Democratic and Popular Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Mohamed Khider - BISKRA
Faculty of Exact Sciences, Natural Sciences and Life
Computer Science Department

Thesis

Presented to obtain the diploma of academic Master in

Computer Science

Option: **Software Engineering and Distributed Systems**

Implementation and Parallelization of Multi-objective Evolutionary Algorithm Based on Decomposition and Directed Mating

---

By:
ABIDALLAH Nedjma

Defended the 06/07/2019, in front of the jury composed of:

4.1 cm	President	
Kahloul Laid	MCA	Supervisor
4.3 cm	Examiner	

University Year: 2019/2020

Acknowledgements

All our thanks and gratitude for Allah the Almighty who gave us all the courage and the will to go till the end of this level.

I owe a major thank and gratitude to my family who understood and supported me to continue my studies, who raised to be a good person.

I owe a major thank to my supervisor **Pr. KAHLOUL Laid** who has always been available and attentive to my questions, my deep respect and thanks for his commitment to giving me the opportunity to carry out this project. Your support and reflections on the work throughout the long-lasting process have been priceless.

I would also like to express my gratitude to the jury members: **...** and **...** for reading and evaluating my thesis.

I never forget to thank all my teachers at Computer Science Department who taught me the basic principles of computer science. At last but not least, I appreciate the support of **Ph.D. student Leila BELAICHE** for her assistance and guidance throughout the planning and execution of this research. Also i would like to thank my friends for their help and support.

Abstract

As an evolutionary approach to solve constrained multi-objective optimization problems (CMOPs), a multi objective evolutionary algorithm (MOEA) using the two-stage non-dominated sorting and the directed mating (TNSDM) [21] has been proposed. However, since TNSDM uses the non-dominated sorting, its search performance deteriorates when the number of objectives is increased. For multi-objective optimization, the decomposing objective space is a promising approach, and MOEA/D [34] is known as its representative algorithm. However, in the conventional MOEA/D only feasible solutions are selected as parents. For solving constrained multi-objective optimization problems (CMOPs), recently a constrained multi-objective optimization evolutionary algorithm (CMOEA) that combines the directed mating concept and the decomposition approach has been proposed which is CMOEA/D-DMA [19] (Constrained Multi-objective Optimization Evolutionary Algorithms based on Decomposition and directed mating). Although, CMOEA/D-DMA has been improved its efficiency in solving CMOPs with large scale, being an evolutionary algorithm means that it will certainly be characterized by long execution time. One of the reasons that push us to adopt parallel evolutionary algorithms (PEAs) is to obtain best results with an execution time much lower than the one of their sequential versions.

In this thesis, we propose a new parallel version of CMOEA/D-DMA (i.e., PCMOEA/D-DMA). The experimental results using modified constrained DTLZ problem (mcDTLZ), show that the proposed parallel algorithm achieves higher search performance by utilizing infeasible solutions even if it doesn't surpass its sequential version in the hypervolume HV values, but a notable time reduction has been achieved.

Key words: *CMOP, CMOEA, PEA, CMOEA/D-DMA, PCMOEA/D-DMA*

Résumé

En tant qu'approche évolutive pour résoudre les problèmes d'optimisation multi-objectifs à contraintes (CMOPs), un algorithme évolutif multi-objectif (MOEA) utilisant le tri non dominé en deux étapes et l'accouplement dirigé (TNSDM) a été proposé [21]. Cependant, comme TNSDM utilise le tri non dominé, ses performances de recherche se dégradent lorsque le nombre d'objectifs augmente. Pour l'optimisation multi-objectifs, la décomposition de l'espace des objectifs est une approche prometteuse et MOEA/D [34] est connu comme son algorithme représentatif. Cependant, dans le MOEA/D conventionnel, seules les solutions faisables sont sélectionnées comme parents. Pour résoudre les problèmes d'optimisation multi-objectifs à contraintes (CMOP), récemment un algorithme évolutif d'optimisation multi-objectifs à contraintes (CMOEA) combinant le concept d'accouplement dirigé et l'approche de décomposition a été proposé CMOEA/D-DMA [19] (algorithme évolutif d'optimisation multi-objectifs à contraintes basés sur la décomposition et l'accouplement dirigé). Bien que CMOEA/D-DMA ait amélioré son efficacité dans la résolution des CMOP à grande échelle, mais le fait qu'il soit un algorithme évolutif signifie qu'il sera certainement caractérisé par un grand temps d'exécution. Une des raisons qui nous pousse à adopter les algorithmes évolutifs parallèles (PEA) est d'obtenir les meilleurs résultats avec un temps d'exécution bien inférieur à celui de leurs versions séquentielles.

Dans ce travail, nous proposons une nouvelle version parallèle de CMOEA/D-DMA (i.e., PCMOEA/D-DMA). Les résultats expérimentaux sur le célèbre problème DTLZ modifié avec contraintes (mcDTLZ), montrent que l'algorithme parallèle proposé améliore les performances de recherche en utilisant des solutions non faisables. En effet, même si la version parallèle ne surpasse pas la version séquentielle dans les valeurs de l'hypervolume, mais une reduction de temps à été atteinte.

Mots clés : *CMOP, CMOEA, PEA, CMOEA/D-DMA, PCMOEA/D-DMA*

Contents

List of Figures

List of Tables

List of Algorithms

General Introduction

General Introduction

Optimization is an important tool in making decisions and in analyzing physical systems. In engineering, an optimization problem is the problem of finding the best solution (s) from among the set of all feasible solutions. Most real-world search and optimization problems are naturally posed as multi-objective optimization problem (MOP). The majority of engineering optimization problems need to make multiple targets all reach the optimal in a given region, but it is regrettable that goals are generally conflicting. A MOP where the objective functions are optimized under given constraints is called constrained multi-objective optimization problem (CMOP) and since most real-world problems expose constraints this kind of problems has been frequently appear. A CMOP differs from a single-objective optimization problem because it contains several objectives that require optimization. For CMOP, with several (possibly conflicting) objectives, there is usually no single optimal solution. Therefore, the decision maker is required to select a solution from a finite set by making compromises. A suitable solution should provide an acceptable performance over all objectives.

Solving Constrained multi-objective Optimization Problems (CMOPs) is challenging task in the field of computer optimization. Many researchers have put efforts to solve CMOPs using techniques such as Dynamic Programming, Non Linear Programming etc. These methods are generally trapped in local optima. The solution to this gap is Evolutionary Algorithms (EAs), which work as a promising technique for wide range of Constrained Optimization Problems. Evolutionary Algorithm is a metaheuristic technique that has been used to solve MOPs and CMOPs based on the mechanism of natural selection and natural genetics. Algorithms that solve a multi-objective optimization problems using EAs are called Multi-ojective Optimization Evolutionary Algorithms (MOEAs). MOEAs try to find Pareto optimal solutions (POS) showing the trade-off among objective functions in multi-objective optimization problems (MOPs) [3]. MOEAs are particularly suited to solve MOPs since they can obtain a set of Pareto optimal solutions (POS) from the population in a single run of the algorithm.

When we address constrained MOPs (CMOPs) involving several constraints, we need to introduce a mechanism to obtain feasible solutions from infeasible ones in MOEAs. So far, several constraint-handling methods studied for single-objective optimization have been extended for solving CMOPs [16]. However most of constrained MOEAs (CMOEAs) are selecting only feasible solutions as parents which guide to a deterioration of the search performance. That's what guided to the occurrence of constraints handling methods selecting also infeasible solution having better scalarizing function than feasible ones as parents. But using those techniques does not improve the search performance enough, because most of real words problems are a large scale which needed to be decomposed into sub-problems using the decomposition concept that had been introduced in this paper [34].

Constrained Multi-objective Optimization Evolutionary Algorithms based on Decomposition and directed mating (CMOEA/D-DMA) is a CMOE algorithm that gather both decom-

position concept and a constraint handling technique that select also infeasible solutions as parents. This algorithm has been proposed to improve the search performance by combining a MOEA based on Decomposition, since MOEA/D is known as its representative algorithm with the Directed Mating and Archives of infeasible solutions. The directed mating has been proposed in TNSDM algorithm, However, since TNSDM uses the non-dominated sorting, its search performance deteriorates when the number of objectives is increased. The directed mating in CMOEA/D-DMA selects useful infeasible solutions having better scalarizing function values than feasible ones as parents and maintains them in archives.
Although CMOEA/D-DMA works to give excellent results compared to other CMOEAs, being an evolutionary algorithm means that it will certainly be characterized by long execution time. It can not be ignored that we are always looking for best results in less time.

The main reason for using parallel evolutionary algorithms (PEAs) is to obtain efficient results with an execution time much lower than the one of their sequential versions in order to resolve more complex problems. In this dissertation, a parallel version of CMOEA/D-DMA (i.e., PCMOEA/D-DMA) is proposed, However we are not the first who tried to parallelize CMOEA/D-DMA. In [18] a study of parallelization has been proposed, but the difference between us is that we use another technique of parallelization and also we test the parallel version on a different test problem (we used mCDTLZ [20] and they used m objectives k knapsacks problems [23]). Also They were not taken into account in their study the execution time, they only measured HV values. PCMOEA/D-DMA tries to improve the search performance by decomposing the population into sub-populations and applies the sequential version.

This thesis starts with an introduction which presents the problem and a proposed solution. The thesis is composed of two parts, the first part focuses on the theoretic level (State of the art), the second part shows our contribution in this work.

Part I is divided into two chapters.

- chapter 1 represents terms related to the chosen algorithm, describes the algorithm itself (CMOEA/D-DMA), and also describe other algorithms that have relation with the mentioned algorithm.
- chapter 2 gives the basic concepts of parallel computing.

Part II concentrates on the implementation and parallelization of CMOEA/D-DMA with the demonstration of its efficiency. It is composed of 2 chapters.

- chapter 3 illustrates analysis and design of our application in three main levels (Global design, CMOEA/D-DMA design, and PCMOEA/D-DMA dsign)
- chapter 4 represents the implementation of the both algorithms (i.e., CMOEA/D-DMA, PCMOEA/D-DMA) and the implementation of the whole application with the demonstration of the implemented algorithms efficiency by comparing their results with existing CMOEA results in literature (i.e., CMOEA/D, TNSDM and CNSGA-III).

The thesis ends with a conclusion which evaluates obtained results, and discusses some perspectives.

Part I

STATE OF THE ART

Chapter 1

Constrained Multi-objective Optimization Evolutionary Algorithm (CMOEA)

Chapter 1

Constrained Multi-objective Optimization Evolutionary Algorithm

Introduction

In the real world, there are many problems having two or more objectives (often conflicting) that we aim to optimize at the same time. This type of problem is called multi-objective optimization problems (MOPs). Since the 1970s, many techniques and methods trying to solve MOPs (searching for uniformly distributed, near optimal, and well-extended Pareto front) have been developed, but the most popular among them is multi-objective evolutionary algorithms (MOEAs) which are an heuristic method, this last were a popular choice, mainly because of their flexibility (i.e., they require little domain specific information) and their ease of use. The EAs were not only used to solve MOPs but also to solve MOPs that expose constraints, because there are problems that require conditions. This class of MOPs called constrained multi-objective optimisation problems (CMOPs). Most of MOEAs solving CMOPs use only feasible solutions for the selecting process, but this deteriorate the search performance because the infeasible solutions may have better objective function values than feasible ones which guide to well-extended Pareto front.

In this chapter, we present some terminologies related to MOEAs for solving CMOPs and we will also introduce one of the algorithms developed recently that use the directed mating and the decomposition concepts to improve the search performance. The algorithm is called constrained multi-objective optimization evolutionary algorithms based on decomposition and directed mating (CMOEA/D-DMA).

1.1 Constrained optimization problem

In this section we will introduce some definitions related to optimization problems.

Definition 1.1.1. *Optimization [22]: is an important tool in making decisions and in analyzing physical systems.*

Definition 1.1.2. *Optimization problem: an optimization problem in mathematical terms, is the problem of finding the best solution from among the set of all feasible solutions [22]. Formally, in a domain P, and subject to an objective function f, there is an optimal solution ($x^* \in P$), where: [7]*

- ***max f(x) :*** $f(x) \leq f(x^*)$, $\forall\ x \in P$. *Where $f(x^*)$ is the optimal solution and there is not any other bigger solution.*

$\Longrightarrow$ *Or*

- ***min f(x) :*** $f(x) \geq f(x^*)$, $\forall\ x \in P$. *Where f(x^*) is the optimal solution and there is not any other smaller solution.*

Here is an example to approximate the optimization problem concept.

Example 1.1.1. *Find dimensions of a square, whose perimeter y is as small as possible and his rib is x.*
Minimize $y = 4 * x$
In this example P is all the values that x can obtain in $\mathbb{R}$, *f is the value of y that will be minimized and* x^* *or the optimal solution is the x picked from P that satisfies Min y.*

Definition 1.1.3. *constrained optimization problem [32]: constrained optimization is the process of optimizing an objective function with respect to some variables in the presence of constraints on those variables. The objective function is either a cost function or energy function, which is to be minimized, or a reward function or utility function, which is to be maximized. A general constrained minimization problem may be written as follows:*

min	$f(X)$	
subject to	$g_i(X) = c_i$ ***for*** $i = 1, ..., k$	***Equality constraints***
	$h_j(X) \geq d_j$ ***for*** $j = 1, ..., b$	***Inequality constraints***

where $g_i(X) = c_i$ ***for*** $i = 1, ..., k$ *and* $h_j(X) \geq d_j$ ***for*** $j = 1, ..., b$ *are constraints that are required to be satisfied, and* $f(X)$ *is the objective function that needs to be optimized subject to those constraints.*

We will take the same example (Example 1.1.1) and we will add a constraint.

Example 1.1.2. *Find dimensions of a square, whose perimeter y is as small as possible and his rib is x, with an area equal to 1000* m^2.
Minimize $y = 4 * x$
constraint $1000 \leq x^2$
In this example P is all values that x can obtain in $\mathbb{R}$, *f is the value of y that will be minimized and* x^* *or the optimal solution is the x picked from P that satisfies Min y and respects the constraint. The constraint here is a* $h_j(X)$ *type.*

1.2 Constrained multi-objective optimization problem

In this section we will introduce the constrained MOPs and few additional definitions that are required to introduce the notion of optimality used in multi-objective optimization.

Definition 1.2.1. *Constrained MOPs (CMOPs) [19]: CMOPs are concerned with finding solutions x minimizing m objective functions* $f_i(i = 1, 2, ..., m)$ *subject to satisfy k constraint functions* $g_j(j = 1, 2, ..., k)$.*CMOPs are defined as*

Minimize/Maximize	$f_i(X)$	$(i = 1, ..., m)$
subject to	$g_j(X) \geq 0$	$(j = 1, ..., k)$

Here is an example for further explanation.

Example 1.2.1. *In the following a simple CMOP description.*

$$\begin{cases} \textbf{\textit{Minimize}}\ f_1(x) = 4 - 2x, & \textbf{\textit{subject to}} \quad g_1(x) = f_1 + 1 \geq 0 \\ \\ \textbf{\textit{Minimize}}\ f_2(x) = 3x - 1, & \textbf{\textit{subject to}} \quad g_2(x) = \sqrt{f_2} - 3 > 2 \end{cases}$$

m *or number of objective functions is 2, where $f_1(x)$ is the first objective function to be minimized and $f_2(x)$ is the second one that will be also minimized. Each objective function has one constraint value ($k = 1$ for each f), $g_1(x)$ is for f_1 and $g_2(x)$ for f_2. P is all values that x can obtain in $\mathbb{R}$. Because there are many objective functions x^* can become more than one value, it can become a vector of all values that respect all the m objective functions and satisfy k constraint functions.*

Remark. *Solutions satisfying all k constraints are called feasible, and solutions not satisfying all k constraints are called infeasible.*

Definition 1.2.2. *constraint violation vector [19]:The constraint violation vector $v(x)$ is defined as*

$$v_j(x) = \begin{cases} |g_j(x)|, & if g_j(x) < 0 \\ \\ 0, & otherwise \end{cases} \qquad (j = 1, 2, ..., k)$$

Also, the sum of constraint violation values is $\Omega(x) = \sum_{j=1}^{k} v_j(x)$

In MOPs, generally, there is not an ideal solution optimizing all m objective functions due to the trade-off among objectives. Therefore, the concept of Pareto dominance is introduced.

Definition 1.2.3. *Pareto dominance [19]: Pareto dominance between x and y in minimization problems is defined as follows: If*

$$\forall i : f_i(x) \leq f_i(y) \ \wedge \ \exists i : f_i(x) < f_i(y) \quad (i = 1, 2, ..., m)$$

is satisfied, x dominates y on objective function values, which is denoted by $x \succ_f y$ in the following. In the case of maximization problems, the inequalities of the equivalent are reversed.

Remark. *A feasible solution x not dominated by any other feasible solution is said to be a non-dominated solution.*
The set of non-dominated solutions in the solution space is called Pareto optimal solutions (POS), and the trade-off among objective functions represented by POS in the objective space is called Pareto front. POS and PF are shown in 1.1

Definition 1.2.4. *Pareto optimal solutions (POS) [20]:The set of Pareto optimal solutions (POS) is defined as:*

$$POS = \left\{ \ x \in F_s \mid \neg \exists y \in F_s : \ y \succ_f x \ \right\}.$$

where, F_s is the set of all feasible solutions in the solution space.

Definition 1.2.5. *Pareto front [20]: the Pareto front is the set of objective vectors of POS. The Pareto front is defined as:*

$$PF = \left\{ \ f(x) \mid x \in POS \ \right\}.$$

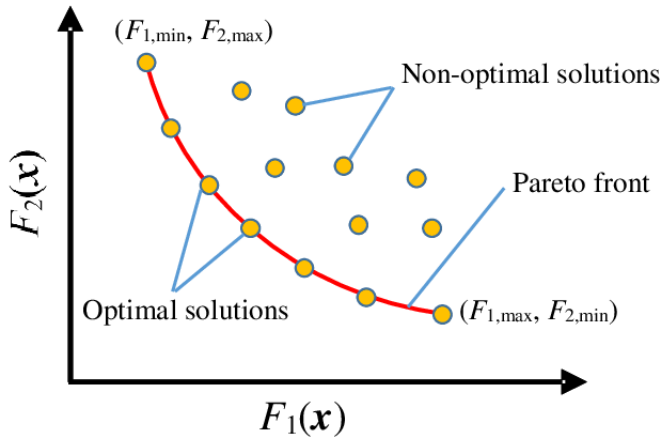


Figure 1.1 – POS and PF (From [15]).

