People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University Mohamed Seddik Ben Yahia
Jijel
Faculty of Exact Sciences and Computer Science
Department of Computer Science

Dissertation

Submitted in partial fulfillment of the Requirements for the Degree of :

MASTER

In **Computer Science**

Option: Artificial Intelligence

Multi-Objective Approaches for Designing Cellular Manufacturing Systems

Student:
Mr. Badis
Bouchama

Supervisor:
Dr. Hamida
Bouaziz

Publicly held before a jury, on the 11th of September 2022.

Dedication

I am dedicating this work

To my most cherished mother, Nacera No matter what I do or say, I cannot adequately express my gratitude. Your affection surrounds me, your benevolence guides me, and your presence by my side has always been my source of courage in the face of adversity; even though you are no longer present, your warmth has not yet left me. May God have mercy on your soul.

To my dearest father, Abdeslam, thank you for the noble values, education, and support. No dedication can express the love, esteem, and respect that I always have for you. Thank you for always being there for me.

To Ramzi, my brother May God grant you success, prosperity, courage, and good health.

To my grandmother, Fatima Zohra, and my aunt, Souhila, I wish from the bottom of my heart a life filled with joy, happiness, and love.

To all my family and dear friends for their support throughout my university career.

To all who care about me and whom I cherish.

Badis

Acknowledgements

I owe everything I am and everything I have to my **Almighty God**, who has blessed me with the fortitude, belief, health, determination, and inspiration to complete even this meager task.

I'd like to give special thanks to Dr. Hamida Bouaziz, my supervisor, for all the time and effort she put into supervising, helping, guiding, advising, and encouraging me as I did my work.
I appreciate all of her hard work and time as my manager, which I greatly appreciate. Work was completed successfully because of her directions.

I'd also like to express my gratitude to the members of the jury who have agreed to read my submission and serve on the jury.
I'd like to thank all of the professors in the computer science department at the University of Jijel for their contributions to my education and development.

To everyone who has helped make this Dissertation possible, thank you from the bottom of my heart.

For their patience and unwavering support throughout my many years of schooling, I am eternally grateful to my loved ones. All of you deserve the highest level of gratitude, respect, appreciation, and admiration for your contributions.

Abstract

Manufacturing cell formation is an important stage in the development of cellular manufacturing systems. It focuses on grouping machines, parts, and workers and assigning them to appropriate cells. This assignment is guided by a number of objectives and is subject to a number of constraints. The focus of this work is on a variant of the cell formation problem known as the "Generalized Cubic Cell Formation Problem." To solve the problem, Multi-Objective Evolutionary Algorithms NSGA-II and its variant NRGA are developed. The performance of NSGA2 and NRGA has been evaluated in terms of the objectives considered and the computation time. The simulation results show that the overall performance of NSGA-II and NRGA algorithms is satisfactory. The NRGA, on the other hand, had a longer execution time.

Keywords : *Cell formation, Cellular manufacturing, Generalized cubic cell formation problem, Group technology, Multi-Objective Evolutionary Algorithms, Multi-Objective Optimisation, NSGA-II, NRGA*

Résumé

La formation des cellules de fabrication est une étape importante dans le développement des systèmes de fabrication cellulaire. Elle consiste à regrouper les machines, les pièces et les travailleurs et à les affecter aux cellules appropriées.
Cette affectation est guidée par un certain nombre d'objectifs et est soumise à un certain nombre de contraintes. Ce travail se concentre sur une variante du problème de formation de cellules connue sous le nom de "problème de formation de cellules cubiques généralisées". Pour résoudre ce problème, des algorithmes génétiques multi-objectifs (NSGA2 et sa variante NRGA) sont développés. Les performances de NSGA2 et NRGA ont été évaluées en termes d'objectifs considérés et de temps de calcul. Les résultats de la simulation montrent que la performance globale des algorithmes NSGA et NRGA est satisfaisante. En revanche, le NRGA présente un temps d'exécution plus long.

Mots-clés: *Formation de cellules, fabrication cellulaire, problème de formation de cellules cubiques généralisées, technologie de groupe, algorithmes évolutionnaires multi-objectifs, optimisation multi-objectifs, NSGA-II, NRGA.*

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

CM	Cellular Manufacturing
CMS	Cellular Manufacturing System
CFP	Cell Formation Problem
GCCFP	Generalized Cubic Cell Formation Problem
GT	Group Technology
GCFP	Generalized Cell Formation Problem
InterCMC	Inter-cellular Movement Cost
InterCMHC	Inter-Cellular Material Handling Cost
CO	Combinatorial Optimization
MOO	Multi-Objective Optimization
EA	Evolutionary Algorithms
MOEA	Multi-Objective Evolutionary Algorithms
ACO	Ant Colony Optimization
GA	Genetic Algorithm
NRGA	Non-dominated Ranking Genetic Algorithms
NSGA-II	Non-dominated Sorting Genetic Algorithm II

NSGA	Non-dominated Sorting Genetic Algorithms
VEGA	Vector Evaluation Genetic Algorithme
NPGA	Niched Pareto Genetic Algorithme

INTRODUCTION

Group technology (GT) is one of the primary applications of cellular manufacturing (CM). CM emerged as a production strategy capable of resolving specific problems associated with batch production's complexity and lengthy manufacturing lead times [1]. Its purpose is to streamline the management of manufacturing industries. By grouping the production of various components into clusters, the manufacturing management is reduced to overseeing numerous small entities. The design of these entities, known as manufacturing cells, is one of the most pressing issues in cellular manufacturing [11].

These cells represent a cluster of machines devoted to the production of a single or multiple parts. The optimal design for cellular manufacturing would make these cells independent from one another. The actuality is somewhat more complex. There is still traffic within and between cells after they have been created. Inter-cellular movement is the transfer of a component between two machines from different cells. The intra-cellular movement, however, involves the transfer of a component between two machines within the same cell. Each component is initially defined by one or more sequences of operations, and each can be manufactured using a specific sequence of machines. To produce each part, a specific sequence of machines must be selected. In addition to the part and machine dimensions, the worker's dimensions are also taken into account. Due to the importance of the human factor, it is possible to improve the quality of the CMS design by grouping workers with similar expertise and skills. This issue is referred to as the Cell Formation Problem (CFP) in the scientific literature.

The cell formation problem is an NP-hard problem. Consequently, precise methods cannot be used to solve large problems in a reasonable amount of time. However, metaheuristics can produce high-quality solutions in a reasonable amount of time. This study examines a variant of the CFP known as the Generalized Cubic Cell Formation Problem (GCCFP). It is characterized by the adjective "Generalized" because each component may have more than one production plan. However, it is a cubic problem because, in addition to the part and machine dimensions, the worker dimension must also be considered. At the level of the problem's constraints, multiple criteria can be specified. In this study, three criteria are taken into account: the inter-cellular movement of parts and workers, the intra-cellular movement of parts, and the quality index.

The ultimate goal is to construct cells and assign parts, machines, and workers to these cells so that:

- To reduce intra-cellular material handling expenses
- To reduce the movement of parts and employees between cells.
- To maximize the quality index of manufactured parts.

Numerous algorithms have been used in the literature to solve the Generalized Cubic Cell Formation Problem [6]. Yet, at the time of writing this dissertation, None of them made use of Multi Objective Optimization. In this study, we attempt to measure the effectiveness of Multi Objective methods to solve GCCFP. For that, two traditional population-based Multi-objective evolutionary algorithms will be used :

1. Non dominated sorting genetic algorithm
2. Non dominated ranking genetic algorithm

The structure of this master's dissertation consists of three chapters:

- In the first chapter, the Cell Formation Problem and its variants will be introduced.
- Multi-Objective Evolutionary Algorithms (MOEA) will be discussed in the second chapter. by describing its Exact and approximate resolution methods and providing an overview of the previously mentioned algorithms.
- In the final chapter of this dissertation, we will discuss the solution we came up with for this issue as well as the findings of our investigation. We use a variety of performance indicators to compare the outcomes produced by our algorithms with those found by Augmecon-R (multi-objective exact method) and also between themselves in order to highlight the value and effectiveness of this approach.
- We come to a general conclusion that summarizes the research that was done and offers some perspectives.

CHAPTER 1 CELL FORMATION PROBLEM.

1.1 Introduction

Using product-oriented departments to create standardized items in machine firms resulted in reduced transportation as early as the 1920s. This is considered the beginning of Group Technology (GT). GT is a manufacturing concept that focuses on structuring and grouping common tasks in order to increase system productivity [1]. Parts are categorized, and parts with comparable features are manufactured together using standardized techniques. As a result, small "dedicated factories" are emerging as separate operating units within larger facilities [2].

The group technology principle is to divide the manufacturing facility into small groups or cells of machines. In this context, the phrase "cellular manufacturing" is frequently employed. Each of these cells is assigned to a specific family or group of part kinds. Usually, a cell would have a machining center with the number of skilled operators it needs and some way to move things around, like a conveyor belt or something similar.

There are several advantages to cellular manufacturing (CM), such as reduced setup and material handling costs, lower in-process inventories, longer component lifespans, and better operator competence [1]. The most important part of making a cellular manufacturing system (CMS) is the phase called "cell formation" (CF) [3] [4].

Cell formation (CF) is one of the most significant processes in the design of a cellular manufacturing system (CMS), which involves grouping machines into cells and grouping parts into discrete families to save costs.Material handling expenses, which include transportation costs between and within cells, are an additional essential aspect of CMS. [5] The cell formation problem is recognized to be a Non-Polynomial (NP)-hard problem [7]. Hence, developing effective machine grouping algorithms has long been a focus of attention in CMS design, leading to a broad spectrum of research [2].

This chapter will be split into three parts. First, the cell formation problem will be explained. Then, the Generalized Cubic Cell Formation Problem and its mathematical model will be laid out.

1.2 Definition of the Cell Formation Problem

1.2.1 The Cell Formation Problem

CFP is defined by its machine-item incidence matrix. The incidence matrix is a binary matrix with rows and columns representing machines and items; each cell in this matrix may have a value of 0 or 1 as seen in Table 1.1.

		Parts				
		1	5	3	4	2
Machines	2	1	1	0	0	1
	4	1	1	1	0	0
	6	1	1	1	0	0
	1	1	1	0	0	0
	3	0	0	1	1	1
	5	0	0	0	1	1
	7	0	0	0	1	1

Table 1.1: Example of an incidence matrix.

The solution to the cell formation problem is a configuration that describes the type of cells that must be constructed, the cell that each machine is assigned to, as well as the cell in which each component is influenced.

		Parts				
		1	5	3	4	2
Machines	2	1	1	🟩 0	0	🟨 1
	4	1	1	1	0	0
	6	1	1	1	0	0
	1	1	1	🟩 0	0	0
	3	0	0	🟨 1	1	1
	5	0	0	0	1	1
	7	0	0	0	1	1

Table 1.2: Example of a two-cell formation of an incidence matrix.

Table 1.2 depicts a binary machine-part incidence matrix. A vacancy is indicated by the green box, and an exceptional element is shown by the yellow box. A void indicates that, despite the fact that the machine and the part are in the same group, this machine will not process the part [8].

An exception, on the other hand, indicates that an inter-cellular movement will be required because the portion will be processed outside of the designated cell. Voids are undesirable because they make the cell less efficient. In the same way, exceptional pieces increase processing time and generate additional issues [8].

1.2.2 Generalized Cell Formation Problem

The simple CFP assumes that every component has only one path. In real-life scenarios, a component may have multiple process paths. The Generalized Cell Formation Problem (GCFP) is a CFP that looks at many different possible process paths [6].

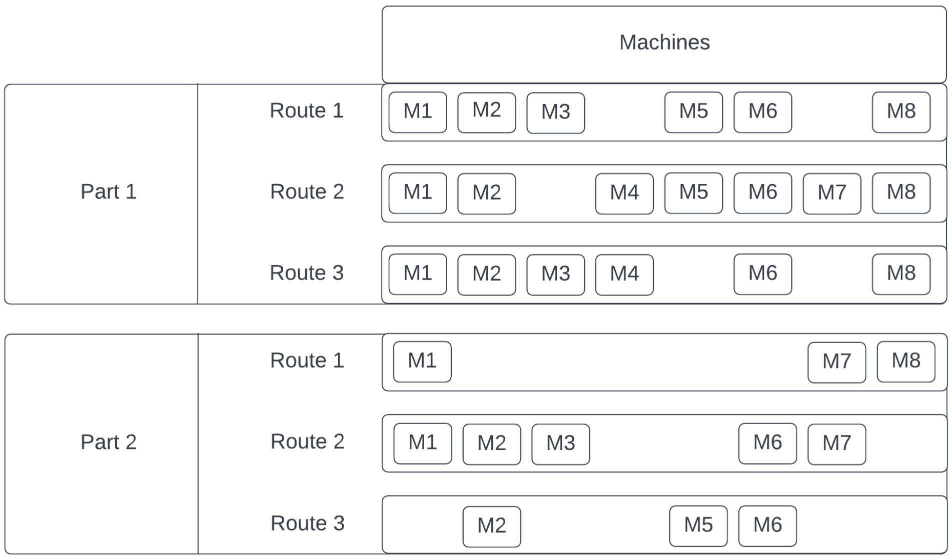


Figure 1.1: Example of two parts having multiple processing routes.

The preceding diagram (Figure 1.1) illustrates how two components can have multiple processing paths. Both the first and second components in this example have three processing plans or routes. Each row contains a number of machines that are arranged from left to right.

In GCFP, it is necessary to define the process plans or routes for the parts. Therefore, a machine is only required to produce a part if it is included in the route for producing the part. In this case, the incidence matrix also describes the order of machines that must be used to make parts [6]. “Although the formation of machine-part manufacturing cells is the essence of GT, its full benefits cannot be gained without forming "human" cells” [9].

1.2.3 Generalized Cubic Cell Formation Problem

Similar to the Generalized Cell Formation Problem (GCFP), the Generalized Cubic Cell Formation Problem (GCCFP) is a special case of the Cell Formation Problem (CFP).

However, (GCCFP) includes an additional dimension called the human dimension.

In light of the significance of personnel, it is possible that the quality of the CMS design could be improved by bringing together people whose abilities and levels of experience are comparable in order to produce part families that are analogous to one another. The problem is transformed into a GCCFP once the human factor, in addition to the part and machine dimensions, is taken into consideration [6].

Throughout the entirety of this article, this particular variant will serve as the primary focal point of our attention.

1.3 Generalized Cubic Cell Formation Problem Mathematical Formulation

The formulation and the assumptions that are presented in [6] are the ones that we go with:

1.3.1 Hypotheses and Assumptions

Research on the GCCFP takes the following hypotheses into consideration:

- The capacity of machines and operators is disregarded.
- The sum of the costs associated with the inter-cellular and intra-cellular movement of the parts constitutes the material handling cost of a particular design.
- The number of cells, as well as the maximum and minimum numbers of machines that can be contained within a cell, are all known and utilized as parameters.
- There is movement possible both inside of and between cells. When two consecutive operations are performed on the selected route of a given component in two different cells, inter-cellular movement is produced. However, intra-cellular movement occurs when two consecutive operations on a component are performed in the same processing cell.
- Each type of machine and worker is represented by a single instance.
- Using a three-dimensional matrix, the quality of each worker's treatment of each component on each machine is specified. The values of this matrix are integers ranging from 1 to 5, with 1 representing the worst and 5 representing the best possible outcome. A value of 0 indicates that a particular component cannot be worked on by a particular operator using a particular machine. These values can be approximated by conducting an investigation into the previously acquired data as well as the mistakes made by workers. When the layout of the production system is being designed for the first time, it is possible to give each worker the same quality

standard. Analyzing his credentials and previous work experience is another method for estimating the level of quality that will be delivered. Following this, a continuous evaluation can be planned in order to acquire the necessary data regarding the workers' ability to produce parts and operate machines. This information could be used for a future configuration update of the system.

- The inter-cellular movement of workers is determined by the presence or absence of workers in the processing cells.
- Several workers may be in charge of a component, but each operation on a component is given to a single worker and done on a single machine.
- Each component type has a minimum of one processing route. This part will be manufactured via a single route.
- An operator can operate multiple machines.

