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

```
for i ← 1 to pop_size do
    for each m ∈ M do                                   ▷ Assign machine m to a random cell
        indiv_i.C_Assign[m] ← Random(1:C)
    end for
    for each w ∈ W do
        indiv_i.C_Assign[M+w] ← Random(1:C)             ▷ Assign worker w to a random cell
    end for
    for each p ∈ P do                                   ▷ Select a random route r for part p
        indiv_i.R_Select[p] ← Random(0:R_p)
    end for
    for each p ∈ P do
        for each m ∈ M do                               ▷ Assign the skillful worker w to op(p,m)
            indiv_i.W_Assign[p][m] ← w                  ▷ with respect to 1.6 and 1.7
        end for
    end for
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)

```
rand ← Random (1:4)
if rand = 1 then
    m ← Random(0:M)
    indiv.C_Assign[m] ← Random(1:C)
else
    if rand = 2 then
        w ← Random(0:W)
        indiv.C_Assign[M+w] ← Random(1:C)
    else
        if rand = 3 then
            p ← Random(0:P)
            r ← Random(0:R_p)
            indiv.R_Select[p] ← r
            for each op(p,m) ∈ r do                          ▷ op is an operation of r
                Select a random worker w that may execute op
                indiv.W_Assign[p][m] ← w
            end for
        else
            p ← Random(0:P)
            m ← Random(0:M)
            Select a random worker w that may execute op(p,m)
            indiv.W_Assign[p][m] ← w
        end if
    end if
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

