To appear in Expert Systems with Applications

A numerical comparison between simulated annealing and evolutionary approaches to the cell formation problem

Andrés Pailla*, Athila R Trindade, Victor Parada Daza*, Luiz Satoru Ochi****

(*) Universidade de Santiago de Chile - Chile

(**) Universidade Federal Fluminense - Brazil

Abstract

The Cell Formation Problem is a crucial component of a cell production design in a manufacturing system. This problem consists of a set of product parts to be manufactured in a group of machines. The objective is to build manufacturing clusters by associating part families with machine cells, with the aim of minimizing the inter-cellular movements of parts by grouping efficacy measures. We present two approaches to solve the cell formation problem. First, we present an evolutionary algorithm that improves the efficiency of the standard genetic algorithm by considering cooperation with a local search around some of the solutions it visits. Second, we present an approach based on simulated annealing that uses the same representation scheme of a feasible solution. To evaluate the performance of both algorithms, we used a known set of CFP instances. We compared the results of both algorithms with the results of five other algorithms from the literature. In 8 out of 36 instances we considered, the evolutionary method outperformed the previous results of other evolutionary algorithms, and in 26 instances it found the same best solutions. On the other hand, simulated annealing not only found the best previously known solutions, but it also found better solutions than existing ones for various problems.

Keywords: heuristics, evolutionary algorithms, simulated annealing, cell formation problem, clustering.

1 - Introduction

Production management is an important factor in the success of an industry. Its goal is to organize the production environment in order to save time and production costs without loss of quality. Industries that focus on small variety and high production volume usually organize the production environment into production lines, and each production line consists of various kinds of machines devoted exclusively to the production of a single product.

In the case of an industry with high variety and medium production volume, the machines need not be dedicated only to the production of a single product. Therefore, one kind of production organization comprises the development of clusters of machines with identical functions (for example, groups of lathes or of milling machines), forming specialized departments. In this way, any part of a given product that needs manufacturing operations by more than one kind of machine will need to go through all the groups that contain the kinds of machines required for its complete manufacture. As a result, this kind of organization favors the gathering of a large volume of material that the various groups of machines will handle, increasing the time needed for production and making it more difficult to control and maintain the production system. An organizational model to represent this scheme of work, called cellular manufacture, has been used over the last two decades. In this scheme, machines with different functions are grouped in one cell that is devoted to the production of a family of parts that are very similar to each other.

An example of the formation of manufacturing cells is shown in Figures 1*a* and 1*b*. Consider a production flow consisting of 6 parts, 4 machines and 14 activities (incidence matrix positions with a value of 1). Figures 1*a* represents the input matrix, and 1*b* represents a possible solution matrix with the formation of two cells (families) identified by the shaded area. The manufacturing cells consist of machines (M2, M3) with family of parts (1, 3, 6) and (M1, M4) with family of parts (2, 4, 5).

	P1	P2	P3	P4	P5	P6
M1		1			1	
M2	1	1	1		1	1
M3	1		1			1
M4		1		1	1	1

1a)

	P1	P3	P6	P2	P4	P5
M2	1	1	1	1		1
M3	1	1	1			
M1				1		1
M4			1	1	1	1

1b)

Figure 1: Cluster matrix.

The clusters of machine cells and parts families must have a high degree of independence between them and must be characterized by the presence of few exceptional elements which correspond to those elements with a value of 1 that do not belong to any cluster. Such elements represent the presence of parts that need more than one machine cell for their complete manufacture, and therefore they generate the need for movement of material between different manufacturing cells. It is also desirable to have only a few voids within the clusters, because this involves differences in the work loads of the machines belonging to the same production cluster. In other words, it is desired that each cluster should be as dense as possible. In the example of Figure 1*b* a void is identified in the second cluster given by the (M1, P4) component.

In practice it is uncommon to find production systems that may become manufacturing systems with completely independent clusters (obviously excluding the particular case in which the system is composed of a single cluster). In view of this, many practitioners have tried to solve the independence of the clusters by doubling the number of machines responsible for the appearance of exceptional elements, or by outsourcing the corresponding activities. However, a decision in this sense must be supported by a technical and economic feasibility analysis, which in some situations puts in evidence the impossibility of choosing such alternative. The solution must then consider the forced optimization of the available resources.

The problem of defining an optimum set of *clusters* that includes parts and machines is known as the Cell Formation Problem (CFP). Considering p clusters, the number of possible ways of grouping n parts and m machines in the p clusters, which in turn corresponds to the number of feasible solutions of the CFP, is given by (1) (Wang,

1999). Considering that this number becomes computationally infeasible, even for small numbers of machines and parts, in recent years research has been more often directed toward heuristic search models.

$$\left(\left(\sum_{i=1}^{p}(-1)^{p-i}i^{n}\Big/\left[i!(p-i)!\right]\right)*\left(\sum_{i=1}^{p}(-1)^{p-i}i^{m}\Big/\left[i!(p-i)!\right]\right)\right) \quad (1)$$

The problem has been studied with several variations, derived from considering different functions to be optimized and different design approaches (Yin & Yasuda, 2006). Some researchers have studied the CFP jointly with the sequencing problem, while others have considered it jointly with the problem of defining machine layout. Literature reviews about CFP can be found in the papers of Selim *et al*. (1998), and Defersha (2006).

