectives and constraints.

Penalty functions

Monitoring constraints with penalty functions is a very popular approach. It consists in transforming a constrained problem into an unconstrained one by adding a non-negative value to the objective function reflecting that one or several constraints are violated. The obtained unconstrained problem can be expressed as follows:

$$\min_{\vec{x}\in\mathscr{D}} F(\vec{x}) = f(\vec{x}) + C\sum_{i=1}^{p} G_i(\vec{x}), \tag{1.5.5}$$

where $G_i(\vec{x}) = \max(0, g_i(\vec{x})), \forall i = 1,\ldots,p$. The penalty term is null when all constraints are respected and is positive if at least one of them is violated. The parameter C is a weight factor monitoring the impact of the penalty term. Depending on how

it is defined, it provides different strategies to deal with the constraints. The simplest way to define this parameter is to fix it to a constant value throughout all the optimization process. In this case, we talk about a static penalty function. Another strategy, called dynamic penalty function, is to update the parameter C in such as way that the penalty term increases over time, i.e., generations. It can be defined as:

$$C(t) = (Wt)^{\alpha}, \tag{1.5.6}$$

where W is a constant weight factor, t is the number of the current generation and α is a constant parameter that determines the rate at which penalty terms will increase over time (Joines and Houck (1994)). Finally, the parameter C can also be updated with respect to a feedback of the search process. This kind of strategy is called adaptive penalty function. In Hadj-Alouane and Bean (1997), the parameter C is updated at every generation as follows:

$$C(t+1) = \begin{cases} \frac{1}{\alpha_1}C(t) & \text{if the best individual in the last } k \text{ generations} \\ & \text{was always feasible,} \\ \alpha_2 C(t) & \text{if the best individual in the last } k \text{ generations,} \\ & \text{was always unfeasible,} \\ C(t) & \text{otherwise,} \end{cases} \tag{1.5.7}$$

where $\alpha_1 > \alpha_2 > 1$. In other words, the penalty term is decreased if the best individual was always feasible in previous generations; it is increased if the best individual was always unfeasible; otherwise, it remains unchanged.

Superiority of feasible points

The main idea of the superiority of feasible points is that feasible individuals are always preferred over unfeasible ones. Such a strategy can be set up by using the following fitness function[3] proposed by Deb (2000):

$$F(\vec{x}) = \begin{cases} f(\vec{x}) & \text{if } g_i(\vec{x}) \leq 0 \qquad \forall i = 1, \ldots, p \\ f_{worst} + \sum_{i=1}^{p} G_i(\vec{x}) & \text{otherwise,} \end{cases} \tag{1.5.8}$$

where f_{worst} is the objective function of the worst feasible individual in the population, $G_i(\vec{x}) = \max(0, g_i(\vec{x})), \forall i = 1, \ldots, p$. If all individuals in the population are unfeasible, f_{worst} is fixed to zero. Using that fitness function to select individuals is equivalent to perform a binary tournament with the following rules. When two individuals are compared:

- if both are feasible, the one with the best objective value is chosen;
- if only one of them is feasible, it is preferred over the unfeasible one;

[3] As a reminder, good solutions have low fitness values for minimization problems, which is the case of the constrained problems considered in this section.

- if both are unfeasible, the one having smaller constraint violation is selected.

This concludes our overview of the fundamental principles of genetic algorithms.

1.6 Genetic algorithm of Minamo

The genetic algorithm used to obtain the results presented in this thesis is the one implemented in Minamo (Sainvitu et al. (2010)), the multi-disciplinary optimization platform developed at the applied research center Cenaero. The latter relies on EAs whose convergence rate is strongly accelerated through an efficient coupling with surrogate models. In particular, the genetic algorithm considered in this thesis is the canonical EA of Minamo (the strategies to accelerate the convergence with surrogate models will be investigated in Chapters 5 and 6). It follows the general steps of a genetic algorithm with a real-valued representation of individuals. A tournament selection embedded with the superiority of feasible points is used to pick up pairs of parents on which are applied arithmetic crossovers. The individuals resulting from the reproduction step are potentially mutated with a small probability of 1 %. For the survival selection, the population of offspring replaces the population of parents at each generation. An elitism of two individuals is carried out.

Chapter 2

Cooperative co-evolutionary algorithms: state of the art

In the last decades, many real-world optimization problems were tackled with Evolutionary Algorithms (EAs) such as Genetic Algorithms (see Chapter 1), Evolutionary Programming (Yao et al. (1999)), Evolution Strategies (Beyer and Schwefel (2002)), and Differential Evolution (Price et al. (2005)), or Swarm Intelligence such as Particle Swarm Optimization (Kennedy and Eberhart (2001)), Artificial Bee Colony (Karaboga and Basturk (2007)) and Firefly Algorithm (Yang (2008)). However, the algorithms performance declines when the problem dimension increases. Indeed, the search space grows exponentially when the optimization problems tackle with a large number of decision variables. This issue is known as the curse of dimensionality (Bellman (1961)). In order to solve these Large Scale Global Optimization (LSGO) problems, Cooperative Co-Evolutionary Algorithms (CC-EAs) rely on a Divide-and-Conquer (DC) strategy (Descartes (1956)). They divide large optimization problems into smaller and simpler subproblems that can be tackled with a traditional EA, namely optimize separately smaller groups of variables. Note that the notion of large scale is not fixed. It changes over time and it differs from problem to problem. In a general way, it may be defined such as the dimension at which existing methods start to fail. In this thesis, problems with several hundreds or thousands of variables will be studied.

In such a DC context, the decomposition of the large problem plays a significant role. An inappropriate decomposition may prevent the CC-EA from reaching the global optimum of the problem. On the contrary, a proper decomposition would allow one to solve the problem with lower computational resources. Another important point is the exchanges of information between subproblems. Most of them occur for the evaluation of the n-dimensional function to be optimized. Indeed, partial solutions from subproblems can not be directly evaluated with the function since they cover only a subset of s variables ($s < n$) of the n-dimensional vector. They must be completed

with $n - s$ variables from other subproblems to be evaluated. The choice of these completion variables may also affect the performance of the CC-EA architecture.

This chapter is organized as follows. Section 2.1 introduces the main concepts of the CC framework including the decomposition and the choice of completion variables outlined above (Mahdavi et al. (2015); Ma et al. (2019); Jian et al. (2020)). Then, the two main types of CC-EAs encountered in the literature are presented in Sections 2.2 and 2.3. One of them relies on multiple random decompositions, the other on an unique decomposition based on interaction detection.

2.1 Cooperative co-evolutionary framework

The first attempt to solve LSGO problems with a CC framework was made by Potter and De Jong (1994). They designed a Cooperative Co-evolutionary Genetic Algorithm (CC-GA) to improve the performance of a standard genetic algorithm in solving high dimensional problems. In the following years, many efforts were spent to apply the CC framework proposed by Potter and De Jong (1994) to various heuristic optimizers: Liu et al. (2001) scaled up Fast Evolutionary Programming with CC; van den Bergh and Engelbrecht (2004) developed a CC variant of Particle Swarm Optimization and Shi et al. (2005) implemented the proposed methodology in a Differential Evolution paradigm. All these CC algorithms share the same structure that can be described in three steps:

1. *Decomposition*: Split the n-dimensional decision vector into k disjoint subcomponents of size s ($k \times s = n$);
2. *Optimization*: Optimize each subcomponent with a Evolutionary/Swarm Intelligence optimizer for a fixed number of iterations in a round-robin[(1)] strategy;
3. *Combination*: Merge solutions from each subcomponent to build the solution of the n-dimensional problem.

In this framework, the decomposition is *static*, meaning that it remains the same during the whole execution of the optimization algorithm. For the first implementations of the CC framework (Potter and De Jong (1994); Liu et al. (2001)), the n-dimensional decision vector is split into n one-dimensional components (i.e., $s = 1$). Later, van den Bergh and Engelbrecht (2004) proposed to decompose the decision vector into k subcomponents of size $s > 1$, fixed by the user. Such a decomposition is illustrated in Figure 2.1. Shi et al. (2005) introduced two variants, one focusing on n one-dimensional components, the other focusing on two $\frac{n}{2}$-dimensional components. In the following of this work, the main types of CC algorithms will be classified according to the decomposition strategy on which they rely (see Figure 2.2). The presented algorithms

[(1)]The round-robin term is often used in the context of scheduling (Arpaci-Dusseau and Arpaci-Dusseau (2015)). The round-robin scheduling consists in assigning time slices to each process of a list, in equal portions and in circular order and in handling all of them without priority. It is also called time-slicing. In the context of CC-EAs, the round-robin term refers to the fact that one iteration is performed in each subcomponent before moving to the next iteration.

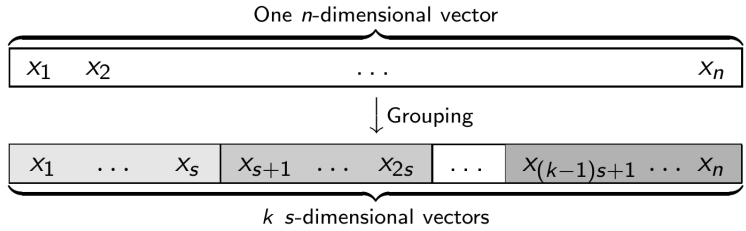


Figure 2.1 – Arbitrarily determined decomposition of n variables into k groups, each one of size s.

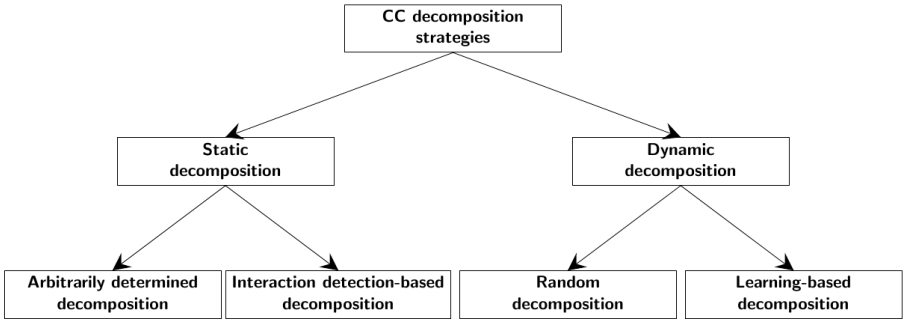


Figure 2.2 – A hierarchical classification of decomposition techniques.

so far belong to the class of *Arbitrarily determined decomposition.* The core principles of algorithms based on a *Random decomposition* will be described in Section 2.2 while those of algorithms relying on an *Interaction detection-based decomposition* will be introduced in Section 2.3. Algorithms belonging to the class of *Learning-based decomposition* are not really considered in this thesis. Only two examples of such algorithms ("MLSoft" and "Delta Grouping") are very briefly presented in Sections 2.2 and 2.3, respectively.

Throughout the optimization step, partial solutions in each subcomponent need to be evaluated with the n-dimensional function. For this purpose, they have to be completed with information extracted from other subcomponents. Potter and De Jong (1994) presented two *cooperation* methods to select *representative individuals* in the remaining subcomponents. The first one selects current best individuals in each subcomponent to complete the individual to be evaluated[(2)]. The resulting function evaluation is illustrated in Figure 2.3. The CC genetic algorithm embedded with this cooperation method is called CC-GA-1 in Potter and De Jong (1994) original publication. It outperforms the standard genetic algorithm on several benchmark functions but it performed much worse on some others. Problematic functions present interacting variables via product terms (see Section 2.2 for further details). Potter and De Jong (1994) felt that the source of the difficulties lies in the cooperation method. They de-

[(2)]This cooperation scheme is used by default in the CC frameworks presented in this thesis.

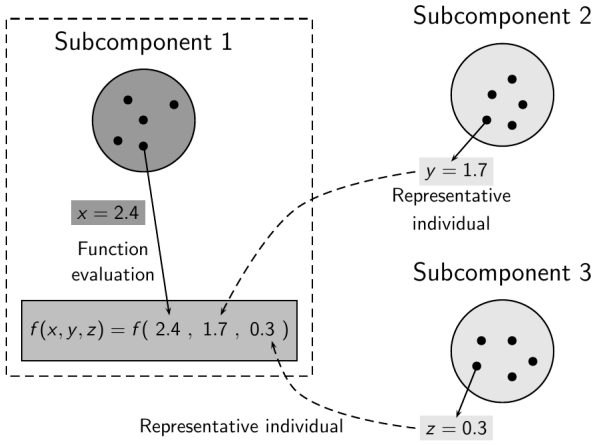


Figure 2.3 – Function evaluation through partial solution completion with representative individuals.

veloped a new algorithm, called CC-GA-2, relying on a new cooperation scheme. The individual to be evaluated is completed either with best individuals from each subcomponent (like in CC-GA-1) or with random individuals picked up in each subcomponent. The two resulting n-dimensional individuals are evaluated and the best function value is assigned to the individual of the considered subcomponent. This new strategy outperforms CC-GA-1 on benchmark functions with interacting variables. However, it is still possible to improve the CC framework to tackle that kind of problems efficiently by introducing new decomposition methodologies. This is the point of the following sections.

2.2 Random decomposition-based CC-EA

As mentioned in the previous section, arbitrarily determined decomposition has some limitations in solving optimization problems with interacting variables. This concept is linked to the notion of *separable function*. In the early 2000s, the terms "separable" and "nonseparable" have been introduced in evolutionary computation even though no formal definition had been established (Suganthan et al. (2005)). Auger et al. (2007) proposed the following definition meaning that if one can achieve the minimum of the function by minimizing on each variable separately, then the function is separable.

Definition 1 *A function $f(\vec{x})$ is separable if*

$$arg \min_{(x_1,\ldots x_n)} f(x_1,\ldots,x_n) = \left(arg \min_{x_1} f(x_1,\ldots), \ldots, arg \min_{x_n} f(\ldots,x_n) \right).$$

Yang et al. (2008a) suggested another definition reflecting the same idea. It states that a function is separable if the influence of a variable on the function value depends only on itself.

Definition 2 *A function $f : \mathscr{D} \subseteq \mathbb{R}^n \to \mathbb{R}$ is separable if,* $\forall\, k \in \{1,n\}$,

$$\left.\begin{array}{ll} \vec{x} \in \mathscr{D} & \vec{x} = (x_1,\ldots,x_k,\ldots,x_n) \\ \vec{x'} \in \mathscr{D} & \vec{x'} = (x_1,\ldots,x'_k,\ldots,x_n) \end{array}\right\} \Rightarrow f(\vec{x}) < f(\vec{x'})$$

implies that

$$\left.\begin{array}{ll} \forall\, \vec{y} \in \mathscr{D} & \vec{y} = (y_1,\ldots,x_k,\ldots,y_n) \\ \forall\, \vec{y'} \in \mathscr{D} & \vec{y'} = (y_1,\ldots,x'_k,\ldots,y_n) \end{array}\right\} \Rightarrow f(\vec{y}) < f(\vec{y'}).$$

Otherwise, $f(\vec{x})$ is called a nonseparable function.

These two definitions are illustrated with the two following functions:

$$\begin{array}{rcll} f(x,y) & = & x^2+y^2, & \forall x,y \in \mathbb{R}; \\ g(x,y) & = & x^2+y^2+4xy, & \forall x,y \in \mathbb{R}. \end{array}$$

One can easily see that function f satisfies both definitions and is therefore separable. On the contrary, the function g is nonseparable. Indeed, due to the product term, one can not minimize the function g by minimizing x and y independently. Besides, the function g does not verify the condition of Definition 2 because

$$g(0,\mathbf{1}) < g(0,\mathbf{2}) \qquad \text{but} \qquad g(-2,\mathbf{1}) > g(-2,\mathbf{2}).$$

In this case, one can say that the variable x *interacts* with the variable y. Random decomposition-based CC-EAs are devoted to efficiently solving optimization problems with such interacting variables.

Let us consider a problem for which two arbitrary variables x_i and x_j interact among each other. In order to efficiently solve such a problem in a CC framework, these two variables should ideally be optimized in the same subcomponent. However, such a decomposition is not possible without any a priori knowledge on the problem at hand. In this context, Yang et al. (2008a) adapted the original CC framework by adding an iterative scheme of random grouping decompositions. This new framework, illustrated in Figure 2.4, is described as follows:

1. Start a new cycle by randomly splitting the n-dimensional decision vector into k disjoint subcomponents of size s;
2. Optimize each subcomponent for a fixed number of iterations in a round-robin strategy;
3. If the maximum number of cycles is not reached, go to Step 1;
4. Merge solutions of subcomponents to build the solution of the n-dimensional problem.

The major change concerns the decomposition scheme which dynamically changes at each new cycle (see the classification in Figure 2.2). The random decomposition, illustrated in Figure 2.5, is performed in two stages. Firstly, the n-dimensional

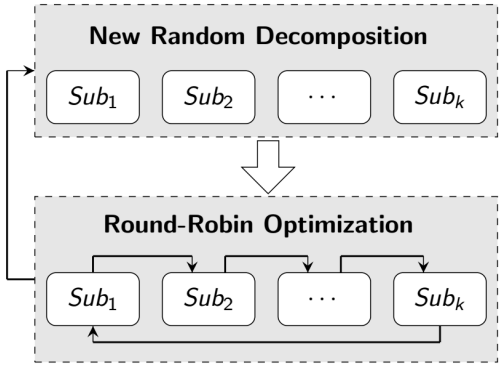


Figure 2.4 – Random decomposition-based CC framework.

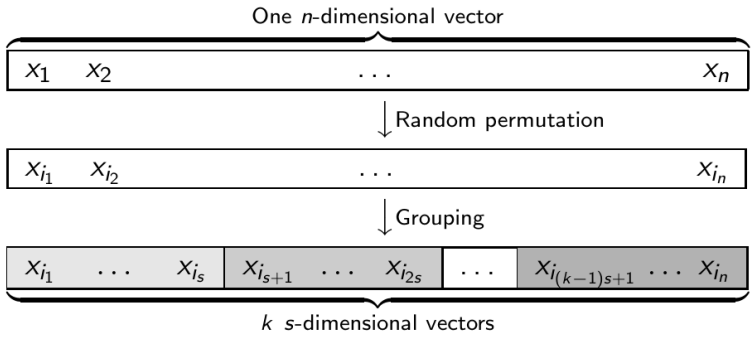


Figure 2.5 – Random grouping decomposition.

decision vector is randomly permuted. Secondly, this permuted decision vector is split into k s-dimensional vectors, the subcomponent size s being predetermined. The newly proposed CC framework relies on the following principle. If the variables x_i and x_j have to be optimized together, if a sufficiently large number of cycles is performed, the probability that the two variables will be optimized together for at least one cycle is close to one. The following theorem introduced by Yang et al. (2008a) provides the probability to optimize two interacting variables together in the newly proposed framework.

Theorem 2.2.1 *(Yang et al. (2008a)) The probability to assign two interacting variables x_i and x_j into a single subcomponent for a least l cycles is*

$$P_l = \sum_{r=l}^{N} C_N^r \left(\frac{1}{k}\right)^r \left(1-\frac{1}{k}\right)^{N-r}, \tag{2.2.1}$$

where N is the total number of cycles and k is the number of subcomponents.

Proof. In each cycle, the probability to assign x_i and x_j in the same subcomponent is given by $p = \frac{1}{k}$. The random variable X counting each time x_i is grouped with x_j follows a binomial distribution $B(N, p)$. Therefore, the probability that x_i and x_j lies in the same subcomponent for exactly r cycles is given by:

$$P(X = r) = C_N^r \left(\frac{1}{k}\right)^r \left(1 - \frac{1}{k}\right)^{N-r}.$$

The probability to assign two interacting variables x_i and x_j into a single subcomponent for a least l cycles is obtained by summing up probabilities for all integer values between l and N. □

Given an optimization problem with 1 000 decision variables, suppose that the random CC framework is applied with $s = 100$. Therefore, based on Equation (2.2.1), the probabilities to assign two interacting variables in the same subcomponent for at least one or two cycles are given by $P_1 = 0.9948$ and $P_2 = 0.9662$, respectively. These high probabilities reflect the random decomposition efficiency in capturing variable interactions.

The original random CC framework proposed by Yang et al. (2008a) was embedded into a Differential Evolution optimizer. It has been enhanced with a Multi-level Cooperative Co-evolution (MLCC) in Yang et al. (2008b) and Omidvar et al. (2010a). The MLCC strategy chooses the adequate group size through a decomposer pool containing different group sizes. Historical information is recorded during the evolutionary process allowing the algorithm to self-adapt the group size. Omidvar et al. (2014) further improved this idea by using widely-used techniques in reinforcement techniques in their MLSoft algorithm. That kind of works can be classified as a *learning-based decomposition* algorithm in the tree proposed in Figure 2.2. Over the same period, Li and Yao (2009, 2012) developed the random CC framework in Particle Swarm Optimization. Later, it has also been applied in other Swarm Intelligence such as CC Artificial Bee Colony (Ren and Wu (2013)) and CC Firefly Algorithm (Trunfio (2014)). More recently, Duan et al. (2019a) embedded the random grouping method in a hierarchical fashion. In this work, the search space is gradually extended in order to escape from Nash equilibrium (Duan et al. (2019b)).

2.3 Interaction detection decomposition-based CC-EA

The dynamic random grouping strategy introduced in the previous section tries to catch interacting variables in a same subcomponent by repeating a large number of random decompositions. Although no a priori information is known about the problem at hand, it is possible to perform better decompositions by trying to learn the interaction between variables. Initial attempts first try to detect interaction during the evolutionary process. Then, they share the variables in different subcomponents based on the learned information and continue the CC optimization with the new structure. Ray and Yao (2009) introduced such a CC algorithm proceeding in two steps: first,

all the variables are evolved with a standard EA in a single component; then the correlation coefficients are computed based on top 50 % solutions and the grouping is performed based on these coefficients. Chen et al. (2010) also presented a two-stage (learning and optimization) CC framework called CCVIL for CC Variable Interaction Learning. The learning stage relies on pioneering studies by Weicker and Weicker (1999). In this work, a new individual whose i-th entry has value $newval_i$ is completed for the function evaluation with different representative individuals for the remaining components. Two candidate solutions $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{x}\,' = (x'_1, \ldots, x'_n)$ are produced as follows:

$$x_j = \begin{cases} newval_i, & \text{if } i = j, \\ bestval_j, & \text{otherwise} \end{cases} \qquad \text{and} \qquad x'_j = \begin{cases} newval_i, & \text{if } i = j, \\ randval_k, & \text{if } k = j, \\ bestval_j, & \text{otherwise} \end{cases}$$

where $bestval_j$ is the value of the j-th entry of the best individual in the population, $randval_k$ is the value of the k-th entry of a random individual in the population, and k is a randomly chosen dimension distinct from i. If $f(\vec{x}\,')$ is better than $f(\vec{x})$, the variable i probably interacts with the variable k. Omidvar et al. (2010b) also proposed a dynamic learning-based decomposition strategy called Delta Grouping. In this paper, delta values, that measure the amount of change in each of the decision variables in every iteration, are introduced. If two interacting variables are grouped in separate components, both their delta values will be small because the improvement interval is shrunk. Sorting the variables according to their delta values for grouping helps to increase the chance to group interacting variables in a same subcomponent.

More recently, wide efforts have been focused on *static interaction detection-based decompositions* (see Figure 2.2). In this scheme, the interaction structure is analyzed before starting the optimization. Then, the decomposition is performed and is fixed for the whole CC evolutionary optimization. Sayed et al. (2012) introduced a Dependency Identification technique based on the definition of additively separable function.

Definition 3 *A function $f(\vec{x})$ is additively separable if it can be written as*

$$f(\vec{x}) = \sum_{i=1}^{m} f_i(\vec{x}_i),$$

where $\vec{x}_i$ $(i = 1, \ldots, m)$ are mutually exclusive k_i-dimensional decision vectors of f_i and m is the number of independent components such that $k_1 + \cdots + k_m = n$.

The proposed technique consists in finding a problem decomposition that minimizes the square difference of $f(\vec{x})$ and the summation of all $f_i(\vec{x}_i)$ $(i = 1, \ldots, m)$. Later, Mahdavi et al. (2014) published a Decomposition Method based on High Dimensional Model Representation (DM-HDMR). In this strategy, the correlation relationship between pairs of variables is computed on the basis of a radial basis function approximation of the function. All these strategies have been outperformed by a strategy called *Differential Grouping*. Initially introduced in the publication of Omidvar

et al. (2013), this strategy has been studied by numerous researchers that proposed several variants and upgrades in the following years. Most of the algorithms presented in this thesis rely on this grouping strategy. The latter is presented in details in the next sections.

2.3.1 Differential Grouping

The Differential Grouping (DG) strategy proposes an automatic decomposition that can uncover the interaction structure of the decision variables. It monitors the changes to the objective function in order to detect the interaction. Such a method had already been proposed by Tezuka et al. (2004) in their LINC-R optimization algorithm, where LINC-R stands for Linkage Identification by Nonlinearity Check for Real-coded genetic algorithms. The LINC-R algorithm uses the uncovered interaction structure to build an island model with periodic migration of individuals between islands[3]. It was tested on a limited set of low dimensional benchmark functions. Some ten years later, Omidvar et al. (2013) reused the idea presented in LINC-R with stronger theoretical analysis to present the DG strategy. Furthermore, unlike in LINC-CR, Omidvar et al. (2013) set up a CC framework that benefits from the accurate decomposition obtained with DG to solve large-scale problems. The proposed DG strategy relies on the following results.

Theorem 2.3.1 *(Omidvar et al. (2013)) Let $f(\vec{x})$ be an additively separable function. $\forall a, b_1 \neq b_2, \delta \in \mathbb{R}, \delta \neq 0$, if the following condition holds*

$$\Delta_{\delta,p}[f](\vec{x})|_{x_p=a,x_q=b_1} \neq \Delta_{\delta,p}[f](\vec{x})|_{x_p=a,x_q=b_2},$$

then x_p and x_q are nonseparable, where

$$\Delta_{\delta,p}[f](\vec{x}) = f(\ldots,x_p+\delta,\ldots) - f(\ldots,x_p,\ldots), \tag{2.3.1}$$

refers to the forward difference of f with respect to variable x_p with interval δ, the other variables remain fixed to the same value for the two function evaluations.

Proof. The reasoning consists in proving the contrapositive. It states that if two variables x_p and x_q are separable, then

$$\Delta_{\delta,p}[f](\vec{x})|_{x_p=a,x_q=b_1} = \Delta_{\delta,p}[f](\vec{x})|_{x_p=a,x_q=b_2} \qquad \forall a, b_1 \neq b_2, \delta \in \mathbb{R}, \delta \neq 0.$$

Let suppose that x_p and x_q are separable. Since $f(\vec{x})$ is additively separable, we have

$$\frac{\partial f(\vec{x})}{\partial x_p} = \frac{\partial f_i(\vec{x}_i)}{\partial x_p}, \qquad \forall x_p \in \vec{x}_i.$$

Moreover, $\forall x_q \notin \vec{x}_i$, we have

$$\left.\frac{\partial f(\vec{x})}{\partial x_p}\right|_{x_q=b_1} = \left.\frac{\partial f(\vec{x})}{\partial x_p}\right|_{x_q=b_2} = \frac{\partial f_i(\vec{x}_i)}{\partial x_p}, \qquad \forall b_1 \neq b_2.$$

[3] In fact, this paradigm is equivalent to the CC framework introduced in this thesis. The term *island* corresponds to a subpopulation of individuals while the migration between islands corresponds to the exchange of representative individuals for the function evaluations.

By integrating, we obtain

$$\int_a^{a+\delta} \frac{\partial f(\vec{x})}{\partial x_p} dx_p \bigg|_{x_q=b_1} = \int_a^{a+\delta} \frac{\partial f(\vec{x})}{\partial x_p} dx_p \bigg|_{x_q=b_2}, \qquad \forall a, \delta \in \mathbb{R}, \delta \neq 0,$$

meaning that the following equation holds

$$\Delta_{\delta,p}[f](\vec{x})|_{x_p=a,x_q=b_1} = \Delta_{\delta,p}[f](\vec{x})|_{x_p=a,x_q=b_2} \qquad \forall a, b_1 \neq b_2, \delta \in \mathbb{R}, \delta \neq 0.$$

□

This theorem states that given an additively separable function, if two variables are separable, then the delta values Δ_1 and Δ_2 resulting from Equation (2.3.1) evaluated with any different values of x_q, are equal. In other words,

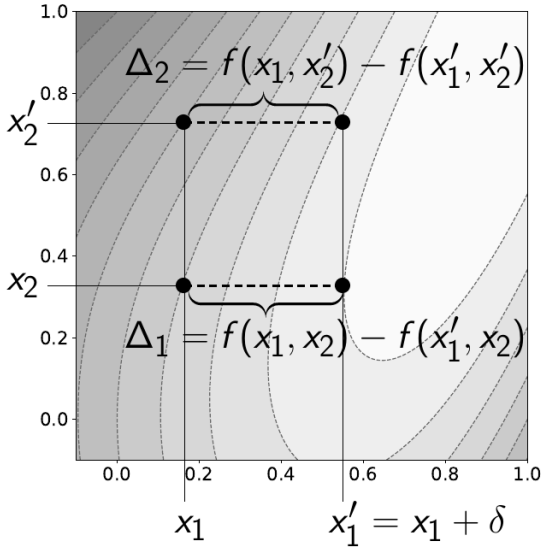
$$\text{separability} \Rightarrow \Delta_1 = \Delta_2. \tag{2.3.2}$$

In the DG strategy, delta values are computed and the contrapositive of Equation (2.3.2) is used to identify nonseparable variables:

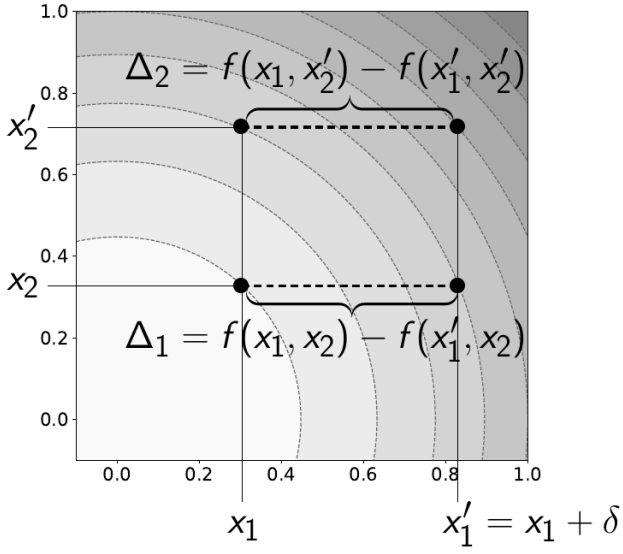
$$\Delta_1 \neq \Delta_2 \Rightarrow \text{ nonseparability}.$$

However, strong conclusions can not be drawn when delta values are equal since the reciprocal of Equation (2.3.2) is not verified. In this case, DG still identifies x_p and x_q such as they were separable variables. Indeed, in Equation (2.3.2), if $\Delta_1 = \Delta_2$, we can not claim that x_p and x_q are separable, but since $\Delta_1 = \Delta_2$, it becomes more likely. That kind of reasoning is presented as a *weak syllogism* in Omidvar and Li (2017). It can be illustrated with a simpler example. Let us consider the following proposition: Rain $\Rightarrow$ Cloud. If it is cloudy, we can not claim that it is raining but it is more likely than if it is sunny.

In order to uncover the interaction structure, the DG strategy computes delta values between pairs of variables and checks if the quantity $\lambda = |\Delta_1 - \Delta_2|$ is greater than a threshold $\varepsilon > 0$. This procedure is illustrated in Figure 2.6 for two-dimensional functions. Theoretically, the parameter ε could be set to zero. However, in practice, such a setting is not suitable due to the limited precision of floating-point numbers. Considering very small values of λ, some of them may be the result of true interactions between variables. Some others may reflect false interaction detections as a result of roundoff errors in the computation of λ. Finding an adequate value for the parameter ε is a challenging task. If ε is too large, lots of interactions will be missed and many nonseparable variables will be identified as separable ones. One the contrary, if ε is too small, lots of false interactions will be detected meaning that many separable variables will be identified as nonseparable ones. Note that the latter case is less detrimental to the CC optimization performance since it is not intrinsically a bad thing to force separable variables detected such as nonseparable ones to be optimized in the same subcomponent. However, if too many separable variables are identified such as nonseparable ones, very large subcomponents will be formed and the advantage of dimension reduction of the CC framework will be lost. In the first publication of



(a) $f(x_1,x_2) = (1-x_1)^2 + (x_2 - x_1^2)^2$



(b) $f(x_1,x_2) = x_1^2 + x_2^2$

Figure 2.6 – Interaction detection with DG for two-dimensional functions. For the first function at the top, it can be seen that $\Delta_1 \neq \Delta_2$ since Δ_1 value corresponds to the gap between three level curves while Δ_2 amounts the difference between four level curves. Therefore, x_1 and x_2 are identified as nonseparable variables. For the second function, at the bottom, both delta values correspond to the gap between three level curves and are therefore equal. The variables x_1 and x_2 are identified as separable ones.

Omidvar et al. (2013) devoted to DG, the parameter ε requires to be specified by the user. This is quite problematic since the specified value greatly determines the performance of the CC framework based on the decomposition.

Another drawback identified in the DG strategy is its inability to detect *indirect interactions* between pairs of variables. Such interactions are defined as follows.