1.3.2 The Decision Variables

The GCCFP resolution includes four decisions that must be made:

- Y_{mk} : each machine m being assigned to a single cell k .
- Z_{wk} : the assignment of each worker w to one cell k.
- R_{pr} : the selection of a unique path r for each component p.
- X_{prsmwk} : details regarding which employee w will perform a specific task s on a specific part pr, on which machine m, and in which cell k.

$$R_{pr} = \begin{cases} 1 & if\ part\ p\ is\ processed\ according\ to\ process\ route\ r \\ 0 & otherwise \end{cases}$$

$$Y_{mk} = \begin{cases} 1 & if\ machine\ m\ is\ assigned\ to\ cell\ k \\ 0 & otherwise \end{cases}$$

$$Z_{wk} = \begin{cases} 1 & if\ worker\ w\ is\ assigned\ to\ cell\ k \\ 0 & otherwise \end{cases}$$

$$X_{prsmwk} = \begin{cases} 1 & if\ operation\ s\ of\ part\ p\ along\ route\ r\ is\ processed \\ & on\ machine\ m\ by\ worker\ w\ in\ cell\ k \\ 0 & otherwise \end{cases}$$

1.3.3 The Constants

The following symbols are used in the formulation of GCCFP:

• C	the total number of cells.
• T	the set of cells, T = 1,...,C.
• M	the total number of machines.
• P	the total number of parts.
• W	the total number of workers.
• R_p	the total number of process routes of part “p”.
• Op_{pr}	the total number of the operations in route “r” of part “p”.
• k	the index of cells, k = 1,2, ..., C.
• p	the index of parts, p =1,2, ...,P.
• m	the index of machines, m =1,2, ...,M.
• w	the index of workers, w =1,2, ...,W.
• r	the index of process routes.
• s	the index of operations within routes.
• UM	the maximum cell size.
• LM	the minimum cell size.

• CO_p	the cost of moving part “p” to an outer cell.
• CL_p	the cost of moving part “p” inside the same cell.
• CW_w	the cost of moving worker “w” from a cell to another one.
• a_{prsm}	indicates whether operation “s” in route “r” of part “p” may be processed on machine “m”.
• b_{mw}	a binary parameter indicating whether worker “w” can use machine “m”.
• c_{wp}	a binary parameter indicating whether worker “w” can process part "p".
• q_{pmw}	quality obtained for part "p" when it is processed on machine "m" by worker “w”.

1.3.4 The Mathematical Model

Equation 1.1 represents the objective function of the model. While doing so, it keeps intra-cellular material handling costs and inter-cellular movement costs to a minimum and increases the quality index of the parts that are manufactured.

Note that InterCMC is simply the result of adding the equations 1.2 and 1.3 together, which refer to inter-cellular material handling costs and inter-cellular worker movement respectively.

Equations 1.4, 1.5, and 1.6 provide the formulas for calculating the inter-cellular movement cost (InterCMC), the intra-cellular material handling cost (IntraCMHC), and the quality index, respectively. The objective of the model is to achieve a more optimal balance between these objectives.

$$\begin{cases} min & InterCMC \\ min & IntraCMHC \\ max & Quality \end{cases} \tag{1.1}$$

$$InterCMHC = \sum_{k \epsilon T} \sum_{k' \epsilon T \setminus \{k\}} \sum_{p=1}^{P} CO_p * \left[\sum_{r=1}^{R_p} \sum_{s=1}^{Op_{pr}-1} \left[\left(\sum_{m=1}^{M} \sum_{w=1}^{W} X_{prsmwk} \right) * \left(\sum_{m=1}^{M} \sum_{w=1}^{W} X_{prs+1mwk'} \right) \right] \right] \tag{1.2}$$

$$InterCMW = \left[\sum_{k=1}^{C}\sum_{p=1}^{P}\sum_{r=1}^{R_p}\sum_{s=1}^{Op_{pr}}\sum_{m=1}^{M}\sum_{w=1}^{W} CW_w * X_{prsmwk} * (1 - Z_{wk})\right] \tag{1.3}$$

$$InterCMC = InterCMHC + InterCMW \tag{1.4}$$

$$IntraCMHC = \sum_{k=1}^{C}\sum_{p=1}^{P} CI_p * \left[\sum_{r=1}^{R_p}\sum_{s=1}^{Oppr-1}\left[\left(\sum_{m=1}^{M}\sum_{w=1}^{W} X_{prsmwk}\right) * \left(\sum_{m=1}^{M}\sum_{w=1}^{W} X_{prs+1mwk}\right)\right]\right] \tag{1.5}$$

$$\text{Quality } = \sum_{k=1}^{C}\sum_{p=1}^{P}\sum_{r=1}^{R_p}\sum_{s=1}^{Op_{pr}}\sum_{m=1}^{M}\sum_{w=1}^{W} q_{pmw} \cdot X_{prsmwk} \tag{1.6}$$

Subject to :

$$X_{prsmwk} \leq R_{pr} * a_{prsm} * Y_{mk} * b_{mw} * c_{wp} \quad \forall(p, r, s, m, w, k) \tag{1.7}$$

$$\sum_{k=1}^{C}\sum_{m=1}^{M}\sum_{w=1}^{W} X_{prsmwk} = R_{pr} \quad \forall(p, r, s) \tag{1.8}$$

$$\sum_{r=1}^{R_p} R_{pr} = 1, \quad \forall(p) \tag{1.9}$$

$$\sum_{k=1}^{C} Z_{wk} = 1, \quad \forall(w) \tag{1.10}$$

$$\sum_{k=1}^{C} Y_{mk} = 1, \quad \forall(m) \tag{1.11}$$

$$\sum_{m=1}^{M} Y_{mk} \leq UM, \quad \forall(k) \tag{1.12}$$

$$\sum_{m=1}^{M} Y_{mk} \geq LM, \quad \forall(k) \tag{1.13}$$

$$X_{prsmwk}, Y_{mk}, Z_{wk}, R_{pr} \in \{0,1\} \quad \forall(p,r,s,m,w,k) \tag{1.14}$$

Equation 1.7 dictates that an operation s will only be performed on machine m by worker w within cell k if the following conditions are met:

1. The route to which s belongs is configured to produce the relevant component p ($\mathrm{R_{pr}}$).
2. The operation s requires the use of a machine m ($\mathrm{a_{prsm}}$).
3. Since machines cannot be moved between cells ($\mathrm{Y_{mk}}$), machine m is already assigned to cell k.
4. The machine m is operable by a worker w ($\mathrm{b_{mw}}$).
5. Worker w is able to process part p ($\mathrm{c_{wp}}$).

Equation 1.8 ensures that an operation s will only be performed on a single machine by a single worker in a single cell if the route to which the operation belongs is configured to produce the component in question. Equation 1.9 indicates that a single route is established for each component.

The constraint 1.10 verifies that each worker is assigned precisely to one cell. Constraint 1.11 guarantees that each machine is assigned to a single cell. Constraints 1.12 and 1.13 specify the minimum and maximum number of machines a cell may contain, respectively.

The final constraint 1.14 specifies the decision variables' logical binary requirements.

1.4 Conclusion

This chapter provides an overview of the Cell Formation Problem. Initially, we define their basic versions. Then, we define the version that will be considered in this study, the Generalized Cubic Cell Formation Problem. Lastly, a mathematical formulation of the Generalized Cubic Cell Formation Problem is presented.

Our problem falls under the category of NP-hard. These problems are algorithmically solvable but computationally insurmountable. No exact method exists for locating optimal global solutions to NP-hard problems in polynomial time. Meta-heuristics and fast approximation heuristics are the most common approaches to finding practical solutions. In our research, we will employ a multi-objective evolutionary algorithm, which is one of the most widely used methods for resolving complex, large-scale optimization problems with multiple objectives. In the following chapter, we will therefore provide an overview of Multi-Objective Evolutionary Algorithms .

CHAPTER 2 MULTI OBJECTIVE EVOLUTIONARY ALGORITHM.

2.1 Introduction

Multi-objective optimization (MOO) is an important part of the field of optimization and has a big impact in the real world because almost every optimization problem can be described in terms of multiple competing goals.
The intention was to tackle multi-objective optimization problems as they exist, whereas most prior methods concentrated on reducing many objectives to a single objective (i.e., transforming the problem into a single-objective optimization problem) [10].

Combinatorial optimization (CO) can use either exact or approximate approaches. By applying meta-heuristics and other approximate methods, we sacrifice the certainty of identifying optimal solutions in order to obtain good solutions in less time. As a result, meta-heuristics have garnered increasing attention during the past few decades.

Although computational techniques for multi-objective optimization problems (MOPs) have been known for some time, the recent application of Evolutionary Algorithms (EAs) to these issues has greatly boosted the ability to solve large-scale MOPs [11].
The research of Multi-Objective Evolutionary Algorithms (MOEAs) is expanding within the discipline of Evolutionary Computation. Schafer introduced the first MOEA, the Vector Evaluated Genetic Algorithm, in 1985 [12].

Numerous EA-based approaches for solving multi-objective optimization (MOP) problems have been published since then. To solve the Generalized Cubic Cell Formation Problem, we will employ two multi-objective evolutionary algorithms. NSGA-II and NRGA are the abbreviations for these Multi-objective Evolutionary Algorithms.

This chapter focuses on Multi-Objective Evolutionary Algorithms (MOEAs). Multi-Objective Optimization (MO) problems are briefly introduced in Section 2.2. Then, in section 2.3, we reviewed the potential solutions to these challenges, including both exact and approximation methods. The subsequent section 2.4 then introduces and describes numerous Evolutionary algorithms (EA).

2.2 Multi Objective Optimization

2.2.1 Overview

Multi-objective optimization is an integral part of optimization activities and has a tremendous practical importance since problems in the real world involve multiple competing objectives. For instance, when purchasing a car, we typically want a powerful one, but we also want to spend as little as possible, as seen in Figure 2.1 [13].

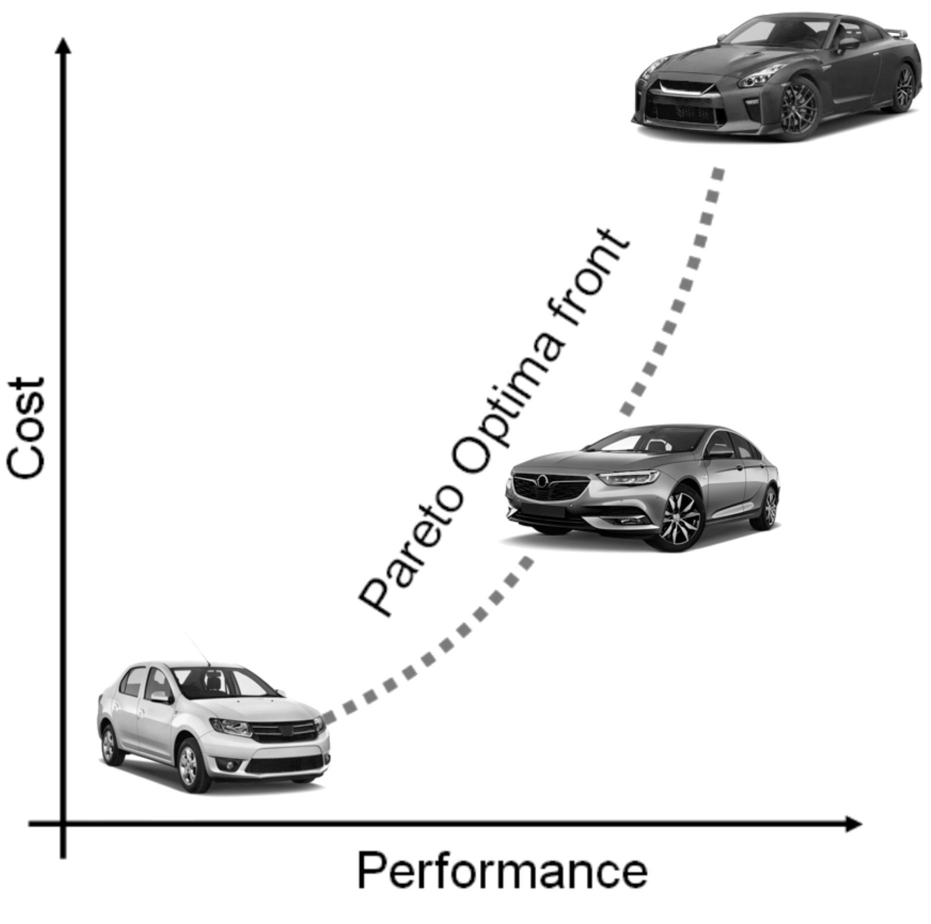


Figure 2.1: Example of a cost-performance problem.

Clearly, there is no single solution to these problems that is optimal when measured against all objectives (note that the term "solution" can also refer to an individual or a point). These problems are examples of a subtype of optimization problems known as multi-objective optimization problems (MOPs).

In general, a solution is Pareto-optimal if there are no viable alternatives [13]. which would increase some objectives (increase or decrease depending on whether it's a maximization or a minimization problem) without simultaneously decreasing at least one other objective [14].

The Pareto optimal set, named after Vilfredo Pareto [15], or the Pareto optimal front (POF), is a plot of vectors that represent multiple trade-off solutions. In this regard, the search for the optimal solution differs significantly from what we observe in the case of single-objective problems [13]. The process of resolving MOPs is referred to as multi-objective optimization (MOO).

2.2.2 Importance of MOO

Multi-objective optimization, also referred to as multi-objective programming, vector optimization, multi-criteria optimization, multi attribute optimization, or Pareto optimization, is a branch of multiple-criteria decision-making that deals with mathematical optimization problems that require the simultaneous optimization of multiple objective functions. Many scientific fields, including engineering, have used multi-objective optimization to make decisions when there are trade-offs between two or more objectives that may be at odds. In many real-world engineering applications, designers must choose between competing goals. For instance, a vehicle's performance may need to be maximized while its fuel usage and pollution emissions are minimized. A multi-objective optimization study should be carried out in these circumstances as it offers multiple solutions that represent the trade-offs between the various objective functions [19].

A compromise process design could be chosen from a set of non-dominant (Pareto) solutions produced by multi-objective optimization, which has recently gained prominence as a crucial tool for decision-making. For instance, Tokos et al.'s [21] bi-objective optimization of the water network in the brewery is just one example of the extensive research that has been done on finding practical industrial solutions for multiple objectives problems. A number of environmental measures, including the Environmental Performance Index [22] and various footprints [23], have been defined and put into use recently because, unlike the measures of economic performance, choosing environmental criteria is more or less optional. Due to the potential for a very large number of objectives, some authors have developed techniques for reducing the number of objectives so that only those that conflict are retained [24] [18].

Numerous search and optimization problems in the real world are naturally formulated as nonlinear programming problems with multiple conflicting objectives. Due to a lack of appropriate solution techniques, such problems were artificially transformed into single-objective problems and then solved. The difficulty arose due to the fact that such problems generate a set of trade-off optimal solutions (also known as Pareto-optimal solutions) as opposed to a single optimal solution. It then becomes important to find more than just one [20].

Due to the absence of a suitable optimization methodology for efficiently locating multiple optimal solutions, classical methods employ a different approach to solving these problems. They typically necessitate repeated applications of an algorithm to identify multiple Pareto-optimal solutions, and in some cases, such applications do not even guarantee the identification of any Pareto-optimal solutions. In contrast, the population approach of evolutionary algorithms (EAs) is an effective method for simultaneously discovering multiple Pareto-optimal solutions during a single simulation run [20].

2.2.3 Approaches and Methodology

Resolving MOPs provides a pareto optimal front, i.e., multiple mutually non-dominating optimal solutions. Nevertheless, users typically require only a single optimal trade-off solution from the set [13]. Finding solutions to MOPs requires, as a result, both searching and making decisions [16]. In support of this claim, the research that has been done thus far [17] identifies four primary methods as follows:

1. **No-preference method:** this approach does not utilize preference data. This method resolves a problem and provides a solution to the decision-maker.
2. **Posterior method:** also referred to as decision-making after search, this technique identifies all possible solutions in the non-dominated set and then employs the user's preferences to determine the optimal solution.
3. **Priori method:** unlike the latter method, this one incorporates preference prior to the optimization process; consequently, it will only produce a single solution, i.e., decision-making before search. This approach consistently imposes the user's bias.
4. **Interactive method:** This approach, which is a hybrid of the second and third methods and is known as decision-making during search, employs manual decision-making to refine the obtained trade-off solutions and thus directs the search.

Since it is less subjective, the second method is generally preferred by the research community [13] and will be our main focus as well.

The phrase "method for generating Pareto optimal solutions" may also be used to describe a posteriori methods. After the Pareto optimal set or at least a portion of it has been formed, the choices are offered to the decision maker, who chooses the most favored option. The disadvantages here are that the generation procedure is typically computationally expensive and occasionally at least somewhat challenging [17].

Of-course all of these Approaches and Methodologies belong to two distinct categories, which are Exact Methods and Approximate Methods.

2.3 Exact and Approximate Methods

2.3.1 Exact Methods

In the past 48 years, numerous methodologies with distinct origins and implementations have developed. There exist numerous exact algorithms and algorithmic variants capable of obtaining the entire non-dominated set for a multi-objective integer problem [51]. Some of them are mentioned below.

Branch-and-bound and branch-and-cut algorithms are the most prevalent and fundamental algorithmic solution strategies for single objective integer programming. Several authors have also extended the concept of implicitly searching an enumeration tree to the multi-objective case. Bitran and Rivera (1982) proposed a branch-and-bound algorithm that includes computation of bounds, fathoming rules, feasibility testing, branching, and dominance for multi-objective binary problems [45]. Sourd and Spanjaard (2008) presented a general multi-objective branch-and-bound framework based on lower and upper bound sets [50].

