# A Metaheuristic Compendium for Scheduling Problems

Carlos Bermudez[1], Gabriela Minetti[1], and Carolina Salto[1,2]

[1] Facultad de Ingeniería, Universidad Nacional de La Pampa
[2] CONICET, Argentina
{minettig,saltoc}@ing.unlpam.edu.ar

**Abstract.** The flexible job shop scheduling problem (FJSSP) is a difficult and complex problem, proved to be NP-hard, in manufacturing environments, because it has to assign each operation to the appropriate machine besides sequencing operations on machines. Due to that complexity, metaheuristics became the best choice to solve in practice this kind of problem. Therefore, the aim of this paper is to offer a reliable compendium in order to cover a wide algorithmic spectrum of different techniques. Further, a study of their accuracy and computational effort is carried out in order to achieve a behavior comparison. This paper shows different algorithmic trends that can be observed through this analysis.

## 1 Introduction

Scheduling operations is one of the most critical issues in manufacturing processes, being the Job Shop Scheduling Problem (JSSP) one of most important and difficult problems [1]. Each job has to undergo multiple operations on the various machines and each job has its own set of processing times and routing characteristics. The decision concerns how to sequence the operations on the machines, so that the time needed to complete all the jobs (makespan) is minimized. The possibility of selecting alternative routes among the machines is useful in production environments where multiple machines are able to perform the same operation (possibly with different processing times), as it allows the system to absorb changes in the demand of work or in the performance of the machines. When this factor is considered, the problem is known as Flexible Job Shop Scheduling Problem (FJSSP).

Generally, the FJSSP is a more realistic production environment and then has more practical applicability. However, the FJSSP is more complex (NP-hard problem [2]) than the JSSP because of its additional decision to assign each operation to the appropriate machine (routing) besides sequencing operations on machines. Due to the mentioned complexity of FJSSP, the adoption of heuristic methods is suggested because they produce reasonably good schedules in a reasonable time, instead of looking for an optimum solution, also for small instances. In recent years, the adoption of metaheuristics [3] has led to better results than classical dispatching or greedy heuristic algorithms [4–6].

The main purpose of this paper is to offer a base reference for FJSSP comparison, using different metaheuristics in order to cover a wide algorithmic spectrum, from trajectory methods to population-based methods. In this way, we obtain a valuable compendium of approaches for FJSSP by specifying their qualities from two points of view: first, adapting the metaheuristics to solve the problem, and, second, finding good quality schedules with a low computational effort. Consequently, the methodology followed in this empirical study consists in adapting five metaheuristics to solve the FJSSP. We evaluate and compare each of them on the well-known set of the problem instances. The planned analysis and comparisons will help us to explain the result qualities and efficiency of these metaheuristics to solve the FJSSP and also, to compare the best studied metaheuristics with different proposals found in the literature.

The paper is organized as follows. In Section 2, we introduce the problem formulation. In Section 3, we explain the adaptations of the selected metaheuristics to solve the FJSSP. In the following section, we introduce the experimental design and in Section 5, we evaluate the results. Further, in Section 6 we make a comparison between these metaheuristics and the ones present in the literature. Some final remarks and future research directions are given in Section 7.

## 2 The Flexible Job Shop Scheduling Problem

The flexible job shop can be described as follows. Let a set $J = \{J_1, J_2, ..., J_n\}$ of independent jobs. A job $J_i$ is formed by a sequence of $O_{i1}, O_{i2}, ..., O_{in_i}$ operations to be performed one after the other according to the given sequence. Given a set $U = \{M_1, M_2, ..., M_m\}$ of machines, each operation $O_{ij}$ can be processed on a subset $U_{ij} \subseteq U$ of compatible machines. We have partial flexibility if there exists a proper subset $U_{ij} \subset U$, for at least one operation $O_{ij}$, while we have $U_{ij} = U$ for each operation $O_{ij}$ in the case of total flexibility. The processing time of each operation is machine-dependent. We denote with $d_{ijk}$ the processing time of operation $O_{ij}$ when executed on machine $M_k$. Pre-emption is not allowed and the machines cannot perform more than one operation at a time. All jobs and machines are available at time 0. The problem is to assign each operation to an appropriate machine (routing problem), and to sequence the operations on the machines (sequencing problem) in order to minimize the makespan. This measure is the time needed to complete all the jobs, which is defined as $C_{max} = max_i\{C_i\}$, where $C_i$ is the completion time of job $J_i$.