Definition 4 *(Sun et al. (2015)) In an objective function $f(\vec{x})$, the decision variables x_i and x_j interact directly with each other if there exists a candidate solution $\vec{x}_\star$ such that*

$$\left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_{\vec{x}_\star} \neq 0,$$

denoted by $x_i \leftrightarrow x_j$. Decision variables x_i and x_j interact indirectly with each other if for all candidate solutions,

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = 0, \tag{2.3.3}$$

and there exists a set of decision variables $\{x_{k_1}, \ldots, x_{k_t}\} \subset \vec{x}$ such that $x_i \leftrightarrow x_{k_1} \leftrightarrow \ldots \leftrightarrow x_{k_t} \leftrightarrow x_j$. Decision variables are independent with each other if for all candidate solutions, Equation (2.3.3) holds and there exists no set of decision variables $\{x_{k_1}, \ldots, x_{k_t}\} \subset \vec{x}$ such that $x_i \leftrightarrow x_{k_1} \leftrightarrow \cdots \leftrightarrow x_{k_t} \leftrightarrow x_j$.

New algorithms proposed by Sun et al. (2015) (xDG, for extended DG) and Ling et al. (2016) (gDG for graph-based DG) are devoted to catch these interactions efficiently. In the same period of time, Omidvar et al. (2015) presented an Improved DG (IDG) addressing the drawback relative to the setting of the parameter ε in DG and providing a new sampling strategy that generates much less points to identify the interaction structure. On the basis of these research works, Omidvar et al. (2017) published a new DG strategy called DG2. This very efficient procedure to detect the interaction structure of large-scale problems is presented in details hereafter.

The first step of DG2 strategy is to evaluate sampling points that will be used to detect interactions by applying Theorem 2.3.1. For each couple of variables x_i and x_j $(i, j = 1, \ldots, n,\ i < j)$, the following quantities need to be evaluated:

$$\begin{cases} \Delta_1 = f(\ldots, x'_i, \ldots) - f(x_1, \ldots, x_n), \\ \Delta_2 = f(\ldots, x'_i, \ldots, x'_j, \ldots) - f(\ldots, x'_j, \ldots), \end{cases} \tag{2.3.4}$$

where $x'_i = x_i + \delta$ $(i = 1, \ldots, n)$. However, the four function evaluations in these equations do not need to be reevaluated for each couple of variables, most of them can indeed be used multiple times. The quantity $f(x_1, \ldots, x_n)$ is the same for all the computations. The point $(x_1, \ldots, x_n)$ is called the *base point* and its function value is denoted by $\underline{f}$. The quantities $f(\ldots, x'_i, \ldots)$ $(i = 1, \ldots, n)$ are used to check interactions between the variable i and all other variables. They are stored in a n-dimensional vector $\vec{f}$. Finally, the quantities $f(\ldots, x'_i, \ldots, x'_j, \ldots)$ must be computed for each couple (i, j), $i < j$. They are stored in a $n \times n$ upper triangular matrix F. This Sampling Points

Evaluation (SPE) scheme is presented in Algorithm 1. In this algorithm, $\vec{lb}$ is the vector of lower bound values of the function domain, $\vec{ub}$ stands for the upper bound, $func$ is the function that we want to analyze. The algorithm computes the function evaluations needed in Equation (2.3.4) for $\vec{x} = \vec{lb}$; the x'_i values ($i = 1, \ldots, n$) are chosen as the centers of the variable x_i domain. In total, the algorithm requires $\frac{n(n+1)}{2} + 1$ function evaluations.

Algorithm 1: $(\underline{f}, \vec{f}, F) = \text{SPE}(\vec{lb}, \vec{ub}, func)$

```
1  m⃗ = (1/2)(lb⃗ + ub⃗) ;
2  x⃗ = lb⃗ ;
3  f = func(x⃗) ;
4  for i = 1, ..., n do
5  |   x⃗' = x⃗, x'_i = m_i ;
6  |   f_i = func(x⃗') ;
7  end
8  for i = 1, ..., n − 1 do
9  |   x⃗' = x⃗, x'_i = m_i ;
10 |   for j = i + 1, ..., n do
11 |   |   x⃗'' = x⃗', x''_j = m_j ;
12 |   |   F_{i,j} = func(x⃗'') ;
13 |   end
14 end
```

On the basis of the information obtained by the SPE algorithm, the *Interaction Structure Matrix* Λ (ISM), i.e., the matrix containing the quantity $\lambda = |\Delta_1 - \Delta_2|$ for all pairs of variables, can be computed as shown in Algorithm 2. After that, the challenge is to identify nonzero values in Λ that really represent interacting variables, in contrast with non-zero values resulting from roundoff errors. Defining a threshold ε, the Design Structure Matrix Θ (DSM) is built according to the following rule: $\Theta_{i,j}$ takes 1 if $\Lambda_{i,j} > \varepsilon$, and 0 otherwise.

Algorithm 2: $\Lambda = \text{ISM}(\underline{f}, \vec{f}, F)$

```
1 for i = 1, ..., n − 1 do
2 |   for j = i + 1, ..., n do
3 |   |   Δ1 = f_i − f ;
4 |   |   Δ2 = F_ij − f_j ;
5 |   |   Λ_{i,j} = |Δ1 − Δ2| ;
6 |   end
7 end
```

As discussed earlier, the parameter ε can not be set to zero due to the roundoff errors. It can neither be set to a static value such as in the original work of Omidvar et al. (2013) since the errors depend on the magnitude of the quantities used in the calculation. As a first step, the greatest lower bound e_{inf} and the least upper bound e_{sup} for the roundoff errors are estimated. The bound e_{inf} takes into account errors resulting from the arithmetic floating-point substraction between the function values $f(x)$ in the calculation of $\Lambda_{i,j}$. The bound e_{sup} reflects the error in the computation of $f(x)$ itself. The entry $\Theta_{i,j}$ takes 1 if $\Lambda_{i,j} > e_{sup}$ and 0 if $\Lambda_{i,j} < e_{inf}$. The values lying between

the two bounds will be later investigated. Let x be a real number, its floating-point representation denoted by $\hat{x}$ is given as follows (Corless and Fillion (2013)):

$$\hat{x} = x + \delta x = x(1+\delta),$$

where $|\delta| < \mu_M$ according to the IEEE-754 standards (IEEE (2008)). The quantity μ_M is called the *unit roundoff* and is equal to the half of the *machine epsilon* ε_m ($= 2^{-52} \approx 2.22 \times 10^{-16}$ for double precision floating-point numbers). Therefore, the greater is the real number x, the greater is the error δx. Furthermore, following the IEEE standards, the floating-point summation of two real numbers x and y is the floating-point representation of the summation of the two numbers, i.e.,

$$x \oplus y = \widehat{x+y}.$$

In order to estimate the bound e_{inf}, let us suppose that the calculation of $f(x)$ is error free and let us estimate the error in the calculation of $\lambda = |\Delta_1 - \Delta_2|$:

$$\begin{aligned}
\hat{\Delta}_1 &= f(x) \ominus f(x') = (f(x) - f(x'))(1+\delta_1) = \Delta_1(1+\delta_1) \\
\hat{\Delta}_2 &= f(y) \ominus f(y') = (f(y) - f(y'))(1+\delta_2) = \Delta_2(1+\delta_2)
\end{aligned}$$

$$\begin{aligned}
\hat{\lambda} &= |\hat{\Delta}_1 \ominus \hat{\Delta}_2| = |\hat{\Delta}_1 - \hat{\Delta}_2|(1+\delta_3) \\
&= |f(x)(1+\delta_1)(1+\delta_3) - f(x')(1+\delta_1)(1+\delta_3) \\
&\quad -f(y)(1+\delta_2)(1+\delta_3) + f(y')(1+\delta_2)(1+\delta_3)|.
\end{aligned} \tag{2.3.5}$$

In this calculation, the products of the form $\prod_{i=1}^{k}(1+\delta_i)$ contain two factors. Therefore, the following theorem can be applied with $k=2$ to estimate e_{inf}.

Theorem 2.3.2 *(Corless and Fillion (2013)) Consider a real-floating-point system satisfying the IEEE standards, so that $|\delta_i| < \mu_M$. Moreover, let $e_i = \pm 1$ and suppose that $k\mu_M < 1$. Then*

$$\prod_{i=1}^{k}(1+\delta_i)^{e_i} = 1 + \theta_k,$$

where

$$|\theta_k| \leq \frac{k\mu_M}{1-k\mu_M} =: \gamma_k.$$

According to this result, the error in the computation of λ is bounded as follows:

$$\begin{aligned}
|\lambda - \hat{\lambda}| &\leq \gamma_2 \left|(f(x) - f(x')) - (f(y) - f(y'))\right| \\
&\leq \gamma_2(|f(x)| + |f(x')| + |f(y)| + |f(y')|) := e_{inf}.
\end{aligned}$$

In order to estimate the bound e_{sup}, it can no longer be assumed that the calculation of $f(x)$ is error free. Besides, it is assumed that the error in calculating $|\lambda - \hat{\lambda}|$ is negligible with respect to the error resulting from the calculation of $f(x)$. Since $f(x)$

is considered as a black-box function, it is not possible to determine an exact estimation of the error in the calculation of $f(x)$. However, it is common practice to assume that the error is proportional to the square root of the number of floating-point operations Φ in the calculation (Higham (2002)), i.e., $k \approx \sqrt{\Phi}$ in Theorem 2.3.2. Once again, as $f(x)$ is a black-box function, Φ is unknown but it is acceptable to assume that it grows linearly with the dimension n. In this case, the least upper bound e_{sup} can be defined as follows:

$$e_{sup} := \gamma_{\sqrt{n}}\, max\{|f(x)|, |f(x')|, |f(y)|, |f(y')|\}.$$

For values of $\Lambda_{i,j}$ between e_{inf} and e_{sup}, a threshold ε is defined as a weighted average of the two bounds:

$$\varepsilon = \frac{\eta_0}{\eta_0+\eta_1} e_{inf} + \frac{\eta_1}{\eta_0+\eta_1} e_{sup},$$

where η_0 and η_1 are the number of entries in Λ which are smaller than e_{inf}, respectively greater than e_{sup}. Corresponding values of the matrix Θ are, respectively, set to 0 or 1. The whole process to build the DSM matrix is synthesized in Algorithm 3.

Algorithm 3: $\Theta = \mathrm{DSM}(\Lambda, \underline{f}, \vec{f}, F)$

```
1  Θ =NaN_{n×n},  η1 = η2 = 0 ;
2  for i = 1,...,n−1 do
3      for j = i+1,...,n do
4          e_inf =
             γ2 (|f_|+|f_i|+|f_j|+|F_ij|);
5          e_sup =
             γ_√n max{|f_|,|f_i|,|f_j|,|F_ij|};
6          if Λ_{i,j} < e_inf then
7              Θ_{i,j} = 0,  η0 = η0 + 1 ;
8          else if Λ_{i,j} > e_sup then
9              Θ_{i,j} = 1,  η1 = η1 + 1 ;
10         end
11     end
12 end
13 ε = η0/(η0+η1) e_inf + η1/(η0+η1) e_sup ;
14 for i = 1,...,n−1 do
15     for j = i+1,...,n do
16         if Θ_{i,j} = NaN then
17             if Λ_{i,j} < ε then
18                 Θ_{i,j} = 0 ;
19             else
20                 Θ_{i,j} = 1 ;
21             end
22         end
23     end
24 end
```

Finally, the groups of variables are created on the basis of the matrix Θ. The latter is employed as a node adjacency matrix of a graph, each node representing a decision variable. Each connected component determines a group of interacting variables. The components containing exactly one variable correspond to the separable variables. All these variables are assigned into the same group. Note that this step can be performed efficiently since the connected components can be identified in a linear time n (Hopcroft and Tarjan (1973)). The whole procedure to create the groups of variables is summarized in Algorithm 4, where the *ConnComp* function identifies the n_c connected components c_i ($i = 1 \ldots, n_c$) of the graph whose adjacency matrix is Θ.

Algorithm 4: $(k, G_1, \ldots, G_k) = \text{DG2}(\vec{lb}, \vec{ub}, func)$

```
(f, f⃗, F) = SPE(lb⃗, ub⃗, func) ;
Λ = ISM(f, f⃗, F) ;
Θ = DSM(Λ, f, f⃗, F) ;
(n_c, c_1, ..., c_{n_c}) = ConnComp(Θ) ;
X_sep = {}, k = 0 ;
for i = 1, ..., n_c do
    if |c_i| = 1 then
        X_sep = X_sep ∪ c_i ;
    else
        k = k + 1, G_k = c_i ;
    end
end
if X_sep ≠ {} then
    k = k + 1, G_k = X_sep ;
end
```

This decomposition based on DG2 is very efficient in a CC optimization framework. Indeed, if the interaction structure is correctly identified, variables from different subcomponents are independent. Therefore, each subcomponent can easily be optimized in a CC framework. The latter is illustrated in Figure 2.7 and is described with the three following steps:

1. *Decomposition*: Split the n-dimensional decision vector into k disjoint subcomponents with DG2;

2. *Optimization*: Optimize each subcomponent for a fixed number of iterations in a round-robin strategy;

3. *Combination*: Merge solutions of subcomponents to build the solution of the n-dimensional problem.

Note that in contrast to the Random decomposition-based CC-EA, cycling in order to try catching interacting variables in the same subcomponent is not needed anymore. Furthermore, thanks to the accurate grouping, the convergence rate in each subcomponent is much better than for the Random decomposition-based CC-EA. The fact that the CC framework embedded with DG clearly outperforms the CC embedded with random decomposition was shown in numerous publications for various optimizers: amongst others, Omidvar et al. (2013, 2017) for Differential Evolution, Blanchard et al. (2017) for Genetic Algorithm and Sun et al. (2018) for Covariance Matrix Adaptation Evolution Strategy.

The following researches devoted to DG were focused on the number of function evaluations required to create the group of variables. As shown earlier, this number is equal to $\frac{n(n+1)}{2} + 1$ for n-dimensional functions. When the budget is limited, if too many function evaluations are used in the DG procedure, the number of allowed function evaluations for the optimization itself is quite reduced. Hu et al. (2017) proposed a Fast Interdependency Identification (FII) strategy proceeding in two steps: (1) each variable is identified as separable or nonseparable (but the couples of interacting variables remain unknown); (2) the structure identification is performed among the nonseparable variables. This strategy allows one to save a large number of function

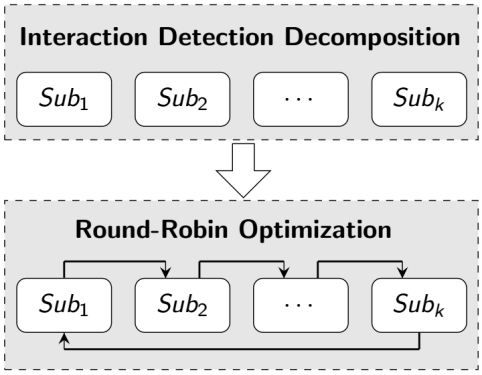


Figure 2.7 – Interaction detection decomposition-based CC framework.

evaluations compared to the DG procedure presented above, especially in the case that the majority of the variables are separable. In the same period, Sun et al. (2017b) presented a promising approach to reduce the number of function evaluations consisting in a recursive variant of DG. This strategy is presented in details in the next section.

2.3.2 Recursive Differential Grouping

By contrast with the DG strategy that checks all pairwise interactions between the decision variables, the Recursive Differential Grouping (RDG) as stated by its name recursively checks interactions between two subsets of variables (Sun et al. (2017b)). This recursive framework allows one to efficiently determine the interaction structure while requiring less than $6n\log(n)$ function evaluations compared with $\frac{n(n+1)}{2}+1$ for DG2. The newly proposed methodology relies on the following result.

Notation 1 *(Sun et al. (2017b)) Let X be the set of decision variables $\{x_1,\ldots,x_n\}$; U_X be the set of unit vectors in the decision space $\mathbb{R}^n$ (i.e., such that $||u||_2 = 1, \forall u \in U_X$). Let X_1 be a subset of decision variables $X_1 \subset X$; and U_{X_1} be a subset of U_X such that for any unit vector $\vec{u} = (u_1,\ldots,u_n) \in U_{X_1}$, we have*

$$u_i = 0, \qquad \text{if } x_i \notin X_1.$$

Theorem 2.3.3 *(Sun et al. (2017b)) Let $f : \mathbb{R}^n \to \mathbb{R}$ be an objective function; $X_1 \subset X$ and $X_2 \subset X$ be two mutually exclusive subsets of decision variables: $X_1 \cap X_2 = \emptyset$. If there exist two unit vectors $\vec{u}_1 \in U_{X_1}$ and $\vec{u}_2 \in U_{X_2}$, two real numbers $l_1, l_2 > 0$, and a candidate solution $\vec{x}$ in the decision space, such that*

$$f(\vec{x}+l_1\vec{u}_1+l_2\vec{u}_2) - f(\vec{x}+l_2\vec{u}_2) \neq f(\vec{x}+l_1\vec{u}_1) - f(\vec{x}),$$

then there is some interaction between decision variables in X_1 and X_2.

On the basis of this result, the interaction identification between two subsets X_1 and X_2 can be achieved with the procedure presented in Algorithm 5. For the sake of clarity, the following notation is preferred in this algorithm.

Notation 2 *Let X be the set of decision variables $\{x_1,\dots,x_n\}$; X_1 and X_2 two disjoint subsets of X such that $X_1 \cup X_2 = X$, a and b two real numbers, the vector $x_{a,b} \in \mathbb{R}^n$ is such that*

$$(x_{a,b})_i = a, \text{ if } i \in X_1, \qquad \text{and} \qquad (x_{a,b})_i = b, \text{ if } i \in X_2.$$

In what follows, the real number 'l' stands for the lower bound, 'u' for the upper bound and 'm' for the mean of the lower and upper bounds.

As a reminder, $\underline{f}$ is the function value of the base point. According to the introduced notation, we have $\underline{f} = f(x_{l,l})$. Therefore, in the algorithm, Δ_1 and Δ_2 evaluate the difference in the function value when the variables from the set X_1 vary from the lower bound l to the upper bound u. For Δ_1, the variables from X_2 are set to the lower bound l while they are fixed to the mean of the bounds m for Δ_2. The algorithm returns the *true* value if some interaction is detected between X_1 and X_2, i.e., if $|\Delta_1 - \Delta_2| > \varepsilon$. In the RDG algorithm published in Sun et al. (2017b), the setting of ε requires the user to specify some parameters. It has been improved in Sun et al. (2018) to become parameter-free. This new variant is implemented in Algorithm 5. The details of this setting is presented later in this section.

Algorithm 5: Interact$(X_1, X_2, \underline{f}, func)$

1 $f_1 = func(\vec{x}_{u,l}),\ f_2 = func(\vec{x}_{l,m})$;
2 $f_3 = func(\vec{x}_{u,m})$;
3 $\Delta_1 = \underline{f} - f_1,\ \Delta_2 = f_2 - f_3$;
4 $\varepsilon = \gamma_{\sqrt{n}+2}(|\underline{f}| + |f_1| + |f_2| + |f_3|)$;
5 **if** $|\Delta_1 - \Delta_2| > \varepsilon$ **then**
6 | **return** true ;
7 **end**
8 **return** false ;

In order to perform the recursive identification of interactions between subsets of variables, another function called `R_Inter` for 'Recursive Interaction' is introduced in Algorithm 6. It takes two subsets X_1 and X_2 as entries and returns a new subset X_1 which is the union of the previous X_1 and all variables in X_2 that directly interact with X_1. Therefore, if no interaction is detected, it simply returns the subset X_1 given as entry. If some interaction is detected, the subset X_2 is divided into two nearly equally-sized groups G_1 and G_2. In this case, interaction between X_1 and G_1, and X_1 and G_2 is recursively checked using the '`R_Inter`' function until all the individual variables that interact with X_1 are identified.

Finally, the whole RDG procedure is presented in Algorithm 7. At line 5, the interaction identification between X_1 and the remaining variables is computed. If no interaction is detected (lines $6 \rightarrow 13$), X_1 is classified as a separable variable if it contains only one variable or as a nonseparable group if it contains several variables. The process is repeated with the next variable available in X_2. If some interaction is detected (lines $14 \rightarrow 17$), the process will be repeated between $X_1^\star$ and remaining variables for the purpose of capturing the variables that indirectly interact with X_1. Once the set X_2 is empty, the last variables in X_1 still have to be classified in the separable

Algorithm 6: R_Inter$(X_1, X_2, \underline{f}, func)$

```
1  if Interact(X1, X2, f, func) then
2      if |X2| = 1 then
3          X1 = X1 ∪ X2 ;
4      else
5          Divide X2 into equally-sized
             groups G1, G2 ;
6          X1^1 = R_Inter(X1, G1, f, func);
7          X1^2 = R_Inter(X1, G2, f, func);
8          X1 = X1^1 ∪ X1^2 ;
9      end
10 end
11 return X1 ;
```

Algorithm 7: Recursive Differential Grouping$(func)$

```
1  seps = {}, nonseps = {} ;
2  X1 = {x1}, X2 = {x2, ..., xn} ;
3  f = func(x_{l,l}) ;
4  while X2 ≠ {} do
5      X1* = R_Inter(X1, X2, f, func) ;
6      if |X1*| = |X1| then
7          if |X1| = 1 then
8              seps = seps ∪ X1 ;
9          else
10             nonseps = nonseps ∪ {X1} ;
11         end
12         X1 = {xj} s.t. j ≤ i ∀xi ∈ X2;
13         X2 = X2 \ {xj} ;
14     else
15         X1 = X1* ;
16         X2 = X2 \ X1 ;
17     end
18 end
19 if |X1| = 1 then
20     seps = seps ∪ X1 ;
21 else
22     nonseps = nonseps ∪ {X1} ;
23 end
24 return seps and nonseps ;
```

or nonseparable sets (lines $19 \rightarrow 23$)[4]. The algorithm returns all the separable variables in the set *seps* and the groups of nonseparable variables in *nonseps*.

Later, Sun et al. (2018) proposed an improved version of RDG called RDG2. This new algorithm is RDG embedded with a threshold parameter estimation of ε. It picks up the estimation developed in DG2 and improves it to define only one threshold taking into account the errors resulting from the arithmetic floating-point substraction between the function values $f(x)$ in the calculation of λ and the error in the computation of $f(x)$ itself. Starting from Equation (2.3.5), considering $\hat{f}(x)$ instead of $f(x)$ and applying Theorem 2.3.2, the estimation of the nonlinearity term is given by:

$$\hat{\lambda} = |\hat{f}(x)(1+\theta_2) - \hat{f}(x')(1+\theta_2) - \hat{f}(y)(1+\theta_2') + \hat{f}(y')(1+\theta_2')|.$$

If it is assumed that $\hat{f}(x) = (1+\theta^{(x)}_{\sqrt{n}})f(x)$, $(1+\theta^{(x)}_{\sqrt{n}})(1+\theta_2) = 1+\theta^{(x)}_{\sqrt{n}+2}$ and one can get:

$$\begin{aligned}\hat{\lambda} &= |f(x)(1+\theta^{(x)}_{\sqrt{n}+2}) - f(x')(1+\theta^{(x')}_{\sqrt{n}+2}) - f(y)(1+\theta^{(y)}_{\sqrt{n}+2}) + f(y')(1+\theta^{(y')}_{\sqrt{n}+2})| \\ &= |\underbrace{f(x)-f(x')-f(y)+f(y')}_{u} + \underbrace{f(x)\theta^{(x)}_{\sqrt{n}+2} - f(x')\theta^{(x')}_{\sqrt{n}+2} - f(y)\theta^{(y)}_{\sqrt{n}+2} + f(y')\theta^{(y')}_{\sqrt{n}+2}}_{v}|.\end{aligned}$$

[4]Note that in the algorithm description provided in Sun et al. (2017b), these lines are missing. The authors probably forgot to mention them during the redaction of the article.

Therefore, using following inequalities, $|u+v| \leq |u|+|v|$, $|u+v| \geq |u|-|v|$, and Theorem 2.3.2, one can conclude that:

$$|\lambda - \hat{\lambda}| \leq \gamma_{\sqrt{n}+2}(|f(x)|+|f(x')|+|f(y)|+|f(y')|) := \varepsilon.$$

This RDG2 strategy will be the one used to detect the interaction structure of high dimensional problems in the algorithm developed in Chapter 3 and in one of the two proposed algorithms in Chapter 6.

2.3.3 Alternative variants of Recursive Differential Grouping

More recently, the promising RDG strategy has been reused and reconceived in different studies in order to further reduce the cost in terms of function evaluations while preserving the same grouping accuracy. Those studies are briefly presented hereafter.

Meselhi et al. (2018) proposed the Enhanced Differential Grouping (EDG) strategy that relies on both DG and RDG. The variables are first identified as separable or nonseparable (but the couple of interacting variables remain unknown) following principles developed in RDG. Then, direct interactions between interacting variables are identified with DG. Finally, indirect interactions between the groups formed in the previous step are checked with RDG. This process proceeding in three steps allows one to save some function evaluations, especially in the case if the function contains separable variables or a low number of nonseparable components.

Chen et al. (2018) introduced an Historical Interdependency based Differential Grouping, called HIDG, relying on a novel criterion which can directly deduce the interaction between some variables without consuming extra function evaluations. This work has been further developed in the Fast Differential Grouping (FDG) algorithm introduced by Ren et al. (2019a). The latter is a reformulation of the RDG strategy in the form of a search process in a binary tree[(5)] that considers subsets of variables as tree nodes. In this process, the interaction in some child nodes can be determined reusing historical information, such as in HIDG, and therefore without consuming extra function evaluations. The same ideas have also been studied in the Efficient Recursive Differential Grouping (ERDG) introduced by Yang et al. (2021).

The Three-level Recursive Differential Grouping (TRDG) proposed in Xu et al. (2020) reduces the depth of recursion of the RDG by dividing the set of variables in three subsets in the recursive step, instead of two subsets for the original RDG. This single change enables to reduce the cost in terms of function evaluations.

Finally, two other variants of the RDG, designed to tackle large-scale overlapping problems, have been proposed by Li et al. (2019) and Sun et al. (2019). They will be described in the Chapter 4 that studies overlapping features in cooperative co-evolution.

[(5)] A binary tree is a tree data structure with at most two children for each node. Formally, it is either empty, or it has a root node, a left binary tree, and a right binary tree (Black (2006)).

Chapter 3

Cooperative co-evolutionary algorithms: our contributions

The CC-EAs based on random and interaction detection decompositions presented in the previous chapter belong to the two main classes of CC-EAs encountered in the literature. Both of them have their pros and cons. On the one hand, CC-EAs embedded with interaction detection decompositions outperform the ones relying on random decompositions in solving additively separable problems. The accurate decomposition clearly facilitates the optimization of subproblems with an EA. On the other hand, even with this accurate decomposition, it may happen that relatively large subcomponents of interacting variables still must be solved with an EA. Therefore, it suffers from the curse of dimensionality and it is preferable to further split the subcomponents even if they are nonseparable. In that case, the best strategy to decompose and optimize large subcomponents remains the seminal random grouping decomposition.

This chapter presents a hybrid decomposition-based CC-EA that performs such a two-step decomposition. It is fully described in Section 3.1. The latter algorithm is then used to solve large-scale constrained problems. This class of problems has been less studied in the CC context. Section 3.2 outlines few works focusing on the optimization of such constrained problems with a CC framework. Then, it describes the changes made to the hybrid decomposition-based CC-EA in order to tackle with constrained problems. The main modifications concern the decomposition which is performed by taking into account the objective function but also all the constraints. Finally, the design of a new benchmark set and the performance analysis of the newly proposed algorithm are presented.

3.1 Hybrid decomposition-based CC-EA

The proposed hybrid decomposition based CC-EA, illustrated in Figure 3.1, performs a two-step decomposition based on both differential and random groupings. It is described as follows:

1. *First decomposition*: Split the n-dimensional decision vector into k disjoint subcomponents $(Sub_1, \ldots, Sub_k)$ with RDG2. The first $k-1$ subcomponents contain the groups of interacting variables; the last one covers all the separable variables.

2. *Second decomposition*: Each subcomponent of interacting variables is randomly split into subsubcomponents (SSCs) containing at most s_{nonsep} variables. The subcomponent of separable variables is arbitrarily split into l SSCs of s_{sep} variables.

3. *Optimization*: Optimize SSCs with an EA for a specified number of iterations in a round-robin strategy.

4. *Cycling*: If the maximum number of cycles is not reached, go to Step 2.

5. *Combination*: Merge solutions of SSCs to build the n-dimensional solution.

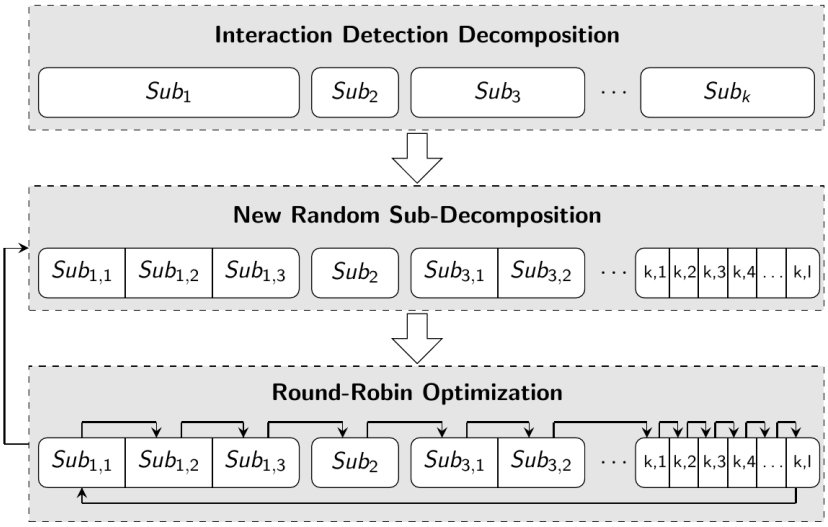


Figure 3.1 – Hybrid decomposition-based CC framework.

In this framework, steps 2 to 4 consist in optimizing each nonseparable subcomponent identified at step 1 with the random decomposition-based CC-EA. The separable subcomponent is evolved with an arbitrarily determined decomposition-based CC-EA. The correct setting of the subcomponent size s for these strategies depends on the features of the problem at hand. For nonseparable problems, a large s increases the probabilities of grouping interacting variables in the same component while for separable ones, a small value of s affects positively the convergence rate of the EA (Trunfio (2015)). Therefore, two different parameters s_{nonsep} and s_{sep} are respectively defined for nonseparable and separable subcomponents. Note that since s_{nonsep} has to be set

to a relatively large value, it may arise that it is greater than the number of variables in the considered subcomponent. In that case, the corresponding class is not split into further SSCs and the optimization of the subcomponent is performed with the standard EA (see Sub_2 in Figure 3.1 for example). This figure also illustrates that the size of SSCs for the separable subcomponent Sub_k is much smaller than for the nonseparable ones. The optimal setting of this parameter is still an open question (Sharawi and El-Abd (2017)). Even for separable problems, a small value is recommended but a too small value may be detrimental (Omidvar et al. (2014)). Furthermore, another parameter that influences the performance of the proposed framework (and more generally that influences all the CC frameworks relying on random groupings), is the specified number of iterations performed for each EA at each cycle. Once again, the optimal setting of this parameter is problem-dependent and is still an open question. For those reasons, the value of s_{sep} and the number of iterations will be arbitrarily fixed for the experimental results presented later in this chapter. Of course, we are aware that better results could be achieved by tuning these parameters but this is beyond the scope of our study.

The population initialization for the optimization of the SSCs at step 3 differs with the kind of considered SSCs. For those classes resulting from the decomposition of a large nonseparable subcomponent into smaller SSCs, the population is randomly generated as explained in Section 1.4. On the other hand, for nonseparable subcomponents that have not be further decomposed into SSCs and for arbitrarily determined separable SSCs, the variables evolved in a SSC are the same at each cycle. In this case, for the optimization in the first cycle, the initial population is randomly generated while for the following cycles, the final population from the previous cycle is reused as the initial population. It allows one to improve the convergence rate, especially for large subcomponents for which it may be quite slow.

3.2 Application to large-scale constrained problems

Although considerable efforts have been made to solve LSGO problems with a CC methodology, most of research projects focus on mono-objective optimization problems. This is the case of all the works presented in Chapter 2. Sayed et al. (2015) were the first to address large-scale constrained (LSC) problems with a CC framework. They extended their previous work on Dependency Identification (Sayed et al. (2012)) to decompose such LSC problems. The optimization of the obtained subproblems is performed with a differential evolution (Price et al. (2005)) algorithm embedded with the superiority of feasible solution for constraint handling (see Section 1.5). Aguilar-Justo and Mezura-Montes (2016) further improved the work of Sayed et al. (2015) by proposing new strategies for the arrangement of the variables and by using simulated annealing (Kirkpatrick et al. (1983)) instead of the greedy search to find the optimal arrangement. Concomitantly, Peng and Hui (2016a) combined the CC Particle Swarm Optimization framework of Li and Yao (2012) with the ε-constrained method (Takahama and Sakai (2005)) to solve LSC problems. In particular, they exploited the potential of the random decomposition with several grouping size selection updat-

ing schemes. In Peng and Hui (2016b), they introduced two new grouping strategies based on the DG methodology presented in Section 2.3.1. The first one focuses only on the objective function (without considering the constraints). The second one considers two grouping schemes: one relative to the objective function, the other relative to the constraints. During the CC optimization, the first grouping is considered if the constraints are respected in compliance with the ε-constrained method, the second grouping is used if the constraints are violated.

Note that the CC framework introduced in this thesis aims to solve LSGO problems. The decomposition is made at the level of the search space, which is divided into subspaces by splitting the set of decision variables into several subsets. Each obtained subproblem is optimized in a CC framework. The latter is not specific to that kind of decomposition. In a broader context, it refers to strategies that rely on the accomplishment of multiple simple tasks to achieve a complex task. For example, solving a constrained optimization problem (even a small-scale one) can be seen such as a complex task. Some CC strategies are devoted to solve that kind of problems. Ghasemishabankareh et al. (2016) proposed to convert a constrained problem into an unconstrained one with the augmented Lagrangian method (Rockafellar (1973)). That problem is optimized in a CC framework tackling two cooperating subpopulations: one focusing on the decision variables of the original problem, the other one focusing on the Lagrangian multipliers. Kieffer et al. (2017) developed an EA that generates one subpopulation for each constraint and optimizes its own local fitness. The latter consists first to try to satisfy its assigned constraints, then to verify the other constraints and finally to optimize the objective function. This kind of strategy will not be discussed further since this is beyond the scope of this thesis.