In previous work, researchers have applied mathematical programming and meta-heuristics to the CFP and its variants. They have used evolutionary algorithms: a kind of metaheuristic that emulates the process of natural evolution. Starting from a population of solutions where each solution is evaluated, the crossover, mutation and selection operators applied to the current population give rise to a new population. When each solution of the problem is represented as a binary string, the resulting algorithm is known as genetic (GA), (Goldberg, 1989; Silver, 2004). Joines *et al*. (1996) developed a GA to solve the CFP with an integer programming formulation with a multicriteria objective function, and previously knowing an upper bound on the number of clusters. Mak *et al*. (2000) also proposed an adaptive genetic algorithm in which the rate of using mutation and crossover changes dynamically with the efficiency of such operators during the execution of the algorithm. Recently, researchers have argued that it is possible to make genetic algorithms for solving optimization problems perform better by considering a local improvement process over each new individual generated in a population. In this respect, Gonçalves and Resende (2004)) used a local search algorithm coupled with the GA to improve the CFP's solution in several instances. James *et al*. (2007) have explored a similar approach using the same local search algorithm. The local search used in both cases is a two-stage iterative process that first analyzes the assignment of one part to a group of machines, then analyzes the assignment of a machine to a family of parts. To identify the best assignment, the algorithm evaluates the grouping efficacy of each assignment. The results of James et al. (2007) are as

good or better than those obtained with a Grouping Genetic Algorithm (GGA), studied by Brown and Sumichrast (2001). Muruganandam *et al*. (2005) have also considered the idea of a local search, has by considering tabu search for the local optimization stage (Glover and Laguna, 1998). In this case, the problem under consideration is a CFP variant that includes machine sequencing and combines an objective function that minimizes the total number of moves along with the cell load variation. Its results in four test instances also show that the algorithm that combines GA with TS performs better than GA and TS in its standard version.

Simulated Annealing (SA) is another of the metaheuristics that have been used extensively to solve combinatorial optimization problems. By simulating the phenomenon that takes place in the cooling of pure substances from the liquid to the solid state, SA improves a solution to an optimization problem gradually until it finds the best solution in the search space (Kirkpatrick et al., 1983; Silver, 2004). In each iteration, the algorithm accepts a randomly generated solution in the neighborhood of the current solution directly if it is better or probabilistically if it is worse. Wu et al. (2008) have studied CFP by SA, but in a similar way to the evolutionary approaches, they proposed a local optimization stage around the current solution, similar to the generation of neighboring solutions used by the tabu search method. They use two kinds of moves; single-move and exchange-move, both of which are operations that exhaustively revise the neighborhood of the current solution in order to obtain the best solution. They compare their results with a genetic algorithm that solves a traveling salesman problem directly associated with the CFP (Cheng et al., 1998), improving the grouping efficacy.

In this paper, we present a new approach for both search strategies: evolution coupled with a local search, and simulated annealing. In the first case, we study the impact both of the use of local search, and of a procedure to generate an initial population with a constructive heuristic algorithm. In the second case, considering the same representation scheme, we propose a simulated annealing approach. Section 2 describes both algorithms, and Section 3 presents a computational experiment showing that our algorithm is efficient. Finally, Section 4 presents the main conclusions of the study.

2. – Proposed methods to solve the CFP

2.1 An evolutionary algorithm

In spite of their proven efficiency in various optimization problems, the performance of genetic algorithms (GAs) has not been satisfactory for many problems that already have efficient heuristic algorithms for finding an approximate solution. Because of this, in many problems other than CFP, several researchers have shown the benefits of the incorporation of local search techniques into these kinds of algorithms. (Drummond et. al., 2001). Consequently, researchers have generated specific algorithms for these problems, with the basic characteristics of the GAs plus additional mechanisms, making them more specialized for some applications. In this paper, we propose an evolutionary algorithm (EA) for the CFP that has the following characteristics in relation to a standard GA like those proposed in the literature for the CFP:

a. The initial population is not generated purely randomly. Some of the individuals are generated by a heuristic procedure whose purpose is to generate, with little computer time, individuals that are different and have a lower average level of the aptitude function, better than those that are obtained by generating the individuals in a purely random manner.
b. Instead of the traditional crossover operator, the algorithm uses a new crossover mechanism to generate individuals from parents. It does not use the mutation operator.
c. We propose an efficient local search method to be used on part of the population in each generation, doing an intensive search in the neighborhood of the best solutions found for the EA, with the purpose of achieving a larger number of optimum local solutions in an optimization problem.

To represent the CFP, the structure of an individual, the fitness function, the generation of the initial population, the strategy for selecting individuals, the crossover mechanism, and the local search procedure must be defined adequately.

Structure of the individual. Considering the different approaches found in the literature for coding a CFP solution as a string of characters, we have used the proposal of Joines *et al*., (1996). Each individual is composed of ($m + n$) genes, where m is the number of machines and n is the number of parts of the problem. In this way, an individual is represented as $s = (x_1,...,x_m \mid y_1,...,y_n)$. Every gene x_i indicates the cluster to which the i^{th} machine is assigned, and every gene y_i indicates the cluster to which the i^{th} part is assigned.