## 3 Metaheuristics for FJSSP

In this section, we describe how the five metaheuristics used in this work are adapted to solve the FJSSP. Two of them are trajectory-based metaheuristics (Simulated Annealing and Iterated Local Search) and the remaining ones are population-based metaheuristics (Genetic Algorithm, Cuckoo Search, and Imperialist Competitive Algorithm). All of them share two main common design points: the representation of solutions handled by algorithms and the definition of the objective function that will guide the search.

We use the encoding proposed by Bierwirth in [7], which is based on permutation with repetitions. A solution, $S$, is a permutation of the set of operations that represents a tentative ordering to schedule them, each one being represented by its job number. For example, $S = [2, 1, 1, 3, 2, 3, 1, 2]$ is a valid solution, which corresponding operation sequence is $O_{21}$, $O_{11}$, $O_{12}$, $O_{31}$, $O_{22}$, $O_{32}$, $O_{13}$, and $O_{23}$. In order to evaluate $S$, the fitness value is the makespan ($C_{max}$) of this solution. To compute this value, each operation $O_{ij}$ in $S$ is assigned to a feasible machine $M_k$ in the subset $U_{ij}$ with the shortest completion time, and then the load of $M_k$ must be updated. As the initial solution we use a random procedure, mainly because high performing construction heuristics for the FJSSP are unknown.

Finally, in order to make a fair comparison among these algorithms, they must do the same computational effort in each run. This can be reached if they are executed during the same time for each problem instance. As suggested in the literature for JSSP, the total executed time is calculated as $ExecTime = \sharp O \times (\frac{\sharp O}{2}) \times 30$, where $\sharp O$ is the total number of operations for a given instance.

### 3.1 Simulated Annealing

Simulated Annealing (SA) is a simple and general purpose Monte-Carlo method which was developed for combinatorial optimization [8]. This version of SA generates an initial solution $S_0$ in a random way, and a neighbor, $S_1$, from $S_0$ using the exchange operator. This operator randomly selects two positions and their respective operations are swapped if they belong to different jobs, since the encoding is a permutation with repetitions. If $S_1$ is worse than $S_0$, $S_1$ can be accepted under the Boltzmann probability. In this way, at high temperatures ($T$) the exploration of the search space is allowed. In contrast, at low temperatures the algorithm only exploits a promising region of the search space. In order to update $T$, the proportional cooling process [8] is used and it is applied after a certain number of iterations given by the Markov Chain Length ($MCL$). Finally, SA ends the search when the total executed time ($ExecTime$) is reached.

### 3.2 Iterated Local Search

The Iterated Local Search (ILS) [9] is a simple but very effective metaheuristic. To solve the FJSSP with ILS, we use the ideas proposed in [10]. The algorithm 1 begins the search from an initial solution $x$ (line 2). After that, the main loop of ILS starts, which consists in the application of local search and perturbation procedures (lines 4-21). A local search procedure (LS) is applied to the current solution $x$ (line 5). This search stops as soon as a better solution $x'$ is found. When no improvements are found, a counter ($count$) is increased (line 12). This counter is set to zero each time the local search improves the current solution (line 14). If the counter exceeds a certain threshold ($threshold$), a perturbation mechanism is applied, with the aim of redirecting the search to more promising regions of the solution space (lines 13-19). This mechanism is applied to the current solution $x$ a certain number of times (defined by $nu\_move$) generating a set $\tilde{s}(r)$ of candidate perturbation solutions. The best of them is selected as the new current solution $x$.