In this section, the hybrid decomposition-based CC-EA developed in Section 3.1 is applied to solve LSC problems. In particular, it is extended to deal with such problems thanks to a new decomposition which is performed by taking into account the objective function but also all the constraints. Then, the performance of the proposed algorithm is evaluated and compared with a standard EA and with the Random decomposed-based CC-EA developed in Section 2.2 on a set of benchmark problems specially designed for this study. Note that the concepts presented hereafter were first introduced in Blanchard et al. (2017). The main difference concerns the decomposition procedure presented here that has been improved in terms of computational efficiency with respect to the original work.

3.2.1 Extension for large-scale-constrained problems

The first task to extend the hybrid decomposition-based CC-EA in order to solve LSC problems is to adapt the first decomposition step (see Section 3.1) relying on RDG2. From now on, the RDG2 procedure will consider the objective function but also all the constraints. In what follows, the term *response* is indistinguishably used for the objective function or each of the constraints. In this context, Theorem 2.3.3 is extended to define the interaction between two sets of decision variables for constrained problems.

Definition 5 *Let the minimization constrained problem*[1] *given by*

$$\begin{array}{lll} \min\limits_{\vec{x}\in\mathbb{R}^n} & r_0(\vec{x}) & \\ \textit{subject to} & r_i(\vec{x}) \leq 0, & (i=1,\ldots,p). \end{array}$$

Let X be the set of decision variables $\{x_1,\ldots,x_n\}$, $X_1 \subset X$ and $X_2 \subset X$ be two mutually exclusive subsets of decision variables. If there exist two unit vectors $\vec{u}_1 \in U_{X_1}$ and $\vec{u}_2 \in U_{X_2}$[2]*, two real numbers $l_1, l_2 > 0$, and a candidate solution $\vec{x}$ in the decision space, such that*

$$r_i(\vec{x}+l_1\vec{u}_1+l_2\vec{u}_2) - r_i(\vec{x}+l_2\vec{u}_2) \neq r_i(\vec{x}+l_1\vec{u}_1) - r_i(\vec{x}) \tag{3.2.1}$$

for at least one response r_i $(i=0,1,\ldots,p)$, then there is some interaction between the decision variables in X_1 and X_2.

Furthermore, as for the original RDG2, Equation (3.2.1) is not directly used to detect interaction since the roundoff errors have to be managed. Therefore, since each response can potentially depend on strongly diverse orders of magnitude, an independent threshold parameter estimation of ε is computed for each of them. In summary, to perform the decomposition step based on RDG2 for LSC problems, the procedure presented in Section 2.3.2 can be followed considering Algorithm 8 presented hereafter instead of Algorithm 5. In this algorithm, *r_eval* denotes the evaluation of all

Algorithm 8: Interact$(X_1, X_2, \underline{r}, r_eval)$

```
1  r_1 = r_eval(x_{u,l}), r_2 = r_eval(x_{l,m}) ;
2  r_3 = r_eval(x_{u,m}) ;
3  for i = 0, 1, ..., p do
4  |  Δ_1 = r_i − r_{1,i}, Δ_2 = r_{2,i} − r_{3,i} ;
5  |  ε = γ_{√n+2}(|r_i| + |r_{1,i}| + |r_{2,i}| + |r_{3,i}|);
6  |  if |Δ_1 − Δ_2| > ε then
7  |  |  return true ;
8  |  end
9  end
10 return false ;
```

the responses. The response values are stored in $(p+1)$-dimensional vectors. It is common to compute them all at once since in many engineering applications, they often depend on the output of common evaluation chains. Besides, following notations introduced in Section 2.3.2, we have $\underline{r} = r\_eval(\vec{x}_{l,l})$. Note that the decomposition results will be the same as the ones obtained with the methodology based on DG introduced in Blanchard et al. (2017). However, the presented decomposition in this thesis is cheaper in terms of function evaluations since it is based on RDG2 and since it considers all the responses at once.

Considering the following steps of the hybrid decomposition-based CC-EA, only the optimization step needs slight changes. Indeed, in the context of constrained problems, the superiority of feasible points introduced in Section 1.5 is embedded in the EA used to optimize the SSCs.

[1] Only inequality constraints are considered since equality ones can be easily transformed to inequality constraints, see Section 1.5.

[2] See Notation 1 page 33.

3.2.2 Experimental settings

In order to evaluate the performance of the hybrid decomposition-based CC-EA, a benchmark set of LSC problems is required. In the field of evolutionary computation, it is very common to validate new algorithms on the test suites introduced for the special sessions and competitions of the annual Congress on Evolutionary Computation (CEC). However, amongst these test suites, some of them are devoted to constrained real-parameter optimization (Liang et al. (2006); Mallipeddi and Suganthan (2010); Wu et al. (2017); Kumar et al. (2020)), some others to LSGO (Tang et al. (2007, 2009); Xi et al. (2013)), but no one concerns large-scale and constrained optimization. A test suite containing such problems that depend approximately on 500 decision variables has been introduced in Blanchard et al. (2017). Amongst the 10 presented problems, 8 of them are built on the basis of well-known benchmark problems encountered in the constrained optimization literature: G2, G3, G10, Hesse, Speed Reducer and Welded Beam (see Appendix A for details). They are described in Table 3.1. Two other problems, presented in Table 3.2, are taken from Sayed et al. (2015). For these two problems, the objective is to minimize a function, f_9 and f_{10}, respectively, while respecting the same constraints g_1, g_2 and g_3.

$$\min f_1(x) = f_{G2}(x_1 : x_{500})$$
$$\text{s.t.} \quad g_{i,G2}(x_1 : x_{500}) \leq 0, \quad i = 1,2 \qquad (\mathbf{n = 500})$$

$$\min f_2(x) = \sum_{k=1}^{10} f_{G2}\left(x_{50(k-1)+1} : x_{50k}\right)$$
$$\text{s.t.} \quad g_{i,G2}\left(x_{50(k-1)+1} : x_{50k}\right) \leq 0, \quad i = 1,2,\ k = 1,\ldots,10 \qquad (\mathbf{n = 500})$$

$$\min f_3(x) = \sum_{k=1}^{10} f_{G3}\left(x_{50(k-1)+1} : x_{50k}\right)$$
$$\text{s.t.} \quad g_{1,G3}\left(x_{50(k-1)+1} : x_{50k}\right) \leq 0, \quad k = 1,\ldots,10 \qquad (\mathbf{n = 500})$$

$$\min f_4(x) = \sum_{k=1}^{62} f_{G10}\left(x_{8(k-1)+1} : x_{8k}\right)$$
$$\text{s.t.} \quad g_{i,G10}\left(x_{8(k-1)+1} : x_{8k}\right) \leq 0, \quad i = 1,\ldots,6,\ k = 1,\ldots,62 \qquad (\mathbf{n = 496})$$

$$\min f_5(x) = \sum_{k=1}^{83} f_{Hesse}\left(x_{6(k-1)+1} : x_{6k}\right)$$
$$\text{s.t.} \quad g_{i,Hesse}\left(x_{6(k-1)+1} : x_{6k}\right) \leq 0, \quad i = 1,\ldots,6,\ k = 1,\ldots,83 \qquad (\mathbf{n = 498})$$

$$\min f_6(x) = \sum_{k_1=1}^{50} f_{Hesse}\left(x_{6(k_1-1)+1} : x_{6k_1}\right) + \sum_{k_2=1}^{4} f_{G2}\left(x_{300+50(k_2-1)+1} : x_{300+50k_2}\right)$$
$$\text{s.t.} \quad g_{i,Hesse}\left(x_{6(k_1-1)+1} : x_{6k_1}\right) \leq 0, \quad i = 1,\ldots,6,\ k_1 = 1,\ldots,50$$
$$g_{i,G2}\left(x_{300+50(k_2-1)+1} : x_{300+50k_2}\right) \leq 0, \quad i = 1,2,\ k_2 = 1,\ldots,4 \qquad (\mathbf{n = 500})$$

$$\min f_7(x) = \sum_{k=1}^{71} f_{SpeedReducer}\left(x_{7(k-1)+1} : x_{7k}\right)$$
$$\text{s.t.} \quad g_{i,SpeedReducer}\left(x_{7(k-1)+1} : x_{7k}\right) \leq 0, \quad i = 1,\ldots,11,\ k = 1,\ldots,71 \qquad (\mathbf{n = 497})$$

$$\min f_8(x) = \sum_{k=1}^{125} f_{WeldedBeam}\left(x_{4(k-1)+1} : x_{4k}\right)$$
$$\text{s.t.} \quad g_{i,WeldedBeam}\left(x_{4(k-1)+1} : x_{4k}\right) \leq 0, \quad i = 1,\ldots,6,\ k = 1,\ldots,125 \qquad (\mathbf{n = 500})$$

Table 3.1 – Constrained benchmark problems $f_1 - f_8$ (Blanchard et al. (2017)).

$\min f_9(x) = \sum_{k=0}^{4}\left\{\sum_{i=1}^{5} x_{100k+i}^2 + \sum_{i=6}^{10-1}[100(x_{100k+i}^2 - x_{100k+i+1})^2 + (x_{100k+i}-1)^2] + \sum_{i=11}^{55} x_{100k+i}^2 + \sum_{i=56}^{60-1}[100(x_{100k+i}^2 - x_{100k+i+1})^2 + (x_{100k+i}-1)^2] + \sum_{i=61}^{100} x_{100k+i}^2\right\}$
$\min f_{10}(x) = \sum_{k=0}^{4}\left\{\sum_{i=1}^{5}[100(x_{100k+i}^2 - x_{100k+i+50})^2 + (x_{100k+i}-1)^2] + \sum_{i=6}^{50}[x_{100k+i}^2 + x_{100k+i+50}^2]\right\}$
s.t. $g_1(x) = \sum_{k=0}^{4}\left\{\sum_{i=1}^{5} x_{100k+i+25}^2\right\} \leq 0,$ $g_2(x) = \sum_{i=1}^{3-1}[100(x_{i+50}^2 - x_{i+50+1})^2 + (x_{i+50}-1)^2] + \sum_{i=1}^{3-1}[100(x_{i+450}^2 - x_{i+450+1})^2 + (x_{i+450}-1)^2] \leq 0,$ $g_3(x) = 100(x_{10}^2 - x_{490})^2 + (x_{10}-1)^2 + 100(x_{16}^2 - x_{496})^2 + (x_{16}-1)^2 \leq 0$
$x_i \in [-100, 100], \quad i = 1, \ldots, 500,$ for both objective and constraint functions

Table 3.2 – Constrained benchmark problems $f_9 - f_{10}$ (Sayed et al. (2015); Blanchard et al. (2017)). Both of them are subject to the same constraints g_1, g_2 and g_3.

In a general framework, the proposed hybrid decomposition-based CC-EA can be embedded with any kind of EAs to optimize SSCs. For the results presented in this thesis, the proposed algorithms have been implemented in the Minamo software and therefore, the EA used to optimize SSCs is the genetic algorithm of Minamo presented in Section 1.6.

Two versions of the hybrid decomposition-based CC-EA will be considered. For the first one, called HD-CC-EA-I, the parameter s_{nonsep} is set to 50 while for the second one, HD-CC-EA-II, it is fixed to 10. For both of them, the parameter s_{sep} is set to 4 and 50 cycles are performed. They are compared with the random decomposition-based CC-EA, RD-CC-EA, that also performs 50 cycles with subcomponents containing $s = 4$ decision variables. Finally, these CC-EAs are also compared with the standard EA of Minamo. In each algorithm, the population size is chosen as 10 times the number of variables of the considered component. In particular, for the standard EA, it is simply set to 10 times the number of variables of the problem at hand. For the four algorithms, the stopping criterion is the total number of function evaluations which is fixed to 10 millions. The number of generations in each EA instance is automatically computed to reach this stopping criterion. All these settings are summarized in Table 3.3. Finally, using RDG2 to study the interaction structure of such analytical problems may sometimes cause some troubles due to the symmetry of the function and their domain. Therefore, as proposed in Blanchard et al. (2017), alternative lower and upper bounds are used for the detection of interactions. They are defined as:

$$\vec{lb}' = \vec{lb} + \alpha(\vec{ub} - \vec{lb}) \qquad \text{and} \qquad \vec{ub}' = \vec{lb} + \beta(\vec{ub} - \vec{lb}),$$

where $\vec{lb}$ and $\vec{ub}$ stand, respectively, for the original lower and upper bounds, α and β are parameters whose values lie between 0 and 1 ($\alpha < \beta$). In this work, α and β are arbitrarily fixed to 0.33 and 0.96, respectively.

	EA	RD-CC-EA	HD-CC-EA-I	HD-CC-EA-II
population size	$10 \times$ # variables			
function evaluations	10 millions			
number of cycles		50		
number of variables per SSC		$s = 4$	$s_{sep} = 4$	
			$s_{nonsep} = 50$	$s_{nonsep} = 10$

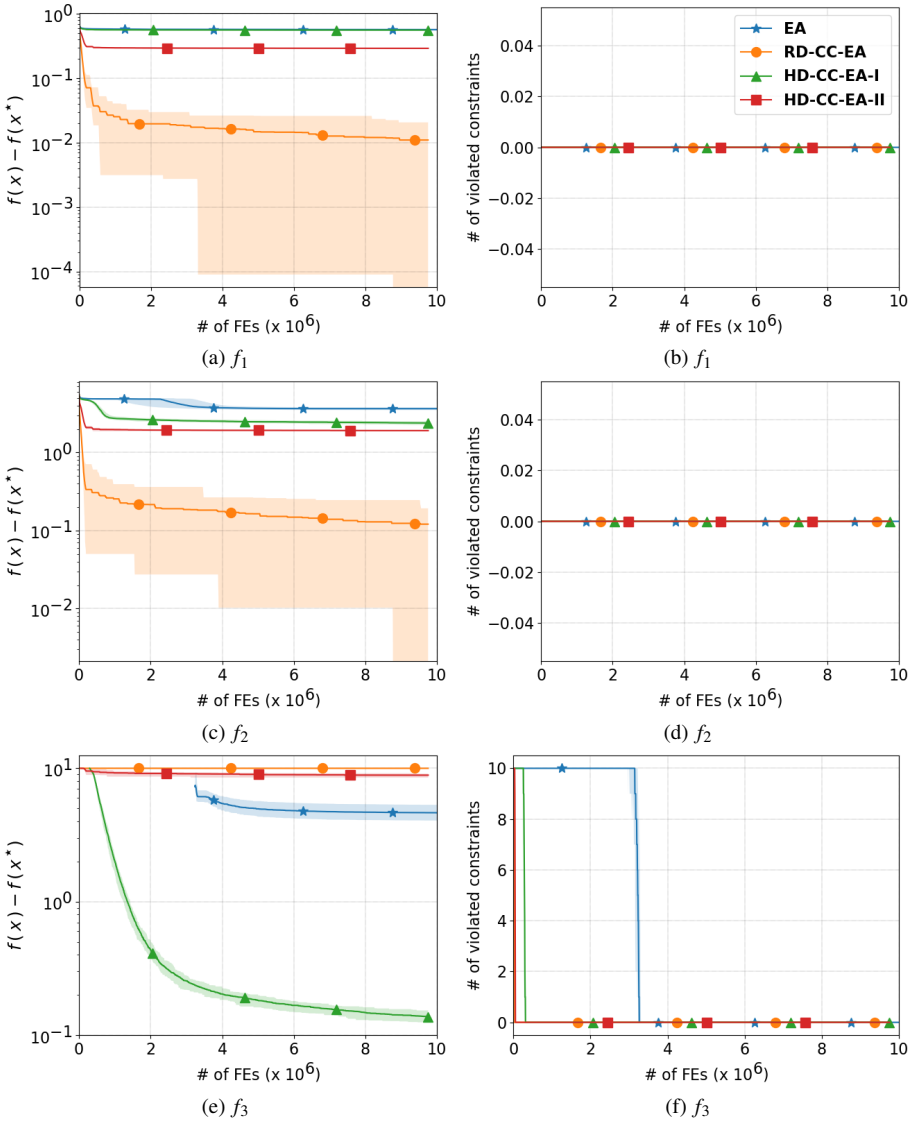
Table 3.3 – Summary of algorithms settings.

Since EAs rely on stochastic processes, several executions of such algorithms lead to different results. Therefore, it is common to repeat the tests several times to analyze their performance. In this work, 51 independent runs[(3)] are executed for each proposed algorithm. Tables reporting statistical features of the final solutions and convergence graphs illustrating the convergence behavior along the optimization process will be presented. The tables allow one to analyze the objective values of the feasible final solutions over the 51 runs. They contain the objective value of the best and the worst feasible solution, the mean, the median and the standard deviation. When no feasible solutions were found over all runs, the '$\times$' symbol is reported in the table. The first kind of convergence graphs describes the evolution of the objective value with respect to the number of Function Evaluations (FEs). More specifically, the y-axis represents, in log-scale, the gap between the objective value of the current solution and the objective value of the optimal solution, i.e., $f(x) - f(x^\star)$. Furthermore, note that since only the objective value of feasible solutions is considered, no line are reported on the graph until a feasible solution is found over the 51 runs. The second kind of convergence graphs describes the evolution of the number of violated constraints with respect to the number of FEs. For both kind of graphs, the solid line depicts the median value while the light-colored area around the solid line represents the gap between the minimal and the maximal values over the 51 runs.

3.2.3 Results

The performance of the four algorithms is analyzed on the set of LSC problems presented above. The obtained results are provided in Table 3.4 for median objective values of the final solutions. The best median values among the four tested algorithms are marked in **bold face**. Detailed statistical values are given in Table 3.5. Convergence graphs are also proposed in Figures 3.2, 3.3, 3.4 and 3.5. As a first observation, it is clear that the HD-CC-EAs outperform the standard EA and the RD-CC-EA. Let us now examine in details the results on each benchmark problem.

[(3)]The general rule of thumb is to compute a few tens of runs. For example, in the first CC studies of Potter and De Jong (1994); Liu et al. (2001); van den Bergh and Engelbrecht (2004); Shi et al. (2005), 50 runs are executed. Later, most of the works studying the large scale instances of the CEC competitions (Tang et al. (2007, 2009); Xi et al. (2013)) only perform 25 runs such as proposed in the experimental protocol of these competitions. In our study case, about fifty runs were computed to make the results analysis more reliable. In particular, an odd number of runs (51) is chosen in order to get a median value that corresponds to a specific run.

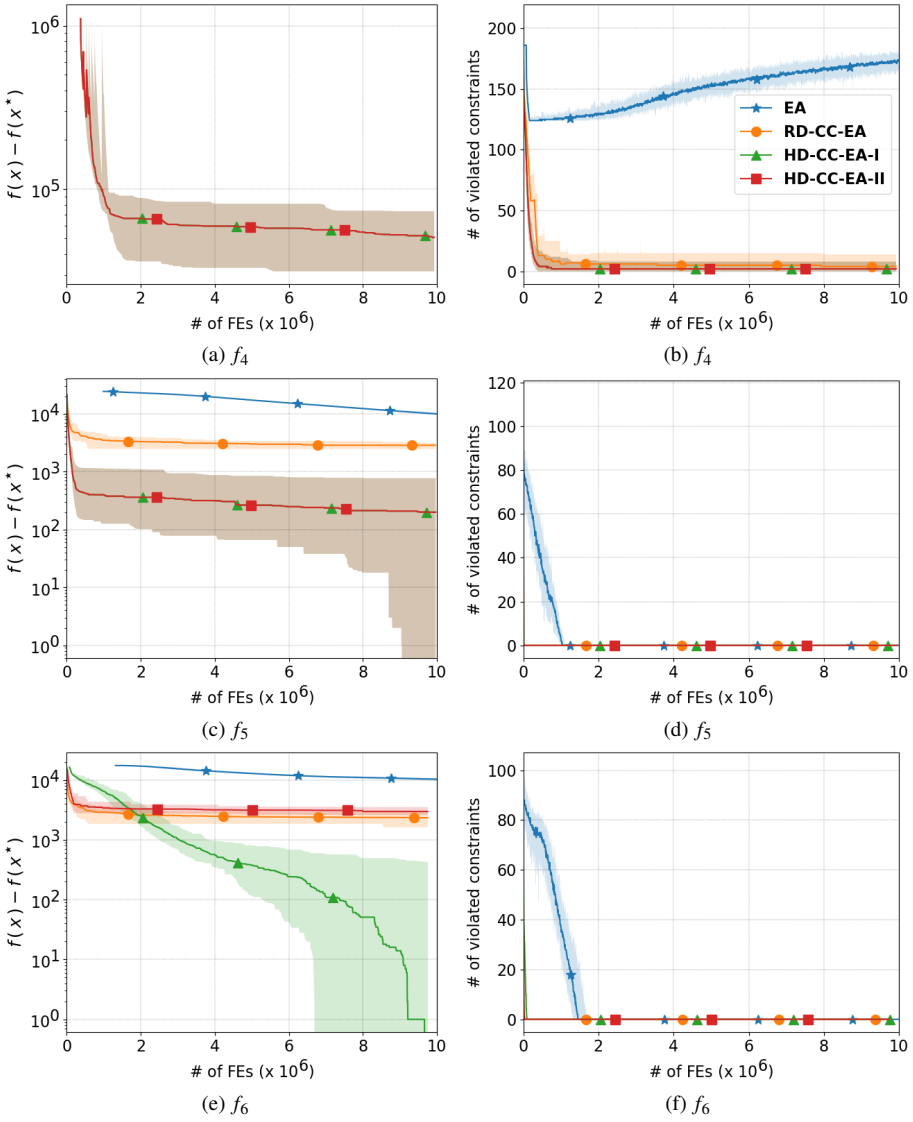


(a) f_1 (b) f_1

(c) f_2 (d) f_2

(e) f_3 (f) f_3

Figure 3.2 – Convergence graphs for problems f_1, f_2 and f_3. Left side: $f(x) - f(x^\star)$ in log-scale, no line are reported on the graph until a feasible solution is found over the 51 runs. Right side: number of violated constraints. EA (blue stars), RD-CC-EA (orange circles), HD-CC-EA-I (green triangles), HD-CC-EA-II (red squares). The solid line depicts the median value while the light-colored area around the solid line represents the interval between the minimal and the maximal values over the 51 runs.

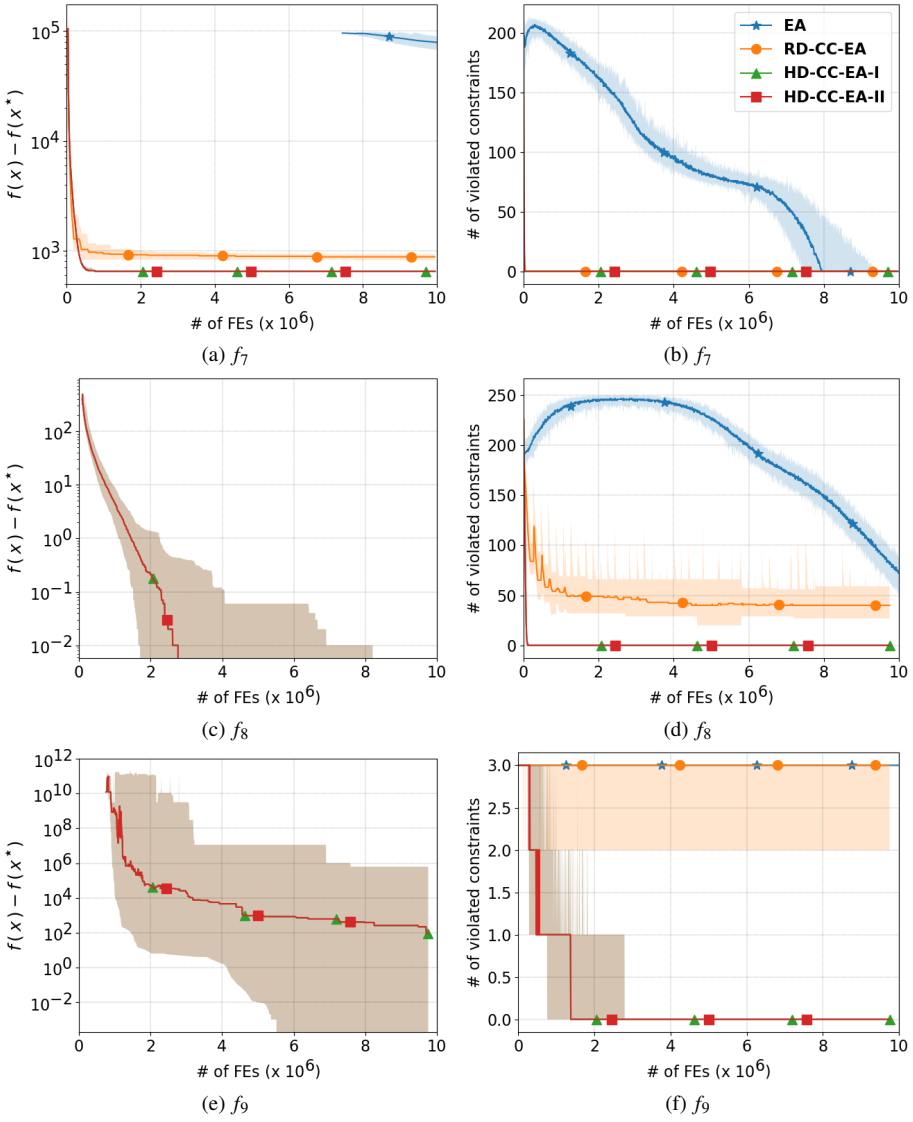


(a) f_4 (b) f_4

(c) f_5 (d) f_5

(e) f_6 (f) f_6

Figure 3.3 – Convergence graphs for problems f_4, f_5 and f_6. Left side: $f(x) - f(x^\star)$ in log-scale, no line are reported on the graph until a feasible solution is found over the 51 runs. Right side: number of violated constraints. EA (blue stars), RD-CC-EA (orange circles), HD-CC-EA-I (green triangles), HD-CC-EA-II (red squares). The solid line depicts the median value while the light-colored area around the solid line represents the interval between the minimal and the maximal values over the 51 runs.



(a) f_7 (b) f_7

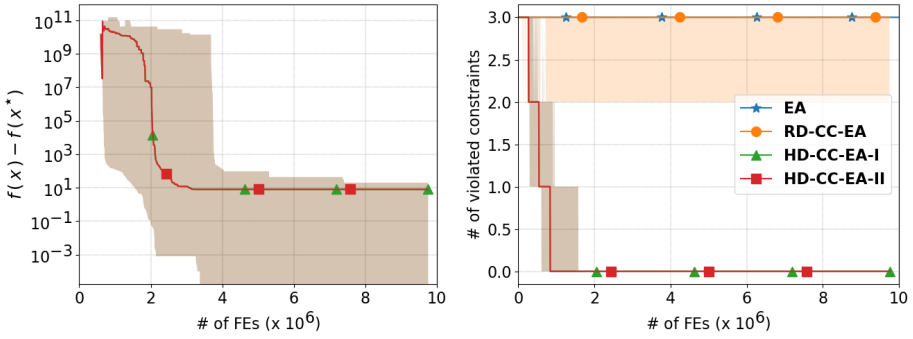
(c) f_8 (d) f_8

(e) f_9 (f) f_9

Figure 3.4 – Convergence graphs for problems f_7, f_8 and f_9. Left side: $f(x) - f(x^\star)$ in log-scale, no line are reported on the graph until a feasible solution is found over the 51 runs. Right side: number of violated constraints. EA (blue stars), RD-CC-EA (orange circles), HD-CC-EA-I (green triangles), HD-CC-EA-II (red squares). The solid line depicts the median value while the light-colored area around the solid line represents the interval between the minimal and the maximal values over the 51 runs.

	EA	RD-CC-EA	HD-CC-EA-I	HD-CC-EA-II
f_1	-1.82e-01	**-7.43e-01**	-1.93e-01(-)	-4.62e-01(-)
f_2	-2.81e+00	**-6.34e+00**	-4.07e+00(-)	-4.55e+00(-)
f_3	-5.36e+00	-3.81e-09	**-9.86e+00**(+)	-1.12e+00(+)
f_4	×	×	**4.88e+05**(+)	**4.88e+05**(+)
f_5	-1.57e+04	-2.27e+04	**-2.54e+04**(+)	**-2.54e+04**(+)
f_6	-9.55e+03	-1.75e+04	**-1.99e+04**(+)	-1.69e+04(-)
f_7	2.91e+05	2.13e+05	**2.13e+05**(+)	**2.13e+05**(+)
f_8	×	×	**2.16e+02**(+)	**2.16e+02**(+)
f_9	×	×	**1.02e+02**(+)	**1.02e+02**(+)
f_{10}	×	×	**1.20e+01**(+)	**1.20e+01**(+)

Table 3.4 – Median objective value on 51 runs for benchmark problems. '×' means no feasible solution has been found over all runs. Best values are marked in bold face. (+), (-) and (=) represent the fact that HD-CC-EA-I and HD-CC-EA-II are significantly better than, worse than, or equivalent to RD-CC-EA according to the Wilcoxon rank-sum test with 5% significance level (Wilcoxon (1945)).



(a) f_{10} (b) f_{10}

Figure 3.5 – Convergence graphs for problem f_{10}. Left side: $f(x) - f(x^\star)$ in log-scale, no line are reported on the graph until a feasible solution is found over the 51 runs. Right side: number of violated constraints. EA (blue stars), RD-CC-EA (orange circles), HD-CC-EA-I (green triangles), HD-CC-EA-II (red squares). The solid line depicts the median value while the light-colored area around the solid line represents the gap between the minimal and the maximal values over the 51 runs.

f_1					
Minamo	Best	Worst	Mean	Median	Std
EA	-2.10e-01	-1.77e-01	-1.84e-01	-1.82e-01	6.68e-03
RD-CC-EA	-7.54e-01	-7.34e-01	-7.44e-01	**-7.43e-01**	4.81e-03
HD-CC-EA-I	-1.97e-01	-1.91e-01	-1.93e-01	-1.93e-01	1.54e-03
HD-CC-EA-II	-4.68e-01	-4.58e-01	-4.62e-01	-4.62e-01	2.02e-03
f_2					
Minamo	Best	Worst	Mean	Median	Std
EA	-2.96e+00	-2.67e+00	-2.80e+00	-2.81e+00	6.64e-02
RD-CC-EA	-6.46e+00	-6.27e+00	-6.35e+00	**-6.34e+00**	4.92e-02
HD-CC-EA-I	-4.21e+00	-3.95e+00	-4.08e+00	-4.07e+00	5.00e-02
HD-CC-EA-II	-4.60e+00	-4.51e+00	-4.55e+00	-4.55e+00	1.89e-02
f_3					
Minamo	Best	Worst	Mean	Median	Std
EA	-5.94e+00	-4.68e+00	-5.34e+00	-5.36e+00	2.82e-01
RD-CC-EA	-2.41e-05	-6.58e-11	-5.35e-07	-3.81e-09	3.34e-06
HD-CC-EA-I	-9.88e+00	-9.85e+00	-9.86e+00	**-9.86e+00**	6.04e-03
HD-CC-EA-II	-1.47e+00	-8.87e-01	-1.14e+00	-1.12e+00	1.45e-01
f_4					
Minamo	Best	Worst	Mean	Median	Std
EA	×	×	×	×	×
RD-CC-EA	×	×	×	×	×
HD-CC-EA-I	4.69e+05	5.10e+05	4.87e+05	**4.88e+05**	1.18e+04
HD-CC-EA-II	4.69e+05	5.10e+05	4.87e+05	**4.88e+05**	1.18e+04
f_5					
Minamo	Best	Worst	Mean	Median	Std
EA	-1.61e+04	-1.51e+04	-1.57e+04	-1.57e+04	1.78e+02
RD-CC-EA	-2.31e+04	-2.25e+04	-2.27e+04	-2.27e+04	1.79e+02
HD-CC-EA-I	-2.56e+04	-2.48e+04	-2.54e+04	**-2.54e+04**	1.68e+02
HD-CC-EA-II	-2.56e+04	-2.48e+04	-2.54e+04	**-2.54e+04**	1.68e+02
f_6					
Minamo	Best	Worst	Mean	Median	Std
EA	-1.02e+04	-9.17e+03	-9.61e+03	-9.55e+03	2.63e+02
RD-CC-EA	-1.83e+04	-1.69e+04	-1.76e+04	-1.75e+04	2.47e+02
HD-CC-EA-I	-1.99e+04	-1.94e+04	-1.98e+04	**-1.99e+04**	1.21e+02
HD-CC-EA-II	-1.73e+04	-1.64e+04	-1.69e+04	-1.69e+04	2.07e+02
f_7					
Minamo	Best	Worst	Mean	Median	Std
EA	2.80e+05	3.02e+05	2.91e+05	2.91e+05	4.49e+03
RD-CC-EA	2.13e+05	2.13e+05	2.13e+05	2.13e+05	2.54e+01
HD-CC-EA-I	2.13e+05	2.13e+05	2.13e+05	**2.13e+05**	3.94e-09
HD-CC-EA-II	2.13e+05	2.13e+05	2.13e+05	**2.13e+05**	3.94e-09
f_8					
Minamo	Best	Worst	Mean	Median	Std
EA	×	×	×	×	×
RD-CC-EA	×	×	×	×	×
HD-CC-EA-I	2.16e+02	2.16e+02	2.16e+02	**2.16e+02**	8.88e-04
HD-CC-EA-II	2.16e+02	2.16e+02	2.16e+02	**2.16e+02**	8.88e-04
f_9					
Minamo	Best	Worst	Mean	Median	Std
EA	×	×	×	×	×
RD-CC-EA	×	×	×	×	×
HD-CC-EA-I	1.29e+01	6.20e+05	1.78e+04	**1.02e+02**	8.68e+04
HD-CC-EA-II	1.29e+01	6.20e+05	1.78e+04	**1.02e+02**	8.68e+04
f_{10}					
Minamo	Best	Worst	Mean	Median	Std
EA	×	×	×	×	×
RD-CC-EA	×	×	×	×	×
HD-CC-EA-I	3.99e+00	2.39e+01	1.21e+01	**1.20e+01**	4.74e+00
HD-CC-EA-II	3.99e+00	2.39e+01	1.21e+01	**1.20e+01**	4.74e+00

Table 3.5 – Statistical objective value on 51 runs for benchmark problems. '×' means no feasible solution has been found over all runs. Best median values are marked in bold face.

