PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

University Mohamed Seddik Ben Yahia of Jijel, Algeria.

Faculty of Exact Sciences and Computer Science

Department of Computer Science

A Dissertation Submitted to the Department of Computer Science in Partial Fulfillment of the Requirements for the Degree of Master

Option: Artificial Intelligence

Solving the Generalized Cubic Cell Formation Problem Using Genetic Algorithm

Submitted by:

Ms. Hadil El-Hana Yallas

Supervised by:

Dr. Hamida Bouaziz

October 2020

Acknowledgment

Above all, I thank my Almighty God, who gave me strength, faith, health, will, and guidance to accomplish this modest work.

I wish particularly to acknowledge my Supervisor ***Dr. Hamida Bouaziz*** for having supervised, helped, guided, advised, and encouraged me throughout my work. I would like to praise the efforts and time that she devoted to me as a supervisor. Thanks to her instructions, the work has been adequately done.

I would also like to thank the jury members who have agreed to review and participate in the jury of my work.

I take this opportunity to express gratitude to our teachers of the computer science department at the University of Jijel, who have ensured my formation during my university cycle.

I would like to extend my sincere thanks to all those who have contributed in one way or another to the realization of this thesis.

My deepest gratitude goes to my family, who have been able to approach me without letting go of the support during all these long years of study.

To all these contributors, I offer my thanks, respect, appreciation, and admiration.

Dedication

With deep gratitude and sincere words, I dedicate this humble work to my beloved and respectful parents, who have sacrificed their lives for my success, and who have given me their eternal and unconditional love, their support, encouragement, and their prayers, day and night, which enlightened my path and helped me to reach the highest expectations in life, hoping that one day, I will be able to give them back some of what they have done for me. May God grant them happiness, health, and long life.

I will never forget the help of my supportive supervisor, who believed in me and encouraged me until the end.

Special credit would be devoted to all those who, with a word, gave me the strength to go on.

Finally, to all those who have taught me throughout my school life.

Hadil El-Hana

Abstract

Manufacturing cell formation is one of the essential stages in the construction of cellular manufacturing systems. It concentrates on grouping machines, parts, workers and assigning them to corresponding cells. This assignment is guided by multiple objectives and is subject to various constraints. In this work, the focus is on a variant of the cell formation problem "the Generalized Cubic Cell Formation Problem". To solve the problem, a Genetic Algorithm is developed. From the viewpoint of the objectives considered, and the computation time, the Genetic Algorithm's performance has been tested. The simulation results reveal that the Genetic Algorithm outperforms the Branch & Bound method and the Simulated Annealing Algorithm, especially for large-sized test problems. On the other hand, regarding the Discrete Flower Pollination Algorithm, GA reaches equal results for some instances. However, for more than half of the instances, DFPA gives better results than GA. For the computational time, the GA's results are better than those of the three other methods for the totality of the instances.

Keywords: Cell formation, Cellular manufacturing, Generalized cubic cell formation problem, Group technology, Genetic algorithm, Discrete flower pollination algorithm.

Résumé

La formation de cellules de fabrication est l'une des étapes essentielles de la construction des systèmes de fabrication cellulaires. Elle se concentre sur le regroupement des machines, des pièces, des ouvriers, et leur affectation à des cellules correspondantes. Cette affectation est guidée par de multiples objectifs et est soumise à diverses contraintes. Dans ce travail, l'accent est mis sur une variante du problème de la formation des cellules "le problème de formation des cellules cubique généralisé". Pour résoudre le problème, un algorithme génétique est développé. Du point de vue des objectifs considérés et du temps de calcul, la performance de l'algorithme génétique a été testée. Les résultats de la simulation révèlent que l'algorithme génétique surpasse la méthode Branch & Bound et l'algorithme de recuit simulé en particulier pour les problèmes de test de grande taille. D'une autre part, en considérant la version discrète de l'algorithme de pollinisation des fleurs, l'AG donne des résultats égaux dans certains cas. Cependant, pour plus de la moitié des cas, l'Algorithme discret de pollinisation des fleurs donne de meilleurs résultats que l'AG. Pour le temps de calcul, les résultats obtenus par l'AG sont meilleurs que ceux des trois autres méthodes pour la totalité des cas.

Mots clés: Formation de cellules, Fabrication cellulaire, Problème de formation des cellules cubique généralisé, Technologie de groupe, Algorithme génétique, Algorithme discret de pollinisation des fleurs.

Contents

List of Figures

List of Tables

List of Acronyms

AAA	Assignment Allocation Algorithm
ACO	Ant Colony Optimization
B&B	Branch and Bound
CFOPT	Cell Formation OPTimization
CFP	Cell Formation Problem
CM	Cellular Manufacturing
CMS	Cellular Manufacturing System
CO	Combinatorial Optimization
CPSO-LP	Combinatorial Particle Swarm Optimization and Linear Programming
DFPA	Discrete Flower Pollination Algorithm
DPSO	Discrete Particle Swarm Optimization
EA	Evolutionary Algorithm
EC	Evolutionary Computation
FPA	Flower Pollination Algorithm
GA	Genetic Algorithm
GA-AUGMECON	Genetic Algorithm AUGMented E-CONstraint
GCCFP	Generalized Cubic Cell Formation Problem
GCFP	Generalized Cell Formation Problem
GT	Group Technology
HCSA-GCF	Hybrid Selection Algorithm-Generalized Cell Formation
InterCMHC	Inter-Cellular Material Handling Cost
InterCWM	Inter-Cellular Worker Movement
IntraCMHC	Intra-Cellular Material Handling Cost
MOGGA	Multi-Objective Grouping Genetic Algorithm
MOVDO	Multi-Objective Vibration Damping Optimization
NRGA	Non-dominated Ranking Genetic Algorithms

NSGA-II	Non-dominated Sorting Genetic Algorithm
OX	Order Crossover
PMCGP	Percentage Multi-Choice Goal Programming
PMX	Partially Matched Crossover
PPX	Precedence Preservative Crossover
PSO	Particle Swarm Optimization
RMCGP	Revised Multi-Choice Goal Programming
SA	Simulated Annealing
TS	Tabu Search

Introduction

Cellular manufacturing (CM) is one of the main applications of group technology (GT). CM emerged as a production strategy capable of solving specific problems of complexity and long manufacturing lead times in batch production [1]. Its objective is to simplify the management of the manufacturing industries. By regrouping different parts' production into clusters, the manufacturing management is reduced to manage different small entities. One of the most critical cellular manufacturing problems is the design of these entities, called manufacturing cells [2].

These cells represent a cluster of machines dedicated to the production of one or several parts. The ideal design of cellular manufacturing is to make these cells independent from one another. The reality is a little more complicated. Once the cells are created, there exists still some traffic within and between them. The inter-cellular movement is the transfer of a part between two machines belonging to different cells. However, the intracellular movement is the transfer of a part between two machines in the same cell. Initially, each part is defined by one or several sequences of operations, and each of them can be produced according to a sequence of machines. A final sequence of machines must be chosen to produce each part. The worker's dimension is also considered in addition to the part and the machine dimensions. Because of the human factor essential role, grouping workers with similar expertise and skills to produce similar families of parts can improve the CMS design quality.

In literature, the above-described problem bears the name of the Cell Formation Problem (CFP).

Objective :

The cell formation problem is an NP-hard problem. Therefore, exact methods cannot be used to solve large problems in a reasonable time. On the other hand, meta-heuristics can generate high-quality solutions in reasonable computing time. In this study, we consider a variant of CFP, called Generalized Cubic Cell Formation Problem (GCCFP). It bears the adjective "Generalized" because each part may have more than a plan according to which it will be produced. However, it is a cubic problem because, besides the part and machine dimensions, we also consider the worker dimension. Several criteria can be defined at the level of the constraints of the problem. In this study, we consider four criteria: the inter-cellular movement and the intracellular movement of parts, inter-cellular movement of workers, and quality index.

The final objective is to build cells and to assign parts, machines, and workers to these cells in such a way:

- To minimize the material handling cost, which is the sum of the inter-cellular and the intracellular movement of the parts.
- To minimize the inter-cellular movement of workers.
- To maximize the produced parts' quality index.

In the literature, three algorithms have been used to solve the Generalized Cubic Cell Formation Problem [3]:

1. The Branch and Bound method, which is an exact method.
2. The Simulated Annealing algorithm, which is is a single-solution-based metaheuristic.
3. The Discrete Flower Pollination Algorithm (DFPA) which is a population-based metaheuristic.

The results showed that DFPA outperforms the other two above mentioned algorithms. In this study, we implement another population-based metaheuristic, which is the Genetic Algorithm that was initially introduced by John Holland [4]. The objective is to verify if GA may beat the results obtained by DFPA.

This master dissertation is organized into three chapters:

- In the first chapter, we will present the Cell Formation Problem and its variants.
- In the second chapter, we will introduce genetic algorithms by describing its vocabulary and operators.
- In the final chapter of this thesis, we will present the method we have developed to solve this problem and the results obtained from this study. To highlight the contribution and the efficiency of this approach, we compare the results obtained by our algorithm with those found in the literature.
- We conclude with a general conclusion that summarizes the performed study and provides some perspectives.

Chapter 1

Cell Formation Problem

1.1 Introduction

In manufacturing systems, GT is a manufacturing philosophy based on organizing and grouping common tasks to improve the productivity of the system [5]. One of the most important applications of GT is CM, which is the grouping of machines and parts so that each family of parts is processed within a machine cell [6]. Many benefits have been reported for CM, including reducing material handling costs, setup times, expediting costs, in-process inventories, part makespan, and improving human relations and operator expertise [5]. The core problem in designing a cellular manufacturing system (CMS) is the cell formation problem.

The CFP in CMSs is an important issue in the operational research literature [7, 8]. It consists of decomposing an entire production system into a set of manufacturing cells, assigning machines, and allocating parts to those cells. Some constraints and objectives must be taken into account to produce the most manageable and independent cells during this decomposition [3]. The cell formation problem is known to be a non-polynomial (NP)-hard problem [6]; therefore, the development of efficient machine grouping algorithms has always been the center of interest in CMS design, which has led to a wide range of research [9].

This chapter is organized as follows: In section 1.2, we define the cell formation problem. In section 1.3, we present related work. In section 1.4, we give a formulation of the Generalized Cubic Cell Formation Problem and its mathematical model. Finally, in section 1.5, we conclude.

1.2 Definition of the Cell Formation Problem

1.2.1 Basic Cell Formation Problem

In the basic cell formation problem, the only provided information is the incidence matrix of parts and machines. The incidence matrix is a binary matrix, where machines and parts are represented in rows and columns; each cell in this matrix may contain a 0 or 1 as a value. 0 means that the part in the column does not

	P1	P2	P3	P4	P5	p6	P7
M1	1	1	0	1	0	0	0
M2	0	1	1	1	1	1	1
M3	0	0	1	0	1	0	1
M4	0	0	1	1	0	1	0
M5	1	1	0	0	0	0	0
M6	1	1	0	0	1	0	0

Figure 1.1: Incidence matrix

	P3	P4	P6	P5	P7	P1	P2
M2	1	1	1	1	1	0	1
M4	1	1	1	0	0	0	0
M3	1	0	0	1	1	0	0
M1	0	1	0	0	0	1	1
M5	0	0	0	0	0	1	1
M6	0	0	0	0	1	1	1

Figure 1.2: The resulting groupement

need the machine in the row. Otherwise, the cell is filled with the value 1. Figure 1.1 shows an example of an incidence matrix. The cell formation problem's output is a configuration that specifies the nature of cells must be built, the cell to which each machine is assigned, and the cell to which each part is affected (see Figure 1.2).

1.2.2 Generalized Cubic Cell Formation Problem

In this work, we consider a variant of the CFP, known as the GCCFP. The basic CFP considers that each part has a single route. However, in real situations, a part may have more than one process routing (e.g., part p_i maybe processed on machines m_1 and m_3 or it may be processed on m_1, m_2, and m_4). The CFP that considers many potential process routings is called Generalized Cell Formation Problem (GCFP) [10]. Although cells' formation by considering the parts and the machines is the essence of group technology, its full advantages cannot be achieved without including the human factor [11]. Because of the workers' essential role, grouping workers with similar expertise and skills to produce similar families of parts can improve the CMS design quality. When the worker's dimension is considered in addition to the part and the machine dimensions, the problem is transformed into a GCCFP [3].

1.3 Related Work

In the literature, a wide variety of CFPs have been described, and many techniques and algorithms have been proposed to solve them, including heuristics, meta-heuristics, exact methods, etc. [8, 7]. The use of exact methods to solve CFPs provides the best existing solutions. However, due to these combinatorial problems' NP-hard nature, as the problem's dimensions increase, these exact methods become incredibly costly in time and memory consumption. For that reason, meta-heuristic techniques are considered more convenient to solve

NP-hard problems and to produce acceptable solutions in a reasonable time. The following review of related work is established in [3].

1.3.1 Basic Cell Formation Problem

Adil et al. [12] have developed a new non-linear mathematical programming model for CFP. The model's objective is to minimize the sum of voids and exceptional elements within and between cells. To simultaneously identify families of parts and groups of machines, the authors developed an Assignment Allocation Algorithm (AAA) and a Simulated Annealing (SA) algorithm. The authors modified voids' weights and exceptional elements to allow multiple configurations, thus providing designers with multiple possible solutions. Tavakkoli-Moghaddam et al. [13] used the SA algorithm to solve the CFP. The authors considered two types of cells: common or general cells and specific cells. The difference between these two types is that common cells can produce different products (parts). However, only one type of product can be processed by a specific cell. Neto and Gonçalves Filho [14] suggested a multi-objective approach. The objective of this approach is to build cells to minimize simultaneously three contradictory objectives, namely (i) the level of the work-in-process, (ii) the inter-cell moves, and (iii) the total machinery investment. The authors used the Genetic Algorithm (GA) to solve the CFP. They adopted the Pareto optimality principle in the solution procedure to cope with the conflicting objectives. Shiyas and Pillai [15] proposed a new mathematical model for designing manufacturing cells. The model considers two contradictory objectives, such as the inter-cell moves and the cells' heterogeneity. To solve the model, the authors developed a GA-based method. In order to propose alternative cell configurations to decision-makers, a weighting parameter is assigned to the heterogeneity of the objective function of the model. Hafezalkotob et al. [16] proposed a hybrid algorithm to solve the CFP. The algorithm is a combination of Discrete Particle Swarm Optimization (DPSO) and SA. The purpose of coupling these two algorithms is to ensure rapid convergence by DPSO and bring out the search from the local optimum by SA. Danilovic and Ilic [17] has developed a new hybrid algorithm called Cell Formation OPTimization (CFOPT) to solve the CFP. The algorithm's strategy is to use the specificity of the input instances to reduce the set of possible solutions to increase the optimization process's efficiency. Mahmoodian et al. [18] presented a new algorithm based on Particle Swarm Optimization (PSO). The algorithm integrates the self-organization map neural networks to the PSA algorithm. Karoum and Elbenani [19] combined a local search mechanism with the cuckoo search algorithm to intensify the search and improve solutions' grouping efficacy.