Fitness Function. Let e be the number of operations to be performed (total number of 1s in the input matrix), e_0 the number of exceptional elements, and e_v the number of empty spaces (voids) in the clusters. Then, we define the evaluation function f to be maximized as in Joines *et al.* (1996) and Gonçalves and Resende (2004). This function (2), also known as *grouping efficacy*, is useful for maximizing the density within each cluster and minimizing the number of voids, decreasing dependence between the clusters.

$$f = (e - e_0)/(e + e_v) \tag{2}$$

Generation of the initial population. The initial population in our algorithm comprises some individuals generated randomly, and others by a constructive heuristic (CH). This procedure is based on the idea of Wang (1999) for solving the CFP, but here it is adapted so that different individuals can be generated in each execution.

a) Random construction: A *string solution* is created, assigning to each element of the *string* a random *cluster* number chosen between the minimum and the maximum number of *clusters* allowed. Then the algorithm verifies that the individual contains neither unit *clusters* nor empty rows or columns. Then it calculates the *fitness* and adds the individual to the population.
b) Heuristic construction: Let K be an input parameter to the CH, chosen randomly between the minimum and the maximum number of allowed clusters. We also define the "similarity" between a machine i and a machine j as the number of parts in common attended by both machines. Then, the algorithm picks one of the three pairs of less similar machines randomly to generate the first initial cluster. Each machine of this pair constitutes one of the first clusters. The algorithm generates the following initial unit clusters by choosing the machine that is less similar to the

already formed clusters (to the first two, and so on successively) until it generates the *K* initial clusters:

$$K \in [2, \lfloor number\ of\ machines/2 \rfloor]$$

The machines that have not been grouped yet (assigned to a cluster) are assigned randomly to one of the already-formed clusters with which they have the greatest similarity, respecting the maximum number of machines per cluster.

Each part has a proximity relation with the formed clusters; the greater the number of machines processing a part in the same cluster, the greater the proximity. To assign each part, we determine the two clusters with the greatest proximity with that part, and then we assign the part randomly to one of the two clusters.

The CH can generate different individuals that represent good feasible solutions, in which machines and parts are clustered according to a similarity criterion. This is in contrast with individuals generated randomly, where there is no criterion for associating machines and parts to the clusters.

Selection of individuals. To select probabilistically the best individuals of a population, we use the probability function given by Joines *et al*. (1996). The individuals are arranged according to their fitness, and then we assign a selection probability, greater for well evaluated individuals and smaller for less apt ones.

Crossover. We replaced the operator that genetic algorithms typically use with a more generic crossover. Instead of generating an offspring from two parents, the algorithm assigns randomly to each gene of the offspring corresponding to a machine x_i, the cluster number corresponding to one of the *C* machine cells in which the presence of this machine is more frequent. This mechanism generates an offspring based on the genetic information of the generating C *strings*, where the number of parents N_p ($N_p > 2$) is an input parameter of the method. The composition of an offspring *string* considers two steps:

a) Identifying a cluster for each machine. The cluster to which machine m_i belongs is determined by the frequency with which this machine is present in the generating *strings*. For example, machine m_3 (Fig. 2) belongs to clusters (1, 4, 1, 3, 1, 4, 1): that

is, most frequently to cluster 1 and next-most-frequently to cluster 4. One of the two highest frequency clusters is chosen randomly ($c = 2$).

b) Identifying a cluster for each part. The cluster to which part p_j of the offspring belongs is determined from the machines that are most frequently associated with part p_j in the generating individuals. For example, part p_1 in Figure 3 is associated with *clusters* (2, 2, 1, 2, 4, 2, 3) in generators C1 to C7. Individual c_1 shares the *cluster* with m_1 y m_5, while c_2 shares it with m_4, and so forth. Therefore, p_1 shares the *cluster* most frequently with m_4, m_5 and m_7. When one of these three machines is chosen randomly, for example m_4, the algorithm sees that it is already associated with *cluster* 3 in the new offspring; so, for p_1 , $y_1 = 3$.

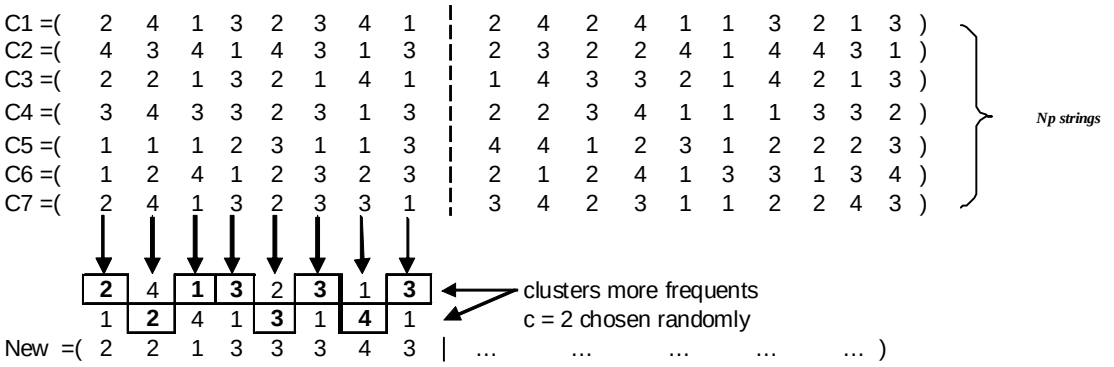

Figure 2. Assigning cluster values to the machines of a new individual.

