




Transgenic Genetic Algorithm to Minimize the Makespan in the Job Shop Scheduling Problem

Monique Simplicio Viana¹^a, Orides Morandin Junior¹^b and Rodrigo Colnago Contreras²^c

¹Department of Computing, Federal University of São Carlos, Rod. Washington Luiz KM 235, São Carlos – SP, Brazil

²Department of Computer Sciences, University of São Paulo, Av. Trabalhador São-carlense 400, São Carlos – SP, Brazil

Keywords: Job Shop Scheduling Problem, Genetic Algorithm, Transgenic Operator, Combinatorial Optimization.

Abstract: In recent years, several research studies have been conducted that use metaheuristics to calculate approximations of solutions for solving NP-Hard problems, within this class of problems there is the Job Shop Scheduling Problem (JSSP), which is discussed in this study. Improved solutions to problems of this type have been created for metaheuristics in the form of additional operators. For the Genetic Algorithm (GA) the transgenic operator has recently been created, whose operation is based on the idea of "genetically modified organisms", with the proposal to direct some population of individuals to a more favorable solution to the problem without removing the diversity of the population with a competitive cost of time. In this study, our main contribution is an adaptation of the GA with transgenic operator to the JSSP. The results obtained by the proposed method were compared with three papers in the literature that work on the same benchmark: one using GA, one using Adaptive GA and another using Ant Colony Optimization. The results confirm that the GA used with the transgenic operator obtains better results in a competitive processing time in comparison to the other techniques, due to its better targeting in the search space.

1 INTRODUCTION

The job shop scheduling problem (JSSP) is a combinatorial optimization problem defined in the literature as in the NP-Hard class (Lu et al., 2018a). Therefore, it is recommended the use of heuristic, metaheuristic and stochastic algorithms to optimize NP Hard class problems (Hasan et al., 2010).


The JSSP is part of a class of problems among the job-based scheduling problems. This class represents a research area of great importance in current studies, such as flexible job scheduling problems (FJSPs), parallel machine scheduling problems (PMSPs), test task scheduling problems (TTSPs) and others (Lu et al., 2018a). Specifically, in this paper, we approach the class of combinatorial optimization problems known as JSSP. In the following paragraphs, some recent works are presented from a vast literature on the use of metaheuristics in job-based problems.


Nguyen et al. (2018) proposed a study of the dynamic flexible job shop scheduling problem with a


new genetic programming algorithm (GP), entitled adaptive charting GP (ACGP), the proposed algorithm. The ACGP can balance its exploration, getting exploitation better than the existing surrogate-assisted algorithm. The proposal performed better than standard genetic programming algorithm.

Romero et al. (2018) proposed a study of the flexible job shop scheduling problem (FJSSP) with Lot Streaming with the Tabu Search (TS) algorithm, the study was compared with a mathematical programming solver, GUROBI. The algorithm obtained better results surpassing the upper limits found of GUROBI.

Öztop et al. (2018) proposed a study of the hybrid flow shop scheduling problem (HFSP) using the Iterated greedy algorithms, IG and IGALL. The objective variable was to minimize total flow time and has been tested in HFSP instances from the literature. The authors emphasize that one of the main contributions of the study was that the results of flow

^a <https://orcid.org/0000-0002-2960-8293>

^b <https://orcid.org/0000-0001-5588-100X>

^c <https://orcid.org/0000-0003-4003-7791>

time criterion have been reported for the HFSP benchmark suite for the first time.

Dao et al. (2018) proposed a study of the JSSP with an algorithm based on parallel versions of the bat algorithm (BA), using as objective variable makespan. The algorithm presented better convergence and competitive results than BA traditional and particle swarm optimization.

Morandin, et al. (2008a) proposed a study to solve the Production Scheduling of Manufacturing Systems, the study has been tested in a benchmark of the type JSSP with a traditional genetic algorithm (GA) as a search method, using as objective variable the makespan measure. The GA achieved competitive results in a shorter processing time.

Morandin et al. (2008b) added an improvement in the GA proposed in Morandin et al. (2008a), adaptive rules were included to the algorithm, entitled adaptive genetic algorithm (AGA). The crossover and mutation rate are dynamically adjusted according to the individual's fitness value. The study has been tested in a benchmark of the type JSSP. The AGA presented solutions with response time acceptable.