1.3.2 Generalized Cell Formation Problem

The GA is used in many works [20, 21, 22, 23]. The SA algorithm is also widely used to solve the problem [21, 24, 25]. To solve the GCFP, Vin et al. [20] proposed a solution entitled Multi-Objective Grouping Genetic Algorithm (MOGGA), which is the combination of GA with an integrated heuristic. The authors used the GA to solve the routing selection problem (to select the most appropriate parts' processing plan). However, the integrated heuristic is combined to address the CFP simultaneously. Ameli et al. [26] used a mathematical programming method named Branch and Bound (B&B) to solve the GCFP. This method, like other exact methods, is not capable of effectively solving large-scale problems. Wu et al. [21] considered a GCFP model that takes as input a binary incidence matrix. This matrix indicates each part's process plans, where each plan mentions the different machines required by the concerned part without establishing an order between them. To solve the model, the authors combined GA with the SA algorithm. In contrast, in the GCFP model solved by Chung et al. [27], the authors consider the sequences of operations, the alternative routing of the

parts, and the reliability of machines (machine failure). Taking machine failures into account during the design of CMS contributes in improving the system's overall performance. The authors combined two techniques to solve the model: Tabu Search (TS) and GA's mutation operator. The use of this operator is justified by its ability to escape local solutions and prevent premature convergence. Jouzdani et al. [25] have extended the model presented in [26] to consider set-up costs. The authors have applied a meta-heuristic method, which is the SA algorithm, to solve the GCFP. Karoum and Elbenani [28] proposed a method entitled Hybrid Selection Algorithm-Generalized Cell Formation (HCSA-GCF). The method aims to reduce the costs of intra-cellular part movements and machine breakdowns. The authors compared the obtained results with those provided by B&B under LINGO software. Hazarika and Laha [23] used a GA heuristic to solve the GCFP with multiple process routes, operation sequences, and parts volume. Five benchmark problems have been used to show the performance of the method.

1.3.3 Cubic Cell Formation Problem

Before 1993, all studies were on two-dimensional manufacturing CFP. Despite the importance of the human dimension, most studies only considered the dimensions of parts and machines. The central issue has been the grouping of similar parts into part families and machines into machine cells [3]. The cubic CFP, which includes the worker (operator) as a third dimension, was first introduced by MIN and SHIN [11]. The authors considered that the group technology cell workers must be multi-disciplined and highly-skilled to work on different machines and perform various tasks. Li [29] presented a new algorithm to solve cubic CFP. This algorithm's added value is to organize all incidence matrices of the problem that links parts to machines, machines to workers, and workers to parts into a single symmetrical incidence matrix. Mahdavi et al. [30] introduce an integer mathematical programming model for the cellular manufacturing system design in a dynamic environment. In the model, the authors took into account multi-period production planning and dynamic reconfiguration of the system. Nikoofarid and Aalaei [31] presented a new mathematical model for a CFP in production planning in a dynamic virtual cellular manufacturing system. The proposed model includes the worker dimension and considers as objectives the minimization of the holding and backorder costs and the management of machines and workers over a specific planning horizon. Mahdavi et al. [32] used the B&B method of the LINGO software package to solve the CFP model. In the model, the authors considered two objectives: the minimization of voids and exceptional elements. The mathematical model catches workers' skills in performing different tasks. Aalaei and Shavazipour [33] defined an integer mathematical programming model for designing the cellular manufacturing systems under data envelopment analysis. The authors tried to minimize the costs of backorders and inter-cellular movement costs produced by exceptional elements. Bootaki et al. [34] proposed a new multi-objective mathematical model for cubic binary CFP. In the model, the authors introduced a new objective called "Quality Index". This objective measures the quality of the parts produced. To calculate the value of this index, data measuring different workers' skills in producing particular parts on special machines must be performed. The authors developed a hybrid genetic algorithm, AUGMented ε-CONstraint (GA-AUGMECON) method to solve the model. Bootaki et al. [35] developed a new multi-objective mathematical model to design dynamic cubic binary CFP. In the model, the authors consider the machine and the concept of worker utilization. To solve the model, the authors have developed a new goal programming method called Percentage Multi-Choice Goal Programming (PMCGP). Motivated by the inefficiency of exact methods to solve large-sized test problems, Sahin and Alpay [36] proposed a GA to solve cubic binary CFP. Taguchi's method was used as a statistical optimization technique to define the parameters' level. Feng et al. [37] consider that the human factor is essential for successfully implementing cellular manufacturing systems. In the proposed model, in opposite to [34] and [35], the author considered operation sequences and alternative process routings. The author also included in

the model the simultaneous consideration of production scheduling, lot splitting, workload balancing between cells, and worker over-assignment to multiple cells. A hybrid approach combining Combinatorial Particle Swarm Optimization and Linear Programming (CPSO-LP) has been proposed to solve the model's real-sized problems efficiently. Bagheri et al. [38] presented a multi-period CFP in a dynamic environment to maximize the total value of grouping efficacy and minimize the total costs and total non-interest workers in cells. The principal idea is to improve the cells' efficiency by assigning workers who have a mutual interest in working with each other. The authors did not consider sequences of operations. A Revised Multi-Choice Goal Programming (RMCGP) method was used to solve the proposed multi-objective mathematical model.

1.4 Generalized Cubic Cell Formation Problem Formulation

We adopt the assumptions and the formulation provided in [3].

1.4.1 Assumptions

The GCCFP is studied according to the following hypotheses :

- The number of cells, the upper, and the lower number of machines in each cell are known and considered as parameters.
- The quality of treating each part on each machine by each worker is specified using a three-dimensional matrix. This matrix values are integers between 1 and 5 (representing very bad, bad, medium, well, very well). The value 0 indicates that a given part cannot be processed on a given machine by a given worker. These values can be estimated by analyzing the historical acquired data and worker errors. During the initial designing of the production system layout, the same quality level can be assigned to each worker. It is also possible to estimate the level of quality by analyzing his qualifications and experience. After that, a continuous evaluation can be planned to acquire the necessary data on the workers' ability to produce parts and handle machines. These data may help later for a future update of the system configuration.
- The machines' and workers' capacity is not considered.
- Each part type has at least one processing route. Exactly one route will be set up to produce this part.
- There are a single machine and a single worker of each type.
- Parts may move within and between cells. The inter-cellular movement is produced if two consecutive operations are executed on the selected route of a given part in two different cells. However, the intracellular movement is incurred when two consecutive operations of a part are performed in the same processing cell.
- The material Handling cost of a given design is the sum of the intercellular and the intracellular movement of the parts.
- The inter-cellular movement of workers is calculated according to the availability or the absence of the workers in the processing cells.
- A worker can work on several machines.

- A part may be processed by multiple workers, but an operation of a part is assigned to a single worker, and it is performed on a single machine.

1.4.2 The constants

To formulate GCCFP, these notations are used:

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.
CI_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.
ap_{rsm}	a binary parameter indicating 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 i when it is processed on machine m by worker w.

The GCCFP resolution consists of four decisions to be taken :

1. The selection of a single route for each part.
2. The assignment of each machine to a single cell.
3. The allocation of each worker to a single cell.
4. The specification of which worker will perform a given operation of a given part, on which machine, and within which cell.

1.4.3 The decision variables

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

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

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

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

1.4.4 The mathematical model

$$\begin{cases} min & InterCMHC\ +\ IntraCMHC\ +\ InterCWM \\ max & Quality \end{cases} \tag{1.1}$$

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

$$IntraCMHC = \sum_{k=1}^{C}\sum_{p=1}^{P} CI_p * \left[\sum_{r=1}^{R_p}\sum_{s=1}^{Oppr-1}\left[(\sum_{m=1}^{M}\sum_{w=1}^{W} X_{prsmwk}) * (\sum_{m=1}^{M}\sum_{w=1}^{W} X_{prs+1mwk})\right]\right] \tag{1.3}$$

$$InterCWM = \sum_{k=1}^{C}\sum_{p=1}^{P}\sum_{r=1}^{R_p}\sum_{s=1}^{Op_{pr}}\sum_{m=1}^{M}\sum_{w=1}^{W} CW_w * X_{prsmwk} * (1 - Z_{wk}) \tag{1.4}$$

$$Quality = \sum_{k=1}^{C}\sum_{p=1}^{P}\sum_{r=1}^{R_p}\sum_{s=1}^{Op_{pr}}\sum_{m=1}^{M}\sum_{w=1}^{W} q_{pmw} * X_{prsmwk} \tag{1.5}$$

Subject to:

$$X_{prsmwk} \leq R_{pr} * a_{prsm} * Y_{mk} * b_{mw} * c_{wp} \qquad \forall(p, r, s, m, w, k) \tag{1.6}$$

$$\sum_{k=1}^{C}\sum_{m=1}^{M}\sum_{w=1}^{W} X_{prsmwk} = R_{pr} \qquad \forall(p, r, s) \tag{1.7}$$

$$\sum_{r=1}^{R_p} R_{pr} = 1, \qquad \forall p \tag{1.8}$$

$$\sum_{k=1}^{C} Z_{wk} = 1, \qquad \forall w \tag{1.9}$$

$$\sum_{k=1}^{C} Y_{mk} = 1, \qquad \forall m \tag{1.10}$$

$$\sum_{m=1}^{M} Y_{mk} \leq UM, \qquad \forall k \tag{1.11}$$

$$\sum_{m=1}^{M} Y_{mk} \geq LM, \qquad \forall k \tag{1.12}$$

$$X_{prsmwk}, Y_{mk}, Z_{wk}, R_{pr} \in \{0, 1\} \qquad \forall(p, r, s, m, w, k) \tag{1.13}$$

The objective function of the model is given in equation 1.1. It minimizes the material handling cost and the inter-cellular movement of workers and maximizes the produced parts' quality index. The formulas of calculating the inter-cellular material handling cost (InterCMHC), the intra-cellular material handling cost (IntraCMHC), the inter-cellular worker movement (InterCWM), and the quality index are respectively given

in equations 1.2, 1.3, 1.4, and 1.5. The purpose of the model is to find a better compromise between these objectives.

Equation 1.6 imposes that an operation s will be executed on machine m by worker w within cell k only if:

- The route, to which s belongs, is set up to produce the concerned part p (R_{pr}).
- Machine m is required to execute the operation s (a_{prsm}).
- Machine m is already assigned to cell k because machines cannot be moved between cells (Y_{mk}).
- Worker w can use machine m (b_{mw}).
- Worker w can process part p (c_{wp}).

Equation 1.7 guarantees that an operation s is performed at most on a single machine by a single worker in a single cell and will only be executed if the route to which s belongs is set up to produce the part concerned. Equation 1.8 means that only one route is established for each part. Constraint 1.9 confirms that each worker is assigned precisely to one cell. Constraint 1.10 ensures that each machine is assigned to one and exactly one cell. Constraints 1.11 and 1.12 present the minimum and the maximum number of machines that a cell can contain. The last constraint represents logical binary requirements on the decision variables.

1.4.5 Linearisation of the model

The non-linearisation in the proposed model is caused by the first three terms of the objective function (InterCMHC, IntraCMHC, InterCWM), and the constraint 1.6 of the model. The following linearisation is performed by [3]. To linearize the model, four auxiliary binary variables are used:

- $F_{prsmm'ww'kk'}$ = X_{prsmwk} * $X_{prs+1m'w'k'}$ $\quad k \neq k'$, s $\leq$ Op_{pr}-1
- $I_{prsmm'ww'k}$ = X_{prsmwk} * $X_{prs+1m'w'k}$ $\quad s \leq Op_{pr}$-1
- J_{prsmwk} = X_{prsmwk}*(1-Z_{wk})
- L_{prmk} = R_{pr} * Y_{mk}

Thus, the first three terms of the objective function are computed like that:

$$InterCMHC = \sum_{k \epsilon T} \sum_{k' \epsilon T \setminus \{k\}} \sum_{p=1}^{P} CO_p * \sum_{r=1}^{R_p} \sum_{s=1}^{Op_{pr}-1} \sum_{m=1}^{M} \sum_{m'=1}^{M} \sum_{w=1}^{W} \sum_{w'=1}^{W} F_{prsmm'ww'kk'} \tag{1.14}$$

$$IntraCMHC = \sum_{k=1}^{C}\sum_{p=1}^{P} CI_p \sum_{r=1}^{R_p}\sum_{s=1}^{Op_{pr}-1}\sum_{m=1}^{M}\sum_{m'=1}^{M}\sum_{w=1}^{W}\sum_{w'=1}^{W} I_{prsmm'ww'k} \tag{1.15}$$

$$InterCWM = \sum_{k=1}^{C}\sum_{p=1}^{P}\sum_{r=1}^{R_p}\sum_{s=1}^{Op_{pr}}\sum_{m=1}^{M}\sum_{w=1}^{W} CW_w * J_{prsmwk} \tag{1.16}$$

The constraint 1.6 is replaced by these three constraints:

$$X_{prsmwk} \leq a_{prsm} * b_{mw} * c_{wp} * L_{prmk} \qquad \forall(p,r,s,m,w,k) \tag{1.17}$$

$$2 * L_{prmk} \leq R_{pr} + Y_{mk} \qquad \forall(p,r,m,k) \tag{1.18}$$

$$L_{prmk} + 1 \geq R_{pr} + Y_{mk} \qquad \forall(p,r,m,k) \tag{1.19}$$

The following additional constraints are used to restrict the introduced variables ($F_{prsmm'ww'kk'}$, $I_{prsmm'ww'k}$, J_{prsmwk}):

$$2 * F \leq X_{prsmwk} + X_{prs+1m'w'k'} \qquad \forall(p,r,s,m,m^{'},w,w^{'},k,k^{'}), k \neq k^{'} \tag{1.20}$$

$$F + 1 \geq X_{prsmwk} + X_{prs+1m'w'k'} \qquad \forall(p,r,s,m,m^{'},w,w^{'},k,k^{'}), k \neq k^{'} \tag{1.21}$$

$$2 * I \leq X_{prsmwk} + X_{prs+1m'w'k} \qquad \forall(p,r,s,m,m^{'},w,w^{'},k) \tag{1.22}$$

$$I + 1 \geq X_{prsmwk} + X_{prs+1m'w'k} \qquad \forall(p,r,s,m,m^{'},w,w^{'},k) \tag{1.23}$$

$$2 * J \leq X_{prsmwk} + 1 - Z_{wk} \qquad \forall(p,r,s,m,w,k) \tag{1.24}$$

$$J \geq X_{prsmwk} - Z_{wk} \qquad \forall(p,r,s,m,w,k) \tag{1.25}$$

1.5 Conclusion

In this chapter, an overview of the Cell Formation Problem is presented. Initially, we have defined its basic version. After, we have presented a definition of the version that we will consider in our study, which is the Generalized Cubic Cell Formation Problem. Next, a study of the related work is provided. Finally, a mathematical formulation of the Generalized Cubic Cell Formation Problem is given.

Our problem belongs to the NP-hard class. The problems of this class are algorithmically solvable but computationally intractable. There is no exact method that can find the optimal global solutions to NP-hard problems in polynomial time. Fast approximate heuristics and meta-heuristics are the popular approaches to search for practical solutions. In our study, we will use the genetic algorithm, which is one of the most popular meta-heuristics, often used to solve complex large-scale optimization problems. So in the next chapter, we will give an overview of the genetic algorithm.

Chapter 2

Genetic Algorithms

2.1 Introduction

In many real-life settings, high-quality solutions to hard optimization problems are required in a short amount of time. Due to the practical importance of the combinatorial optimization problems for industry and science, many algorithms to tackle them have been developed [39]. In combinatorial optimization (CO), algorithms can be classified as either exact or approximate algorithms. In approximate methods such as metaheuristics, we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time. Thus, the use of metaheuristics has received more and more attention in the last decades.