1.3 Constraint-handling in MOEAs

When we use EAs(Evolutionary algorithms) to solve CMOPs, infeasible solutions are generated during the search. Therefore, the EAs need to use the constraint-handling method. Generally constraint-handling methods are classified into two approaches.

1. The first approach only use feasible solutions to fill up the population. The simplest method is death penalty [2] which does not allow infeasible solutions to be included in a population. It is easy to use but difficult to solve problems having a small feasible region. In addition, there are repair methods which turn infeasible solutions into feasible ones [36, 9], preserving the feasibility of the solutions methods generating only feasible solutions by using special genetic operators and/or chromosome representations. While these methods can indeed provide feasible solutions, they require a problem specific procedure and have limited applicability.

2. The other approach use both infeasible and feasible solutions by evolving the infeasible solutions to feasible ones. Penalty methods modifying objective function values based on constraint violation values [4, 33] and constraint-dominance considering constraints and objectives separately are included to this approach. Recently, some methods utilizing infeasible solutions for the solution search even if the population has many feasible solutions [21, 25] have been proposed. The algorithm that we will implement follows the second approach as it is shown in the next section.

1.4 CMOEA/D-DMA

For this part we will talk about the algorithm that we will implement CMOEA/D-DMA [19] and about algorithms that are related to it.
We will present basic definitions that are needed to understand the algorithm before providing its description.

Definition 1.4.1. *Scalarizing function [19]: scalarizing function or so-called decomposition method transform a MOP or CMOP to a single-objective problem by combining the objective functions to form a single scalar objective function. There are many decomposition methods, in this algorithm the one used is the weighted Tchebycheff function optimization formulated by*

$$Minimize\ g(x|\lambda) = \max_{1\leq j\leq m} \{|f_j(x) - z_j| \ .\ \lambda_j\} \tag{1.1}$$

where, z is the base point to decompose the objective space and λ is a weight vector.

Definition 1.4.2. *Weight vector λ [19]: In each decomposition problem, a non negative weight vector defines a single scalarizing function. The selected algorithm use C_{H+m-1}^{m-1} $(= N)$ kinds of weight vectors that satisfy $\Sigma_{j=1}^{m}\lambda_j = 1.0$. The λ values can be one of $\{0/H, 1/H, ..., H/H\}$ (H is the decomposition parameter).*

Definition 1.4.3. *Directed mating [18]: the directed mating we are talking about is based on archives. To generate an offspring y, CMOEA/D-DMA focuses on a weight vector λ^l and select the solution x^l belonging to the weight vector as a primary parent. If x^l is feasible and the archive population A^l is not empty ($|A^l| > 0$), the secondary parent is selected from A^l randomly. This mating is called directed mating (view the Figure 1.3) . Otherwise, we select the second parent from neighbor solutions randomly, and this is the conventional mating (view the Figure 1.2).*

For solving constrained MOPs (CMOPs), Constrained MOEA/D with Directed Mating and Archives of useful solutions (CMOEA/D-DMA) [19] has been proposed. In CMOEA/D-DMA, each weight vector maintains not only the best feasible solution but also infeasible solutions having better scalarizing function values than the feasible solutions as an archive population. The archived infeasible solutions may have useful information for feasible solution to converge toward the Pareto front. To utilize them, the best feasible solution and one of archived infeasible solutions belonging to the same weight vector are mated to generate offspring. Therefore, it does not need to use the neighbor solutions like in classical EAs . Based on the description above we can conclude that the algorithm works as follow:

- Firstly, the algorithm decomposes CMOP into a number of single-objective sub-problems using a set of weight vectors and scalarizing functions.
- Then it creates an archive of useful infeasible solutions A_i for each weight vector.
- After that, it applies directed mating if it's possible.
- Finally,the algorithm updates solutions and archive with the update of base point.

The last 2 steps are repeated until the condition (number of generations in our case) is fulfilled and then we obtain POF and POS.

Before passing to detail the CMOEA/D-DMA, we will introduce other algorithms that are related to the selected algorithm.

1.4.1 TNSDM

The first algorithm is TNSDM (Two stage Non-dominated Sorting and Directed Mating) [21], where the directed mating was proposed. To utilize generated infeasible solutions, TNSDM selects infeasible solutions having better scalarizing function than feasible ones as parents.

TNSDM introduces two-stage non-dominated sorting and directed mating to rank solutions while considering constraint violation values and objective function ones and to utilize useful infeasible solutions as a parent, respectively. In TNSDM, firstly, the entire population is classified into several fronts by non-dominated sorting based on constraint violation values and then each front is reclassified into fronts by non-dominated sorting based on objective function values. As the result, upper front includes solutions having lower constraint violation values and better objective function values. The half of the entire population is selected as the parent population from upper fronts while simultaneously considering the crowding

distance (CD) [6]. After that, to generate the offspring population, directed mating is performed. First, a primary parent p_a is selected from the parent population by using the tournament selection. Next, a set of candidate solutions M dominating p_a in the objective space is picked from the entire population including infeasible solutions. If the primary parent p_a is feasible and the number of solutions in M is more than or equal to two ($|M| \geq 2$), the secondary parent p_b is selected from M by using the tournament selection. Otherwise, p_b is selected from the parent population by using the tournament selection. Then, an offspring is generated from the parents p_a and p_b. Although infeasible solutions can be a secondary parent p_b, there is a possibility that p_b has valuable genetic information to enhance the convergence of primary p_a toward Pareto front since p_b dominates p_a in the objective space [19].

1.4.2 CMOEA/D

The second algorithm is CMOEA/D [11] (Constrained Multi-Objective Evolutionary Algorithm based on Decomposition). Which is a variant of MOEA/D [34] involving a constraint handling technique. MOEA/D decomposes a multi-objective problem into a number of sub-problems. Each sub-problem is formulated by a scalarizing function g with one of uniformly distributed weight vector set $\lambda^i (i = 1, 2, ..., N)$. Each element λ^i_j $(j = 1, 2, ..., m)$ is one of$\{0/H, 1/H, ..., H/H\}$ based on the decomposition parameter H, and C^{m-1}_{H+m-1} $(= N)$ kinds of weight vectors satisfying $\Sigma^m_{j=1}\lambda^i_j = 1.0$ are used for the solution search. The weighted Tchebycheff function optimization [34, 17] is the one used in CMOEA/D-DMA.

To generate an offspring y, CMOEA/D focuses on a weight vector λ^i , randomly selects parents from T-neighbor solutions of the focused weight λ^i , and applies genetic operators to them. An example is shown in Figure 1.2. In this case, they focus on weight vector λ^4, and x^3 and x^4 are selected from $T = 3$ neighbors as parents. MOEA/D tries to replace existing solutions paired with T-neighbors of the focused λ^i with the generated offspring y. In the case of CMOEA/D, if any of the following conditions are true, an existing solution x^j is replaced by y [19].

1. x^j and y are feasible, and $g(y|\lambda^j)$ is better than $g(x^j|\lambda^j)$.
2. x^j is infeasible and y is feasible.
3. x^j and y are infeasible, and x^j has a higher value of the sum of constraint violation values than y.

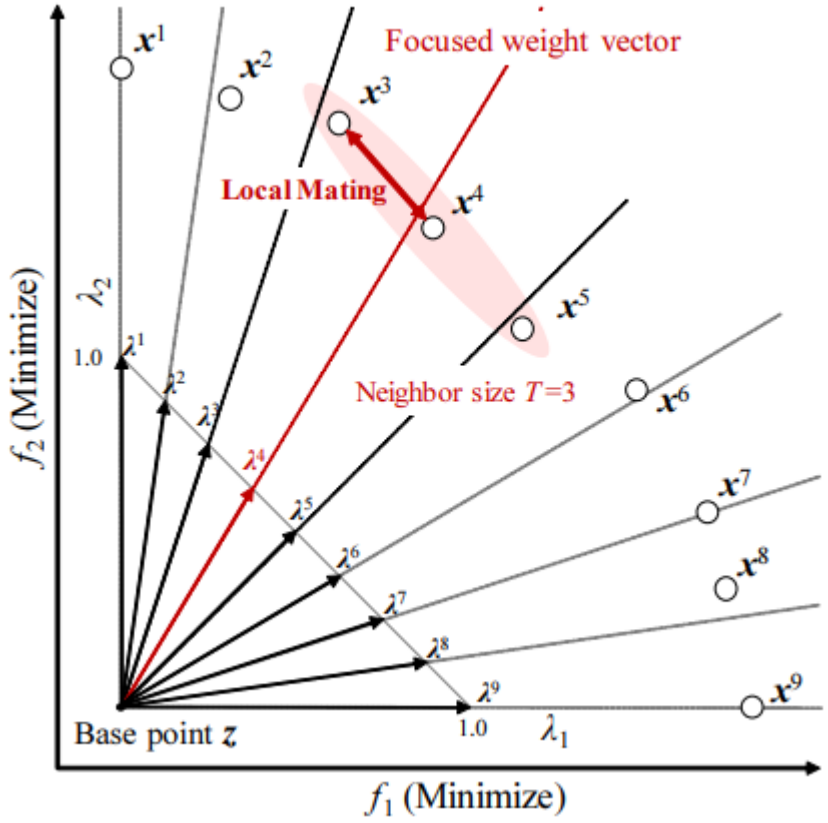


Figure 1.2 – Local mating in the conventional CMOEA/D (From [19]).

1.4.3 The chosen algorithm CMOEA/D-DMA

As we mentioned before, the directed mating has been introduced in TNSDM algorithm. This last improved the search performance by utilizing infeasible solutions having better objective function values than feasible ones in the population as parents in solving CMOPs. However, the search performance of TNSDM is deteriorated as the number of objectives is increased since the TNSDM is a dominance-based algorithm.The most of solutions in the population become non-dominated, and the selection pressure to evolve the solutions toward the Pareto front is deteriorated [10]. Among the approaches that are used to solve MOPs, the decomposition approach has been known as one of the promising approaches for solving MOPs [30]. And Since the MOEA/D is the representative algorithm employing the decomposition approach it was used to improve the search performance introducing the directed mating concept for solving CMOPs in the proposed algorithm CMOEA/D-DMA.

The pseudo-code of the proposed algorithm CMOEA/D-DMA is shown in **Algorithm 1**. The main differences between the conventional CMOEA/D and the proposed CMOEA/D-DMA is as follows:

1. Archive of useful infeasible solutions
2. Directed mating with archives
3. Update of solutions and archives
4. Update of base point

Algorithm 1 Pseudo-code of the chosed CMOEA/D-DMA [19]

Input: the number of objectives m, the decomposition parameter H, the number of weight vectors and solutions in the population N, the neighborhood size T

Output: the non-dominated set of solutions

1: $\mathcal{L} = \{\lambda^1, ..., \lambda^N\} \leftarrow$ Generate weight vectors (H, m)
2: **for each** $\lambda^i \in \mathcal{L}$ **do**
3: $\quad \mathcal{B}_i = \{i_1, ..., i_T\} \leftarrow$ Find nearest neighbor weight indices
4: **end for**
5: $\mathcal{P} = \{x^1, x^2, ..., x^N\} \leftarrow$ Randomly generate the population
6: $\{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_\mathcal{N}\} \leftarrow$ Initialize with $\{\emptyset, \emptyset, ..., \emptyset\}$
7: **repeat**
8: $\quad$ **for each** $i \in \{1, 2, ..., N\}$ **do**
9: $\quad\quad P_a \leftarrow$ Select x^i as a parent
10: $\quad\quad$ **if** P_a is feasible and $|\mathcal{A}_i| > 0$ **then**
11: $\quad\quad\quad P_b \leftarrow$ Randomly select a parent from archive $\mathcal{A}_i$
12: $\quad\quad$ **else**
13: $\quad\quad\quad P_b \leftarrow$ Randomly select a parent from neighbors $\mathcal{B}_i$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: $\quad y \leftarrow$ Generate offspring (P_a, P_b)
17: $\quad$ z $\leftarrow$ Update current best point (y,i)
18: $\quad$ Update solution and archive (y,i)
19: **until** The termination criterion is satisfied
20: **return** The non-dominated solutions picked from $\mathcal{P}$

1.4.3.1 Archive of useful infeasible solutions

The conventional CMOEA/D has the set of weight vectors $L = \{\lambda^1, \lambda^2, ..., \lambda^N\}$ and the solution set $P = \{x^1, x^2, ..., x^N\}$. For each weight vector, only one solution is maintained. After a feasible solution is founded for a weight vector, generated infeasible solutions are just discarded even if they have better objective values since CMOEA/D does not have a mechanism to maintain multiple solutions for each weight vector. To utilize these useful infeasible solutions as parents, the proposed CMOEA/D-DMA introduces the achieves $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_\mathcal{N}$for each weight vector.

as described at 6th line of **Algorithm 1**, each archive is initialized as the empty set. For each weight vector $\lambda^\mathbf{i}$, infeasible solutions showing better scalarizing function value than existing solution $\mathbf{x^i}$ are stored in its archive $\mathcal{A}_i$. **Figure 1.3** shows an example. For weight vector $\lambda^\mathbf{4}$, the objective area achieving better Tchebycheff scalarizing function value than the existing $\mathbf{x^4}$ is filled with red color. Two infeasible solutions inside the area are stored in the archive $\mathcal{A}_4$ and utilized as parents in the directed mating.

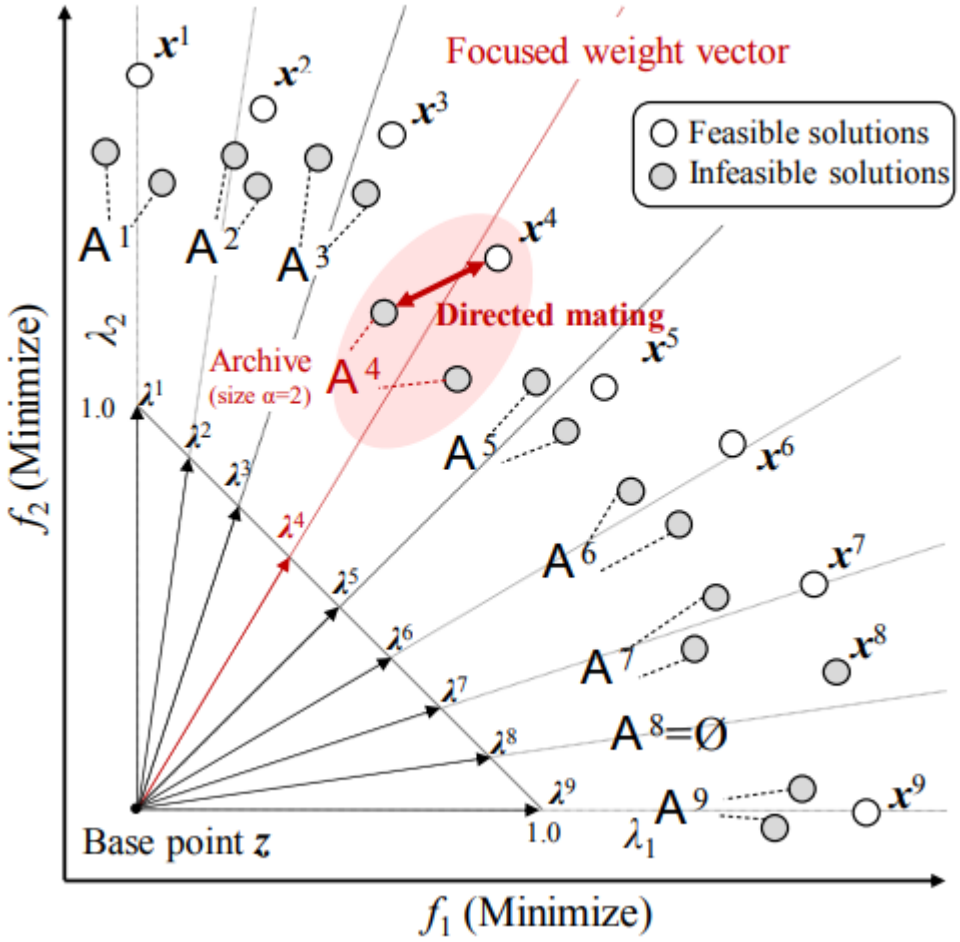


Figure 1.3 – Directed mating in the CMOEA/D-DMA [18].

1.4.3.2 Directed mating with archives

The proposed CMOEA/D-DMA utilizes useful infeasible solutions having better scalarizing function values as parents and stores the in archive $\mathcal{A}_i$ for each weight $\lambda^i (i = 1, 2, ..., N)$. To generate one offspring, CMOEA/D-DM focuses on a weight $\mathcal{A}_i$ and selects its paired solution x^i as the first parent p_a (9th line of **Algorithm 1**). If the first parent p_a is feasible and its archive is not empty ($|\mathcal{A}_i| > 0$), the second parent p_b is randomly selected from the archive $\mathcal{A}_i$ (11th line of **Algorithm 1**). This case is the directed mating shown in **Figure 1.3**. Otherwise, the second parent p_b is randomly selected from T-neighbor solutions of the focused $\mathcal{A}_i$ (13th line of Algorithm1). This case is the conventional local mating also used in CMOEA/D shown in **Figure 1.2**. Then, the crossover and the mutation operators are applied to p_a and p_b , an offspring y is obtained (15th line of **Algorithm 1**).

1.4.3.3 Update of Solutions and Archives

The proposed CMOEA/D-DMA tries to replace existing solutions of T-neighbors of the focused λ^i with the generated offspring **y** and update their archives. The pseudo-code of this procedure is shown in **Algorithm 2**, and it is called from 17th line of **Algorithm 1**. For solution update, if any of the following conditions are true, an existing x^j is replaced with y.

1. x^j and y are feasible, and $g(y|\lambda^j)$ is better than $g(x^j|\lambda^j)$.
2. x^j is infeasible, and y is feasible.

3. x^j and y are infeasible, and y dominates x^j on constraint violation values.

4. x^j and y are infeasible, y and x^j are non-dominated on constraint violation values, and $g(y|\lambda^j)$ is better than $g(x^j|\lambda^j)$.