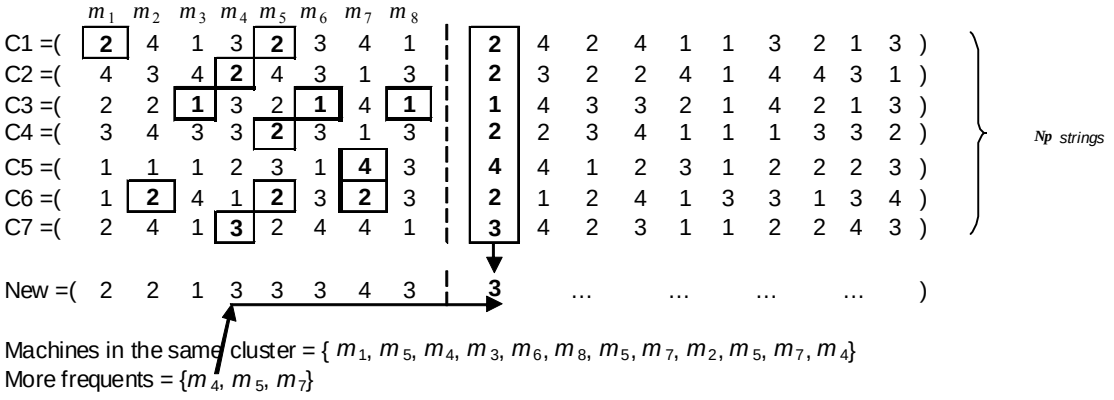

Figure 3. Assigning cluster values to the parts of a new individual.

Local search procedure. The algorithm does a local search among the solutions neighboring the good solutions through a change in the values of the individuals' genes. In this search, the algorithm tries to associate the parts with other *clusters* according to their association with the machines, creating a new set of *clusters.* The algorithm accepts the new arrangement if it improves the fitness of the solution. Then the same operation is carried out, but while rearranging the *clusters* associated with the machines. This procedure has two stages:

Stage 1: For each part *j* a coefficient α_{jk} is calculated with respect to each *cluster k*. Let:

O_{jk} = the number of machines in *cluster k* that process part *j*. The higher this value, the better the *cluster.*

N_{jk} = the number of machines that do not belong to *cluster k* and process part *j*. The higher this value, the worse the *cluster.*

Z_{jk} = the number of machines of *cluster k* that do not process part *j*.

Therefore, coefficient α_{jk} evaluated according to (3) measures the contribution made by part *j* to *cluster k*.

$$\alpha_{jk} = \begin{cases} O_{jk} - N_{jk} - Z_{jk}, & \text{if } O_{jk} \neq 0; \\ -\infty, \text{otherwise.} \end{cases} \tag{3}$$

After evaluating α_{jk} for all *j, k*, the part is associated with the cluster for which α_{jk} is maximal**.** If there is no improvement in the solution, the search ends for the current solution and the next solution is revised.

Stage 2: If stage 1 improves the solution, the algorithm looks for a change of machines between clusters. Let:

O'_{ik} = the number of parts of *cluster k* processed by machine *i*.

N'_{ik} = the number of parts not belonging to *cluster k* and processed by machine *i*.

Z'_{ik} = the number of parts of *cluster k* in which machine *i* does not participate.

Coefficient β_{ik} obtained from (4) measures the contribution of machine *i* to cluster *k*.

$$\beta_{ik} = \begin{cases} O'_{ik} - N'_{ik} - Z'_{ik}, & \text{if } O'_{ik} \neq 0; \\ -\infty, \text{otherwise.} \end{cases} \tag{4}$$

Each machine is associated with a new *cluster* according to the value of β_{ik}. If the solution improves, the algorithm must return to stage 1. Otherwise, the search ends.

2.2) A simulated annealing approach for CFP

Simulated Annealing (SA) is a metaheuristic to determine an optimal or a sub-optimal solution of optimization problems by means of a random search over the solution space (Kirkpatrick et al.. 1983; Silver, 2004).

The algorithm is based on a physical process known as annealing, in which solids are first liquefied by raising the temperature, and then slowly cooled by carefully lowering the temperature to allow intermediate, thermodynamic equilibrium states.

As a result, a regular crystalline, stable and low-energy structure of the solid is produced. Based on the analogy with the annealing process, SA solves an optimization problem by visiting the solution space from an initial solution corresponding to the solid heated at high temperature.

In SA, the temperature is a control parameter that is gradually reduced during the search process. The parameter permits regulation of the acceptance probability of a new solution, analogously with a new thermodynamic equilibrium state simulated during the cooling. The solid formed corresponds to the solution of the optimization problem, and the energy is analogous with the value of the objective function.

Representing optimization problem so that it can be solved by the SA method entails representing a generic solution, generating an initial solution, defining a rule to construct a neighboring solution from a current solution, and defining an evaluation function.

Solution representation. We used the same representation that identifies an individual in the EA described above.

Initial solution. The initial solution is generated by the constructive process described in the previous section to generate initial individuals for the EA parametrically.

Neighborhood. A neighboring solution is constructed by changing a machine or a part of the current solution from one cluster to another. The machine or part is chosen

randomly and the new cluster to which it must belong is also generated randomly from the existing clusters in the current solution.