The epsilon-constraint scalarization is one of the most widely used scalarization methods in multi-objective optimization. Although initially developed by Haimes et al. (1971) [46] to find a single weakly efficient solution, this technique can be used iteratively to find the entire non-dominated set of multi-objective integer optimization problems. Typically,epsilon-constraint methods can be easily implemented in a rudimentary manner while offering numerous and substantial opportunities for performance enhancement via constraint selection and bound shifting [51].

"AUGMECON" [43] [44] refers to an augmented epsilon-constraint method. In contrast to the traditional epsilon-constraint method, slack or surplus variables are introduced to transform the objective function constraints into equality constraints. AUGMECON2 [47] [48] [49] is an upgrade to AUGMECON. Using a so-called bypass coefficient, the information obtained from the slack or surplus variables in each iteration is incorporated here. This bypass eliminates unnecessary iterations that produce the same nondominated images as previous iterations [51].

2.3.2 Augmecon-R

AUGMECON-R [39] is a robust variant of the augmented epsilon-constraint algorithm for solving multi-objective linear programming problems. It takes advantage of the flaws in AUGMECON 2, one of the most widely used improvements to the method. These flaws can be summed up as the ineffective handling of the true nadir points of the objective functions and the absence of a robust system for determining the true nadir points. Notably, as more objective functions are added to a problem, the amount of time required to apply it increases significantly. Motivating the development of AUGMECON-R was the need to overcome these deficiencies. Figure 2.2 below shows the process of AUGMECON-R algorithm.

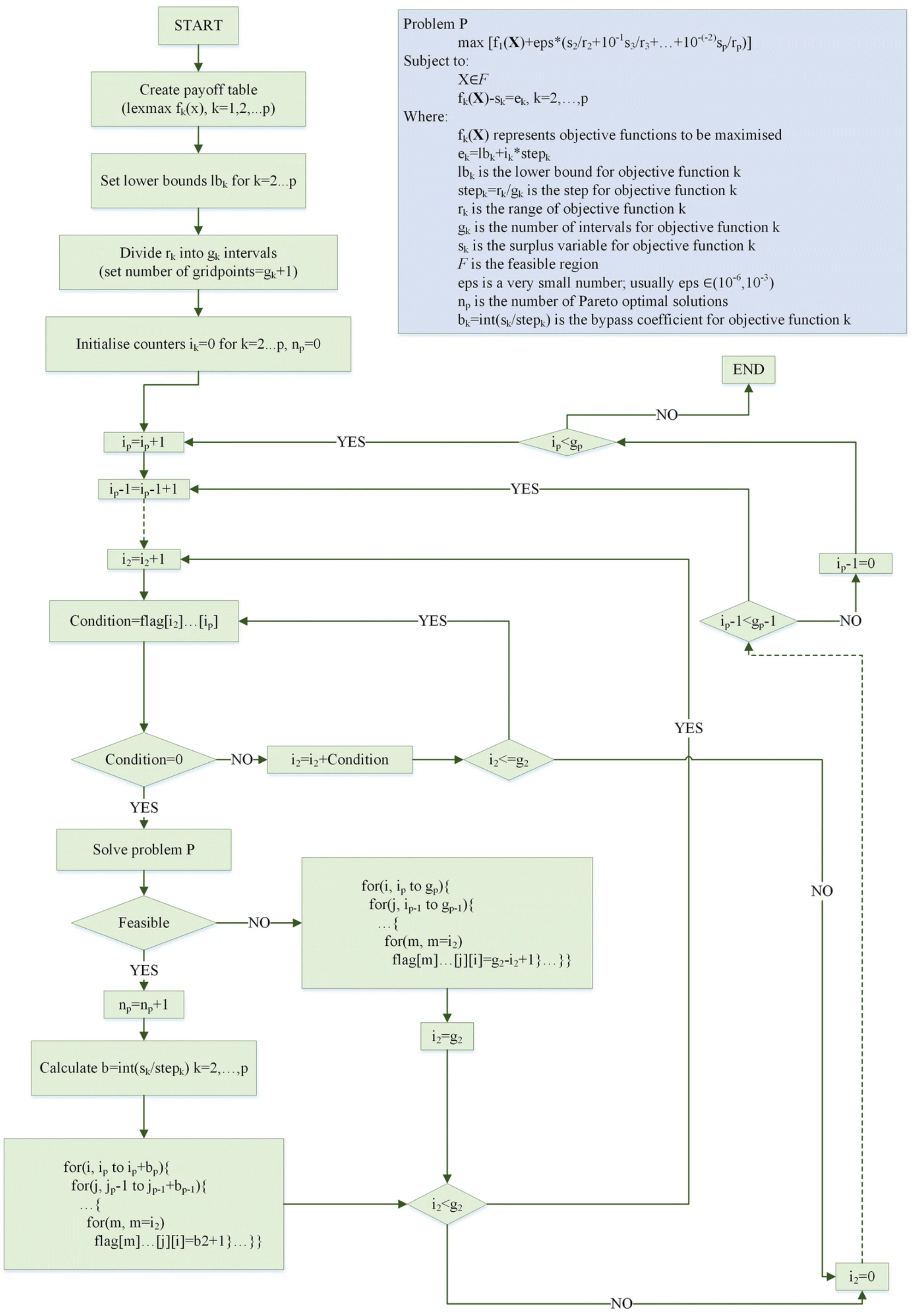

START
Create payoff table (lexmax $f_k(x)$, k=1,2,...p)
Set lower bounds lb_k for k=2...p
Divide r_k into g_k intervals (set number of gridpoints=g_k+1)
Initialise counters i_k=0 for k=2...p, n_p=0
Problem P
max $[f_1(\mathbf{X})+eps*(s_2/r_2+10^{-1}s_3/r_3+\ldots+10^{-(-2)}s_p/r_p)]$
Subject to:
$X \in F$
$f_k(\mathbf{X})-s_k=e_k$, k=2,…,p
Where:
$f_k(\mathbf{X})$ represents objective functions to be maximised
$e_k=lb_k+i_k*step_k$
lb_k is the lower bound for objective function k
$step_k=r_k/g_k$ is the step for objective function k
r_k is the range of objective function k
g_k is the number of intervals for objective function k
s_k is the surplus variable for objective function k
F is the feasible region
eps is a very small number; usually eps $\in(10^{-6},10^{-3})$
n_p is the number of Pareto optimal solutions
$b_k=int(s_k/step_k)$ is the bypass coefficient for objective function k
END
NO
$i_p=i_p+1$
YES
$i_p<g_p$
i_p-1=i_p-1+1
YES
$i_2=i_2+1$
i_p-1=0
Condition=flag[i_2]…[i_p]
YES
i_p-1<g_p-1
NO
YES
Condition=0
NO
$i_2=i_2$+Condition
$i_2<=g_2$
YES
Solve problem P
for(i, i_p to g_p){
for(j, i_{p-1} to g_{p-1}){
…{
for(m, m=i_2)
flag[m]…[j][i]=g_2-i_2+1}…}}
NO
Feasible
NO
YES
$n_p=n_p+1$
$i_2=g_2$
Calculate b=int(s_k/$step_k$) k=2,…,p
for(i, i_p to i_p+b_p){
for(j, j_p-1 to j_{p-1}+b_{p-1}){
…{
for(m, m=i_2)
flag[m]…[j][i]=b2+1}…}}
$i_2<g_2$
i_2=0
NO

Figure 2.2: Flowchart of the AUGMECON-R algorithm [39].

2.3.3 Limitations of Exact Algorithms

Combinatorial optimization problems are frequently represented using integer programming [38], as is the case here. However, these problems are frequently extremely difficult to solve, as evidenced by the fact that a large number of them are NP-hard [15]. Numerous solution techniques for tackling NP-hard integer and combinatorial optimization problems have been proposed due to their difficulty and immense practical significance.

The optimal solution to the problem is guaranteed to be found by exact algorithms. However, keep in mind that the solution space becomes more complex as the problem size increases. In turn, this may result in a slowdown of the exact algorithms.
Exact algorithms that can solve problems to optimality and advancements in computational power make exact algorithms a very appealing solution method. However, achieving an acceptable balance between solution time and quality cannot be attained using exact algorithms. Depending on the problem at hand and any integrality restrictions or constraints placed on all or some of the variables, integer and combinatorial optimization problems either maximize or minimize functions of numerous variables. Many practical issues, such as those of crew scheduling, production planning, Internet routing, packaging and cutting, and so on, fall under this general category of problems [41].

2.3.4 Approximate Methods

There are primarily two categories of algorithms that can be used: exact algorithms and approximate algorithms. For each instance of a combinatorial optimization problem with a finite size, exact algorithms are guaranteed to find the optimal solution and to prove its optimality within an instance-dependent, finite run-time. In the event that optimal solutions cannot be efficiently computed in practice, the only option left is to make a compromise between optimality and efficiency. To put it another way, the assurance of obtaining solutions that are optimal can be compromised in order to achieve the goal of obtaining very good solutions in a polynomial amount of time. Heuristic methods, also known simply as heuristics, are a category of approximate algorithms that are designed to accomplish the aforementioned objective [41].
There are two main methods that are considered approximate:

Heuristics: Heuristics are a class of approximate algorithms created with a focused goal in mind—to address a particular issue. They are used to arrive at an acceptable solution by following a sequence of steps. However, there is no assurance that the optimal solution will be found [38].

Meta-heuristics: are general-purpose algorithms that can be used to solve nearly any optimization issue. They could be thought of as higher level general methodologies that serve as a framework for developing underlying heuristics [38].

There are numerous meta-heuristics that draw inspiration from biological processes like ant colony optimization (ACO) and evolutionary algorithms (including genetic algorithms, GA). For example, the ACO meta-heuristic is based on how real ants communicate and work together to find the shortest routes from their nest to food sources. While evolutionary algorithms are based on Darwin's natural selection principles.

2.4 Evolutionary Algorithms

2.4.1 Definition

Evolutionary algorithms (EAs) are currently among the most in-demand tools for search, optimization, machine learning, and the resolution of design problems. These algorithms seek solutions to complex problems by simulating evolution. There are numerous distinct varieties of evolutionary algorithms. Genetic algorithms and evolution strategies are two of the earliest evolutionary algorithm types [28].

Under the direction of John Holland and his students, genetic algorithms were created in the US. The importance of selection, crossover, and mutation acting on a genotype that is decoded and assessed for fitness is heavily emphasized in this tradition. Crossover, also known as recombination, is prioritized over mutation [28].
More straightforward representations are typically used in evolution strategies [29]. This time, emphasis is placed on mutation rather than recombination. For optimization, techniques from both genetic algorithms and evolution strategies have been used. While the majority of the research in evolution strategies has concentrated on optimization [32], [33], [34], genetic algorithms have long been seen as multipurpose tools with applications in search, optimization, design, and machine learning [30], [31]. These two disciplines have historically interacted, and numerous new algorithms freely incorporate ideas from both traditions.

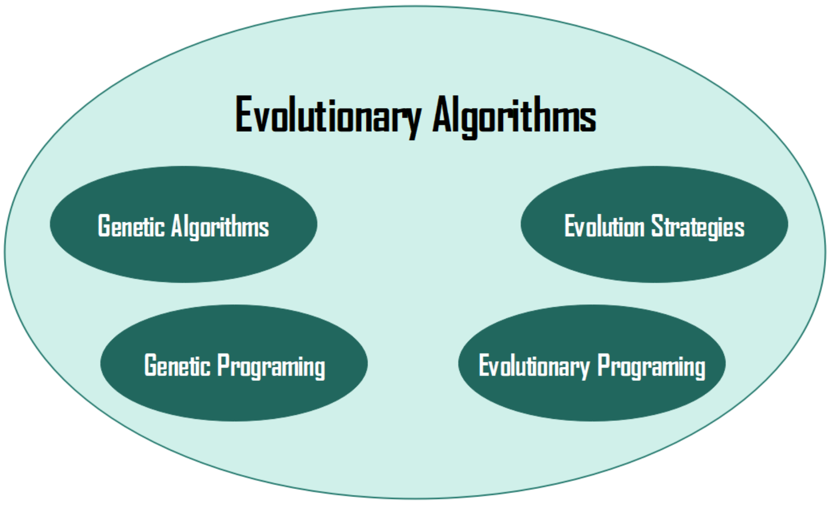


Figure 2.3: Classification of EAs.

Additionally, a crucial new area within evolutionary algorithms has emerged: genetic programming [35], [36]. Genetic programming has been developed specifically as an evolutionary method for automatic programming and machine learning. Design applications have also demonstrated their significance. Another sub-field of evolutionary computing is evolutionary programming. Evolutionary programming has its origins in the 1960s [37] but was dormant for many years until its rebirth in the 1990s [38] in a form that is highly similar to evolution strategies.

We will delve deeper into the specifics of Genetic Algorithms (GAs) for the benefit of our work and to better understand how they can be used.

2.4.2 Genetic Algorithm

Genetic Algorithms (GA) are stochastic search methods that combine two main search strategies: exploiting better solutions and exploring the global search space. These algorithms are based on Darwin's natural selection principles and natural genetics. The principles of Darwin's theory of natural selection and the laws of natural genetics serve as the foundation for these algorithms. The two main goals of the research conducted by John Holland and his students are:

1. to systematically abstract and describe the adaptive processes of natural systems.
2. To develop software for artificial systems that retains the fundamental mechanisms of natural systems.

This methodology has yielded significant discoveries in both natural and artificial systems science. In [68], Goldberg provided a primer on GAs and outlined their typical structure.
Using genetic algorithms , numerous optimization problems in a variety of fields that are difficult to address with conventional mathematical programming have been successfully resolved.

In the sections that follow, key GA terminology and concepts are introduced:

- **Populations and Generations:**
  A set of chromosomes constitutes a population. GA begins with a set of randomly created individuals (chromosomes). This set is called the "initial population". Generations are the iterations of genetic algorithms. Each iteration entails picking individuals with closely related traits and recombining them until a new generation is generated to replace the previous one [69].
- **Fitness:**
  The fitness function is the objective function that identifies the goal of optimization. For each individual, it denotes their "goodness" or "badness."
- **Parents and children:**
  Chromosome selection involves picking individuals using a probabilistic method from one generation to the next [69]. Being selected for crossover and the production of new chromosomes known as children or offspring is more likely for individuals with high fitness values. The crossover rate—also known as the crossover probability—determines the crossover's likelihood to occur. It includes a haphazardly chosen subset of the parent chromosome crossover sites, where the expected mixing of the parents' genetic material should take place.

- **Elitism:**
  In order for GA to improve its performance, it is imperative that the best individuals continue to reproduce. Nevertheless, such individuals can be lost if they are destroyed through the use of crossover or mutation operators. Consequently, the initial best chromosome, or the best few, are replicated in the new population [70].

- **Genes and Chromosomes:**
  The gene is the GA's primary constituent. A chromosome is a sequence of genes. Chromosomes can be encoded as binary strings, real number strings, and so forth.

- **Mutation:**
  Numerous new points are introduced into the search space through this procedure. It prevents a less-than-ideal solution from being chosen during a period of aggressive selection. Simply put, it delays the point at which the solution converges to a local minimum. It's accomplished by performing a series of mutations on specific chromosomes at extremely low rates of probability (mutation rate).

Six fundamental tasks must generally be accomplished by a genetic algorithm [71]:

Incorporating solution components into genetic code. Construct a chromosome by stringing together genes. Generate a set of specific chromosomes, typically at random, to establish a starting population. Assess and assign fitness values to individuals within the population. Perform reproduction by selecting population members based on their fitness levels. Perform Crossover and mutation in order to produce members of the subsequent generation.

Consequently, a GA is an iterative optimization technique that simulates the adaptation and evolution of a single species of organism. Using a system for chromosomal mapping, the GA begins with a large number of potential design configurations. The range of possible configurations is constrained by the constraints of the problem and the method for encoding all configuration information into the chromosome [69], [72]. Figure 2.4 depicts an example of a typical GA.