The local search uses iteratively the exchange operator and consists in the following steps. The operation $O_{ij}$ in the first position of the current solution is exchanged by an operation located in a different randomly selected position in the permutation. If this new permutation $x'$ has a better makespan ($f(x') < f(x')$), then it will replace the solution $x$ and the local search procedure ends.

**Algorithm 1** ILS to solve the FJSSP

```
1: $count = 0$;
2: Initialize $x$; {ILS generates a random solution}
3: $x_{best} = x$;
4: while $stop\ criterion\ not\ met$ do
5:     $x' = \text{LS}(x)$; {Interchange operator}
6:     if $f(x') < f(x)$ then
7:         $x = x'$;
8:         if $f(x') < f(x_{best})$ then
9:             $x_{best} = x'$;
10:        end if
11:    else
12:        $count = count + 1$;
13:        if $count > threshold$ then
14:            $count = 0$;
15:            for $r = 1$ to $nu\_move$ do
16:                $\tilde{s}(r) = perturbation(x)$;
17:            end for
18:            $x = \tilde{s}_{best}$;
19:        end if
20:    end if
21: end while
22: return $x_{best}$;
```

**Algorithm 2** CS to solve the FJSSP

```
1: $t = 0$; {current generation}
2: initialize($H(t)_{1,N}$);
3: evaluate($H(t)$);
4: sort($H(t)$); {Rank the solutions}
5: while ($stop\ criterion\ not\ met$) do
6:     $pos$ =random_pos($1, N \times p_{best}$);
7:     $h$ = newCuckoo($H(t)_{pos}$);
8:     $pos1$ =random_pos($1, N$)
9:     if $f(h) < f(H(t)_{pos1})$ then
10:        $H(t)_{pos1} = h$;
11:    end if
12:    initialize($H(t)_{N \times p_a, N}$);
13:    LS($H(t)$);
14:    sort($H(t)$); {Rank the solutions}
15:    $t = t + 1$;
16: end while
17: return best solution
```

On the contrary, the procedure continues with the operation in the second position of the permutation, and so on. The search iterates in the majority of the operations, without repetition. The perturbation mechanism consists in the insertion operator. Basically, one job is randomly selected and inserted in a different position of the permutation, which is also randomly selected.

### 3.3 Cuckoo Search Algorithm

Cuckoo search (CS) algorithm is a novel metaheuristic [11], which is inspired on the obligate brood parasitic behavior of cuckoo birds in combination with the Lévy flight behavior. As the first step of the CS to solve the FJSSP (see Algorithm 2), a population $H(0)$ of $N$ eggs (solutions) is randomly generated (line 2). This initial population is evaluated and then sorted regarding the quality of each solution (lines 3 and 4, respectively). After these steps, the main loop follows. A cuckoo selects randomly a position *pos* from the $p_{best}$ best solutions in the current population $H(t)$ (line 6). A new candidate solution $h$ is generated by perturbing the current $H(t)_{pos}$ solution following Lévy flights (line 7). The new solution $h$ can replace a randomly selected solution from $H(t)$, following an elitist selection strategy (lines 8-11). After that, a fraction $p_a$ of worse nests is abandoned and new nests are built at new locations (solutions created following a random process) (line 12). Afterwards, a local search step is applied to each individual in $H(t)$ with a $p_{LS}$ probability in order to improve the solution (line 13). At this step the elitist selection strategy is also used. Finally, the loop ends with a sorting process which arranges the individuals in decreasing fitness value (line 14), being the best solution the one present in position 0 of $H(t)$.

To move from a current solution to an other one, the concept of Lévy flight is used. A Lévy flight can be described as a random walk in which the step length (distance between two solutions) is decided by certain probability distribution functions which are heavily tailed. Following the ideas of Ouaarab et al.[12], three different moves or operators, controlled directly by the value generated by Lévy distribution, are used: exchange, insertion, and inversion. These operators are traditional for the problem at hand and generate different move steps. Exchange move is used for small perturbations and large ones are made by inversion move. Insertion move is considered to introduce median perturbations.