For problem f_1, each variable interacts with all other variables. It is therefore fully nonseparable and the HD-CC-EAs create only one nonseparable component containing all the variables. Therefore, they act like the RD-CC-EA. The three algorithms only differs in the number of variables per SSCs. The influence of this parameter on the algorithm performance is specific to each problem. For this particular case, a smaller value seems to be more convenient. On the contrary, for problem f_3, the HD-CC-EAs form 10 nonseparable subcomponents containing 50 variables. Results show that in this case, a large number of variables in each SSC is more suitable since HD-CC-EA-I performs better than HD-CC-EA-II. In particular, both HD-CC-EAs benefit from the efficient grouping obtained by the RDG2 procedure and offer better results than the RD-CC-EA. This is not always the case. For example, for problem f_2, although the 10 nonseparable components are correctly identified in the HD-CC-EAs, they are outperformed by the RD-CC-EA that benefits from the small subcomponent size. These first observations show that the performance of the algorithms on different problems depends on their degree of nonseparability. For highly nonseparable problems, it is better to work with larger SSCs and vice versa. The key challenge is to choose the suitable compromise. If the SSCs are too large, the internal EA shows some difficulties to converge by essence, but if they are too small, it is not suitable for nonseparable problems. Another problem that illustrates the effect of this parameter is f_6. It contains one separable subcomponent of 300 variables and 4 nonseparable subcomponents of 50 variables. For this problem, the better results achieved with the HD-CC-EA-I indicate that it is preferable to keep the subcomponent of 50 variables as a whole.

The two variants of the HD-CC-EA acts similarly on problems f_4, f_7, f_8, f_9 and f_{10}. For these problems, the nonseparable subcomponents contain less than 10 variables and therefore, they are not further split into smaller SSCs with neither of the HD-CC-EAs. Moreover, the efficient decomposition reached with the HD-CC-EAs clearly helps to find a feasible solution faster. For example, Figure 3.4 shows that for problem f_8, the number of violated constraints quickly drops to 0 for HD-CC-EAs while it stabilizes around 50 for the RD-CC-EA. This behavior can also be seen for f_9 and f_{10} for which 2 or 3 constraints remain violated during all the optimization with RD-CC-EA while the HD-CC-EAs are rapidly able to find feasible solutions. Furthermore, for the standard EA, it is also very difficult to find feasible solutions. Indeed, as it can be seen from the convergence graphs of f_3, f_7 and f_8, the number of violated constraints sometimes decreases but only after considerable efforts in terms of FEs. It may also seem odd that the number of violated constraints increases for f_4 and also for early stages of f_7 and f_8. However, it is not in contradiction with the superiority of feasible individuals. Indeed, the sum of the violated constraints is really decreasing even if the number of violated constraints is increasing.

Problem f_5 is fully separable. Both the RD-CC-EA and the HD-CC-EAs act on SSCs containing 4 variables. For each cycle, the RD-CC-EA considers new random groups of variables and thus new populations. The HD-CC-EAs use the same arbitrarily fixed groups of variables for all the cycles and therefore continue to evolve the same populations from one cycle to the next. In this context, the HD-CC-EAs converge faster than the RD-CC-EA.

All the results presented in this section rely on the number of FEs as a measure of the cost, as it is frequently the case to compare evolutionary algorithms. We close the results analysis with a few words about another important measure of the cost: the execution time. Although very few studies mention it, CC-EAs require far less computational resources than the standard EAs on which they rely to complete an optimization with a given number of FEs. Potter and De Jong (1994) already observed it in their paper presenting the first attempt to solve LSGO problems with a CC framework. The reason is quite simple: for the standard EA, each individual to be evaluated is obtained following the process of genetic operators that act on all the decision variables, while for the CC-EA, the operators only act on a subset of the variables and therefore require fewer computational resources. In our study case, the standard EA of Minamo takes approximately 8 hours[(4)] to address each of the 10 benchmark problems. The three CC variants, the RD-CC-EA and the two HD-CC-EAs, are much faster. They take between 10 minutes and 2 hours, depending on the considered benchmark problems and the CC variants, to complete the optimization. Indeed, the cost of the FEs themselves for problems f_1, f_2, f_3, f_9 and f_{10} is quite small and therefore, the CC variants only need a few tens of minutes to run the optimization. On the contrary, the remaining benchmark problems have quite expensive FEs (because they involve a very large number of responses (objective and constraints)) and therefore, the CC variants require up to 2 hours to complete the execution. Finally, among the CC variants, the execution time also depends on the size of subcomponents created during the decomposition step: the smaller the size of the subcomponents, the faster the execution. Therefore, due to the choice of parameters defined in Table 3.3, the RD-CC-EA is the fastest among the three CC variants. It is followed by the HD-CC-EA-II and then, by the HD-CC-EA-I.

3.3 Conclusion

This chapter presents a hybrid decomposition-based CC-EAs that performs a two-step decomposition based on both differential and random groupings. As a first step, it splits the n-dimensional decision vector into disjoint subcomponents with RDG2. As a second step, it further splits large subcomponents (difficult to manage with a standard EA) into smaller subsubcomponents (much easier to manage with the EA). In particular, the separable variables are arbitrarily split (for the whole optimization process) into several SSCs while the variables of large nonseparable subcomponents are randomly split in SSCs. The interaction between those SSCs are addressed by repeating several cycles with random decompositions.

The proposed hybrid decomposition-based CC-EA has shown good capabilities to tackle high dimensional problems. In particular, it has been extended to deal with large-scale constrained problems and its performance has been assessed on a benchmark set of such problems specially designed for this study. It clearly outperformed the random decomposition-based CC-EA and the standard EA, especially on dealing

[(4)] Experiments were conducted on Haswell Intel E5-2680v3 processors installed in the Tier-I supercomputer of the Fédération Wallonie-Bruxelles, see `https://tier1.cenaero.be/en` for further details.

with the violated constraints. Finally, although the proposed decomposition offers two different settings for the size of nonseparable and separable SSCs, the optimal choice of these values is still a very open question, as for most of CC-EAs.

Chapter 4

Overlapping features in cooperative co-evolution

One of the main features that distinguishes several kinds of cooperative co-evolutionary algorithms is the decomposition strategy. As presented in Chapter 2, the two main types of decomposition that emerged in CC-EAs are the random and the interaction detection decompositions. The first one tries to catch interacting variables in a same subcomponent by means of an iterative scheme of random grouping decompositions. The second one starts by studying the interaction structure and performs a unique decomposition for the whole optimization. In addition to these two strategies, a hybrid one, taking benefits from both of them, has also been presented in Chapter 3. The CC-EAs embedded with an interaction detection-based decomposition, as well as the hybrid one, provide better results than random grouping for problems that can be efficiently decomposed, i.e., separable or partially separable problems.

However, that kind of problems is not representative of large-scale problems encountered in many real-world applications. Indeed, most of them are composed of several components that potentially interact with each other. For example, in mechanical engineering, the design of turbomachines such as aircraft propellers relies on a complex optimization problem that captures aerodynamic, acoustic and mechanical constraints, all intrinsically linked (Baert et al. (2020)). Therefore, traditional CC-EAs encounter some difficulties to tackle such problem instances. On the one hand, interaction detection decomposition-based CC-EAs would assign all the variables into a single group and would therefore fail to reduce the problem dimensionality. On the other hand, random decomposition-based CC-EAs would allow one to break the dimensionality but would also produce poor results. Indeed, the random grouping is not strong enough to efficiently optimize problems with interconnect components.

These problems that share interacting variables between components, also referred as *overlapping problems* (see Figure 4.1), raise new questions. What would be the best strategy to decompose and optimize them with CC-EAs ? How to per-

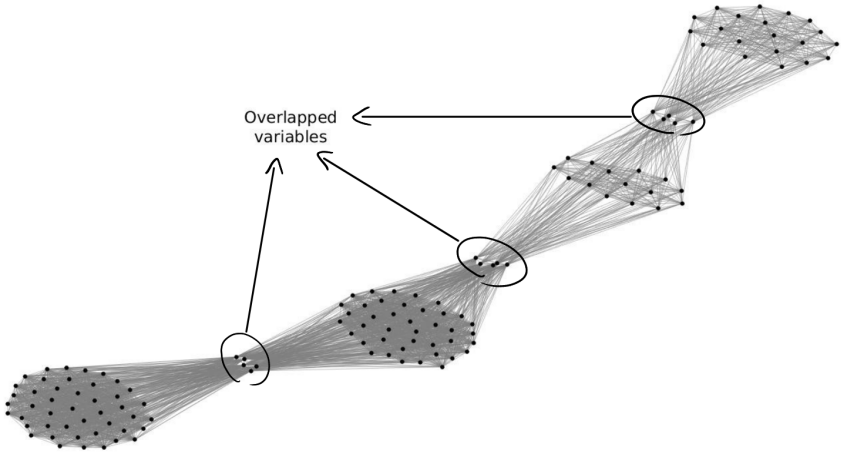


Figure 4.1 – Interaction structure of a large-scale overlapping problem. Each node represents a decision variable. An edge connects two nodes if the corresponding decision variables interact with each other. Four components, in which the interaction graph is fully-connected, can be identified. These components are linked through some variables that interact with the variables of several components. They are called *overlapped variables*.

form an efficient decomposition? Could we imagine performing a decomposition with shared/overlapped variables between subcomponents? If yes, how to manage these overlapped variables during the optimization? These questions are addressed throughout this chapter. First, studies from the state of the art related to overlapping features in cooperative co-evolution are presented in Section 4.1. Then, a new overlapped decomposition-based CC-EA, that has also been presented in Blanchard et al. (2021), is introduced in Section 4.2. It is fully described and its performance is compared with a standard CC framework on large-scale overlapping unconstrained problems. An extended analysis of the new algorithm on the Rosenbrock function is also provided in Section 4.3. Finally, a discussion on promising tracks and limitations of the approach closes the chapter.

4.1 Related work

This section presents the recent studies on overlapping. They can be classified in two categories: overlapped CC strategies to tackle various LSGO problems (A, B and C) and non-overlapped strategies to tackle LSGO overlapping problems (D and E).

A. Overlapped cooperative particle swarm optimizer with interdependence learning

The Cooperative Particle Swarm Optimizer (CPSO) proposed by Sun et al. (2012) is, to the best of our knowledge, the first CC framework performing a decomposition that shares variables among several subcomponents. The latter relies on a statistical vari-

able interdependence learning very close to the interaction learning of the differential grouping strategy presented in Section 2.3.1[(1)]. However, although the interaction learning is similar, the decomposition based on the uncovered structure is completely different. In Sun et al. (2012), the decomposition of an n-dimensional problem first consists in initializing n subsets $S_1 = \{x_1\}, S_2 = \{x_2\}, \ldots, S_n = \{x_n\}$, where x_i is called the core of subset S_i. Then, for each subset S_i, variables that interact with x_i are added to S_i. The overlapped variables are managed using the following CC framework:

1. Initialize an n-dimensional context vector randomly.
2. Perform the problem decomposition.
3. Optimize each subproblem with a separate PSO. For the function evaluations, the variables of the context vector are used for the completion of partial solutions.
4. Update the context vector by concatenating the optimized subproblem cores.
5. If the maximum of allowed FEs is reached, stop. Otherwise, go to step 3.

However, although the study provides an interesting overlapped approach, it has not been developed to optimize overlapping problems. In this context, the choice of the benchmark set does not bring to the fore the utility and the efficiency of the overlapped decomposition. Furthermore, the results for 1000-d problems are presented considering 2×10^8 FEs. This number is very high compared to the 3×10^6 usually considered in CEC special sessions on LSGO (Tang et al. (2009); Xi et al. (2013)). Therefore, we cannot conclude if the proposed algorithm provides good results with a smaller (and more reasonable) budget in terms of FEs. Finally, some details remain unclear in the presented methodology (amongst others: How long should be the optimization of step 2? Is it better to perform a long optimization with therefore few updates of the context vector at step 4 or a shorter one that allows one to perform more frequent updates? What would be the best compromise?).

B. Factored evolutionary algorithm

The Factored Evolutionary Algorithm (FEA) presented by Strasser et al. (2017) is a new class of evolutionary algorithm that factors the objective function by creating overlapping subcomponents. Each of them can be optimized with any kind of EA. It relies on three major subfunctions: (a) solving, (b) competition, and (c) sharing.

(a) The solving function allows each factor to optimize its set of variables while keeping remaining variables to a fixed value.

(b) The competition function creates a complete solution $G = (\bar{x}_1, \bar{x}_2, \ldots..\bar{x}_n)$ used by the factors to evaluate partial solutions. For every variable x_i, the function iterates over each subpopulation containing x_i. For each of them, the value of the variable x_i of the best individual is substituted into G and the produced solution is evaluated with the objective function. The fittest value of x_i over all the subpopulations containing x_i is recorded in G and the function moves to the

[(1)] In fact, interaction learning schemes presented by Sun et al. (2012) and Omidvar et al. (2013) have been developed over the same period and are quite close. The work proposed by Omidvar et al. (2013) has received considerable attention and has become a reference in the "CC community" while the work by Sun et al. (2012) has received significantly less attention.

next variable. This procedure does not guarantee to find the best combination of values from each subcomponent. However, the iterations over the variables and over the subpopulations are performed in random sequence. It allows the algorithm to explore different combinations throughout the whole optimization.

(c) The sharing function simply shares the complete solution G among each subcomponent in order to evaluate partial solutions. It also replaces the worst individual of each subpopulation with a new one composed of the variables of the complete solution G.

Unfortunately, although the approach provided by Strasser et al. (2017) is attractive, the study is not devoted to overlapping problems, it is limited to 50-dimensional problems and it does not propose any automatic decomposition strategy based on the interaction structure. Furthermore, it suggests good performances of the FEA but the results are presented in a biased manner. Indeed, the FEA and the standard EA are compared with the same number of iterations but the former requires many more FEs than the latter at each iteration.

C. Cooperative co-evolution with overlapping of influential decision variables

The framework introduced by Song et al. (2017) does not use the overlapping to facilitate the decomposition of overlapping problems. Instead, it takes advantages of overlapped variables in order to allocate more computational resources to the decision variables that have strong impacts on the optimization. Those influential variables are identified using a delta-disturbance strategy. The latter measures the impact of variables by successively disturbing each dimension with a small delta value and by measuring the impact on the objective function. The overlapped grouping is performed in two steps. Firstly, mutually-exclusive groups of variables are created with any standard grouping strategy. Secondly, the influential variables identified by the delta-disturbance strategy are added to some subsets in order to be overlapped. In this way, they benefit, in the CC framework, from additional computational resources in comparison with variables appearing in only one subset. The study also introduces the following rule to manage the overlapped variables: when the groups are successively optimized in the CC framework, the values of overlapped variables in previous groups are replaced with the value in the current groups.

D. Differential grouping with spectral clustering

The decomposition based on differential grouping with spectral clustering designed by Li et al. (2019) aims to split overlapping LSGO problems into subproblems focusing on mutually-exclusive groups of variables. In this framework, the variables are treated as nodes of an undirected graph, the adjacency matrix of this graph being the design structure matrix Θ obtained with DG2 (see Section 2.3.1). The latter is used as the similarity matrix of spectral clustering. Based on this matrix, another matrix, called unnormalized Laplacian matrix L, is computed. A *k-means* algorithm (Jain (2010)) is used to cluster the k eigenvectors of L, corresponding to the k smallest eigenvalues, in the eigenvectors space. In that way, the graph is divided into k subgraphs and the groups of variables used in the CC framework are given by the variables

of the nodes in each subgraph. The obtained CC framework embedded with a differential evolution optimizer shows promising results on LSGO overlapping problems. However, its major drawback is that the number of groups k needs to be artificially set.

E. Recursive differential grouping for overlapping problems

The new recursive differential grouping, called RDG3, introduced by Sun et al. (2019), is a slightly modified version of the RDG2 presented in Section 2.3.2. This new version aims to decompose overlapping problems into disjoint subproblems by breaking the linkage at shared variables between components. Such a breaking is obtained by imposing a new condition that limits the size of the components. As a reminder, in Algorithm 7 (see page 35), the interaction detection between variables of X_1 and remaining variables is repeated until no more variable is added to X_1. In the new RDG3, it is repeated until at least one of the two following stopping criteria is verified:

1. no more variable is added to X_1 (such as for RDG2) ;
2. $X_1^\star$ contains more that ε_n variables, where ε_n needs to be artificially set.

From an algorithmic point of view, the new RDG3 can be described with Algorithm 7 (page 35) by simply replacing the "if" condition at line 6 by the following one:

$$\text{"}\textbf{if } |X_1^\star| = |X_1| \textbf{ or } |X_1^\star| > \varepsilon_n \textbf{ then } \ldots\text{"}.$$

The CC framework based on this new decomposition and using the Covariance Matrix Adaptation Evolution Strategy (Hansen and Ostermeier (2001)) as components optimizer is very efficient and is the winner of the CEC 2019 competition on LSGO[(2)]. Its main current weakness concerns the choice of ε_n that needs to be artificially set.

4.2 Overlapped decomposition-based CC-EA

Of course it may be thought that the best way to optimize overlapping LSGO problems in a CC framework is through overlapped strategies. However, none of the studies presented in the previous section (and to the best of our knowledge, none of the studies in the literature, except the one on which this section relies (Blanchard et al. (2021))), provide such a framework. The new overlapped decomposition-based CC-EA presented in this section responds to that expectation and aims to tackle LSGO overlapping problems within an overlapped CC framework. Therefore, it has to deal with subcomponents that share several variables. This raises two fundamental questions. The first one concerns the way in which an intelligent decomposition strategy can efficiently detect overlapped variables and share them among several components. A new strategy, based on recursive differential grouping, that addresses this issue is presented in Section 4.2.1. The second question is relative to the management of overlapped variables during the optimization, in particular for function evaluations. It is addressed in the overlapped cooperative co-evolutionary framework described in Section 4.2.2. Finally, the performance of the resulting overlapped decomposition-based CC-EA is analyzed in Section 4.2.3.

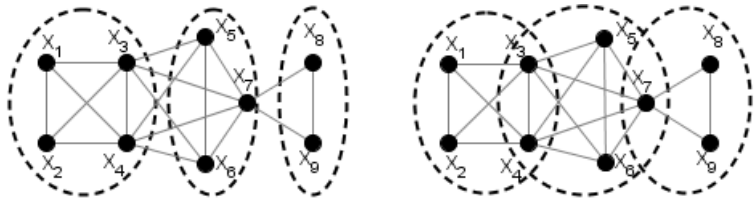
[(2)] See `http://www.tflsgo.org/special_sessions/cec2019`, last visited December 10, 2020.

4.2.1 Overlapped Recursive Differential Grouping

The newly proposed Overlapped Recursive Differential Grouping (ORDG) is another alternative version of the RDG2 presented in Section 2.3.2. The latter is modified for the purpose of identifying variables that make the link between several components and share them among these subcomponents. Therefore, the new framework deals with overlapping problems in some way other than the RDG3 presented in Section 4.1. The difference between the two strategies is illustrated in Figure 4.2. In the presented interaction graph, three main components can be identified:

$$S_1 = \{x_1, x_2, x_3, x_4\},\ S_2 = \{x_3, x_4, x_5, x_6, x_7\} \text{ and } S_3 = \{x_7, x_8, x_9\}.$$

Within those components, the interaction graph is fully-connected, meaning that variables strongly interact with each other. Moreover, variables from different components do not directly interact with each other, i.e., $\forall\, i, j\,(i \neq j),\ k, l\,(k \neq l)$ such that $x_i \in S_k \backslash S_l$ and $x_j \in S_l \backslash S_k,\ x_i$ does not interact with x_j. In order to split such a problem into subproblems focusing on mutually exclusive subsets of variables, the RDG3 breaks the linkage at shared variables and produces the grouping illustrated in Figure 4.2a. It may not be the best approach since x_3 and x_4 (resp. x_7) are not optimized with $x_5,\ x_6$ and x_7 (resp. x_8 and x_9) while they directly interact with each other. By means of the allowed overlap between components, the new ORDG provides a grouping that prevents from breaking these important linkages. It is illustrated in Figure 4.2b.



(a) RDG3 (b) ORDG

Figure 4.2 – The two obtained decompositions for an overlapping problem using RDG3 and ORDG strategies, respectively (Blanchard et al. (2021).

The new ORDG is fully described in Algorithm 9. New features specific to the ORDG find themselves in the "else" statement at line 11. The algorithm goes into this statement, as well as in RDG2, if some interaction has been identified between X_1 and X_2 at line 5[(3)]. The variables in X_2 that interact with X_1 are known and are added to X_1 to produce the set $X_1^\star$. However, the variables in X_1 responsible of the interaction are unknown. They should be identified to perform the desired overlapped decomposition. If X_1 contains a single variable, the latter is inherently the one responsible of the interaction. The algorithm passes through the same update that for the RDG2 before moving on the next iteration (lines 12-13). Otherwise (i.e., if X_1 contains several variables), those interacting with X_2 are detected with the `L_Inter` function (see Algorithm 10) that exploits the recursive mechanism for interaction detection again. Then, the new instructions in lines 16-18 deliver the desired overlapped decomposition.

[(3)]The `R_Inter` function is the same that the one defined for the RDG. Its pseudo-code is given in Algorithm 6 at page 35.

Algorithm 9: Overlapped Recursive Differential Grouping($func$)

$seps = \{\}$, $nonseps = \{\}$;
$X_1 = \{x_1\}$, $X_2 = \{x_2, \ldots, x_n\}$;
$\underline{f} = f(\vec{x}_{l,l})$;
while $X_2 \neq \{\}$ **do**
  $X_1^\star$ =R_Inter($X_1, X_2, \underline{f}, func$) ;
  if $|X_1^\star| = |X_1|$ **then**
    if $|X_1| = 1$ **then** $seps = seps \cup X_1$;
    else $nonseps = nonseps \cup \{X_1\}$;
    $X_1 = \{x_j\}$ s.t. $j \leq i \; \forall x_i \in X_2$;
    $X_2 = X_2 \setminus \{x_j\}$;
  else
    if $|X_1| = 1$ **then**
      $X_1 = X_1^\star$, $X_2 = X_2 \setminus X_1$;
    else
      $X_1^{\star\star} = L\_inter(X_1, X_2, \underline{f}, func)$;
      $nonseps = nonseps \cup \{X_1\}$;
      $X_1 = (X_1^\star \setminus X_1) \cup X_1^{\star\star}$;
      $X_2 = X_2 \setminus X_1^\star$;
    end
  end
end
if $|X_1| = 1$ **then** $seps = seps \cup X_1$;
else $nonseps = nonseps \cup X_1$;
return *seps and nonseps*;

Algorithm 10: L_Inter($X_1, X_2, \underline{f}, func$)

if Interact($X_1, X_2, \underline{f}, func$) **then**
  if $|X_1| = 1$ **then**
    return X_1 ;
  else
    Divide X_1 into equally-sized groups G_1, G_2 ;
    X_1^1 = L_Inter($G_1, X_2, \underline{f}, func$);
    X_1^2 = L_Inter($G_2, X_2, \underline{f}, func$);
    $X_1 = X_1^1 \cup X_1^2$;
  end
end
return X_1 ;

Note that the `R_Inter` and `L_Inter` functions differ primarily in that the former focuses on the set X_2 while the latter acts on X_1. Moreover, the instruction in the base case of the recursion (line 3) also differs in the two functions. In the `R_Inter` function, variables in X_2 that interact with X_1 are added to X_1 while in the `L_Inter` function, only the variables in X_1 that interact with X_2 are returned.

4.2.2 Overlapped cooperative co-evolutionary framework

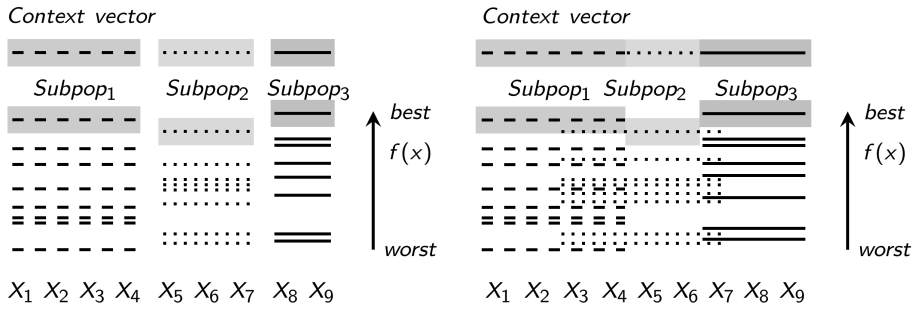
The new overlapped cooperative co-evolutionary algorithm (OCC-EA) introduced in this section follows the guidelines of the CC framework presented in Section 2.1, with some adjustments to deal with the overlapped variables. The first modification concerns the decomposition step that is performed with the ORDG.

The optimization itself remains unchanged in the sense that it still consists in optimizing each subcomponent with a standard EA in a round-robin strategy. However, in this step, the cooperation between subproblems needs to be revised. As a reminder (see page 19), in a standard CC framework, the individuals from a considered subcomponent are completed with the variables of the representative individuals of the other subcomponents in order to be evaluated. The latter are generally chosen as current best individuals in each subcomponent. In the CC literature, the term *context vector* is often used to name the n-dimensional vector obtained by concatenating all the representative individuals (van den Bergh and Engelbrecht (2004)). The function evaluations are therefore performed through the completion with the variables of the context vector. Considering disjoint subcomponents, there is only one way to build this context vector but, if we now consider overlapped subcomponents, this arrangement is no longer unique. This introduces the issue of which value of a variable x_i has to be shared in the context vector when the variable in question is involved in several subcomponents. In this context, the new OCC-EA relies on the following principle: for each variable x_i, its value in the context vector is the one of the best individual among the two subpopulations focusing on x_i (or in the only subpopulation if x_i is not overlapped). In particular, if none of the variables are overlapped, the standard construction of the context vector is retrieved. Both constructions, in a standard and in an overlapped framework, are presented in Figure 4.3.

It is important to note that the choice of the best individual within two different subpopulations does not require any extra function evaluations. Indeed, the individuals are compared depending on their function values computed during the optimization of the corresponding subcomponents in the round-robin fashion loop. Furthermore, the context vector is updated each time a better solution is reached. It guarantees frequent exchanges of information between the subcomponents.

4.2.3 Experimental settings and results

This section presents the simulation experiments used to evaluate the performance of the new OCC-EA. Firstly, analysis will be conducted on the ORDG decomposition itself. Secondly, the effects of the new decomposition and the related OCC framework on the optimization results will be assessed. The reference algorithm used for



(a) Standard CC (b) Overlapped CC

Figure 4.3 – Management of the context vector within a standard and an overlapped CC framework. The illustrative example relies on the interaction structure presented in Figure 4.2. Dashed, dotted and solid lines represent individuals from subpopulations 1, 2 and 3, respectively. In each subpopulation, they are ranked according to their fitness values, the best ones lying on the top. The context vector is built with the variables values of the best individual in each subpopulation (Blanchard et al. (2021)).

comparison is a standard CC-EA based on the RDG3 decomposition. The benchmark set is composed of 6 overlapping benchmark problems derived from the CEC'2013 competition on LSGO (Xi et al. (2013)). Two of them are picked directly from the CEC'2013 suite: f_5 is the 905-d shifted Schwefel's function with conflicting overlapping subcomponents, f_6 is the 1000-d shifted Rosenbrock function. The four others, f_1 to f_4, are built by replacing the Schwefel basis function in f_5 by Ackley, Elliptic, Rastrigin and Rosenbrock functions, respectively. Further details on the benchmark set are provided in Appendix B.

The decomposition results of the ORDG and the RDG3 on the 6 benchmark function are presented in Table 4.1. Two different threshold values $\varepsilon_n = 50$ and $\varepsilon_n = 0$ are used for the RDG3: the first one is the value used to study optimization results in the paper of Sun et al. (2019), the second one aims to systematically cut the overlapping at shared variables and therefore produce as many components as possible. For each de-

	RDG3 ($\varepsilon_n = 50$)			RDG3 ($\varepsilon_n = 0$)			ORDG		
	k	r	FEs	k	r	FEs	k	r	FEs
f_1	12	905	16273	20	905	16597	12	1011	16702
f_2	12	905	16252	19	905	16666	17	1000	18214
f_3	12	905	16249	20	905	16615	17	1000	18214
f_4	12	905	16252	20	905	16666	17	1000	18214
f_5	13	905	16288	21	905	16669	17	1003	18202
f_6	20	1000	49891	500	1000	25435	999	1998	59848

Table 4.1 – Decomposition results of RDG3 (with $\varepsilon_n = 50$ and $\varepsilon_n = 0$) and ORDG strategies. k is the number of components generated, r is the sum of the number of variables in each group and FEs is the number of function evaluations computed.

composition strategy, the number of components generated (k), the sum of the number of variables in each group (r) as well as the number of required FEs are reported. Since it does not perform any overlap, the r values obtained with the RDG3 are equal to the number of variables of the benchmark functions. On the contrary, the ORDG involves a larger number of variables. Theoretically, for functions f_1 to f_5, it should identify the components corresponding to each term of the sum in the benchmark construction. It should therefore involve 1 000 variables. This is indeed the case for f_2 to f_4 but some additional variables are considered for f_1 and f_5. This is caused by some false interaction detection between independent variables arising from computational roundoff errors. Moreover, still with regard to the benchmark construction, f_1 to f_5 are composed of 20 components. The RDG3 with $\varepsilon_n = 50$ only forms 12 (or 13) components since those containing less than 50 variables are merged with other ones. The RDG3 with $\varepsilon_n = 0$ nearly captures the 20 components, the small variations observed for f_1 and f_5 are again due to computational roundoff errors. Regarding the new ORDG, it captures 17 of the 20 components for f_2 to f_5[(4)]. They match the ones produced in the benchmark construction except that some of them have been merged. Indeed, if the ORDG starts the interaction identification with a variable of a component that shares some overlapped variables with two other components, the latter are merged into one single component. The same situation is recurring with the two components linked to that single component. Therefore, some components are composed of two subsets of variables that do not directly interact. That will not affect the optimization efficiency. Furthermore, even if the 20 components are not exactly identified, the ORDG objective is achieved since the LSGO overlapping problems are split into several subproblems sharing some variables with other subproblems. Finally, results obtained for function f_6 (Rosenbrock) fit with the expected ones. Indeed, for this function, each variable x_i directly interacts with x_{i+1} ($i = 1, \ldots, 999$). The RDG3 with $\varepsilon_n = 50$ ($\varepsilon_n = 0$) cuts the overlapping at some points to produce 20 (500) components of 50 (2 resp.) variables. The ORDG correctly identifies the overlapping structure by grouping each variable x_i with the following one and therefore produces 999 components of 2 variables.