For archive update, y is added to each archive $\mathcal{A}_j$ if an existing x^j is feasible, y is infeasible and $g(y|\lambda^j)$ is better than $g(x^j|\lambda^j)$. The maximum archive size is limited by a parameter α. When $|\mathcal{A}_j|$ exceeds α after adding y, the most inferior solution in $\mathcal{A}_j$ according to the proposed replacement criterion is eliminated. If solution x^j and offspring y are feasible and x^j is replaced with y, solutions in $\mathcal{A}_j$ having a worse scalarized function value than y are eliminated. All solutions maintained in $\mathcal{A}_i$ are infeasible, however, they have better scalarizing function values than x^i . Since there is a possibility that they would have useful variable information to improve the convergence of feasible solutions toward the Pareto front.

Algorithm 2 Update of Solution and Archive(From [19])

Input: offspring y, the focused index i, the archive size α

for each $j \in \mathcal{B}_i$ **do**
  if x^j and y are feasible, and $g(y|\lambda^j)$ is better than $g(x^j|\lambda^j)$ **then**
    $x^j \leftarrow y$
    $\mathcal{A}_j \leftarrow \{a \in \mathcal{A}_j \| g(a|\lambda^j)$ is better than $g(x^j|\lambda^j)\}$
  else if x^j is feasible and y is infeasible, and $g(y|\lambda^j)$ is better than $g(x^j|\lambda^j)$ **then**
    $\mathcal{A}_j \leftarrow \mathcal{A}_j \cup y$
    if $|\mathcal{A}_j| > \alpha$ **then**
      Remove the worst solution from $\mathcal{A}_j$
    end if
  else if x^j is infeasible and y is feasible **then**
    $x^j \leftarrow y$
  else if x^j and y are infeasible **then**
    if y dominates x^j on constraint violations **then**
      $x^j \leftarrow y$
    else if y and x^j are non-dominated on violations, and $g(y|\lambda^j)$ is better than $g(x^j|\lambda^j)$ **then**
      $x^j \leftarrow y$
    end if
  end if
end for

1.4.3.4 Update of Base Point

To calculate scalarizing function values g of **Eq 1.4.1**, the base point z to decompose the objective space is needed. Although the base point z should be the ideal point z^* which is the minimum objective vector of the true Pareto front, z^* is generally unknown. Therefore, the best objective vector found during the search is generally employed as the base point z.

In the proposed CMOEA/D-DMA, the base point z is updated by the best objective value among the new offspring y, the population $\mathcal{P}$, the archives $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_\mathcal{N}$ as follows:

$$z_i = \min_{x \in \{\mathcal{P} \cup \mathcal{A}_1 \cup ... \cup \mathcal{A}_\mathcal{N} \cup y\}} f_i(x) \qquad (i = 1, 2, ..., m)$$

Since the archives involves infeasible solutions, the objective values of the obtained base point z may be better than the ideal point z^* of the true Pareto front. When the distance between the obtained z and the true ideal point z^* is large, the approximation granularity of the Pareto front with the set of weight vectors will be low. However, since archived infeasible solutions are evolved to minimize constraint violation values, the distance between z and z^* is shorten during the search. The overestimated base point z in the proposed method works to improve the spread of the solutions in the objective space and becomes a clue of the search since the true ideal point is generally unknown during the search.

Conclusion

In this chapter, we have presented some definitions related to constrained multi-objective optimization problems (CMOPs) and the techniques to handle constraints in MOEAs. We have also introduced CMOEA/D-DMA that we aim to implement. CMOEA based on the decomposition technique and the directed mating tries to improve the search performance by utilizing the infeasible solutions. However, this last may suffer from the computational time, so we will try to parallelize it. But before that we need to introduce the parallel computing and its different techniques in the next chapter.

Chapter 2

Parallel Computing

Introduction

Evolutionary algorithms (EAs) have attracted attention as a promising way to solve multi-objective optimization problems (MOPs). Since objectives are usually conflicting each other, the goal of multiobjective optimization is to approximate the Pareto front, the optimal trade-off among objectives, with a set of solutions. To solve MOPs with constraints a constraint-handling techniques are needed to be employed. But since most of constraint-handling methods prefer feasible solutions than infeasible ones, and generated infeasible solutions are just discarded after feasible solutions are founded, a decomposition based algorithm that utilize infeasible solutions has been developed which is CMOEA/D-DMA. This last by utilizing infeasible solutions having better scalarizing function than feasible ones improved the search performance , however, the fitness evaluation time is an important factor. So we aim to use the parallelization concept and try to run the algorithm in reasonable time.

In this chapter we will introduce the parallel computing concept, its various techniques and we will talk about multicore processing, to determine by the end of this chapter which technique are most suitable for use .

2.1 Definition

Parallel computing is a type of computing architecture in which several processors execute or process an application or computation simultaneously. Parallel computing helps in performing large computations by dividing the workload between more than one processor, all of which work through the computation at the same time. Parallel computing is also known as parallel processing [29].

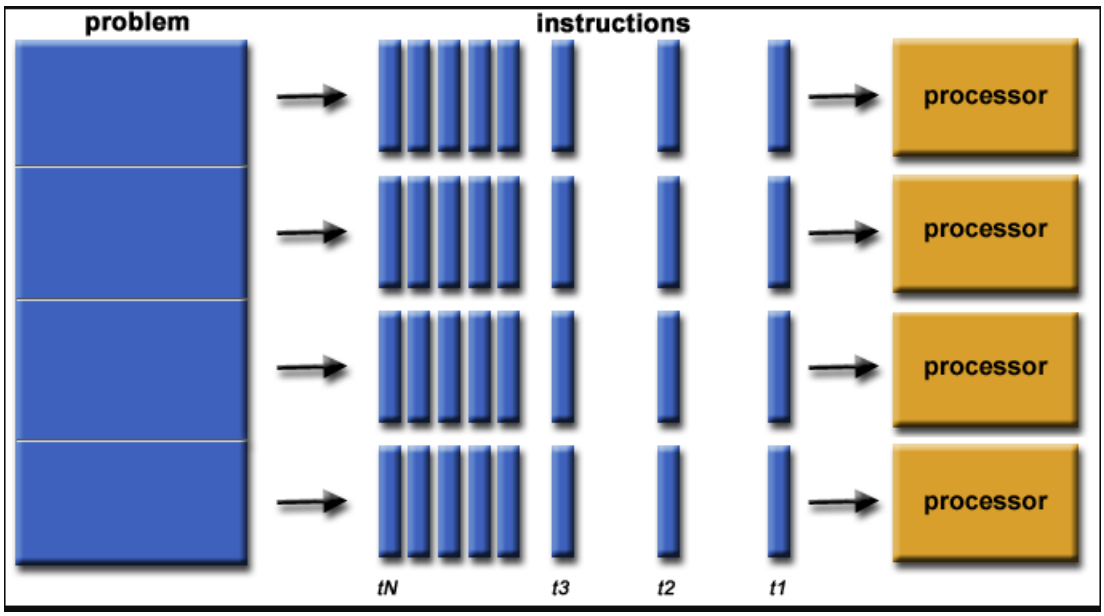


Figure 2.1 – Parallel Computing (From [1]).

Advantages of Parallel Computing over Serial Computing are as follows:

1. It saves time and money as many resources working together will reduce time and cut potential costs.
2. It can be impractical to solve larger problems on Serial Computing.
3. It can take advantage of non-local resources when the local resources are finite.
4. Serial Computing 'wastes' the potential computing power, thus Parallel Computing makes better work of hardware.

2.2 Why parallel computing?

Those are reasons why we need parallel computing [8].

- The whole real world runs in dynamic nature i.e. many things happen at a certain time but at different places concurrently. This data is extensively huge to manage.
- Real world data needs more dynamic simulation and modeling, and for achieving the same, parallel computing is the key.
- Parallel computing provides concurrency and saves time and money.
- Ensures the effective utilization of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of hardware was used and the rest rendered idle.

2.3 Types of Parallelism

we can identify 3 main types [8]:

1. **Bit-level parallelism:** It is the form of parallel computing which is based on the increasing processor's size. It reduces the number of instructions that the system must execute in order to perform a task on large-sized data.
   Example: Consider a scenario where an 8-bit processor must compute the sum of two 16-bit integers. It must first sum up the 8 lower-order bits, then add the 8 higher-order bits, thus requiring two instructions to perform the operation. A 16-bit processor can perform the operation with just one instruction.

2. **Instruction-level parallelism:** A processor can only address less than one instruction for each clock cycle phase. These instructions can be re-ordered and grouped which are later on executed concurrently without affecting the result of the program. This is called instruction-level parallelism.

3. **Task Parallelism:** Task parallelism employs the decomposition of a task into subtasks and then allocating each of the subtasks for execution. The processors perform execution of sub tasks concurrently.

2.4 Parallel Computing Applications

Historically, parallel computing has been considered to be "the high end of computing", and has been applied in many fields

- Data bases and Data mining.
- Real time simulation of systems.
- Science and Engineering.
- Advanced graphics, augmented reality and virtual reality.

2.5 Concepts and Terminology

These are the most important terminologies in parallel computing.

2.5.1 Parallel Computers

Parallel computers provide great amounts of computing power, but they do so at the cost of increased difficulty in programming and using them. Certainly, a uni-processor that was fast enough would be simpler to use [12]. Virtually all stand-alone computers today are parallel from a hardware perspective:

- Multiple functional units (floating point, integer, GPU, etc.),
- Multiple execution units / cores,
- Multiple hardware threads.

Networks connect multiple stand-alone computers (nodes) to create larger parallel computer clusters, where each computer node is a multi-processor parallel computer in itself, Multiple compute nodes are networked together with an infinite-Band network and special purpose nodes, also multi-processor.

2.5.2 Von Neumann Computer Architecture

Named after the Hungarian mathematician/genius John von Neumann(link wikipedia) who first authored the general requirements for an electronic computer in his 1945 papers. Also known as "stored-program computer" - both program instructions and data are kept in electronic memory. Differs from earlier computers which were programmed through "hard wiring". Since then, virtually all computers have followed this basic design:

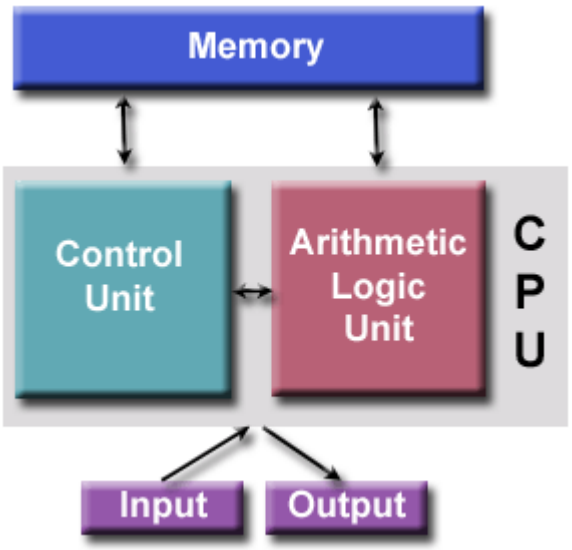


Figure 2.2 – von Neumann Architecture (From [1]).

Where its basic components comprised of four main components: Memory, Control Unit, Arithmetic Logic Unit, and Input/Output.

2.5.3 Instruction Stream and Data Stream

The term 'stream' refers to a sequence or flow of either instructions or data operated on by the computer. In the complete cycle of instruction execution, a flow of instructions from main memory to the CPU is established. This flow of instructions is called an instruction stream. In the same time, there is a flow of operands between processor and memory bidirectionally [1]. This flow of operands is called a data stream.

2.5.4 Flynn's Classification

There are different ways to classify parallel computers for more explanation(link).One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy (classification).
Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of Instruction Stream and Data Stream. Each of these dimensions can have only one of two possible states: Single or Multiple.The matrix below defines the 4 possible classifications according to Flynn:

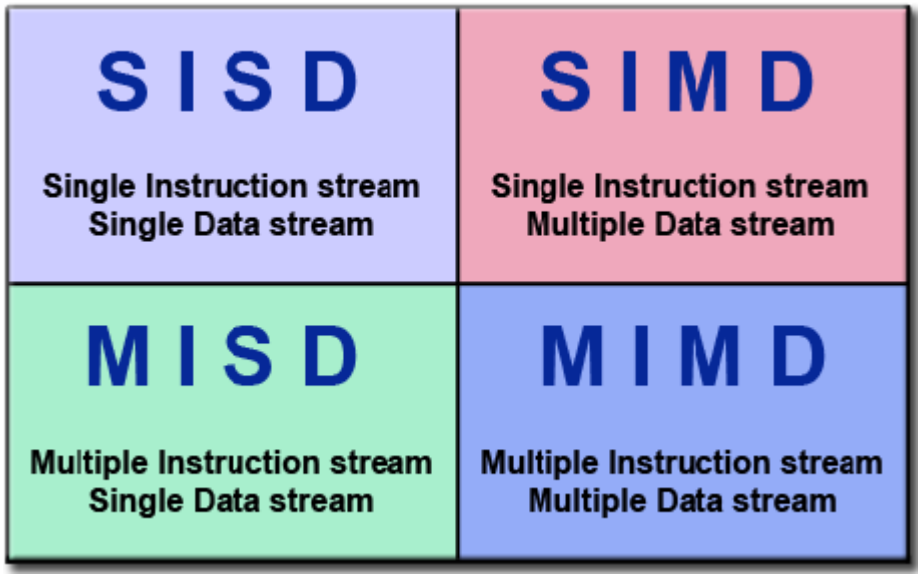


Figure 2.3 – Flynn's classification (From [1]).

Single Instruction and Single Data stream (SISD)

In this organization, the sequential execution of instructions is performed by one CPU containing a single processing element (PE). Therefore, SISD machines are conventional serial computers that process only one stream of instructions and one stream of data.

Single Instruction and Multiple Data stream (SIMD)

In this organization, multiple processing elements work under the control of a single control unit. It has one instruction and multiple data stream. All the processing elements of this organization receive the same instruction broadcast from the CU. Main memory can also be divided into modules for generating multiple data streams acting as a distributed memory. Therefore, all the processing elements simultaneously execute the same instruction and are said to be 'lock-stepped' together. Each processor takes the data from its own memory and hence it has on distinct data streams. (Some systems also provide a shared global memory for communications.) Every processor must be allowed to complete its instruction before the next instruction is taken for execution. Thus, the execution of instructions is synchronous.

Multiple Instruction and Single Data stream (MISD)

In this organization, multiple processing elements are organized under the control of multiple control units. Where each control unit is handling one instruction stream and processed through its corresponding processing element. But each processing element is processing only a single data stream at a time. Therefore, for handling multiple instruction streams and single data stream, multiple control units and multiple processing elements are organized in this classification. All processing elements are interacting with the common shared memory for the organization of single data stream [1].
This classification can be very helpful for specialized applications. But MISD organization is not popular in commercial machines as the concept of single data streams executing on multiple processors is rarely applied [1]. For example, Real-time computers need to be faulty where several processors execute the same data for producing the redundant data. All these redundant data are compared as results which should be same; otherwise faulty unit is replaced. Thus MISD machines can be applied to fault tolerant real time computers.

Multiple Instruction and Multiple Data stream (MIMD)

In this organization, multiple processing elements and multiple control units are organized as in MISD. But the difference is that now in this organization multiple instruction streams operate on multiple data streams. Therefore, for handling multiple instruction streams, multiple control units and multiple processing elements are organized such that multiple processing elements are handling multiple data streams from the Main memory. The processors work on their own data with their own instructions. Tasks executed by different processors can start or finish at different times. They are not lock-stepped, as in SIMD computers, but run asynchronously. This classification actually recognizes the parallel computer. That means in the real sense MIMD organization is said to be a Parallel computer. All multiprocessor systems fall under this classification [1].

2.5.5 Limits and Costs of Parallel Programming

Suppose we wish to compare a parallel and sequential computer built from the same units, to argue that a new parallel algorithm is many times faster than the best sequential algorithm (the same reasoning applies to logic gates on an integrated circuit). Given N parallel units and an algorithm that run times faster on sufficiently large inputs, one can simulate the parallel system on the sequential system by dividing its time between computational slices. Since this simulation is roughly N times slower, it runs M/N times faster than the original sequential algorithm. If this original sequential algorithm was the fastest possible, we have $M \leq N$. In other words, a fair comparison should not demonstrate a parallel speedup that exceeds the number of processors. a super linear speedup can indicate an inferior sequential algorithm or the availability of a larger amount of memory to N processors. The bound is reasonably tight in practice for small N and can be violated slightly because N CPUs include more CPU cache, but such violations alone do not justify parallel algorithms [1].

2.6 Parallel Computer Memory Architectures

Parallel architectures are evolving quickly. Nowadays, the classification of Flynn is not sufficient to describe the different types of parallel architectures and their characteristics.So we will describe other sufficient parallel computer memory architectures that been proposed:

2.6.1 Shared Memory

General Characteristics:

Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.Multiple processors can operate independently but share the same memory resources. Changes in a memory location effected by one processor are visible to all other processors. Historically, shared memory machines have been classified as UMA and NUMA, based upon memory access times.

Uniform Memory Access (UMA):

Most commonly represented today by Symmetric Multiprocessor (SMP) machines, they have Identical processors and equal access and access times to memory. Sometimes called CC-UMA - Cache Coherent UMA. Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update. Cache coherency is accomplished at the hardware level [1].

Non-Uniform Memory Access (NUMA):

Often made by physically linking two or more SMPs , where one SMP can directly access memory of another SMP. Not all processors have equal access time to all memories. Memory access across link is slower and cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA [1].

Advantages:

Firstly the global address space provides a user-friendly programming perspective to memory. Secondly, data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs.

Disadvantages:

Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management. Then, programmer responsibility for synchronization constructs that ensure "correct" access of global memory.

2.6.2 Distributed Memory

General Characteristics:

Like shared memory systems, distributed memory systems vary widely but share a common characteristic. Distributed memory systems require a communication network to connect inter-processor memory. Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors. Because each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply. When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility. The network "fabric" used for data transfer varies widely, though it can be as simple as Ethernet [1].

Advantages:

Memory is scalable with the number of processors. Increase the number of processors and the size of memory increases proportionately. Then, each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain global cache coherency. Finally, cost effectiveness: can use commodity, off-the-shelf processors and networking.

Disadvantages:

First, the programmer is responsible for many of the details associated with data communication between processors. Second, it may be difficult to map existing data structures, based on global memory, to this memory organization. Then non-uniform memory access times data residing on a remote node takes longer to access than node local data.