3. Computational Results

Procedure

To evaluate our proposal, we used the 36 instances available in the literature (Joines et al., 1996; Goncalves and Resende, 2004). The first and second columns in Table 1 identify the instances. The size of the instances is specified as the number of machines and the number of parts.

The proposed algorithms EA and SA in Table 2 are compared with the hybrid genetic algorithm HGA (Goncalves and Resende, 2004), which according to its authors generates the best results in the literature for the CFP, and with our own implementation of the GA-JCK algorithm proposed by (Joines, Culbreth et al., 1996).

We chose the GA-JCK algorithm as representative of the standard GAs because it uses the same objective function. Some of the 36 instances were also solved by Wu *et al*. (2008) with their version of simulated annealing, SACF.

We also compare the results with those of the Zodiac (Chandrasekharan, 1987), Grafics and Ca (Srinivasan and Narendran, 1991), and Ga (Cheng et al., 1998) algorithms.

The computational tests were done on a 1.410 Ghz AMD Athlon microcomputer. Both EA and SA were executed 10 times for each instance, and we report the best results of each case.

The number of generations was 150 for the EA and the same number used in the literature for the HGA. For the execution of the GA-JCK, the number of generations varied according to the size of the instances, in an attempt to execute it the same way as its authors did: for instances 1 to 9, 150 generations; for 10 to 14, 1000 generations; for 15 to 34, 5000 generations; and finally for 35 to 36, 20000 generations. We executed SA until the final temperature reached a value less than or equal to 10^{-6}.

As an initial population of EA, 40% of the individuals were generated randomly and 60% using the HCP heuristic. Such values were defined by preliminary computer tests in

which different rates were used. We found that using percentages of 30, 40 and 50% the results obtained were very similar and the differences were seen mainly in computing time.

The local search proposed in EA is always activated for the 30% best solutions of each iteration (discarding duplications).

To achieve high performance with SA, we must first determine the values of the method's parameters. We calibrated the parameters considering that their values depend more on the type of problem than on its size, so we did the calibration only with instance 34 (Table 1). We used the values of the resultant parameters of this calibration with all the other instances to analyze the final performance of our algorithm.

Preliminary tests of the sensitivity of the objective function to variations of the parameters yielded an order for calibrating the parameters: T_0, number of iterations of the internal cycle, α and T_f. Following this procedure, we determined that SA performs best with T_0=6000, 125 iterations of the internal cycle, and α = 0.9.

Table 1 describes the results obtained by the above described algorithms and by EA. Columns 3-10 show the values obtained by the best solutions of each algorithm.

Influence of a heuristic initial population in EA

We studied the computational impact of the procedure for the CH heuristic construction in the initial population. In this case the 36 instances are solved, with 10 executions of EA each time versus the same algorithm but generating the initial population in a completely random manner (EA-C).

In Table 1, after the size column, each pair of the remaining columns indicates, respectively: the fitness of the best solution of EA and the EA-C, the CPU time in seconds, the average fitness for the 10 executions, and the number of the iteration in which the best solution is found for the first time (bsol). The numbers in boldface point out the best solution found.

Table 1. Results of the proposed algorithm with and without the construction procedure for the initial solution.