## 3.4 Genetic Algorithm

Genetic Algorithms (GA) [13] simulate the evolution of individual structures via the Darwinian natural selection process. The GA proposed to solve FJSSP begins with the creation of an initial population of $\mu$ solutions in a random way, which are then evaluated. After that, the population goes into a cycle (evolution), which consists of the application of genetic operators, to create $\lambda$ offspring, obtaining a new population. Each parent for mating is selected using binary tournament selection and they are recombined using a specially designed crossover known as the Job-based Order Crossover (JOX) [14], under a certain probability ($p_c$). Each individual in the new population is mutated, with a certain probability ($p_m$), by using the exchange operator. Finally, each iteration ends by selecting $\mu$ individuals to build up the new population from the set of $(\mu + \lambda)$ existing ones by using proportional selection (a typical selection method in this step).

## 3.5 Imperialist Competitive Algorithm

The Imperialist Competitive Algorithm (ICA) [15], is inspired by the imperialistic competition. The Algorithm 3 shows the ICA to solve FJSSP which is based on the ICA version presented in [16]. An initial set of countries (solutions) of size $N_C$ is randomly created (line 1). Then the imperialistic countries are determined (lines 2-4). For that and after calculating the fitness function $c_i = 1/C_{max}$ for each country $i$, the best $N_{imp}$ of them are selected as imperialists. The rest $N_{col} = N_C - N_{imp}$ countries are the colonies. These colonies are divided among imperialists (lines 5-7), based on their power, in order to form the initial empires. The power $p_j$ of each imperialistic country is first calculated according to $p_j = c_j / \sum_{i=1}^{N_{imp}} c_i$. The number of colonies of each imperialistic country ($N_{col_j}$) is proportional to its power, and it is determined by $N_{col_j} = p_j \times N_{col}$. A total of $N_{col_j}$ colonies are randomly assigned to an imperialist $j$.

After this initialization process, the colonies in each of the empires start moving toward their relevant imperialist country. In this work, those movements are accomplished by variation operators specially designed by the adopted representation. One of them is the JOX operator [14], which is applied in such a way that a colony and its imperialist country are considered as parents. The other one is the exchange operator; it is applied to each colony in order to simulate a randomly deviated direction, as the original proposal of ICA. At this point and following the improvement done in [16], ICA applies a local search to the imperialist countries with a certain probability ($p_{LS}$), using the same procedure described in Section 3.2. When moving toward the imperialist country, a colony might reach a position with higher fitness than its imperialist country (lines 13-15). In this case, the imperialist and the colony should change their positions. If there are several colonies better than the imperialist country, then the imperialist will be replaced with the best colony.

**Algorithm 3** ICA to solve the FJSSP

```
1: initialize $N_C$ countries;
2: evaluate $c_i$ for each country;
3: select the best $N_{imp}$ countries as imperialists;
4: choose the remaining $N_{col}$ countries as colonies;
5: compute $p_j$ for each empire;
6: determine $N_{col_j}$ for each empire $j$;
7: select the colonies for assigning to each imperialist in a random way;
8: while (stop criterion is not meet) do
9:     apply JOX between each colony and its imperialist;
10:    apply the exchange operator to each colony;
11:    evaluate $c_i$ for all colonies;
12:    apply a local search to $j^{th}$ imperialist;
13:    if $c_i$ is better than $c_j$ then
14:        exchange the positions between the imperialist and colony;
15:    end if
16:    compute $tp_j$ for each empire;
17:    select the weakest colony from the weakest empire and give it to the strongest empire;
18:    update $N_{col_j}$ of the weakest and strongest empires;
19:    eliminate the empire with no colonies and $N_{imp} = N_{imp} - 1$;
20: end while
21: return the strongest imperialist;
```