2.6.3 Hybrid Distributed-Shared Memory

The largest and fastest computers in the world today employ both shared and distributed memory architectures.The shared memory component can be a shared memory machine and/or graphics processing units (GPU).The distributed memory component is the networking of multiple shared memory/GPU machines, which know only about their own memory - not the memory on another machine. Therefore, network communications are required to move data from one machine to another.Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future [1].

Advantages and Disadvantages:

Whatever is common to both shared and distributed memory architectures. Then, Increased scalability is an important advantage. Finally, Increased programming complexity is a major disadvantage.

2.7 Parallel Programming Models

Parallel Programming Models exist as an abstraction above hardware and memory architectures.he standard parallel architectures support a variety of decomposition strategies, such as decomposition by task (task parallelism) and decomposition by data (data parallelism). Our introductory treatment will concentrate on data parallelism be-cause it represents the most common strategy for scientific programs on parallel machines. In data parallelism, the application is decomposed by subdividing the data space over which it operates and assigning different processors to the work as associated with different data subspace. Typically this strategy involves some data sharing at the boundaries, and the programmer is responsible for ensuring that this data sharing is handled correctly is, data computed by one processor and used by another are correctly synchronized.Once a specific decomposition strategy is chosen, it must be implemented. Here,the programmer must choose the programming model to use [1]. The two most common models are The shared-memory model, in which it is assumed that all data structures areal located in a common space that is accessible from every processor, and the message-passing model, in which each processor (or process) is assumed to have its own private data space, and data must be explicitly moved between spaces as needed.
In the message-passing model, data structures are distributed across the processor memories; if a processor needs to use a data item that is not stored locally, the processor that owns that data item must explicitly "send" it to the requesting processor. The latter must execute an explicit "receive" operation, which is synchronized with the send, before it can use the communicated data item.

2.7.1 Shared Memory Model

In this programming model, processes/tasks share a common address space, which they read and write to asynchronously. Various mechanisms such as locks / semaphores are used to control access to the shared memory, resolve contentions and to prevent race conditions and deadlocks. This is perhaps the simplest parallel programming model. An advantage of this model from the programmer's point of view is that the notion of data "ownership" is lacking, so there is no need to specify explicitly the communication of data between tasks. All processes see and have equal access to shared memory. Program development can often be simplified with an important disadvantage in terms of performance is that it becomes more difficult to understand and manage data locality [1].

2.7.2 Threads Model

This programming model is a type of shared memory programming.In the threads model of parallel programming, a single "heavy weight" process can have multiple "light weight", concurrent execution paths. For applications where the workload depends on application input that can vary widely, delay the decision about the number of threads to employ until run-time when the input sizes can be examined. Examples of workload input parameters that affect the thread count include things like matrix size, database size, image/video size and resolution, depth/breadth/bushiness of tree based structures, and size of list based structures. Similarly, for applications designed to run on systems where the processor count can vary widely, defer the number of threads to employ decision till application run-time when the machine size can be examined.

For applications where the amount of work is unpredictable from the input data, consider using a calibration step to understand the workload and system characteristics to aid in choosing an appropriate number of threads. If the calibration step is expensive, the calibration results can be made persistent by storing the results in a permanent place like the file system. Avoid creating more threads than the number of processors on the system, when all the threads can be active simultaneously; this situation causes the operating system to multiplex the processors and typically yields sub-optimal performance.

When developing a library as opposed to an entire application, provide a mechanism whereby the user of the library can conveniently select the number of threads used by the library, because it is possible that the user has higher-level parallelism that renders the parallelism in the library unnecessary or even disruptive.

Finally, for OpenMP, use the *num_threads* clause on parallel regions to control the number of threads employed and use the if clause on parallel regions to decide whether to employ multiple threads at all. The *omp_set_num_threads* function can also be used but it is not recommended except in specialized well-understood situations because its affect is global and persists even after the current function ends, possibly affecting parents in the call tree. The *num_threads* clause is local in its effect and so does not impact the calling environment [1].

2.7.3 Distributed Memory / Message Passing Model

The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum, which has over 40 participating organizations, including vendors, researchers, software library developers, and users. The goal of the Message Passing Interface is to establish a portable, efficient, and flexible standard for message passing that will be widely used for writing message passing programs. As such, MPI is the first standardized, vendor independent, message passing library. The advantages of developing message passing software using MPI closely match the design goals of portability, efficiency, and flexibility. MPI is not an IEEE or ISO standard, but has in fact, become the "industry standard" for writing message passing programs on HPC platforms [1]. Message passing is an approach that makes the exchange of data cooperative. Data must both be explicitly sent and received. An advantage is that any change in the receiver's memory is made with the receiver's participation [1]. One-sided operations between parallel processes include remote memory reads and write. An advantage is that data can be accessed without waiting for another process [1]. MPI is a specification for the developers and users of message passing libraries. By itself, it is NOT a library but rather the specification of what such a library should be. MPI primarily addresses the message passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process. Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs. The interface

attempts to be Practical, Portable, Efficient and Flexible [1]

2.7.4 Data Parallel Model

May also be referred to as the Partitioned Global Address Space (PGAS) model. The data parallel model demonstrates the characteristics which are :

- Address space is treated globally,
- Most of the parallel work focuses on performing operations on a data set. The data set is typically organized into a common structure, such as an array or cube,
- A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure,
- Tasks perform the same operation on their partition of work, for example, "add 4 to every array element".

On shared memory architectures, all tasks may have access to the data structure through global memory.On distributed memory architectures, the global data structure can be split up logically and/or physically across tasks [1].

2.7.5 Hybrid Model

A hybrid model combines more than one of the previously described programming models. Currently, a common example of a hybrid model is the combination of the message passing model (MPI) with the threads model (OpenMP). Threads perform computationally intensive kernels using local, on-node data and Communications between processes on different nodes occurs over the network using MPI. This hybrid model lends itself well to the most popular hardware environment of clustered multi/many-core machines. Another similar and increasingly popular example of a hybrid model is using MPI with CPU-GPU (Graphics Processing Unit) programming.
MPI tasks run on CPUs using local memory and communicating with each other over a network, computationally intensive kernels are off-loaded to GPUs on-nod. A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure.Data exchange between node-local memory and GPUs uses CUDA (or something equivalent) [1].

2.7.6 SPMD and MPMD

Single Program Multiple Data (SPMD):

SPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models. SINGLE PROGRAM: All tasks execute their copy of the same program simultaneously. This program can be threads, message passing, data parallel or hybrid. SPMD programs usually have the necessary logic programmed into them to allow different tasks to branch or conditionally execute only those parts of the program they are designed to execute. That is, tasks do not necessarily have to execute the entire program perhaps only a portion of it. The SPMD model, using message passing or hybrid programming, is probably the most commonly used parallel programming model for multi-node clusters [1].

Multiple Program Multiple Data (MPMD):

Like SPMD, MPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models.MULTIPLE PROGRAM: Tasks may execute different programs simultaneously. The programs can be threads, message passing, data parallel or hybrid. MULTIPLE DATA: All tasks may use different data. MPMD applications are not as common as SPMD applications, but may be better suited for certain types of problems, particularly those that lend themselves better to functional decomposition than domain decomposition [1].

Conclusion

as a conclusion to this chapter, we introduced some of the important concepts and architectures that related to the parallel programming . We began with a discussion on parallel computing - what it is why and how it's used, followed by a discussion on concepts ,terminologies and architers associated with parallel computing.in the next chapter, firstly we will implement the sequential CMOEA/D-DMA that described in the first chapter .then we will use one of the techniques mentioned in the second chapter to parallelize it.

Part II

PARALLELIZATION OF CMOEA/D-DMA

Chapter 3

Analysis & Design

Chapter 3

Analysis & Design

Introduction

After taking in consideration what we have learned in the previous chapters, we will move to the production process.

This chapter is sectioned into five parts. In the first section we will describe the problem of this project. In the second section we will provide an explanation of the project cycle. The third section will be used to present the global design of the application. In the last two sections, a detailed design of the both versions CMOEA/D-DMA and PCMOEA/D-DMA are given, where each section contains analysis and design of each version.

3.1 Problem description

MOEAs have attracted attention as a promised way to solve MOPs. However, the problem that faced this type of algorithms when the MOPs is at large scale, is converging to the optimal solutions slowly. This what guided to the occurrence of the decomposition techniques, where the MOPs decomposed into a number of sub-problems. But since we are looking for optimality of both results and time, we will choose a based decomposition algorithm and try to rerun it in a parallel way. In this project we desire to

1. Firstly, implement a decomposition based algorithm and apply it to a MOPs, which will help us to take knowledge of this kind of algorithms and to understand how to make the comparison.

2. Secondly, propose a parallel version (i.e., PCMOEA/D-DMA) and compare between the results of the two implementations (sequential and parallel) regardless of the techniques used to make this algorithm in parallel.

So, in the end what we aim for behind this project is not just to implement any MOEA and parallelize it, but is a good implementation of the sequential version of the chosen algorithm (CMOEA/D-DMA). And a good design and implementation of the parallel version (PCMOEA/D-DMA) to achieve a well-extended pareto front in acceptable time. Figure 3.1 shows a simple description to the problem

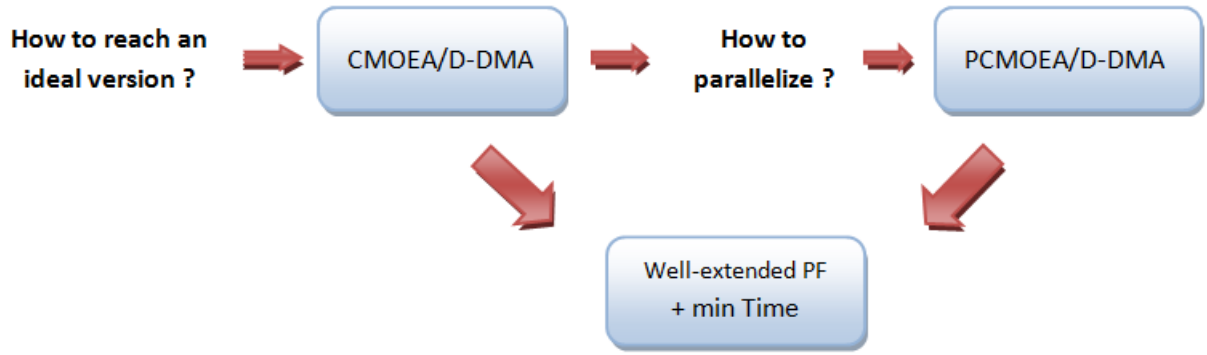


Figure 3.1 – Project Problem.

3.2 project cycle

After taking knowledge of the problem in the previous section, now the necessary steps must be put in place to handle the problem. Our project will be divided into

- Implementation of the sequential version of CMOEA/D-DMA.
- Parallelization of CMOEA/D-DMA to obtain the parallel CMOEA/D-DMA (PCMOEA/D-DMA).
- Comparison step: where we will compare CMOEA/D-DMA and PCMOEA/D-DMA results, CMOEA/D-DMA and other CMOEA (CMOEA/D-DMA [19], CMOEA/D [11], TNSDM [21], CNSGA-III [6]) results.

All these steps are described in Figure 3.2.

Figure 3.2 – Project Cycle.

3.3 Application Global Design

Firstly, we start by presenting a conceptual model in Figure 3.3 which shows the functional architecture of the whole system.

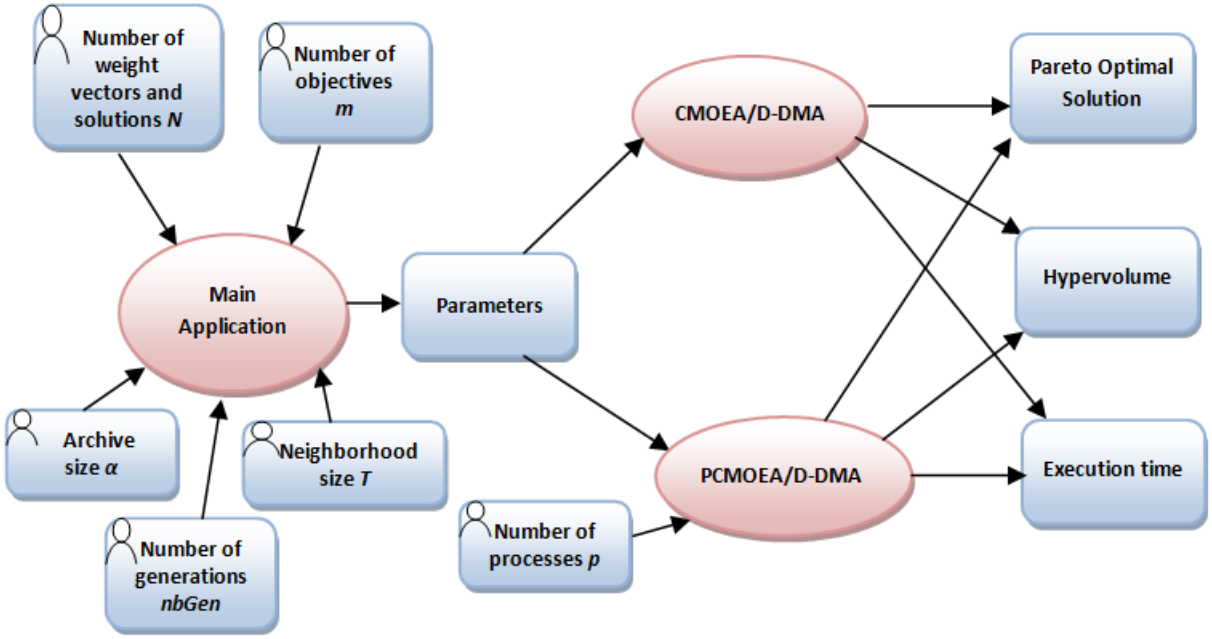


Figure 3.3 – Global Design of the Application.

Where the blue boxes in the left side represent inputs the user allowed to identify, the blue boxes in the right side represent our system outputs (outputs of the algorithms + comparison factors) and the red spheres represent functionalities in our system. Inputs are:

- N is the number of weight vectors, solutions and also the initial population size.
- T is the neighbourhood size.
- $nbGen$ is the number of generations.
- m is the number of objectives (it can be [2,4,6,8]).
- α is the archive size.

The sysetm will use m and N to create the initial population then that will served to one of the versions to obtain the non-dominated sort and use its pareto front to evalute both time and results quality of objective functions (hypervolume HV).

3.4 Hypervolume HV

The chosen metric to evaluate the algorithms performance is HV [35], because it was the most preferred and used metric in the last years according to the statistics below Figure 3.4 (EMO: Evolutionary Multi-Criterion Optimization).

Hypervolume is an unary metric (The metric is said to be unary if it receives as parameter only one approximation set A to be evaluated.) which measures m-dimensional volume covered by the obtained POS and a reference point r in the objective space. HV is a representative metric to simultaneously evaluate the convergence and the diversity of the obtained solutions. Good convergence toward the true Pareto front contributes to increasing the value of HV, and good diversity also contributes to increasing HV. Obtained solutions showing a higher value of HV can be considered as a better set of solutions in term of both the convergence and the diversity toward the true pareto front [20].

Ranking	Citations	Metrics	Classification	
			Aspects	Sets
1°	91	Hypervolume (HV)	- *Accuracy* - *Diversity*	Unary
2°	26	Generational distance (GD)	- *Accuracy*	Unary
3°	23	Epsilon family (ϵ)	all	Binary
4°	17	Inverted generational distance (IGD)	- *Accuracy* - *Diversity*	Unary
	17	Spread: Delta indicator (Δ)	- *Diversity*	Unary
	17	Two set coverage (C)	all	Binary
5°	9	ONVG	- *Cardinality*	Unary
	9	R-metric	all	Binary
6°	8	Convergence measure (Υ)	- *Accuracy*	Unary
7°	6	Convergence metric (CM)	-*Accuracy*	Unary
	6	D_1R	- *Accuracy* - *Diversity*	Unary
	6	Spacing (Sp)	- *Diversity*	Unary
8°	5	M_3^*metric	- *Diversity*	Unary
9°	4	M_1^* metric	- *Accuracy*	Unary
10°	3	Diversity metric (DM)	- *Diversity*	Unary
	3	Entropy metric	- *Diversity*	Unary
	3	Spread measure	- *Diversity*	Unary

Figure 3.4 – statics shown top ten of the most used metrics in EMO from 2005 to 2013 [26].

3.5 Sequential CMOEA/D-DMA

3.5.1 Analysis

Sequential CMOEA/D-DMA is the implementation of the algorithm as it is described in [19]. The sequential version as we described in the first chapter 1 is a based decomposition algorithm, but what makes this algorithm different from the rest of its class the fact that it chooses also infeasible solutions that have better scalarizing function than feasible ones as parents, rather than choosing just feasible solutions which improved the search performance. CMOEA/D-DMA is composed of a set of methods that will be explained below executed in a sequential way.

3.5.2 Global Design

In this part we will provide a global design of the sequential version of CMOEA/D-DMA, that shows the main steps that the algorithm passes through. We will also use a class diagram and a sequence diagram to explain the intern interactions between objects.

The algorithm applies the decomposition to CMOP after the user enters the parameters, then it creates the initial population which contains individuals. After that, CMOEA/D-DMA selects parents by choosing between the directed mating and the conventional one. In the next two steps the algorithm updates the base point and solutions and the archive, to obtain the population desired in the last step .
Figure 3.5 shows the global architecture of sequential CMOEA/D-DMA.

