AN ADAPTIVE GENETIC ALGORITHM FOR SOLVING JOB-SHOP SCHEDULING PROBLEMS

Guilherme S. Ferreira

Heder S. Bernardino

ferreira.guilherme@gmail.com

heder@ice.ufjf.br

Universidade Federal de Juiz de Fora

Rua José Lourenço Kelmer, s/n - Bairro São Pedro, CEP: 36036-900, Juiz de Fora, MG, Brasil

Abstract. *Flexible Job-Shop Scheduling Problem (FJSP) is a problem of the manufacturing environment where jobs are scheduled with the objective of optimizing the key performance indicators chosen by the decision-maker. In the literature, metaheuristics are normally applied to this kind of problem due to their ability on finding right solutions. However, the selection of suitable parameters can be a hard task, as they are problem-dependent. In fact, right parameter settings are usually obtained by a trial and error process. This work proposes an adaptive genetic algorithm for solving FJSPs with identical parallel machines with the objective of reducing the makespan. The use of this type of algorithm is a trend in the evolutionary computation as they automatically choose the movement operators (such as crossover and mutation) and their respective parameters, reducing the time of setup and test of an algorithm. Also, our proposal includes a well-known heuristic to initialize the individuals, three crossover operators and two mutation operators. The proposed adaptive technique was tested using instances of different sizes, such as 10 $\times$ 10, 20 $\times$ 10, 20 $\times$ 20, 50 $\times$ 10, and 50 $\times$ 50 (jobs $\times$ stages). A genetic algorithm is considered to evaluate the quality of the obtained solutions comparatively. The results indicate that the use of adaptation improves the reliability of the search method when solving a large number of instances of FJSP.*

Keywords: *Genetic algorithms, Adaptive operator selection, Adaptive parameter control, Adaptive pursuit, Job-shop scheduling*

1 INTRODUCTION

Scheduling is the activity that selects the order of a set of tasks to be processed in a set of resources (Werner, 2011), aiming to achieve the performance goals previously chosen. It is a process performed by the decision-maker in different kinds of manufacturing or services companies. Table 1 presents some resources and tasks which can be found in organizations. Also, the tasks may have particularities, such as: release date, due date and priority level. Makespan, the sum of the weighted tardiness, and resource use are examples of objectives commonly adopted in scheduling problems.

Table 1: Example of resources and tasks for scheduling problems

Tasks	Resources
Jobs	Machines (or stages)
Trains	Railroad stations
Take-offs and landing (airport)	Runways (airport)
Construction site	Crews
Computer programs	Processing Units

In this work, the Flexible Job-Shop Scheduling Problems (FJSP) are solved here. It consists in a scheduling problem wherein the jobs can have different production routes. Flexible means that each stage is allowed to have more than one machine, so in this case, only identical machines are applied. In common schedule problems, one desires to reduce the inventory, to reduce the manufacturing time and to increase the credibility of the companies. FJSPs are one of the hardest combinatorial optimization problems, and there is no known algorithm able to solve it in polynomial time. This attribute motivated several works in different fields (Chaudhry & Khan, 2016). Thus, heuristic methods are commonly used for solving this kind of problem, such as dispatching rules and branch & bound algorithms. Therefore, the need of better scheduling propitiates the use of metaheuristics, once these earlier methods are inefficient to solve larger problems (Ripon *et al.*, 2007). The use of metaheuristics have been proposed for solving scheduling problems, and their use grew as the performance of the computers was increasing. Among them, the most used artificial intelligence method is Genetic Algorithm (GA) (Çaliş & Bulkan, 2015).

Though the use of GAs is useful for solving the scheduling problems, their setups can be hard work. This difficulty is since there are many crossover and mutation techniques, and each one has their occurrence rates (here we use parameters to refer the movement operators and their rates, to make the reading more comfortable). Moreover, the set of techniques and rates from a GA is problem dependent. According to Eiben *et al.* (1999), there are two options to change the parameters of GAs: before the run (i.e., the conventional GA), and during the run. In the last case, there are three subdivisions listed as:

1. Deterministic parameter control: GA uses a deterministic rule to change its parameters. Usually, the number of generations is used to determine a phase and make changes.

2. Adaptive parameter control: GA uses some information from the search to make changes. This feedback may be the average fitness of the population, the genotype, and so on.
3. Self-adaptive parameter control: There are some parameters incorporated into the chromosome, and the movement operators are also applied to them.

Adaptive Genetic Algorithms (AGAs) are used to work around the GA setup problem. This type of algorithm automatically reduces the time of setup and test of an algorithm since it chooses the parameters available in its framework.

Adaptive evolutionary algorithms were applied in the context of the scheduling problems. In (Pan *et al.*, 2011), an AGA for solving FJSPs was introduced. That approach uses adaptation to select the crossover, mutation and selection rates. Also, that proposal changes the action area of a chromosome that the movement operators are applied. That AGA exceeded the traditional GA, improving the solution. However, that work uses deterministic equations to adapt. The AGA proposed by Asokan *et al.* (2008) uses adaptation to select the rates for the crossover and mutation operators. The rates are increased or decreased in fixed steps, and there is a lower and an upper bound for them. Hongyan & Hong (2010) proposed a multi-objective Self-Adaptation GA, named as SAGA. That algorithm does not match the adaptive definition previously explained, here it is called adaptive. The SAGA adapts the crossover and the mutation rates through the average population fitness, the minimum population fitness, and the offspring fitness. The equation also uses the lower and upper bounds.

The work proposed by Wang & Tang (2011) used an Improved Adaptive Genetic Algorithm (IAGA) for solving Job-Shop Scheduling Problems (JSPs). It adaptively changes the crossover and mutation rates. The IAGA uses a method based on hormone modulation to adjust the rates. An adaptive selection for a Cellular GA was proposed in (Li *et al.*, 2014). The main objective of that algorithm is to avoid to get in local optimal. Then, that algorithm selects the individuals for the next generation based on the normalized fitness of the individuals. In (Nalepa *et al.*, 2015) an adaptive memetic algorithm was proposed to reduce makespan. That proposal uses high-low fit selection scheme, so the adaptation is applied to select between parameters of this method. According to the authors, the adaptation aims to balance the exploitation and exploration.