Instance	Size	EA	EA-C	EA Time [s]	EA-C Time [s]	EA Average	Average EA-C	bsol EA	bsol EA-C	EA-LS Solution	Improvement
1	5 x 7	**73.68**	**73.68**	0.06	0.06	**73.68**	**73.68**	2	2	73.68	0.00
2	5 x 7	**62.50**	**62.50**	0.06	0.06	62.17	**62.50**	2	3	62.50	0.00
3	5 x 18	**79.59**	**79.59**	0.11	0.11	**79.59**	**79.59**	2	2	75.92	-4.61
4	6 x 8	**76.92**	**76.92**	0.09	0.09	**76.92**	**76.92**	2	2	76.92	0.00
5	7 x 11	**53.13**	**53.13**	0.16	0.15	**53.13**	52.81	2	2	46.51	-12.46
6	7 x 11	**70.37**	**70.37**	0.15	0.15	**70.37**	**70.37**	2	2	65.52	-6.89
7	8 x 12	**68.29**	**68.29**	0.20	0.20	**68.29**	**68.29**	2	2	65.00	-4.82
8	8 x 20	**58.72**	**58.72**	0.29	0.29	**58.72**	**58.72**	2	2	56.88	-3.13
9	8 x 20	**85.25**	**85.25**	0.30	0.29	**85.25**	83.01	2	2	49.52	-41.91
10	10 x 10	**70.59**	**70.59**	0.28	0.29	**70.59**	**70.59**	2	2	70.59	0.00
11	10 x 15	**92.00**	**92.00**	0.36	0.35	**92.00**	**92.00**	2	2	70.18	-23.72
12	14 x 24	**69.86**	**69.86**	1.17	1.12	**69.74**	69.54	3	3	-	-
13	14 x 24	**69.33**	**69.33**	1.25	1.08	**69.33**	69.10	2	2	-	-
14	16 x 24	**51.96**	**51.96**	1.68	1.46	**51.96**	51.33	3	3	26.18	-49.62
15	16 x 30	**67.83**	**67.83**	2.09	1.76	**67.83**	**67.83**	2	2	27.38	-59.63
16	16 x 43	**54.86**	**54.86**	3.26	2.49	**54.86**	**54.86**	2	2	-	-
17	20 x 35	**75.71**	**75.71**	4.02	3.02	**75.71**	**75.71**	2	2	-	-
18	18 x 24	**54.46**	**54.46**	2.24	1.92	**54.12**	53.20	9	7	20.16	-62.98
19	20 x 20	**42.96**	42.86	2.71	1.94	**42.85**	42.78	28	92	26.94	-37.29
20	20 x 23	**49.65**	**49.65**	2.84	2.41	**49.65**	**49.65**	2	2	21.91	-55.87
21	20 x 35	76.14	**76.22**	3.76	3.37	**76.14**	**76.14**	2	62	-	-
22	20 x 35	**58.07**	**58.07**	4.15	3.39	**58.07**	**58.07**	2	2	19.53	-66.36
23	24 x 40	**100.00**	**100.00**	5.61	3.95	**100.00**	**100.00**	2	2	-	-
24	24 x 40	**85.11**	**85.11**	6.77	5.12	**85.11**	**85.11**	2	2	-	-
25	24 x 40	**73.51**	**73.51**	8.76	6.28	**73.51**	70.27	2	2	-	-
26	24 x 40	**51.97**	**51.97**	9.62	8.13	**51.89**	51.52	25	27	-	-
27	24 x 40	**47.06**	46.84	9.96	7.32	**46.75**	45.98	26	83	-	-
28	24 x 40	**44.87**	44.85	9.69	7.08	**44.38**	42.96	139	45	-	-
29	27 x 27	**54.27**	**54.27**	6.46	5.01	**54.26**	**54.26**	2	2	29.05	-46.47
30	28 x 46	**44.62**	44.28	14.64	12.00	**44.26**	44.02	110	4	15.23	-65.86
31	30 x 41	**58.14**	57.86	17.23	13.94	**57.73**	57.24	57	63	-	-
32	30 x 50	**59.66**	**59.66**	18.69	15.82	**59.50**	57.71	2	54	-	-
33	30 x 50	**50.51**	**50.51**	20.63	16.87	**50.33**	46.99	21	11	-	-
34	30 x 90	**43.37**	42.16	39.46	28.29	**41.83**	39.83	63	40	-	-
35	37 x 53	57.54	**58.89**	20.99	18.74	56.67	**56.99**	10	12	41.16	-28.47
36	40 x 100	**84.03**	**84.03**	79.45	61.29	**84.03**	77.72	2	4	-	-

The results show that the use of the individual constructive procedure (CH) on the initial population of the evolutionary algorithm yields a better final solution in 12 of the 36 instances; the same final solution as the EA-C algorithm in 28 cases; and a slightly lower solution than in the EA-C algorithm in only 2 instances. The fact that the EA-C finds a better solution in two cases is not necessarily due to the exclusively random generation of the initial population. Based on the results, we say that regardless of where the search starts,

the algorithm can perform well in a few cases. Also, the average of the best solution of the EA was better in 17 instances, equal in 17 instances, and in only 2 instances was the average slightly lower than the average results of the EA-C. Therefore, we see that in most cases CH tends to improve the quality of the final solution of the EA.

Influence of a local search on EA

We analyzed the impact of local search procedure on the quality of the solution in EA. We experimented with the algorithm without considering the local search module (EA-LS).

The next-to-last column in Table 1 represents the best solution obtained by the EA-LS in 10 executions for each instance and the last column in Table 1 represents the percent improvement over the best solution found by the original EA.

Results with a dash mean that the EA-LS did not succeed in getting feasible solutions. The results show that without a local search, the EA does not find significant results for most of the instances, and in many cases it does not even reach feasible solutions. This demonstrates the importance of an efficient local search in our EA..

Table 2. Efficiency of the grouping obtained with 9 algorithms for 36 instances from the literature.

Instance	Zodiac	Grafics	Ca	Ga	HGA	GA-JCK	SACF	EA	SA
1	73.68	73.68	-	-	73.68	73.68	-	73.68	**75.00**
2	56.52	60.87	-	68.00	62.50	62.50	**69.57**	62.50	**69.57**
3	77.36	-	-	77.36	79.59	79.59	79.59	79.59	**80.85**
4	76.92	-	-	76.92	76.92	76.92	76.92	76.92	**79.17**
5	39.13	53.12	-	46.88	53.13	53.13	**60.87**	53.13	60.00
6	70.37	-	-	70.37	70.37	70.37	**70.83**	70.37	**70.83**
7	68.29	68.29	-	-	68.29	65.12	-	68.29	**69.44**
8	58.33	58.13	**58.72**	58.33	**58.72**	57.26	58.41	**58.72**	**58.72**
9	85.24	85.24	85.24	85.24	**85.25**	**85.25**	**85.25**	**85.25**	**85.25**
10	70.59	70.59	70.59	70.59	70.59	70.59	**75.00**	70.59	**75.00**
11	**92.00**	**92.00**	-	**92.00**	**92.00**	**92.00**	**92.00**	**92.00**	**92.00**
12	64.36	64.36	64.36	-	69.86	69.86	-	69.86	**74.24**
13	65.55	65.55	-	67.44	69.33	67.05	71.21	69.33	**72.86**
14	32.09	45.52	48.70	-	51.96	47.69	-	51.96	**53.33**
15	67.83	67.83	67.83	-	67.83	67.83	-	67.83	**69.92**
16	53.76	54.39	54.44	53.89	54.86	50.88	52.44	54.86	**57.96**
17	-	-	-	-	-	75.71	-	75.71	**78.88**
18	41.84	48.91	44.20	-	54.46	51.24	-	54.46	**57.73**
19	21.63	38.26	-	37.21	42.96	41.14	41.04	42.96	**43.97**
20	38.66	49.36	43.01	46.62	49.65	44.57	**50.81**	49.65	**50.81**