The term metaheuristic was first introduced in [40]. A metaheuristic is an iterative generation process that guides a subordinate heuristic by combining intelligently different concepts to explore and exploit the search space to find efficiently near-optimal solutions [41]. Metaheuristics may be classified into methods that perform a single solution vs. population-based search. This classification refers to the number of solutions used by a metaheuristic at any time. Generally, algorithms that work on a single solution at any time are referred to as trajectory methods. They all share the property that the search process describes a trajectory in the search space (e.g., tabu search, iterated local search, and simulated annealing). Population-based metaheuristics deal at each algorithm iteration with a set of solutions rather than with a single one. From this set of solutions, the next iteration population is produced by the application of some operators. Population-based metaheuristics provide a natural, intrinsic way for the exploration of the search space. However, the final performance strongly depends on the way the population is manipulated. The most studied population-based methods are evolutionary computation (EC) and ant colony optimization (ACO) [39].

EC can be regarded as a metaphor for building, applying, and studying algorithms based on Darwinian natural selection principles. The instances of algorithms based on evolutionary principles are called Evolutionary Algorithms (EA) [42]. EAs can be characterized as computational models of evolutionary processes. There has been a variety of slightly different EAs proposed over the years. In our work, we will use an evolutionary algorithm to solve the Generalized Cubic Cell Formation Problem. This evolutionary algorithm is the Genetic Algorithm.

In this chapter, we briefly introduce the genetic algorithms. In section 2.2, we give some definitions and terminology. In section 2.3, we exhibit the basic genetic algorithms. In section 2.4, we discuss the genetic algorithm operators. Finally, in section 2.5, we conclude.

2.2 Definitions and Terminology

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 the principles of natural selection proposed by Darwin and natural genetics.

GA was initially introduced by John Holland, his colleagues, and his students at the University of Michigan [4]. Their research goals have been twofold: (i) to abstract and rigorously explain the adaptive processes of natural systems, and (ii) to design artificial systems software that retains the important mechanisms of natural systems. This approach has led to important discoveries in both natural and artificial systems science. Goldberg presented the fundamentals of GAs and described its usual form [43].

GAs have been successfully applied to many optimization problems in different disciplines that are difficult to solve by classical mathematical programming [14, 15, 20, 22, 23, 34, 36, 44, 45, 46, 47, 48, 49]. In the following sections, some important terminology and concepts of GA are presented.

2.2.1 Genes and Chromosomes

The gene is the basic component of the GA. A string of genes is called a chromosome. Chromosomes can be encoded as binary strings, as strings of real numbers, etc.

2.2.2 Populations and Generations

A population is a set of chromosomes. GA begins with a set of randomly created individuals (chromosomes). This set is called the **initial population**. The iterations of GA are called generations. Each iteration involves selecting individuals with closely related characteristics and recombining them until a new generation is created to replace the old one [50].

2.2.3 Parents and Children

The selection of chromosomes from one generation to another consists of choosing individuals in a probabilistic method [50]. Those with high fitness values have a high probability of being selected to undergo crossover and produce new chromosomes called children or offsprings. The crossover happens with a priori fixed probability called crossover rate. It includes a random selection of the parent chromosomes' crossover points, where the mixing of parent's genetic information should be happening.

2.2.4 Mutation

The mutation is a process by which many new points are introduced into the search space. It ensures that aggressive selection does not result in a suboptimal solution. In other words, it prevents premature convergence

to a local optimum. It is achieved by randomly changing some chromosome characteristics and is carried out at very low probability values (mutation rate).

2.2.5 Fitness

The objective function that defines the optimization purpose is called the fitness function. It indicates "goodness" or "badness" for each individual.

2.2.6 Elitism

To improve GA's performance, the best individuals must always participate in reproduction. However, such individuals can be lost if they are destroyed by crossover or mutation operators. Thus, the first best chromosome or the few best chromosomes are copied into the new population [51].

2.3 A Basic Genetic Algorithm

In general, a genetic algorithm must be able to achieve six basic tasks [52] :

1. Encoding the solution elements in the form of genes.
2. Create a string of genes to form a chromosome.
3. Initialize a starting population by generating a set of specific chromosomes, usually randomly.
4. Evaluate and assign fitness values to individuals in the population.
5. Perform reproduction by the fitness weighted selection of individuals of the population.
6. Perform recombination and mutation to produce individuals of the following generation.

A GA, then, is an iterative optimization method that simulates the adaptation and evolution of a single kind of organism. Using a chromosomal mapping system, the GA starts with a large number of possible design configurations. The range of potential configurations is defined by the limitations of the problem and the method of encoding all configuration information into the chromosome [50, 53].

A typical GA is represented in Figure 2.1

To start the optimization, the GA selects a set of configurations, almost always at random. This set is called the initial population, just as in biology. The GA evaluates the performance of each individual of the population using a cost function that compares the individual's performance to the desired or ideal performance and returns to the GA a single number that is a measure of its fitness. As in the evolutionary process of "survival of the fittest", high-quality strings combine and produce offspring, while low-quality strings are removed from the population[52]. Offspring can be generated by many different methods, each of which is essentially a method of combining information from two or more parent chromosomes to form a child with the potential to surpass

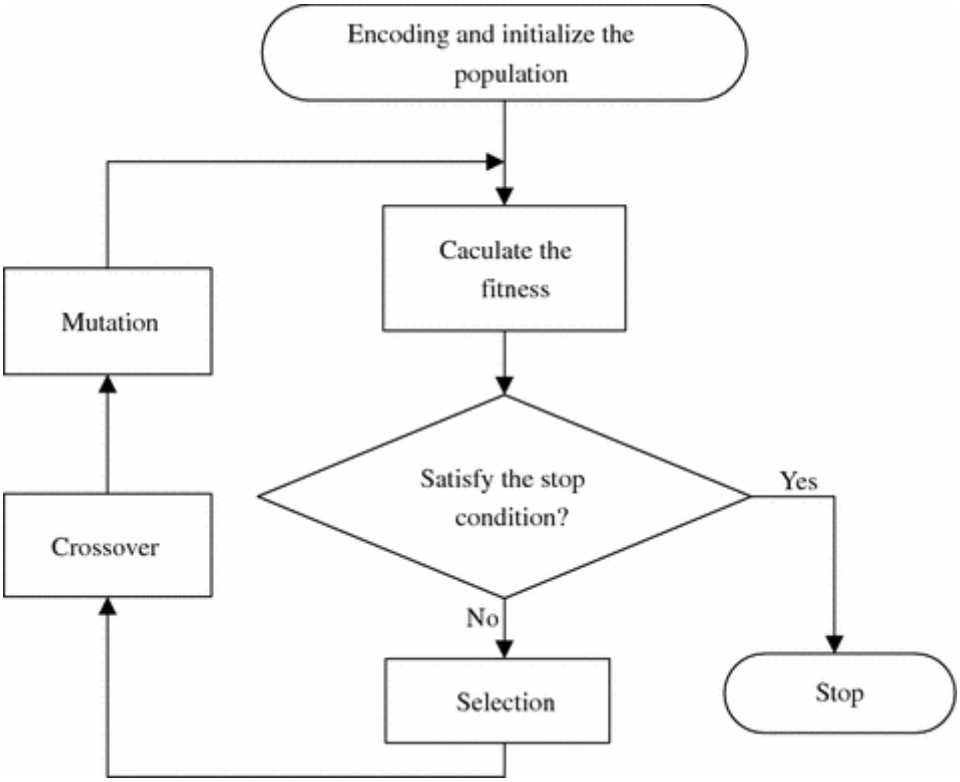


Figure 2.1: The basic process of genetic algorithm

its parents. With succeeding generations, the individuals' quality is continuously improved, and an optimized solution is finally reached. "Champions" will have many offsprings, while those who do not perform well will die without offspring. In this way, after some generations, a good solution is usually achieved [54].

2.4 GA Operators

The tasks that a genetic algorithm must complete and that were outlined in the previous section guide to the presence of three phases in the genetic algorithm optimization [50].

- Initiation;
- Reproduction;
- Generation replacement.

2.4.1 Initiation

Initiation means filling the initial population with encoded parameter strings or chromosomes, usually generated randomly. The coding is a mapping from parameter space to chromosome space [54].

2.4.1.1 Encoding

Encoding is a process of representing individual genes. The process can be performed using bits, numbers, trees, arrays, lists, or other objects. The encoding depends mainly on solving the problem [55].

An encoding function is used to represent the object variables' mapping to a string code. The mapping of string code to its object variable is achieved through the decoding function, as shown in Figure 2.2 [56].

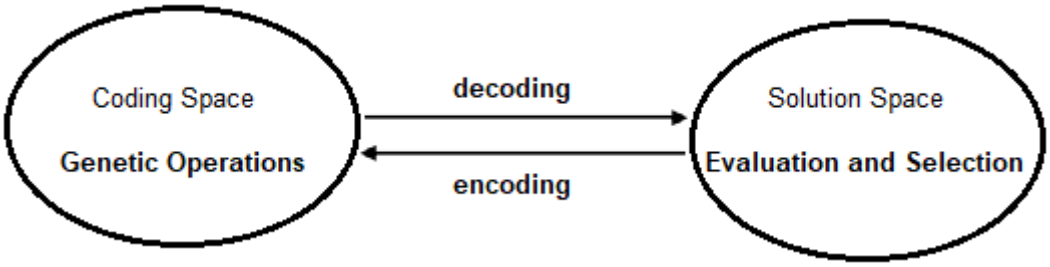


Figure 2.2: Encoding – Decoding method

1. **Binary Encoding**

   The most common way of encoding is a binary string, which would be represented as in Figure 2.3.

   Each chromosome is encoded in the form of a binary string. Every bit in the string may represent some characteristics of the solution. Each string, therefore, is a solution but not necessarily the best solution. Another possibility is that the entire string may represent a number.

   Binary encoding gives many potential chromosomes with a smaller number of alleles [55].

Chromosome 1	1	1	0	1	0	0	0	1
Chromosome 2	0	1	1	1	1	1	1	1

Figure 2.3: Binary encoding

2. **Hexadecimal Encoding**

   This encoding uses a string composed of hexadecimal numbers (0–9, A–F).

Chromosome 1	9	C	E	7	0	A	2	F
Chromosome 2	3	D	B	5	1	4	D	6

Figure 2.4: Hexadecimal encoding

3. **Real Number Encoding**

   The real number encoding is usually used for ordering issues. In this type of encoding, each chromosome represents a sequence of reals; for example, in the traveling salesman problem, the string of numbers represents the sequence of cities visited by the salesman [56]. Figure 2.5 shows an example of the real number encoding.

Chromosome 1	1	5	3	2	6	4	7	9
Chromosome 2	8	5	6	7	2	3	1	4

Figure 2.5: Real number encoding

There are other sorts of encoding, such as octal encoding, value encoding, tree encoding, etc. For more information about these types, an illustrated is provided in [55].

2.4.1.2 Fitness Function

A major problem in optimization is the formulation or the choice of an appropriate fitness function that determines the selection criterion in particular problems. For minimizing a function using genetic algorithms, a simple way to create a fitness function is to use 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 $y = f(x)$. The objective is to maximize the fitness function and then minimize the objective function $f(x)$. Alternatively, for a minimization problem, we can define a fitness function $F = 1/f(x)$, but it can have a singularity when $f \to 0$. There are many different ways to define a fitness function [57].

The appropriate form of the fitness function will ensure that solutions with higher fitness are selected efficiently. A poor fitness function may result in wrong or meaningless solutions.

2.4.2 Reproduction

It consists of three principal operators: selection, crossover, and mutation. These operators are discussed in the following.

2.4.2.1 Selection Strategies

The selection consists simply of choosing the best individuals to crossover. It aims to take advantage of these individuals' good characteristics by considering their fitness values, which is a measure of "goodness". In theory, there are many selection strategies; however, the most commonly used schemes are described in what follows [54].

1. **Population Decimation**

   This scheme relates to the so-called deterministic strategies [52]. The idea behind this method is simply the survival of the fittest with the elimination of the weakest fit. Since the population is decimated before being replaced by reproduction, this method is called population decimation. An arbitrary minimum fitness is chosen as the cutoff point, and any individual with a lower fitness is eliminated from the population. As an example of population decimation, consider the individuals in Figure 2.6 who are ranked, then population decimation consists of keeping the 50% best individuals. Thus, the results in the lower table of Figure 2.6 are obtained. Notice that the individuals whose values are below the threshold value are all rejected, which is a disadvantage because these individuals may possess some good characteristics that could have been obtained by crossover and/or mutation processes in the following generations.

   The advantage of population decimation is its simplicity. However, it has the disadvantage that: once an individual is eliminated from the population, any unique characteristics that this individual holds are lost [50]. For this reason, stochastic selection techniques have been developed.

2. **Proportionate Selection**

   This method is also known as the roulette wheel. Its philosophy is that individuals are selected based on a selection probability given by equation 2.1 [50, 52, 53].

$$P_{selection} = \frac{f(parent_i)}{\sum_i f(parent_i)} \tag{2.1}$$

   Where :

   - $P_{selection}$ is the probability of an individual parent being selected.
   - $f(parent_i)$ is the fitness value of $parent_i$.

   Initially, individuals are sorted according to their fitness. Then, the probabilities of the different individuals are calculated using equation 2.1. These probabilities are classified into a vector that contains the cumulative sums of these probabilities. A random number (between 0 and 1) is "thrown" (as a die roll), and depending on its value, will choose the individual who will participate in the crossover. Figure 2.7 illustrates the proportionate selection [58], while Table 2.1 shows this method's application to the individuals in Figure 2.6.

   It should be noted that the most qualified individuals have a higher probability of being selected to mate in this selection strategy. This drives to the problem of one or more individuals will dominate the next generations. Finally, the algorithm will saturate, i.e., at a certain generation. Only a group of individuals that are all the same will be found. The following selection strategy overcomes this problem [54].

Individual	Chromosome	Fitness
I_1	010010	5
I_2	101100	22
I_3	101111	11
I_4	001010	3
I_5	010101	6
I_6	101010	17
I_7	110110	1
I_8	000111	9

Rank	Individual	Chromosome	Fitness
1	I_2	101100	22
2	I_6	101010	17
3	I_3	101111	11
4	I_8	000111	9
5	I_5	010101	6
6	I_1	010010	5
7	I_4	001010	3
8	I_7	110110	1

Rank	Individual	Chromosome	Fitness	
1	I_2	101100	22	keep
2	I_6	101010	17	keep
3	I_3	101111	11	keep
4	I_8	000111	9	keep
5	I_5	010101	6	discard
6	I_1	010010	5	discard
7	I_4	001010	3	discard
8	I_7	110110	1	discard

Figure 2.6: An illustrative example of the population decimation

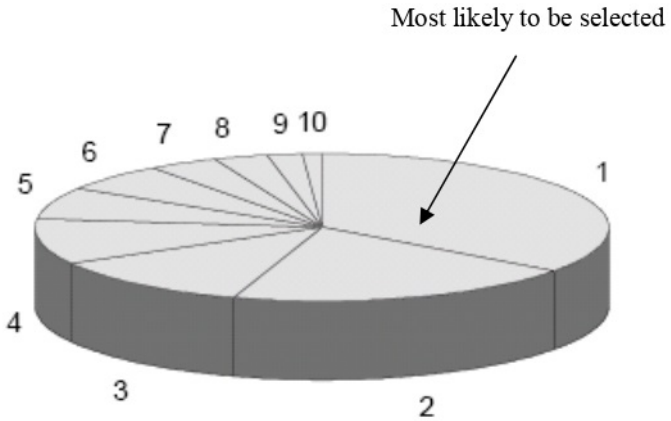


Figure 2.7: Proportionate selection represented as a roulette wheel

Rank	Individual	Chromosome	Cost	probability
1	I_2	**101100**	**22**	**0.282**
2	I_6	**101010**	**17**	**0.218**
3	I_3	**101111**	**11**	**0.141**
4	I_8	**000111**	**9**	**0.115**
5	I_5	**010101**	**6**	**0.077**
6	I_1	**010010**	**5**	**0.064**
7	I_4	**001010**	**3**	**0.038**
8	I_7	**110110**	**1**	**0.013**

Table 2.1: Application of proportionate selection to the individuals in Figure 2.6

3. **Rank Selection**

   The Roulette wheel will have a problem when the fitness values differ very much. If the best chromosome fitness is 90%, its circumference occupies 90% of the Roulette wheel, and then other chromosomes have too few chances to be selected. Rank Selection ranks the population, and every chromosome receives fitness from the ranking. The worst has fitness 1, and the best has fitness N. It results in slow convergence but prevents too quick convergence. It also keeps up selection pressure when the fitness variance is low. It preserves diversity and hence leads to a successful search. In effect, potential parents are selected, and a tournament is held to decide which of the individuals will be the parent. There are many ways this can be achieved, and two suggestions are [55],

   (a) Select a pair of individuals at random. Generate a random number, R, between 0 and 1. If $R < r$ use the first individual as a parent. If $R >= r$, then use the second individual as the parent. Moreover, this is repeated to select the second parent. The value of r is a parameter to this method.