A multi-population genetic algorithm for solving JSPs in (Wang *et al.*, 2016). The adaptation is used in crossover and mutation probabilities and it evaluates these probabilities based on fitness. Then, in (Lu *et al.*, 2017) a flow-shop scheduling problem can be solved using an adaptive algorithm. Thus, the adaptation aims to select an appropriate population to improve the diversity. In (Yan & Wu, 2015) it was selected four crossover and four mutation operators from the literature. Thus, the adaptation tries to select the best operator combination, only one crossover and one mutation is allowed. That algorithm, in the credit assignment, uses a time window to keep records from the previous performance of the operators. The implanted method increases the probability of the best combination of operators in odd generations. In the even iterations, the best operators are always used. Zuo *et al.* (2016) established an adaptive multimeme algorithm. That approach has a Multi-Armed Bandit technique to select between two stochastic variations and three local search procedure. It results in six different methods since it allows one component of each type. An adaptive multi-objective genetic algorithm was proposed in (Chang *et al.*, 2007). That algorithm adapt the crossover and mutation rate with some feedback of the population, but it also uses the number of the current generation to calculate the new values for their parameters. A differential evolution proposed in (Zhang

et al., 2016) was used to reduce the makespan, so that approach develops a self-adaptive double mutation operator, which selects between two different movements already established for differential evolution. The adaptation happens according to the diversity of individual position of the population. For the classification used here, that adaptation is called adaptive, since the chromosomes are not used to represent the parameters. The approach in (Riley *et al.*, 2016) is a genetic programming for solving JSPs. That work evolves dispatching rules and also uses adaptation to guide the algorithm to concentrate on the more essentials terminals. The terminal weights are adapted based on frequency and fitness of an individual. Finally, the work proposed by Ferreira & Bernardino (2017) used the adaptive pursuit technique to set up a hybrid GA with a local search. In that work, the adaptive method selects the GA movement operators and their parameters. Also, it selects the local search rate. In addition, that algorithm was applied in an offline context.

In previous works, adaptive evolutionary algorithms were used to solve scheduling problems. As early explained, adaptive algorithms were used to work around the GAs setup problems. However, it is observed that almost all of these works are far from the established literature for adaptive algorithms. So, the AGA proposed here applies an adaptive technique previously established, the Adaptive Pursuit method, proposed by Thierens (2005). The present proposal differs from the work by Zuo *et al.* (2016) as that uses differentiate from traditional AGs where when a crossover is applied, and mutation is not implemented. Moreover, here the algorithm is applied in an online environment, which differs from the work of by Ferreira & Bernardino (2017). Also, this work proposes a discussion of two different methods of credit assignment. Hence, an AGA for solving FJSPs is proposed here to select the GA movement operators and their rate values.

The remaining sections of this work are those that follow. The scheduling problem solved here and GA are described in Sections 2.1 and 2.2, respectively. A previously established method for adaptive algorithms was used during this study, and it is described in Section 2.3. Section 4 describes how the computational experiments were performed and their results are presented in Section 4.1. The conclusion and some future works are shown in Section 5.

2 BACKGROUND

In this work, an AGA is proposed for solving FJSPs. The proposed algorithm has in its framework three crossover techniques and two mutation methods, from which the most appropriate can be selected. Also, the adaptation automatically chooses their rates.

2.1 Flexible Job-Shop Scheduling Problem

The FJSPs can be defined in different ways according to the assumptions or situation. So, the FJSP studied here has the following features:

- each job must be processed in every machine;
- each job has a technological route, and it may be different;
- operation can be defined as the processing of each job in each stage;
- each stage may have identical parallel machines;
- each operation requires the use of one machine in the stage, for an uninterrupted duration;

- once one operation starts, it runs until the end;
- each machine can perform one operation after another;
- no re-circulation is allowed;
- there is no setup times.

Defining the job as $i \in \mathcal{N}$, the stages as $j \in \mathcal{S}$ and the machine in one stage as $m \in \mathcal{M}$. The example of the input data for the problem can be observed in Table 2. In this example, the job $i = 0$, has to be processed in the following stages (5 – 3 – 2 – $\cdots$ – $\cdots$ – 7) with the respective duration (15 – 27 – 20 – $\cdots$ – $\cdots$ – 12) in time units. Each instance provides the number of identical parallel machines for each stage, which can be found in Table 3.

Table 2: An instance example for one problem with 10 jobs and 10 stages

i	j	d_{ij}	j	d_{ij}	j	d_{ij}	j	d_{ij}	j	d_{ij}
0	5	15	3	27	2	20	$\cdots$	$\cdots$	7	12
1	0	82	1	55	8	58	$\cdots$	$\cdots$	6	29
2	3	87	7	53	8	70	$\cdots$	$\cdots$	1	57
3	1	99	6	47	9	85	$\cdots$	$\cdots$	7	41
4	1	21	2	20	6	19	$\cdots$	$\cdots$	0	5
5	6	62	2	61	8	17	$\cdots$	$\cdots$	5	90
6	3	50	5	69	9	21	$\cdots$	$\cdots$	7	12
7	5	98	6	24	0	97	$\cdots$	$\cdots$	1	45
8	7	65	6	32	9	9	$\cdots$	$\cdots$	4	52
9	1	61	7	46	8	31	$\cdots$	$\cdots$	0	70

Table 3: An example of the number of machines for a problem with 10 stages

j	0	1	2	3	4	5	6	7	8	9
$\mathcal{M}_j$	3	2	3	2	2	2	1	3	3	2

In this work, a mono-objective problem is studied. The proposal aims to reduce the makespan:

$$C_{\max} = max_{i \in \mathcal{N}}\{f_i\} \ , \tag{1}$$

where f_i is the flowtime of the job i. We can observe in Fig. 1 that job 1, job 2 and job 3 finish in 12, 23 and 32 time units, respectively. So, the makespan for this job-shop environment is 32.

2.2 Genetic Algorithms

As indicated by Çaliş & Bulkan (2015), genetic algorithms are the most used Artificial Intelligence (AI) technique when solving JSPs. In that survey, GAs were used in 26,4% of the

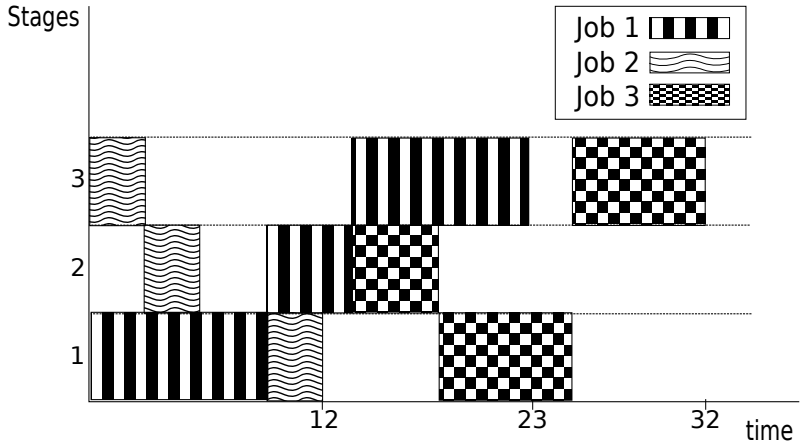


Figure 1: Gantt chart representing a solution for a job-shop problem with 3 jobs and 3 stages.

references from 1997 to 2015 in this context. GAs was proposed by Holland (1975) and it was inspired by the evolution theory. In GAs, there is a population where each individual represents one solution to the problem. These individuals undergo operations like crossover and mutation and finally are selected by an elitist scheme. A standard GA is presented in Algorithm 1.