Following, the total power of each empire ($tp_j$) is computed including the power of the imperialist country and the power of its colonies, according to $tp_j = c_j + \sigma \times \sum_{i=1}^{N_{col_j}} c_i$ [15] (line 16). The parameter $\sigma \in [0,1]$ causes the total power of the empire to be determined by just the imperialist country ($\sigma = 0$) or by the colonies increasing the $\sigma$ value ($\sigma > 0$). We adopt the value of $\sigma = 0.1$ in our implementation as suggested in [15]. In line 17, ICA selects the empire with the highest $tp_j$ as the best one and increase its number of colonies by one ($N_{col_j} + 1$). The colony of the weakest empire, $k$, with the lowest $c_i$ is considered as the weakest one, and its number of colonies is reduced by one ($N_{col_k} - 1$). The number of empires also decreases by one ($N_{imp} - 1$) if the weakest empire $k$ has $N_{col_k} = 0$ (line 19). Finally, if more than one empire remained or the difference between the previous best imperialist and the current one is greater than $1^{e^{-6}}$, and the execution time is less than the total time then go to line 8.

## 4 Experimental Design

In this section, we describe the experimental design used in this approach. We have selected a wide range of instances used in the literature taking into account their complexity, which is given by the number of jobs and machines, and the wide variation of flexibility in the amount of available machines per operation. In this sense, we considered the data set proposed by Brandimarte [17], since the number of jobs ranges from 10 to 20, the number of machines belongs to the set {4,15} and the number of operations for each job ranges from 5 to 15, consequently the total number of operations ranges from 55 to 240. Taking into account the flexibility varies between 1.43 and 4.10.

The parameter values of the proposed algorithms are selected based on some preliminary trials. The selected parameters are those values that gave the best results concerning both the solution quality and the computational effort. In

**Table 1.** Parametric configuration of the SA, ILS, GA, CS and ICA algorithms.

| SA | | ILS | | GA | | CS | | ICA | |
|---|---|---|---|---|---|---|---|---|---|
| *MCL* | 100 | *threshold* | 15 | $\mu$ | 50 | $N$ | 50 | $N_C$ | 50 |
| | | *un_move* | 30 | $\lambda$ | 50 | $p_{LS}$ | 0.1 | $p_{LS}$ | 0.1 |
| | | $p_{LS}$ | 0.1 | $p_c$ | 0.8 | $p_{best}$ | 60% | $N_{imp}$ | 5 |
| | | | | $p_m$ | 0.3 | $p_a$ | 20% | $\sigma$ | 0.1 |
| *stop criterion* $\sharp O \times (\frac{\sharp O}{2}) \times 30$ | | | | | | | | | |

order to make a fair comparison among these algorithms, they are executed during the same time for each problem instance. The parametric configuration used in each algorithm is shown in Table 1.

Because of the stochastic nature of the algorithms, we performed 30 independent runs of each test to gather meaningful experimental data and apply statistical confidence metrics to validate our conclusions. As a no normal distribution is followed by the data, we used the Kruskal-Wallis (KW) test. This statistical study allows us to assess whether or not there were meaningful differences between the compared algorithms with a confidence level of 99%.

## 5 Experimental Results

In this section, we analyze the quality of results considering the $C_{max}$ values obtained for each algorithm described in Section 3 to solve the FSSSP instances. Additionally, we study the hit rate and the distribution of the normalized gap between the best solution found by each proposed metaheuristic and the best known $C_{max}$ for each instance and for all algorithms.

Analyzing the algorithm performance from the hit rate obtained for each of them (see Table 2), the general trend is that trajectory-based algorithms are more efficient than population-based algorithms. SA obtains the highest hit rate values in the majority of the instances. Regarding the population-based algorithms, GA presents the highest mean hit values, followed by CS and ICA, respectively.