   (b) Select two individuals at random. The individual with the highest evaluation becomes the parent. Repeat to find a second parent.

4. **Tournament Selection**

   An ideal selection strategy should be able to adjust its selective pressure and population diversity to fine-tune GA search performance. Unlike the Roulette wheel selection, the tournament selection strategy provides selective pressure by holding a tournament competition among N_u individuals.
   The best individual from the tournament is the one with the highest fitness, which is the winner of N_u. The winner is then inserted into the mating pool. The tournament competition is repeated until the mating pool for generating new offspring is filled. The mating pool comprising of the tournament winner has higher average population fitness. The fitness difference provides the selection pressure, which drives GA to improve the fitness of the succeeding genes [55].

2.4.2.2 Crossover Strategies

The crossover operator is usually the primary operator working only as a mechanism to introduce variety into the population. The schemes differ from binary to real number encoding.

1. **Binary GA Crossover**

For binary encoding GA, there are many ways to perform a crossover. The selected parents simply interchange parts of their chromosomal structure according to one or more randomly set interchange points, called crossover sites. The number of exchange points is left to the programmer's discretion[54].

- **Single Point Crossover**

After reproduction, the crossover can be done in two steps. First, members of the recently reproduced chromosomes in the mating pool are mated randomly. Then, each pair of chromosomes undergoes crossover as follows: an integer position k on the chromosome is selected uniformly, randomly chosen between 1 and the length of the chromosome minus one. Two new chromosomes are generated by swapping all the genes between k+1 and the length of the chromosomes [43] (see Figure 2.8).

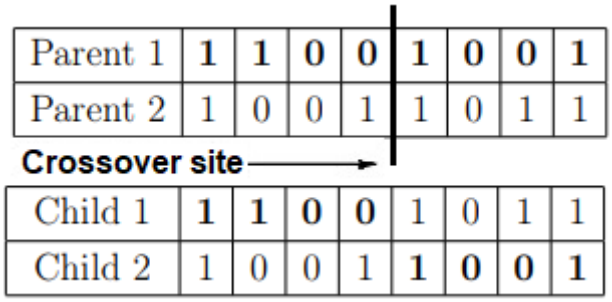


Figure 2.8: Single point crossover

- **Two-Point Crossover**

Two-point crossover is very similar to single-point crossover, except that two cutoff points are randomly generated instead of one (see Figure 2.9).

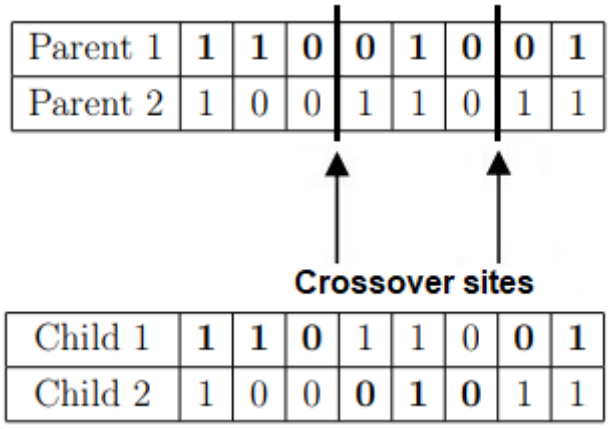


Figure 2.9: Two-point crossover

- **Multi-Point Crossover (N-Point Crossover)**

  There are two cases in this crossover. The first one is an even number of crossover sites; they are randomly selected around a circle, and information is exchanged. The other case is an odd number of crossover sites.

- **Uniform Crossover**

  Uniform crossover is very different from the multi-point crossover. Each gene in the offspring is generated by copying the corresponding gene from parents according to a randomly created binary crossover mask of the same length as the chromosomes. When there is a 1 in the crossover mask, the gene is copied from the first parent, and when there is a 0 in the mask, the gene is copied from the second parent. A new crossover mask is randomly generated for each pair of parents. Offsprings, therefore, contains a mixture of genes from each parent. The number of active crossover points is not fixed but is the average of L/2 (where L is the length of the chromosome) [55].
  In Figure 2.10, new children are produced using the uniform crossover method. It can be seen that in the production of child 1, when there is a 1 in the mask, the gene is copied from parent 1, otherwise from parent 2. In the production of child 2, when there is a 1 in the mask, the gene is copied from parent 2. Else the gene is copied from parent 1.

Parent 1	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
Parent 2	0	0	0	1	1	0	1	0
Mask	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**

Child 1	**1**	**0**	0	**1**	1	**0**	**1**	0
Child 2	0	0	**1**	1	**0**	0	1	**1**

Figure 2.10: Uniform crossover

- **Three Parent Crossover**

  In this crossover technique, three parents are chosen at random. Each bit of the first parent is compared with the bit of the second parent. If the two are identical, the bit is taken for the offspring. If not, the bit of the third parent is taken for the offspring. This concept is illustrated in Figure 2.11.

Parent 1	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
Parent 2	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**
Parent 3	1	1	0	1	0	1	1	0

Child	1	**0**	0	**1**	0	**0**	**1**	0

Figure 2.11: Three parent crossover

2. **Crossover Techniques in Order Coded GA**

Binary crossover techniques are not applicable to order coded GA. For example, in Figure 2.12, by applying binary single-point crossover, the obtained offspring chromosomes are not valid.

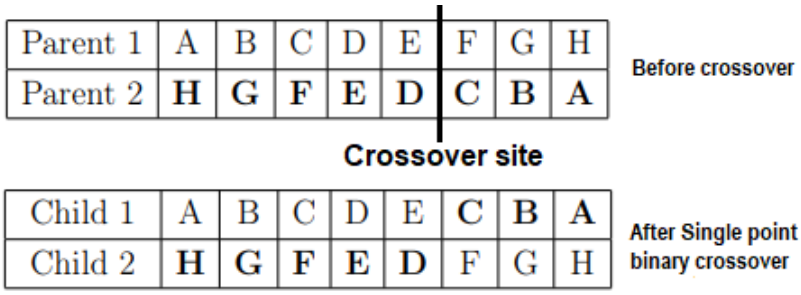


Figure 2.12: Crossover in order coded GA

Since the sequence of gene values is important, Binary crossover techniques are not applicable to order coded GA.

- **Order Crossover (OX)**

  - **Single Point Order Crossover**

    With two parents and a random crossover point. The single point order crossover behaves as follows:
    Child 1 inherits its left section from parent 1, and child 2 inherits its left section from parent 2. For the right section of child 1, copy the gene value from parent 2 in the same order as they appear but not already present in the left section. For the right section of child 2, copy the gene value of parent 1 in the same order as they appear but are not already present in the left section (see Figure 2.13).

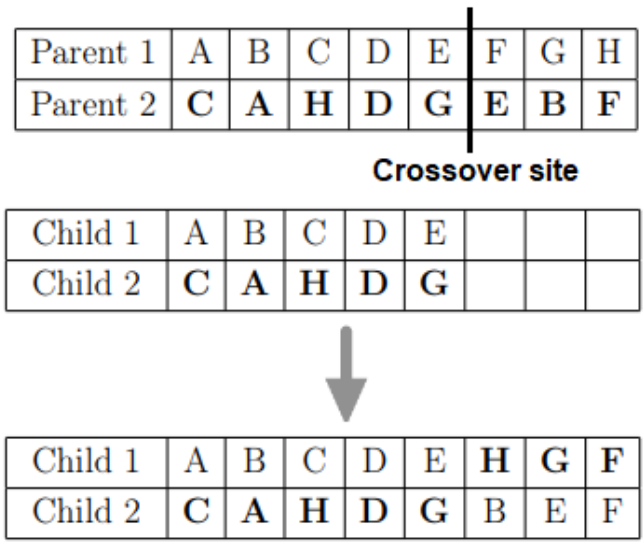


Figure 2.13: Single-point order crossover

– **Two Point Order Crossover**

In the presence of two chromosomes as parents, and two random crossover points are selected, separating them into a left, middle, and right portion. The ordered two-point crossover behaves as follows:
Child 1 and child 2 inherit their middle section from parent 1 and parent 2, respectively. The left and right sections of child 1 are determined by the genes of the left and right sections of parent 1 in the order in which the values appear in parent 2. A similar process is applied to determine child 2. The process is illustrated in Figure 2.14.

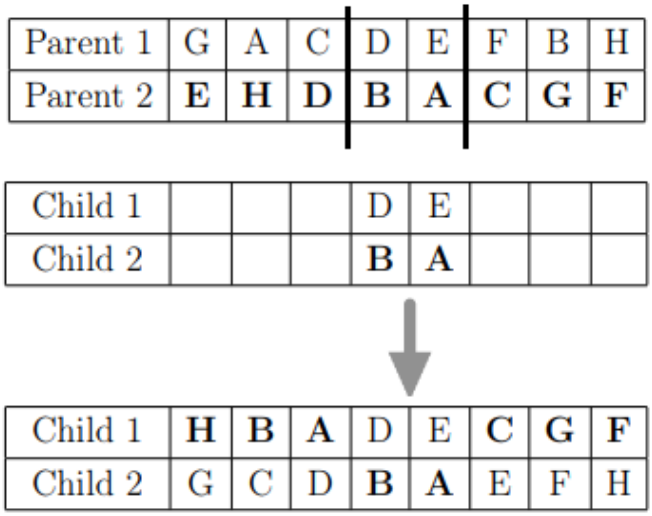
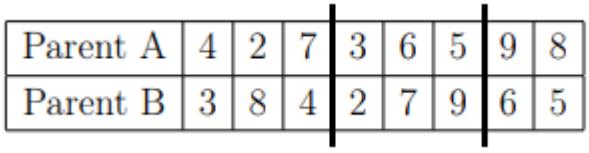


Figure 2.14: Two-point order crossover

- **Partially Matched Crossover (PMX)**

In Partially Matched Crossover, two strings are aligned, and two crossover points are selected uniformly at random along the length of the strings. The two crossover points give a matching selection, which is used to affect across through position by position exchange operations [55].
Consider two strings:



Two crossover points were selected at random, and PMX proceeds by position-wise exchanges. In-between the crossover points, the genes get exchanged, i.e., the 3 and the 2, the 6 and the 7, the 5 and the 9 exchange places. This is by mapping parent B to parent A. Now mapping parent A to parent B, the 7 and the 6, the 9 and the 5, the 2 and the 3 exchange places. Thus after PMX, the offspring produced as follows:

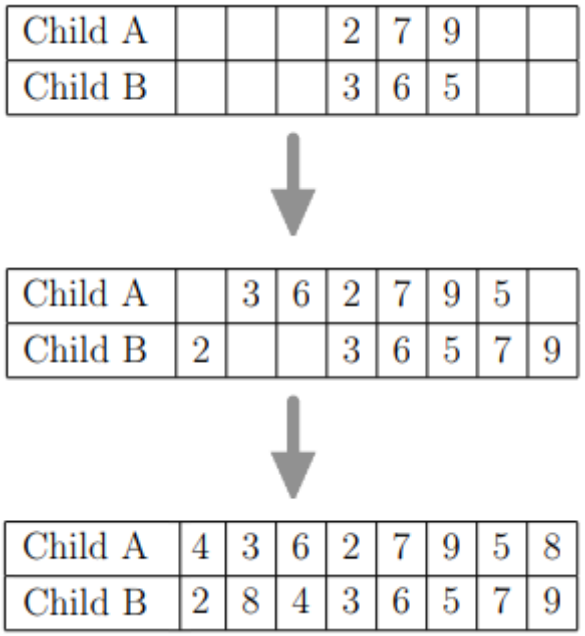


Figure 2.15: Partially matched crossover

- **Precedence Preservative Crossover (PPX)**

  PPX is illustrated in Figure 2.16, for a problem consisting of six genes A–F. The operator works as follows:

  - Create a vector of length equal to chromosomes' length, randomly filled with elements from the set {1,2}. This vector defines the order in which the operations are successively drawn from parent 1 and parent 2.
  - We can also consider the parent and offspring permutations as lists, for which the operations 'append' and 'delete' are defined.
  - First, we start by initializing an empty offspring.
  - The leftmost operation in one of the two parents is selected in accordance with the order of parents given in the vector.
  - After an operation is selected, it is deleted in both parents.
  - Finally, the selected operation is appended to the offspring.
  - This step is repeated until both parents are empty, and the offspring contains all operations involved [55].

2.4.2.3 Mutation Strategies

The Mutation is a background operator that produces spontaneous random changes in various chromosomes. A simple way to achieve mutation would be to alter one or more genes. In GA, mutation serves the crucial role

Parent 1	A	B	C	D	E	F
Parent 2	C	A	B	F	D	E

Select parent (1 or 2)	1	2	1	1	2	2

Offspring permutation	C	A	B	F	D	E

Figure 2.16: Precedence preservative crossover

of either replacing the genes lost from the population during the selection process or providing the genes that were not present in the initial population.

The mutation probability is defined as the percentage of the total number of genes in the population. The mutation probability controls the probability with which new genes are introduced into the population for trial. If it is too low, many genes that would have been useful are never tried out. However, if it is too high, there will be much random perturbation, the offspring will start losing their resemblance to the parents, and the algorithm will lose the ability to learn from the history of the search [59].

1. **Binary GA Mutation**

   It is simply changing a 1 to 0 or vice-versa depending on a probability. In general, the probability of mutation is very small (typically less than 0.2) to avoid losing the chromosomes' good properties (see Figure 2.17).

Before mutation	1	1	**0**	1	0	0	1	0
After mutation	1	1	**1**	1	0	0	1	0

Figure 2.17: Binary mutation

2. **Real GA Mutation**

   Up to now, several mutation operators have been proposed for real numbers encoding.

   - Random mutation*:* operators such as uniform mutation, boundary mutation, and plain mutation belong to the conventional mutation operators, which simply replace a gene with a randomly selected real number with a specified range.
   - Dynamic mutation (non-uniform mutation): is designed for fine-tuning capabilities to achieve high precision, which is classified as the arithmetical mutation operator.

- Directional mutation*:* operator is a kind of direction-based mutation, which uses the gradient expansion of the objective function. The direction can be given randomly as a free direction to avoid the chromosomes jamming into a corner. If the chromosome is near the boundary, the mutation direction given by some criteria might point toward the close boundary, and then jamming could occur.