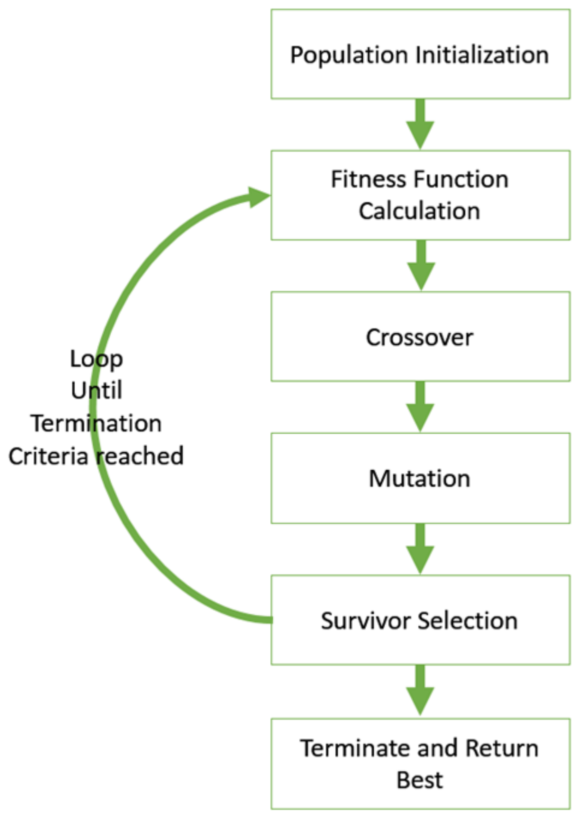

Population Initialization
Fitness Function Calculation
Crossover
Mutation
Survivor Selection
Terminate and Return Best
Loop Until Termination Criteria reached

Figure 2.4: The fundamental operation of a genetic algorithm.

The GA chooses a set of configurations, almost always at random, to initiate the optimization process. As in biology, this group is referred to as the initial population. The GA evaluates the performance of each individual in the population using a cost function that compares the individual's performance to the desired or optimal performance and returns to the GA a single number representing its fitness. As in the evolutionary process of "survival of the fittest," high-quality strings mate and produce offspring while inferior strings are eliminated. from the populace [71]. There are numerous ways to produce offspring, each of which is essentially a way of combining information from two or more parent chromosomes to produce a child with the potential to outperform its parents. With successive generations, the quality of the individuals continually improves, and an optimal solution is attained. Those who excel will produce many descendants, while those who fail will leave no descendants behind. On average, a satisfactory solution is reached after a number of generations [73].

2.4.3 GA Operators

The typical optimization process is divided into three distinct stages based on the prerequisites for a genetic algorithm that were previously discussed [69].

2.4.3.1 Initiation

Initiation refers to the process of adding chromosomes or encoded parameter strings to the initial population, which is typically chosen at random. Coding consists of a mapping from parameter space to chromosome space [73].

Individual genes are represented through the encoding process. The procedure may be carried out with bits, numbers, trees, arrays, or other objects. The encoding largely depends on the solution to the problem [74]. The mapping of object variables to a string code is represented by an encoding function. the decoding function enables the mapping of string code to its corresponding object variable.

Formulation or selection of an appropriate fitness function that determines the selection criterion for specific problems is a significant challenge in optimization. Using the simplest form $F = A - y$, where A is a large constant ($A = 0$ is sufficient if the fitness is not required to be non-negative), and $y = f(x)$, one can easily create a fitness function for genetic algorithms that minimize a function. Prior to minimizing the objective function $f(x)$, the goal is to maximize the fitness function. A fitness function $F = 1/f(x)$ can also be defined for a minimization problem, but it may singularize if $f \to 0$. A fitness function can be defined in a variety of ways [75]. The right configuration of the fitness function will guarantee effective selection of solutions with higher fitness. Incorrect or meaningless solutions may be the outcome of a poor fitness function.

2.4.3.2 Reproduction

Three main operations make up this system: crossover, mutation, and selection. The following section discusses these operators.

Simply selecting the top candidates for crossover constitutes the selection process. By taking into account these individuals' fitness values, which is a gauge of "goodness," it seeks to capitalize on their positive traits.

Typically, the crossover operator is the primary operator, serving only as a mechanism for introducing diversity into the population. The encoding schemes range from binary to real numbers.

The Mutation is a background operator that generates random, spontaneous mutations in multiple chromosomes. Modifying one or more genes is a straightforward way to induce mutation. In GA, mutation plays a crucial role in either replacing the genes lost from the population during the selection process or introducing genes that were absent from the initial population. The mutation probability is expressed as a proportion of the total number of genes within a population. The mutation probability determines the likelihood of introducing new genes into a population for testing. If it is insufficient, many genes that would have been advantageous are never tested. Nonetheless, if it is too high, there will be a great deal of random perturbation, the offspring will lose resemblance to their parents, and the algorithm will lose the ability to learn from the search's past [76].

2.4.3.3 Substitution

We must introduce the new offspring solutions to the parental population after they have been produced using crossover and mutation. We can approach this in a variety of ways. In light of the fact that the parent chromosomes have already been chosen based on their fitness, we anticipate that the offspring (which includes parents who did not undergo crossover) will be among the population's fittest individuals. Consequently, we would expect the average fitness level of the populace to gradually increase. Several of the most prevalent methods are outlined below [77].

- **Delete-all:** This technique deletes the entire current population and replaces them with the same number of newly created chromosomes. This is most likely the most typical method. Due to its relative ease of application, it will be the preferred method for the vast majority of people. In contrast to those listed below, it does not require any parameters.
- **Steady-state:** This method eliminates N old individuals and replaces them with N new individuals. The number of items to be deleted and replaced, N, is a parameter of this deletion technique. Choosing which individuals to remove from the current population is a further consideration for this method. Do you delete the worst individuals, select them at random, or delete the parents' chromosomes?
- **Steady-state-no-duplicates:** This technique is identical to steady-state, except the algorithm ensures that no duplicate chromosomes are added to the population. This increases the computational load, but may allow for a greater search space to be explored.

2.4.3.4 Stopping Criteria

In evolutionary modeling, the stopping criterion is an important factor to consider. Early termination may result in inadequate solutions, whereas late termination may result in a significant time investment. If any of the following conditions are met [78], the proposed GA is terminated:

- Limit on the number of generations;
- Attained an appropriate level of fitness;
- The number of generations that could have passed without replacing the most fit chromosome was reached.

2.4.4 Multi-objective Evolutionary Algorithms

The goal of solving optimization problems with multiple (often competing) objectives is typically very challenging. The first extensions and applications of evolutionary algorithms (EAs) were made in the middle of the 1980s in an effort to stochastically solve issues belonging to this general class. Numerous multi-objective EA techniques have been put forth and used in numerous scientific and engineering applications over the past years [40].

The problem of maximizing or minimizing a set of competing objective functions while taking into account a number of constraints is known as multi-objective optimization (MOO). There isn't a single solution for multi-objective optimization that fully maximizes or minimizes each objective [63]. This occurs as a result of the fact that the problem's various objective functions frequently clash. Because of this, the goal of MOO is to identify the Pareto front of effective solutions that allow for a trade-off between the various goals [64], [65]. Due to the population-based search strategy and the straightforward selection strategy, EAs are ideally suited for extension into the domain of multi-objective optimization problems. Instead of having to run the algorithm multiple times, as is the case with conventional mathematical programming techniques, we are able to find multiple members of the Pareto optimal set in a single run thanks to the ability of EAs to deal with populations of potential solutions simultaneously. Additionally, whereas discontinuous and concave Pareto fronts are known to be problematic for mathematical programming techniques [67], [66], EAs are less sensitive to the shape or continuity of the Pareto front. There are two objectives to be achieved during multi-objective optimization.

The solutions should, on the one hand, cover the entire Pareto-front and, on the other hand, should be as close to the global Pareto-optimal front as possible. The first objective is frequently attained through elitism by substituting Pareto front members for random individuals. By penalizing individuals who are too close to one another (Fitness Sharing), the second objective can be accomplished [40].

The conventional evolutionary algorithms, according to Coello [62], are ineffective at handling multi-objective optimization problems for two reasons:

1. If run for a sufficient number of iterations, evolutionary algorithms will converge to a single solution due to stochastic noise. Therefore, it is necessary to disable the selection mechanism so that different solutions (those that are not dominated) are maintained in the population of an evolutionary algorithm.
2. All non-dominant solutions should be sampled at the same rate during the selection phase, given that all non-dominant solutions are of equal quality.

Using EAs, researchers have developed multiple methods for solving multi-objective optimization problems in recent years. The first implementation of a multi-objective evolutionary algorithm (MOEA) occurred in the middle of the 1980s [61]. Since then, significant research has been conducted in this field, which is now known as evolutionary multi-objective optimization. The following provides a brief summary of the most prominent MOEAs:

- **Vector Evaluation Genetic Algorithm (VEGA):** In the middle of the 1980s, Schaffer and Grefenstette, introduced the Vector Evaluation Genetic Algorithm (VEGA) [61], the first implementation of a multi-objective evolutionary algorithm (MOEA). In reality, VEGA consisted of a straightforward genetic algorithm with a modified selection mechanism. Schaffer claims that a drawback of the VEGA approach is the local non-dominance of the generated solutions, which is limited to the current population. Schaffer drew attention to a different issue in genetics known as "speciation," which is the evolution of "species" within a population that perform exceptionally well in one area without taking into account the other areas. Consequently, the VEGA approach tends to reject individuals whose performance is acceptable, perhaps above average, but not exceptional for any of the objective functions. This tendency of VEGA to reject individuals with acceptable performance for any objective function is VEGA's most serious flaw, according to Schaffer, who characterized it as "middling".

- **Niched Pareto Genetic Algorithm (NPGA):** Horn, Nafpliotis, and Goldberg developed NPGA [60]. Utilizing Pareto domination ranking and fitness sharing (or niching), NPGA expands the traditional GA to include multiple objectives. The primary benefits of this method are its adaptability to a wide variety of multiobjective optimization problems and its capacity to search non-linear and discontinuous search spaces without requiring continuous first and second derivatives.

- **Nondominated Sorting Genetic Algorithm (NSGA):** Srinivas and Deb [56] developed the Nondominated Sorting Genetic Algorithm (NSGA). Goldberg proposed a nondominated sorting procedure to counter VEGA's tendency to favor Pareto-optimal solutions. Schaffer, VEGA's creator, was aware of VEGA's partiality for Pareto-optimal solutions. NSGA eliminates the bias in VEGA, distributing the population across all Pareto-optimal regions.

- **Nondominated Sorting Genetic Algorithm II (NSGA-II):** Deb and co propose NSGA-II [52], a fast non-dominated sorting-based multi-objective evolutionary algorithm with $O(MN2)$ complexity. NSGA-II implements a selection operator that generates a mating pool by combining the parent and offspring populations and selecting the optimal N solutions in terms of fitness and spread.

- **Non-dominated Ranking Genetic Algorithm (NRGA):** al jadaan and co [58]. The Non-dominated ranked genetic algorithm (NRGA), which was proposed for use in solving multi-objective optimization problems, was tested on five benchmark test problems. The results demonstrate that NRGA was able to converge significantly faster than NSGA-II while maintaining a somewhat better spread of solutions than NSGA-II. Most of the challenges that non-dominated sorting and sharing evolutionary algorithms face are largely resolved by NRGA.

Given that these two algorithms, NSGA-II and NRGA, are the primary focus of this work, we will provide a thorough explanation of their specifics in order to facilitate comprehension of the following chapter's material.

2.4.5 Non-dominated Sorting Genetic Algorithm II

The Non-Dominated Sorting Genetic Algorithm (NSGA-II) is a robust decision space exploration engine based on the Genetic Algorithm (GA) for resolving Multi-objective Optimization (MOP) problems. Deb et al. [53] first proposed it in 2000 at the "International Conference on Parallel Problem Solving from Nature."

Since its 2002 publication in the journal IEEE Transactions on Evolutionary Computation [52] as a full-length research article, it has been cited more than 25685 times, according to IEEE Xplore. Moreover, according to Google scholar, it has been cited over 42007 times. This evidence is sufficient to demonstrate the widespread use of NSGA-II for solving MOPs. During its 21-year existence, NSGA-II has been applied to a vast array of MOPs with continuous and discrete variables.

NSGA-II is one of the most popular multi-objective optimization algorithms with three distinctive features: a fast non-dominated sorting approach, a fast crowded distance estimation- procedure, and a straightforward crowded comparison operator [42].

NSGA-II is an enhanced version of the non-dominated sorting genetic algorithm (NSGA) [55], which has been criticized by researchers for its shortcomings, including the absence of elitism, the need to define sharing parameters for diversity preservation, and its high computational complexity. NSGA-II's design, on the other hand, exhibits elitism and does not require any sharing parameters. It uses the crowding distance operator for the mechanism of diversity preservation. In addition, it is computationally efficient and lives up to the name "Fast Elitist NSGA-II." NSGA-II has a maximum overall complexity of O(MN2), where M is the number of objective functions and N is the population size [54].

NSGA-II begins with the generation of an initial population Pt of size N. Then, a new population Qt is created after performing crossover and mutation operations on the population Pt . Then, the populations Pt and Qt are merged to form the new population Rt, and the non-dominated sorting procedure is carried out on Rt. The members of the Rt population are then ranked according to their levels of non-denomination [54].

The subsequent step is to select N members from Rt to form the subsequent population, Pt+1. If the size of the first front is greater than or equal to N, then only N members from the least crowded region of the first front are chosen to form Pt+1. In contrast, if the size of the first front is less than N, the members of the first front are directly transferred to the next generation, and the remaining members are taken from the second front's least crowded region and added to Pt+1. If the size of Pt+1 is still smaller than N, the same procedure is repeated for the subsequent iteration. There will be successive fronts until the size of Pt+1 equals N. The populations Pt+2, Pt+3, Pt+4,... for subsequent generations are generated using the same method until the stopping criteria are met [54]. The operation of NSGA-II is shown in Figure 2.5 .

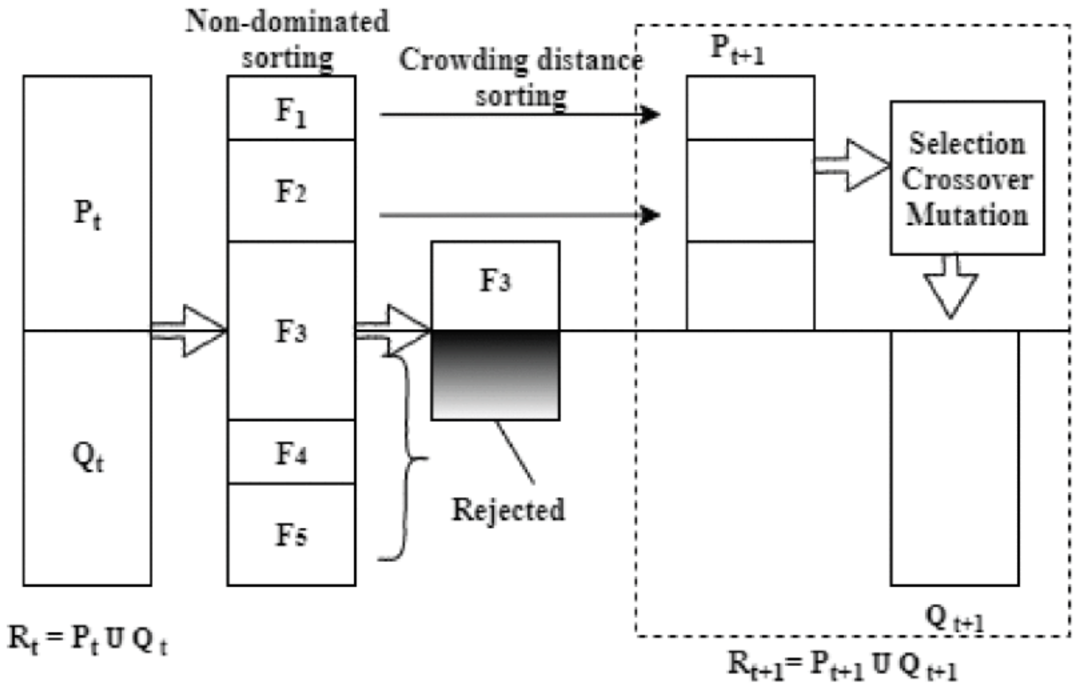


Figure 2.5: The fundamental operation of NSGA 2 [54].

2.4.6 NSGA-2 Structure

The NSGA-II philosophy is founded on four fundamental principles: Non-Dominated Sorting, Elite Preserving Operator, Crowding Distance, and Selection Operator. These are briefly described in the sections that follow [54].

2.4.6.1 Non-Dominated Sorting

Using the concept of Pareto dominance, the population members are sorted in this procedure. The initial step in the non-dominated sorting procedure is to assign the highest rank to the non-dominated members of the initial population. These members are then placed in the first front and the initial population is reduced by one. The non-dominant sorting procedure is then applied to the remaining population members. In addition, the non-dominant members of the remaining population are assigned the second rank and positioned in the second front. As depicted in Figures 2.6 and 2.5, this process continues until the entire population is divided into different fronts based on their ranks.