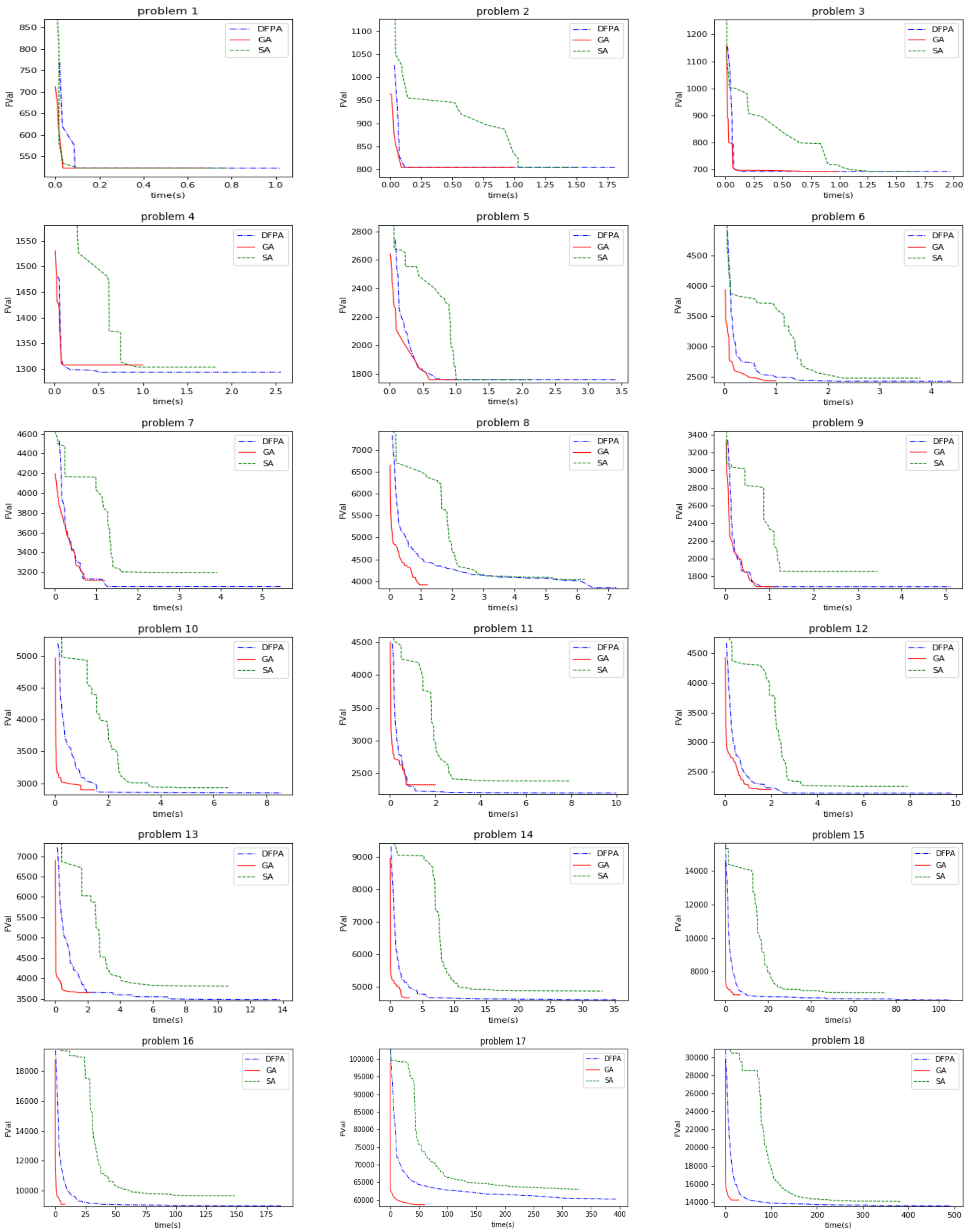

Figure 3.1: Convergence comparison of GA, DFPA, and SA

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:

```
1 2 3 4
6 11 3 4 2                                        5
Number of routes for each Part=
2 1 2 2 2 2                                       6
Number of operations in each route for each Part=
3 3
3 0
3 3                                               7
3 3
3 3
3 3

InterCelluar material handling cost per part =
52 53 82 62 52 62                                 8

IntraCelluar material handling cost per part =
45 46 72 49 48 52                                 9

InterCelluar movement cost per worker =
91 90 90 94                                       10

PARTS_ROUTES_OPERATIONS_MACHINES :
P0:r0: 0 1 2
P0:r1: 0 1 2
P1:r0: 1 0 2
P2:r0: 2 0 1
P2:r1: 0 1 2                                      11
P3:r0: 2 0 1
P3:r1: 0 1 2
P4:r0: 2 0 1
P4:r1: 0 2 1
P5:r0: 2 0 1
P5:r1: 2 1 0

MACHINES_WORKERS :
1 1 1 1
1 0 1 1                                           12
1 1 1 1

WORKERS_PARTS :
1 1 1 1 1 1
1 1 1 1 0 1                                       13
0 1 0 0 1 0
1 0 1 1 1 1

Quality :
3 3 0 4  4 0 0 2  1 1 0 1
4 1 2 0  5 0 3 0  1 2 5 0
5 4 0 4  1 0 0 5  1 4 0 4                         14
1 1 0 4  3 0 0 5  3 1 0 5
5 0 3 3  5 0 2 5  4 0 4 4
2 5 0 5  3 0 0 4  1 1 0 3
```