Several mutation operators for integer encoding have been proposed [59].

- Inversion mutation*:* selects two positions within a chromosome at random and then inverts the substring between these two positions (see Figure 2.18).
- Insertion mutation*:* selects a gene at random and inserts it in a random position (see Figure 2.19).
- Displacement mutation*:* selects a substring of genes at random and inserts it in a random position. Therefore, insertion can be viewed as a particular case of displacement (see Figure 2.20).
- Reciprocal exchange mutation*:* selects two positions random and then swaps the genes on the positions (see Figure 2.21).

2.4.3 Generation Replacement

Once the new offspring solutions are created using crossover and mutation, we need to introduce them to the parental population. There are many ways we can approach this. Bear in mind that the parent chromosomes have already been selected according to their fitness, so we hope that the children (which includes parents who did not undergo crossover) are among the fittest in the population. So we would hope that the population will gradually, on average, increase their fitness. Some of the most common techniques are outlined below [60].

- Delete-all: This technique deletes all the current population individuals and replaces them with the same number of chromosomes that have just been created. This is probably the most common technique and will be the choice technique for most people due to its relative ease of implementation. It is also parameter-free, which is not the case for those listed below.
- Steady-state: This technique deletes **n** old individuals and replaces them with **n** new individuals. The number to delete and replace, n, at any one time is a parameter to this deletion technique. Another consideration for this technique is deciding which individuals to delete from the current population. Do you delete the worst individuals, pick them at random, or delete the chromosomes that you used as parents?
- Steady-state-no-duplicates: This is the same as the steady-state technique, but the algorithm checks that no duplicate chromosomes are added to the population. This adds to the computational overhead but can mean that more of the search space is explored.

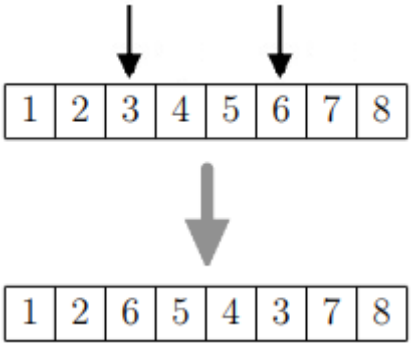


Figure 2.18: Inversion mutation

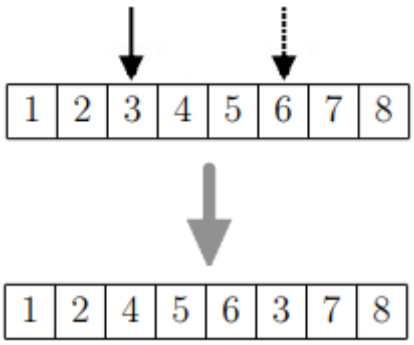


Figure 2.19: Insertion mutation

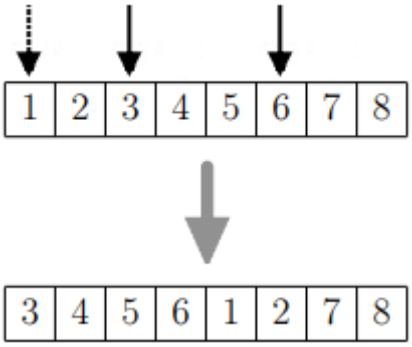


Figure 2.20: Displacement mutation

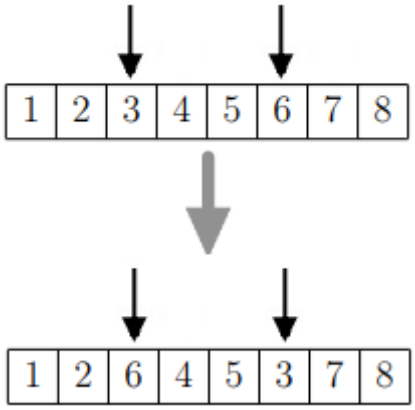


Figure 2.21: Reciprocal exchange mutation

2.4.4 Stopping Criteria

The stopping criteria is an important issue in evolutionary modeling. Early termination may generate poor solutions, whereas late termination might cause high time consumption. The proposed GA is terminated if one of the following cases is reached [61]:

- Maximum number of generations;
- Acceptable fitness reached;
- The maximum number of generations allowed without replacing the fittest reached chromosome.

2.5 Conclusion

In this chapter, an overview of the genetic algorithm and its appearance is provided. Initially, we have given the important definitions and terminology. After we have presented the basic process of genetic algorithms and the tasks must be able to achieve. Next, the genetic algorithm operators are detailed. Finally, the stopping criteria are given.

In the next chapter, we will apply the Genetic Algorithm to the problem already described in chapter 1, which is the Generalized Cubic Cell Formation Problem.

Chapter 3

Our Approach To Solve The Generalized Cubic Cell Formation Problem

3.1 Introduction

In the previous chapters, we have defined the Generalized Cubic Cell Formation Problem, and we have given an overview of the genetic algorithm. In this chapter, we will show how we applied the genetic algorithm to GCCFP. Then, we compare our method with other methods, namely B&B, SA, and DFPA. Thus, this chapter is organized as follow:

In section 3.2, we present the adopted representation and evaluation of the solution. In section 3.3, we detail the solution approach containing a description of the proposed GA. In section 3.4, we exhibit computational results. In section 3.5, we show the application's interface and instances. Finally, we conclude in section 3.6.

3.2 Solution Representation and Evaluation

3.2.1 Solution Representation

In this study, the solution is represented using two vectors and one matrix:

- The first vector (C_Assign) has a size equal to M+W, where M is the number of machines, and W is the number of workers. The first piece includes the cell to which each machine is assigned. However, the second piece models the cell of each worker. By adopting this structure, each worker and each machine can not be assigned to more than one cell because they have precisely one devoted box in the C_Assign vector. This makes constraint 1.9 (it verifies that a worker must be affected to a single cell) and constraint 1.10 (it imposes that a machine must be assigned to a single cell) syntactically preserved. It is still to ensure, during the resolution process, the specification of each worker's cell and each machine's cell.

- The second vector (R_Select) specifies the selected route to process each part. Thus, it has a size equal to P, where P is the number of parts. A single route can be selected for each part by reserving a single box in the R_Select vector. Thus, this structure preserves the feasibility concerning constraint 1.8 (it verifies that a single route is selected to process each part) of the mathematical model.
- Finally, the matrix W_Assign is used to specify the worker in charge of executing each operation. Each operation is defined by the part to which it belongs and the machine on which it is executed. Thus, the matrix has the dimension P×M, and each cell contains at most one worker. The fact of reserving a single box in the W_Assign matrix for each operation of the selected route ensures that each operation can be executed by one worker. To satisfy constraint 1.7, it is still just to ensure during the resolution process that the execution of an operation s happens just if its route r of part p has been selected (R_Select[p]=r).

Infeasibility in respecting constraint 1.11 and constraint 1.12 is accepted but penalized during the evolutionary process.

3.2.2 Solution Evaluation

The evaluation of a solution is obtained by the combination of the different objectives: the inter-cellular material handling cost (InterCMHC), the intra-cellular material handling cost (IntraCMHC), the inter-cellular worker movement (InterCWM), and the quality of the produced parts (Quality).

$minf = \alpha_1.InterCMHC + \alpha_2.InterCMHC + \alpha_3.InterCWM + \alpha_4.(5.P.M - Quality) + \alpha_5.Penalty$

In this study, a scalar approach is used to solve the problem, which is the weighted sum method. The principle is to combine all the objectives into one function and associate each objective with a weight α_i. Thus, the decision-maker may implement his preferences by defining the values $\{\alpha_i\}$.

The model includes some objectives to minimize and one objective to maximize. The objective to maximize is the quality of the produced parts. Thus to convert it into a minimization problem, the maximization of the quality is transformed into a minimization of the function $5.P.M - Quality$. The value $5.P.M$ represents the upper limit of the quality value that a solution may reach. This value can be achieved when all the parts need all the machines, and each part on each machine is supposed to be processed by one of the workers that do very well (having a quality value equal to 5) with the concerned part on the concerned machine.

Infeasible solutions that do not respect constraints 1.11 and 1.12 of the mathematical model are penalized using the factor "$\alpha_5 Penalty$". $Penalty$ represents the number of times the constraints 1.11 and 1.12 that control the cells' size in term of machines being violated. Thus, the penalty value is increased by one each time a cell exceeds the maximum number of machines (UM) or when it does not contain enough number of machines (LM).

3.3 The Genetic Algorithm

During the creation of the initial population (see Algorithm 3.2), feasibility with respect to constraints 1.6 - 1.10 is guaranteed. The assignment of machines and workers to cells (lines [2-7]) and selecting the part's routes

(lines [8-10]) are made randomly. However, the third part of the solution is constructed by selecting the more skilled workers to execute each operation (lines [11-15]).

In the proposed algorithm 3.1, the best individual **best*** of the current population **POP** is saved (line [4]), and the best 10% individuals of POP are copied to the new population (line [7]). After that, every two randomly selected individuals of the population are copied to the new population after being modified according to the instructions mentioned in algorithm 3.3 and algorithm 3.4. In algorithm 3.1, Crossover (line [13]), and Mutation (lines [15-22]) are integrated. They can be imitated by the behavior described in the next subsections.

A counter (no_improve_counter) is associated with the best individual in the population. Its role is to save the number of generations within best* did not enhance. After reaching a threshold called "limit", The algorithm will stop (lines [32-34]).

Algorithm 3.1 Genetic Algorithm

```
1: Initialize the GA parameters (pop_size, nbr_generations, crossover_rate, mutation_rate, limit).
2: Create initial population POP of pop_size individuals (solutions).
3: Create temp_pop of pop_size individuals.
4: Find the best solution best* in the initial population POP.
5: Initialize the counter of iterations without improvement of best* : no_improve_counter ← 0.
6: while generation ≤ nbr_generations do
7:     Copy the best 10% solutions of POP into temp_pop.
8:     while temp_pop not full do
9:         Select two random individuals ind1, ind2 from POP.
10:        Creat a copy indiv1 of ind1, and a copy indiv2 of ind2.
11:        rand ← Random (0:1)
12:        if rand < crossover_rate then
13:            Crossover(indiv1,indiv2)
14:        end if
15:        rand ← Random (0:1)
16:        if rand < mutation_rate then
17:            Mutation(indiv1)
18:        end if
19:        rand ← Random (0:1)
20:        if rand < mutation_rate then
21:            Mutation(indiv2)
22:        end if
23:        Add indiv1 and indiv2 to temp_pop.
24:    end while
25:    Find the best solution new_best in temp_pop.
26:    Update POP by temp_pop.
27:    Clear temp_pop.
28:    if f(new_best) > f(best*) then
29:        Update best* by new_best
30:        no_improve_counter ← 0
31:    else
32:        no_improve_counter ← no_improve_counter+1
33:        if no_improve_counter = limit then
34:            break;
35:        end if
36:    end if
37: end while
```

Algorithm 3.2 Create initial population

```
1: for i ← 1 to pop_size do
2:     for each m ∈ M do                          ▷ Assign machine m to a random cell
3:         indiv_i.C_Assign[m] ← Random(1:C)
4:     end for
5:     for each w ∈ W do
6:         indiv_i.C_Assign[M+w] ← Random(1:C)    ▷ Assign worker w to a random cell
7:     end for
8:     for each p ∈ P do                          ▷ Select a random route r for part p
9:         indiv_i.R_Select[p] ← Random(0:R_p)
10:    end for
11:    for each p ∈ P do
12:        for each m ∈ M do                      ▷ Assign the skillful worker w to op(p,m)
13:            indiv_i.W_Assign[p][m] ← w         ▷ with respect to 1.6 and 1.7
14:        end for
15:    end for
16: end for
```

3.3.1 Crossover

The crossover is defined as the global process that allows the solution to jump toward the best current solution. In this study, a crossover procedure adapted to GCCFP is developed. This crossover is occurred between two random individuals in the population (see algorithm 3.3). In the proposed algorithm, crossover acts with a probability called Crossover_rate on the assignment of cells (machines or workers) or in the routes selection of parts.

The crossover consists of an exchange between the two selected individuals with three crossover sites randomly generated in : (i) the cell affectation of machines (lines [3-13]), or (ii) the cell affectation of workers (lines [16-26]), (iii) the routes selection of parts (lines [29-39]) with the exchange in the workers' assignment of operations (lines [40-53]). This last action allows us to keep constraints 1.6 and 1.7 verified.

3.3.2 Mutation

In GA, the mutation procedure (see algorithm 3.4) is used to escape local optima. The mutation acts with a probability called mutation_rate randomly on the assignment of cells (machines or workers) or in the routes selection of parts or the workers' assignment of operations. It consists of changing a machine or a worker to a random cell (lines [2-8]), or changing the selected route for a part to another route randomly (lines [11-13]), the lines [14-17] consists of changing the assignment of workers to the operations of this route. This action allows us to keep constraints 1.6 and 1.7 verified. Finally, the mutation procedure can act on the workers' assignment of operations, and it consists of selecting a random operation (lines [19-20]) and a random worker w that may execute this selected operation (line [21]). The mutation is done by assigning w the concerned operation (line [22]).

3.4 Computational Results

The GA was coded in java using the integrated development environment: NetBeans IDE 8.1 (Build 201510222201), under Windows 8.1 operating system, and run on a PC Intel(R) Core(TM) i5-6200U CPU

Algorithm 3.3 Crossover(indiv1,indiv2)

```
rand ← Random (1:3)
if rand = 1 then                                   ▷ exchange in the cell affectation of machines
    generate three random positions r1, r2 and r3 between (1:M)        ▷ r1 ≤ r2 ≤ r3
    for i ← r1 to r2 do
        val ← indiv1.C_Assign[i]
        indiv1.C_Assign[i] ← indiv2.C_Assign[i]
        indiv2.C_Assign[i] ← val
    end for
    for i ← r3 to M do
        val ← indiv1.C_Assign[i]
        indiv1.C_Assign[i] ← indiv2.C_Assign[i]
        indiv2.C_Assign[i] ← val
    end for
else
    if rand = 2 then                               ▷ exchange in the cell affectation of workers
        generate three random positions r1, r2 and r3 between (1:W)    ▷ r1 ≤ r2 ≤ r3
        for i ← r1 to r2 do
            val ← indiv1.C_Assign[M+i]
            indiv1.C_Assign[M+i] ← indiv2.C_Assign[M+i]
            indiv2.C_Assign[M+i] ← val
        end for
        for i ← r3 to W do
            val ← indiv1.C_Assign[M+i]
            indiv1.C_Assign[M+i] ← indiv2.C_Assign[M+i]
            indiv2.C_Assign[M+i] ← val
        end for
    else
        if rand = 3 then
            generate three random positions r1, r2 and r3 between (1:P)    ▷ r1 ≤ r2 ≤ r3
            for i ← r1 to r2 do                    ▷ exchange in the routes selection of parts
                val ← indiv1.R_Select[i]
                indiv1.R_Select[i] ← indiv2.R_Select[i]
                indiv2.R_Select[i] ← val
            end for
            for i ← r3 to M do
                val ← indiv1.R_Select[i]
                indiv1.R_Select[i] ← indiv2.R_Select[i]
                indiv2.R_Select[i] ← val
            end for
            for i ← r1 to r2 do
                for m ← 0 to M do          ▷ exchange in the workers assignment of operations
                    val ← indiv1.W_Assign[i][m]
                    indiv1.W_Assign[i][m] ← indiv2.W_Assign[i][m]
                    indiv2.W_Assign[i][m] ← val
                end for
            end for
            for i ← r3 to P do
                for m ← 0 to M do
                    val ← indiv1.W_Assign[i][m]
                    indiv1.W_Assign[i][m] ← indiv2.W_Assign[i][m]
                    indiv2.W_Assign[i][m] ← val
                end for
            end for
        end if
    end if
end if
```

Algorithm 3.4 Mutation(indiv)

```
1: rand ← Random (1:4)
2: if rand = 1 then
3:     m ← Random(0:M)
4:     indiv.C_Assign[m] ← Random(1:C)
5: else
6:     if rand = 2 then
7:         w ← Random(0:W)
8:         indiv.C_Assign[M+w] ← Random(1:C)
9:     else
10:        if rand = 3 then
11:            p ← Random(0:P)
12:            r ← Random(0:R_p)
13:            indiv.R_Select[p] ← r
14:            for each op(p,m) ∈ r do                    ▷ op is an operation of r
15:                Select a random worker w that may execute op
16:                indiv.W_Assign[p][m] ← w
17:            end for
18:        else
19:            p ← Random(0:P)
20:            m ← Random(0:M)
21:            Select a random worker w that may execute op(p,m)
22:            indiv.W_Assign[p][m] ← w
23:        end if
24:    end if
25: end if
```

running at 2.30GHz 2.40GHz with 8 GB of RAM. In this study, we will evaluate the performance of our algorithm GA against other methods developed in [3]: B&B, SA, and DFPA.