**Table 2.** Hit rate obtained by SA, ILS, GA, CS, and ICA for all FJSSP instances.

| Inst. | SA | ILS | GA | CS | ICA |
|---|---|---|---|---|---|
| 1 | **100.00** | **100.00** | **100.00** | 96.67 | 63.33 |
| 2 | **90.00** | 53.33 | 10.00 | 23.33 | 0.00 |
| 3 | **100.00** | **100.00** | **100.00** | **100.00** | **100.00** |
| 4 | **100.00** | **100.00** | 40.00 | 16.67 | 0.00 |
| 5 | **30.00** | 3.33 | 0.00 | 3.33 | 0.00 |
| 6 | **33.33** | 10.00 | 0.00 | 0.00 | 0.00 |
| 7 | **26.67** | 3.33 | 0.00 | 0.00 | 0.00 |
| 8 | 96.67 | **100.00** | **100.00** | **100.00** | **100.00** |
| 9 | 93.33 | **100.00** | 70.00 | 6.67 | 0.00 |
| 10 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Mean | 67.00 | 56.99 | 42.00 | 34.66 | 26.33 |

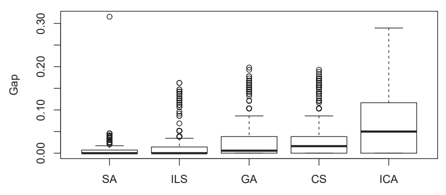

**Fig. 1.** Gap values obtained by SA, ILS, GA, CS, and ICA considering all FJSSP instances.

**Table 3.** Mean execution time (sec.) to find the best $C_{max}$

| Inst. | SA | ILS | GA | CS | ICA | KW |
|---|---|---|---|---|---|---|
| 1 | 2.27 | 0.23 | 0.73 | 3.20 | 0.23 | + |
| 2 | 8.23 | 6.97 | 8.60 | 7.33 | 0.13 | + |
| 3 | 0.00 | 0.00 | 0.10 | 0.03 | 0.00 | + |
| 4 | 3.50 | 13.83 | 58.37 | 56.93 | 0.37 | + |
| 5 | 19.40 | 7.60 | 37.97 | 25.80 | 0.10 | + |
| 6 | 121.13 | 196.40 | 164.13 | 160.83 | 1.20 | + |
| 7 | 24.23 | 42.53 | 65.27 | 66.17 | 0.97 | + |
| 8 | 0.00 | 1.00 | 2.03 | 8.83 | 0.20 | + |
| 9 | 16.03 | 148.00 | 421.13 | 519.27 | 1.60 | + |
| 10 | 711.87 | 610.30 | 658.17 | 605.93 | 9.83 | + |
| Mean | 90.67 | 102.69 | 141.65 | 145.43 | 1.46 | + |

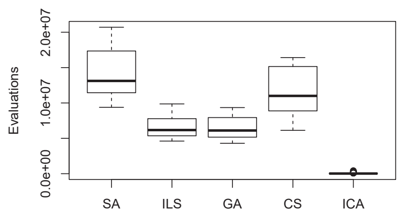

**Fig. 2.** Total number of evaluations done by running SA, ILS, GA, CS, and ICA considering all FJSSP instances.

Complementary information to the previous analysis is the distribution of the gap values shown by the algorithms. For that purpose, Figure 1 illustrates these results by means of a box-plot graph. These results support the previous analysis. This allows to remark that the trajectory-based algorithms achieve schedules with the lowest $C_{max}$ values, solving the FJSSP more accurately than the population-based ones.

Regarding the mean execution time to find the best $C_{max}$, the results presented in Table 3 show that ICA is the fastest algorithm to find its best solutions but their qualities are very poor, as observed in previous analysis. In general, this algorithm ends because one empire conquers the remaining empires with its colonies at the first iterations. As a consequence, ICA does not consume the total execution time computed for each instance. Instead, the longest processing time corresponds to CS and GA. In order to check the confidence of the results a non-parametric test, the KW test, was applied (normality conditions not met). In the last column of the Table 3, the results of this test are shown where the symbol "+" indicates significant differences between the algorithms. Consequently, a post-hoc statistic analysis reveals that CS, GA, and ILS present a similar execution time to find the best solution, but it is noticeable that ILS is able to find better quality solutions than the remaining ones (see Table 2). On the other hand, ILS and SA have statistically similar execution times, but exhibiting significant differences with the population-based algorithms.