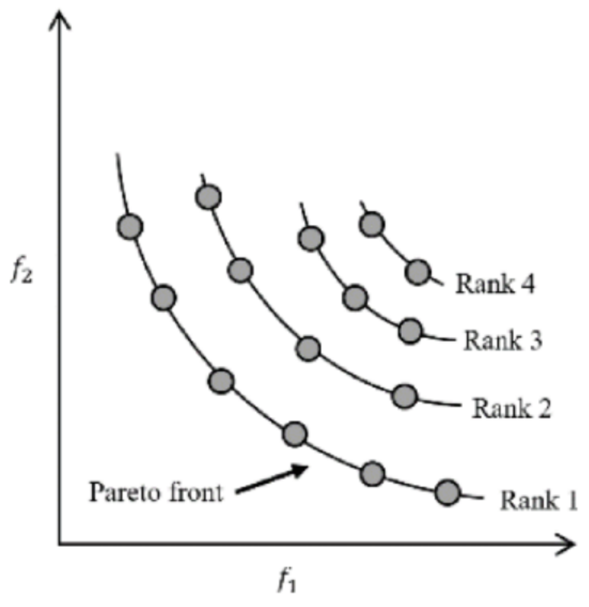


Figure 2.6: Non-Dominated sorting.

2.4.6.2 Elite-Preserving Operator

A strategy that preserves the elite solutions of a population by transferring them directly to the following generation. In other words, the non-dominant solutions found in each generation are passed on to subsequent generations until they are dominated by other solutions.

2.4.6.3 Crowding Distance

The crowding distance is computed to approximate the density of solutions surrounding a specific solution. It represents the average distance between two solutions on either side of the solution along each of the objectives. Comparing two solutions with varying crowding distances, the solution with the greater crowding distance is deemed to be in a less crowded region. As shown in Figure 2.7, the crowded distance of the i^{th} solution is the average side length of the cuboid. If f_j^{min} is the j^{th} value of an objective function for the i^{th} individual and f_j^i and f_j^{min} are the maximum and minimum values of the j^{th} objective function across all individuals, then f_j^{min} is the minimum value of the j^{th} objective function. Then, the crowding distance of the i^{th} individual is defined as the average distance between the two closest solutions on either side, as shown in Equation 2.1.

$$cd(i) = \sum_{i=1}^{k} \frac{f_j^{i+1} - f_j^{i-1}}{f_j^{\max} - f_j^{\min}} \tag{2.1}$$

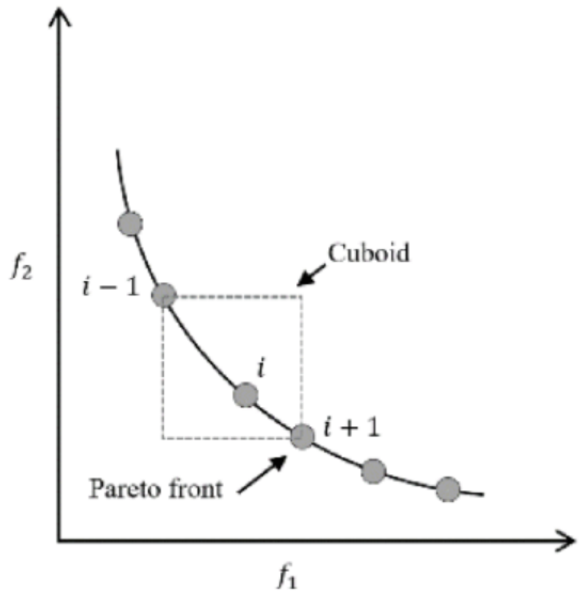


Figure 2.7: Crowding distance calculations.

2.4.6.4 Selection Operator

The population for the next generation is chosen using a crowded tournament selection operator, which takes into account the population members' ranks and crowding distances. The criterion for selecting one individual out of every two for the next generation is as follows:

1. If two members of a population have different ranks, the one with the higher rank is chosen for the next generation.
2. If both members of a population have the same rank, the individual with the greater crowding distance is chosen for the next generation.

2.4.7 Non-dominated Ranked Genetic Algorithm

Non-dominated Ranking Genetic Algorithm is referred to as NRGA. In order to optimize non-convex, nonlinear, and discrete functions, a new population-based multi-objective evolutionary algorithm based on non-dominated ranking has been successfully developed by al jadaan and Co [58]. They researched multi-objective sorting algorithms that didn't rely on dominant algorithms. They identified three issues with these algorithms [57].

1. The computational complexity (M: the number of targets, N: the population size) was $O(mn2^m)$.
2. Ineffective elitism.
3. They must specify parameters for the division process.

They created a new approach called NRGA (non-dominated ranking genetic algorithm) by fusing the ranking-based roulette wheeling algorithm and the Pareto-based population ranking algorithm in response to the issues with their prior methods. Their suggested algorithm resolves the three issues with the earlier methods. A two-layer ranking based on the selection operator of roulette wheeling is provided in this combination, which randomly chooses the new generation from the parent generation based on the selection of the best solutions (in terms of fit and extent). Compared to other multi-objective evolutionary algorithms, this algorithm is typically capable of achieving better scalability of the solutions at the Pareto boundary and earlier convergence at the Pareto optimal boundary [57].

Algorithm 1 NRGA

```
1:  Initialize Population P
2:     { Generate Random population $-$ size $N$
3:        Evaluate Objective Values
4:           Assign Rank (level) Based on Pareto dominance sort }
5:  Generate Child Population Q
6:     { Ranked Roulette Wheel Selection
7:        Crossover and Mutation }
8:  for $i = 1$ $to$ $g$ do
9:      for each member of the combined population $(P \cup Q)$ do
10:         Assign Rank (level) based on Pareto - sort
11:         Generate sets of non-dominated fronts
12:         Calculate the crowding distance between members on each front
13:     end for
14:     (elitist) Select the members of the combined population
        based on least dominated $N$ solution to make the population of the next
        generation. Ties are resolved by taking the less crowding distance.
15:     Create next generation
16:        { Ranked Roulette Wheel Selection
17:           Crossover and Mutation }
18: end for
```

The algorithm 1 demonstrates how easy-to-understand and simple NRGA is. A combined population $P \cup Q$ is first created. The total population is $2N$ in size. Following that, the combined population is sorted.

In line with non-domination. Elitism is guaranteed because the combined population includes every member of the past and present population. This process will pick N solutions from a possible $2N$.

The N-size new population is used for selection. The solutions belonging to the best non-dominated set $Front_1$ have the highest probabilities of selection in the two tiers ranked based roulette wheel selection [57], where one tier is used to select the front and the other to select a solution from the front. As a result, solutions from set $Front_2$ are chosen less frequently than those from set $Front_1$, and so forth. Then a new population P of size N is created by applying crossover and mutation. By ranking the solutions (in the same front) according to their crowding distance, the second tier of ranked based roulette wheel selection introduces diversity among non-dominated solutions. Higher probabilities will be assigned to the solutions with the smallest crowding distance [58].

In conclusion, NRGA is similar to NSGA-II in all respects, with the exception of the selection process, where NRGA employs a ranked-based roulette wheel.

2.4.8 Ranked Based Roulette Wheel Selection

The authors of [57] employ a modified roulette wheel selection algorithm in which each individual is assigned a fitness value equal to its rank in the population; the individual with the highest rank has the highest probability of being chosen (in the case of maximization).
The probability is computed as shown in the following equation 2.2:

$$Pi = \frac{2 \times Rank}{N \times (N+1)} \tag{2.2}$$

Where N represents the number of people in the population. Keep in mind that the crowding distance of individuals in a front determines their rank, while the non-dominated rank of a front determines the front's overall position.

2.5 Conclusion

This chapter is concerned with Multi-Objective Evolutionary Algorithms . Initially, we presented Multi-Objective Optimization . Then, we discussed the multitude of available resolution strategies; exact and approximate, leading to the introduction and explanation of various Evolutionary algorithms , which together with Multi-Objective Optimization constitute Multi-Objective Evolutionary Algorithms .

In the following chapter, we will apply the NSGA-II and NRGA to the problem presented in Chapter 1, the Generalized Cubic Cell Formation Problem, and compare our results.

CHAPTER 3

SOLVING THE GENERALIZED CUBIC CELL FORMATION PROBLEM.

3.1 Introduction

In the previous chapters, we defined the Generalized Cubic Cell Formation Problem and introduced the Multi-Objective Evolutionary Algorithms (MOEA), focusing on the NSGA-II and NRGA algorithms. In this chapter, we'll demonstrate how we implemented these algorithms in GCCFP. Then, we compare the results of NSGA-II and NRGA with those obtained by Augmecon-R and to one another which are introduced in the previous chapter. This chapter is therefore structured as follows:

Section 3.2 presents the solution's adopted representation and evaluation. Section 3.3 describes the solution's implementation in depth, including a description of the proposed NSGA-II and NRGA. We show the computational results and instances in section 3.4. The software is presented in section 3.5.

3.2 Solution Representation and Evaluation

3.2.1 Solution Representation

The solution in this study is represented by three vectors and one matrix:

- The first vector (machinesAssigned) has a size of M, where M is the number of machines. Each slot represents a cell to which each machine is assigned; the same logic is applied to the second vector (workersAssigned), but instead of machines, it's workers. Its size corresponds to W, the number of workers. Because each worker and machine has its own slot in the machinesAssigned vector and workersAssigned vector, this design prevents them from being assigned to more than one cell. Since this is the case, both constraints 1.10 (which ensures that a worker must be assigned to a single cell) and 1.11 (which mandates that a machine be assigned to a single cell) hold true. During the resolution process, it is still necessary to ensure that the cell sizes of both workers and machines are accurately specified.

- The third vector (routesSelected) represents the processing path selected for each component. It has a size equal to P, where P is the number of parts. Each component can select only a single route by reserving a single slot in the routesSelected vector. Consequently, this structure maintains the model's feasibility with regard to constraint 1.9 (it verifies that a single route is chosen to process each component)

- Lastly, the matrix workerAssignments identifies the worker responsible for executing each operation. Each operation is characterized by the component to which it belongs and the machine on which it is carried out. Consequently, the dimensions of the matrix are P by M, and each slot contains at most one worker. The fact that a single slot is reserved in the workerAssignments matrix for each operation of the selected route guarantees that each operation can be performed by a single worker. To satisfy constraint 1.8, it is sufficient to ensure, during the resolution procedure, that the execution of operation s occurs only if its route r of component p has been selected (routesSelected[p] = r).

During execution, restriction 1.13 is enforced while restriction 1.12 is disregarded.

3.2.2 Solution Evaluation

Every solution must be evaluated. In our case, there are three objective functions: Quality 1.6, InterCMC 1.4, and IntraCMHC 1.5. We're attempting to reduce two of them. while attempting to maximize the quality index 1.1. Because these goals are at odds, we can't simply add them up and see which solution is better. Our first strategy is fitness dominance. Simply put, we sort solutions according to how many other solutions dominate them. Meaning that rank 0 represents the best solutions that are unrivaled, rank 1 the second-best solutions, and so on.

However, as you might expect, this only gives us groups of incomparable answers, also referred to as "fronts." The goal of multi-objective optimization, as discussed before in Chapter 2, is to identify a pareto optimum front or an approximation substitute when employing approximate approaches, as we have done.

We will compare our previously identified Pareto optimum front derived using the precise Augmecon-R approach (addressed previously in chapter 2). To determine which of the Rank 0 Fronts generated by our implementations of NSGA2 and NRGA is closer to the Pareto front.

This is possible for small problems, but for larger problems, Augmecon-R was simply unsuitable, and so we will simply compare the two approaches. All of this is only feasible with the use of appropriate Performance indicators and metrics.

3.2.3 Performance Indicators and Metrics

To evaluate the effectiveness of NSGA-II and NRGA. to solve the GCCFP, each test instance is launched ten times. The following metrics are used to assess the non-dominated set obtained in the i^{th} run, $Front_0$.

- **Average Ratio of non-dominated frontier :** The $Ratio\_ND$ function determines the proportion of solutions in $Front_0^i$ that are not dominated by any other solutions in the exact pareto frontier P.

$$Ratio\_ND = \frac{Front_0 - \{x \in Front_0 \mid \exists y \in P : y \prec x\}}{N} \tag{3.1}$$

  $y \prec x$ indicates that solution y dominates solution x. This metric ranges between 0 and 1 in value. The higher the value of this metric, the more dominant the resulting frontier [59].

- **Convergence Metric :** This metric measures the distance between the solutions in $Front_0^i$ and the exact pareto frontier P. c_i indicates the euclidean distance between the i^{th} seed in $Front_0^i$ and the closest seed in the exact pareto frontier. N is the number of solutions contained within $Front_0^i$. The lower the value of this metric, the closer the method converges to the Pareto optimal frontier [52].

$$Convergence = \frac{\sum_{i=1}^{N} c_i}{N} \tag{3.2}$$

- **Execution Time:** Analysis of the method's CPU time consumption.

3.3 Algorithms and Implementation

3.3.1 Non-dominated Sorting Genetic Algorithm II

Algorithm 2 Create initial population

```
1: for $i \leftarrow 1$ to pop_size do
2:     for each $w \in W$ do
3:         $indiv_i.workersAssigned\,[w] \leftarrow Random\,(1 : C)$
4:     end for
5:     for each $m \in M$ do
6:         $indiv_i.machinesAssigned\,[m] \leftarrow Random\,(1 : C)$
7:     end for
8:     if $\exists\ c \in C$ which does not have at least a machine assigned then
9:         badSolution = true
10:    end if
11:    for each $p \in P$ do
12:        $indiv_i.routesSelected\,[p] \leftarrow Random\,(1 : R_p)$
13:    end for
14:    for each $p \in P$ do
15:        for each $m \in M$ do
16:            $indiv_i.workerAssignments\,[p]\,[m] \leftarrow Random\ from\ Candidate\ Workers$
17:        end for
18:    end for
19: end for
```

During the generation of the initial population (see Algorithm 2), the feasibility of constraints 1.6 to 1.10 is ensured. Randomness governs the assignment of machines and workers to cells (lines [2-7]) and the selection of part routes (lines [11-13]). However, the third component of the solution is constructed by selecting capable workers at random (lines [14-18]).

At lines 8 and 9 a trigger is made to signal that this solution violates constraint 1.13 and must be removed later.

Our implementation of the non-dominated sorting genetic algorithm is outlined in the proposed algorithm 3. The implementation is straightforward; in line 16 we use Binary Tournament By Rank Domination selection, which entails selecting the dominator of a pair random parents each time.

In lines [25-28], when we reach the end of generation based on the stopping criteria, we simply select the first front as the resulting front and eliminate all non-conforming solutions to constraint 1.13 as well as duplicates of any kind. It may be possible to increase the probability of locating a global optimal solution by not removing them prior to execution.

Algorithm 3 NSGA-II

```
1: Initialize Population $Pt$
2:    { Generate Random population − size $N$
3: Generate Child Population $Qt$
4:    Crossover and Mutation }
5: $Rt = Pt \cup Qt \quad \rightarrow$ Combined population Rt − size $2N$
6: Evaluate Objective of values of each member of $Rt$
7: $i = 1$
8: while $i \leq maxGen$ do
9:     for each member of the combined population $Rt$ do
10:        Assign Rank (level) based on Pareto - sort
11:        Generate a set of non-dominated fronts
12:        Calculate the crowding distance between members on each front
13:    end for
14:     Select the members of the combined population $Rt$
        based on least dominated $N$ solution to make the population of the next
        generation $Pt_{+1}$ . Ties are resolved by taking the less crowding distance.
15:    Create next generation $Qt_{+1}$
16:       { Binary Tournament Selection By Rank Domination
17:         Crossover and Mutation }
18:    $Rt_{+1} = Pt_{+1} \cup Qt_{+1}$
19:    $Rt = Rt_{+1}$
20: end while
21: for each member of the combined population $Rt$ do
22:    Assign Rank (level) based on Pareto - sort
23:    Generate a set of non-dominated fronts
24: end for
25: LocalFront = the first and best front from the set of non-dominated fronts
26: remove double solutions from the local front
27: remove badSolutions which do not respect constraint 1.13 from the local front
28: remove solutions with duplicate fitness from the local front
```

The crossover is defined as the global process that permits the solution to leap to the optimal current solution. In this study, an adaptation of the crossover procedure to GCCFP is developed. This crossover occurs between two randomly selected members of the population (see Algorithm 4 5). In the proposed algorithm, crossover acts with a probability known as Crossover rate on the assignment of cells (machines or workers) or the selection of routes for parts. Three crossover sites are randomly generated in:

1. the cell affectation of machines (lines [4-14]).
2. the cell affectation of workers (lines [17-27]).
3. the routes selection of parts (lines [29-39]) with the exchange in the workers' assignment of operations (lines [1-13] in Algorithm 5).

This final step permits us to maintain the validity of constraints 1.6 and 1.7.