3.4.1 Parameter Setting and Stopping Criterion

The correct choice of parameter values highly affects the efficiency of meta-heuristic algorithms. It is not always suitable to set them by referring to the previous literature. In this study, the traditional trial-and-error method is adopted. Thus, after intensive testing, the parameters are set as follows: nbr_generations=20000, pop_size=120, crossover_rate=0.8, mutation_rate=0.2, limit=1000.

3.4.2 GA vs. B&B

Ten runs of GA were conducted on each test problem. The objective value of the best found solution in these ten runs for each test problem is shown in Table 3.1. This table also presents the average time and the average objective value obtained for each instance.

The obtained results of GA are compared with those of B&B. Table 3.1 shows that GA and B&B offer the same results regarding the objective function's value for the four test instances (#1, #2, #3, and #5). However, regarding the computational time, GA takes less time to find the global optimal solution. For problem #4, the B&B reached the global optimal solution in more than 2 hours. But, the solution provided by the GA is just 1% larger. However, the GA's computational time is much less. For the remainder problem instances, LINGO

Table 3.1: Results GA vs. B&B.

In bold, the best found value of the objective function for each problem instance.

* problem instances could not be solved on our machine using LINGO software.

	The problem characteristics						LINGO software				GA		
No.	P	$\sum_{p=1}^{p}$	R_P	M	W	C	NV_{LM}	NC_{LM}	S_{LM}	T_{LM}	best S_{GA}	Avg S_{GA}	T_{GA}
1	4		8	3	3	2	12600	10437	**523**	00:00:01	**523**	523	00:00:01
2	5		10	4	3	2	36588	34063	**804**	00:00:54	**804**	804	00:00:01
3	6		11	3	4	2	32932	27995	**694**	00:09.27	**694**	697	00:00:01
4	7		12	4	3	2	51212	45418	**1294**	02.11.16	1308	1308	00:00:01
5	8		16	4	3	3	115433	131810	**1761**	>5.00.00	**1761**	1858	00:00:01
6	9		18	5	4	4	734818	827401	2734	>5.00.00	**2427**	2560	00:00:01
7	10		20	5	4	3	738511	598040	3368	>5.00.00	**3115**	3198	00:00:01
8	10		20	7	6	4	4986876	6391346	6976	>5.00.00	**3919**	4063	00:00:01
9	11		22	4	4	3	279098	275112	2120	>5.00.00	**1681**	1823	00:00:01
10	12		24	6	7	3	3084979	3417658	4742	>5.00.00	**2897**	3129	00:00:01
11	13		26	8	7	4	13139866	6914452	3725	>5.00.00	**2328**	2403	00:00:02
12	13		26	8	7	5	19686265	10303427	2929	>5.00.00	**2198**	2274	00:00:02
13	14		28	10	7	4	-	-	-	-	*3651	3814	00:00:02
14	20		40	13	15	5	-	-	-	-	*4655	4731	00:00:03
15	30		80	15	12	5	-	-	-	-	*6602	6771	00:00:05
16	35		85	18	18	6	-	-	-	-	*9086	9448	00:00:05
17	40		80	20	15	7	-	-	-	-	*58448	59338	00:01:05
18	50		148	22	15	7	-	-	-	-	*13868	14318	00:00:15

software could not reach better or equal values to those obtained by GA in less than 5 hours. Regarding the elapsed time measure, as can be seen in Table 3.1, GA outperforms B&B highly.

3.4.3 GA vs. SA

The assessment of GA against SA is shown in Table 3.2. By considering the objective function value of the best found solution, it can be seen that for the problems (#1, #2, #3, #4 and #5), GA and SA converge to almost the same value. For the last thirteen problems (#6, #7, #8, #9, #10, #11,#12, #13, #14, #15, #16, #17, and #18), the convergence values of GA are better than those of SA, Regarding the time-consuming GA takes much less time. Summarily, GA outperforms SA, especially for large-sized test problems. A third meta-heuristic is used as a reference to compare our algorithm, which is the DFPA. The discussion of the obtained results is shown in the next subsection.

3.4.4 GA vs. DFPA

The DFPA is an adaptation of the Flower Pollination Algorithm (FPA) to the discrete GCCFP [3]. The fast convergence and the simple computation of FPA make it a good choice to solve continuous and discrete problems. It has been extensively used in recent years to solve problems in many fields such as computer science, bioinformatics, operational research, the food industry, ophthalmology, engineering, etc.

In [3], an adaptation of DFPA is defined to solve the GCCFP.

The assessment of GA against DFPA is shown in Table 3.3. By considering the objective function value of the best found solution, it can be seen that for the problems (#1, #2, #3, #4, #5, #6 and #9), GA and DFPA converge almost to the same value. For problems (#7, #8, #10, #11,#12, #13, #14, #15, #16, and #18), the convergence values of DFPA are better than those of GA. And for problem #17, GA's best found solution is better than the solution of DFPA. Regarding the computational time, GA is better in time-consuming it takes much less time. Summarily, by considering the convergence of algorithms, we can see that GA performs better than SA. However, DFPA outperforms both of them.

3.4.5 The Convergence of Algorithms

The convergence curves of GA, DFPA, and SA for the eighteen problems are shown in Figure 3.1. The figure shows that GA and DFPA has a faster speed to converge. This fast convergence may be explained by their principle, which is a population-based optimization technique.

The population-based algorithms (GA, DFPA) tend to converge faster than the single-solution-based algorithm (SA) because the population-based metaheuristics deal at each algorithm iteration with a set of solutions rather than a single one. In other words, the population-based algorithm can complete the searching process with multiple initial points in a parallel approach. This technique has the advantage where it can provide the search space for the exploration in an effective way. This method is suitable for searching globally because it has the ability of global exploration and local exploitation.

Table 3.2: Results GA vs. SA.

In bold, the best found value of the objective function for each problem instance.

	The problem characteristics						SA			GA		
No.	P	$\sum_{p=1}^{P}$	R_P	M	W	C	best S_{SA}	Avg S_{SA}	T_{SA}	best S_{GA}	Avg S_{GA}	T_{GA}
1	4		8	3	3	2	**523**	523	00:01	**523**	523	00:00:01
2	5		10	4	3	2	**804**	814	00:02	**804**	804	00:00:01
3	6		11	3	4	2	**694**	697	00:02	**694**	697	00:00:01
4	7		12	4	3	2	**1294**	1302	00:02	1308	1308	00:00:01
5	8		16	4	3	3	**1761**	1805	00:03	**1761**	1858	00:00:01
6	9		18	5	4	4	2472	2508	00:04	**2427**	2560	00:00:01
7	10		20	5	4	3	3196	3263	00:04	**3115**	3198	00:00:01
8	10		20	7	6	4	4040	4078	00:07	**3919**	4063	00:00:01
9	11		22	4	4	3	1854	1888	00:05	**1681**	1823	00:00:01
10	12		24	6	7	3	2902	2942	00:08	**2897**	3129	00:00:01
11	13		26	8	7	4	2385	2410	00:09	**2328**	2403	00:00:02
12	13		26	8	7	5	2249	2334	00:08	**2198**	2274	00:00:02
13	14		28	10	7	4	3811	3871	00:11	**3651**	3814	00:00:02
14	20		40	13	15	5	4859	4924	00:33	**4655**	4731	00:00:03
15	30		80	15	12	5	6739	6809	01:26	**6602**	6771	00:00:05
16	35		85	18	18	6	9660	9690	02:29	**9086**	9448	00:00:05
17	40		80	20	15	7	62993	63484	05:36	**58448**	59338	00:01:05
18	50		148	22	15	7	13988	14357	05:34	**13868**	14318	00:00:15

Table 3.3: Results GA vs. DFPA.

In bold, the best found value of the objective function for each problem instance.

	The problem characteristics						DFPA			GA		
No.	P	$\sum_{p=1}^{P}$	R_P	M	W	C	best S_{DFPA}	Avg S_{DFPA}	T_{DFPA}	best S_{GA}	Avg S_{GA}	T_{GA}
1	4		8	3	3	2	**523**	523	00:01	**523**	523	00:01
2	5		10	4	3	2	**804**	804	00:02	**804**	804	00:01
3	6		11	3	4	2	**694**	694	00:02	**694**	697	00:01
4	7		12	4	3	2	**1294**	1294	00:03	1308	1308	00:01
5	8		16	4	3	3	**1761**	1761	00:04	**1761**	1858	00:01
6	9		18	5	4	4	**2421**	2427	00:05	2427	2560	00:01
7	10		20	5	4	3	**3053**	3054	00:08	3115	3198	00:01
8	10		20	7	6	4	**3906**	3924	00:08	3919	4063	00:01
9	11		22	4	4	3	**1681**	1681	00:05	**1681**	1823	00:01
10	12		24	6	7	3	**2840**	2855	00:09	2897	3129	00:01
11	13		26	8	7	4	**2202**	2237	00:11	2328	2403	00:02
12	13		26	8	7	5	**2122**	2133	00:10	2198	2274	00:02
13	14		28	10	7	4	**3472**	3494	00:15	3651	3814	00:02
14	20		40	13	15	5	**4598**	4653	00:39	4655	4731	00:03
15	30		80	15	12	5	**6299**	6328	01:48	6602	6771	00:05
16	35		85	18	18	6	**8973**	9021	03:30	9086	9448	00:05
17	40		80	20	15	7	60345	60523	07:24	**58448**	59338	01:05
18	50		148	22	15	7	**13526**	13553	09:43	13868	14318	00:15

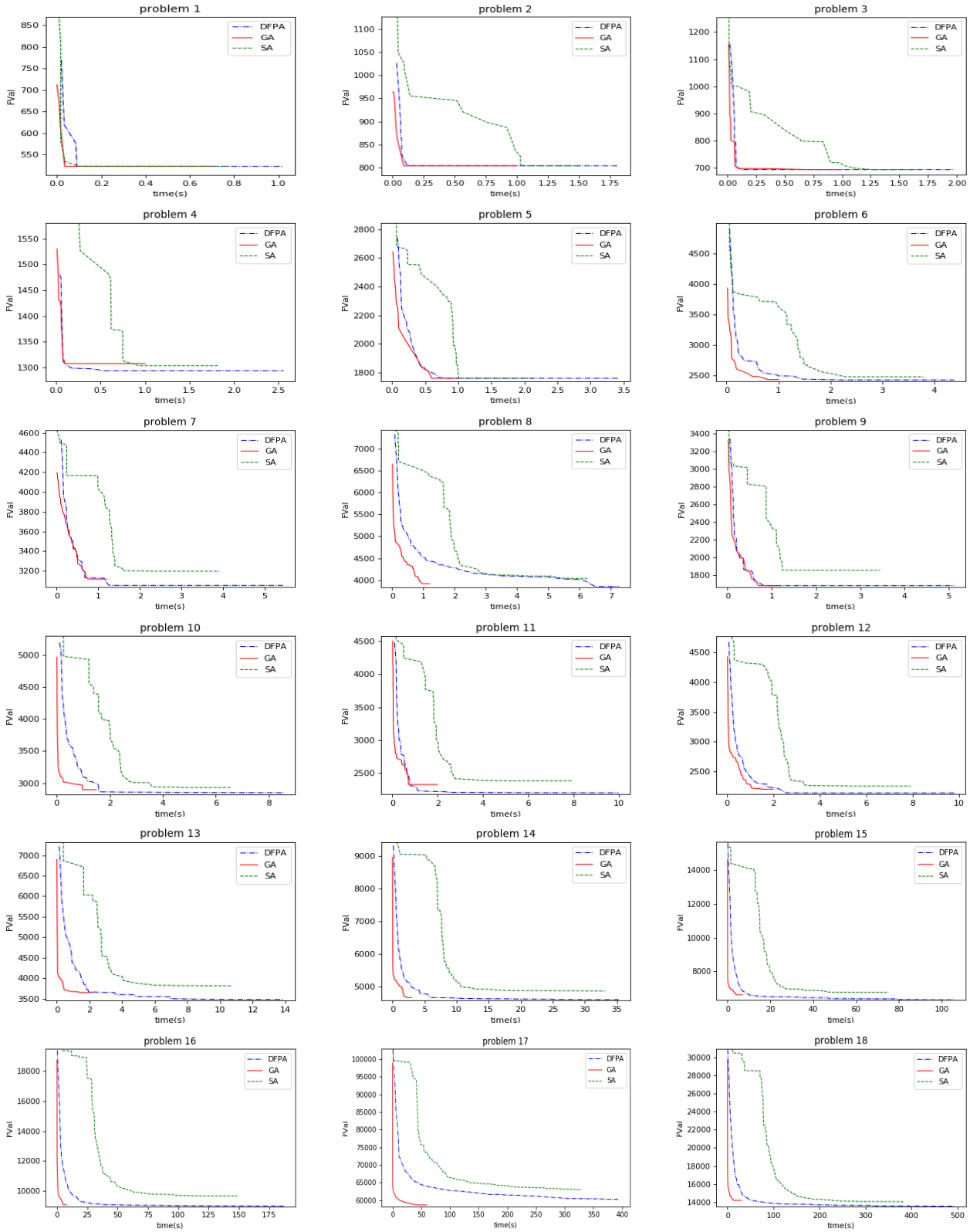


Figure 3.1: Convergence comparison of GA, DFPA, and SA

3.5 Application Interface and Instances

3.5.1 Instances

Each instance is represented in a text file and organized, as shown in Figure 3.2.

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.

3.5.2 Graphical User Interface (GUI)

The development of our application revolves around the main window, shown in Figure 3.3.

1 : The import button. By selecting this function, the window shown in Figure 3.4 is displayed.

In Figure 3.4:

2 : This window contains a button that allows you to determine the path to the file(s) that are already stored in memory (Hard Disk) and which contains the instances. By pushing the button ”Open”, the file selection is validated, as shown in Figure 3.5.