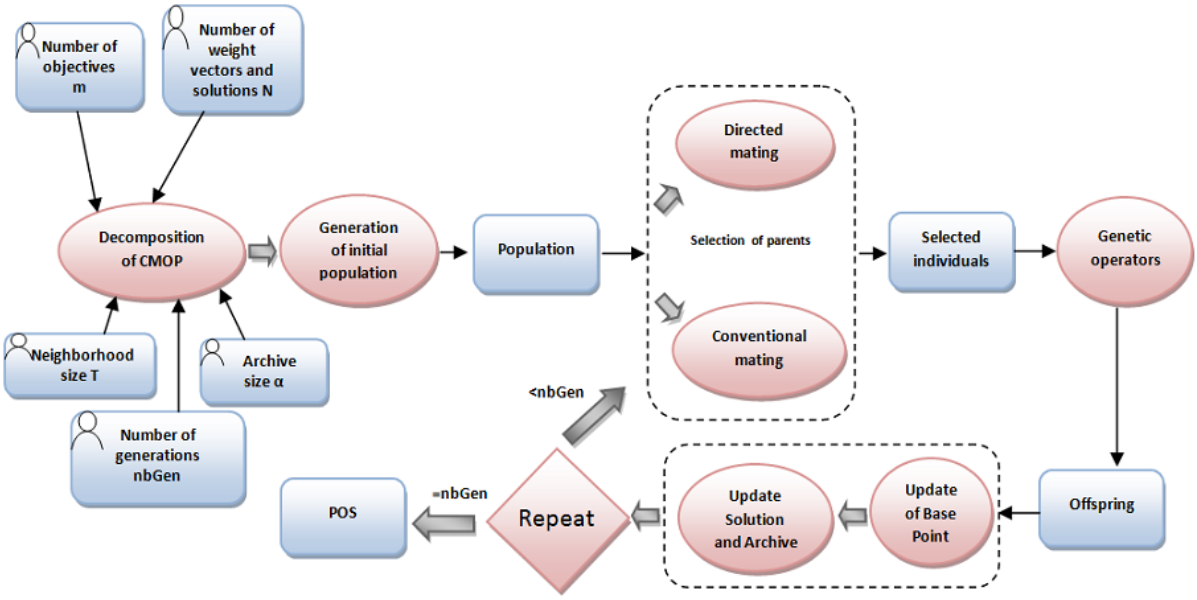


Figure 3.5 – Sequential CMOEA/D-DMA design.

Figure 3.6 shows the Classes diagram of the sequential version.

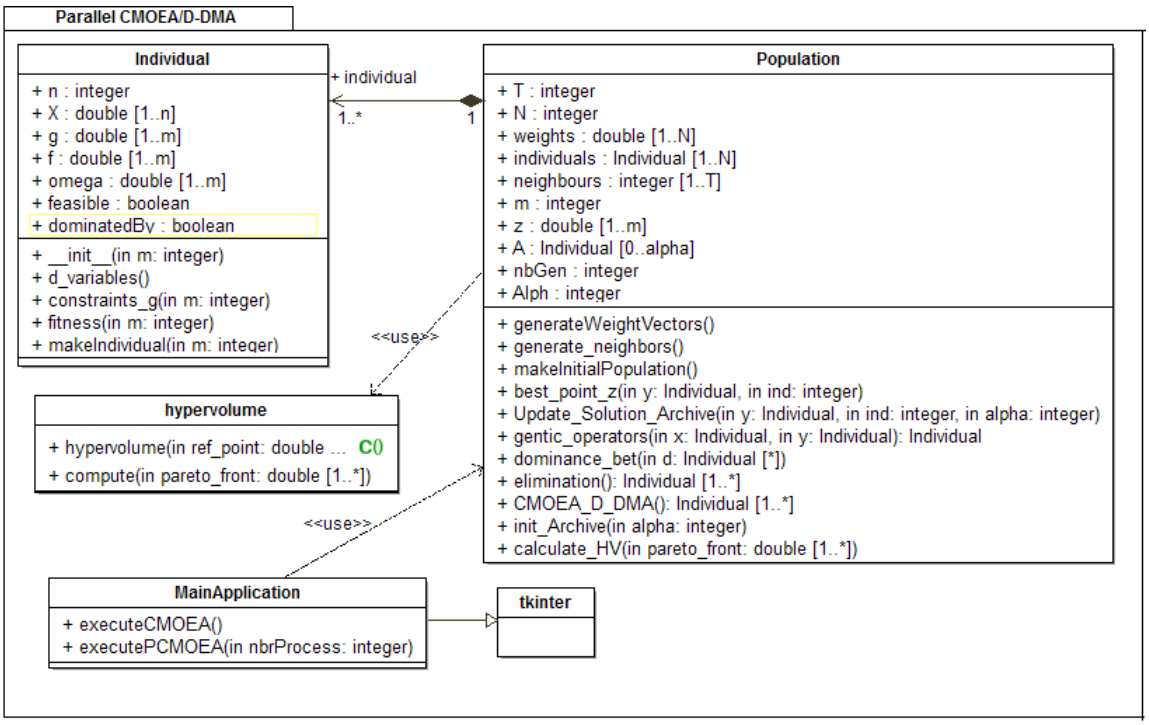


Figure 3.6 – Sequential CMOEA/D-DMA class diagram.

Figure 3.7 shows the sequence diagram of the sequential version.

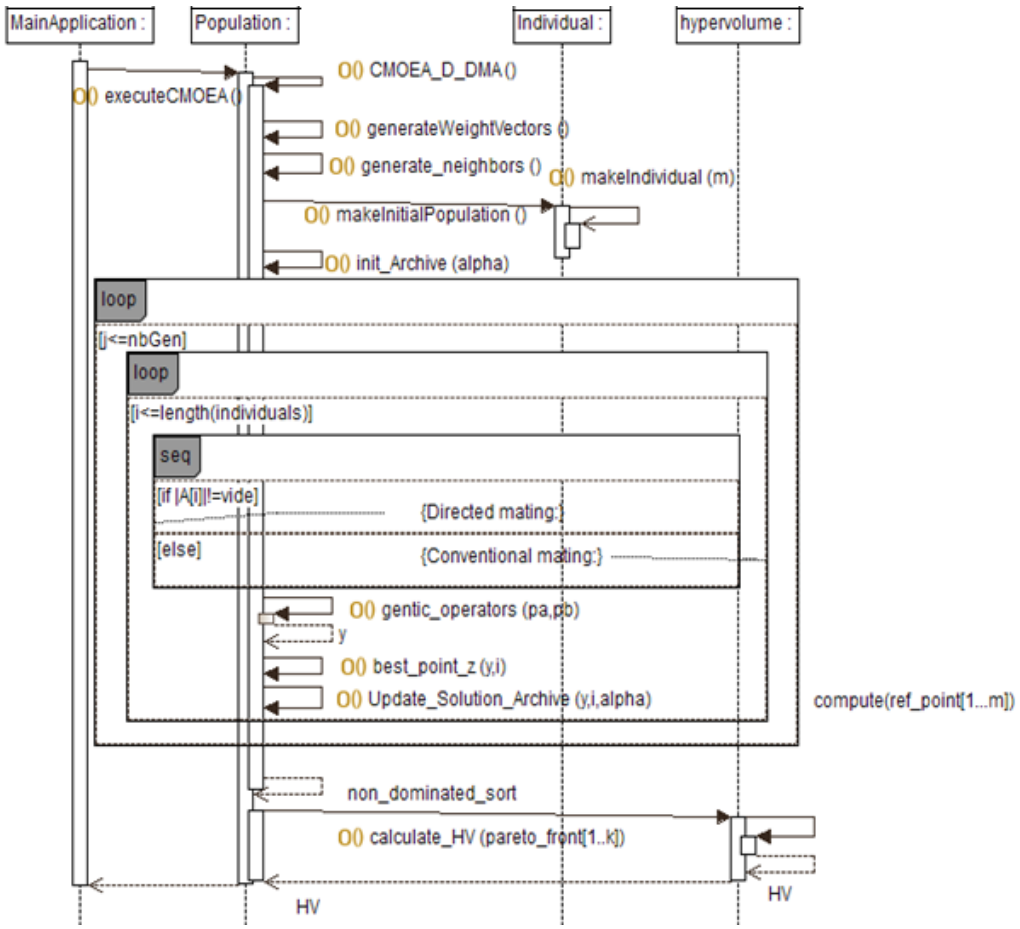


Figure 3.7 – Sequential CMOEA/D-DMA sequence diagram.

3.5.3 Detailed Design

In this detailed design, a detailed description will be discussed starting with exposing classes and detailing each attribute and method in each class.

Remark.
- *The symbol // means that what came after the symbol is a comment (explanation to that line or the line after)*
- *The symbol /* means that what came after the symbol is an application of a function that is related to a programming language or will be explained later.*

3.5.3.1 Individual class :

In this class the CMOP individuals will be created. An Individual object is the smallest unit in this design, where each individual is defined by a decision variable X represented by array with n elements, a set of objective functions f_i $(i = 1, 2, ..., m)$ and for each objective function a set of constraints g_j $(j = 1, 2, ..., m)$. Also, we add two other variables the first one is feasible which used to determine the feasibility of the individual. The second one is dominatedBy, it's used to decide if the individual is dominated by another one in the population or not.

- **init:**
  This method represent the constructor of class Individual, it's used to initialize variables. It takes as a parameter number of objectives.
- **d_variables**
  This method is used to generate the decision variables randomly.

- **constraints g**
  In this method the constraints created and they formulated according to a mathematical function mentioned in [19]. It takes as a parameter number of objectives.

- **fitness**
  This method creates the fitness of individual object. In our case,the fitness of individual is a mathematical formula defined in [19]. It takes as a parameter number of objectives.

- **make Individual**
  This method used to create individuals. It takes as a parameter number of objectives.

3.5.3.2 Population class :

CMOEA/D-DMA as we said before is a based decomposition algorithm specified in using the directed mating concept. Population class has the attributes we presented the inputs ones before in Figure 3.3. In this section, we will discuss each attribute of them and we will also detailed the methods that represented in the UML diagrams 3.6 and 3.7.
Firstly we will talk about attributes.

1. **Attributes**

   (a) N: number of weight vectors and solutions in the population (initial population size).

   (b) T: the neighbourhood size means the number of neighbours for each weight vector.

   (c) *nbGen*: number of generations which is stop condition.

   (d) m: is the number of objectives (he is only allowed to enter [2,4,6,8]) and also constraints number for each objective function for this problem.

   (e) α: the archive size.

   (f) A: the archive vector contains N vector of real values of size α.

   (g) z: vector of real values of size m.

   (h) $neighbours$: neighbourhood contains N vector of integer values of size T for each weight vector.

   (i) $weights$: this is the set λ that we mentioned in chapter 1, λ is a vector contains N vector of real values of size m.

   (j) $individuals$: represent the population, contains N individual.

2. **Methods**

   - **Generate weight vectors**
     This method developed initially in [34] .This algorithm decomposes a CMOP into a number of sub-problems with single objective, we can only use from 1 to 3 objectives.
     The algorithm 3 shows the detail of this function

Algorithm 3 Generate weight vectors

if $m = 2$ **then**// m is the number of objectives
  for $i = 1 : N$ **do**// N is the population size
    /* weight is a vector of size m each element in weight $\in [0, 1]$
    weight[1]$\leftarrow \frac{i}{N}$
    weight[2]$\leftarrow \frac{N-i}{N}$
    /* add the weight to the list of weight vectors weights
  end for
else
  if $m = 3$ **then**
    for $i = 1 : N$ **do**// N is the population size
      for $j = 1 : N$ **do**
        if $i + j \leq N$ **then**
          $k = N - i - j$
          /* define a vector weight of 3 elements
          weight[1] $\leftarrow \frac{i}{N}$
          weight[2] $= \leftarrow \frac{j}{N}$
          weigh[3] $= \leftarrow \frac{k}{N}$
          /* add the vector weight to the list of weight vectors weights
        end if
      end for
    end for
    /* reduce the number of weight vectors to fit the size of population
  end if
end if

- **Generate neighbours**
  This algorithm initializes for each weight vector a set of weight neighbours.it means that it computes the Euclidean distances between any two weight vectors and then work out the closest weight vectors to each weight vector.
  The algorithm 4 shows the detail of this function.

Algorithm 4 generate_neighbors

for $i = 1 : N$ **do** //
  for $j = 1 : N$ **do** //
    /* calculate the euclidean distance between the weight vector λ^i and the other λ^j $(j = 1, 2, ..., N)$
  end for
  /* find the T min distance indexes between the values of the euclidean distance of λ^i
  /* add the indexes vector to the set of neighbours
end for

- **Make initial population**
  In this method the initial population will be created. N individual will be add to the set individuals, each individual created based on the CMOP nature.
  Algorithm 5 shows the initialization of the population.

Algorithm 5 makeInitialPopulation

for $i = 1 : N$ **do**
  /* create object individual
  individual.makeIndividual
  /*add individual to the population (individuals)
end for

- **Calculate the best point z**
  This method calculate the best point or best objective function values vector in all the generations (minimum values in case of minimization) . The equation 1.4.3.4 is the mathematical formulation of the algorithm.
  Algorithm 6 describe the method which calculates the best point z in each generation.

Algorithm 6 best_point_z

Input: y is the individual that obtained after the application of the genetic operators (offspring), ind is the index of the focused weight vector λ^{ind}

for $j = 1 : m$ **do**
  //calculate the min (or max in case of maximization problem) value of the objective function j among the values of the fitness in the population and the archive and y.
  $z_i \leftarrow \min f_i(x)$ // x is an individual, where $x \in \{\mathcal{P} \cup \mathcal{A}_1 \cup ... \cup \mathcal{A}_\mathcal{N} \cup y\}$
end for

- **Calculate Tchebychef function**
  Tchebychef is the scalarizing function used in the paper [19], formulated by 1.1.
  Algorithm 7 the method to calculate the tchebycheff value.

Algorithm 7 Tchebychef function

Input: x is the individual that we aim to calculate its scalarizing function value , i is the index of the focused weight vector λ^i

Output: maximum a real variable

for $j = 1 : m$ **do**
  $maximum \leftarrow \max\ (|f_j(x) - z_j|.\lambda_j^i)$ // maximum is a real variable used only inside this algorithm
end for
return $maximum$

- **Update solution and archive**
  This algorithm is already explained in chapter 1, and the pseudo-code is shown in Algorithm 2.
- **Genetic operators**
  When we talk about genetic operators we mean selection, crossover and mutation. In selecting parents we said that we will be using the directed or the conventional mating. In this method we will only mention both crossover and mutation which have been described in [5].
  Algorithm 8 shows the detailed function.

Algorithm 8 genetic_operators

Input: x the first parent, y is the second parent

Output: offspring

/*apply crossover to x and y the result will be a vector q of size 2
$q \leftarrow$ crossover(x,y)
$child_1 \leftarrow q[1]$
$child_2 \leftarrow q[2]$
/*apply the mutation function to both $child_1$ and $hild_2$
$child_1 \leftarrow$ mutation($child_1$)
$child_2 \leftarrow$ mutation($child_2$)
/*choose randomly one of the children and stock it in the variable offspring
return offspring $\leftarrow child_i$

- **Elimination**
  This function does not belong to the Algorithm 1 as it is mentioned in [19]. But i is been added to remove the repeated individuals in the population.
  This function is detailed in Algorithm 9.

Algorithm 9 elimination

Input: the population set individuals **Output:** individuals after elimination

```
for i = 1 : N do //i is an index used to fetch inside the population
    for j = i + 1 : N do
        if individuals[i]=individuals[j] then
            /* remove individual[j] from the population individuals
        end if
    end for
end for
return the population set individuals
```

- **CMOEA/D-DMA**
  This is the method that represent the main function where the all the previous algorithm will be executed sequentially. The pseudo-code of this method is shown in Algorithm 1. The algorithm 10 is the same as Algorithm 1 with some modifications to suit the programming language.

Algorithm 10 CMOEA_D_DMA

```
/* generate weight vectors (application of algorithm 3)
/* generate neighbours (application of algorithm 4)
/* initial population (application of algorithm 5)
/* initialize the archive set with zeroes
/* initialize the z vector with zeroes
for i = 1 : nbGen do // number of generations desired
    for j = 1 : N do
        // selecting parents p_a and p_b
        p_a ← individuals[i]
        /*select p_b with directed mating or conventional mating
        // genetic operators
        y ← genetic_operators(p_a,p_b) //(application of algorithm 8)
        best_point_z(y,i) //(application of algorithm 1.4.3.4)
        update_solution_archive(y,i,α) //(application of algorithm 2)
    end for
end for
sort ← elimination() //(application of algorithm 9)
return non-dominated sort
```

3.6 Parallel CMOEA/D-DMA

3.6.1 Analysis

Parallel CMOEA/D-DMA (PCMOEA/D-DMA) is a CMOEA proposed in this thesis because we aim to get better results with less execution time. In this parallel version beside all the classes mentioned before, we add the class Parallel. In this class a set of methods have been used to parallelize CMOEA_D_DMA based on python processes.
The main idea is to divide the population into p sub-populations, each process will handle one of the sub-populations and execute CMOEA/D-DMA. After that based on the HV results one of the evaluated sub-population will be chosen. That means we will use SIMD technique by duplicating the algorithm, where each sub-population has its own weight vectors set .

3.6.2 Global Design

In the global architecture of PCMOEA/D-DMA application a set of components will be identified, we will also identify and discuss the relation between them in a detailed way. Figure 3.8 shows the global architecture of PCMOEA/D-DMA.

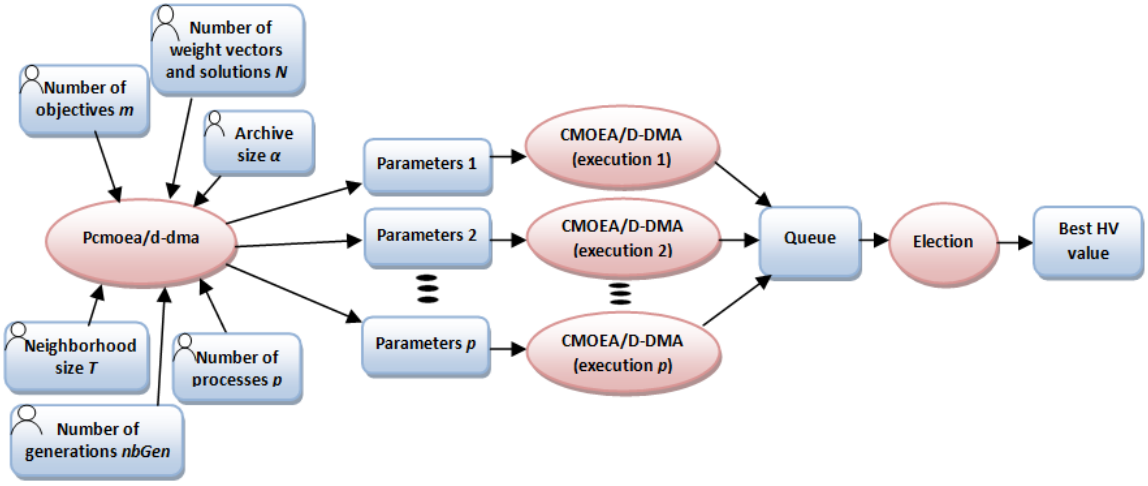


Figure 3.8 – Parallel CMOEA/D-DMA design.

In this part of design we use several UML diagrams to explain classes created in this parallel version and also to represent relationship between their components.