Algorithm 4 Crossover(indiv1,indiv2)

```
1:  $CellsAssigned = WorkersAssigned \cup MachinesAssigned$
2:  rand ← Random (1:3)
3:  if rand = 1 then            ⇒ exchange in the cell affectation of machines
4:      generate three random positions r1,r2 and r3 between (1:M) ⇒ r1≤r2≤r3
5:      for i← r1 to r2 do
6:          val←indiv1.CellsAssigned[i]
7:          indiv1.CellsAssigned[i]← indiv2.CellsAssigned[i]
8:          indiv2.CellsAssigned[i]← val
9:      end for
10:     for i← r3 to M do
11:         val←indiv1.CellsAssigned[i]
12:         indiv1.CellsAssigned[i]← indiv2.CellsAssigned[i]
13:         indiv2.CellsAssigned[i]← val
14:     end for
15: else
16:     if rand = 2 then            ⇒ exchange in the cell affectation of workers
17:         generate three random positions r1,r2 and r3 between (1:W)
                                                  ⇒ r1≤r2≤r3
18:         for i← r1 to r2 do
19:             val←indiv1.CellsAssigned[M+i]
20:             indiv1.CellsAssigned[M+i]← indiv2.CellsAssigned[M+i]
21:             indiv2.CellsAssigned[M+i]← val
22:         end for
23:         for i← r3 to W do
24:             val←indiv1.CellsAssigned[M+i]
25:             indiv1.CellsAssigned[M+i]← indiv2.CellsAssigned[M+i]
26:             indiv2.CellsAssigned[M+i]← val
27:         end for
28:     else
29:         if rand = 3 then
30:             generate three random positions r1,r2 and r3 between (1:P)
                                                  ⇒ r1≤r2≤r3
31:             for i← r1 to r2 do    ⇒ exchange in the routes selection of parts
32:                 val←indiv1.routesSelected[i]
33:                 indiv1.routesSelected[i]← indiv2.routesSelected[i]
34:                 indiv2.routesSelected[i]← val
35:             end for
36:             for i← r3 to M do
37:                 val←indiv1.routesSelected[i]
38:                 indiv2.routesSelected[i]← indiv2.routesSelected[i]
39:                 indiv2.routesSelected[i]← val
40:             end for
```

Algorithm 5 Crossover Resume

```
1:              for i← r1 to r2 do
2:                for m←0 to M do  ⇒ exchange in the workers assignment of operations
3:                  val←indiv1.WorkerAssignments[i][m]
4:                  indiv1.WorkerAssignments[i][m]← indiv2.WorkerAssignments[i][m]
5:                  indiv2.WorkerAssignments[i][m]← val
6:                end for
7:              end for
8:              for i← r3 to P do
9:                for m←0 to M do
10:                 val←indiv1.WorkerAssignments[i][m]
11:                 indiv1.WorkerAssignments[i][m]← indiv2.WorkerAssignments[i][m]
12:                 indiv2.WorkerAssignments[i][m]← val
13:               end for
14:             end for
15:           end if
16:         end if
17:       end if
```

Algorithm 6 Mutation(indiv)

```
$CellsAssigned = WorkersAssigned \cup MachinesAssigned$
rand ← Random (1:4)
if rand = 1 then
    m ← Random(0:M)
    indiv.CellsAssigned[m] ← Random(1:C)
else
    if rand = 2 then
        w ← Random(0:W)
        indiv.CellsAssigned[M+w] ← Random(1:C)
    else

        if rand = 3 then
            p ← Random(0:P)
            r ← Random(0 : $R_p$)
            indiv.routesSelected[p] ← w
            for each op(p,m) ∈ r do                    ⇒ op is an operation of r
                Select a random worker w that may execute op
                indiv.WorkerAssignments[p][m] ← w
            end for
        else
                p ← Random(0:P)
                m ← Random(0:M)
                Select a random worker w that may execute op(p,m)
                indiv.WorkerAssignments[p][m]← w
        end if
    end if
end if
```

In GAs, local optima are avoided using the mutation procedure (see algorithm 6). The mutation acts at random with a probability called mutation rate on the assignment of cells (machines or workers), or in the route selection of parts, or in the operation assignment of workers. It entails randomly assigning a machine or a worker to a random cell, or randomly assigning the chosen route for a part to another route, and randomly assigning workers to the operations of this route. Constraints 1.6 and 1.7 are kept in mind.

Lastly, the mutation procedure can affect the assignment of operations to workers, and it consists of selecting a random operation and a random worker w who may execute this operation. The mutation is accomplished by assigning w to the applicable operation.

Algorithm 7 Crowding Distance method used in NSGA-II algorithm.

```
1:  Input: List $P$ of non-dominated solutions with $p := |P|$
2:  Number of Objectives M
3:  Output:list $P$ with added Crowding Distance (CD) values for each solution
4:  For $i \in \{1, .., M\}$ do
5:      $f_{i,max}$ = maximum of values for $i$-th objective in $P$
6:      $f_{i,min}$ = minimum of values for $i$-th objective in $P$
7:  end for
8:  For $i \in \{1, .., P\}$ do
9:      P[j].CD =0            // initialize CD of $j$-th solution im $P$
10: end for
11: For $i \in \{1, .., M\}$ do
12:     $P'$ = sort $P$ ascending based on $i$-th objective
13:     $P'[1]$.CD =$\infty$
14:     $P'[p]$.CD =$\infty$
15:     For $j \in \{2, ..., p-1\}$ do
16:         $P'[j].CD+ = \frac{P'[j+1].f_i - P'[j-1].f_i}{f_{i,max} - f_{i,min}}$
17:     end for
18: end for
19: return$P$
```

As previously observed, the crowding distance is calculated to approximate the density of solutions surrounding a particular solution. The solution with the greater crowding distance is deemed to be in a less populated region, indicating that it is quite unique and should be retained for a bit longer. Its implementation is dependent on the number of objectives M. In our case, M = 3 (see Algorithm 7).

3.3.2 Non-dominated Ranked Genetic Algorithm

The algorithm 8 is an exact copy of 3 ; our implementation of NRGA consisted of simply changing the selection to Ranked Based Roulette Wheel Selection in line 16; it is essentially a roulette wheel selection with the probability seen in 2.2; in our case, Rank equals the number of ranks minus the individual's rank.

Algorithm 8 NRGA

```
1: Initialize Population $Pt$
2:    { Generate Random population − size $N$
3: Generate Child Population $Qt$
4:    Crossover and Mutation }
5: $Rt = Pt \cup Qt \quad \rightarrow$ Combined population Rt − size $2N$
6: Evaluate Objective of values of each member of $Rt$
7: $i = 1$
8: while $i \leq maxGen$ do
9:     for each member of the combined population $Rt$ do
10:        Assign Rank (level) based on Pareto - sort
11:        Generate a set of non-dominated fronts
12:        Calculate the crowding distance between members on each front
13:    end for
14:     Select the members of the combined population $Rt$
        based on least dominated $N$ solution to make the population of the next
        generation $Pt_{+1}$ . Ties are resolved by taking the less crowding distance.
15:    Create next generation $Qt_{+1}$
16:       { Ranked Based Roulette Wheel Selection
17:         Crossover and Mutation }
18:    $Rt_{+1} = Pt_{+1} \cup Qt_{+1}$
19:    $Rt = Rt_{+1}$
20: end while
21: for each member of the combined population $Rt$ do
22:    Assign Rank (level) based on Pareto - sort
23:    Generate a set of non-dominated fronts
24: end for
25: LocalFront = the first and best front from the set of non-dominated fronts
26: remove double solutions from the local front
27: remove bad solutions which do not respect constraint 1.13 from the local front
28: remove solutions with duplicate fitness from the local front
```

3.4 Computational Results

The following tools were used to code NSGA-II and NRGA:

Language : Java 18 LibericaJDK-18-Full

Integrated Development Environment : IntelliJ IDEA 2022.1.4

Operating system : Windows 11 64x

CPU : AMD Ryzen 7 5800H at 3.20Ghz and 16 GB of RAM

We will compare the performance of our algorithms using an exact pareto front obtained by the Augmecon-R method. However, the optimal pareto fronts will only be available for the small sized intances; for more challenging problems, we will refer to each other's performance using the previously discussed metrics.

3.4.1 Parameter Setting and Stopping Criterion

The effectiveness of meta-heuristic algorithms is significantly influenced by the proper parameter value selection. Setting them by using prior literature isn't always the best course of action. In this study, the trial-and-error approach is utilized. Thus, after extensive testing, the parameters have been determined as follows:
maxGeneration=15000,populationSize=160, crossoverRate=0.85, mutationRate=0.25, Repetition=10.

3.4.2 Instances

Each instance is represented in a text file and organized, as shown in Figure 3.1.
These files were acquired from the authors of [6].

1 : The total number of parts.

2 : The total number of routes.

3 : The total number of machines.

4 : The total number of workers.

5 : The total number of cells.

6 : The vector that represents the number of routes for each part.

7 : The matrix that represents the number of operations in each route for each part.

8 : The vector that represents the InterCelluar material handling cost per part.

9 : The vector that represents the IntraCellular material handling cost per part.

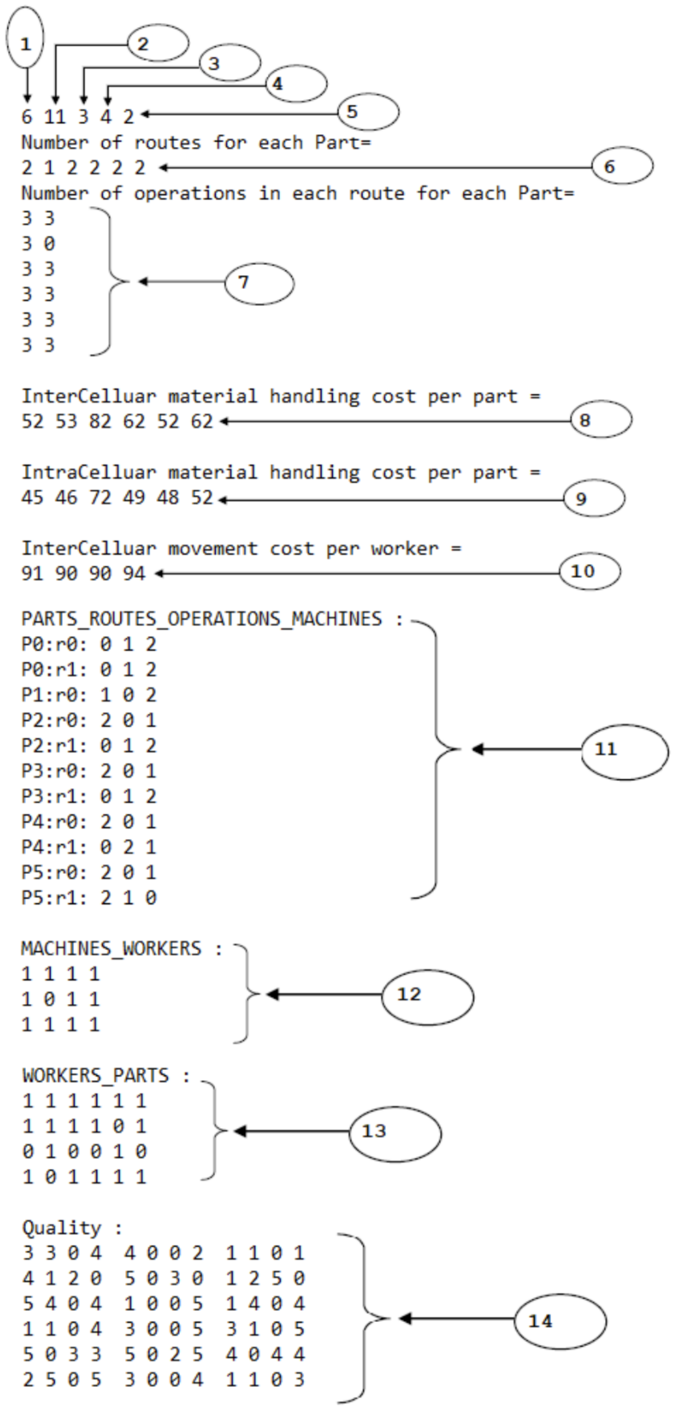
10 : The vector that represents the InterCelluar movement cost per worker.

11 : The three-dimensional matrix that indicates which machine is used in each operation in each rout for each part.

12 : The matrix that indicates whether the worker can use the concerned machine.

13 : The matrix that indicates whether the worker can process the concerned part.

14 : Three-dimensional matrix represents the quality obtained for each part when it is processed on each machine by each worker.



```
6 11 3 4 2
Number of routes for each Part=
2 1 2 2 2 2
Number of operations in each route for each Part=
3 3
3 0
3 3
3 3
3 3
3 3

InterCelluar material handling cost per part =
52 53 82 62 52 62

IntraCelluar material handling cost per part =
45 46 72 49 48 52

InterCelluar movement cost per worker =
91 90 90 94
PARTS_ROUTES_OPERATIONS_MACHINES :
P0:r0: 0 1 2
P0:r1: 0 1 2
P1:r0: 1 0 2
P2:r0: 2 0 1
P2:r1: 0 1 2
P3:r0: 2 0 1
P3:r1: 0 1 2
P4:r0: 2 0 1
P4:r1: 0 2 1
P5:r0: 2 0 1
P5:r1: 2 1 0

MACHINES_WORKERS :
1 1 1 1
1 0 1 1
1 1 1 1

WORKERS_PARTS :
1 1 1 1 1 1
1 1 1 1 0 1
0 1 0 0 1 0
1 0 1 1 1 1

Quality :
3 3 0 4  4 0 0 2  1 1 0 1
4 1 2 0  5 0 3 0  1 2 5 0
5 4 0 4  1 0 0 5  1 4 0 4
1 1 0 4  3 0 0 5  3 1 0 5
5 0 3 3  5 0 2 5  4 0 4 4
2 5 0 5  3 0 0 4  1 1 0 3
```

Figure 3.1: Instance representation calculations.

3.4.3 NSGA-II Performance

The instances shown in Table 3.1 (#1 to #8) are considered to be of a small size. Consequently, there is a precise pareto front for each of them. These instances were executed ten times using NSGA-II.

The preceding three metrics and the Pareto optimal front were used to evaluate these results.

We observe in #1 that the average convergence is zero, indicating that the distance between the obtained front and pareto front is zero. i.e., it is the same front, while also emphasizing that the obtained front has no duplicate solutions, #8 is nearly identical, whereas the remaining solutions appear to be quite distant from the pareto frontier.

Low convergence indicates that the front is far from the pareto front, and one might assume that this automatically means that the resulting front is worse. However, as demonstrated in all instances except for #7, the ND ratio is on average 1.0, indicating that no solution from the pareto front is dominant over the solutions from the obtained front. #7 is essentially identical.

3.4.4 NRGA Performance

The instances shown in Table 3.2 (#1 to #8) are of a small size. there is a precise pareto front for each of them. These instances were executed ten times using NRGA.
The preceding three metrics and the Pareto optimal front were used to evaluate these results.

These results are essentially the same as NSGA 2 excluding the Execution time which is on average over 50% higher.