In Figure 3.5, the numbered elements are defined as follows:

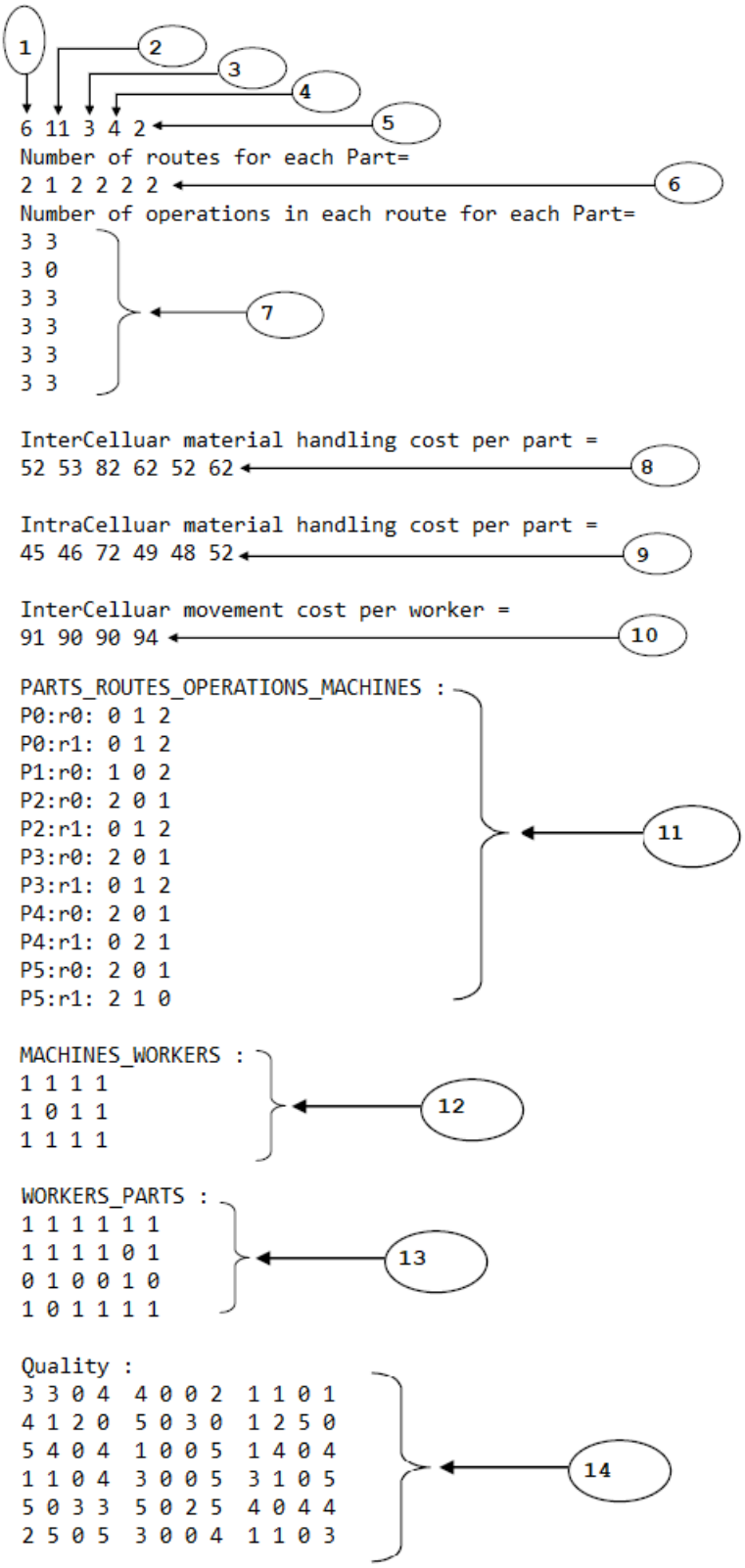


Figure 3.2: Instance representation

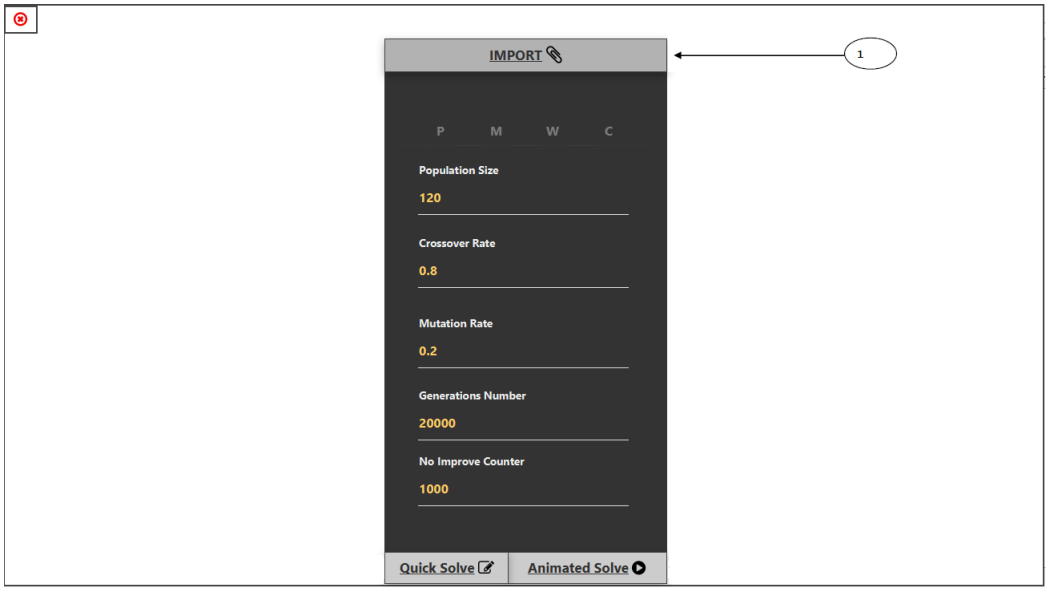


Figure 3.3: The main window

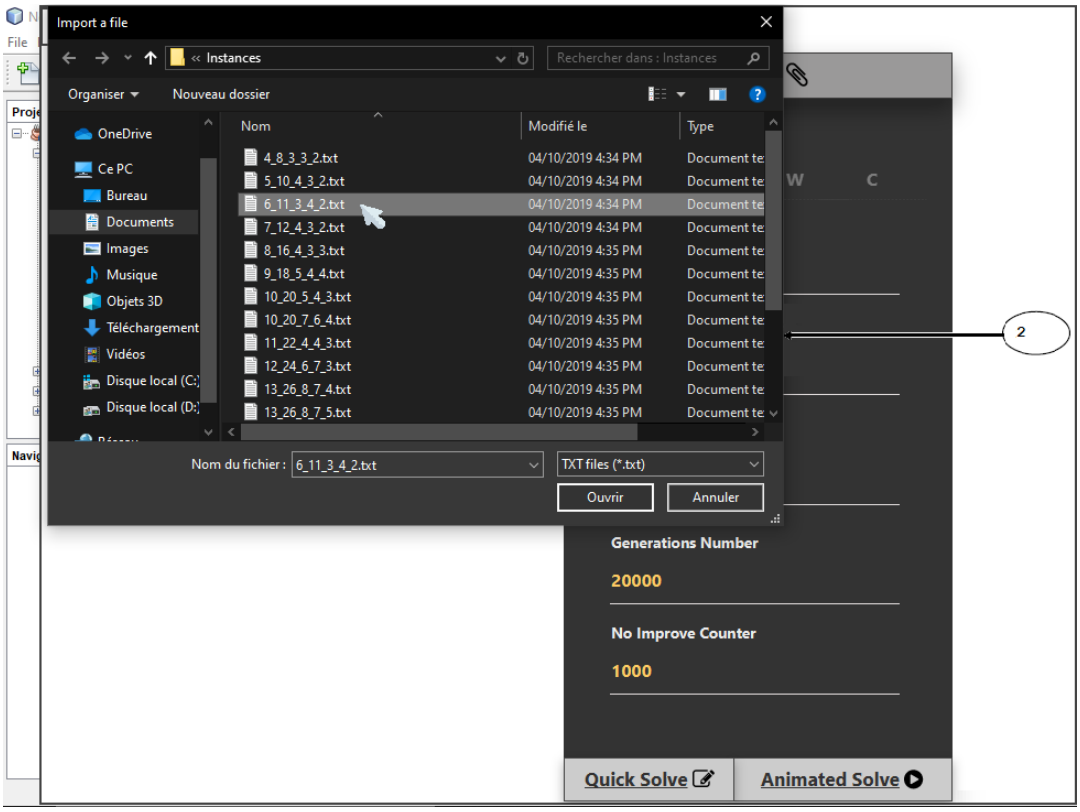


Figure 3.4: The import window

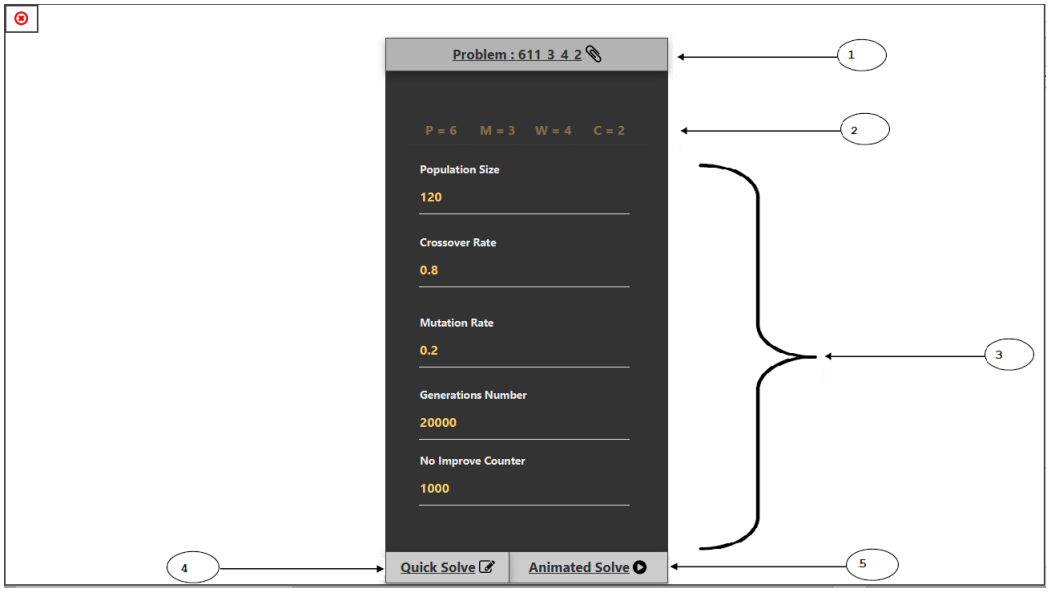


Figure 3.5: The GA's parameters insert window

1 : The selected instance file.

2 : The entries of the selected file.

3 : Input fields for entering the genetic algorithm parameters.

4 : Quick Solve button. This button launches the algorithm, and when the execution is finished, it displays the final result.

5 : Animated Solve button. This button launches the algorithm and displays the evolution of the solution during the runtime.

By pushing the "Quick Solve" button or the "Animated Solve" button, the window shown in Figure 3.6 is displayed.

In Figure 3.6, the numbered elements are defined as follows:

1 : The cells' visualization and the assignment of parts and machines and workers to these cells.

2 : The evaluation value of the final best solution.

3 : The evaluation value of the previous best solution.

4 : The evaluation value of the current best solution.

5 : Details button. By pushing this button, the window shown in Figure 3.7 is displayed.

6 : Back button. It allows going back to the main window (Figure 3.3).

The numbered elements in Figure 3.7 are defined as follows:

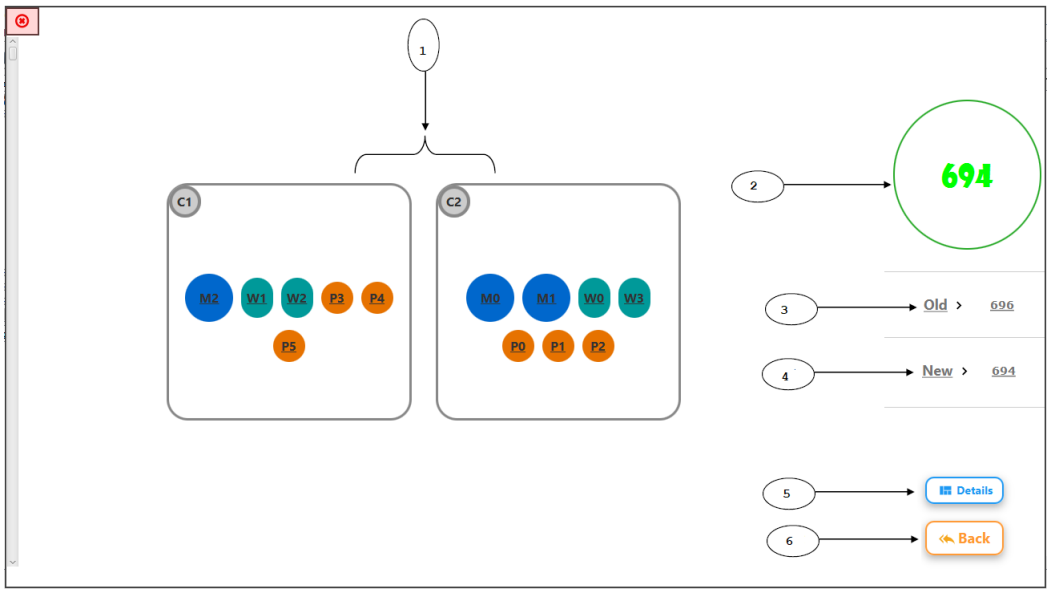


Figure 3.6: Cell visualization window

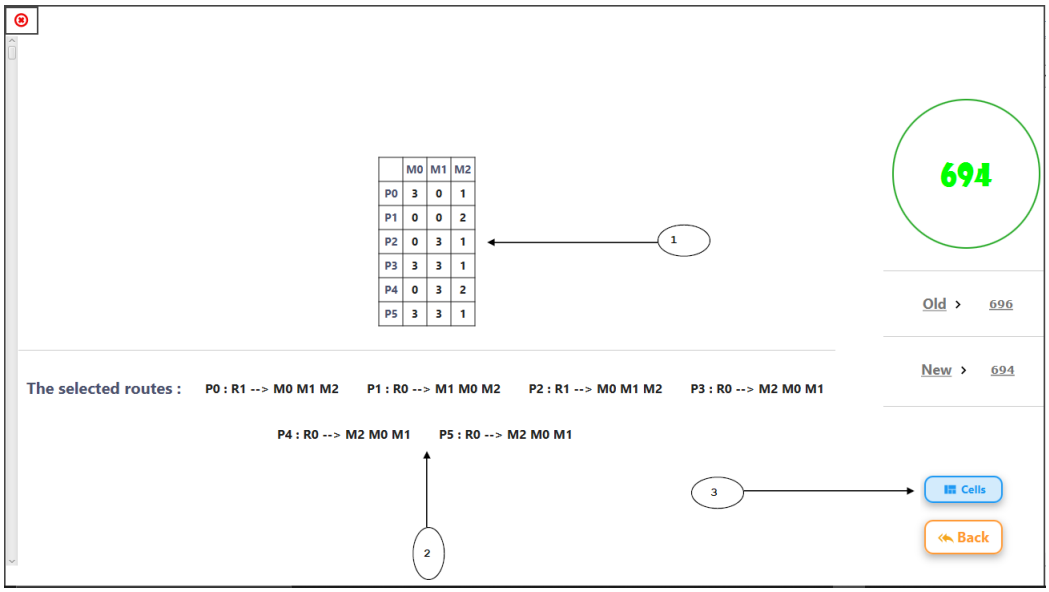


Figure 3.7: The details window

1 : The workers' assignment matrix to specify the worker in charge of executing each operation.