Figure 3.9 shows the classes diagram of this parallel version.

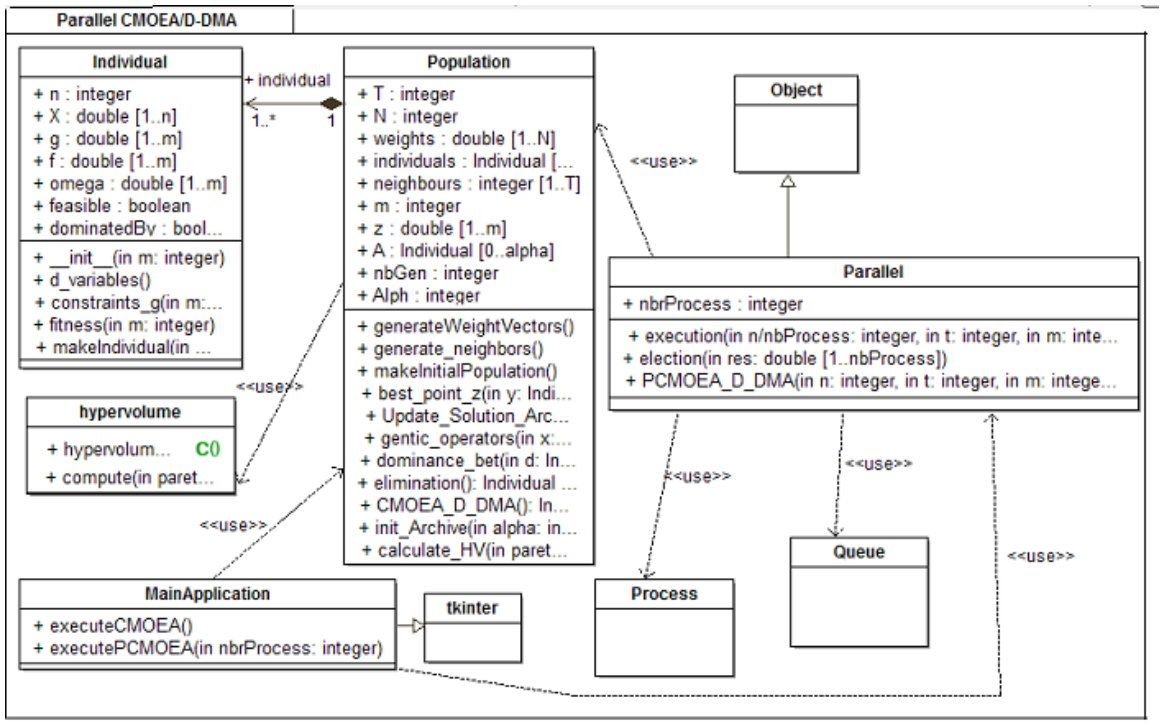


Figure 3.9 – Parallel CMOEA/D-DMA class diagram.

Figure 3.10 shows the sequences diagram this parallel version.

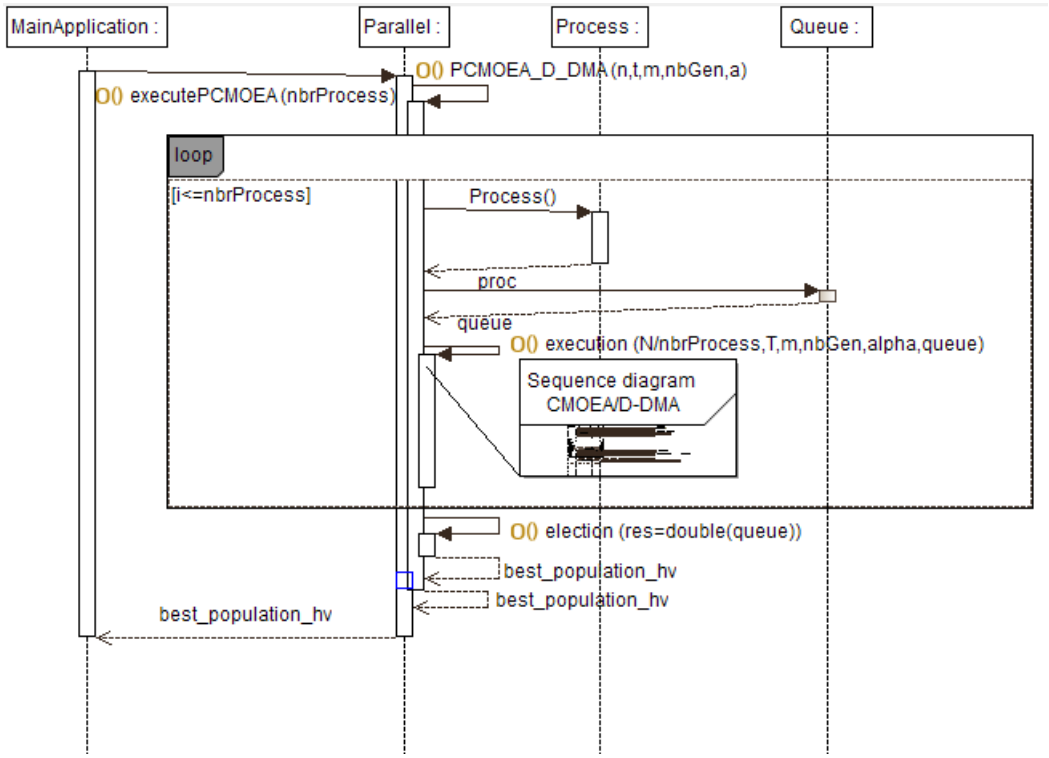


Figure 3.10 – Parallel CMOEA/D-DMA sequence diagram.

3.6.3 Detailed Design

In the parallel version a new class is added which is Parallel. The difference starts when we use the multiprocessing library to create k process. PCMOEA/D-DMA divides N (N is the population size) into p processes and chooses the result of the best one. The class and its different methods will be detailed inthe following.

3.6.3.1 Class Parallel:

This class contains 3 methods and only one parameter which is the processes number $NUM\_WORKERS$. In the following we will talk about the methods.

- **Execution**
  This method is concerned with executing the CMOEA/D-DMA. The processes takes this method as a parameter to execute the sequential version.
  Algorithm 11 shows the detailed function.

Algorithm 11 execution

Input: the sub-population size $\frac{N}{NUM\_WORKERS}$, the neighbourhood size T, number of objective functions m, number of generations $nbGen$, the archive size α, the queue where we save sub-populations HV values queue.

$pop \leftarrow$Population($N/NUM\_WORKERS$,T,m,$nbGen$,α) //creation of Population object with the parameters
//execution of CMOEA/D-DMA which gives as a result POS set
d$\leftarrow$pop.CMOEA/D-DMA()
/* obtain the PF from the POS set
//calculate the HV value and store it in the queue
queue$\leftarrow$ pop.calculate_HV(PF)

Where the Queue class is a near clone of Queue.Queue.

- **Election**
  In this method the best HV result will be chosen among those stored in the queue. This function is detailed in Algorithm 12.

Algorithm 12 election

Input: the parameter queue where we save sub-populations HV values.
Output: maxi the best HV value

```
/* store the queue values in the variable result
/*calculate the max value among HV values and save it in maxi
return maxi
```

- **PCMOEA/D-DMA**
  This method is concerned with creating processes and executing CMOEA/D-DMA. This function is detailed in Algorithm 13.

Algorithm 13 PCMOEA/D-DMA

Input: the population size N, the neighbourhood size T, number of objective functions m, number of generations $nbGen$, the archive size α.
Output: the best HV value

```
queue← Queue()//create the queue
for i = 1 : NUM_WORKERS do
    //create processes and assign to them CMOEA/D-DMA
    process←Process(target=execution,args(N/NUM_WORKERS,T,m,nbGen,α,queue))
    process.start()
    process.join()
election(queue)
end for
return the best HV value
```

Conclusion

In this chapter, we have presented our work. Where we have described the problem and given our solution, we have also provided a global design of our application. Then a description of both versions has been presented using UML diagrams and algorithmic. The next chapter is dedicated to implement the whole application which gathers the both versions (i.e., CMOEA/D-DMA and PCMOEA/D-DMA).

Chapter 4

Implementation & Experimental study

Introduction

After we have analyzed and given the design of each version (i.e., sequential and parallel) with a detailed description for all the project cycle. Now, we pass to the implementation of both versions CMOEA/D-DMA and PCMOEA/D-DMA. The goal behind this chapter is to present how we have implemented both versions, and the application that gathers both CMOEA/D-DMA and PCMOEA/D-DMA, test their efficiency by using one of the famous CMOP and compare the results obtained with other CMOEA results (CMOEA/D-DMA [19], CMOEA/D [11], TNSDM [21], CNSGA-III [6]).

This chapter is divided into 4 sections. In the first section we will review a set of tools and languages used to code the whole application. In the second one, we will present the main implementation results of our final application. The third section is to talk about experimental setup and we will show the final obtained results of both versions. In the last part, we will discuss and compare both versions results with other CMOEA results and mention the problems that we found during the implementation.

4.1 Development Tools and Languages

In this section, we present different tools and languages, that help us during the realization of our project in both levels (programming level, and theoretical level).

4.1.1 Python programming language

Python is an intelligent programming language that we have used it in the implementation of our application. It is easy to learn, because it is flexible, and its syntax doesn't hard to learn. A Python program is short than other languages' programs, because of the availability of many implemented functions. Python is an open source and untyped programming language. It is available for all these operating system (Windows, LINUX, Mac OS).

4.1.2 PyCharm Programming Editor

PyCharm is an open source Integrated Development Environment (IDE), used for python programming. It is a powerful coding assistant, it can highlight errors and introduces quick fixes based on an integrated Python debugger. It is a suitable editor for writing and testing many lines of code and classes, since it offers a structural project view, and a quick files navigation.

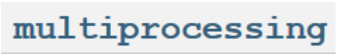
4.1.3 Tool Kit Interface "Tkinter" Package

Tool Kit Interface in short "*Tkinter*" [29], it is an open source *Graphical User Interface* (*GUI*) package. It is intended for Python programming language. We have preferred the *Tkinter* toolkit for developing GUIs of our application, because it is simple to learn it , and it is a powerful toolkit. It is available on both operating systems (Windows, Linux, and Mac OS).

4.1.4 Plotting Library "matplotlib"

matplotlib [14] is an open source Python library. It is used for 2D plotting. With a short code, one can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc, and can produce quality figures for the generated plots in a variety of hardcopy formats. Line styles, font properties, and axes properties are controlled by simple lines of code. Since in our project we have results that must be plotted, thus we have chosen *matplotlib* to plot them.

4.1.5 Process-based "threading" "interface"



multiprocessing [24] is a package that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both local and remote concurrency, effectively side-stepping the Global Interpreter Lock by using sub-processes instead of threads. Due to this, the multiprocessing module allows the programmer to fully leverage multiple processors on a given machine. It runs on both Unix and Windows. The multiprocessing module also introduces APIs which do not have analogs in the threading module. A prime example of this is the Pool object which offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism)[24].

4.1.6 Document Preparation System LaTeX

LaTeX [13] is a powerful and flexible typesetting system for producing high quality technical and scientific papers. It based on the tags language. It follows the design philosophy of separating presentation from content, thus authors focus on what they are writing, not on what is displayed, because the appearance is handled by LaTeX. The appearance includes many aspects, document structure (part, chapter, section, ..etc), figures, cross-references and bibliographies. It is more familiar to a computer programmer, because it follows the code-compile-execute cycle.

4.1.7 Typesetting Editor (TEX MAKER)

TEX MAKER [31] is a free and open source editor for drafting papers, based on LATEX system. It supports a powerful spell-checker, code auto-completion, and a *pdf* displayer. We have used TEX MAKER to draft our report and make our presentation, because it produces high quality papers and talks.

4.1.8 UML and BPMN modeling tool "Modelio"

Modelio [27] is a UML modeling tool available on Windows, Linux and Mac platforms. It also integrates BPMN modeling, and support for requirements modeling, dictionary, business rules and objectives.
Modelio offers a range of tools extending its functionalities allowing, among other things, the implementation of the MDA approach.

4.2 Implementation

The studied algorithms are implemented using oriented object programming (OOP) paradigm. In the following we will present the implementation process.

4.2.1 Software and Hardware

The software and hardware that have been used to implement the application are summarized in Table 4.1.

Software/Hardware	Version
OS	Microsoft Windows 7 Professional, 64bits, version 6.1.7601
CPU	Intel(R) Pentium(R) CPU G630 @2.70GHz 2.70GHz
RAM	4.00Go
Python Interpreter	3.7.7
PyCharm	2019.2.4
matplotlib	3.3.1
Tkinter	8.6

Table 4.1 – Software/Hardware versions

4.2.2 Main Application:

The GUI of our main application is shown in Figure 4.1.

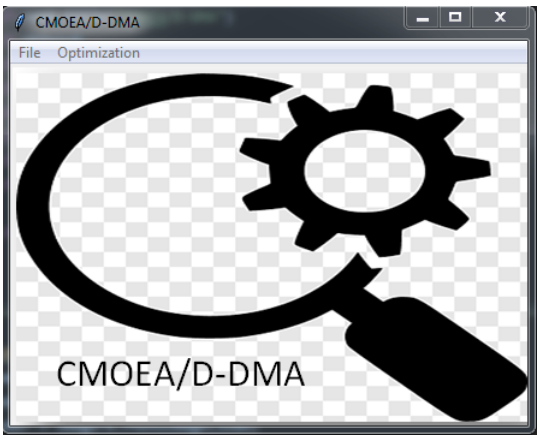


Figure 4.1 – Main Application GUI.

We developed many graphical user interfaces (GUIs) to facilitate the use of the application. As it is shown in Figure 4.1 The main application contains a menu bar with two menus (File and Optimization), Figure 4.2 and Figure 4.3 illustrate each menu and its commands.

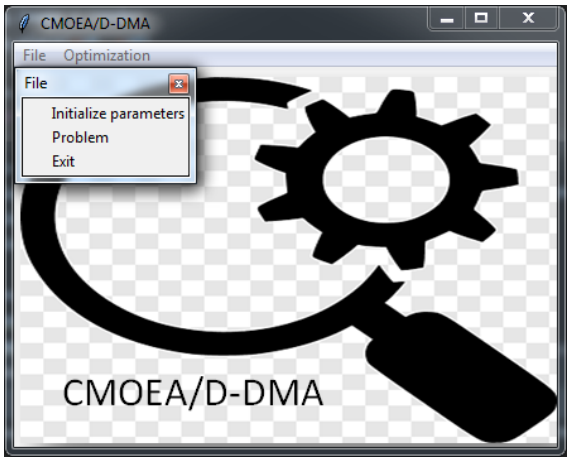


Figure 4.2 – File menu.

Figure 4.3 – Optimization menu.

File menu:
File menu offers three functionalities. The first one is to initialize the parameters, the second one is to check the CMOP formula. The last one is to quit the application.

- With the command **'Initialize parameters'** a GUI created. It allows to the user to initialize the parameters N, T, m, $nbGen$ and α and save them using the button **save**, because they will be used to execute both versions later.
  The input data are entered through the window shown in Figure 4.4

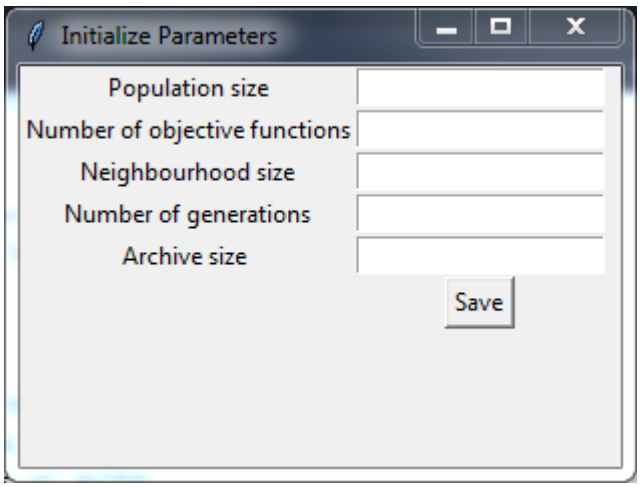


Figure 4.4 – Initialize Parameters frame.

- The command **'Problem'** shows a GUI that contains the resolved CMOP formula with a **Return** button. The problem formula GUI is shown in Figure 4.5

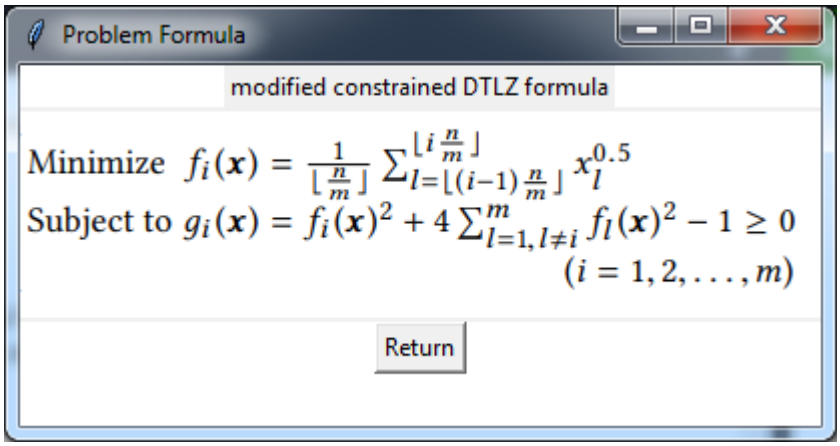


Figure 4.5 – Problem Formula frame [19].

- The command **'Exit'** is used as we said to close the main application.

Optimization menu:
This menu is dedicated to optimize the CMOP with the 2 versions (CMOEA/D-DMA, PCMOEA/D-DMA). Optimization menu offers three functionalities, optimization with CMOEA/D-DMA, optimization with PCMOEA/D-DMA and a comparative case of the two versions.

- With the command **'CMOEA/D-DMA'** the sequential version executed, With creation of new GUI. The created GUI allows to the user to view the algorithm results. The results shown are the HV value and the execution time. Also it allows to the user to return to the main application GUI 4.1 by pressing the **Return** button.
  The following GUI (Figure 4.6) shows the results of pressing CMOEA/D-DMA after initializing the parameters with GUI shown in Figure 4.4 .