Table 3.1: Small sized instances run 10 times on NSGA-II

	The problem characteristics					Convergence		ND Ratio		Time	
No.	P	$\sum_{p=1}^{P} R_p$	M	W	C	Average	Best	Average	Best	Average	Best
1	2	1	3	2	2	0.0	0.0	1.0	1.0	35.24	34.71
2	2	2	3	2	2	6.36	6.36	1.0	1.0	35.51	34.93
3	3	2	3	3	2	13.08	13.08	1.0	1.0	36.11	35.44
4	3	3	3	3	2	7.59	7.09	1.0	1.0	35.40	34.93
5	4	2	3	2	2	6.11	6.11	1.0	1.0	36.16	35.61
6	4	2	3	3	2	10.99	10.99	1.0	1.0	35.32	34.64
7	5	2	4	3	2	8.80	8.44	0.98	1.0	35.74	34.85
8	6	2	3	4	2	0.07	0.0	1.0	1.0	39.52	38.70

Table 3.2: Small sized instances run 10 times on NRGA

	The problem characteristics					Convergence		ND Ratio		Time	
No.	P	$\sum_{p=1}^{P} R_p$	M	W	C	Average	Best	Average	Best	Average	Best
1	2	1	3	2	2	0.0	0.0	1.0	1.0	58.68	57.60
2	2	2	3	2	2	6.36	6.36	1.0	1.0	58.76	57.66
3	3	2	3	3	2	13.08	13.08	1.0	1.0	59.20	58.34
4	3	3	3	3	2	7.52	7.09	1.0	1.0	58.67	57.71
5	4	2	3	2	2	6.11	6.11	1.0	1.0	59.35	58.51
6	4	2	3	3	2	10.99	10.99	1.0	1.0	35.32	57.53
7	5	2	4	3	2	8.34	8.34	0.97	1.0	59.04	57.80
8	6	2	3	4	2	0.0	0.0	0.99	1.0	63.29	61.83

3.4.5 NSGA-II VS NRGA

	The problem characteristics						ND Ratio		Time	
No.	P	E	Rp	M	W	C	Average	Best	Average	Best
0	4		8	3	3	2	1.0	1.0	20.37	20.17
1	5		10	4	3	2	0.93	1.0	21.05	20.86
2	6		11	3	4	2	1.0	1.0	22.62	22.21
3	7		12	4	3	2	0.85	1.0	22.06	21.75
4	8		16	4	3	3	0.77	1.0	20.97	20.31
5	9		18	5	4	4	0.80	1.0	20.58	19.97
6	10		20	5	4	3	0.73	1.0	22.22	21.24
7	10		20	7	6	4	0.95	1.0	23.31	22.54
8	11		22	4	4	3	0.64	0.98	21.24	20.73
9	12		24	6	7	3	0.87	1.0	24.52	23.75
10	13		26	8	7	4	0.82	1.0	27.32	25.97
11	14		28	10	7	4	0.82	1.0	29.84	27.88
12	20		40	13	15	5	0.76	1.0	50.94	47.97
13	30		90	15	12	5	0.86	1.0	64.09	60.70
14	35		85	18	18	6	0.78	0.99	100.81	95.96
15	40		80	20	15	7	0.82	0.95	148.32	142.28
16	50		148	22	15	7	0.84	1.0	151.58	144.40

Table 3.3: Large sized instances run 10 times on NSGA-II referenced with NRGA

	The problem characteristics						ND Ratio		Time	
No.	P	E	Rp	M	W	C	Average	Best	Average	Best
0	4		8	3	3	2	1.0	1.0	36.57	36.02
1	5		10	4	3	2	0.91	0.98	37.47	37.22
2	6		11	3	4	2	0.98	1.0	39.27	38.77
3	7		12	4	3	2	0.87	0.98	38.74	38.31
4	8		16	4	3	3	0.80	0.97	37.68	36.62
5	9		18	5	4	4	0.63	1.0	37.35	36.31
6	10		20	5	4	3	0.54	1.0	38.99	37.87
7	10		20	7	6	4	0.61	0.96	39.72	39.06
8	11		22	4	4	3	0.83	1.0	38.17	37.52
9	12		24	6	7	3	0.64	0.94	41.15	40.70
10	13		26	8	7	4	0.83	1.0	42.89	42.03
11	14		28	10	7	4	0.70	1.0	47.24	44.51
12	20		40	13	15	5	0.91	1.0	67.18	64.39
13	30		90	15	12	5	0.90	1.0	80.50	76.71
14	35		85	18	18	6	0.89	1.0	118.52	112.0
15	40		80	20	15	7	0.82	1.0	163.40	155.49
16	50		148	22	15	7	0.75	1.0	161.08	153.48

Table 3.4: Large sized instances run 10 times on NRGA referenced with NSGA-II

Previously, when running small instances, we had a pareto optimal front as a point of reference; however, this is no longer the case when dealing with large-scale problems.

Therefore, in order to compare these results, we will use the ND Ratio Metric and reference the results of each algorithm against one another; this will allow us to determine which one is more dominant.

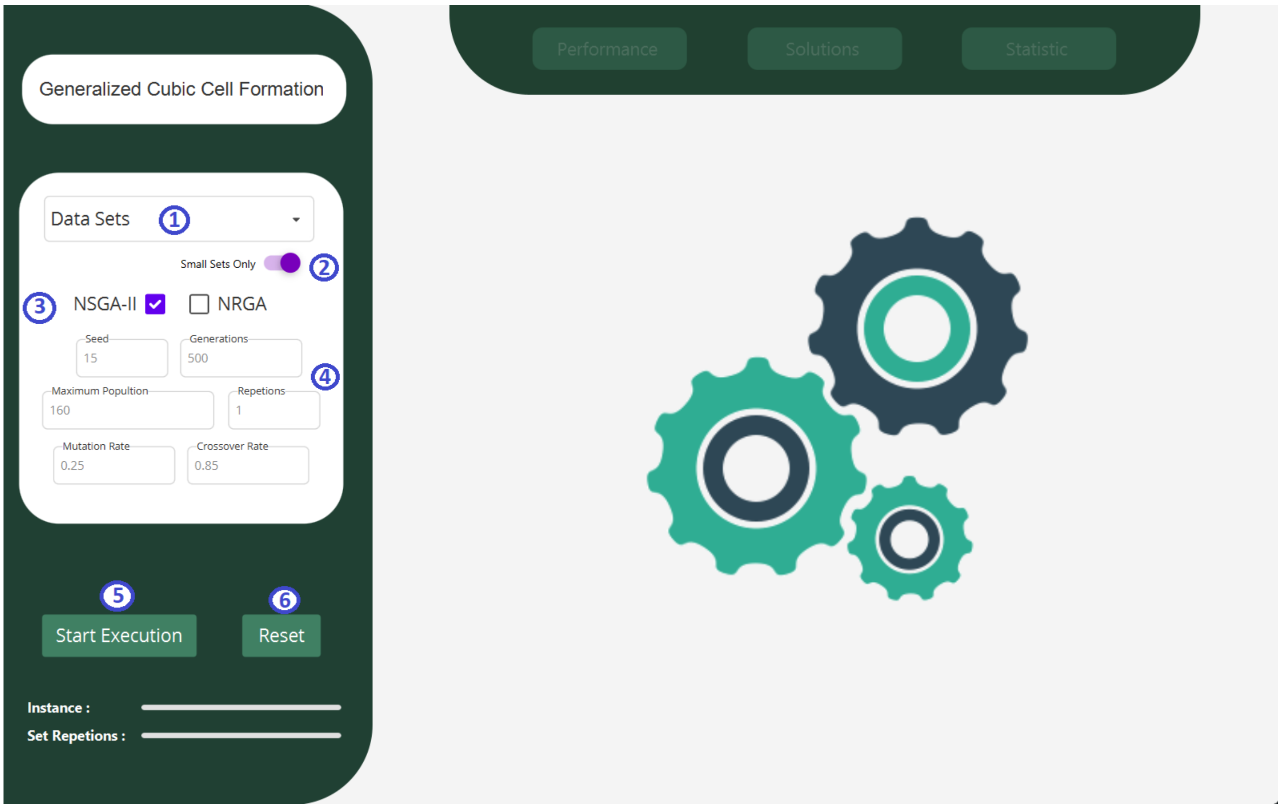
This is precisely what the two tables 3.3 and 3.4 demonstrate. We observe that, on average and in exceptional cases, NSGA2 and NRGA have nearly identical dominance over one another. It differs from instance to instance, but is consistent overall.

3.5 Software presentation

3.5.1 Graphical user interface

In our project, we used the JavaFX library because of its convenience in designing applications with graphical user interface. The Graphical User Interface will be presented using the following Figures 3.2,3.3,3.4,3.5.

3.5.1.1 Standby window



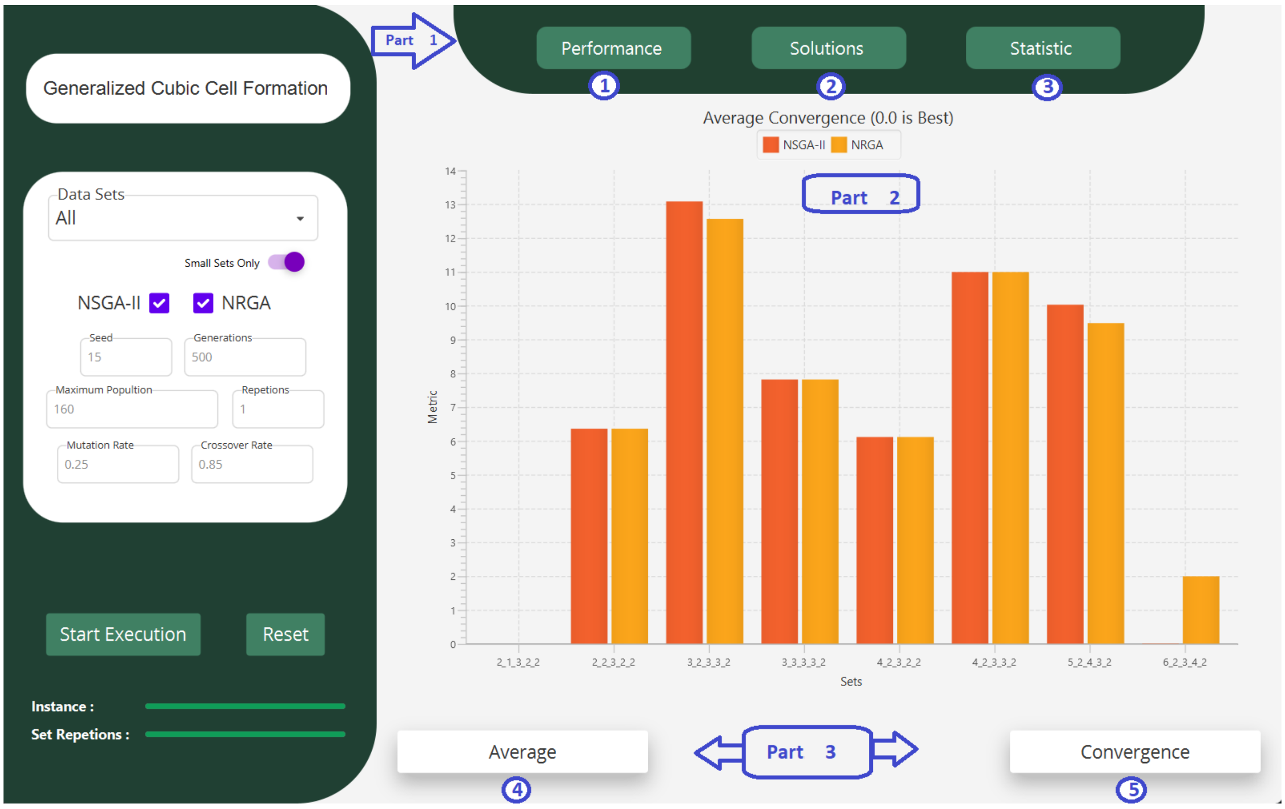
Small Sets Only ✓
NSGA-II [✓] NRGA [ ]
Seed: 15
Generations: 500
Maximum Popultion: 160
Repetions: 1
Mutation Rate: 0.25
Crossover Rate: 0.85

Figure 3.2: Standby window.

As seen in figure 3.2, the standby window is completely empty aside from the sizable, crucial side panel, which is attainable from all windows. The figure also shows its features, which are as follows:

1. Load instance.
2. Choose instance type.
3. Choose resolution method.
4. Set parameters.
5. Start execution .
6. Reset all.

3.5.1.2 Performance window



Generalized Cubic Cell Formation

Data Sets: All

Small Sets Only

NSGA-II ✓ NRGA ✓

Seed	15
Generations	500
Maximum Popultion	160
Repetions	1
Mutation Rate	0.25
Crossover Rate	0.85

Instance :

Set Repetions :

Figure 3.3: Performance window.

When the execution is complete, as indicated by the progress bars beneath the start execution button, three buttons illuminate, as shown in figure 3.3 part 1:

1. Performance button.
2. Solutions button.
3. Statistic button.

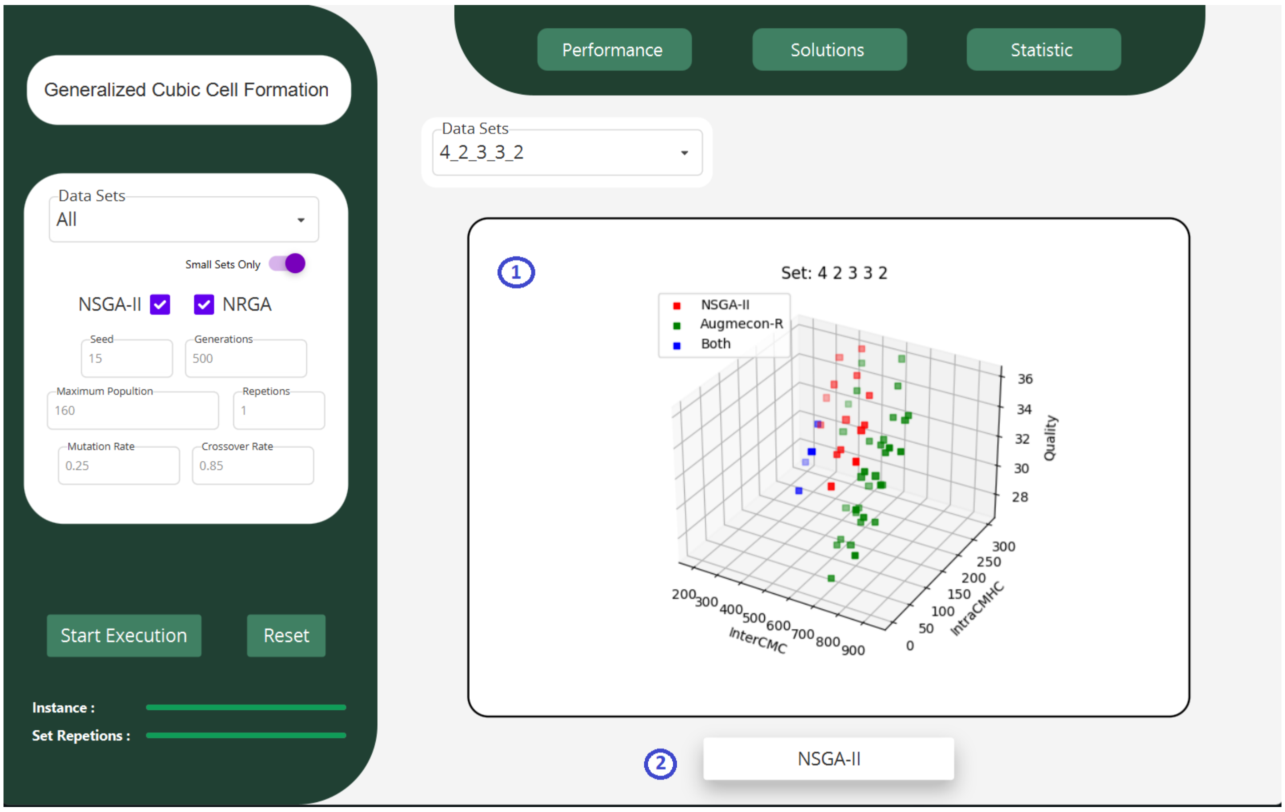
Parts 2 and 3—which represent the performance window—will appear when we select the performance button.

The second part is the result area, and in the case of performance, it displays a histogram. In Part 3, two buttons are shown:

4. Changing from average to best results.

5. Changing Metrics.

3.5.1.3 Solutions window


Data Sets: All

Small Sets Only

NSGA-II ☑ NRGA ☑

Seed: 15

Generations: 500

Maximum Popultion: 160

Repetions: 1

Mutation Rate: 0.25

Crossover Rate: 0.85

Data Sets: 4_2_3_3_2

Figure 3.4: Solutions window.

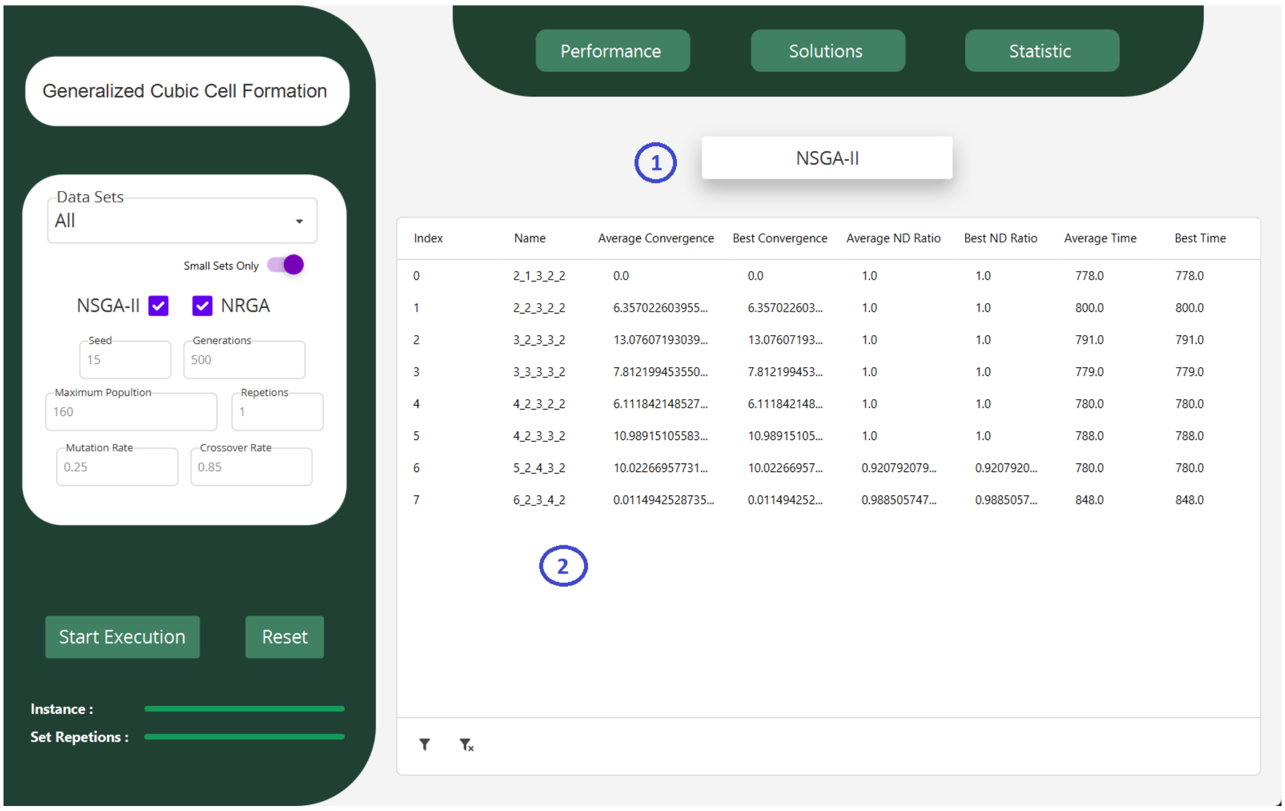
When you click the Solutions button and choose a solved dataset (if more than one is available), you'll see the following in figure 3.4:

1. Scatter plot representing the selected solved set.

2. Switch between the resolution methods if more than one are selected.

Red dots indicate the chosen resolution method, green dots indicate the referenced resolution method, and blue dots indicate the points that are shared by the two.