Algorithm 1: A Standard Genetic Algorithm

```
1 INITIALIZE population;
2 EVALUATE each candidate;
3 while Termination condition is not satisfied do
4     while Number of individuals is less than population size do
5         SELECT parents;
6         CROSSOVER the pair of parents;
7         MUTATE each new generated offspring;
8         EVALUATE the new candidates;
9     SELECT individual for the next generation;
```

In the context of JSPs, different representations can be used, in this work is adopted a representation that not allow unfeasible solutions. Also, Jorapur *et al.* (2014) studied different kinds of representations as operation-based, job-based, priority rule-based, random keys, preference list based, and machine-based. The job-based representation is used here to represent the individuals, which is the one recommended by that study. Also, a permutation-code with length equals to the number of jobs is adopted. Figure 2 illustrates an individual encoded using the job-based representation. In this figure, the job 9 is the first to be executed, the job 3 is the second one, and so on, until the job 0.

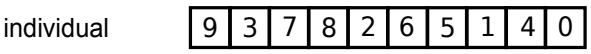


Figure 2: The job-based representation for a problem with 10 jobs.

There are specific movement operators (crossover and mutation) that can be applied to each

representation. Among those for job-based representation, the following five were chosen. They are available in Werner (2011):

- The Order-Preserving one-point crossover (OP) randomly selects a cut point. The jobs are copied from the parent 1 to the offspring until this point. The empty genes in the offspring are filled with the missing jobs in the same order that they are positioned in the parent 2.
- The Linear-Order crossover (LOX) sets up two cut points. The segment between the cut points is copied from the parent 1 to the offspring in the same position. The empty genes are completed from parent 2, only missing jobs, keeping the same order that they appear, this is, their relative position.
- Partial mapped crossover (PMX) sets up two cut points, and this segment is copied from parent 1 to the proto-offspring in the same location. The empty genes are filled from the second parent in their positions. The proto-offspring probably is infeasible. To correct that, the jobs that appear twice are mapped from parent 1 to parent 2. As demonstrated in Fig 3 the jobs 7 and 4 are duplicated in the proto-offspring. The duplicity of the job 4 is solved mapping this to the job 8 and replacing it outside the segment. The second repetition requires one step more to solve. The job 7 was mapped to 5 but replace with 5 cause another job repetition. The second step mapped the job 5 to job 2, and then a feasible offspring was created.
- Shift mutation randomly selects a gene and shift a different arbitrary position. The others jobs are shifted keeping these relative order. Figure 4 represents a shift mutation, where the job 5 is shifted to the first gene position.
- Pairwise interchange neighborhood mutation (Interchange mutation) arbitrary selects two different jobs, and they change their places on the chromosome.

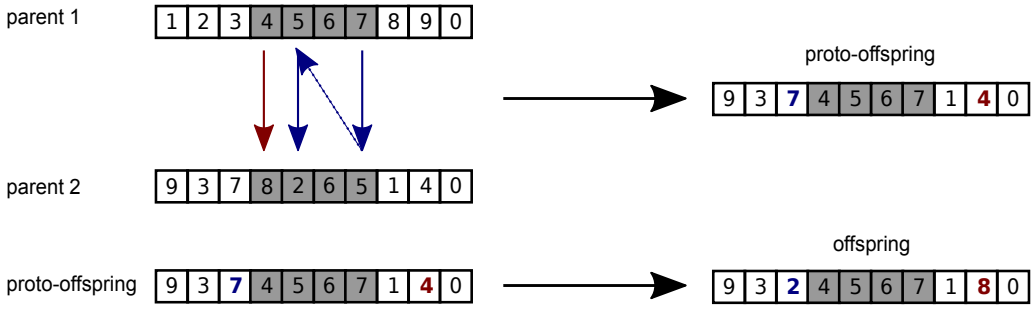


Figure 3: PMX crossover.

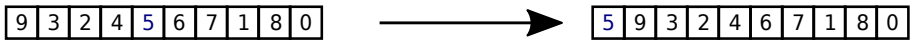


Figure 4: Shift mutation.

2.3 Adaptive Genetic Algorithms

AGAs are a variant of GAs that automatically set up their best parameters during the search. In this work, the Adaptive Pursuit (AP) algorithm introduced by Thierens (2005) is used to solve JSPs. That method increases the use of the best parameters when there are no changes in the

behavior of the problem. The increase in probability values happens faster in AP when it is compared with the probability matching method. That algorithm depends on two components: the Credit Assignment (CA), which evaluates how the parameters improve the offspring, and the Parameter Selection, which is a function that selects the operators between those available.

The CA component uses a function for approximate the quality of the new solutions generated by a given operator. Usually, the fitness improvements are used to evaluate the quality, but in some problems may be desired other information. There are different ways to evaluate the credit assignment (Aleti & Moser, 2015). According to that work, in the Instantaneous Reward (IR), CA uses the last application of a parameter to forecast its performance. In the Extreme Credit Assignment (EX), there is a time window W for each parameter. This time window keeps records of the performance of operators, so the reward is based on the best in this set (Fialho *et al.*, 2008). The following equation defines CA:

$$\mathcal{R}_a(t) = \begin{cases} \delta(t) & \text{if IR} \ , \\ \max\limits_{i=1,\ldots,W} \{\delta(t_i)\} & \text{if EX} \ , \end{cases} \tag{2}$$

where $\delta(t)$ is the fitness improvement at time t. Given the adaptation rate α as adaptation rate, the quality of an operator can be evaluated as:

$$\mathcal{Q}_a(t+1) = \mathcal{Q}_a(t) + \alpha[\mathcal{R}_a(t) - \mathcal{Q}_a(t)] \ . \tag{3}$$