2 : The specification of the routes selected to process each part and the machines used in each route.

3 : Cells button allows going back to the cell visualization window (Figure 3.6).

3.6 Conclusion

In this chapter, we have shown how we applied the genetic algorithm to GCCFP. Initially, the adopted representation and evaluation of the solution are presented. Next, the solution approach containing a description of the proposed GA is detailed. After, the computational results are exhibited. Next, the application's interface and instances are shown.

Conclusion and Perspectives

In this study, we have tackled the Generalized Cubic Cell Formation Problem, which is a variant of the Cell Formation Problem. In this problem, we consider:

- workers as the third dimension besides parts and machines.

- multiple plans (routs).

Our method is based on using the genetic algorithm to solve the generalized cubic cell formation problem. To evaluate the performance of our implementation of the genetic algorithm, we compared our obtained results with the results obtained by the LINGO software by solving the problem instances using the exact Branch & Bound method. We also made a comparison with the simulated annealing algorithm and also with the discrete flower pollination algorithm.

The Comparison with Branch & Bound reveals that the GA outperforms B&B highly. For 22.22% of the instances, we obtained equal results. For 72.23%, our method offers better results. However, for 5.55%, our method gives larger results than B&B.

By comparing the objective value of the best found solution by our algorithm with those of SA, we found that our method gives equal results for 22.22% of the instances. For 72.23% of the instances, our method gives better results. However, for 5.55% of the instances, our method gives larger results than SA.
GA outperforms SA, especially for large-sized test problems.

The Comparison of GA with DFPA reveals that we obtained equal results for 27.78% of the instances. For 5.56%, GA offers better results. However, for 66.66%, DFPA gives better results than GA.

For the computational time, our method's results are better than those of the three other methods for the totality of the instances.

As it is well known in optimization, the combination of parameter' values has a great impact on the obtained results. In this study, we have used the trial and error method to fix them. Our choice of the parameter values enabled us to obtain these results, but there is a possibility that if we make more experiments, we fall on

a combination that gives better results than those exhibited in this manuscript. In the future, we have the intention to apply a statistical method called the "Taguchi method" to fix the level of each parameter.

As a perspective, we aim to solve the problem using Multi-objective methods. These laters allow us to solve this problem and provide multiple solutions instead of a single one. From these methods, we can cite the Non-dominated Sorting Genetic Algorithm (NSGA-II), Multi-Objective Vibration Damping Optimization algorithm (MOVDO), Non-dominated Ranking Genetic Algorithms (NRGA).

Bibliography

[1] Murugan, M., & Selladurai, V. (2005). Manufacturing cell design with reduction in setup time through genetic algorithm. Journal of Theoretical and Applied Inform.

[2] Vin, E. (2010). Genetic algorithm applied to generalized cell formation problems. Ph.D. thesis, Free University of Brussels, Faculty of Applied Sciences and Engineering.

[3] Bouaziz, H., Berghida, M., & Lemouari, A. (2020). Solving the generalized cubic cell formation problem using discrete flower pollination algorithm. Expert Systems with Applications, 150, 113345. DOI:10.1016/j.eswa.2020.113345.

[4] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor, MI: University of Michigan Press.

[5] Greene, T. J., & Sadowski, R. P. (1984). A review of cellular manufacturing assumptions, advantages and design techniques. Journal of Operations Management, 4(2), 85–97. DOI:10.1016/0272-6963(84)90025-1.

[6] Ballakur, A., & Steudel, H. J. (1987). A within-cell utilization based heuristic for designing cellular manufacturing systems. International Journal of Production Research, 25(5), 639–665. DOI:10.1080/00207548708919868.

[7] Nouri, H., Tang, S., Tuah, B., Ariffin, M., & Samin, R. (2013). Metaheuristic techniques on cell formation in cellular manufacturing system. Journal of Automation and Control Engineering, 1(1), 49-54.

[8] Joines, J., King, R., & Culbreth, C. (1996). A comprehensive review of production-oriented manufacturing cell formation techniques. International Journal of Flexible Automation and Intelligent Manufacturing, 3, 254-264.

[9] Utkina, I. E., Batsyn, M. V., & Batsyna, E. K. (2018). A branch-and-bound algorithm for the cell formation problem. International Journal of Production Research, 56(9), 3262–3273. DOI:10.1080/00207543.2018.1444811.

[10] KUSIAK, A. (1987). The generalized group technology concept. International Journal Of Production Research, 25(4), 561-569. DOI:10.1080/00207548708919861.

[11] MIN, H., & SHIN, D. (1993). Simultaneous formation of machine and human cells in group technology: a multiple objective approach. International Journal Of Production Research, 31(10), 2307-2318. DOI:10.1080/00207549308956859.

[12] Adil, G., Rajamani, D. & Strong, D. (1997) Assignment allocation and simulated annealing algorithms for cell formation. IIE Transactions, 29(1), 53-67. DOI:10.1080/07408179708966312.

[13] Tavakkoli-Moghaddam, R., Rahimi-Vahed, A. R., Ghodratnama, A., & Siadat, A. (2009). A simulated annealing method for solving a new mathematical model of a multi-criteria cell formation problem with capital constraints. Advances in Engineering Software, 40(4), 268-273. DOI:10.1016/j.advengsoft.2008.04.008.

[14] Neto, A. R. P. & Goncalves Filho, E. V. (2010). A simulation-based evolutionary multiobjective approach to manufacturing cell formation. Computers & Industrial Engineering, 59(1), 64-74. DOI:10.1016/j.cie.2010.02.017.

[15] Shiyas, C. R., & Pillai, V. M. (2014). A mathematical programming model for manufacturing cell formation to develop multiple configurations. Journal of Manufacturing Systems, 33(1), 149-158. DOI:10.1016/j.jmsy.2013.10.002.

[16] Hafezalkotob, A., Tehranizadeh, M. A., Rad, F., & Sayadi, M. K. (2015). An improved DPSO algorithm for cell formation problem. Journal of Industrial and Systems Engineering, 8(2), 30-53.

[17] Danilovic, M., & Ilic, O. (2019). A novel hybrid algorithm for manufacturing cell formation problem. Expert Systems With Applications, 135, 327-350. DOI:10.1016/j.eswa.2019.06.019.

[18] Mahmoodian, V., Jabbarzadeh, A., Rezazadeh, H., & Barzinpour, F. (2019). A Novel Intelligent Particle Swarm Optimization Algorithm for Solving Cell Formation Problem. Neural Computing and Applications, 31(2), 801-815.

[19] Karoum, B., & el Benani, Y. B. (2019). Discrete cuckoo search algorithm for solving the cell formation problem. International Journal of Manufacturing Research, 14, 245–264.

[20] Vin, E., De Lit, P., & Delchambre, A. (2005). A multiple-objective grouping genetic algorithm for the cell formation problem with alternative routings. Journal of intelligent manufacturing, 16(2), 189-205. DOI:10.1007/s10845-004-5888-4.

[21] Wu, T. H., Chung, S. H., & Chang, C. C. (2009). Hybrid simulated annealing algorithm with mutation operator to the cell formation problem with alternative process routings. Expert Systems with Applications, 36(2), 3652-3661. DOI:10.1016/j.eswa.2008.02.060.

[22] Deep, K., & Singh, P. K. (2015). Design of robust cellular manufacturing system for dynamic part population considering multiple processing routes using genetic algorithm. Journal of Manufacturing Systems, 35, 155–163. DOI:10.1016/j.jmsy.2014.09.008.

[23] Hazarika, M., & Laha, D. (2018). Genetic Algorithm approach for Machine Cell Formation with Alternative Routings. Materials Today: Proceedings, 5(1), 1766–1775. DOI:10.1016/j.matpr.2017.11.274.

[24] Tavakkoli-Moghaddam, R., Safaei, N., & Sassani, F. (2008). A new solution for a dynamic cell formation problem with alternative routing and machine costs using simulated annealing. Journal Of The Operational Research Society, 59(4), 443-454. DOI:10.1057/palgrave.jors.2602436.

[25] Jouzdani, J., Barzinpour, F., Shafia, M. A., & Fathian, M. (2014). Applying Simulated Annealing To A Generalized Cell Formation Problem Considering Alternative Routings And Machine Reliability. Asia-Pacific Journal of Operational Research, 31(04), 1450021. DOI:10.1142/s0217595914500213.

[26] Ameli, M. S. J., Arkat, J., & Barzinpour, F. (2008). Modelling the effects of machine breakdowns in the generalized cell formation problem. The International Journal of Advanced Manufacturing Technology, 39(7), 838-850. DOI:10.1007/s00170-007-1269-4.

[27] Chung, S. H., Wu, T. H., & Chang, C. C. (2011). An efficient tabu search algorithm to the cell formation problem with alternative routings and machine reliability considerations. Computers & Industrial Engineering, 60(1), 7–15. DOI:10.1016/j.cie.2010.08.016.

[28] Karoum, B., & Elbenani, Y. B. (2017). A clonal selection algorithm for the generalized cell formation problem considering machine reliability and alternative routings. Production Engineering, 11(4), 545-556. DOI:10.1007/s11740-017-0751-6.

[29] Li, M. L. (2003). The algorithm for integrating all incidence matrices in multi-dimensional group technology. International Journal of Production Economics, 86(2), 121-131. DOI:10.1016/s0925-5273(03)00010-0.

[30] Mahdavi, I., Aalaei, A., Paydar, M. M., & Solimanpur, M. (2010). Designing a mathematical model for dynamic cellular manufacturing systems considering production planning and worker assignment. Computers & Mathematics with Applications, 60(4), 1014–1025. DOI:10.1016/j.camwa.2010.03.044.

[31] Nikoofarid, E., & Aalaei, A. (2012). Production planning and worker assignment in a dynamic virtual cellular manufacturing system. International Journal of Management Science and Engineering Management, 7(2), 89–95. DOI:10.1080/17509653.2012.10671211.

[32] Mahdavi, I., Aalaei, A., Paydar, M. M., & Solimanpur, M. (2012). A new mathematical model for integrating all incidence matrices in multi-dimensional cellular manufacturing system. Journal of Manufacturing Systems, 31(2), 214–223. DOI:10.1016/j.jmsy.2011.07.007.

[33] Aalaei, A., & Shavazipour, B. (2013). The Tchebycheff Norm for Ranking DMUs in Cellular Manufacturing Systems with Assignment Worker. International Journal of Applied, 3(3), 41-57.

[34] Bootaki, B., Mahdavi, I., & Paydar, M. M. (2014). A hybrid GA-AUGMECON method to solve a cubic cell formation problem considering different worker skills. Computers & Industrial Engineering, 75, 31-40. DOI:10.1016/j.cie.2014.05.022.

[35] Bootaki, B., Mahdavi, I., & Paydar, M. M.(2015). New bi-objective robust design-based utilisation towards dynamic cell formation problem with fuzzy random demands. International Journal of Computer Integrated Manufacturing, 28(6), 577-592. DOI:10.1080/0951192x.2014.880949.

[36] Buruk Sahin, Y., & Alpay, S. (2016). A metaheuristic approach for a cubic cell formation problem. Expert Systems With Applications, 65, 40-51. DOI:10.1016/j.eswa.2016.08.034.

[37] Feng, H., Da, W., Xi, L., Pan, E., & Xia, T. (2017). Solving the integrated cell formation and worker assignment problem using particle swarm optimization and linear programming. Computers & Industrial Engineering, 110, 126-137. DOI: 10.1016/j.cie.2017.05.038.

[38] Bagheri, F., Safaei, A., Kermanshahi, M., & Paydar, M. (2019). Robust Design of Dynamic Cell Formation Problem Considering the Workers Interest. International Journal of Engineering, 32(12), 1790-1797. DOI: 10.5829/ije.2019.32.12c.12.

[39] Blum, C., & Roli, A. (2008). Hybrid metaheuristics: An introduction. Studies in Computational Intelligence, 114, 1-30. DOI:10.1007/978-3-540-78295-7.

[40] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. Computers & Operations Research, 13(5), 533–549. DOI:10.1016/0305-0548(86)90048-1.

[41] Osman, I. H., & Laporte, G. (1996). Metaheuristics: A bibliography. Annals of Operations Research, 63(5), 511–623. DOI:10.1007/bf02125421.

[42] Bäck, T., Fogel, D., & Michalewicz, Z. (1997). Handbook of Evolutionary Computation. New York: Institute of Physics Publishing Ltd, Bristol and Oxford University Press.

[43] Goldberg, D. (1989). Genetic algorithms in search, optimization and machine learning. Boston: Addison Wesley.

[44] Gen, M., & Cheng, R. (2000). Genetic Algorithms and Engineering Optimizations. New York: Wiley.

[45] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on Evolutionary Computation, 6(2), 182-197. DOI:10.1109/4235.996017.

[46] Mello-Román, J. D., & Hernandez, A. (2020). KPLS optimization approach using genetic algorithms. Procedia Computer Science, 170, 1153–1160. DOI: 10.1016/j.procs.2020.03.051.

[47] Čuboňová, N., Dodok, T., & Ságová, Z. (2019). Optimisation of the machining process using genetic algorithm. Scientific Journal of Silesian University of Technology. Series Transport, 104, 15-25. DOI: 10.20858/sjsutst.2019.104.2.

[48] Xu, Z., & Brill, M. H. (2019). Reflective color reduction using genetic algorithm optimization. Color Research and Application, 44, 526– 530. DOI: 10.1002/col.22361.

[49] Hemati, A., & Shooshtari, A. (2019). Suspension damping optimization using genetic algorithms. International Journal of Automotive Engineering and Technologies, 8 (4), 178-185. DOI: 10.18245/ijaet.531810.

[50] Coley, D. (1999). An introduction to genetic algorithms for scientists and engineers. Singapore: World Scientific Publishing.

[51] Sharma, P., Wadhwa, A., & Komal, K. (2014). Analysis of Selection Schemes for Solving an Optimization Problem in Genetic Algorithm. International Journal Of Computer Applications, 93(11), 1-3. DOI:10.5120/16256-5714.

[52] Rahmat-Samii, Y. & Michielssen, E. (1999). Electromagnetic optimization by genetic algorithms. New York: John Wiley & Sons.

[53] Haupt, R., & Haupt, S. (2004). Practical genetic algorithms. Hoboken, N.J.: John Wiley.

[54] Recioui, A. (2006). Use of Genetic Algorithms in Antennas: Application to Yagi-Uda antenna and antenna arrays. Thesis. Boumerdes: M'hamed Bouguerra University – Faculty of Engineering.

[55] Sivanandam, S., & Deepa, S. (2008). Introduction to genetic algorithms. New York: Springer.

[56] Kumar, A. (2013). Encoding schemes in genetic algorithm. International Journal of Advanced Research in IT and Engineering, 2 (3), 1-7.

[57] Yang, X. (2014). Nature-Inspired Optimization Algorithms. London: Elsevier.

[58] Linden, D. (1997). Automated design and optimization of wire antennas using genetic algorithms. Ph.D. Thesis. MIT.

[59] Gen, M., Cheng, R., & Lin, L. (2008). Network models and optimization: Multiobjective genetic algorithms approach. London: Springer.

[60] Burke, E., & Kendall, G. (2014). Search methodologies: Introductory tutorials in optimization and decision support techniques. New York: Springer.

[61] Lughofer, E., & Sayed-Mouchaweh, M. (2019). Predictive Maintenance in Dynamic Systems: Advanced Methods, Decision Support Tools and Real-World Applications. Springer.