Figure 2 shows the distribution of the total number of evaluations carried out by each algorithm. We observe that the algorithms with more evaluations are SA and CS, although SA outperforms CS from the result quality point of view (see Table 2). This means that the SA procedure has a better balance between exploration and exploitation than CS. Instead, ILS and GA do a statistically similar total number of evaluations. However, ILS presents the best behavior, obtaining good solutions in a low number of evaluations. These suggest that ILS does a better exploration and exploitation during the search than GA. Finally, ICA carried out the minimal total number of evaluations and this justifies its poor performance to solve FJSSP.

Considering the quality of results and the computational effort of the algorithms, we can observe two main groups of them and this division corresponds to the classification of the metaheuristics: the trajectory-based algorithms (SA

**Table 4.** Comparison between SA and metaheuristics from the literature

| inst | SA | PVNS | TSPCB | hGA | BEDA | IACO |
|---|---|---|---|---|---|---|
| 1 | **40** | **40** | **40** | **40** | **40** | **40** |
| 2 | **26** | **26** | **26** | **26** | **26** | **26** |
| 3 | **204** | **204** | **204** | **204** | **204** | **204** |
| 4 | **60** | **60** | 62 | **60** | **60** | **60** |
| 5 | **172** | 173 | **172** | 173 | **172** | 173 |
| 6 | **58** | 60 | 65 | 60 | 60 | 60 |
| 7 | **139** | 141 | 140 | 140 | **139** | 140 |
| 8 | **523** | **523** | **523** | **523** | **523** | **523** |
| 9 | **307** | **307** | 310 | **307** | **307** | **307** |
| 10 | **200** | 208 | 214 | 205 | 206 | 208 |

and ILS) and the population-based ones (GA, CS, and ICA). The algorithms in the first group outperforms the algorithms in the second one, showing a relatively good trade-off between solution quality and computational effort. This is highly related with the algorithmic procedure carried out by each one of these metaheuristics and by the number of evaluations per iteration. Particularly, SA obtains the best solution quality (the highest hit rate values and the lowest gap values) in a low mean execution time, exhibiting between exploration and exploitation.

## 6 Comparison with other algorithms

To determine the goodness of the metaheuristics considered in this work, this section presents a comparison of the results from the best performing algorithm (SA) with several competitive algorithms present in the literature. This allows a comparative assessment of the algorithms for the FJSSP. In this comparison different metaheuristics to solve the FJSSP are considered: *i*) a variable neighborhood search (PVNS) [18], *ii*) a tabu search (TSPCB) [19], *iii*) a hybrid algorithm combining chaos particle swarm optimization with genetic algorithm (hGA) [20], *iv*) a bi-population based estimation of distribution algorithm (BEDA) [21], and finally *v*) an ant colony optimization (IACO) [22].

The $C_{max}$ values of SA and the algorithms included in the comparison are listed in Table 4. From the results, the $C_{max}$ values of SA are equal or lower than the values of remaining algorithms for dealing with almost all ten instances. This observation suggests that the SA developed in this work is a competitive algorithm to solve the FJSSP. Comparisons regarding computational effort are hard to be carried out because the majority of the works do not report the number of evaluations and also, the hardware used for the experimentation has different configurations. Consequently, the relative efficiency of the referred algorithms is difficult to contrast in order to obtain meaningful comparisons.

## 7 Conclusion

In this article, we have presented a compendium of different metaheuristics in order to solve the FJSSP. In this study two trajectory-based metaheuristics and three population-based ones are included to give a wide spectrum of possible solutions to the mentioned scheduling problem. The results indicate that the trajectory-based metaheuristics were accurate and efficient for the FJSSP. In the study carried out in this work, SA is the best performing algorithm to solve the FJSSP. Moreover, when SA is contrasted with algorithms in the literature, it also becomes in the best approach. As a consequence, SA gives accurate solutions to this NP-hard problem in an efficient and competitive way.

As future research activities, we plan to widen the scope of the study by including another set of instances with high dimensionality. Furthermore, variants of the FJSSP with more constraints will be evaluated considering the approaches developed in this article.