The Parameter Selection uses a roulette wheel to select the operators. The AP method also suggests a lower bound, and consequently an upper bound, for the probability values. They allow for the future application of any component. Thus, a parameter value which is not good at the beginning can be selected in another stage of the search if it becomes a good choice. Defining the number of components as K, the learning rate as β, the minimum probability as $P_{\min}$, the maximum probability as $P_{\max} = [1 - (K-1)P_{\min}$, and the best operator as $a^* = max_a[\mathcal{Q}_a(t+1)]$ then the probability of the best operator is evaluated can be given by:

$$\mathcal{P}_{a^*}(t+1) = \mathcal{P}_{a^*}(t) + \beta[P_{\max} - \mathcal{P}_{a^*}(t)] \ , \tag{4}$$

and the probabilities for the other parameters as:

$$\mathcal{P}_a(t+1) = \mathcal{P}_a(t) + \beta[P_{\min} - \mathcal{P}_a(t)] \ . \tag{5}$$

2.4 NEH Heuristic

GAs are techniques that may contain other heuristic methods in their framework. In this work, it is possible to use the Nawaz, Enscore e Ham (NEH) heuristic to initialize some individuals. That method was proposed in Nawaz *et al.* (1983) and it is described in Algorithm 2.

3 THE ADAPTIVE PURSUIT GENETIC ALGORITHM

Two AGAs are proposed here for selecting the crossover and mutation operators, as well as, their application rate. These parameters are selected automatically in the search process,

Algorithm 2: NEH heuristic

```
CALCULATE, for each job, the sum of duration of all operations;
SORT the jobs in descending order of its processing time;
PICK the two first sorted jobs, then evaluate the makespan for all possible sequences;
SELECT the best makespan and always keep this relative order of those jobs;
while There are jobs not picked do
    PICK the next job sorted job and evaluate the makespan for all possible sequences;
    SELECT the best makespan and always keep this relative order of those jobs;
```

based on the fitness improvements. Both AGAs uses the AP method previously described. The difference between the proposed techniques is: the first method uses IR while the second one adopts EX.

The reward of both algorithms is calculated as: for the crossover operators, it is the difference between the fitness of the best parent and the best offspring; for the mutation operators, it is the average of the fitness values of the current population and the fitness value of the generated offspring. The quality indicators of the operators are updated after the mutation step, as soon as the new individuals are evaluated. A binary tournament is used to select the parents, and the selection scheme keeps the best individual of the generation g replacing the worse in the generation $g + 1$. The proposed algorithms may use the NEH heuristic to initialize 20% of the initial individuals. Here, AP is used to indicate the adaptive pursuit with IR while APEX represents the adaptive pursuit with EX. These techniques are described in Algorithm 3.

Algorithm 3: The Proposed Adaptive Genetic Algorithm

```
INITIALIZE population;
EVALUATE each candidate;
while Termination condition is not satisfied do
    while Number of individuals is less than population size do
        SELECT parents;
        SELECT the crossover (operator and rate);
        CROSSOVER the pair of parents;
        SELECT the mutation (operator and rate);
        MUTATE each new generated offspring;
        EVALUATE the new candidates;
        EVALUATE the used parameters;
        UPDATE the parameters probabilities;
    SELECT individual for the next generation;
```

4 COMPUTATIONAL EXPERIMENTS

Extensive experiments were conducted to evaluate the proposed algorithms, using instances generated by the algorithm described by Taillard (1993). Since the problem studied is the FJSP, the number of machines at each stage was inserted in the original instances. Only integer values

are allowed as the number of machines, and they belong to the interval [1, 3]. A total of 100 instances were generated following this method, with five different sizes, 10×10, 20×10, 20×20, 50×10 and 50×50. The proposed algorithms are compared to a standard GA. Table 4 shows all parameters tested in the GA or all parameters that the AP and APEX may select during the run. The parameters are: P is the population size, C_{R} is the crossover rate, C_{O} is the crossover operator, M_{R} is the mutation rate, and M_{O} is the mutation operator. For each combination of these parameters, the algorithms run 5 times to solve each instance. A low standard deviation was achieved with this number of runs, for the worst case the standard deviation was 5.13% of the mean.

AP and APEX were developed aiming to control the parameters available in their framework automatically. However, the AP method has two parameters α and β, previously described as adaptation rate and learning rate, respectively. The values of α and β allowed in this work are {(2,2), (2,8), (5,5), (8,2), (8,8)}. The time window W assumes the same value as the population size. As the developed adaptive methods cannot select the population size and the use of the NEH heuristic, there are 30 possible combinations to set up AP and APEX.

Table 4: Values allowed to be used for each GA, AP and APEX parameters.

Parameters	Values		
P	50	76	100
C_{R}	0.7	0.8	0.9
C_{O}	PMX crossover	LOX crossover	OP crossover
M_{R}	0.3	0.4	0.5
M_{O}	Shift Mutation	Interchange Mutation	–
NEH heuristic	Used	Not used	–

A standard GA was used to compare the results. To make a fair comparison, this algorithm, as well as the adaptive, uses the same techniques available to the adaptive algorithms, as the mechanism to select the parents, and the method used to select the individuals that will survive across the generation. Also, it may use the NEH heuristic to initialize the population. The same options presented in Table 4 are used to set up the parameters $C_{\mathrm{R}}, C_{\mathrm{O}}, M_{\mathrm{R}}$ and M_{O} from a GA. Thus, there are $3^4 \times 2^2 = 324$ options when the user is defining the GA parameters.

The stopping criterion used for all algorithms is based on the number of evaluations, and it must be less than or equal to $(800 + 10 \times \mathcal{N}) \times \mathcal{N}^2$, where $\mathcal{N}$ is the number of jobs. This stopping criterion was used by Arroyo & Pereira (2011).

After the run of all allowed possibilities, the algorithms were analyzed according to each problem size. Then five different analyses were made, and they are namely 10×10, 20×10, 20×20, 50×10 and 50×50, with respect the number of the jobs and stages. A sixth analysis receives all problems. The last analysis was realized in all problems, but it has two phases. In the first phase, the 10×10 problems are used to select the best parameter values, which are used in the search techniques for solving all problems in the second phase. This analysis is called 10-ALL and the main idea is to reduce the time spent to select right parameters.