Figure 3.2: Instance representation

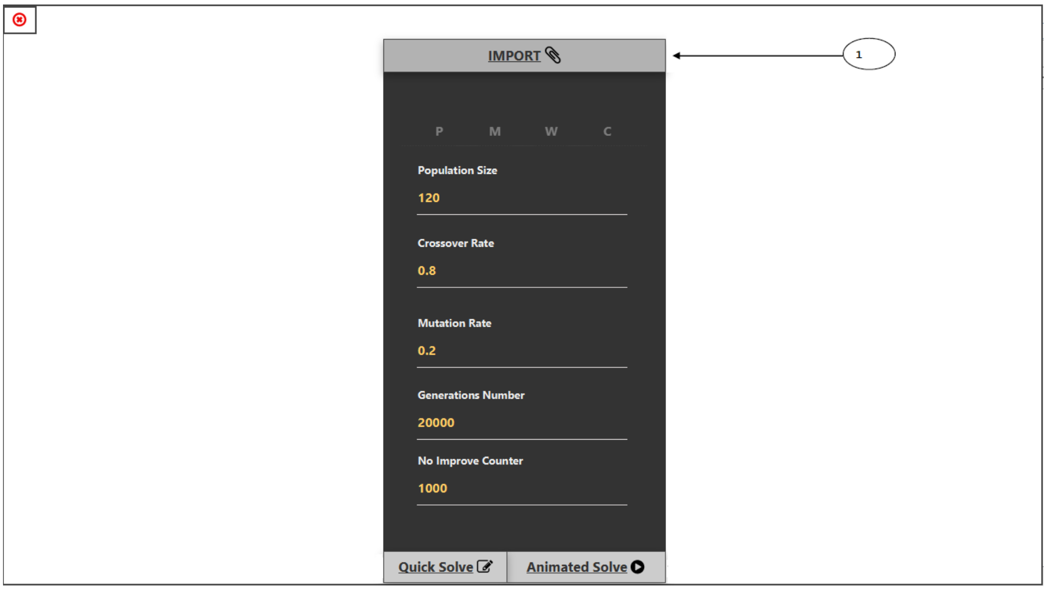

Figure 3.3: The main window

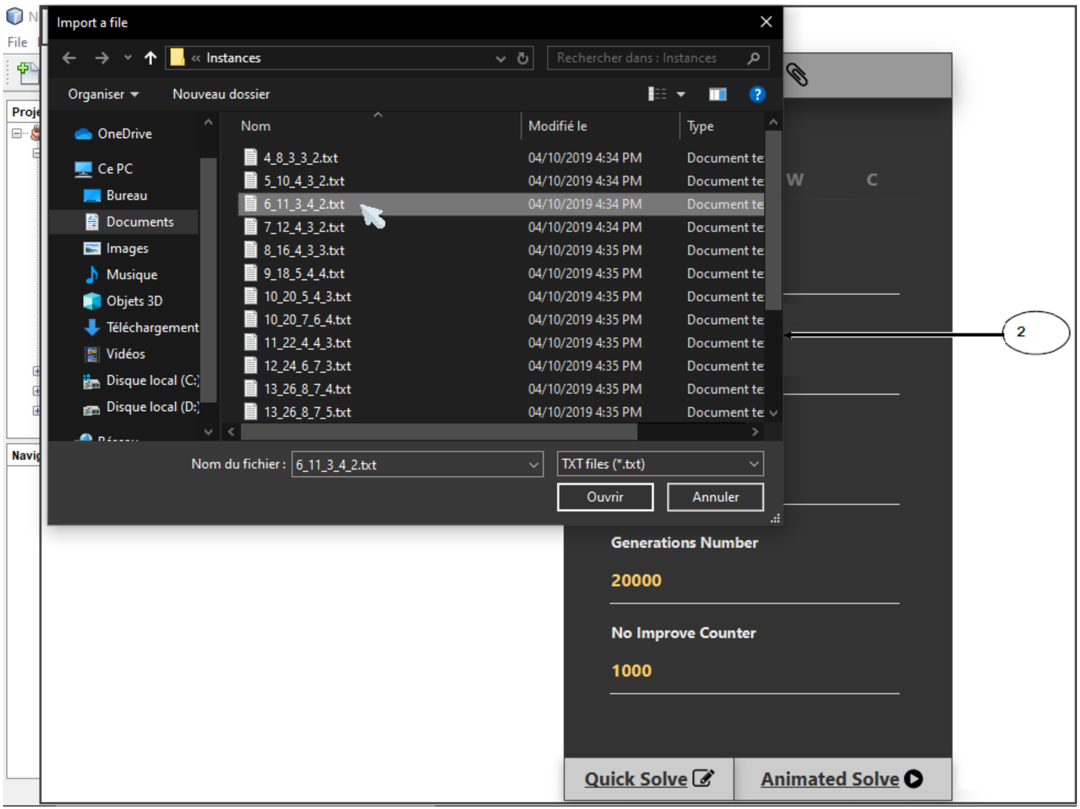

Figure 3.4: The import window

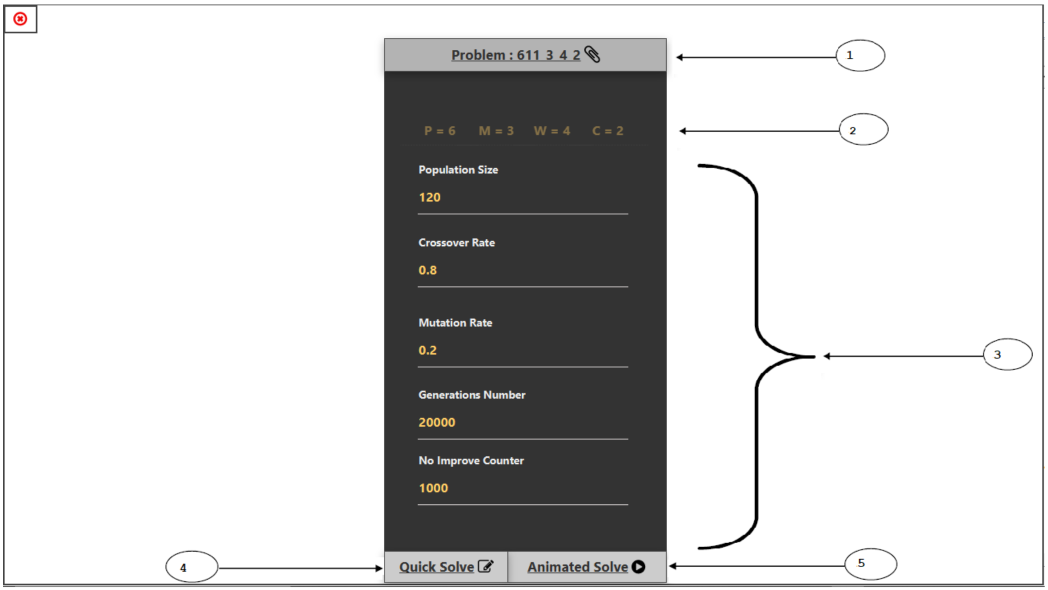

Figure 3.5: The GA's parameters insert window

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:

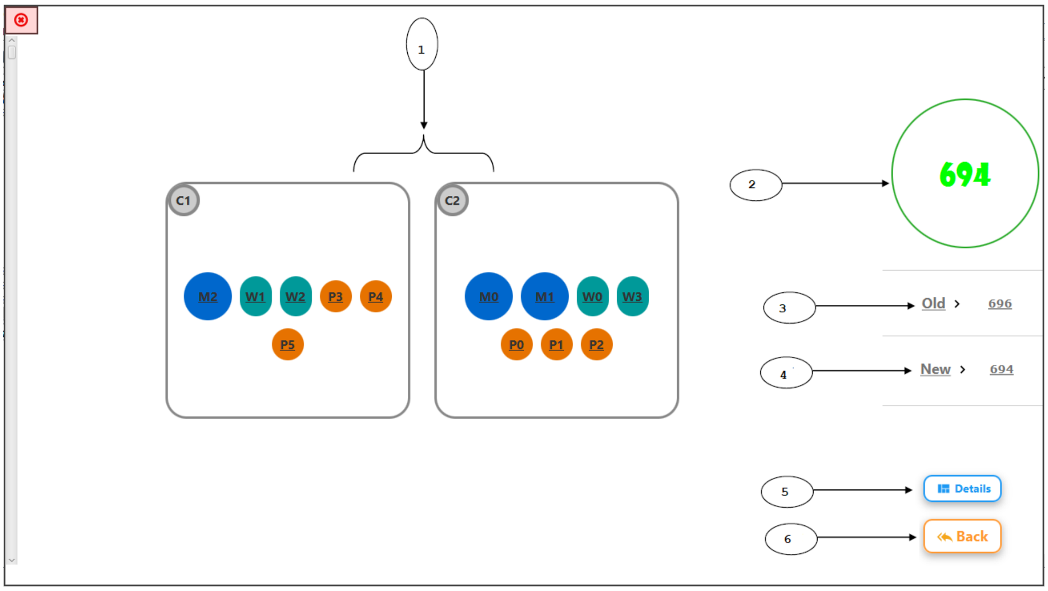

Figure 3.6: Cell visualization window

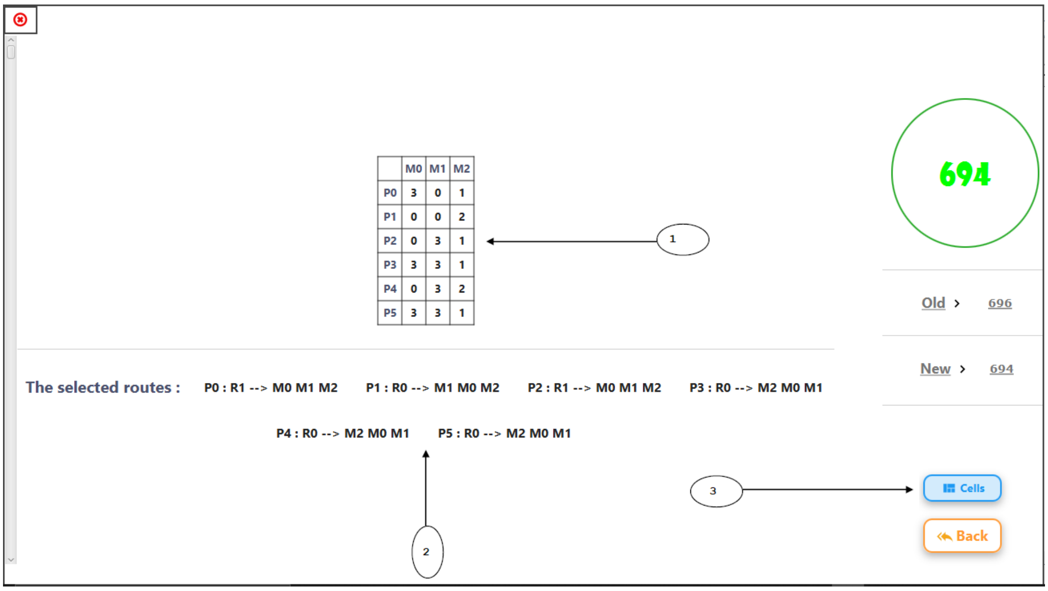

Figure 3.7: The details window

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).