The impact of the new ORDG decomposition on the optimization efficiency is assessed by comparing the OCC-EA presented above with a standard CC framework relying on the RDG3 decomposition, called RDG3-CC-EA. Both algorithms use the genetic algorithm of Minamo, described in Section 1.6, as subcomponents optimizer. Within each genetic algorithm, the population size is chosen as 10 times the number of variables of the considered component. The round-robin fashion optimization loop of the CC framework is repeated until the maximum number of FEs, fixed to 3×10^6 for the decomposition and the optimization, is reached. The statistical features of the solutions obtained with both algorithms are reported in Table 4.2. Considering the median values, the OCC-EA provides better solutions quality for 4 of the 6 functions. The RDG3-CC-EA with $\varepsilon_n = 0$ produces the best results for the remaining two. The convergence behavior along the optimization process is illustrated in Figure 4.4. Note that these graphs simply depict the evolution of $f(x)$ (instead of $f(x) - f(x^\star)$ as in

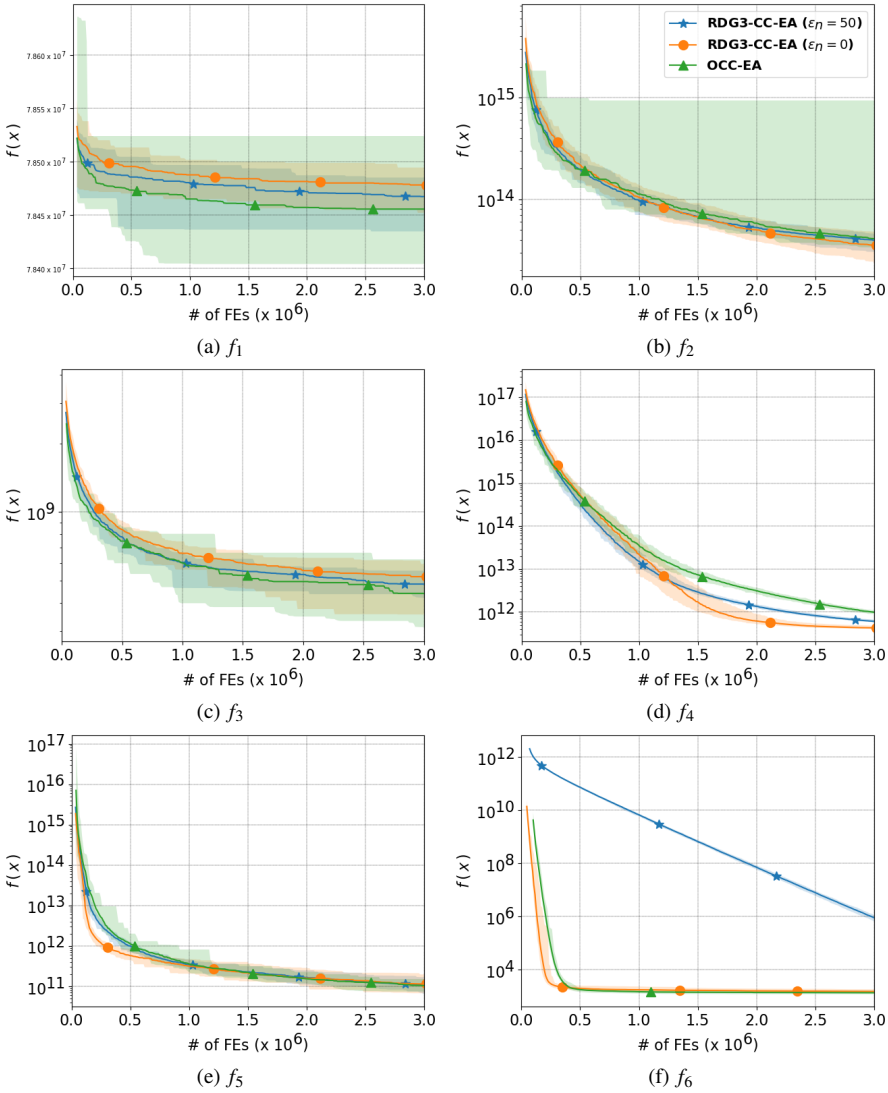
[(4)] Once again, it should theoretically also be the case for f_1 but the results for this particular function are affected by roundoff errors.

the previous chapter) because the optimum values of functions f_1 to f_5 are unknown. For f_6, the optimum value $f(x^\star)$ is equal to zero. The three algorithms follow a similar trend for f_1 to f_5[(5)]. For these functions, the two variants of the RDG3-CC-EA, only differing by the slightly different number of subcomponents generated, present close results. However, for f_6, the variation between the number of subcomponents is stronger. In this case, the results show that the RDG3-CC-EA with $\varepsilon_n = 0$ that produces a very large number of small components is more appropriate to perform

[(5)]Note that for f_2, the large-green colored area for the OCC-EA is simply caused by some runs stuck in pseudo-optima. Except that, the three algorithms also follow the same trend.

f_1					
Minamo	Best	Worst	Mean	Median	Std
RDG3-CC-EA ($\varepsilon_n = 50$)	6.97e+07	7.05e+07	7.02e+07	7.03e+07(+)	1.77e+05
RDG3-CC-EA ($\varepsilon_n = 0$)	7.00e+07	7.07e+07	7.04e+07	7.04e+07(+)	1.48e+05
OCC-EA	6.92e+07	7.12e+07	7.00e+07	**7.01e+07**	2.84e+05
f_2					
Minamo	Best	Worst	Mean	Median	Std
RDG3-CC-EA ($\varepsilon_n = 50$)	3.05e+13	4.61e+13	3.94e+13	3.95e+13(=)	3.56e+12
RDG3-CC-EA ($\varepsilon_n = 0$)	2.32e+13	4.81e+13	3.50e+13	**3.53e+13**(-)	5.24e+12
OCC-EA	2.85e+13	9.35e+14	8.68e+13	3.85e+13	1.67e+14
f_3					
Minamo	Best	Worst	Mean	Median	Std
RDG3-CC-EA ($\varepsilon_n = 50$)	4.22e+08	5.54e+08	4.86e+08	4.83e+08(+)	2.81e+07
RDG3-CC-EA ($\varepsilon_n = 0$)	3.56e+08	5.93e+08	5.15e+08	5.22e+08(+)	4.45e+07
OCC-EA	3.08e+08	6.22e+08	4.42e+08	**4.17e+08**	8.11e+07
f_4					
Minamo	Best	Worst	Mean	Median	Std
RDG3-CC-EA ($\varepsilon_n = 50$)	5.79e+11	6.57e+11	6.05e+11	6.01e+11(-)	1.82e+10
RDG3-CC-EA ($\varepsilon_n = 0$)	4.02e+11	4.81e+11	4.30e+11	**4.27e+11**(-)	1.40e+10
OCC-EA	7.08e+11	8.48e+11	7.80e+11	7.80e+11	3.58e+10
f_5					
Minamo	Best	Worst	Mean	Median	Std
RDG3-CC-EA ($\varepsilon_n = 50$)	6.40e+10	1.69e+11	1.10e+11	1.04e+11(+)	2.15e+10
RDG3-CC-EA ($\varepsilon_n = 0$)	6.71e+10	1.98e+11	1.13e+11	1.15e+11(+)	2.66e+10
OCC-EA	6.73e+10	1.52e+11	9.76e+10	**9.64e+10**	1.75e+10
f_6					
Minamo	Best	Worst	Mean	Median	Std
RDG3-CC-EA ($\varepsilon_n = 50$)	6.38e+05	1.11e+06	8.69e+05	8.68e+05(+)	9.80e+04
RDG3-CC-EA ($\varepsilon_n = 0$)	1.31e+03	1.82e+03	1.51e+03	1.50e+03(+)	1.21e+02
OCC-EA	1.19e+03	1.64e+03	1.37e+03	**1.34e+03**	1.01e+02

Table 4.2 – Statistical objective value on 51 runs for overlapping benchmark problems derived from the CEC'2013 special session on LSGO. RDG3-CC-EA and OCC-EA, respectively, stand for the standard CC framework with the RDG3 decomposition and the new overlapped CC framework. Best median values are marked in bold face. (+), (-) and (=) represent the fact that OCC-EA is significantly better than, worse than, or equivalent to the compared algorithm according to the Wilcoxon rank-sum test with 5% significance level.



(a) f_1 (b) f_2

(c) f_3 (d) f_4

(e) f_5 (f) f_6

Figure 4.4 – Convergence graphs for overlapping benchmark problems derived from the CEC'2013 special session on LSGO. RDG3-CC-EA with $\varepsilon_n = 50$ (blue stars), RDG3-CC-EA with $\varepsilon_n = 0$ (orange circles), OCC-EA (green triangles). The solid line depicts the median value while the light-colored area around the solid line represents the interval between the best and the worst solutions over the 51 runs.

an efficient optimization. Moreover, the surprisingly similar results offered by the RDG3-CC-EA with $\varepsilon_n = 0$ and the OCC-EA can be explained as follows. In the OCC-EA, the overlapped variables shared among two subcomponents usually converge to the same value at a similar rate in the two subcomponents. Therefore, the overlapped decomposition performed in the OCC-EA does not significantly improve the cooperation between subcomponents in comparison with the standard exchange of representative individuals through the context vector in a standard CC framework. For this reason, although results in Table 4.2 are encouraging about the potential of the OCC-EA, it would be premature to claim that the new overlapped CC strategy is better than the standard CC framework.

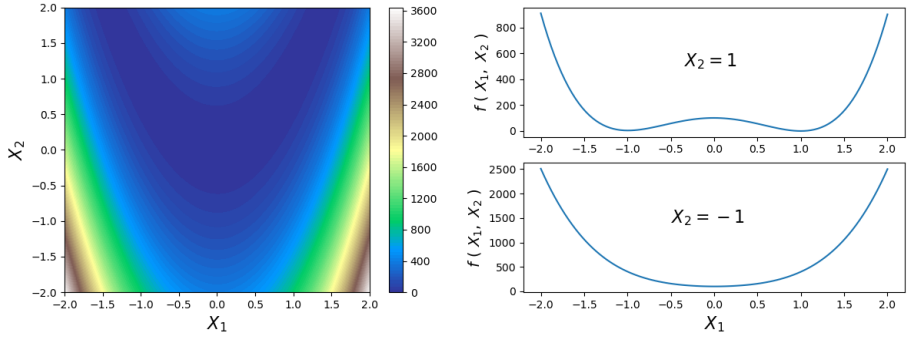
4.3 Extended analysis on the Rosenbrock function

The Rosenbrock function is a well-known benchmark problem often used in optimization. Its global optimum lies in a long, narrow and parabolic shaped valley in the 2-dimensional case, see Figure 4.5a. Its analytical expression is given by

$$f_{Rosenbrock}(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2],$$

where n is the number of decision variables, $x_i \in [-2,2]$, $\forall i = 1,\ldots,n$. Its global minimum lies at $x_i^\star = 1$, $\forall i = 1,\ldots,n$ and $f(x^\star) = 0$. Optimizing a large-scale instance of this function is very tricky, especially in a CC framework. Each variable x_i strongly interacts with x_{i+1} ($i = 1,\ldots,n-1$) and an efficient cooperation between these pairs of variables is fundamental to achieve precise optimization results. Indeed, if one focuses on the optimization of an arbitrary variable x_i, the following quantities have to be minimized: (a) $100(x_i - x_{i-1}^2)^2$, (b) $(x_i - 1)^2$, and (c) $100(x_{i+1} - x_i^2)^2$. If we assume that the variable x_{i-1} already reached its optimum value $x_{i-1}^\star = 1$, choosing $x_i^\star = 1$ would minimize quantities (a) and (b). Concerning the last term (c), it can be minimized by choosing $x_i^\star = \pm\sqrt{x_{i+1}}$ if x_{i+1} is positive or $x_i^\star = 0$ otherwise (see illustration in Figure 4.5b). Therefore, in the optimization process, variables whose index is close to 1 will converge to 1 from the beginning of the optimization. On the contrary, variables whose index is far away from 1 such that the variables x_1 does not influence their convergence behavior will start to converge to 0. Moreover, this trend is accentuated by the fact that the multiplicative factor in the term $100(x_{i+1} - x_i^2)^2$ gives it a larger influence in the objective function with respect to the term $(x_i - 1)^2$. The challenge to achieve efficient optimization results is to put forward the influence of the variable x_1 on other variables as quickly as possible to move them away from 0 to 1. In that way, they will converge to their optimal values one after the other.

Let us look at the convergence behavior of the variables with both RDG3-CC-EA (with $\varepsilon_n = 0$) and OCC-EA. For that particular study case, a budget of 4×10^5 FEs is allocated to each algorithm in order to optimize the 10-dimensional Rosenbrock function. This budget is relatively large with respect to the problem dimensionality but allows an analysis of the convergence behavior until each variable converges to its optimum value. Moreover, the conclusions drawn for this small-scale problem will



(a) Rosenbrock contour plot

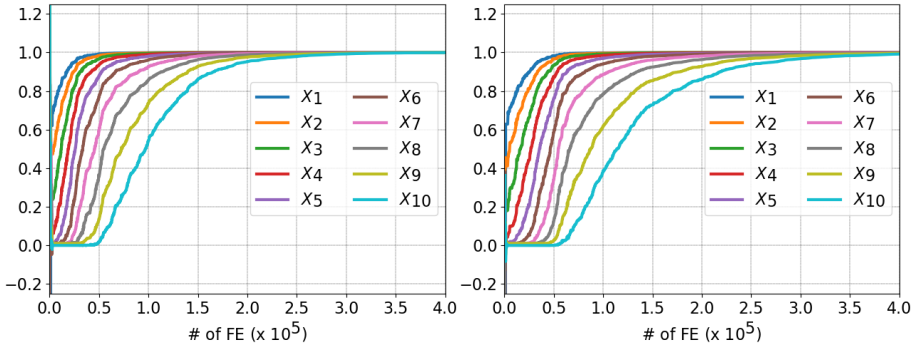
(b) Function landscapes

Figure 4.5 – Representation of the 2-d Rosenbrock function: contour plot (panel *a*) and different function landscapes with X_2 fixed to 1 or -1 (panel *b*). In the first case, the optimum value of $X_1 = \pm\sqrt{X_2} = \pm 1$. In the second case, the only optimum value is $X_1 = 0$.

be extended for LSGO problems. Figure 4.6 illustrates the convergence behavior of the variables when they are optimized with RDG3-CC-EA and OCC-EA, respectively. For both algorithms, the results are consistent with the expectations presented above. The variable x_1 immediately converges to 1, followed by x_2, etc. The variables far away from x_1 start by converging to 0 in order to decrease the objective value and move successively from 0 to 1 throughout the optimization process. Such a study case helps us to learn two important aspects of both RDG3-CC-EA and OCC-EA. Firstly, they offer a sufficiently good cooperation between subcomponents to be able to lead all the variables up to their optimal values, in spite of the strong interaction linking them two by two. Secondly, they show some limitations to precisely optimizing large-scale instances of the Rosenbrock function until the objective value tends to 0. Indeed, even in a CC context, the variables successively converge to their optimal values. That sequential process clashes with the divide-and-conquer nature of CC algorithms and prevents both RDG3-CC-EA and OCC-EA from finding the global optima in a reasonable time limit.

4.4 Conclusion

The new algorithm presented in this chapter is designed to optimize overlapping LSGO problems within a CC framework relying on an overlapped decomposition. This new strategy is structured around two pillars. The first one is the accurate overlapped decomposition obtained by adapting the RDG in order to detect and share overlapped variables among several components. The second one is the extension of the cooperation between subcomponents in order to efficiently manage overlapped variables during the optimization and properly share information through the context vector. Simulation experiments indicate that the new ORDG provides a proper over-



(a) RDG3-CC-EA (b) OCC-EA

Figure 4.6 – Evolution of the variables of the 10-d Rosenbrock function with respect to the number of function evaluations. Left side: RDG3-CC-EA. Right side: OCC-EA.

lapped decomposition. However, although the optimization results suggest that the management of the overlapped variables offered by the OCC-EA helps to get slightly better solutions, there are no certainties, at this stage, that the proposed OCC-EA significantly outperforms the standard CC framework embedded with the RDG3. Indeed, during the present study, we discovered that the exchange of information through the context vector in a standard CC framework is stronger than we could expect. It offers an efficient cooperation even if some linked variables are detached in different subcomponents. Moreover, the extended analysis also shows that, whatever the CC framework is overlapped or not, the exchange of information all along LSGO instances of Rosenbrock function may be time-consuming and limits the potential of the optimizer.

In any way, the work presented in this chapter is a first attempt to use an overlapped decomposition to deal with overlapping problems in a CC framework. The proposed approach shows promising prospects and we think that there is scope to further improve it, particularly by further exploiting the strength of the cooperation through the context vector.

Chapter 5

Surrogate-assisted optimization

Genetic algorithms, and in a more general way, evolutionary algorithms, are very popular to solve black-box optimization problems such as those presented in Section 1.1. They have been widely used to solve problems arising in engineering and sciences (Keane (1995); Wang et al. (2006); Chiesa et al. (2020)). However, in the last decades, increasingly complex numerical models have been developed to design any kind of engineering applications, especially in aerospace sciences (Forrester and Keane (2009)). These numerical simulations can become time-consuming and therefore the experiments needed for fitness evaluations are prohibitively expensive. In this context, it is inconceivable to solve such expensive optimization problems with genetic algorithms since they often require a high number of FEs. Surrogate-assisted optimization (SAO) has been introduced to address that issue and therefore reduce the number of FEs (Grefenstette and Fitzpatrick (1985); Jin (2011)). In such a framework, an efficient computational model of the fitness landscape, called *surrogate model*, is built and is used for most of the FEs instead of the expensive evaluations of the true function. A small number of real fitness evaluations are still required to build the surrogate model but also to update the latter during the optimization.

This chapter is organized as follows. Section 5.1 introduces the general scheme of the surrogate-assisted optimization framework. Then, several techniques to generate the data set of samples points, called the *design of experiments*, used to initialize the surrogate model are presented in Section 5.2. Thereafter, surrogate models based on radial basis function networks are presented in Section 5.3. Finally, some illustrations of the surrogate-assisted optimization efficiency are given in Section 5.4.

5.1 General scheme

The SAO framework considered in this thesis relies on EAs whose convergence rate is strongly accelerated through an efficient coupling with surrogate models. Its gen-

eral scheme is illustrated in Figure 5.1. The first step consists in building the initial database used to construct the initial surrogate model. For this purpose, a number of sampling points are randomly chosen across the design space and are evaluated with the true function. There are several techniques to choose the selected points in order to cover the design space. They are discussed in Section 5.2. Based on the information collected in the design of experiments, data-fitting modeling techniques are used to build the surrogate model. The latter is very cheap to evaluate compared with the expensive simulations. Therefore, a search process is carried on the strength of the surrogate model to design new sampling points that will be evaluated with the real function. There are several criteria, called *infill criteria*, to select those points (Forrester and Keane (2009)). The most common practice is to check the accuracy

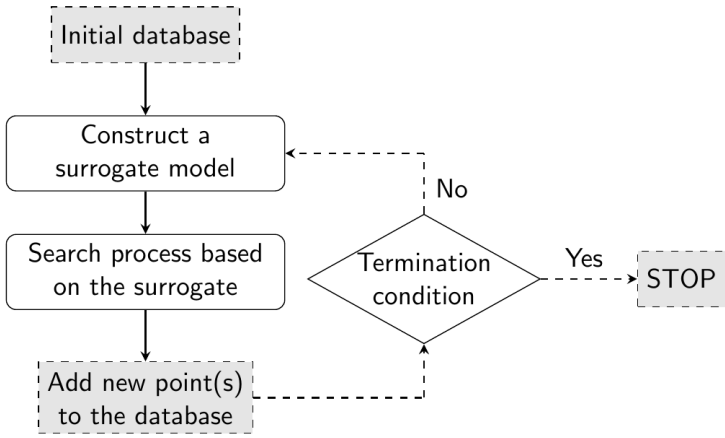


Figure 5.1 – Surrogate-assisted optimization framework.

of the surrogate model at its optimum (Queipo et al. (2005)). To this end, the surrogate model is optimized with an EA (a genetic algorithm in our framework) and the predicted optimum is evaluated with the expensive function. This kind of process encourages the exploitation of the surrogate models. However, an efficient optimization framework must rely on a right balance between exploitation and exploration of the design search space. Some exploration criteria select additional design points in areas where there are high uncertainties about the model accuracy. For example, when a Kriging model (Jones et al. (1998)) is used as surrogate, one can select points with high values of the mean squared error in order to improve the quality of the surrogate (Booker et al. (1999)). If error estimates are not available, another technique for space-filling is to select one point that minimizes the value of the surrogate model (if a minimization problem is considered) and that must also be far away from previous design points. This point strikes a right balance between exploitation

and exploration. It can be chosen by minimizing the following *merit* function (Torczon and Trosset (1998)):

$$\Phi(x) = \hat{f}(x) - \rho d(x), \tag{5.1.1}$$

where $\hat{f}(x)$ denotes the surrogate model approximation of the expensive function $f(x)$, $\rho > 0$ is a scale factor and $d(x)$ is the distance from x to the nearest design point. Finally, the selected point(s) is (are) evaluated with the high-fidelity simulations and is (are) added to the database. Proceeding along these lines, the surrogate model becomes even more accurate iteration by iteration. The whole process is repeated until a stopping criterion is met: in most cases the maximum number of design iterations.

In particular, the SAO framework considered in this thesis is the one implemented in Minamo (Sainvitu et al. (2010)), the multi-disciplinary optimization platform developed at the applied research center Cenaero. The latter provides very efficient methods for simulation-based designs and includes, amongst others, a powerful SAO framework following the guidelines of the one presented in this chapter.

5.2 Design of experiments

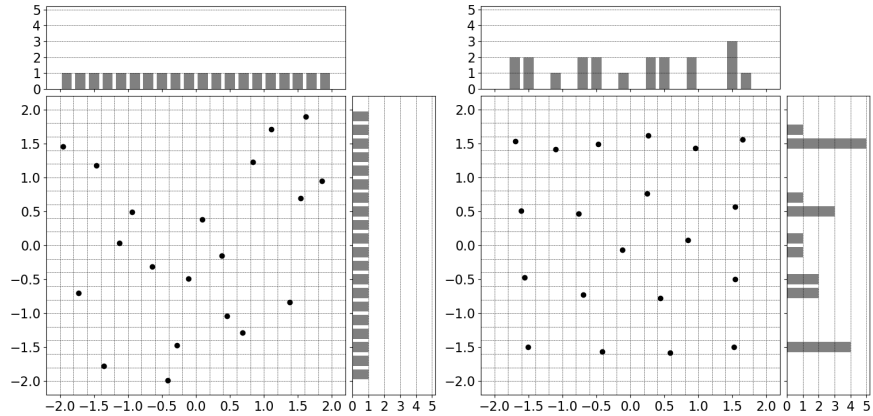
The design of experiments consists in sampling points in the search space. The surrogate model accuracy depends on the choice of these points. The more uniformly distributed they are, the more accurate the surrogate model will be. There exist two classes of sampling techniques: *a priori* techniques that sample points in the search space regardless the function knowledge and *a posteriori* ones (Jin et al. (2002)) that use the latter to better capture the function landscape. The aim of this section is to introduce the following *a priori* techniques: Latinized Hypercube Sampling (LHS), Centroidal Voronoi Tessellation sampling (CVT), and Latinized CVT sampling (LCVT) (Saka et al. (2007)).

The LHS sampling can be viewed as a n-dimensional extension of Latin squared sampling. In a two-dimensional space, the latter consists first in creating a grid of n_s^2 equally-sized squares, n_s being the desired number of sampling points. Then, the n_s points are distributed on the grid in such a way that each line and each column contains exactly one point. The LHS sampling is the n-dimensional extension, considering a n-dimensional hypercube instead of a two-dimensional square. The use of this technique allows one to obtain sampling points whose projections on coordinate axes are uniformly distributed, see Figure 5.2a.

The CVT sampling is based on a centroidal Voronoi tessellation that is a particular case of Voronoi tessellation. The latter is a partition of space into regions such that all the points in a region are closer to the referring object of the region itself than to any other objects in space. For a centroidal Voronoi tessellation, each referring object is a centroid, i.e., the mean position of all the points in all of the coordinate directions. The CVT sampling consists in partitioning the search space into n_s subspaces according to a centroidal Voronoi tessellation and in choosing the centroids as sampling points. This technique produces a better distribution of sample points over the n-dimensional hypercube than the LHS, see Figures 5.2a and 5.2b. The latter could

sometimes create samplings with clustered groups of points, the most extreme example (in the two-dimensional case) being a square with all the points on the diagonal. However, the CVT sampling has also its limits. Indeed, projecting the sample points on coordinate axes, one can observe that they are grouped in widely spaced clusters, see point projections in Figure 5.2b.

The LCVT sampling combines the advantages of both LHS and CVT samplings, i.e., the appropriate distribution of projections on coordinate axes of LHS and the efficient space-filling of CVT. It consists first in creating a CVT sampling. Then the latter is latinized, i.e., the points of the sampling are moved in a relatively close neighborhood in order to fulfill the LHS property. Such a sampling is illustrated in Figure 5.2c



(a) LHS

(b) CVT

(c) LCVT

Figure 5.2 – Sample sets of 20 points, in the 2-dimensional space, obtained with LHS, CVT and LCVT, respectively. The histograms represent the distributions of point projections on coordinate axes.

5.3 Radial basis function models

As presented earlier, the SAO framework relies on surrogate models that approximate the fitness landscape. Different kinds of surrogates such as Radial Basis Function (RBF), Kriging or Support Vector Regression models are frequently used in this context (Forrester et al. (2008)). This section focuses on RBF models. The latter, introduced in Broomhead and Lowe (1988), are interpolation functions relying on a weighted sum of simple functions, called *radial basis functions*. They can also be interpreted as a simple kind of neural network. Given a database composed of n sampling points $x^{(i)}$ $(i = 1,\ldots,n)$ and the values of the function f to be approximated at these points $y^{(i)} = f(x^{(i)})$ $(i = 1,\ldots,n)$, the RBF model of f, denoted by $\hat{f}$, is given by the following equation (Forrester et al. (2008)):

$$\hat{f}(x) = \omega^T \psi = \sum_{i=1}^{n_c} \omega_i \psi(||x - c^{(i)}||),$$

where n_c is the number of radial basis functions, $c^{(i)}$ denotes the center of the i-th basis function $(i = 1,\ldots,n_c)$, ω is the n_c-dimensional vector of weight factors and ψ is a n_c-dimensional vector containing the values of the basis functions. The latter are evaluated at the Euclidean distance between the basis center $c^{(i)}$ and the point x at which one wants to approximate the value of $f(x)$. Very simple radial basis functions (linear, cubic and thin plate spline) only depend on this distance r. There also exist more sophisticated basis functions (Gaussian, multiquadric and inverse multiquadric) depending on an additional parameter σ that can be tuned to get a better approximation. All these radial basis functions are described in Table 5.1.

Whatever the choice of the basis functions, the weight factors are always determined using the same method. According to the following interpolation condition,

$$\hat{f}(x^{(j)}) = \sum_{i=1}^{n_c} \omega_i \psi(||x^{(j)} - c^{(i)}||) = y^{(j)}, \qquad j = 1,\ldots,n$$

they are computed as a solution of a system of linear equations. In order to get a unique solution, the latter must be square, i.e., $n_c = n$. This condition is verified by choosing centers of the basis functions coinciding with the sampling points, i.e., $c^{(i)} = x^{(i)}$, $\forall\, i = 1,\ldots,n$. Therefore, the system to solve is given by

$$\Psi\omega = y$$

where Ψ, called the *Gram matrix*, is defined by $\Psi_{i,j} = \psi(||x^{(i)} - x^{(j)}||)$. Figure 5.3 presents an example in one dimension of a RBF model built on the basis of three sampling points. In this thesis, the RBF models used in the SAO framework are the auto-adaptive ones implemented in the Minamo software. The latter can autonomously choose between Gaussian and multiquadric basis functions without the user having to prescribe it. It can also automatically adjust the width parameter σ of each basis function. This powerful tuning process is performed on the basis of the efficient Leave-One-Out procedure proposed by Rippa (1999).

Radial basis function	Expression
Linear	$\psi(r) = r$
Cubic	$\psi(r) = r^3$
Thin plate spline	$\psi(r) = r^2 \ln r$
Gaussian	$\psi(r, \sigma) = e^{-r^2/2\sigma^2}$
Multiquadric	$\psi(r, \sigma) = \sqrt{r^2 + \sigma^2}$
Inverse multiquadric	$\psi(r, \sigma) = 1/\sqrt{r^2 + \sigma^2}$

Table 5.1 – Different types of radial basis functions.

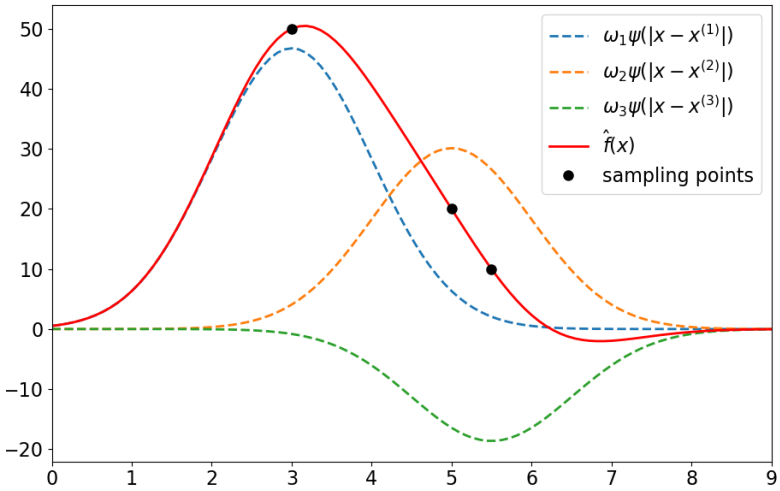


Figure 5.3 – A RBF model example in one dimension built on the basis of three sampling points. $x^{(1)} = 3$, $x^{(2)} = 5$ and $x^{(3)} = 5.5$. The radial basis functions are Gaussian functions with fixed width $\sigma = 1$. The computed weights are $\omega_1 = 46.7$, $\omega_2 = 30.1$ and $\omega_3 = -18.6$. The contribution of each basis function is drawn with dotted lines while the solid line represents the RBF model.

5.4 Illustrations

This section illustrates the efficiency of the SAO framework, especially when a limited budget in terms of FEs is considered because of the time-consuming nature of the evaluation of the studied optimization problem. Two well-known benchmark problems, Ackley and Rosenbrock, are used to compare the performance of a genetic algorithm embedded in the SAO framework with the performance of a genetic algorithm alone. Ackley and Rosenbrock are both minimization problems. Their analytical expressions are given as follows:

$$f_{Ackley}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + \exp(1),$$

$$f_{Rosenbrock}(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2],$$

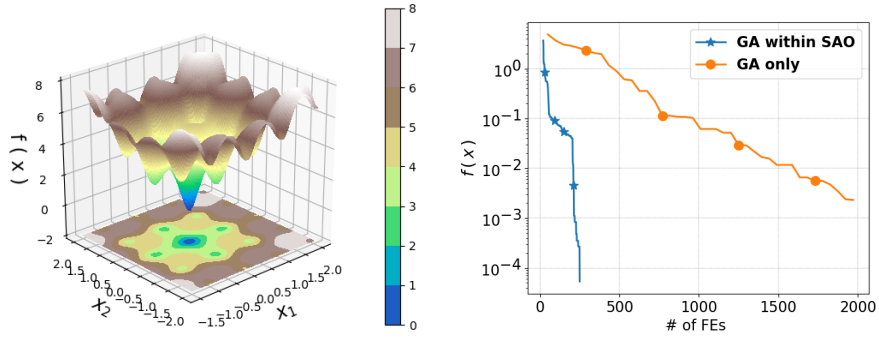
where n is the number of decision variables, $x_i \in [-2,2],\ \forall i = 1,\ldots,n$. Ackley is often chosen as a highly multimodal function. Its representation in the 2-dimensional case is provided in Figure 5.4a. Its global minimum lies at $x_i^\star = 0,\ \forall i = 1,\ldots,n$ and $f(x^\star) = 0$. Rosenbrock is a function whose optimum lies in a long, narrow and parabolic shaped valley in the 2-dimensional case, see Figure 5.4c. Its global minimum lies at $x_i^\star = 1,\ \forall i = 1,\ldots,n$ and $f(x^\star) = 0$. In particular, the 5-dimensional Ackley and Rosenbrock functions are used for the results presented in this section.

The algorithms used for the comparison are the genetic algorithm and the SAO algorithm of the Minamo software. The genetic algorithm evolves a population of 50 individuals for 40 generations[(1)] and therefore requires 2 000 FEs. The SAO algorithm follows the general scheme presented in Section 5.1. In particular, the initial database contains 20 sampling points obtained with the LCVT procedure; the surrogate model is the auto-adaptive RBF model; the search process is an optimization relying on the genetic algorithm (with a population of 50 individuals for 40 generations, such as for the genetic algorithm only); the infill criteria is a mono-point criteria based on the merit function described in Equation (5.1.1). The design loop is repeated 230 times. The total number of exact FEs is therefore 250.

The comparison between both algorithms is presented with convergence graphs in Figures 5.4b and 5.4d. They show that the genetic algorithm embedded in the SAO framework clearly outperforms the genetic algorithm alone. Indeed, the former consumes 8 times less FEs and better approaches the optimal value $f(x^\star) = 0$. Besides, in the context of time-consuming functions, the additional cost to build and optimize the surrogate models in the SAO framework is negligible compared with the cost required to evaluate the true function. Based on this assumption, one can therefore compare the two algorithms using the number of FEs instead of the CPU time.

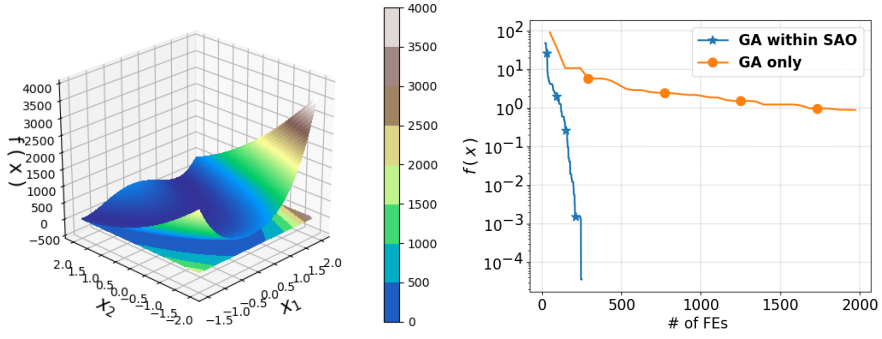
Additional contour plots illustrating the behavior of the SAO algorithm on the 5-dimensional Rosenbrock function are provided in Figure 5.5. For the three plots,

[(1)] As a reminder, the description of the genetic algorithm of Minamo was presented in Section 1.6.



(a) Ackley surface and coutour plot

(b) Ackley convergence graph

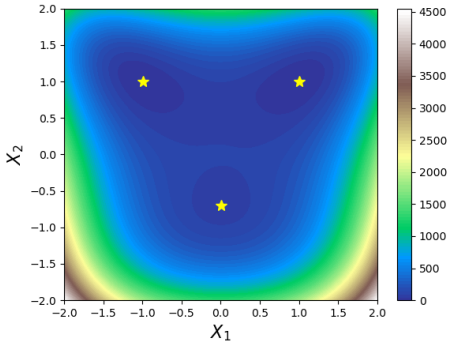


(c) Rosenbrock surface and coutour plot

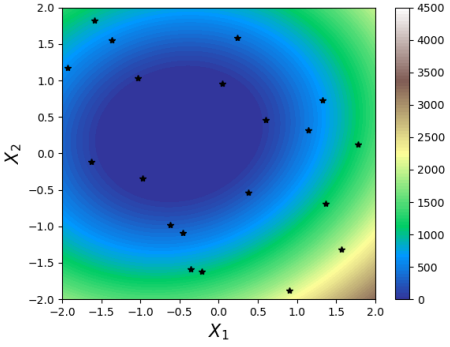
(d) Rosenbrock convergence graph

Figure 5.4 – Representations of 2-d Ackley and 2-d Rosenbrock functions and convergence graphs for the 5-d cases comparing the GA within SAO (blue stars) and the GA only (orange circles).