The Performance Profiles (PP) was used to compare the results. Suggested in (Dolan & Moré, 2002), PP is a visualization scheme that makes algorithms comparison easier. Also, in (Barbosa *et al.*, 2010) demonstrated the proprieties available in PP and it has other algorithms comparison. Considering the set TA of problem instances ta_i, with $i = 1, 2, \cdots, n_{ta}$, and a set A of algorithms a_j, with $j = 1, 2, \cdots, n_a$, and $C_{ta,a} > 0$ the average makespan, the performance ratio is defined as:

$$r_{ta,a} = \frac{C_{ta,a}}{min\{C_{ta,a} : a \in A\}} \quad . \tag{6}$$

Thus, the performance profile of an algorithm a is defined as:

$$\rho_a(\tau) = \frac{1}{n_{ta}} |\{ta \in TA : r_{ta,a} \leq \tau\}|. \tag{7}$$

The $\rho_a(\tau)$ represents the performance ratio $r_{ta,a}$ of algorithm $ainA$ is inside a factor $\tau \geq 1$ of the best possible algorithm. While a set of representative problems are compared, the algorithm with larger $\rho_a(\tau)$ should be preferred.

4.1 Results

In this section, the results provided by the computational experiments are discussed. In all analysis, the best parameter settings of the techniques GA, AP and APEX were obtained. Thus, in all analyses performed here, the algorithms are compared when using their best parameter values.

Table 5 resumes the best parameter for the GA in each analysis. This table shows that the LOX crossover is an excellent choice for every problem and the shift mutation provides better results than interchange mutation. However, the rates of the movement operators vary in the results.

Table 5: Best parameters for GA for each grouping.

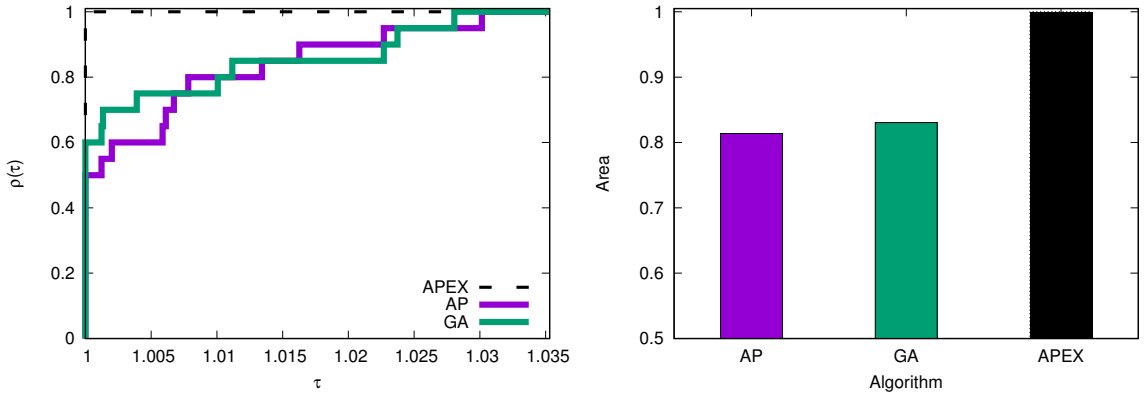
Group of Analysis	GA					
	P	C_{R}	C_{O}	M_{R}	M_{O}	NEH
10×10	100	0.7	LOX	0.4	Shift	Used
20×10	76	0.9	LOX	0.4	Interchange	Used
20×20	100	0.9	LOX	0.4	Shift	Used
50×10	100	0.8	LOX	0.3	Shift	Used
50×50	100	0.7	LOX	0.4	Shift	Used
All problems	100	0.7	LOX	0.4	Shift	Used
10–ALL	100	0.7	LOX	0.4	Shift	Used

Table 6 shows the best parameter settings for all analysis for AP and APEX. One can see that the parameters α and β in AP and APEX have different values in each analysis, and different from the GA there is no default setup.

Table 6: Best parameters for AP and APEX in each grouping.

Group of analysis	AP				APEX			
	P	α	β	NEH	P	α	β	NEH
10×10	76	2	2	Not used	50	8	2	Used
20×10	100	8	2	Used	76	5	5	Not used
20×20	100	8	8	Used	100	2	8	Used
50×10	100	2	2	Used	100	8	8	Used
50×50	50	2	8	Used	100	2	2	Used
All problems	100	2	2	Used	100	2	8	Used
10–ALL	50	8	2	Used	76	2	2	Not used

Figures 5a and 6a present the PP curves of GA, AP and APEX when solving the problems with 10×10 and 20×10 (jobs $\times$ stages), respectively. In both cases, APEX reaches better results than GA and AP. APEX was able to find the best results on most of the problems tested of sizes 10×10 and 20×10. GA and AP obtained a similar performance. Figures 5b and 6b present the are under the PP curves, which is a overall performance measure when using PP.

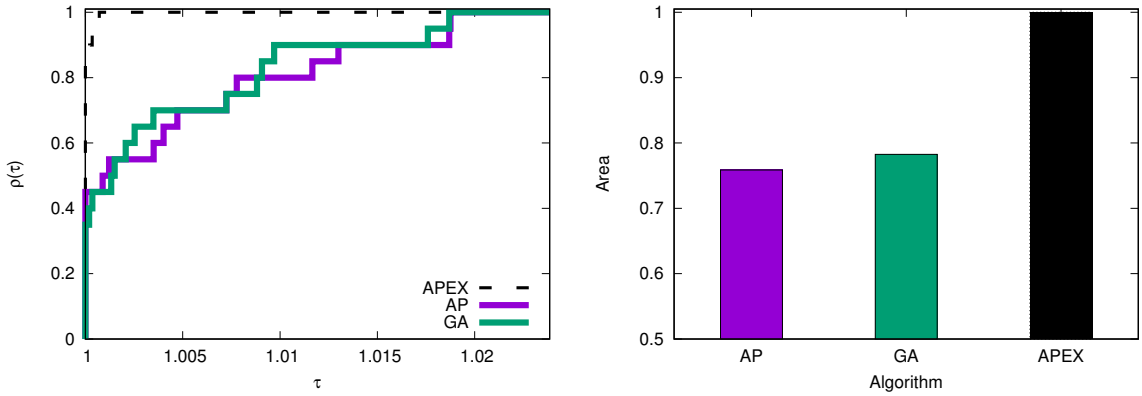


(a) PP for the three algorithms. **(b) Normalized area under curve of the PP.**

Figure 5: Comparison between the three algorithms for the 10 jobs $\times$ 10 stages problems.