21	75.14	75.14	75.14	75.28	76.14	76.22	78.40	76.14	**79.38**
22	51.13	-	-	55.14	58.06	56.52	56.04	58.06	**58.79**
23	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
24	**85.11**	**85.11**	**85.11**	**85.11**	**85.11**	69.59	**85.11**	**85.11**	**85.11**
25	**73.51**	**73.51**	**73.51**	73.03	**73.51**	**73.51**	**73.51**	**73.51**	**73.51**
26	20.42	43.27	51.81	49.37	51.85	50.30	52.44	51.97	**53.29**
27	18.23	44.51	44.72	44.67	46.50	45.14	47.13	47.06	**48.57**
28	17.61	41.67	44.17	42.50	44.85	43.64	44.64	44.87	**46.00**
29	52.14	41.37	51.00	-	54.27	54.23	-	54.27	**54.82**
30	33.01	32.86	40.00	-	43.85	27.71	-	44.62	**47.68**
31	33.46	55.43	55.29	53.80	57.69	55.32	62.42	58.14	**62.86**
32	46.06	56.32	58.70	56.61	59.43	52.54	**60.12**	59.66	59.66
33	21.11	47.96	46.30	45.93	50.51	49.23	50.51	50.51	**50.55**
34	32.73	39.41	40.05	-	41.71	15.78	-	43.37	**47.93**
35	52.21	52.21	-	-	56.14	56.83	-	57.54	**61.16**
36	83.66	83.92	83.92	**84.03**	**84.03**	**84.03**	**84.03**	**84.03**	**84.03**

Solution quality

Comparing EA with other evolutionary approaches (Ga, HGA and GA_JCK), the results are better in 8 of the 36 instances; the same in 26 other cases, and for instances 2 and 21, worse than the best fitness value in the literature.

However, the solution that algorithm Ga found for instance 2 contains unit clusters that in general are considered invalid because they refer to situations in which a cell contains a single machine or a family has a single part.

In this way, when we execute our algorithm to accept solutions with such clusters for instance 2, we see that the EA also finds the best solution obtained by Ga. Therefore, using the same feasibility criteria, the EA performs worse than the best results in the literature in a single case (instance 21), with a difference of 1%. On the other hand, SA not only found the best solutions known so far, but it also found better solutions for some problems. Specifically, new better solutions were found for 23 instances. The SACF algorithm found better solutions than SA in instances 5 and 32.

Execution time

Table 3 shows, for EA, SA and GA-JCK, the average execution time in seconds, obtained from the 10 executions for each instance.

For the HGA and SACF, the execution times available from Gonçalves and Resende (2004)and from Wu *et al*. (2008) are shown.

Table 3. Average execution time in seconds and generation of the best solutions for the GA-JCK, EA, and HGA

Instance	Time GA-JCK [s]	Time HGA [s]	Time SACF [s]	Time EA [s]	Time SA [s]
1	0.04	0.28	-	0.06	0.14
2	0.03	0.35	< 0.01	0.06	0.16
3	0.05	0.47	0.01	0.11	0.27
4	0.05	0.41	< 0.01	0.09	0.17
5	0.07	0.73	0.02	0.16	0.27
6	0.07	0.73	0.01	0.15	0.26
7	0.09	0.96	-	0.20	0.33
8	0.14	1.27	0.03	0.29	0.56
9	0.13	1.28	0.02	0.30	0.53
10	0.97	1.36	0.02	0.28	0.32
11	1.10	1.46	0.02	0.36	0.44
12	2.97	4.60	-	1.17	0.81
13	3.01	4.67	0.45	1.25	0.78
14	3.92	6.53	-	1.68	0.89
15	24.10	7.51	-	2.09	1.30
16	32.21	10.50	1.10	3.26	1.56
17	50.49	-	-	4.02	1.63
18	27.51	8.28	-	2.24	0.98
19	29.85	9.12	0.35	2.71	1.29
20	27.48	9.97	0.60	2.84	1.29
21	52.28	12.09	1.21	3.76	1.60
22	52.92	13.44	0.80	4.15	1.85
23	166.91	14.65	1.57	5.61	2.04
24	89.63	20.04	1.82	6.77	2.18
25	96.22	21.50	1.51	8.76	2.30
26	102.63	22.81	3.41	9.62	2.22
27	94.14	23.05	5.82	9.96	2.30
28	99.81	22.56	5.01	9.69	2.41
29	79.70	19.90	-	6.46	2.24
30	124.90	34.12	-	14.64	2.25
31	161.02	34.19	9.63	17.23	2.09
32	165.59	40.95	15.44	18.69	2.94
33	224.97	41.68	17.59	20.63	3.02
34	238.49	68.99	-	39.46	4.03
35	3387.51	58.35	-	20.99	3.37
36	3684.26	125.33	106.93	79.45	6.15

In terms of computing time, EA requires on the average less time than the HGA and the GA-JCK. Comparing the average times of the EA and the HGA, the difference in favor of the EA increases as the size of the instance increases. This is because in EA, the local search analyzes fewer solutions. The shortest computing times for 18 of the first 25 instances correspond to SACF.