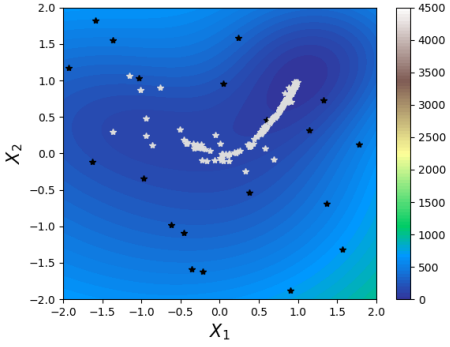
the decision variables x_3, x_4 and x_5 are fixed to 1 (i.e., to their optimal values) to allow a visualization in the 2-dimensional plane. The first plot (Figure 5.5a) represents the landscape of the 5-dimensional Rosenbrock function. On that slice of the function landscape, three local optima can be identified, the two on the top lying in a valley such as in the case of the 2-dimensional function. Among these two optima, the global one lies at $(1, 1)$. The second plot (Figure 5.5b) represents the landscape of the auto-adaptive RBF model that approximates the function at the beginning of the SAO procedure, i.e., the auto-adaptive RBF model built with the 20 points of the initial database (these points are drawn with black stars on the graph). At that stage, it provides a very rough approximation of the true function. Then, the knowledge of the function landscape becomes even more accurate iteration after iteration, especially in the area where are located the predicted optima of the successive surrogate models. This is illustrated on the third contour plot (Figure 5.5c) that represents the landscape of the auto-adaptive RBF model at the end of the SAO procedure. One can observe



(a) Contour plot of the true function. Local optima are drawn with yellow stars.



(b) Contour plot of the auto-adaptive RBF at the start of the SAO algorithm



(c) Contour plot of the auto-adaptive RBF at the end of the SAO algorithm

Figure 5.5 – Contour plots illustrating the behavior of the SAO algorithm on the 5-dimensional Rosenbrock function, the decision variables x_3, x_4 and x_5 are fixed to 1 (i.e., to their optimal values) to allow representations in the two-dimensional plane. Points of the initial database are drawn with black stars, selected points at each iteration are drawn with white stars.

that the selected points at each iteration drawn with white stars) have progressively led the algorithm to the global optimum at $(1, 1)$. Moreover, the approximation of the function around this optimum is very accurate. On the contrary, it can also be seen that other areas of the search space that benefited from fewer attention and were thus less explored, because they have not been predicted as promising by the successive surrogate models, are not necessarily perfectly modeled at the end of the SAO algorithm. In this way, we can conclude that the latter provides an approximation of the function landscape which is refined as the optimization progresses in order to converge to the global optimum.

Chapter 6

Surrogate-assisted cooperative co-evolutionary algorithms

In recent years, many research efforts have been focused to solve large-scale optimization problems by means of evolutionary algorithms (Jian et al. (2020)). Cooperative co-evolutionary algorithms, introduced in Chapter 2, have been developed to solve such problems that depend on thousands of variables. This approach has been shown to be very effective but it still requires a relatively large number of FEs in order to get sufficiently precise results. Therefore, it is inappropriate to deal with computationally expensive optimization problems. The surrogate-assisted optimization framework, introduced in Chapter 5, efficiently handles this issue by exploiting surrogate models that approximate the fitness landscape. Although such a framework is well suited for small dimensional problems, it suffers from the curse of dimensionality when dealing with LSGO problems. In this case, not only the search process on the surrogate model becomes costly but the surrogate model building itself is also difficult (Stork et al. (2020)). Taking into account strengths and weaknesses of both SAO and CC frameworks, to combine those two approaches should be a promising way to optimize *High dimensional, Expensive and Black-box* problems (HEB, Shan and Wang (2010)).

Such a Surrogate-Assisted Cooperative Co-evolutionary (SACC) methodology has been first introduced in Ong et al. (2002). It shows promising results but it is limited to 20-dimensional nonseparable functions. Almost a decade later, Goh et al. (2011) proposed another SACC approach to solve expensive constrained optimization problems but this study is still restricted to small dimensional cases (up to 13 decision variables). In recent years, the research activity around this topic has increased. First approaches that really tackle high dimensional problems (up to 1000 decision variables) were introduced in De Falco et al. (2017) (work further detailed in De Falco et al. (2019)) and Blanchard et al. (2019a). They both rely on several random groupings of the variables. They are presented and compared in Section 6.1. A more recent study which relies on random feature selection is also proposed in Fu et al. (2020).

Although it is presented in a significantly different way, the proposed strategy is, over the long run, more or less equivalent to decompositions based on several random groupings of the variables. Moreover, once compared with the algorithm presented in De Falco et al. (2017), it only produces minor improvements with respect to the reference algorithm. Other approaches are based on the interaction structure of the variables in the objective function. Ren et al. (2019b) perform an ideal decomposition (i.e., the decision variables are manually grouped according to the prior knowledge of the function) while Blanchard et al. (2019b) uncover the interaction structure with the recursive differential grouping introduced in Section 2.3.2. This SACC algorithm is presented in Section 6.2. Finally, there also exist other algorithms in the literature that claim to solve HEB problems in a surrogate-assisted evolutionary framework. However, they usually do not divide the original problems into subproblems and are therefore limited to medium scale problems up to one hundred variables (Sun et al. (2017a); Tian et al. (2019); Cai et al. (2019, 2020)).

6.1 Random decomposition-based algorithm (SACC-EAM)

The algorithm presented in this section combines the SAO and CC frameworks and is embedded in the Minamo software. The acronym SACC-EAM is used to name this Surrogate-Assisted Cooperative Co-Evolutionary Algorithm of Minamo. It relies on the random decomposition strategy. The motivation behind this choice is twofold. On the one hand, when first SACC approaches to solve HEB problems emerged, the most successful decomposition strategy was the one based on differential grouping (DG2). The latter requires a relatively high number of FEs $\left(\frac{n(n+1)}{2}+1\right)$ to uncover the interaction structure and was therefore not suitable to tackle HEB problems[(1)]. In this context, the random strategy that does not uncover interaction structure but performs several random decompositions in order to increase the chance to group interacting variables together was preferred. On the other hand, in the case of fully nonseparable problems, the random strategy is still nowadays the best way to divide the large problems into subproblems. The proposed SACC-EAM structure is given as follows:

1. Start a new cycle by randomly splitting the n-dimensional decision vector into k disjoint subcomponents of size s. Generate a design of experiments for each subcomponent.

2. Optimize each subcomponent with a SAO algorithm for a fixed number of iterations in a round-robin strategy. At each iteration, combine appropriate proposed points, called *candidate individuals*, from each subcomponent to build a *global candidate* and evaluate it.

3. If the maximum number of cycles is not reached, go to Step 1.

[(1)]The much less expensive recursive approaches introduced in Section 2.3.2 (RDG and RDG2) only emerged some time later. The RDG2 strategy will be used in the interaction detection decomposition-based algorithm (SACC-EAM-II) presented in Section 6.2.

This framework is further detailed in the pseudo-code provided in Algorithm 11. The fittest global candidate encountered during the whole optimization process is the *global individual* (*globa_indi* in the pseudo-code). It is used to complete partial solutions to be evaluated with the expensive function in each subproblem (lines 13 and 29). It is cyclically updated at each cycle (line 47) and remains unchanged during the execution of a given cycle. Unlike the standard CC framework introduced in Section 2.1, the global individual, used for completion, must not be modified during the optimization procedure performed at each cycle. Such an update would definitely pull down the consistency of the databases, and therefore of the surrogate models, specific to each subcomponent. Furthermore, since a new random decomposition is computed at each cycle (line 8), new designs of experiments need to be generated for consistency sake (line 12). Although this may results in an extra cost in terms of evaluations of the expensive function, it remains reasonable. Indeed, the small size of the subcomponents allows one to use designs of experiments containing an acceptable number of sampling points.

The SAO procedure performed at each cycle consists in optimizing each subcomponents with a SAO algorithm such as the one presented in Figure 5.1. In particular, this procedure starts with the loop at lines 11 to 16. It creates new designs of experiments, evaluates them with the expensive function and identifies the best individuals in each database as the candidate individuals. Note that at this step, the global individual is included in each database. Therefore, it does not need to be re-evaluated. The recorded FEs at line 14 concern the other points from the databases evaluated while being completed with the variables of the global individual. Thereafter, the global candidate is built by concatenating the candidate individuals from each subcomponent. It is also evaluated with the exact function. Then, the loop from lines 24 to 46 constitutes the iterative design process. During this process, surrogate models are built and optimized with the genetic algorithm. They are updated by adding the proposed points, i.e., the points that verify the infill criteria, in the database after their evaluations with the exact function. Furthermore, these points become new candidate individuals if they have better function values than the previous ones. Finally, at the end of each iteration, a new global candidate is built and evaluated. The best one encountered during the whole execution of the algorithm provides the solution of the optimization problem.

Concerning the evaluations of the expensive function computed in the iterative design process (lines 24 to 46), they are performed at two different levels. Firstly, at each iteration, the points identified by the mono-point infill criteria of each SAO are evaluated through their completions with the decision variables of the global individual (line 29). Secondly, if a new global candidate is produced, i.e., if an improvement is observed in at least one subcomponent, it is also evaluated. Therefore, during the iterative design process, s or $s+1$ FEs are computed at each iteration: s for infill points of each subcomponent, plus an additional one if a new global candidate is evaluated.

Efficient management of the FEs is very important in the SACC-EAM since it is devoted to tackle HEB problems. It can be achieved by exploiting parallel computing. The FEs required to generate the designs of experiments at step (1) can be easily computed in parallel since they are independent. At this stage, one does not profit

Algorithm 11: SACC-EAM algorithm

```
1  FEs ← 0 ;
2  global_indi ← new_random_individual() ;
3  new_global_indi ← global_indi ;
4  evaluate(global_indi, f) ;
5  FEs ← FEs + 1 ;
6  f_min = f(global_indi) ;
7  for cyc = 1 to max_cycles do
8      all_groups ← random_split(s) ;
9      data_bases = [ ] ;
10     candidate_indis = [ ] ;
11     for i = 1 to size(all_groups) do
12         data_bases[i] ← new_doe(all_groups[i]) ;
13         evaluate(data_bases[i], f, global_indi) ;
14         FEs ← FEs + size(data_bases[i]) − 1 ;
15         candidate_indis[i] ← find_best(data_bases[i]) ;
16     end
17     global_candidate ← concatenate(candidate_indis) ;
18     evaluate(global_candidate, f) ;
19     FEs = FEs + 1 ;
20     if f(global_candidate) < f_min then
21         f_min = f(global_candidate) ;
22         new_global_indi ← global_candidate ;
23     end
24     for ite = 1 to max_iterations do
25         update_flag ← false ;
26         for i = 1 to size(all_groups) do
27             f̃ ← build_surrogate(data_bases[i]) ;
28             proposed_point ← optimize_with_GA(f̃) ;
29             evaluate(proposed_point, f, global_indi) ;
30             FEs ← FEs + 1 ;
31             add(proposed_point, data_bases[i]) ;
32             if f(proposed_point) < f(candidate_indis[i]) then
33                 candidate_indis[i] ← proposed_point ;
34                 update_flag ← true ;
35             end
36         end
37         if update_flag then
38             global_candidate ← concatenate(candidate_indis) ;
39             evaluate(global_candidate, f) ;
40             FEs = FEs + 1 ;
41         end
42         if f(global_candidate) < f_min then
43             f_min = f(global_candidate) ;
44             new_global_indi ← global_candidate ;
45         end
46     end
47     global_indi ← new_global_indi;
48 end
49 return f_min
```

from the CC approach since the FEs of the designs of experiments are always independent, even for a standard SAO framework. However, the benefit from the CC approach can be fully exploited for the FEs computed during the iterative design process at step (2). Indeed, at each iteration, the infill points from each subcomponent can be simultaneously evaluated with the expensive function. Furthermore, the global candidates can also be evaluated at the same time as some infill points. Indeed, for a fixed iteration ite, the function value of the global candidate does not need to be known to start the following design iteration for each subcomponent. Therefore, the global candidate built at the end of iteration ite can be evaluated simultaneously with the k infill points from iteration $ite+1$. In this context, the parallel computation can be extended until $k+1$ threads, k being the number of subcomponents. It represents a considerable gain in terms of execution time.

6.1.1 Comparison with SA-EAM

This section compares the performance of the proposed SACC-EAM based on the random decomposition strategy and the standard Surrogate-Assisted Evolutionary Algorithm of Minamo, named SA-EAM, presented in Chapter 5. The benchmark set used for the comparison, introduced for the CEC'2008 special session on LSGO (Tang et al. (2007)), contains seven scalable problems with a wide variety of properties. The scalability of the functions also allows one to assess the performance of the SACC-EAM in comparison with the SA-EAM on 100-dimensional problems. For this study, the global optimum $x^\star$ of functions f_1 to f_6 is shifted with the data vector provided in Tang et al. (2007) while the optimum value $f(x^\star)$ is not shifted. The latter is equal to 0 for functions f_1 to f_6 and is unknown for f_7.

Both algorithms are executed 50 times, as done in Blanchard et al. (2019a). In order to underline the importance of the parallel computations of FEs (as a reminder, we assume that these FEs are computationally expensive), the results are recorded assuming that they are computed using 1, 5 or 10 parallel threads. Instead of a comparison based on the number of FEs (as done in the previous chapters), a comparison based on the number of parallel function evaluations (pFEs) is proposed. The counting of such pFEs is performed according to the number of threads that can be used to compute the FEs. For example, if 5 independent points have to be evaluated with the expensive function and if 5 threads are available, they can be computed simultaneously. In this case, only one pFE is recorded. For the standard SA-EAM, only the FEs computed for the design of experiments can be parallelized. On the contrary, for the proposed SACC-EAM, most of FEs can be computed in a parallel way as far as the number of threads is less or equal to $k+1$, where k is the number of subcomponents, as mentioned in the previous section. In particular, the budget in terms of pFEs is fixed to 500 for both algorithms. It meets the pressure imposed by the expense of the FEs considered in the studied context. Assuming that FEs are very costly and therefore consume most of the computational resources, such a comparison is equivalent to compare the algorithms with respect to the elapsed real time (also called wall-clock time).

For the SA-EAM, the design of experiments contains 200 points and the surrogate model is refined during at most 480 iterations depending on the number of threads

considered. The SA-EAM only runs 300 iterations if a single thread is considered (200 pFEs for the design of experiments, plus one per iteration = 500); it stops after 460 if 5 threads are used ($200/5 = 40$ pFEs for the design of experiments, plus 460 for the iterative process) and finally, it can run 480 iterations when 10 threads are available. At each iteration, the genetic algorithm used to optimize the surrogate model evolves a population of 1 000 individuals during 500 generations. For the SACC-EAM, the initial 100-dimensional decision vector is split into 25 subcomponents of size 4. The number of computed cycles depends on the number of threads considered. In each subcomponent, the SAO starts with 5 sampling points and performs 35 iterations. The internal genetic algorithm deploys a population of 40 individuals for 100 generations. In order to stay in the allocated budget in terms of pFEs, 5 cycles are performed when the algorithms runs on 10 threads, only two and a halve cycles (two complete cycles of 35 iterations plus a third cycle interrupted when all the allocated FEs are consumed) are computed when it runs on 5 threads and finally, a halve cycle is performed when a single thread is used (the budget is so much reduced that the algorithm is interrupted during the execution of the first cycle). All these settings are summarized in Table 6.1.

The median objective values of the final solutions for both algorithms, depending on the number of threads used for the computations, are provided in Table 6.2. Convergence graphs that illustrate the convergence behavior of the median solution along the optimization process are presented in Figure 6.1. Note that for functions f_1 to f_6, as in the case of the convergence graphs presented in previous chapters, the y-axis represents, in log-scale, the gap between the objective value of the current solution and the objective value of the optimal solution, i.e., $f(x) - f(x^\star)$. For function f_7, since $f(x^\star)$ is unknown and $f(x)$ takes negative values, the y-axis simply represents the evolution of $f(x)$. Moreover, unlike in previous chapters, the interval between best and worst solutions over the 50 runs is not depicted for ease of reading. The results show that exploiting the parallelization potential of the SACC-EAM allows it to outperform the standard SA-EAM. Indeed, when 10 threads are considered, median final solutions are better for the SACC-EAM for all the functions except f_2 which will be further discussed below. On the contrary, on a single thread, the SA-EAM performs better. Indeed, the 300 iterations are already sufficient to allow the SA-EAM to get significant improvements while the half cycle performed by the SACC-EAM can only provide very limited improvements in a such short period. Between these two extremes, the results on 5 threads represent a tipping point. In this case, the SACC-EAM provides better solutions for functions f_3, f_4, f_6 and f_7 but does not for remaining ones.

Furthermore, for the SACC-EAM, the graphs illustrate an oscillating behavior with varying convergence rate linked to the start of new cycles. For most functions, some "slow-down" occurs at the end and at the beginning of each cycle. Indeed, for a fixed cycle (and therefore for a fixed decomposition), after a few iterations needed to get sufficiently accurate surrogate models, the function value can be easily improved during the following stages of the SAO procedure. Then, as time goes by, it requires harder efforts to get further improvements. Finally, the SACC-EAM provides very poor results on f_2. For this particular function, modeling a surrogate in each subcomponent is hazardous. Since the analytical expression of f_2 is given by $f(x) = \max_i\{|x_i|, 1 \leq i \leq n\}$, when the design of experiments is evaluated for a fixed subcomponent, if the maxi-

Number of ...	SA-EAM	SACC-EAM
... sampling points	200	5
... design iterations	300/460/480	35
... cycles	–	0.5/2.5/5
... subcomponents	–	25
... generations in the genetic algorithm	500	100
... individuals in the populuation	1 000	40

Table 6.1 – Summary of SA-EAM and SACC-EAM settings depending on the number of threads, 1, 5 or 10, that can be used to compute the expensive function evaluations.

	1 thread		5 threads		10 threads	
	SA-EAM	SACC-EAM	SA-EAM	SACC-EAM	SA-EAM	SACC-EAM
f_1	**6.98e-02**	9.79e+02	**1.58e-02**	4.39e+01	1.47e-02	**1.84e-03**
f_2	**9.79e+01**	1.69e+02	**9.72e+01**	1.52e+02	**9.71e+01**	1.45e+02
f_3	**7.65e+07**	1.30e+09	1.52e+07	**3.43e+06**	1.40e+07	**1.25e+04**
f_4	**3.59e+02**	5.62e+02	2.91e+02	**2.80e+02**	2.69e+02	**2.17e+02**
f_5	**9.53e-01**	1.14e+01	**8.38e-01**	1.12e+00	8.38e-01	**2.87e-01**
f_6	**1.96e+01**	2.04e+01	1.92e+01	**1.63e+01**	1.92e+01	**6.81e+00**
f_7	**-9.46e+02**	-8.33e+02	-9.91e+02	**-1.11e+03**	-9.94e+02	**-1.28e+03**

Table 6.2 – Median objective values on 50 runs for benchmark problems from the CEC'2008 special session on LSGO. Results are presented depending on the number of threads used for the computations. For each number of threads, the best median values among SA-EAM and SACC-EAM are marked in bold face.

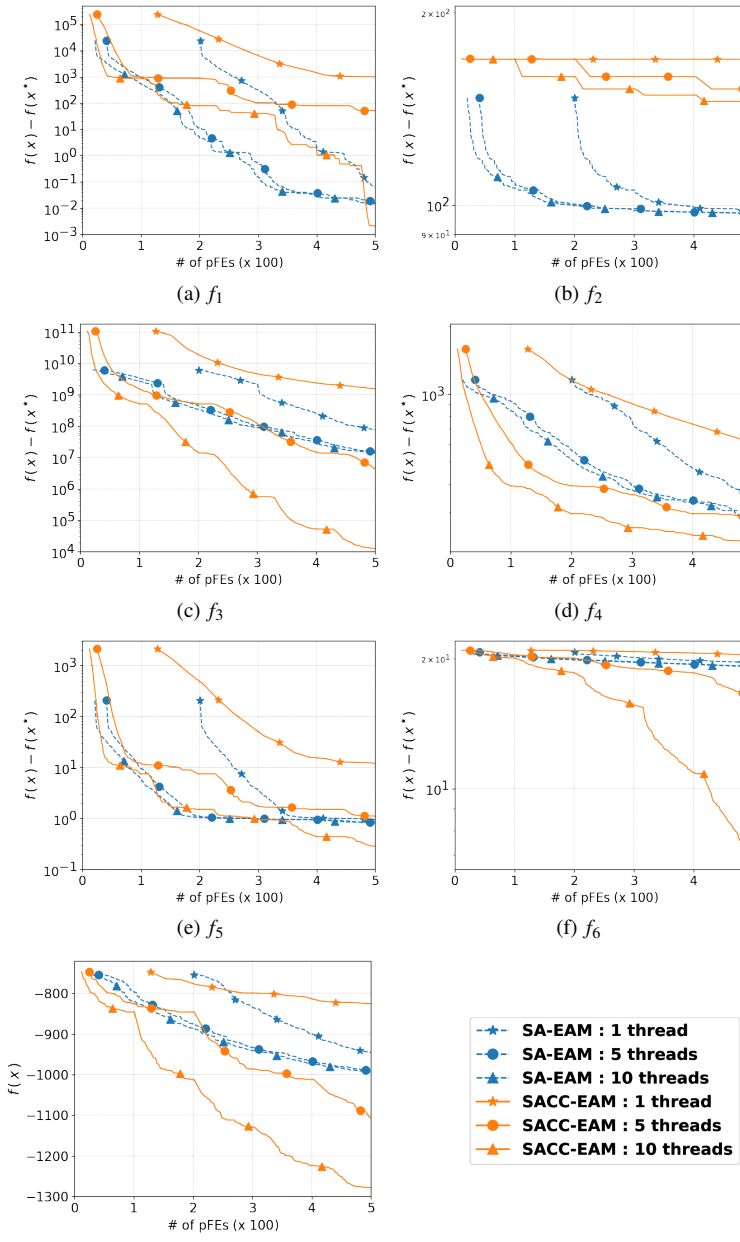


Figure 6.1 – Convergence graphs of the median solution over 50 runs for 100-dimensional problems from the CEC'2008 special session on LSGO. For f_1 to f_6, the evolution of $f(x) - f(x^\star)$ (in log-scale) is depicted while only $f(x)$ is depicted for f_7. Blue dashed lines stand for the SA-EAM while orange lines stand for the SACC-EAM. Results are presented according to the number of threads: 1 thread (stars), 5 threads (circles) and 10 threads (triangles).

mum $|x_i|$ comes from another subcomponent, the function values may be the samefor many points in the database. It leads to quite poor surrogate models unable to predict promising points. In particular, little improvements observed on the convergence graphs are not due to the optimization itself. They are achieved through the random points generated in the databases at the start of new cycles.

6.1.2 Comparison with SACC-JADE

This section compares the SACC-EAM with another algorithm from the literature called SACC-JADE (De Falco et al. (2017, 2019)). To the best of our knowledge, it is the only SACC approach, except the SACC-EAM presented in this thesis and in Blanchard et al. (2019a), that effectively tackles HEB problems within a random decomposition framework. In this algorithm, the optimizer chosen to optimize subcomponents is JADE, an adaptive version of differential evolution. As in SACC-EAM, the surrogate models built in each subcomponent are RBF models. However, they are used in a slightly different context. In each subcomponent, the surrogate is used to evaluate most of the offspring produced by JADE. Indeed, once the random decomposition is performed, each subcomponent is directly optimized by JADE in a CC context. During the first iterations, the evaluation of individuals is computed with the expensive exact function. Each of them are recorded in a database and once it contains enough information, a surrogate model is built. The latter is used instead of the expensive function to evaluate the offspring of the following iterations. During each of them, some exact FEs are still computed in order to ensure that the exact function value of the best individual in the offspring is known. The obtained information is added to the database and allows one to build more accurate models iteration by iteration.

The two SACC algorithms are compared on the 100-dimensional functions of the CEC'2008 special session on LSGO presented in the previous section and their 500 and 1 000-dimensional extensions. This benchmark set is the one used for the results presented in De Falco et al. (2017). Moreover, although the budget in terms of FEs is fixed to $500 \times n$ in De Falco et al. (2017), the budget in this study is restricted to $100 \times n$ FEs in order to be consistent with the HEB considered problems. This reduced budget may still seem excessive when dealing with very expensive simulations. For that reason, particular attention will be paid to the convergence rate in the results analysis, especially at the early stages of the optimization. Concerning the experimental settings, 50 runs are performed for both algorithms, as done in Blanchard et al. (2019a). Parameters for the SACC-EAM are quite similar that those chosen for the comparison with the SA-EAM while parameters for the SACC-JADE are fixed according to the original publication (De Falco et al. (2017)). They are summarized in Table 6.3.

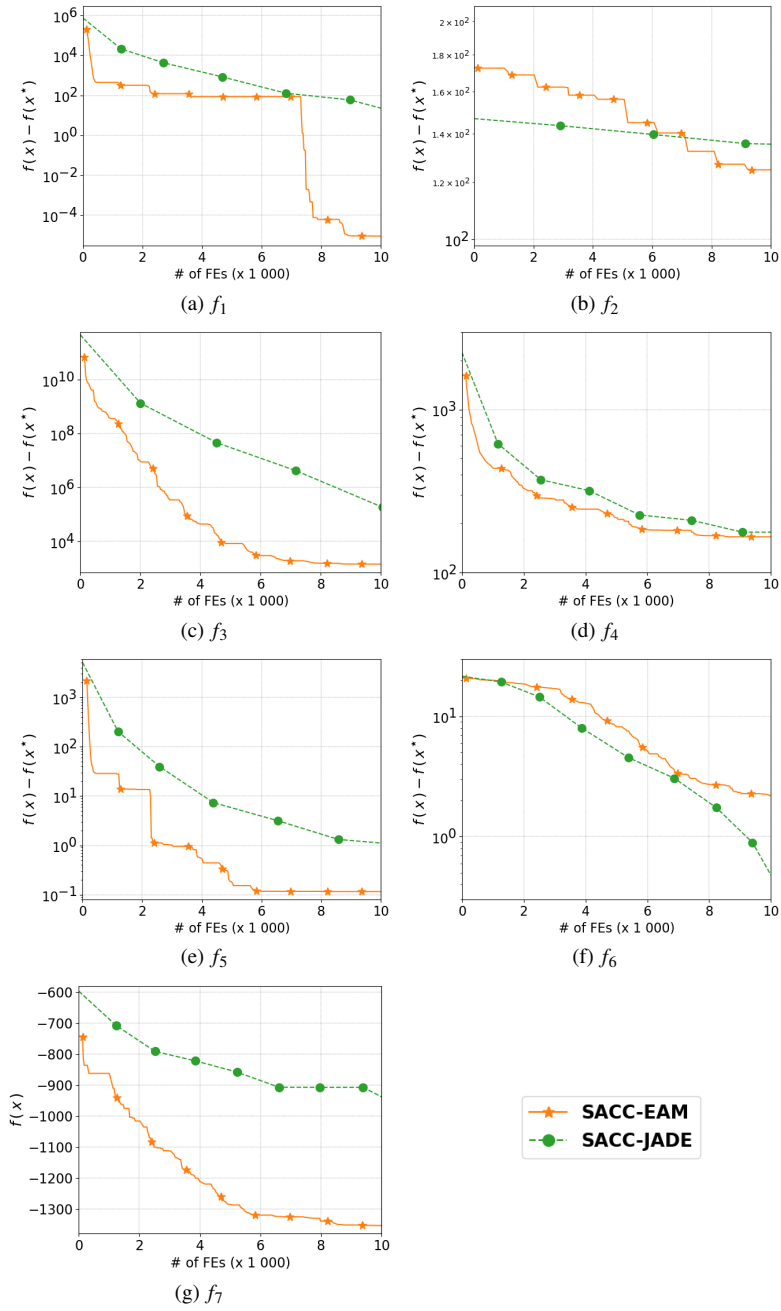
The median objective values of final solutions for both algorithms, depending on the dimension of benchmark functions, are provided in Table 6.4. Convergence graphs of the median objective values of both algorithms on 100-dimensional problems are presented in Figure 6.2. The results for 500 and 1 000-dimensional instances follow the same trend and are proposed in Figures 6.3 and 6.4. For functions f_1, f_3, f_5 and f_7, the SACC-EAM provides a significantly better convergence rate at the start of the

Number of ...	SACC-EAM	SACC-JADE
... sampling points	5	-
... design iterations	35, 36, 36	-
... cycles	5	until FEs available
... subcomponents	25, 125, 250	25, 125, 250
... generations in the GA / JADE	100	10
... individuals in the population	40	25

Table 6.3 – Summary of SACC-EAM and SACC-JADE settings for 100, 500 and 1 000-dimensional problems, respectively.

	100-d		500-d		1 000-d	
	SACC-EAM	SACC-JADE	SACC-EAM	SACC-JADE	SACC-EAM	SACC-JADE
f_1	**9.08e-06**	2.29e+01	**3.96e-05**	6.43e+01	**1.45e-04**	7.73e+01
f_2	**1.25e+02**	1.35e+02	1.65e+02	**1.48e+02**	1.75e+02	**1.47e+02**
f_3	**1.44e+03**	1.97e+05	**5.71e+03**	1.33e+06	**9.87e+03**	3.86e+06
f_4	**1.66e+02**	1.77e+02	**7.89e+02**	1.07e+03	**1.58e+03**	1.98e+03
f_5	**1.16e-01**	1.12e+00	**2.54e-02**	1.36e+00	**1.65e-01**	1.92e+00
f_6	2.18e+00	**4.80e-01**	2.09e+00	**8.25e-01**	2.20e+00	**1.09e+00**
f_7	**-1.35e+03**	-9.38e+02	**-6.44e+03**	-4.53e+03	**-1.27e+04**	-8.75e+03

Table 6.4 – Median objective values on 50 runs for benchmark problems from the CEC'2008 special session on LSGO. Results are presented depending on the dimension of benchmark functions. For each dimension, the best median values among SACC-EAM and SACC-JADE are marked in bold face.



(a) f_1 (b) f_2 (c) f_3 (d) f_4 (e) f_5 (f) f_6 (g) f_7

Figure 6.2 – Convergence graphs of the median solution over 50 runs for 100-dimensional problems from the CEC'2008 special session on LSGO. For f_1 to f_6, the evolution of $f(x) - f(x^\star)$ (in log-scale) is depicted while only $f(x)$ is depicted for f_7. The green dashed line stands for the SACC-JADE while the orange line stands for the SACC-EAM.