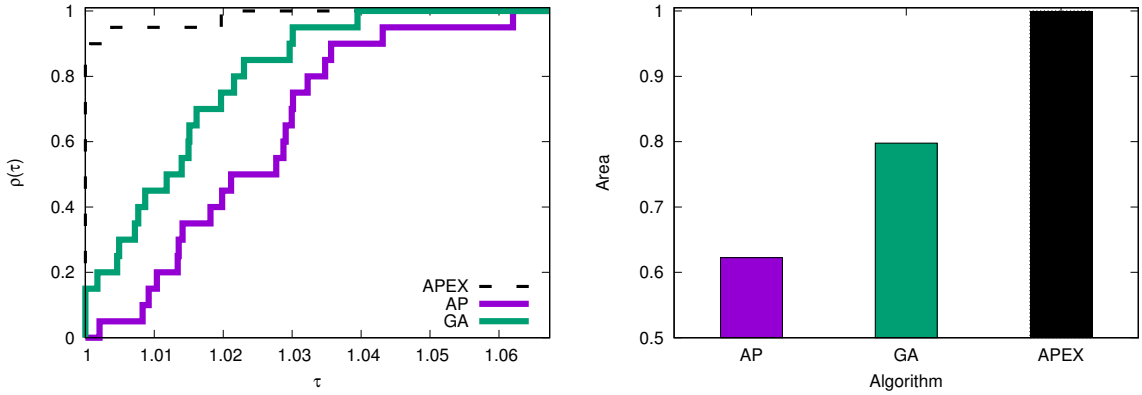
In the other analysis, the APEX is the better choice, solving all the problems with performance better than the others algorithms. However AP is not too good losing performance mainly in problems with 50 jobs. In the worse condition, the algorithm APEX solves the problems losing 2% of performance, as it is showed in Figures 7a, 8a and 9a. In Figures 7b, 8b and 9b one can see that when the size of the problem increases, the performance of APEX distinguishes even more from the other methods. The same can be observed when AP and GA are compared, where the AP is shown to be less reliable.

Testing the algorithms with the best parameters for solving all problems results in Figure 10a. The APEX solves all problems losing 2% of the performance. The AP and GA solve



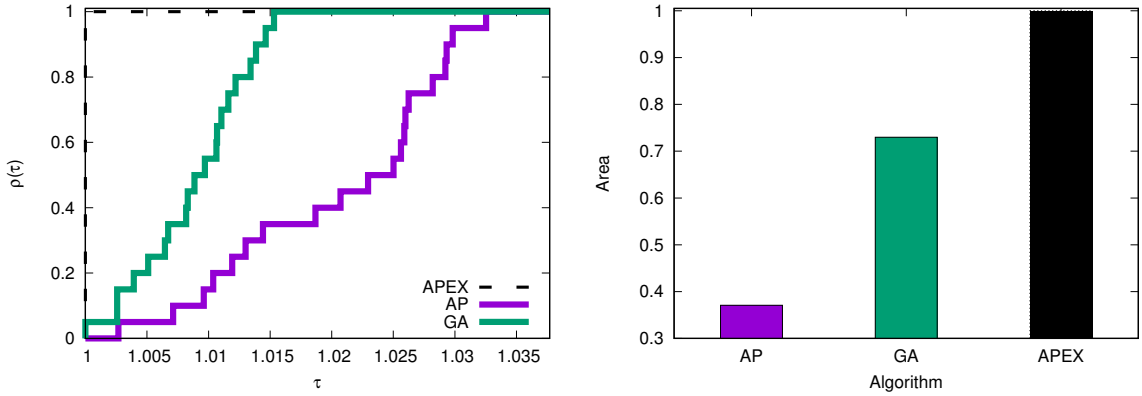
(a) PP for the three algorithms. **(b) Normalized area under curve of the PP.**

Figure 6: Comparison between the three algorithms for the 20 jobs $\times$ 10 stages problems.



(a) PP for the three algorithms. **(b) Normalized area under curve of the PP.**

Figure 7: Comparison between the three algorithms for the 20 jobs $\times$ 20 stages problems.

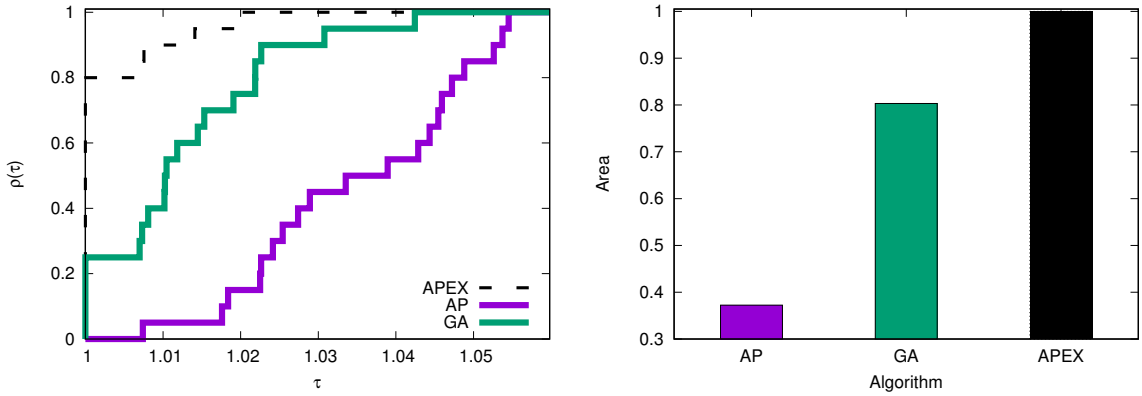


(a) PP for the three algorithms. **(b) Normalized area under curve of the PP.**

Figure 8: Comparison between the three algorithms for the 50 jobs $\times$ 10 stages problems.

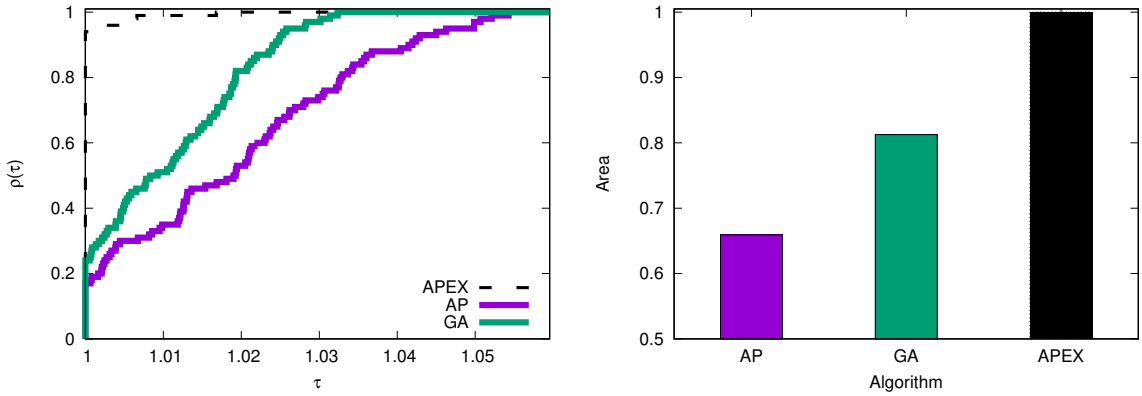
losing approximately 5% and 3% of the performance, respectively. APEX is again the best performing technique and AP the worse one, as illustrated in Figure 10b.