3.5.1.4 Statistics window



Data Sets: All
Small Sets Only
NSGA-II ✓ NRGA ✓
Seed: 15
Generations: 500
Maximum Popultion: 160
Repetions: 1
Mutation Rate: 0.25
Crossover Rate: 0.85

Index	Name	Average Convergence	Best Convergence	Average ND Ratio	Best ND Ratio	Average Time	Best Time
0	2_1_3_2_2	0.0	0.0	1.0	1.0	778.0	778.0
1	2_2_3_2_2	6.357022603955...	6.357022603...	1.0	1.0	800.0	800.0
2	3_2_3_3_2	13.07607193039...	13.07607193...	1.0	1.0	791.0	791.0
3	3_3_3_3_2	7.812199453550...	7.812199453...	1.0	1.0	779.0	779.0
4	4_2_3_2_2	6.111842148527...	6.111842148...	1.0	1.0	780.0	780.0
5	4_2_3_3_2	10.98915105583...	10.98915105...	1.0	1.0	788.0	788.0
6	5_2_4_3_2	10.02266957731...	10.02266957...	0.920792079...	0.9207920...	780.0	780.0
7	6_2_3_4_2	0.0114942528735...	0.011494252...	0.988505747...	0.9885057...	848.0	848.0

Figure 3.5: Statistics window.

In figure 3.5, which appears when you press the Statistic button, you can see the following:

1. Switch between the resolution methods if more than one are selected.
2. Table with all the information on the results

3.6 Conclusion

In this chapter, we demonstrated our application of the genetic algorithm to GCCFP. Initial presentation includes the solution's adopted representation and evaluation. Next, the solution strategy containing a description of the proposed NSGA-II and NRGA is outlined. Finally, the instances and computed results and software are presented.

CONCLUSION AND PERSPECTIVES

The Generalized Cubic Cell Formation Problem, a variant of the Cell Formation Problem, has been addressed in this study. Regarding this issue, we consider:

- workers as the third dimension in addition to machines and parts.
- alternative routes for the components

Our method relies on Multi-Objective techniques to solve the problem of generalized cubic cell formation. Thus, we have selected the ever-popular NSGA-II and one of its variants, NRGA. To evaluate the performance of our implementation of the two algorithms, we compared our results with the results obtained by the LINGO software by solving the problem instances using the exact Augmecon-R method as a reference pareto frontier and using several performance indicators.
In addition, we compared the obtained results to one another by measuring their relative dominance.

Despite the fact that its convergence toward the pareto front was low, the results obtained on average were never dominated by the pareto front. This was the conclusion that could be drawn from the comparison. NSGA-II's performance was deemed to be satisfactory overall. The NRGA, on the other hand, performed almost exactly the same, but it was significantly slower in terms of its execution time.

This paper's work can be expanded to achieve further goals. As potential future extensions of our work, we plan to apply newer, more robust, and up-to-date multi-objective algorithms that can tackle this problem significantly more quickly and precisely. While Creating extra constraints in the GCCFP implementation to better depict difficult real-world scenarios

BIBLIOGRAPHY

[1] Timothy J. Greene and Randall P. Sadowski. A review of cellular manufacturing assumptions, advantages and design techniques. *Journal of Operations Management*, 4(2):85–97, feb 1 1984.

[2] Irina Utkina, Mikhail Batsyn, and Ekaterina Batsyna. A branch-and-bound algorithm for the cell formation problem. *International Journal of Production Research*, 56:1–12, mar 7 2018.

[3] Haval Nouri, S. Tang, B T Baharudin, Mohd khairol anuar Mohd ariffin, and Razali Samin. Metaheuristic Techniques on Cell Formation in Cellular Manufacturing System. *Journal of Automation and Control Engineering*, 1:49–54, jan 1 2013.

[4] Jeffrey Joines, Russell King, and C.T. Culbreth. A Comprehensive Review of Production-Oriented Manufacturing Cell Formation Techniques. *International Journal of Flexible Automation and Integrated Manufacturing*, 3:225–265, sep 1 1996.

[5] Sayyed Mahdi Sadat Khorasgani and Mahdi Ghaffari. Developing a cellular manufacturing model considering the alternative routes, tool assignment, and machine reliability. *Journal of Industrial Engineering International*, 14(3):627–636, sep 1 2018.

[6] Hamida Bouaziz, Meryem Berghida, and Ali Lemouari. Solving the Generalized Cubic Cell Formation Problem Using Discrete Flower Pollination Algorithm. *Expert Systems with Applications*, 150:113345, feb 1 2020.

[7] ARVIND BALLAKUR and HAROLD J. STEUDEL. A within-cell utilization based heuristic for designing cellular manufacturing systems. *International Journal of Production Research*, 25(5):639–665.

[8] Rogério Malta Branco and Carlos Rocha. Group Technology: Genetic Algorithm Based on Greedy Constructive Structure and Refinement by k-Means Method Applied to Manufacturing Cell Formation Problems. sep 21 2018.

[9] HOKEY MIN and DOOYOUNG SHIN. Simultaneous formation of machine and human cells in group technology: a multiple objective approach. *International Journal of Production Research*, 31(10):2307–2318.

[10] Kalyanmoy Deb and Kalyanmoy Deb. *Multi-objective Optimization*, pages 403–449. Springer US, Boston, MA.

[11] David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithm research: A history and analysis. Technical report, Citeseer.

[12] Dragan A. Savic, Godfrey A. Walters, and Martin Schwab. Multiobjective genetic algorithms for pump scheduling in water supply. pages 227–235, Berlin, Heidelberg. Springer Berlin Heidelberg.

[13] Lam Thu Bui and Sameer Alam. *An introduction to multi-objective optimization*, pages 1–19. IGI Global.

[14] C.A. Coello Coello. Evolutionary multi-objective optimization: a historical view of the field. *IEEE Computational Intelligence Magazine*, 1(1):28–36.

[15] A. W. Flux. Vilfredo Pareto. Cours d'Economie Politique. Tome Premier. *The Economic Journal*, 6(22):249–253.

[16] Jeffrey Horn. F1. 9 Multicriterion decision making. *Handbook of evolutionary computation*, 97(1).

[17] Kaisa Miettinen. Nonlinear Multiobjective Optimization. 12.

[18] Zorka Novak Pintarič and Zdravko Kravanja. *Suitable Process Modelling for Proper Multi-Objective Optimization of Process Flow Sheets*, volume 33, pages 1387–1392. Elsevier.

[19] Kuang-Hua Chang. *Chapter 5 - Multiobjective Optimization and Advanced Topics*, pages 325–406. Academic Press, Boston.

[20] Kalyanmoy Deb and Kalyanmoy Deb. *Multi-objective Optimization*, pages 403–449. Springer US, Boston, MA.

[21] Hella Tokos, Zorka Novak Pintarič, and Yongrong Yang. Bi-objective optimization of a water network via benchmarking. *Journal of Cleaner Production*, 39:168–179.

[22] Angel Hsu, Ainsley Lloyd, and John W Emerson. What progress have we made since Rio? Results from the 2012 Environmental Performance Index (EPI) and Pilot Trend EPI. *Environmental Science Policy*, 33:171–185.

[23] Lidija Čuček, Jiří Jaromír Klemeš, and Zdravko Kravanja. A review of footprint analysis tools for monitoring impacts on sustainability. *Journal of Cleaner Production*, 34:9–20.

[24] Andrei Kostin, Gonzalo Guillén-Gosálbez, and Laureano Jiménez. Dimensionality reduction applied to the simultaneous optimization of the economic and life cycle environmental performance of supply chains. *International Journal of Production Economics*, 159:223–232.

[25] L. Zadeh. Optimality and non-scalar-valued performance criteria. *IEEE Transactions on Automatic Control*, 8(1):59–60.

[26] Saul Gass and Thomas Saaty. The computational algorithm for the parametric objective function. *Naval Research Logistics Quarterly*, 2(1-2):39–45.

[27] Yv Haimes Yv, Leon S. Lasdon, and Dang Da. On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-1(3):296–297.

[28] Darrell Whitley. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, 43(14):817–831.

[29] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In *Proceedings of the fourth international conference on genetic algorithms*. Citeseer.

[30] David E Goldberg. *Genetic algorithms*. Pearson Education India.

[31] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[32] Thomas Bäck. *Evolutionary algorithms in theorie and practice*.

[33] Hans-Paul Schwefel. *Numerical optimization of computer models*. John Wiley Sons, Inc.

[34] Hans-Paul Paul Schwefel. *Evolution and optimum seeking: the sixth generation*. John Wiley Sons, Inc.

[35] John R Koza. *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*, volume 34. Stanford University, Department of Computer Science Stanford, CA.

[36] John R Koza, David E Goldberg, David B Fogel, and Rick L Riolo. *Genetic Programming 1996: proceedings of the first annual conference*. Mit Press.

[37] David B Fogel. *Artificial intelligence through simulated evolution*. Wiley-IEEE Press.

[38] DB Fogel. Evolving artificial intelligence [Ph. D. thesis]. *University of California, San Diego*.

[39] Alexandros Nikas, Angelos Fountoulakis, Aikaterini Forouli, and H. Doukas. A robust augmented ε-constraint method (augmecon-r) for finding exact solutions of multi-objective linear programming problems. *Operational Research*, 22, 04 2022.

[40] K. Metaxiotis and K. Liagkouras. Multiobjective evolutionary algorithms for portfolio management: A comprehensive literature review. *Expert Systems with Applications*, 39(14):11685–11698, 2012.

[41] Irina Dumitrescu and Thomas Stützle. Combinations of local search and exact algorithms. pages 211–223, 01 2003.

[42] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[43] George Mavrotas. Generation of efficient solutions in multiobjective mathematical programming problems using gams. effective implementation of the ε-constraint method. *Lecturer, Laboratory of Industrial and Energy Economics, School of Chemical Engineering. National Technical University of Athens*, 2007.

[44] George Mavrotas. Effective implementation of the ε-constraint method in multi-objective mathematical programming problems. *Applied mathematics and computation*, 213(2):455–465, 2009.

[45] Gabriel R Bitran and Jorge M Rivera. A combined approach to solve binary multicriteria problems. *Naval Research Logistics Quarterly*, 29(2):181–201, 1982.

[46] Yacov Haimes. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics*, 1(3):296–297, 1971.

[47] George Mavrotas and Kostas Florios. Augmecon2: A novel version of the ε-constraint method for finding the exact pareto set in multi-objective integer programming problems. *Applied Mathematics and Computation*, 219(18):9652–9669, 2013.

[48] George Mavrotas and Kostas Florios. Finding the exact pareto set in multiple objective integer programming problems using an improved version of the augmented epsilon constraint method. *Διαχείριση Ενεργειακών* Π*όρων & Συστημάτων*, page 17, 2012.

[49] George Mavrotas and Kostas Florios. An improved version of the augmented ε-constraint method (augmecon2) for finding the exact pareto set in multi-objective integer programming problems. *Applied Mathematics and Computation*, 219(18):9652–9669, 2013.

[50] Francis Sourd and Olivier Spanjaard. A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20(3):472–484, 2008.

[51] Pascal Halffmann, Luca E Schäfer, Kerstin Dächert, Kathrin Klamroth, and Stefan Ruzika. Exact algorithms for multiobjective linear optimization problems with integer variables: A state of the art survey. *Journal of Multi-Criteria Decision Analysis*, 2022.

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

[53] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International conference on parallel problem solving from nature*, pages 292–301. Springer, 1998.

[54] Shanu Verma, Millie Pant, and Vaclav Snasel. A comprehensive review on nsga-ii for multi-objective combinatorial optimization problems. *IEEE Access*, 9:57757–57791, 2021.

[55] Matthew Vavrina and Kathleen Howell. Multiobjective optimization of low-thrust trajectories using a genetic algorithm hybrid. volume 134, 02 2009.

[56] N. Srinivas and Kalyanmoy Deb. Multiobjective function optimization using non-dominated sorting genetic algorithms. 2, 06 2000.

[57] Mahziar Taghizadeh, Amir Shojaie, Amir Sarfaraz, and Sadigh Raissi. A multiobjective mathematical model for truck scheduling problem in multidoor cross-docking system. *Discrete Dynamics in Nature and Society*, 2022:15, 07 2022.

[58] Omar al jadaan, Lakishmi Rajamani, and C. Rao. Non-dominated ranked genetic algorithm for solving multi-objective optimization problems: Nrga. *Journal of Theoretical and Applied Information Technology*, 01 2008.

[59] Fulya Altiparmak, Mitsuo Gen, Lin Lin, and Turan Paksoy. A genetic algorithm approach for multi-objective optimization of supply chain networks. *Computers Industrial Engineering*, 51(1):196–215, 2006. Special Issue on Computational Intelligence and Information Technology: Applications to Industrial Engineering.

[60] Jeffrey Horn, N. Nafpliotis, and D.E. Goldberg. A niched pareto genetic algorithm for multi-objective optimization. volume 1, pages 82 – 87 vol.1, 07 1994.

[61] J. Schaffer and John Grefenstette. Multi-objective learning via genetic algorithms. pages 593–595, 01 1985.

[62] Carlos A Coello Coello. Evolutionary multi-objective optimization and its use in finance. *Handbook of Research on Nature Inspired Computing for Economy and Management. Idea Group Publishing*, 2006.

[63] Xavier Gandibleux and Matthias Ehrgott. 1984-2004–20 years of multiobjective metaheuristics. but what about the solution of combinatorial problems with multiple objectives? In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 33–46. Springer, 2005.

[64] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

[65] Eckart Zitzler, Marco Laumanns, Lothar Thiele, Carlos M Fonseca, and Viviane Grunert da Fonseca. Why quality assessment of multiobjective optimizers is difficult. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 666–674, 2002.

[66] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007.

[67] Carlos A Coello and Gregorio Toscano Pulido. Multiobjective optimization using a micro-genetic algorithm. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 274–282, 2001.

[68] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989.

[69] David Alexander Coley. *An introduction to genetic algorithms for scientists and engineers*. World Scientific Publishing Company, 1999.

[70] Poonam Sharma and Amit Wadhwa. Analysis of selection schemes for solving an optimization problem in genetic algorithm. *International Journal of Computer Applications*, 93(11), 2014.

[71] Yahya Rahmat-Samii and Eric Michielssen. Electromagnetic optimization by genetic algorithms. *Microwave Journal*, 42(11):232–232, 1999.

[72] Randy L Haupt and Sue Ellen Haupt. *Practical genetic algorithms*. John Wiley & Sons, 2004.

[73] Abdelmadjid Recioui. *Use of genetic algorithms in antennas. Application to yagi-uda antenna and antenna arrays*. PhD thesis, Boumerdes, University M'hamed Bougara. Faculty of Engineering, 2006.

[74] SN Sivanandam and SN Deepa. Introduction to genetic algorithms, by springer berlin heidelberg new york, 2008.

[75] Nature-inspired optimization algorithms. In Xin-She Yang, editor, *Nature-Inspired Optimization Algorithms*, page i. Elsevier, Oxford, 2014.

[76] Mitsuo Gen, Runwei Cheng, and Lin Lin. *Network models and optimization: Multiobjective genetic algorithm approach*. Springer Science & Business Media, 2008.

[77] Edmund K Burke, Edmund K Burke, Graham Kendall, and Graham Kendall. *Search methodologies: introductory tutorials in optimization and decision support techniques*. Springer, 2014.

[78] Edwin Lughofer and Moamar Sayed-Mouchaweh. *Predictive maintenance in dynamic systems: advanced methods, decision support tools and real-world applications*. Springer, 2019.