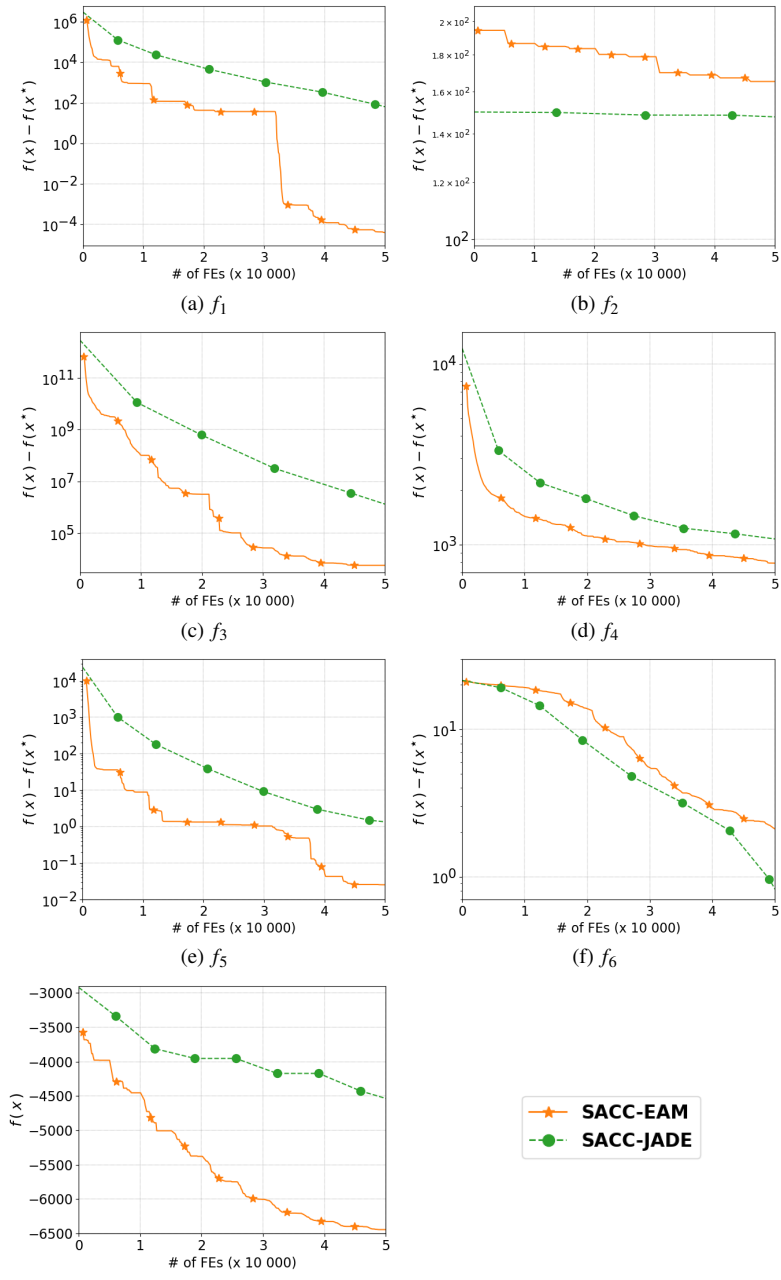
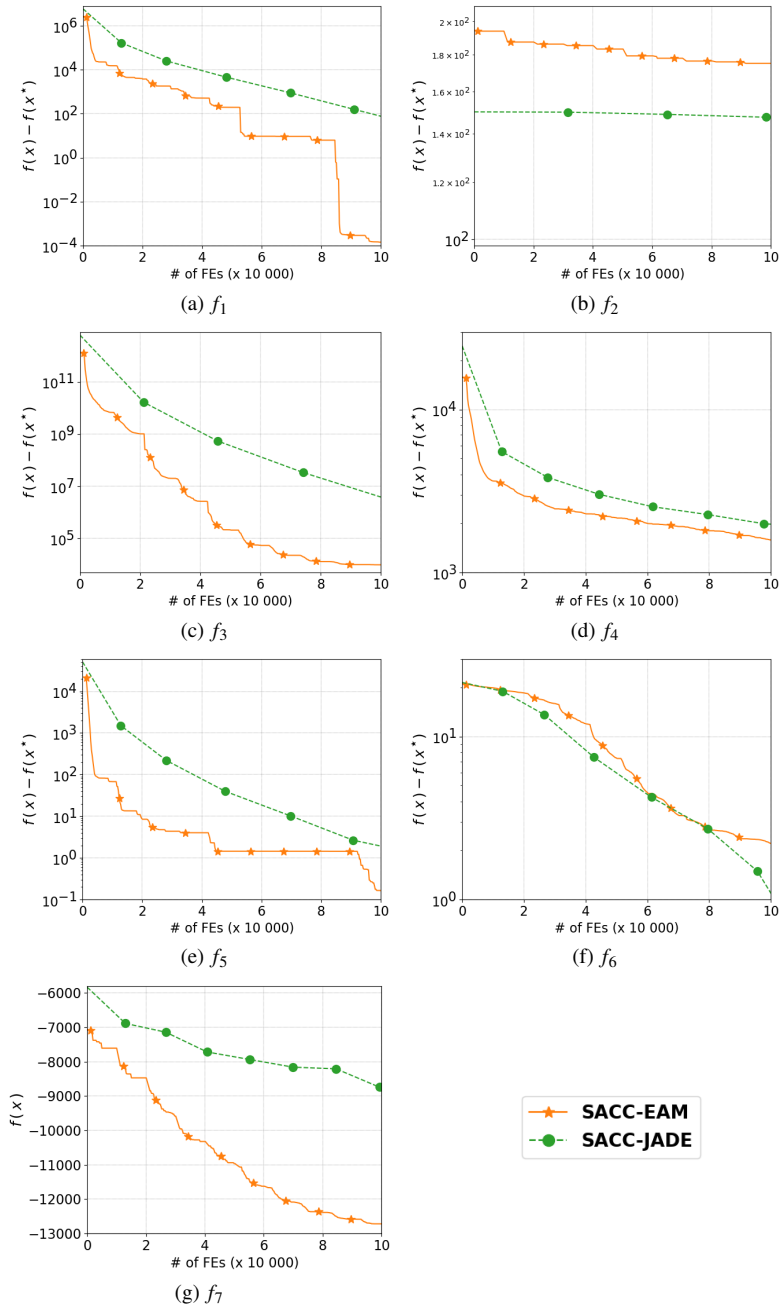


Figure 6.3 – Convergence graphs of the median solution over 50 runs for 500-dimensional problems from the CEC'2008 special session on LSGO. For f_1 to f_6, the evolution of $f(x) - f(x^\star)$ (in log-scale) is depicted while only $f(x)$ is depicted for f_7. The green dashed line stands for the SACC-JADE while the orange line stands for the SACC-EAM.



(a) f_1 (b) f_2 (c) f_3 (d) f_4 (e) f_5 (f) f_6 (g) f_7

Figure 6.4 – Convergence graphs of the median solution over 50 runs for 1 000-dimensional problems from the CEC'2008 special session on LSGO. For f_1 to f_6, the evolution of $f(x) - f(x^\star)$ (in log-scale) is depicted while only $f(x)$ is depicted for f_7. The green dashed line stands for the SACC-JADE while the orange line stands for the SACC-EAM.

optimization. Although slowing down at a later stage, it still provides better results than the SACC-JADE. For some other functions (f_4, f_6), the results are quite close. Finally, function f_2 being hard to model in a CC context, both algorithms show very poor capacities to improve the objective values.

6.2 Interaction detection decomposition-based algorithm (SACC-EAM-II)

The algorithm presented in this section takes benefit from the computationally effective identification of separable and nonseparable components provided by RDG2 (see Section 2.3.2). That is, in some way, an improved version of the SACC-EAM obtained by replacing the iterative random decomposition process by a single decomposition taking into account the interaction structure[(2)]. The acronym SACC-EAM-II is therefore used to name this updated SACC-EAM. This improvement allows one to perform only one cycle since the formed subcomponents are independent. It also allows one to save computational resources since new databases do not need to be created anymore at each new cycle. The proposed SACC-EAM-II is detailed in the pseudo-code provided in Algorithm 12 (Blanchard et al. (2019b)). Its main structure is given as follows:

1. Identify the separable and nonseparable groups of decision variables with RDG2. Arbitrarily split the group of separable variables into subgroups containing s variables; Generate a design of experiments for each subcomponent, i.e., for each nonseparable group and each separable subgroup.

2. Optimize each subcomponent with a SAO algorithm for a fixed number of iterations in a round-robin strategy. At each iteration, combine *candidate individuals* from each subcomponent to build a *global candidate* and evaluate it.

As mentioned before, the major improvement of the SACC-EAM-II in comparison with the SACC-EAM is the identification of separable and nonseparable components with RDG2. The lower number of FEs consumed with the recursive approach (in comparison with the original DG approach) allows one to use this differential grouping strategy even with a strictly limited budget in terms of FEs (which was not possible when the SACC-EAM was developed since the most advanced strategy at this time was DG2). Furthermore, despite the reduced cost obtained by RDG2, it still requires a significant budget of FEs. However, it will be shown, in the following results, that the SAO in each subcomponent performs so much better when the interactions are identified (in comparison with the random grouping strategy) that it is worth paying attention to detect them, even if the remaining budget for the optimization itself is therefore reduced.

[(2)]Note that the newer RDG3 strategy presented in Chapter 4, although interesting in CC-EAs, is not suitable, as things currently stand, in a SACC context. Indeed, in this context, if a single decomposition is performed, the formed subcomponents must be independent since the SACC framework does not offer as many exchanges of information between subcomponents as the standard CC-EA does. However, the aim of the RDG3 is precisely to allow one to form subcomponents which are not fully independent.

Algorithm 12: SACC-EAM-II algorithm

```
FEs ← 0 ;
sep_group, nonsep_groups ← RDG2(f, FEs) ;
sep_groups ← split(sep_group, s) ;
all_groups ← sep_groups ∪ nonsep_groups ;
global_indi ← new_random_individual() ;
evaluate(global_indi, f) ;
FEs ← FEs + 1 ;
f_min = f(global_indi) ;
data_bases = [ ] ;
candidate_indis = [ ] ;
for i = 1 to size(all_groups) do
    data_bases[i] ← new_doe(all_groups[i]) ;
    evaluate(data_bases[i], f, global_indi) ;
    FEs ← FEs + size(data_bases[i]) − 1 ;
    candidate_indis[i] ← find_best(data_bases[i]) ;
end
global_candidate ← concatenate(candidate_indis) ;
evaluate(global_candidate, f) ;
FEs = FEs + 1 ;
if f(global_candidate) < f_min then
    f_min = f(global_candidate) ;
end
for ite = 1 to max_iterations do
    update_flag ← false ;
    for i = 1 to size(all_groups) do
        f̃ ← build_surrogate(data_bases[i]) ;
        proposed_point ← optimize_with_GA(f̃) ;
        evaluate(proposed_point, f, global_indi) ;
        FEs ← FEs + 1 ;
        add(proposed_point, data_bases[i]) ;
        if f(proposed_point) < f(candidate_indis[i]) then
            candidate_indis[i] ← proposed_point ;
            update_flag ← true ;
        end
    end
    if update_flag then
        global_candidate ← concatenate(candidate_indis) ;
        evaluate(global_candidate, f) ;
        FEs = FEs + 1 ;
    end
    if f(global_candidate) < f_min then
        f_min = f(global_candidate) ;
    end
end
return f_min
```

The performance of the newly proposed SACC-EAM-II is compared with the SACC-EAM on benchmark functions introduced for the CEC'2010 special session and competition on LSGO (Tang et al. (2009)). This benchmark set is preferred to the CEC'2008 one used in the previous section because it provides 1 000-dimensional functions with different characteristics in terms of separability (the CEC'2008 set only contains fully separable or fully nonseparable functions). Indeed, most of them are additively separable functions composed of several nonseparable independent components (see Definition 3 page 24). These separable properties as well as the name of basic functions used to build the 1 000 benchmark problems are provided in Table 6.5.

For all these functions, except for F_3, F_6 and F_{11}, the decomposition obtained with RDG2 perfectly matches with the structure of components described in the table. For F_3, F_6 and F_{11}, RDG2 fails to identify the separable component since they are based on the Ackley function. The latter is separable but is not additively separable, which is the fundamental assumption to apply differential grouping strategies (see Theorem 2.3.1). For these functions, the separable component is thus identified as a nonseparable one and is therefore considered as whole in the SAO framework of the SACC-EAM-II. That will have a detrimental impact on the algorithm efficiency.

The SACC-EAM and SACC-EAM-II are compared with the same budget in terms of FEs. The parameter s, corresponding to the subcomponents size of SACC-EAM and to the separable subcomponents size of SACC-EAM-II is fixed to 50. The nonseparable subcomponents of SACC-EAM-II are kept as a whole. Therefore, their sizes depend on the identification performed by RDG2. The number of points in the designs of experiments in each subcomponent is set to the number of variables in that subcomponent $+1$. The number of iterations ite in the SAO loop is set to 33 for the SACC-EAM and to 500 for the SACC-EAM-II. Furthermore, the number of FEs consumed by RDG2, noted FE_{det} differs from one problem to another. The number of cycles cyc in the SACC-EAM is handled to approximately compute the same total number of FEs, noted FE_{tot} for both algorithms. This number is given by:

$$FE_{tot} = 1 + cyc \times [n + 1 + ite \times (k+1)] \text{ for SACC-EAM,}$$

$$FE_{tot} = FE_{det} + n + 2 + ite \times (k+1) \text{ for SACC-EAM-II,}$$

where k is the total number of subcomponents and n is the number of decision variables[(3)]. For functions for which the SACC-EAM-II creates only one subcomponent (i.e., for F_3, F_{19} and F_{20}), the term $n+2+ite\times(k+1)$ must be replaced by $n+1+ite$ in the presented equation. Indeed, the global candidate never needs to be evaluated since it coincides with the only candidate individual. All these parameters affecting the number of FEs are summarized in Table 6.6. Finally, for the internal optimization with the genetic algorithm, the population is evolved during 100 generations. Its size is fixed to 10 times the number of variables of the considered subcomponent.

Both algorithms have been executed 50 times, as done in Blanchard et al. (2019b). Statistical objective values across all functions are provided in Table 6.7

[(3)] In these formulae, we assume that, for both algorithms, a new global candidate is built and evaluated at each iteration. In practice, for problems with many subcomponents, it is almost always the case.

	separable component		nonseparable components	
	#variables	basic function	#groups (#variables)	basic function
F_1	1 000	Elliptic	0(0)	–
F_2	1 000	Rastrigin	0(0)	–
F_3	1 000	Ackley	0(0)	–
F_4	950	Elliptic	1(50)	Rotated Elliptic
F_5	950	Rastrigin	1(50)	Rotated Rastrigin
F_6	950	Ackley	1(50)	Rotated Ackley
F_7	950	Sphere	1(50)	Schwefel 1.2
F_8	950	Sphere	1(50)	Rosenbrock
F_9	500	Elliptic	10(50)	Rotated Elliptic
F_{10}	500	Rastrigin	10(50)	Rotated Rastrigin
F_{11}	500	Ackley	10(50)	Rotated Ackley
F_{12}	500	Sphere	10(50)	Schwefel 1.2
F_{13}	500	Sphere	10(50)	Rosenbrock
F_{14}	0	–	20(50)	Rotated Elliptic
F_{15}	0	–	20(50)	Rotated Rastrigin
F_{16}	0	–	20(50)	Rotated Ackley
F_{17}	0	–	20(50)	Schwefel 1.2
F_{18}	0	–	20(50)	Rosenbrock
F_{19}	0	–	1(1000)	Schwefel 1.2
F_{20}	0	–	1(1000)	Rosenbrock

Table 6.5 – Description of the CEC'2010 benchmark set. For each function the separable and nonseparable components are identified. Their size and the basic functions from which they are designed are presented.

	SACC-EAM		SACC-EAM-II	
	$s = 50$ $ite = 33$		$s = 50$ $ite = 500$	
	cyc	FE_{tot}	FE_{det}	FE_{tot}
F_1	9	15 247	2 998	14 500
F_2	9	15 247	2 998	14 500
F_3	4	6 777	5 992	7 493
F_4	9	15 247	4 198	15 700
F_5	9	15 247	4 144	15 646
F_6	7	11 859	8 905	11 407
F_7	9	15 247	4 222	15 724
F_8	10	16 941	5 599	17 101
F_9	15	25 411	14 026	25 528
F_{10}	15	25 411	14 008	25 510

	SACC-EAM		SACC-EAM-II	
	$s = 50$ $ite = 33$		$s = 50$ $ite = 500$	
	cyc	FE_{tot}	FE_{det}	FE_{tot}
F_{11}	12	20 329	13 684	20 686
F_{12}	15	25 411	14 308	25 810
F_{13}	24	40 657	29 233	40 735
F_{14}	19	32 187	20 554	32 056
F_{15}	19	32 187	20 512	32 014
F_{16}	19	32 187	20 908	32 410
F_{17}	19	32 187	20 758	32 260
F_{18}	36	60 985	49 852	61 354
F_{19}	31	52 515	50 992	52 493
F_{20}	31	52 515	50 866	52 367

Table 6.6 – Parameters settings and corresponding number of FEs for SACC-EAM and SACC-EAM-II. s is the subcomponents size in SACC-EAM and the separable subcomponents size in SACC-EAM-II, ite the number of iterations in the SAO loop, cyc the number of cycles, FE_{det} the number of FEs consumed for the interaction detection by RDG2 and FE_{tot} the total number of FEs.

F_1					
	Best	Worst	Mean	Median	Std
S-I	1.37e+10	1.88e+10	1.68e+10	1.69e+10(+)	9.58e+08
S-II	1.38e+09	4.01e+09	2.32e+09	**2.19e+09**	6.43e+08
F_2					
	Best	Worst	Mean	Median	Std
S-I	9.25e+03	1.00e+04	9.75e+03	9.75e+03(+)	1.45e+02
S-II	4.09e+03	5.07e+03	4.52e+03	**4.54e+03**	2.31e+02
F_3					
	Best	Worst	Mean	Median	Std
S-I	1.81e+01	1.90e+01	1.87e+01	**1.87e+01**(-)	2.30e-01
S-II	2.08e+01	2.09e+01	2.09e+01	2.09e+01	3.51e-02
F_4					
	Best	Worst	Mean	Median	Std
S-I	1.45e+14	2.41e+15	6.19e+14	4.99e+14(+)	3.81e+14
S-II	2.46e+13	1.63e+14	6.95e+13	**6.41e+13**	3.66e+13
F_5					
	Best	Worst	Mean	Median	Std
S-I	2.30e+08	5.17e+08	3.24e+08	3.20e+08(+)	5.47e+07
S-II	1.36e+08	3.74e+08	2.52e+08	**2.42e+08**	6.21e+07
F_6					
	Best	Worst	Mean	Median	Std
S-I	6.79e+06	2.10e+07	1.43e+07	**1.25e+07**(-)	5.42e+06
S-II	1.72e+07	1.96e+07	1.89e+07	1.89e+07	4.28e+05
F_7					
	Best	Worst	Mean	Median	Std
S-I	6.40e+10	5.29e+11	1.75e+11	1.58e+11(+)	8.69e+10
S-II	4.47e+10	1.42e+11	8.37e+10	**8.06e+10**	2.54e+10
F_8					
	Best	Worst	Mean	Median	Std
S-I	9.53e+11	1.28e+13	3.42e+12	**2.48e+12**(-)	2.61e+12
S-II	3.48e+11	1.02e+15	7.02e+13	2.02e+13	1.64e+14
F_9					
	Best	Worst	Mean	Median	Std
S-I	5.02e+10	3.04e+11	1.96e+11	1.97e+11(+)	6.03e+10
S-II	8.52e+08	2.81e+09	1.50e+09	**1.35e+09**	4.02e+08
F_{10}					
	Best	Worst	Mean	Median	Std
S-I	1.30e+04	1.41e+04	1.36e+04	1.35e+04(+)	2.71e+02
S-II	3.28e+03	4.11e+03	3.73e+03	**3.74e+03**	2.01e+02

Table 6.7 – Statistical objective value on 50 runs for benchmark problems F_1 to F_{10} from the CEC'2010 special session on LSGO. S-I and S-II, respectively, stand for SACC-EAM and SACC-EAM-II. Best median values are marked in bold face. (+), (-) and (=) represent the fact that S-II is significantly better than, worse than, or equivalent to S-I according to the Wilcoxon rank-sum test with 5% significance level.

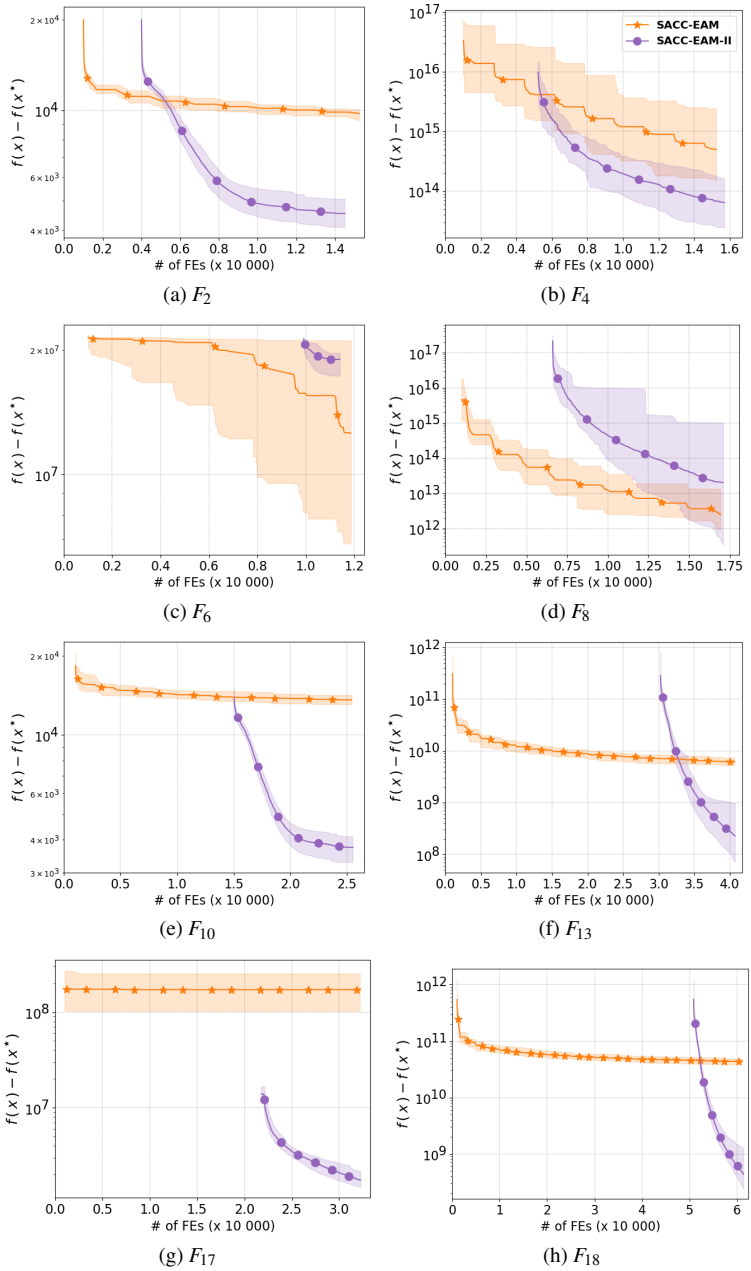
and Table 6.8. Best median values, marked in bold face, indicate that the new algorithm outperforms the SACC-EAM on 16 of the 20 benchmark functions. Among the 4 problematic cases, 3 of them are logically explained by the fact that the SACC-EAM-II is unable to deal with some large subcomponents obtained with RDG2 (one subcomponent of size 1 000 for F_3, one of size 950 for F_6, both due to the inaccurate detection for additively separable functions, and only one subcomponent containing all the nonseparable variables for F_{20}). The fourth case, F_8, will be discussed later in this section. Moreover, for function F_{11}, the SACC-EAM-II produces better results than the SACC-EAM even if it has to deal with the 500-dimensional nonseparable

F_{11}					
	Best	Worst	Mean	Median	Std
S-I	2.20e+02	2.36e+02	2.33e+02	2.34e+02(+)	3.32e+00
S-II	2.11e+02	2.15e+02	2.13e+02	**2.13e+02**	1.08e+00
F_{12}					
	Best	Worst	Mean	Median	Std
S-I	3.90e+07	1.62e+08	1.04e+08	1.04e+08(+)	2.61e+07
S-II	7.41e+05	1.18e+06	9.45e+05	**9.49e+05**	8.86e+04
F_{13}					
	Best	Worst	Mean	Median	Std
S-I	5.28e+09	7.28e+09	6.17e+09	6.04e+09(+)	4.95e+08
S-II	7.06e+07	9.80e+08	2.99e+08	**2.29e+08**	1.92e+08
F_{14}					
	Best	Worst	Mean	Median	Std
S-I	1.93e+11	4.65e+11	3.67e+11	3.67e+11(+)	5.51e+10
S-II	8.52e+08	1.61e+09	1.14e+09	**1.14e+09**	1.65e+08
F_{15}					
	Best	Worst	Mean	Median	Std
S-I	1.48e+04	1.61e+04	1.54e+04	1.54e+04(+)	3.53e+02
S-II	3.39e+03	4.64e+03	3.78e+03	**3.75e+03**	2.08e+02
F_{16}					
	Best	Worst	Mean	Median	Std
S-I	3.98e+02	4.29e+02	4.24e+02	4.26e+02(+)	6.65e+00
S-II	3.80e+02	3.85e+02	3.82e+02	**3.82e+02**	1.28e+00
F_{17}					
	Best	Worst	Mean	Median	Std
S-I	1.02e+08	2.53e+08	1.66e+08	1.71e+08(+)	3.20e+07
S-II	1.48e+06	2.16e+06	1.76e+06	**1.75e+06**	1.49e+05
F_{18}					
	Best	Worst	Mean	Median	Std
S-I	3.82e+10	4.87e+10	4.32e+10	4.30e+10(+)	1.96e+09
S-II	2.45e+08	1.28e+09	5.27e+08	**4.35e+08**	2.52e+08
F_{19}					
	Best	Worst	Mean	Median	Std
S-I	1.62e+07	1.57e+10	4.76e+09	4.01e+09(+)	4.38e+09
S-II	8.14e+07	2.10e+08	1.54e+08	**1.58e+08**	3.44e+07
F_{20}					
	Best	Worst	Mean	Median	Std
S-I	5.77e+10	7.10e+10	6.46e+10	**6.47e+10**(-)	3.28e+09
S-II	5.17e+11	7.38e+11	6.08e+11	5.95e+11	5.87e+10

Table 6.8 – Statistical objective value on 50 runs for benchmark problems F_{11} to F_{20} from the CEC'2010 special session on LSGO. S-I and S-II, respectively, stand for SACC-EAM and SACC-EAM-II. Best median values are marked in bold face. (+), (-) and (=) represent the fact that S-II is significantly better than, worse than, or equivalent to S-I according to the Wilcoxon rank-sum test with 5% significance level.

component falsely detected by RDG2. The advantages offered by the correct identification of the 10 nonseparable components of size 50 offset this negative effect.

To complete the analysis, some convergence graphs are also given in Figure 6.5. On these graphs, one can observe that the optimization process of the SACC-EAM-II starts well after those of the SACC-EAM. Indeed, the preliminary detection with RDG2 is quite expensive and costs over half of the budget in terms of FEs in this study. However, in most cases, this additional cost is rapidly recovered since the objective value with the SACC-EAM-II plummets drastically, especially at the beginning of the optimization. It still decreases with a significant rate later. The F_8 function is the only



(a) F_2 (b) F_4

(c) F_6 (d) F_8

(e) F_{10} (f) F_{13}

(g) F_{17} (h) F_{18}

Figure 6.5 – Convergence graphs for some benchmark problems from the CEC'2010 special session on LSGO. SACC-EAM (orange stars), SACC-EAM-II (purple circles). The solid line depicts the median value while the light-colored area around the solid line represents the interval between the best and the worst values over the 50 runs.

case in this study where the additional cost, albeit reduced, is not totally recovered. On the contrary, for most cases, the SACC-EAM encountered difficulties to improve the objective values significantly. Indeed, since benchmark functions contain quite large groups of interacting variables, the random grouping strategy has almost no chance to put all variables of a group in the same subcomponent. To increase these probabilities, a very large number of cycles should be performed but this is not possible in the context of computationally expensive problems.

6.3 Conclusion

This chapter presents a framework which brings together CC-EAs, designed to optimize high dimensional problems, and SAO, designed to address computationally expensive problems. Such a SACC framework has been shown to efficiently handle high dimensional and expensive optimization problems with a restricted budget in terms of function evaluations. In particular, two SACC evolutionary algorithms have been developed. The first one, called SACC-EAM, relies on multiple random decompositions while the second one, called SACC-EAM-II relies on a single decomposition based on interaction detection with RDG2.

The SACC-EAM has been compared with the standard SAO algorithm of Minamo and has shown promising results when it exploits its potential parallel architecture to compute several function evaluations of the high-fidelity and expensive optimization problem simultaneously. Moreover, the comparison with the state of the art algorithm, called SACC-JADE, allowed one to bring out the SACC-EAM efficiency to find a satisfactory solutions with a very small budget in terms of FEs.

The SACC-EAM-II has been shown to produce very strong results, compared to the SACC-EAM, on a benchmark composed of 20 high dimensional functions with different characteristics in terms of separability. The experimental results have emphasized the effectiveness of the interaction detection procedure introduced in the SACC framework. Although this step is expensive, it has a very central importance on the optimization process of the subcomponents. However, the proposed algorithm is not able to reduce the dimensionality of overlapping or fully nonseparable problems. Considering overlapping in the SACC framework would be an interesting perspective. It will be further discussed in the perspectives at the end of the thesis.

Conclusion

Synthesis

In this thesis, the challenge of high dimensionality in evolutionary optimization is addressed with cooperative co-evolutionary algorithms. In this context, we have developed new tools in order to allow such algorithms to tackle optimization problems with common features appearing in engineering and sciences. In particular, the main innovations proposed in this thesis concern the optimization of large-scale problems with the following features: constrained problems, overlapping problems and computationally expensive problems.

As a first step, the guidelines of a genetic algorithm, the standard evolutionary algorithm used in numerical experiments, is presented. It is directly followed by a description of the cooperative co-evolutionary framework. The latter benefits from the divide-and-conquer strategy to better handle large-scale optimization problems with evolutionary algorithms. In particular, the two main types of algorithms encountered in the literature rely on different decomposition strategies: the first one consists in multiple random decompositions, the other in a unique decomposition based on interaction detection. Considerable attention was given to interaction detection through the differential grouping strategy. It meets the authors' willingness of proposing a synthesis of past and ongoing work on this much studied and very useful topic.

Following that state of the art, we have introduced a new CC-EA relying on a hybrid decomposition between random and interaction detection-based decompositions. It first splits the n-dimensional decision vector into disjoint subcomponents using a differential grouping strategy. Then, it further splits large subcomponents (difficult to manage with a standard EA) into smaller subsubcomponents (much easier to manage with the EA). In this context, the interaction between SSCs derived from large nonseparable components are handled with several random decompositions. This framework has also been extended to deal with large-scale constrained problems thanks to a decomposition performed by taking into account the objective function and all the constraints. It delivers very good results, especially to deal with the violated constraints, since the proposed decomposition helps the CC-EA to quickly find a feasible solution.

A further aspect of CC-EAs studied in this thesis concerns the exploitation of overlapping features to tackle with problem instances composed of several components that interact with each other. We have introduced a CC framework that performs a decomposition with overlapped variables between subcomponents. For this purpose, we have developed a new decomposition strategy based on the recursive differential grouping and we have adapted the exchange of information between subcomponents during the optimization. This part of the thesis constitutes, in our opinion, a valuable work because it helps to address a lack that we have identified in the literature. We think that the best way to optimize large-scale overlapping problems in a CC framework is through overlapped strategies. However, although the notion of overlapping has been studied several times in the CC literature, none of these studies really provide such a framework. Our new overlapped CC-EA has been assessed on a set of large-scale overlapping problems. Unfortunately, it does not deliver as good results as expected. It seems to offer slightly better solutions but we have no certainties that it significantly outperforms the standard CC framework embedded with the RDG3. This could be due to the fact that the exchange of information through the context vector in a standard CC framework is stronger that we could expect.

Finally, the last part of this thesis focuses on computationally expensive optimization problems. They can be addressed in a reasonable amount of time with surrogate-assisted optimization techniques. For problems that are both high dimensional and computationally expensive, we have proposed to combine the cooperative co-evolutionary framework and surrogate-assisted optimization techniques. In this context, two SACC evolutionary algorithms relying on random and interaction detection decompositions have been developed. They have been shown to be able to tackle high dimensional and expensive optimization problems with a restricted budget in terms of function evaluations. In particular, the SACC algorithm relying on the decomposition obtained with RDG2 produces strong results on additively separable problems. However, it does not reduce the dimensionality when dealing with overlapping or fully nonseparable problems. At the time of writing, the SACC algorithm relying on the several random decomposition remains the best alternative for such problems, although it does not embrace the benefits of the interaction structure.

To conclude, this thesis illustrates, in some way, the benefits which could be drawn from considering natural evolution as source of inspiration for problem solving. It is well-known that evolutionary algorithms are inspired by Darwin's theory whereby evolution proceeds through natural selection, or to use the common shorthand phrase, the "survival of the fittest". The work presented in this thesis goes even further by considering cooperative co-evolution. It is inspired by the biological concept of mutualism, wherein different species live together and each benefit by working closely together (Bronstein (1994)). Using this concept to optimize large-scale problems consists in considering several species, each focusing on a part of the problem, that cooperate together to improve the objective value of the high dimensional problem. In this thesis, it has been studied through the decomposition and the optimization of large-scale problems such as constrained, overlapping and computationally expensive problems. The obtained results confirm that the CC paradigm is an efficient way ahead for solving high dimensional optimization problems with evolutionary algorithms.

Perspectives

This thesis opens up lots of attractive perspectives for future works. The main ones are presented in the following.

Concerning the hybrid-decomposition based CC-EA (and more generally concerning most CC-EAs encountered in the literature), we have focused on continuous optimization problems. It would be interesting to study the potential of the CC framework to optimize large-scale discrete optimization problems, or even mixed-variable optimization problems (Pelamatti et al. (2018)).

In the purpose of optimizing large-scale constrained problems, we have presented a CC-EA that performs only one decomposition by applying the differential grouping on both the objective function and all the constraints. This strategy may sometimes be problematic since real-world optimization problems often consider lots of constraints. One perspective would be to propose a dynamic grouping that would give more importance to the violated constraints (or to the most compromised ones at least initially) in the grouping decision scheme and update it frequently during the optimization process.

In Chapter 4, we have proposed an attempt to use an overlapped decomposition to deal with overlapping problems in a CC framework. Although the desired results were not completely achieved, the developed approach shows promising prospects. We are confident that there is scope for further improvements. The main perspective would be to further exploit the strength of the cooperation through the context vector. Other paths, such as the use of a contribution-based CC algorithm to solve overlapping problems, could also be explored. It has already been partially studied by Sun et al. (2019) but seems to be a very challenging task requiring deepest investigations.

Finally, it would also be interesting to incorporate overlapping strategies into the surrogate-assisted CC framework presented in the last part of the thesis. However, the SACC approach requires to be improved/modified before being able to exploit overlapping strategies. Indeed, as things currently stand, the global individual (i.e., the context vector but in the SACC context), is only updated when new databases are built but is fixed from one iteration to the other. The exchange of information between subcomponents are therefore quite limited and insufficient to handle overlapping problems split into several subcomponents. A key objective would be to allow more frequent updates without pulling down the consistency of databases (and therefore of the surrogate models specific to each subcomponents).

Appendices

Appendix A

Constrained benchmark problems

G2 (Michalewicz and Schoenauer (1996))

Minimize:

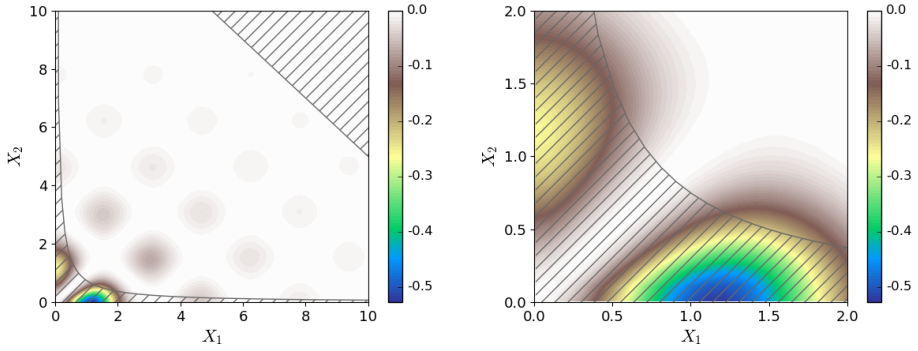
$$f(x) = -\left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

Subject to:

$$g_1(x) = 0.75 - \prod_{i=1}^{n} x_i \leq 0$$

$$g_2(x) = \sum_{i=1}^{n} x_i - 7.5n \leq 0$$

Domain: $x_i \in]0, 10]$ for $i = 1..n$



(a) $x_1, x_2 \in]0, 10]$ (b) Zoom: $x_1, x_2 \in]0, 2]$

Figure A.1 – Representation of the 2-dimensional G2 problem. Contour plots represent the objective function. Unfeasible regions are hatched.