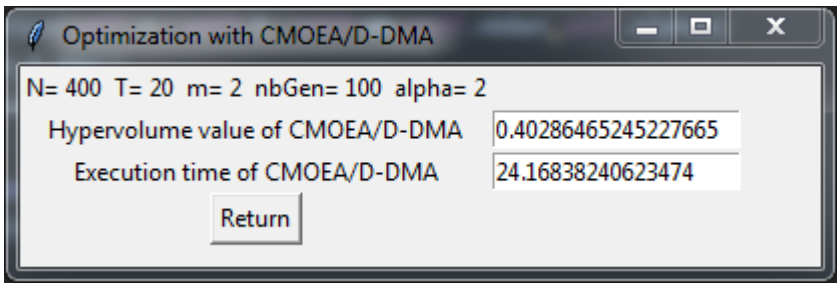


Figure 4.6 – Sequential CMOEA/D-DMA frame.

- The command **'PCMOEA/D-DMA'** is concerned of running the parallel version and shows its results in a GUI. But before accessing to see results, a GUI that is used to enter the created processes number is created. The Figure 4.7 shows the GUI concerned of the processes number.

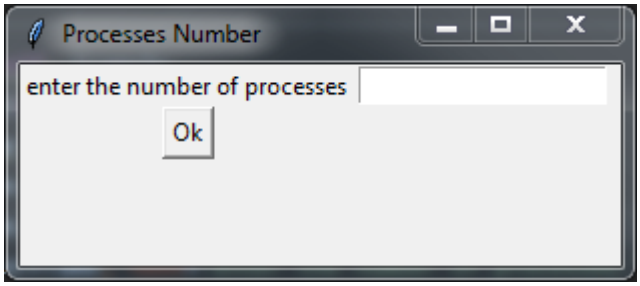


Figure 4.7 – Processes Number frame.

After entering the processes number and pressing **Ok** button a GUI contains the HV and execution time results shown. The GUI also contains a **Return** button which allows to the user to return to the main application. The Parallel CMOEA/D-DMA GUI is shown in Figure 4.9

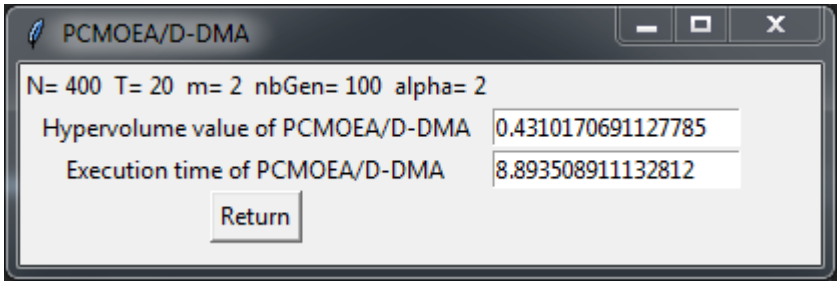


Figure 4.8 – Parallel CMOEA/D-DMA frame.

- The command **'Comparative case'** shows a GUI which contains results of HV and execution time of both versions to facilitate the comparison between them. The GUI also contains a **Return** button which allows to the user to return to the main application..

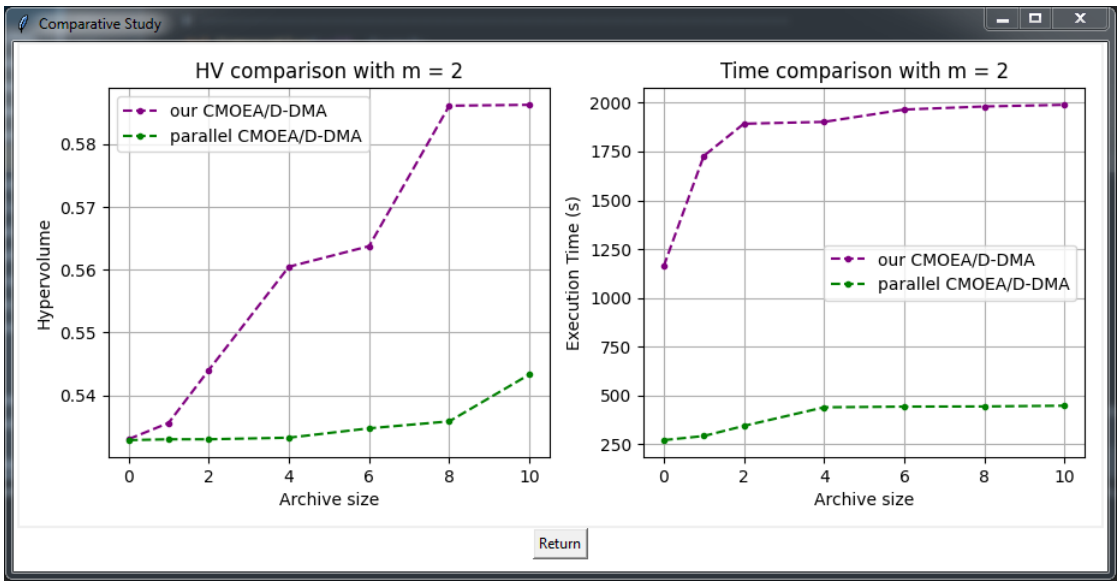


Figure 4.9 – Comparative case frame.

The diagram on the right represents the *execution time* related to both versions and the diagram on the left represents the *HV* values in each version. Where the purple dashed line refers to the sequential version values and the green dashed line refers to the parallel version values.

4.3 Test & Experimental Study

We divided this section into three parts. The first part is to present the experimental environment. The last two parts are to show the obtained results of both versions.

4.3.1 Experimental setup

4.3.1.1 Test Problem

mCDTLZ (modified constrained DTLZ) [20] is a continuous minimization problem involving m objectives and m constraints. mCDTLZ is extended from C3-DTLZ4 [11] and DTLZ9 [31] in the DTLZ test suite and defined as follows:

$$\begin{cases} \text{Minimize } f_i(x) = \frac{1}{[\frac{n}{m}]} \sum_{l=[(i-1)\frac{n}{m}]}^{[i\frac{n}{m}]} x_l^{0.5} \\ \text{Subject to } g_i(x) = f_i(x)^2 + 4 \sum_{l=1,l\neq i}^{m} f_l(x)^2 - 1 \geq 0 \quad s \\ \qquad\qquad (i = 1, 2, ..., m) \end{cases}$$

A solution (variable vector) x consists of n variables $(x1, x2, ..., xn)$, and all the variables are real parameters in the range $[0, 1]$. Each bound $gi = 0$ $(i = 1, 2, ..., m)$ becomes a part of Pareto front. In this problem, the numbers of objectives, constraints and variables are user-defined parameters (as we have presented in Figure 3.3). However, the number of constraints is equivalent to the number of objectives.

4.3.1.2 Algorithms

In this section, we will present briefly algorithms that are compared to sequential version results. We have chosen 4 constrained MOEAs, their results are obtained from the paper [19]. The four algorithms are:

1. CMOEA/D-DMA [19] is the algorithm as the developers program it not the sequential version that we implemented.

2. CMOEA/D [11] which is already mentioned in chapter 1 section 1.4.2.

3. TNSDM [21] also has been mentioned in chapter 1 section 1.4.1.

4. CNSGA-III [6] is the constrained NSGA-III [11], NSGA-III is one of the latest version of NSGA [28]. NSGA-III is especially designed for solving MOPs and involved the decomposition approach of the objective space such as MOEA/D. The main difference from the representative NSGA-II [6] is the maintenance mechanism of diversity of solutions in the objective space. NSGA-III employs the perpendicular distances between each solution and the pre-defined reference lines corresponding to the weight vectors used in MOEA/D while NSGA-II uses the crowding distance accumulating relative densities of solutions on each objective value. NSGA-III also employ the non-dominated sorting as the first criterion to discriminate solution in the population. In CNSGA-III, the constraint dominance is used to compare two solutions instead of the conventional Pareto dominance. For two solutions x and y, x constrain-dominates y ($x \prec_\Omega y$) if any of the following conditions are satisfied (Ω is the sum of constraint violation values, check chapter 1 section 1.2.2).

   (a) x and y are feasible, and x dominates y on the objective function (f) value ($x \prec_f y$)

   (b) x is feasible and y is infeasible.

   (c) x and y are infeasible, and the sum of constraint violation values of x is lower than the one of y.

4.3.1.3 Parameters

The problem parameters are $m = 2$ objectives (also 2 constraints for each objective) and $n = 10m$ variables. For the genetic operators we use SBX [5] for the crossover (the crossover ratio $P_c = 1.0$, the distribution parameter $\eta_c = 20$) and the polynomial mutation [5] (the mutation rate $P_m = 1/n$, the distribution parameter $\eta_m = 20$). The problem and genetic operators parameters are applied for all the algorithms that will be mentioned.
Algorithms parameters will be presented later in the comparative study, because each case study has its own parameters.

4.3.1.4 Performance metrics

As we have presented in the global design (Figure 3.3), our application outputs are POS, HV and the execution time. To measure algorithms performance, we will use HV values of the obtained PF and the execution time. For HV we need the algorithm PF and a reference point r where $r = [1.1_1, ..., 1.1_m]$ in this project, and for execution time we will use python **time library**.

4.3.2 Experimental results of CMOEA/D-DMA

In this section, experimental results of the sequential version will be shown. Where we will change the population size N and measure HV values and the execution time in terms of the archive size α.
Figure 4.10 shows HV and execution time results when the population size $N = 100$.

- Population size $N = 100$,
- Neighbour size $T = 20$,
- Archive size $\alpha = \{0, 1, 2, 4, 6, 8, 10\}$,
- Generation number $nbGen = 5000$,
- Run number= 30.

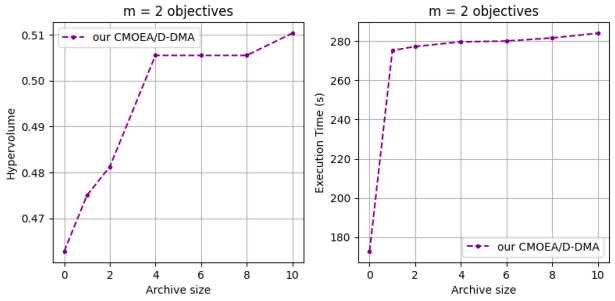


Figure 4.10 – Sequential version results with $N = 100$

Figure 4.11 shows HV and execution time results when the population size $N = 201$.

- Population size $N = 201$,
- Neighbour size $T = 20$,
- Archive size $\alpha = \{0, 1, 2, 4, 6, 8, 10\}$,
- Generation number $nbGen = 5000$,
- Run number= 30.

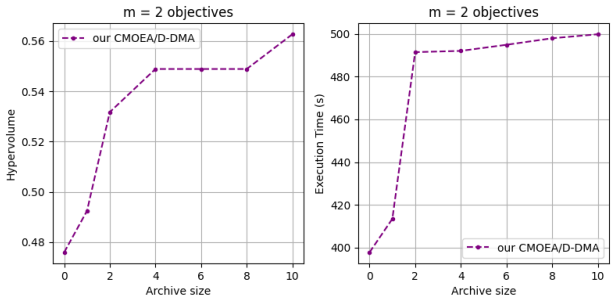


Figure 4.11 – Sequential version results with $N = 201$

Figure 4.12 shows HV and execution time results when the population size $N = 300$.

- Population size $N = 300$,
- Neighbour size $T = 20$,
- Archive size $\alpha = \{0, 1, 2, 4, 6, 8, 10\}$,
- Generation number $nbGen = 5000$,
- Run number= 30.

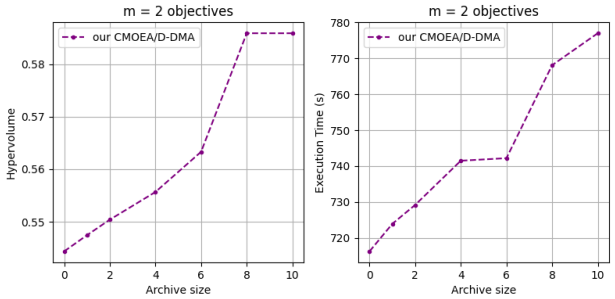


Figure 4.12 – Sequential version results with $N = 300$

Figure 4.13 shows HV and execution time results when the population size $N = 400$.

- Population size $N = 400$,
- Neighbour size $T = 20$,
- Archive size $\alpha = \{0, 1, 2, 4, 6, 8, 10\}$,
- Generation number $nbGen = 5000$,
- Run number= 30.

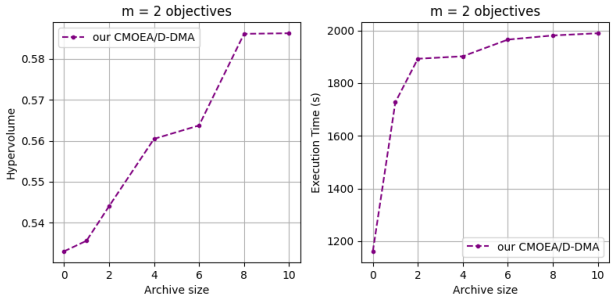


Figure 4.13 – Sequential version results with $N = 400$

These were the results which we had obtained in our test of CMOEA/D-DMA, which represent the evaluation of hypervolume and execution time. As we said in the beginning, what we are looking for is a constrained decomposition based algorithm that can handle CMOPs with large scale, and gives well extended pareto-front in a reasonable execution time. We tried to change the population size and study HV values and the execution time in terms of archive size.

4.3.3 Experimental results of PDMOEA:

Now, we will pass to test the parallel version of CMOEA/D-DMA which is CMOEA/D-DMA by using two methods. The first method is the same method that we used to test the sequential version, the second one is to measure HV and execution time values in terms of the number of processes p. We used those methods based on the factors effecting the parallel version performance.

First method

The first factor is the population size and the archive size.
Figure 4.10 shows HV and execution time results when the population size $N = 100$.

- Population size $N = 100$,
- Neighbour size $T = (N/p) - 2 = 8$,
- Archive size $\alpha = \{0, 1, 2, 4, 6, 8, 10\}$,
- Generation number $nbGen = 5000$,
- Run number= 30,
- Processes number $p = 10$.

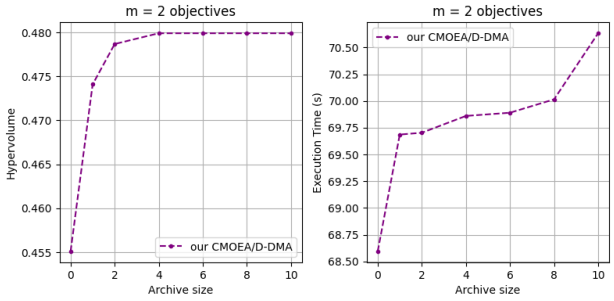


Figure 4.14 – Parallel version results with $N = 100$

Figure 4.10 shows HV and execution time results when the population size $N = 200$.

- Population size $N = 200$,
- Neighbour size $T = (N/p) - 2 = 18$,
- Archive size $\alpha = \{0, 1, 2, 4, 6, 8, 10\}$,
- Generation number $nbGen = 5000$,
- Run number= 30,
- Processes number $p = 10$.

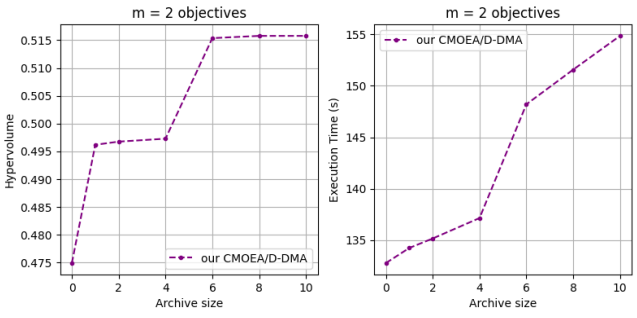


Figure 4.15 – Parallel version results with $N = 200$

Figure 4.10 shows HV and execution time results when the population size $N = 300$.

- Population size $N = 300$,
- Neighbour size $T = (N/p) - 2 = 28$,
- Archive size $\alpha = \{0, 1, 2, 4, 6, 8, 10\}$,
- Generation number $nbGen = 5000$,
- Run number= 30,
- Processes number $p = 10$.

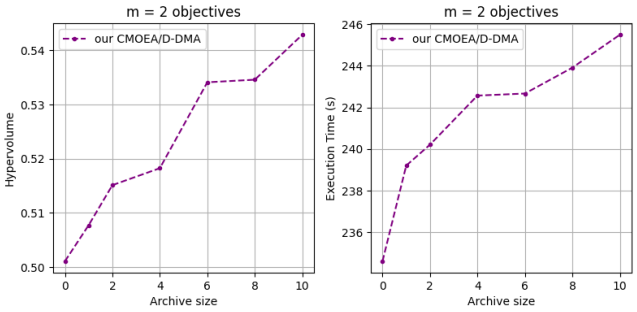


Figure 4.16 – Parallel version results with $N = 300$

Figure 4.10 shows HV and execution time results when the population size $N = 400$.

- Population size $N = 400$,
- Neighbour size $T = (N/p) - 2 = 38$,
- Archive size $\alpha = \{0, 1, 2, 4, 6, 8, 10\}$,
- Generation number $nbGen = 5000$,
- Run number= 30,
- Processes number $p = 10$.

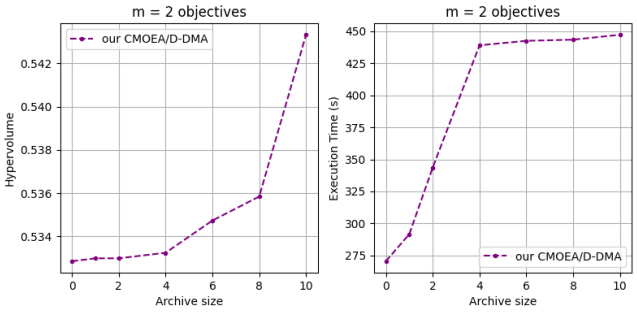


Figure 4.17 – Parallel version results with $N = 400$

As we can see HV values have improved whenever we increase the population size in an acceptable execution time .

Second method

In this method we will measure HV and execution time values in terms of processes number p, where we will change the number of processes and fix all the other parameters.
The table below show the parameters used in this test.