Finally, the last experiment uses the best parameters for solving the problems with 10 jobs



(a) PP for the three algorithms. **(b) Normalized area under curve of the PP.**

Figure 9: Comparison between the three algorithms for the 50 jobs × 50 stages problems.



(a) PP for the three algorithms. **(b) Normalized area under curve of the PP.**

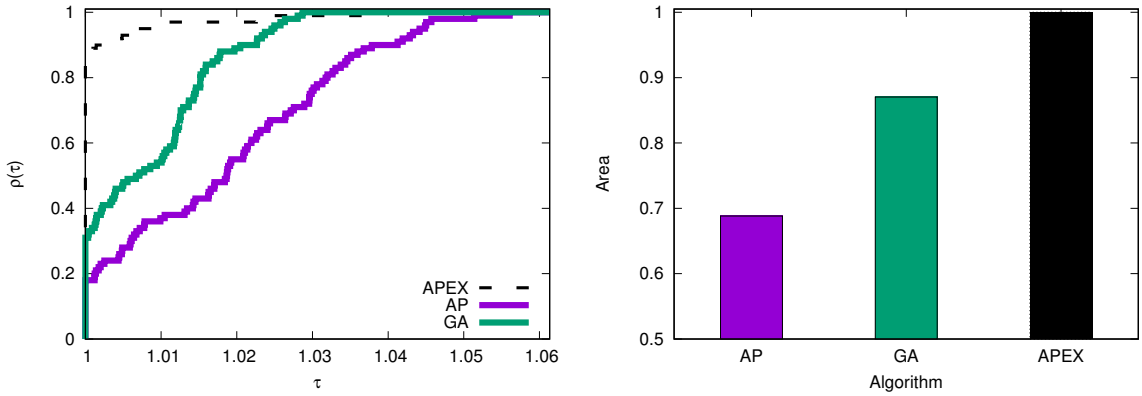
Figure 10: Comparison between the three algorithms for all problems.

and 10 stages and applies it to all problems, and it results in Figure 11a. Through the Figure 11b one can see that the APEX outperformed the others algorithms what is expected, once it can choose the correct parameters.

AP does not achieve results as good as GA. Hence, it is suggested that the use of a time window is suitable for adaptive algorithms. A reason that can justify it is that the time window reduces the effect of a bad selection during the process. For example, after the recombination, if the same individual was twice selected to be parents, the recombination results in an offspring identical to the parents, which can punish a parameter. For this reason, the instantaneous reward CA may punish a parameter strongly. Otherwise, the APEX outperformed GA due to the fact it can choose the appropriate parameter during the run. In Fig. 12 shows the APEX selecting the crossover operators for a problem with 20 jobs and 20 stages. One can see that the parameters are not stable in the entire process, and these changes can help the search for good solutions.

5 CONCLUDING REMARKS AND FUTURE WORKS

In this paper, we proposed the use of adaptive methods to select the parameters from a standard Genetic Algorithm (GA). They were applied for solving Flexible Job-Shop Scheduling Problems (FJSPs), and the problem instances were provided based on Taillard (1993). The



(a) PP for the three algorithms. **(b) Normalized area under curve of the PP.**

Figure 11: Comparison between the three algorithms for all problems, but using the best setups provided by the 10 jobs $\times$ 10 stages.

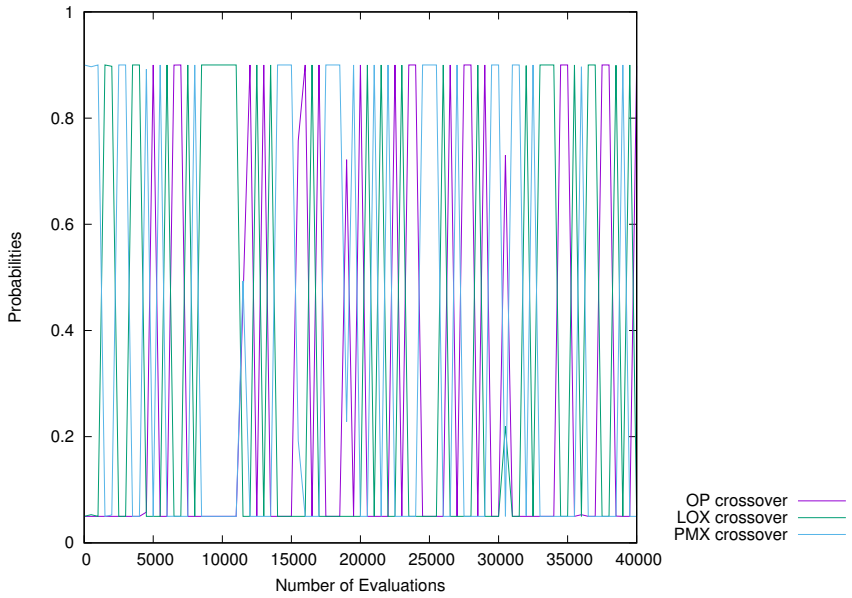


Figure 12: An example of APEX controlling the crossover operator probabilities through the search process, applied for a problem with 20 jobs and 20 stages.

Adaptive Pursuit method was applied to two different credit assignments, one with instantaneous reward and another with extreme credit assignment, where they are called here AP and APEX, respectively. Also, the reward is based on fitness improvement. Their results were compared with a standard GA with may use the same parameters available in the adaptive methods framework.

APEX outperformed GA and AP in all instances. This result suggests that the time window may avoid a parameter punishment due to a bad roulette, as in parents selection. Also, the APEX shows that the best parameters vary during the search process and it may contribute the performance of the adaptive methods. Finally, this study showed the APEX is reliable for solving FJSPs. The AP method achieves similar results compared with the best GA for solving problems of small size (10 jobs and 10 stages, and 20 jobs and 10 stages), but its relative

performance decreases when the size of the problems increases.

In future works, there are two approaches, one with the perspective of adaptive methods and the second one concerning the problem. In the first, other adaptive methods can be used to compare their performance for solving FJSPs, such as Dynamic Multi-Armed Bandit. To approximate to the industries is also an exciting research avenue. It can be done by introducing setup time or re-circulation to the FJSP model or working with multiple objectives.