For the first 11 instances, EA requires a shorter computing time than SA, while for instances 12 to 36 it solves the CFP in the shortest computer time. It is clear that as the size of the instances increases, so does computer time to solve them with whichever algorithm, however the smallest time increase occurs with SA, which requires an average of 6.15 seconds to solve the last instance.

4. Conclusions

We have introduced a specialized evolutionary algorithm to solve the Manufacturing Cell Formation Problem. The evolutionary algorithm combines a heuristic for individual construction, a generic crossover procedure, and an efficient local search procedure to achieve better performance. Using the same representation of a solution, we also have proposed a simulated annealing approach. We also implemented a genetic algorithm available in the literature (GA-JCK) for comparison. We compared the results of the algorithms with the results of 5 other algorithms from the literature.

Our experiments demonstrated the importance of the heuristic that generates good quality solutions to initialize a genetic algorithm, as well as the need for an efficient local search mechanism.

To counterbalance the additional computer work required by these components, we showed that it is sufficient to reduce the number of iterations of a GA. That is, the introduction of a previously semi-optimized set to initialize a genetic algorithm and an efficient local search tend to reduce drastically the number of iterations needed to reach suboptimal solutions.

With such elements used to strengthen the performance of a GA, it has not yet been possible to compete with the efficiency shown by SA in solving this set of instances for the problem. It not only finds the best solution known to have been found by other algorithms for various instances of the problem, but in several of the instances studied, SA finds a better solution.

Acknowledgments

The first and third authors were supported by The Millennium Scientific Institute: Complex Engineering Systems.

References

Brown, E. C., & Sumichrast, R. T. (2001). CF-GGA: a grouping genetic algorithm for the cell formation problem. *International Journal of Production Research*, *39*(16), 3651-3669.

Chandrasekharan, M. P. (1987). ZODIAC: an algorithm for concurrent formation of part-families and machine-cells. *International Journal of Production Research* , *25*(6), 835-850.

Cheng, C. H., Gupta, Y. P., Lee, W. H., & Wong, K. F. (1998). A TSP-based heuristic for forming machine groups and part families. *International Journal of Production Research*, *36*(5), 1325-1337.

Defersha, F. M., & Chen, M. (2006). A comprehensive mathematical model for the design of cellular manufacturing systems. *International Journal of Production Economics*, *103*(2), 767-783.

Drummond, L. M. A., Ochi, L. S., & Vianna, D. S. (2001). An asynchronous parallel metaheuristic for the period vehicle routing problem. *Future Generation Computer Systems*, *17*(4), 379-386.

Glover, F. W., & Laguna, M. (1998). *Tabu Search* (1º ed., p. 408). Springer.

Goldberg, D. (1989). *Genetics Algorithms in Search, Optimizaton and Machine Learning*. Reading: Addison-Wesley Professional.

Goncalves, J. F., & Resende, M. G. (2004). An evolutionary algorithm for manufacturing cell formation. *Computers & Industrial Engineering*, *47*(2-3), 247-273.

James, T. L., Brown, E. C., & Keeling, K. B. (2007). A hybrid grouping genetic algorithm for the cell formation problem. *Computers & Operations Research*, *34*(7), 2059-2079.

Joines, J. A., Culbreth, C. T., & King, R. E. (1996). Manufacturing cell design: An integer programming model employing genetic algorithms. *Iie Transactions*, *28*(1), 69-85.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671--680.

Mak, K. L., Wong, Y. S., & Wang, X. X. (2000). An adaptive genetic algorithm for manufacturing cell formation. *International Journal of Advanced Manufacturing Technology*, *16*(7), 491-497.

Muruganandam, A., Prabhaharan, G., Asokan, P., & Baskaran, V. (2005). A memetic algorithm approach to the cell formation problem. *International Journal of Advanced Manufacturing Technology*, *25*(9-10), 988-997.

Selim, H. M., Askin, R. G., & Vakharia, A. J. (1998). Cell formation in group technology: Review, evaluation and directions for future research. *Computers & Industrial Engineering*, *34*(1), 3-20.

Silver, E. A. (2004). An overview of heuristic solution methods. *Journal of the Operational Research Society*, *55*(9), 936-956.

Srinivasan, G., & Narendran, T. (1991). Grafics - a Nonhierarchical Clustering-Algorithm for Group Technology. *International Journal of Production Research*, *29*(3), 463-478.

Wang, J. (1999). A linear assignment clustering algorithm based on the least similar cluster representatives. *Ieee Transactions on Systems Man and Cybernetics Part a-Systems and Humans*, *29*(1), 100-104.

Wu, T., Chang, C., & Chung, S. (2008). A simulated annealing algorithm for manufacturing cell formation problems. *EXPERT SYSTEMS WITH APPLICATIONS*, *34*(3), 1609-1617.

Yin, Y., & Yasuda, K. (2006). Similarity coefficient methods applied to the cell formation problem: A taxonomy and review. *International Journal of Production Economics*, *101*(2), 329-352.