	Population size N	Generation number $nbGen$	Neighbour size T	Archive size α	Run number	processes number p
PCMOEA/D-DMA	800	5000	$(N/p) - 2$	10	30	2
PCMOEA/D-DMA	800	5000	$(N/p) - 2$	10	30	4
PCMOEA/D-DMA	800	5000	$(N/p) - 2$	10	30	8
PCMOEA/D-DMA	800	5000	$(N/p) - 2$	10	30	10
PCMOEA/D-DMA	800	5000	$(N/p) - 2$	10	30	20
PCMOEA/D-DMA	800	5000	$(N/p) - 2$	10	30	22
PCMOEA/D-DMA	800	5000	$(N/p) - 2$	10	30	24

Table 4.2 – Changing processes number for PCMOEA/D-DMA

The results of this experiment are illustrated in the Figure 4.18

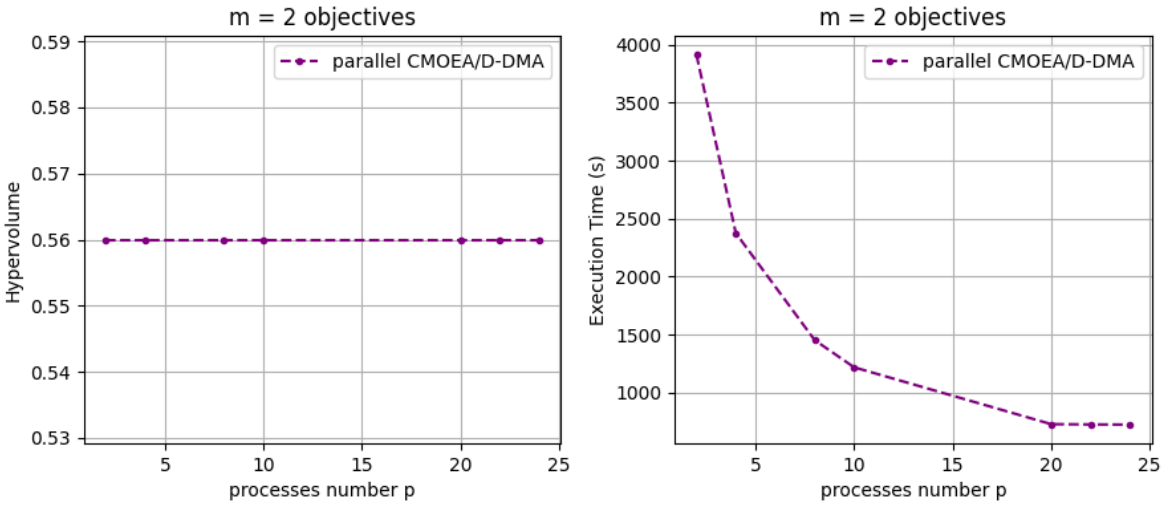


Figure 4.18 – PCMOEA/D-DMA results changing processes number.

Execution time results have been improved in terms of processes number p as it is shown in the plot in the right. However, HV results didn't change when we changed processes number. That means the processes number factor effect the execution time in term of good.

4.4 Comparative study

We will present in this part the comparative results and discuss them. This part where we will compare the sequential version with other

4.4.1 CMOEA/D-DMA Vs CMOEAs

In this section, we will compare our sequential version algorithm with other CMOEAs (algorithms are mentioned in section 4.3.1.2) that share the same nature with our implemented algorithm.
The Table below shows algorithms parameters.

	Initial population size N	Neighbor size T	Archive size α	total number of generations
Our CMOEA/D-DMA	201	20	$\{0, 1, 2, 4, 6, 8, 10\}$	5000
CMOEA/D-DMA	201	20	$\{0, 1, 2, 4, 6, 8, 10\}$	5000
CMOEA/D	201	20	$\{0, 1, 2, 4, 6, 8, 10\}$	5000
TNSDM	201	20	$\{0, 1, 2, 4, 6, 8, 10\}$	5000
CNSGA-III	201	20	$\{0, 1, 2, 4, 6, 8, 10\}$	5000

Table 4.3 – Algorithms parameters setting for CMOEA/D-DMA Vs CMOEAs

The results of this comparison are represented in Figure 4.19.

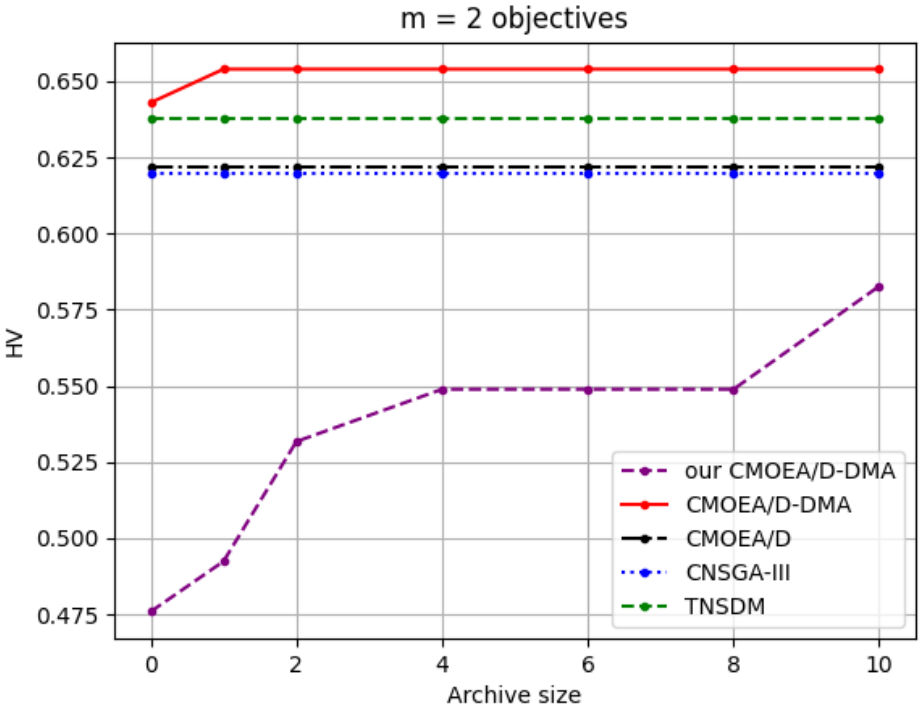


Figure 4.19 – Comparative results of algorithms with $N = 201$.

Observation and discussion

These results are the HV results of all the algorithms. As we can see HV results have improved in our CMOEA/D-DMA, but it didn't show the best results compared to the other algorithms. But the original algorithm CMOEA/D-DMA shows the best HV results.

Problems:
In the implementation process of the sequential version we faced some problems that obstruct us from obtaining the best results. The main two problems that we faced are:

1. The first problem is that EAs characterized by generating individuals randomly, and when we used **python's random** library to generate individuals we didn't obtain a variation in the results.

2. The second problem is in the application of SBX crossover and polynomial mutation. We have no clue of how it's applied in the original paper [19].

4.4.2 CMOEA/D-DMA Vs PCMOEA/D-DMA

We had compared the execution time and HV results of both versions, and we had concluded that in each execution of this both versions using the same inputs, the CMOEA/D-DMA is better than PCMOEA/D-DMA.
Now we will show the results of both versions in terms of archive size and we will discuss them.
The table below shows algorithms parameters settings.

	Initial population size N	Neighbor size T	Archive size α	total number of generations	Processes number	Run number
Our CMOEA/D-DMA	400	20	$\{0, 1, 2, 4, 6, 8, 10\}$	5000	10	30
PCMOEA/D-DMA	400	20	$\{0, 1, 2, 4, 6, 8, 10\}$	5000	10	30

Table 4.4 – Algorithms parameters setting for CMOEA/D-DMA Vs PCMOEA/D-DMA.

The results of this comparison are represented in Figure 4.20.

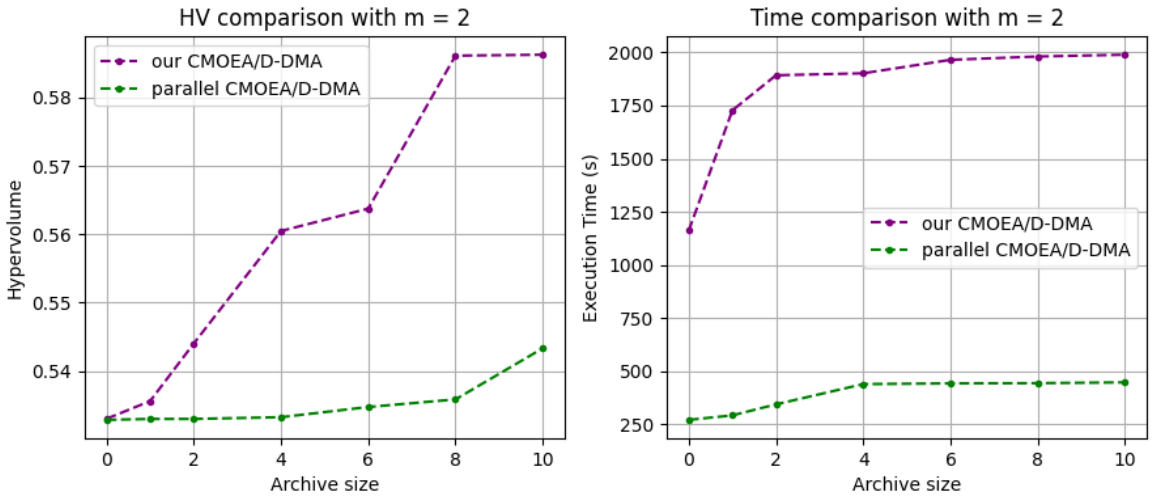


Figure 4.20 – Comparative results of CMOEA/D-DMA Vs PCMOEA/D-DMA.

Observation and discussion

As writing in the legend of each plot the purple dashed line represents the sequential version results and the green dashed line represents parallel version results.

Observation

The plot in the right side of the figure 4.20 shows the execution time results in term of archive size. As we can see the execution time of the PCMOEA/D-DMA is better than CMOEA/D-DMA execution time. From the plot in the left side of the figure 4.20 we can observe HV results of both versions. HV results of CMOEA/D-DMA are much better than PCMOEA/D-DMA results.

Discussion

After the observation of execution time plot we can say that we achieved our first goal from the parallelization which is less execution time.
However, when we observed the HV plot we figured that the second goal which is well-extended PF is not achieved. That what guide us to conclude that CMOEA/D-DMA is better than PCMOEA/D-DMA.

Conclusion

In this chapter, we have presented the tools that we have used to release our application, then the results of our implementation as a set of GUIs and plots. The last section discussed the results of running the implemented algorithms with various CMOEAs.

General Conclusion

General Conclusion

In the last few years there has been a growing interest in solving constrained multi-objective optimization problems using evolutionary algorithms. Our project has dealt with a constrained multi-objective optimization evolutionary algorithm based on decomposition approach that proposed recently which is CMOEA/D-DMA. Also it has dealt with parallelization of CMOEA/D-DMA since the nature of this kind of algorithm is characterized by long execution time.

During the project realisation, we have learned knowledge about:

- Constrained Multi-objective optimization problems (CMOPs)
- Constrained Multi-objective optimization evolutionary algorithms (CMOEAs),
- Constrained handling methods in MOEAs,
- Constrained Multi-objective Optimization Evolutionary Algorithm based on Decomposition and Directed Mating (CMOEA/D-DMA),
- Parallel computing, including parallel terminologies, parallel architectures and parallel programming models,
- Parallelization of CMOEA/D-DMA (i.e., PCMOEA/D-DMA).

To prove the efficiency of evolutionary algorithms for solving constrained multi-objective optimization problems, we have implemented an algorithm named CMOEA/D-DMA. CMOEA/D-DMA is designed based on MOEA/D which is a decomposition approach promising for solving multi-objective optimization, the directed mating enhancing the search on problems with constraints by utilizing useful infeasible solutions having better scalarizing function values than feasible ones. We used as test problem the same test problem mentioned in [19] and have obtained hypervolume HV values which have compared with other evolutionary algorithms results. Without forgetting that our main objective is not limited to implement the CMOEA/D-DMA and being satisfied with its results, but the main objective is to improve obtained results in terms of minimizing computational time and maximizing HV values. This reason have led us to think about parallelizing the CMOEA/D-DMA and implement this parallel version of CMOEA/D-DMA. We have implemented algorithms using Python programming language. Our application is useful for any test function.
In our future research we intend to concentrate on addressing other questions which remain to resolve, some of which are:

- Test the algorithms with other CMOPs,
- Increase the number of objectives (more than two objectives),
- Try to impove the search performance of PCMOEA/D-DMA even more by using other parallelization techniques.
- Implement other CMOEAs (their sequential as well as their parallel versions) and compare them with CMOEA/D-DMA and PCMOEA/D-DMA.

Bibliography

[1] Blaise Barney et al. "Introduction to parallel computing". In: *Lawrence Livermore National Laboratory* 6.13 (2010), p. 10.

[2] Carlos A Coello Coello and Alan D Christiansen. "MOSES: A multiobjective optimization tool for engineering design". In: *Engineering Optimization* 31.3 (1999), pp. 337–368.

[3] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*. Vol. 16. John Wiley & Sons, 2001.

[4] Kalyanmoy Deb et al. "Evolutionary algorithms for multi-criterion optimization in engineering design". In: *Evolutionary algorithms in engineering and computer science* 2 (1999), pp. 135–161.

[5] Kalyanmoy Deb and Mayank Goyal. "A combined genetic adaptive search (GeneAS) for engineering design". In: *Computer Science and informatics* 26 (1996), pp. 30–45.

[6] Kalyanmoy Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[7] Xavier Gandibleux et al. *Metaheuristics for multiobjective optimisation*. Vol. 535. Springer Science & Business Media, 2004.

[8] geeksforgeeks. *Introduction to Parallel Computing*. `https://www.geeksforgeeks.org/introduction-to-parallel-computing`.

[9] Hisao Ishibuchi and Shiori Kaige. "Effects of repair procedures on the performance of EMO algorithms for multiobjective 0/1 knapsack problems". In: *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.* Vol. 4. IEEE. 2003, pp. 2254–2261.

[10] Hisao Ishibuchi, Noritaka Tsukamoto and Yusuke Nojima. "Evolutionary many-objective optimization: A short review". In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, pp. 2419–2426.

[11] Himanshu Jain and Kalyanmoy Deb. "An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: handling constraints and extending to an adaptive approach". In: *IEEE Transactions on evolutionary computation* 18.4 (2013), pp. 602–622.

[12] Yoram Koren, Xi Gu and Weihong Guo. "Reconfigurable manufacturing systems: Principles, design, and future trends". In: *Frontiers of Mechanical Engineering* 13.2 (2018), pp. 121–136.

[13] Leslie Lamport. LaTeX. `https://en.wikipedia.org/wiki/LaTeX`.

[14] matplotlib. `http://matplotlib.org/`.

[15] PE Mergos and Anastasios Sextos. "Multi-objective optimum selection of ground motion records with genetic algorithms". In: (2018).

[16] Efrén Mezura-Montes. "Constraint-Handling in Evolutionary Optimization; Efrén Mezura-Montes (Editor)". In: *Journal of Computer Science and Technology* 9.01 (2009), pp. 34–35.

[17] K. Miettinen. "Nonlinear multiobjective optimization". In: (1999).

[18] Minami Miyakawa, Hiroyuki Sato and Yuji Sato. "A Study for Parallelization of Multi-Objective Evolutionary Algorithm Based on Decomposition and Directed Mating". In: *Proceedings of the 2019 3rd International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence*. 2019, pp. 25–29. DOI: 10.1145/3325773.3325790.

[19] Minami Miyakawa, Hiroyuki Sato and Yuji Sato. "Directed mating in decomposition-based MOEA for constrained many-objective optimization". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2018, pp. 721–728. DOI: 10.1145/3205455.3205554.

[20] Minami Miyakawa, Keiki Takadama and Hiroyuki Sato. "Controlling selection areas of useful infeasible solutions for directed mating in evolutionary constrained multi-objective optimization". In: *Annals of Mathematics and Artificial Intelligence* 76.1-2 (2015), pp. 25–46. DOI: 10.1007/s10472-015-9455-9.

[21] Minami Miyakawa, Keiki Takadama and Hiroyuki Sato. "Two-stage non-dominated sorting and directed mating for solving problems with multi-objectives and constraints". In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. 2013, pp. 647–654.

[22] NEOS. *Introduction to Optimization*. https://neos-guide.org/content/optimization-introduction. Accessed on 2020-03-12.

[23] U Pferschy, H Kellerer and D Pisinger. *Knapsack Problems*. 2004.

[24] multiprocessing Process based threading interface Python documentation. https://docs.python.org/2/library/multiprocessing.html.

[25] Tapabrata Ray et al. "Infeasibility driven evolutionary algorithm for constrained optimization". In: *Constraint-handling in evolutionary optimization*. Springer, 2009, pp. 145–165.

[26] Nery Riquelme, Christian Von Lücken and Benjamin Baran. "Performance metrics in multi-objective optimization". In: *2015 Latin American Computing Conference (CLEI)*. IEEE. 2015, pp. 1–11.

[27] Modelio Open Source-UML. https://www.modelio.org/.

[28] Nidamarthi Srinivas and Kalyanmoy Deb. "Muiltiobjective optimization using non-dominated sorting in genetic algorithms". In: *Evolutionary computation* 2.3 (1994), pp. 221–248.

[29] techopedia. *Parallel Computing*. https://www.techopedia.com/definition/8777/parallel-computing.

[30] Anupam Trivedi et al. "A survey of multiobjective evolutionary algorithms based on decomposition". In: *IEEE Transactions on Evolutionary Computation* 21.3 (2016), pp. 440–462.

[31] L'editeur LaTeX universel. .http://www.xm1math.net/texmaker/index_fr.html.

[32] wikipedia. *Constrained optimization*. https://en.wikipedia.org/wiki/Constrained_optimization. Accessed on 2020-03-10.

[33] Yonas Gebre Woldesenbet, Gary G Yen and Biruk G Tessema. "Constraint handling in multiobjective evolutionary optimization". In: *IEEE Transactions on Evolutionary Computation* 13.3 (2009), pp. 514–525.

[34] Qingfu Zhang and Hui Li. "MOEA/D: A multiobjective evolutionary algorithm based on decomposition". In: *IEEE Transactions on evolutionary computation* 11.6 (2007), pp. 712–731. DOI: 10.1145/3325773.3325790.

[35] Eckart Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications*. Vol. 63. Citeseer, 1999.

[36] Eckart Zitzler and Lothar Thiele. "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach". In: *IEEE transactions on Evolutionary Computation* 3.4 (1999), pp. 257–271.