G3 (Michalewicz and Schoenauer (1996))

Minimize:

$$f(x) = -(\sqrt{n})^n \prod_{i=1}^{n} x_i$$

Subject to:

$$g_1(x) = \sum_{i=1}^{n} x_i^2 - 1 \leq 0$$

Domain: $x_i \in [0,1]$ for $i = 1..n$

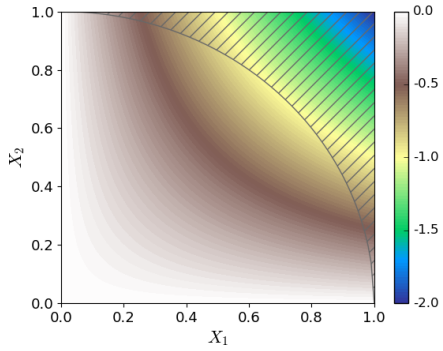


Figure A.2 – Representation of the 2-dimensional G3 problem. Contour plots represent the objective function. Unfeasible regions are hatched.

G10 (Michalewicz and Schoenauer (1996))

Minimize:

$$f(x) = x_1 + x_2 + x_3$$

Subject to:

$$\begin{aligned}
g_1(x) &= -1 + 0.0025(x_4 + x_6) \leq 0 \\
g_2(x) &= -1 + 0.0025(x_5 + x_7 - x_4) \leq 0 \\
g_3(x) &= -1 + 0.01(x_8 - x_5) \leq 0 \\
g_4(x) &= -x_1 x_6 + 833.33252 x_4 + 100 x_1 - 83333.333 \leq 0 \\
g_5(x) &= -x_2 x_7 + 1250 x_5 + x_2 x_4 - 1250 x_4 \leq 0 \\
g_6(x) &= -x_3 x_8 + 1250000 + x_3 x_5 - 2500 x_5 \leq 0
\end{aligned}$$

Domain: $x_1 \in [100, 10000]$, x_2 and $x_3 \in [1000, 10000]$ and $x_i \in [10, 1000]$ $\forall i = 4..8$

Hesse (Hesse (1973))

Minimize:

$$f(x) = -25(x_1-2)^2-(x_2-2)^2-(x_3-1)^2-(x_4-4)^2-(x_5-1)^2-(x_6-4)^2$$

Subject to:

$$\begin{aligned} g_1(x) &= 2-x_1-x_2 \le 0 \\ g_2(x) &= x_1+x_2-6 \le 0 \\ g_3(x) &= -x_1+x_2-2 \le 0 \\ g_4(x) &= x_1-3x_2-2 \le 0 \\ g_5(x) &= 4-(x_3-3)^2-x_4 \le 0 \\ g_6(x) &= 4-(x_5-3)^2-x_6 \le 0 \end{aligned}$$

Domain: $x_1 \in [0,5]$, $x_2 \in [0,4]$, $x_3 \in [1,5]$, $x_4 \in [0,6]$, $x_5 \in [1,5]$ and $x_6 \in [0,10]$

Speed Reducer (Golinski (1970); Rao (2009))

Minimize:

$$f(x) = 0.7854x_1x_2^2A - 1.508x_1B + 7.477C + 0.7854D$$

Subject to:

$$\begin{aligned} g_1(x) &= 27-x_1x_2x_3 \le 0 \\ g_2(x) &= 397.5-x_1x_2^2x_3^2 \le 0 \\ g_3(x) &= 1.93-\frac{x_2x_3x_6^4}{x_4^3} \le 0 \\ g_4(x) &= 1.93-\frac{x_2x_3x_7^4}{x_5^3} \le 0 \\ g_5(x) &= \frac{A_1}{B_1}-1100 \le 0 \\ g_6(x) &= \frac{A_2}{B_2}-850 \le 0 \\ g_7(x) &= x_2x_3-40 \le 0 \\ g_8(x) &= 5-\frac{x_1}{x_2} \le 0 \\ g_9(x) &= \frac{x_1}{x_2}-12 \le 0 \\ g_{10}(x) &= 1.9+1.5x_6-x_4 \le 0 \\ g_{11}(x) &= 1.9+1.1x_7-x_5 \le 0 \end{aligned}$$

Variables defined:

$$\begin{aligned} A &= 3.3333x_3^2+14.9334x_3-43.0934,\ B=x_6^2+x_7^2, \\ C &= x_6^3+x_7^3,\ D=x_4x_6^2+x_5x_7^2 \\ A_1 &= \sqrt{\left(\frac{745x_4}{x_2x_3}\right)^2+(16.91\times10^6)},\ B_1=0.1x_6^3 \\ A_2 &= \sqrt{\left(\frac{745x_5}{x_2x_3}\right)^2+(157.5\times10^6)},\ B_2=0.1x_7^3 \end{aligned}$$

Domain: $x_1 \in [2.6,3.6]$, $x_2 \in [0.7,0.8]$, $x_3 \in [17,28]$, x_4 and $x_5 \in [7.3,8.3]$,
$x_6 \in [2.6,3.9]$, $x_7 \in [5,5.5]$

Welded Beam (Ragsdell and Philipps (1976); Rao (2009))

Minimize:

$$f(x) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0+x_2)$$

Subject to:

$$\begin{aligned}
g_1(x) &= t - t_{max} \leq 0 \\
g_2(x) &= s - s_{max} \leq 0 \\
g_3(x) &= x_1 - x_4 \leq 0 \\
g_4(x) &= 0.10471x_1^2 + 0.04811x_3x_4(14.0+x_2) - 5.0 \leq 0 \\
g_5(x) &= d - d_{max} \leq 0 \\
g_6(x) &= P - P_c \leq 0
\end{aligned}$$

Variables defined:

$$\begin{aligned}
P &= 6000,\ L = 14,\ E = 30\times 10^6,\ G = 12\times 10^6 \\
t_{max} &= 13600,\ s_{max} = 30000,\ x_{max} = 10,\ d_{max} = 0.25 \\
M &= P\left(L + \frac{x_2}{2}\right),\ R = \sqrt{0.25(x_2^2 + (x_1+x_3)^2)} \\
J &= \sqrt{2}x_1x_2\left(\frac{x_2^2}{12} + 0.25(x_1+x_3)^2\right) \\
P_c &= \frac{4.013E}{6L^2}x_3x_4^3\left(1 - 0.25x_3\frac{\sqrt{\frac{E}{G}}}{L}\right) \\
t_1 &= \frac{P}{\sqrt{2}x_1x_2},\ t_2 = \frac{MR}{J}, t = \sqrt{t_1^2 + \frac{t_1t_2x_2}{R} + t_2^2} \\
s &= \frac{6PL}{x_4x_3^2},\ d = \frac{4PL^3}{Ex_4x_3^3}
\end{aligned}$$

Domain: $x_1 \in [0.125, 10],\ x_i \in [0.1, 10]$ for $i = 2,3,4$

Appendix B

Overlapping benchmark problems

The overlapping benchmark set is composed of 6 overlapping benchmark problems derived from the CEC'2013 competition on LSGO (Xi et al. (2013)). In particular, the problems f_1 to f_5 are built according to the following equation:

$$f(\vec{x}) = \sum_{i=1}^{|S|} \omega_i f_{\text{basis}}(\vec{z}_i) \tag{B.0.1}$$

where

- $\vec{x}$ is the 905-dimensional decision vector.
- S is a list that describes the size of subcomponents:

  $S = [50, 50, 25, 25, 100, 100, 25, 25, 50, 25, 100, 25, 100, 50, 25, 25, 25, 100, 50, 25]$.

- ω_i is a weight factor.
- $\vec{z}_i = \Phi(\vec{y}_i)$ is a modified version of the vector $\vec{x}_i$. This modification includes a transformation to create local small irregularities, a transformation to break the symmetry and a rotation of the fitness landscape.
- $\vec{y}_i$ is a shifted and permutated version of $\vec{x}_i$. In particular, $\vec{y}_i$ contains the decision variables whose indices vary from $P[C_{i-1} - (i-1)m + 1]$ to $P[C_i - (i-1)m]$.
- $\vec{x}_i$ is a $|S_i|$-dimensional subvector of $\vec{x}$.
- P is the vector that contains the indices permutation.
- $m = 5$ is the overlap size (see illustration in Figure B.1).
- $C_i = \sum_{j=1}^{i} S_i$ and $C_0 = 0$.

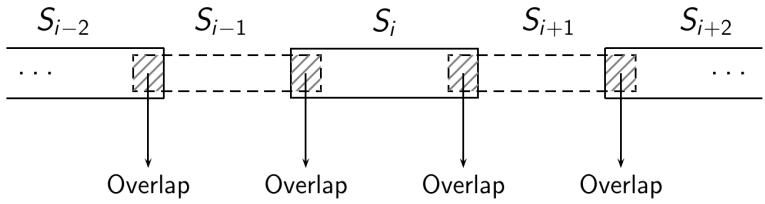


Figure B.1 – Overlapped structure of benchmark problems f_1 to f_5. Each subcomponent shares $m = 5$ overlapped variables with the previous and the next subcomponents.

The complete description of weight factors, shifts, permutations, rotations and transformations are described in the benchmark definition of the CEC'2013 competition (Xi et al. (2013)). In the benchmark set considered in this thesis, functions f_1 to f_5 are built by using Ackley, Elliptic, Rastrigin, Rosenbrock and Schwefel function as basis function in Equation (B.0.1). In particular, the f_5 function of our benchmark set is the shifted Schwefel's function with conflicting[(1)] overlapping subcomponents of the CEC'2013 benchmark set. Finally, the f_6 function is the 1000-d shifted Rosenbrock function of the CEC'2013 benchmark set.

[(1)]Note that the CEC'2013 suite also contains a benchmark function with conforming overlapping subcomponents, meaning that they share the same optimum value with respect to both subcomponent functions. Such a function can easily be optimized with a standard CC-EA and therefore, it has not been included in the benchmark set built for this study.

Bibliography

A. E. Aguilar-Justo and E. Mezura-Montes. Towards an improvement of variable interaction identification for large-scale constrained problems. In *2016 IEEE Congress on Evolutionary Computation*, pages 4167–4174, Piscataway, New Jersey, USA, 2016. IEEE.

R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, Madison, Wisconsin, USA, 2015.

A. Auger, N. Hansen, N. Mauny, R. Ros, and M. Schoenauer. Bio-inspired continuous optimization: The coming of age. Invited Talk at CEC'2007, 2007.

L. Baert, C. Dumont, C. Beauthier, C. Sainvitu, I. Lepot, and J. Blanchard. Multidisciplinary design of a low-noise propeller: Part II - Efficient aero-acoustic-mechanical design methodology exploiting surrogate models in an adaptive design space. In *Proceedings of ASME Turbo Expo 2020*, New York, NY, USA, 2020. ASME Digital Collection.

J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic algorithms and Their Applications*, pages 14–21, Hillsdale, New Jersey, USA, 1987. Lawrence Erlbaum.

R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, Princeton, New Jersey, USA, 1961.

H.-G. Beyer and H.-P. Schwefel. Evolution strategies: A comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

P. E. Black. *Dictionary of Algorithms and Data Structures*. U.S. National Institute of Standards and Technology, 2006.

J. Blanchard, C. Beauthier, and T. Carletti. A cooperative co-evolutionary algorithm for solving large-scale constrained problems with interaction detection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 697–704, New York, NY, USA, 2017. ACM.

J. Blanchard, C. Beauthier, and T. Carletti. A surrogate-assisted cooperative co-evolutionary algorithm for solving high dimensional, expensive and black box optimization problems. In *EngOpt 2018 Proceedings of the 6th International Conference on Engineering Optimization*, pages 41–52, Heidelberg, Germany, 2019a. Springer.

J. Blanchard, C. Beauthier, and T. Carletti. A surrogate-assisted cooperative co-evolutionary algorithm using recursive differential grouping as decomposition strategy. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 689–696, Piscataway, New Jersey, USA, 2019b. IEEE.

J. Blanchard, C. Beauthier, and T. Carletti. Investigating overlapped strategies to solve overlapping problems in a cooperative co-evolutionary framework. In *Optimization and Learning*, Cham, Switzerland, 2021. Springer International Publishing (accepted).

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

A. J. Booker, J. E. Dennis Jr., P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural optimization*, 17:1–13, 1999.

J. L. Bronstein. Our current understanding of mutualism. *The Quarterly Review of Biology*, 69(1):31–51, 1994.

D. S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2, 1988.

X. Cai, L. Gao, X. Li, and H. Qiu. Surrogate-guided differential evolution algorithm for high dimensional expensive problems. *Swarm and Evolutionary Computation*, 48:288–311, 2019.

X. Cai, L. Gao, and X. Li. Efficient generalized surrogate-assisted evolutionary algorithm for high-dimensional expensive problems. *IEEE Transactions on Evolutionary Computation*, 24(2):365–379, 2020.

A. Chen, Z. Ren, Y. Yang, Y. Liang, and B. Pang. A historical interdependency based differential grouping algorithm for large scale global optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 1711–1715, New York, NY, USA, 2018. ACM.

W. Chen, T. Weise, Z. Yang, and K. Tang. Large-scale global optimization using cooperative coevolution with variable interaction learning. In *Parallel Problem Solving from Nature, PPSN XI*, pages 300–309, Heidelberg, Germany, 2010. Springer.

M. Chiesa, G. Maioli, G. I. Colombo, and L. Piacentini. GARS: Genetic algorithm for the identification of a robust subset of features in high-dimensional datasets. *BMC Bioinformatics*, 21(54), 2020.

C. A. C. Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191:1245–1287, 2002.

D. A. Coley. *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific, Singapore, 1999.

R. M. Corless and N. Fillion. *A graduate introduction to numerical methods*. Springer, Heidelberg, Germany, 2013.

C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London, United Kingdom, 1859.

I. De Falco, A. D. Cioppa, and G. A. Trunfio. Large scale optimization of computationally expensive functions: An approach based on parallel cooperative coevolution and fitness metamodeling. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1788–1795, New York, NY, USA, 2017. ACM.

I. De Falco, A. D. Cioppa, and G. A. Trunfio. Investigating surrogate-assisted cooperative coevolution for large-scale global optimization. *Information Sciences*, 482: 1–26, 2019.

K. Deb. An efficient constraint handling method for genetic algorithms. In *Computer Methods in Applied Mechanics and Engineering*, pages 311–338, Amsterdam, Netherlands, 2000. Elsevier.

R. Descartes. *Discourse on method.* Liberal Arts Press, Stockbridge, Massachusetts, USA, 1956.

Q. Duan, L. Qu, C. Shao, and Y. Shi. Hierarchical decomposition based cooperative coevolution for large-scale black-box optimization. In *2019 IEEE Symposium Series on Computational Intelligence*, Piscataway, New Jersey, USA, 2019a. IEEE.

Q. Duan, C. Shao, L. Qu, Y. Shi, and B. Niu. When cooperative co-evolution meets coordinate descent: Theoretically deeper understandings and practically better implementations. In *2019 IEEE Congress on Evolutionary Computation*, Piscataway, New Jersey, USA, 2019b. IEEE.

A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Natural computing series. Springer, Heidelberg, Germany, 2015.

L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. *Foundations of Genetic Algorithms*, 2:187–202, 1993.

A. I. J. Forrester and A. J. Keane. Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1-3):50–79, 2009.

A. I. J. Forrester, A. Sobester, and A. J. Keane. *Engineering Design via Surrogate Modelling - A Practical Guide.* Wiley, 2008.

G. Fu, C. Sun, Y. Tan, G. Zhang, and Y. Jin. A surrogate-assisted evolutionary algorithm with random feature selection for large-scale expensive problems. In *Bäck T. et al. (eds) Parallel Problem Solving from Nature – PPSN XVI. PPSN 2020. Lecture Notes in Computer Science*, volume 12269, pages 125–139, Heidelberg, Germany, 2020. Springer.

M. Gavrilas. Heuristic and metaheuristic optimization techniques with application to power systems. In *Proceedings of the 12th WSEAS international conference on Mathematical methods and computational techniques in electrical engineering*, pages 95–103, New York, NY, USA, 2010. ACM.

B. Ghasemishabankareh, X. Li, and M. Ozlen. Cooperative coevolutionary differential evolution with improved augmented lagrangian to solve constrained optimisation problems. *Information Sciences*, 369:441–456, 2016.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

C. K. Goh, D. Lim, L. Ma, Y. S. Ong, and P. S. Dutta. A Surrogate-Assisted Memetic Co-evolutionary Algorithm for Expensive Constrained Optimization Problems. In *IEEE Congress on Evolutionary Computation, CEC 2011*, pages 744–749, Piscataway, New Jersey, USA, 2011. IEEE.

D. E. Goldberg. *Genetic algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Boston, Massachusetts, USA, 1989.

J. Golinski. Optimal synthesis problems solved by means of nonlinear programming and random methods. *Journal of Mechanisms*, 5(3):287–309, 1970.

J. J. Grefenstette and J. M. Fitzpatrick. Genetic search with approximate function evaluation. *Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications*, pages 112–120, 1985.

A. B. Hadj-Alouane and J. C. Bean. A genetic algorithm for the multiple-choice integer program. *Operations Research*, 45:92–101, 1997.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

R. Hesse. A heuristic search procedure for estimating a global solution of nonconvex programming problems. *Operations Research*, 21(6):1267–1280, 1973.

N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, Pennsylvania, USA, 2002.

J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. The University of Michigan Press, Ann Arbor, Michigan, USA, 1975.

J. E. Hopcroft and R. E. Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

X. M. Hu, F. L. He, W. N. Chen, and J. Zhang. Cooperation coevolution with fast interdependency identification for large scale optimization. *Information Sciences*, 381:142–160, 2017.

IEEE. Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

J. R. Jian, Z. H. Zhan, and J. Zhang. Large-scale evolutionary optimization: a survey and experimental comparative study. *International Journal of Machine Learning and Cybernetics*, 11:729–745, 2020.

R. Jin, W. Chen, and A. Sudjianto. On sequential sampling for global metamodeling in engineering design. In *Proceedings of the ASME Design Engineering Technical Conference*, Evanston, Illinois, USA, 2002. Northwestern University Press.

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

J. A. Joines and C. R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 579–584, Piscataway, New Jersey, USA, 1994. IEEE.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.

D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39(3):687–697, 2007.

A. J. Keane. Passive vibration control via unusual geometries: the application of genetic algorithm optimization to structural design. *Journal of Sound and Vibration*, 185(3):441–453, 1995.

J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, Northern California, USA, 2001.

E. Kieffer, G. Danoy, P. Bouvry, and A. Nagih. A new co-evolutionary algorithm based on constraint decomposition. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops*, Piscataway, New Jersey, USA, 2017. IEEE.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

A. Kumar, G. Wu, M. Z. Ali, R. Mallipeddi, P. N. Suganthan, and S. Das. A test-suite of non-convex constrained optimization problems from the real-world and some baseline results. *Swarm and Evolutionary Computation*, 56:1–27, 2020.

Y. Leung, Y. Gao, and Z.-B. Xu. Degree of population diversity - a perspective on premature convergence in genetic algorithms and its markov chain analysis. *IEEE Transactions on Neural Networks*, 8(5):1165–1176, 1997.

L. Li, W. Fang, Q. Wang, and J. Sun. Differential grouping with spectral clustering for large scale global optimization. In *2019 IEEE Congress on Evolutionary Computation*, pages 334–341, Piscataway, New Jersey, USA, 2019. IEEE.

X. Li and X. Yao. Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. In *2009 IEEE Congress on Evolutionary Computation*, pages 1546–1553, Piscataway, New Jersey, USA, 2009. IEEE.

X. Li and X. Yao. Cooperatively coevolving particle swarms for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 16(2):210–224, 2012.

J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. A. C. Coello, and K. Deb. Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization. Technical report, Nanyang Technological University, Singapore, 2006.

Y. Ling, H. Li, and B. Cao. Cooperative co-evolution with graph-based differential grouping for large scale global optimization. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*, pages 95–102, Piscataway, New Jersey, USA, 2016. IEEE.

Y. Liu, X. Yao, Q. Zhao, and T. Higuchi. Scaling up fast evolutionary programming with cooperative coevolution. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 2, pages 1101–1108, Piscataway, New Jersey, USA, 2001. IEEE.

X. Ma, X. Li, Q. Zhang, K. Tang, Z. Liang, W. Xie, and Z. Zhu. A survey on cooperative co-evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 23:421–441, 2019.

S. Mahdavi, M. E. Shiri, and S. Rahnamayan. Cooperative co-evolution with a new decomposition method for large-scale optimization. In *2014 IEEE Congress on Evolutionary Computation*, pages 1285–1292, Piscataway, New Jersey, USA, 2014. IEEE.

S. Mahdavi, M. E. Shiri, and S. Rahnamayan. Metaheuristics in large-scale global continues optimization: A survey. *Information Sciences*, 295:407–428, 2015.

R. Mallipeddi and P. N. Suganthan. Problem definitions and evaluation criteria for the CEC 2010 special session on constrained real-parameter optimization. Technical report, Nanyang Technological University, Singapore, 2010.

M. A. Meselhi, R. A. Sarker, D. L. Essam, and S. M. Elsayed. Enhanced differential grouping for large scale optimization. In *Proceedings of the 10th International Joint Conference on Computational Intelligence, IJCCI 2018*, pages 217–224, Setúbal, Portugal, 2018. SciTePress.

Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

B. L. Miller and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9(3):293–212, 1995.

M. N. Omidvar and X. Li. Evolutionary large-scale global optimization: An introduction. Tutorial talk at CEC'2017, 2017.

M. N. Omidvar, X. Li, Z. Yang, and X. Yao. Cooperative co-evolution for large scale optimization through more frequent random grouping. In *2010 IEEE Congress on Evolutionary Computation*, pages 1–8, Piscataway, New Jersey, USA, 2010a. IEEE.

M. N. Omidvar, X. Li, and X. Yao. Cooperative co-evolution with delta grouping for large scale non-separable function optimization. In *2010 IEEE Congress on Evolutionary Computation*, pages 1–8, Piscataway, New Jersey, USA, 2010b. IEEE.

M. N. Omidvar, X. Li, Y. Mei, and X. Yao. Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Transactions on evolutionary computation*, 10(10):1–17, 2013.

M. N. Omidvar, Y. Mei, and X. Li. Effective decomposition of large-scale separable continuous functions for cooperative co-evolutionary algorithms. In *2014 IEEE Congress on Evolutionary Computation*, pages 1305–1312, Piscataway, New Jersey, USA, 2014. IEEE.

M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao. IDG: A faster and more accurate differential grouping algorithm. Technical report, School of Computer Science, University of Birmingham, Birmingham, United Kingdom, 2015.

M. N. Omidvar, Y. M, Y. Mei, X. Li, and X. Yao. DG2: A faster and more accurate differential grouping for large-scale black-box optimization. *IEEE Transactions on evolutionary computation*, 21(6):929–942, 2017.

Y. Ong, A. J. Keane, and P. B. Nair. Surrogate-Assisted Coevolutionary Search. In *Proceedings of the 9th International Conference on Neural Information Processing*, pages 2195–2199, Piscataway, New Jersey, USA, 2002. IEEE.

J. Pelamatti, L. Brevault, M. Balesdent, E.-G. Talbi, and Y. Guerin. How to deal with mixed-variable optimization problems: An overview of algorithms and formulations. In *Advances in Structural and Multidisciplinary Optimization*, Cham, Switzerland, 2018. Springer International Publishing.

C. Peng and Q. Hui. ε-constrained CCPSO with different improvement detection techniques for large-scale constrained optimization. In *2016 49th Hawaii International Conference on System Sciences*, Piscataway, New Jersey, USA, 2016a. IEEE.

C. Peng and Q. Hui. Comparison of differential grouping and random grouping methods on εCCPSO for large-scale constrained optimization. In *2016 IEEE Congress on Evolutionary Computation*, Piscataway, New Jersey, USA, 2016b. IEEE.

M. A. Potter and K. A. De Jong. A cooperative coevolutionary approach to function optimization. In *Davidor Y., Schwefel HP., Männer R. (eds) Parallel Problem Solving from Nature — PPSN III. PPSN 1994. Lecture Notes in Computer Science*, volume 866, pages 249–257, Heidelberg, Germany, 1994. Springer.

K. Price, R. Storn, and J. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer, Heidelberg, Germany, 2005.

A. Pugachev, J. G. Li, A. L. Boyer, S. L. Hancock, Q.-T. Le, S. S. Donaldson, and L. Xing. Role of beam orientation optimization in intensity-modulated radiation therapy. *International Journal of Radiation Oncology*Biology*Physics*, 50(2): 551–560, 2001.

N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41(1): 1–28, 2005.

K. M. Ragsdell and D. T. Philipps. Optimal design of a class of welded structures using geometric programming. *Journal of Manufacturing Science and Engineering*, 98 (3):1021–1025, 1976.

S. S. Rao. *Engineering Optimization: Theory and Practice: Fourth Edition.* John Wiley and Sons, Hoboken, New Jersey, USA, 2009.

Y. Ray and X. Yao. A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning. In *2009 IEEE Congress on Evolutionary Computation*, pages 983–989, Piscataway, New Jersey, USA, 2009. IEEE.

Y. Ren and Y. Wu. An efficient algorithm for high-dimensional function optimization. *Soft Comput*, 17:995–1004, 2013.

Z. Ren, C. Chen, Y. Jin, W. Guo, Y. Liang, and Z. Feng. A fast differential grouping algorithm for large scale black-box optimization. *arXiv: Optimization and Control*, 2019a.

Z. Ren, B. Pang, M. Wang, Z. Feng, Y. Liang, A. Chen, and Y. Zhang. Surrogate model assisted cooperative coevolution for large scale optimization. *Applied Intelligence*, 49:513–531, 2019b.

S. Rippa. An algorithm for selecting a good value for the parameter c in radial basis function interpolation. *Advances in Computational Mathematics*, 11(2-3):193–210, 1999.

R. T. Rockafellar. A dual approach to solving nonlinear programming problems by unconstrained optimization. *Mathematical Programming*, 5:354–373, 1973.

K. Rosset, V. Mounier, E. Guenat, and J. Schiffmann. Multi-objective optimization of turbo-ORC systems for waste heat recovery on passenger car engines. *Energy*, 159: 751–765, 2018.

J. Rowe, D. Whitley, L. Barbulescu, and J.-P. Watson. Properties of gray and binary representations. *Evolutionary Computation*, 12(1):47–76, 2004.

C. Sainvitu, V. Iliopoulou, and I. Lepot. Global optimization with expensive functions - sample turbomachinery design application. In Springer, editor, *Recent Advances in Optimization and its Applications in Engineering*, pages 499–509. 2010.

Y. Saka, M. D. Gunzburger, and J. Burkardt. Latinized, improved LHS, and CVT point sets in hypercubes. *International Journal of Numerical Analysis and Modeling*, 4 (3):729–743, 2007.

E. Sayed, D. Essam, and R. Sarker. Dependency identification technique for large scale optimization problems. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, Piscataway, New Jersey, USA, 2012. IEEE.

E. Sayed, D. Essam, and R. Sarker. Decomposition-based evolutionary algorithm for large scale constrained problems. *Information Sciences*, 316:457–486, 2015.

S. Shan and G. Wang. Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. *Structural and Multidisciplinary Optimization*, 41(2):219–241, 2010.

M. Sharawi and M. El-Abd. A cooperative co-evolutionary LSHADE algorithm for large-scale global optimization. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 777–784, Piscataway, New Jersey, USA, 2017. IEEE.

Y. Shi, H. Teng, and Z. Li. Cooperative co-evolutionary differential evolution for function optimization. In *Wang L., Chen K., Ong Y.S. (eds) Advances in Natural Computation. ICNC 2005*, volume 3611 of *Lecture Notes in Computer Science*, pages 1080–1088, Heidelberg, Germany, 2005. Springer.

H. K. Singh, K. Alam, and T. Ray. Use of infeasible solutions during constrained evolutionary search: A short survey. In *Proceedings of Australian Conference on Artificial Life and Computational Intelligence*, pages 193–205, Heidelberg, Germany, 2016. Springer.

S. N. Sivanandam and S. N. Deepa. *Introduction to Genetic Algorithms*. Springer, Heidelberg, Germany, 2008.

A. Song, W. N. Chen, P. T. Luo, Y. J. Gong, and J. Zhang. Overlapped cooperative co-evolution for large scale optimization. In *2017 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3689–3694, Piscataway, New Jersey, USA, 2017. IEEE.

J. Stork, M. Friese, M. Zaefferer, T. Bartz-Beielstein, A. Fischbach, B. Breiderhoff, B. Naujoks, and T. Tušar. Open issues in surrogate-assisted optimization. In Springer, editor, *High-Performance Simulation-Based Optimization. Studies in Computational Intelligence*, pages 225–244. 2020.

S. Strasser, J. Sheppard, N. Fortier, and R. Goodman. Factored evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 2(21):281–293, 2017.

P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore, 2005.

C. Sun, Y. Jin, R. Cheng, J. Ding, and J. Zeng. Surrogate-assisted cooperative swarm optimization of high-dimensional expensive problems. *IEEE Transactions on Evolutionary Computation*, 21(4):644–660, 2017a.

L. Sun, S. Yoshida, X. Cheng, and Y. Liang. A cooperative particle swarm optimizer with statistical variable interdependence learning. *Information Sciences*, 1(186): 20–39, 2012.

Y. Sun, M. Kirley, and S. K. Halgamuge. Extended differential grouping for large scale global optimization with direct and indirect variable interactions. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 313–320, New York, NY, USA, 2015. ACM.

Y. Sun, M. Kirley, and S. K. Halgamuge. A recursive decomposition method for large scale optimization. *IEEE Transactions on evolutionary computation*, 22(5): 647–661, 2017b.

Y. Sun, M. N. Omidvar, K. M, and X. Li. Adaptive threshold parameter estimation with recursive differential grouping for problem decomposition. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 889–896, New York, NY, USA, 2018. ACM.

Y. Sun, X. Li, A. Ernst, and M. N. Omidvar. Decomposition for large-scale optimization problems with overlapping components. In *2019 IEEE Congress on Evolutionary Computation*, pages 326–333, Piscataway, New Jersey, USA, 2019. IEEE.

T. Takahama and S. Sakai. Constrained optimization by ε constrained particle swarm optimizer with ε-level control. *Soft Computing as Transdisciplinary Science and Technology. Advances in Soft Computing*, 29:1019–1029, 2005.

K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang. Benchmark functions for the CEC'2008 special session and competition on large-scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, 2007.

K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise. Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, 2009.

M. Tezuka, M. Munetomo, and K. Akama. Linkage identification by nonlinearity check for real-coded genetic algorithms. In *Deb K. (eds) Genetic and Evolutionary Computation – GECCO 2004*, volume 3103 of *Lecture Notes in Computer Science*, pages 222–233, Heidelberg, Germany, 2004. Springer.

J. Tian, Y. Tan, J. Zeng, C. Sun, and Y. Jin. Multiobjective infill criterion driven gaussian process-assisted particle swarm optimization of high-dimensional expensive problems. *IEEE Transactions on Evolutionary Computation*, 23(3):459–472, 2019.

V. Torczon and M. V. Trosset. Using approximations to accelerate engineering design optimization. Technical report, Institute for Computer Applications in Science and Engineering, Hampton, Virginia, USA, 1998.

G. A. Trunfio. Enhancing the firefly algorithm through a cooperative coevolutionary approach: An empirical study on benchmark optimization problems. *International Journal of Bio-inspired Computation*, 6(2):108–125, 2014.

G. A. Trunfio. Adaptation in cooperative coevolutionary optimization. *Adaptation and Hybridization in Computational Intelligence*, 18:91–109, 2015.

F. van den Bergh and A. P. Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8:225–239, 2004.

F. Wang, W. Shen, D. Boroyevich, S. Ragon, V. Stefanovic, and M. Arpilliere. Design optimization of industrial motor drive power stage using genetic algorithms. In *2006 CES/IEEE 5th International Power Electronics and Motion Control Conference*, Piscataway, New Jersey, USA, 2006. IEEE.

K. Weicker and N. Weicker. On the improvement of coevolutionary optimizers by learning variable interdependencies. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1627–1632, Piscataway, New Jersey, USA, 1999. IEEE.

F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6): 80–83, 1945.

A. H. Wright. Genetic algorithms for real parameter optimization. *Foundations of Genetic Algorithms*, 1:205–218, 1991.

G. Wu, R. Mallipeddi, and P. N. Suganthan. Problem definitions and evaluation criteria for the CEC 2017 special session on constrained real-parameter optimization. Technical report, National University of Defense Technology, Changsha, 2017.

L. Xi, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin. Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization. Technical report, RMIT University, Melbourne, 2013.

H.-B. Xu, F. Li, and H. Shen. A three-level recursive differential grouping method for large-scale continuous optimization. *IEEE Access*, 8:141946–141957, 2020.

M. Yang, A. Zhou, C. Li, and X. Yao. An efficient recursive differential grouping for large-scale continuous problems. *IEEE Transactions on Evolutionary Computation*, 25(1):159–171, 2021.

X.-S. Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, Frome, United Kingdom, 2008.

Z. Yang, K. Tang, and X. Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008a.

Z. Yang, K. Tang, and X. Yao. Multilevel cooperative coevolution for large scale optimization. In *2008 IEEE Congress on Evolutionary Computation*, pages 1663–1670, Piscataway, New Jersey, USA, 2008b. IEEE.

X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, 1999.