**Acknowledgements**

The authors acknowledge the support of Universidad Nacional de La Pampa, and ANPCYT under contract PICTO 2011-0278 and the Incentive Program from MINCyT. The third author is also funded by CONICET.

## References

1. M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Springer Publishing Company, Incorporated, 2008.
2. M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976.
3. E.-G. Talbi, *Metaheuristics: From Design to Implementation.* Wiley, 2009.
4. G. Vilcot and J.-C. Billaut, "A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem," *European Journal of Operational Research*, vol. 190, no. 2, pp. 398 – 411, 2008.
5. G. Zhang, X. Shao, P. Li, and L. Gao, "An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem," *Computers & Industrial Engineering*, vol. 56, no. 4, pp. 1309 – 1318, 2009.
6. G. Zhang, L. Gao, and Y. Shi, "An effective genetic algorithm for the flexible job-shop scheduling problem," *Expert Syst. Appl.*, vol. 38, pp. 3563–3573, 2011.
7. C. Bierwirth, "A generalized permutation approach to job shop scheduling with genetic algorithms," *Operations-Research-Spektrum*, vol. 17, pp. 87–92, 1995.
8. S. Kirkpatrick, C. G. Jr, and M. Vecchi, "Optimization by simulated annealing," *Science*, no. 220, pp. 671–680, 1983.
9. T. Sttzle, "Local search algorithms for combinatorial problems analysis, algorithms and new applications," DISKI Dissertationen zur Kunstliken Intelligenz, Sankt Augustin, Germany, Tech. Rep., 1999.
10. B. Naderi, R. Ruiz, and M. Zandieh, "Algorithms for a realistic variant of flowshop scheduling," *Computers & Operations Research*, vol. 37, no. 2, pp. 236 – 246, 2010.
11. D. S. Yang X-S, "Engineering optimization by cuckoo search," *Int. Journal of Mathematical Modelling and Numerical Optimisation*, vol. 1, pp. 330–343, 2010.
12. A. Ouaarab, X.-S. Y. B. Ahiod, and M. Abbad, "Discrete cuckoo search algorithm for job shop scheduling problem," in *2014 IEEE International Symposium on Intelligent Control (ISIC) Part of 2014 IEEE Multi-conference on Systems and Control*, 2014, pp. 1872–1876.
13. D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

14. R. Mencía, M. R. Sierra, C. Mencía, and R. Varela, "A genetic algorithm for job-shop scheduling with operators enhanced by weak lamarckian evolution and search space narrowing," *Natural Computing*, vol. 13, no. 2, pp. 179–192, 2014.
15. E. Atashpaz-Gargari and C. Lucas, "Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, Sept 2007, pp. 4661–4667.
16. G. G. Minetti and C. Salto, "Imperialist competitive algorithm for the flowshop problem," in *Anales del XXI Congreso Argentino de Ciencias de la Computacin*, 2015, pp. 1–10.
17. P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations Research*, vol. 41, p. 157183, 1993.
18. M. Yazdani, M. Amiri, and M. Zandieh, "Flexible job-shop scheduling with parallel variable neighborhood search algorithm," *Expert Systems with Applications*, vol. 37, no. 1, p. 678687, 2010.
19. J.-Q. Li, Q.-K. Pan, P. N. Suganthan, and T. J. Chua, "A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem," *The International Journal of Advanced Manufacturing Technology*, vol. 52, no. 5, pp. 683–697, 2011.
20. J. Tang, G. Zhang, B. Lin, and B. Zhang, "A hybrid algorithm for flexible job-shop scheduling problem," *Procedia Engineering*, vol. 15, pp. 3678 – 3683, 2011.
21. L. Wang, S. Wang, Y. Xu, G. Zhou, and M. Liu, "A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem," *Computers & Industrial Engineering*, vol. 62, no. 4, pp. 917 – 926, 2012.
22. L. Wang, J. Cai, M. Li, and Z. Liu, "Flexible job shop scheduling problem using an improved ant colony optimization," *Scientific Programming*, pp. 1–11, 2017.