ACKNOWLEDGEMENTS

The authors thank the financial support provided by Programa de Pós-Graduação em Modelagem Computacional da UFJF, CAPES, FAPEMIG and CNPq.

REFERENCES

Aleti, A. & Moser, I., 2015. A Systematic Literature Review of Adaptive Parameter Control Methods for Evolutionary Algorithms.

Arroyo, J. E. C. & Pereira, A. A. S., 2011. A GRASP heuristic for the multi-objective permutation flowshop scheduling problem. *The International Journal of Advanced Manufacturing Technology*. vol. 55, n. 5, pp. 741–753.

Asokan, P., Jerald, J., Arunachalam, S., & Page, T., 2008. Application of adaptive genetic algorithm and particle swarm optimisation in scheduling of jobs and AS/RS in FMS. *International Journal of Manufacturing Research*. vol. 3, n. 4, pp. 393–405.

Barbosa, H. J., Bernardino, H. S., & Barreto, A. M., 2010. Using performance profiles to analyze the results of the 2006 CEC constrained optimization competition. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE. pp. 1–8.

Çaliş, B. & Bulkan, S., 2015. A research survey: review of AI solution strategies of job shop scheduling problem. *J. of Intelligent Manufacturing*. vol. 26, n. 5, pp. 961–973.

Chang, P.-C., Hsieh, J.-C., & Wang, C.-Y., 2007. Adaptive multi-objective genetic algorithms for scheduling of drilling operation in printed circuit board industry. *Applied Soft Computing*. vol. 7, n. 3, pp. 800–806.

Chaudhry, I. A. & Khan, A. A., 2016. A research survey: review of flexible job shop scheduling techniques. *Int. Trans. in Operational Research*. vol. 23, n. 3, pp. 551–591.

Dolan, E. D. & Moré, J. J., 2002. Benchmarking optimization software with performance profiles. *Mathematical programming*. vol. 91, n. 2, pp. 201–213.

Eiben, A. E., Hinterding, R., & Michalewicz, Z., 1999. Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation*. vol. 3, n. 2, pp. 124–141.

Ferreira, G. S. & Bernardino, H. S., 2017. An Adaptive Pursuit Genetic Algorithm for Solving Job-Shop Scheduling Problems. In *Congresso Brasileiro de Inteligência Computacional - (submitted)*. ABRICOM, Niterói, RJ. pp. 1–12.

Fialho, A., Da Costa, L., Schoenauer, M., & Sebag, M., 2008. Extreme value based adaptive operator selection. In *International Conference on Parallel Problem Solving from Nature*. Springer. pp. 175–184.

Holland, J. H., 1975. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. The University of Michigan Press, Ann Arbor, Michigan.

Hongyan, X. & Hong, H., 2010. Research on job-shop scheduling problem based on Self-Adaptation Genetic Algorithm. In *Int. Conf. Logistics Syst. and Intell. Manage.*. vol. 2. IEEE. pp. 940–943.

Jorapur, V., Puranik, V. S., Deshpande, A. S., & Sharma, M. R., 2014. Comparative Study of Different Representations in Genetic Algorithms for Job Shop Scheduling Problem. *Journal of Software Engineering and Applications*. vol. 07, n. 07, pp. 571–580.

Li, Z. M., Zhang, Y., Zheng, X. D., & Wan, X. Y., 2014. Solving the Problem of General Job Shop Problem by Using Improved Cellular Genetic Algorithm. *Advanced Materials Research*. vol. 945-949, pp. 3130–3135.

Lu, C., Gao, L., Li, X., Pan, Q., & Wang, Q., 2017. Energy-efficient permutation flow shop scheduling problem using a hybrid multi-objective backtracking search algorithm. *Journal of Cleaner Production*. vol. 144, pp. 228–238.

Nalepa, J., Cwiek, M., & Kawulok, M., 2015. Adaptive memetic algorithm for the job shop scheduling problem. In *Int. Joint Conf. of Neural Networks (IJCNN)*. IEEE. pp. 1–8.

Nawaz, M., Enscore, E. E., & Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*. vol. 11, n. 1, pp. 91–95.

Pan, Y., Zhang, W. X., Gao, T. Y., Ma, Q.-Y., & Xue, D. J., 2011. An adaptive Genetic Algorithm for the Flexible Job-shop Scheduling Problem. In *Int. Conf. of Comput. Sci. and Automation Eng. (CSAE)*. vol. 4. IEEE. pp. 405–409.

Riley, M., Mei, Y., & Zhang, M., 2016. Improving job shop dispatching rules through terminal weighting and adaptive mutation in genetic programming. In *IEEE congress on evolutionary computation*. pp. 3362–3369.

Ripon, K. S. N., Tsang, C.-H., & Kwong, S., 2007. An Evolutionary Approach for Solving the Multi-Objective Job-Shop Scheduling Problem. In *Evolutionary Scheduling*. vol. 49. Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 165–195.

Taillard, E., 1993. Benchmarks for basic scheduling problems. *european journal of operational research*. vol. 64, n. 2, pp. 278–285.

Thierens, D., 2005. An adaptive pursuit strategy for allocating operator probabilities. In *Genetic and Evolutionary Computation Conference*. ACM. pp. 1539–1546.

Wang, L., Cai, J.-C., & Li, M., 2016. An adaptive multi-population genetic algorithm for job-shop scheduling problem. *Advances in Manufacturing*. vol. 4, n. 2, pp. 142–149.

Wang, L. & Tang, D., 2011. An improved adaptive genetic algorithm based on hormone modu-

lation mechanism for job-shop scheduling problem. *Expert Systems with Applications*. vol. 38, n. 6, pp. 7243–7250.

Werner, F., 2011. Genetic algorithms for shop scheduling problems: a survey. *Preprint*. vol. 11, p. 31.

Yan, J. & Wu, X., 2015. A genetic based hyper-heuristic algorithm for the job shop scheduling problem. In *Int. Conf. Intell. Human-Machine Syst. and Cybern. (IHMSC)*. vol. 1. IEEE. pp. 161–164.

Zhang, H., Yan, Q., Guohui, Z., & Jian, Z., 2016. A Chaotic Differential Evolution Algorithm for Flexible Job Shop Scheduling. In *Theory, Methodology, Tools and Applications for Modeling and Simulation of Complex Systems*. pp. 79–88.

Zuo, Y., Gong, M., & Jiao, L., 2016. Adaptive multimeme algorithm for flexible job shop scheduling problem. *Natural Computing*, pp. 1–22.