Kato et al. (2010) proposed a study to solve the Production Scheduling of Manufacturing Systems. The study has been tested in a benchmark proposed in Morandin et al. (2008b). The authors use a Max-Min Ant System algorithm as a search method. The proposal was compared with Morandin et al. (2008b) and obtained quality solutions in a shorter time.

In the literature, it is possible to find several recent studies belonging to the job-based scheduling problems class that approach the task with AG (Asadzadeh, 2015; Kundakci and Kulak, 2016; Kurdi, 2016; Lu et al., 2018b; Peng et al., 2018; Hosseinabadi et al., 2019).

The GA is a metaheuristic widely used in current studies due to several advantages that this algorithm has, but it also has some disadvantages such as not solve complex problems easily (Guo et al., 2010). The GA has as one of its main disadvantages the high consumption of resources, that is, domain of large solutions will use longer search time (Kazemi et al., 2012; Nie et al., 2013).

Amaral and Hruschka (2014) have developed an operator for evolutionary algorithms entitled Transgenic Operator. This operator was inspired by genetic engineering, in which there is the possibility of manipulating the genetic material of individuals by adding features that are believed to be important. This type of approach can be understood as a strategy of elitism focused on specific genes. The Transgenic Operator must direct a portion of individuals of the

population for a better solution, without loss of diversity in population and in a smaller cost of time.

The objective of this work is the application of an alternative version of Transgenic Operator (Amaral and Hruschka, 2014) in the job shop scheduling problem. In this paper, we approached the reasoning proposal used by Morandin et al. (2008a), Morandin et al. (2008b), Kato et al. (2010), which uses the same benchmark for the job shop problem and are solved, respectively, by the metaheuristics GA, AGA and Ant Colony Optimization (ACO). In this way, the comparison of the results of our method with the methods of such studies becomes more natural, since we will use the same benchmark.

The remainder of this paper is organized as follows. Section 2 contains the JSSP specification and the fundamentals of GA and description of the Transgenic Operator. The components of the proposed algorithm are presented in Section 3. The computational experiments and analyses of the obtained results are presented in Section 4. Finally, Section 5 presents the conclusions of the paper.

2 PROBLEM DESCRIPTION

2.1 Job Shop Scheduling Problem

In this context, in a manufacturing system, there is a set of n jobs $\{P_1, P_2, \dots, P_n\}$ that are produced by manufacturing, and such products make shared use of a set of m machines $\{M_1, M_2, \dots, M_m\}$. A job contains a set of operations and a predetermined sequence of machines. Each operation makes use of one of the machines for a predetermined time interval to complete a job. A schedule can be defined as the assignment of operations, established by a sequence of elements in the set $O = \{O_1, O_2, \dots, O_{n \cdot m}\}$, which determine the priority order in which a job should be processed on a machine.

For each possible operations sequence O , we associate the value $T_i(O)$, which is the time it takes the job P_i to be processed by all machines in the respective script, which consists of the production itinerary detailed in this same section, and thus be considered finished according to the defined sequence of operations at O .

Thus, the makespan value of a sequence O can be defined as being the time taken for finish the production of all the jobs, as described in the Equation (1):

$$MKS = \max_i T_i(O). \quad (1)$$

Dao et al. (2018) do a similar and rigorous modeling of the makespan measurement in their work. Moreover, in this work, for each job P_i , a set \mathcal{R}_i of n_i possible scripts is considered, as defined in Equation (2), which increases the complexity of the task of minimizing the makespan value in the space of operations. A script of a job determines by which machines it should be processed and the order in which it must occur for it to be considered finished. Studies such as those from Morandin et al. (2008a), Morandin et al. (2008b) and Kato et al. (2010) are characterized by this specification.

$$\mathcal{R}_i = \{R_{i,1}, R_{i,2}, \dots, R_{i,n_i}\}. \quad (2)$$

2.2 Search with Genetic Algorithm

Genetic Algorithms (GAs) were developed in the 1970s by Holland (Holland, 1975) with the objective of optimizing complex and non-linear systems. This type of technique has a strong appeal to biological inspiration derived from the theory of evolution to perform its operation, so that its use does not require very elaborate mathematical theories.

Many improvements have been implemented over the last few years (Antonio and Coello, 2017) to the GAs that Holland formulated in his initial work (Holland, 1975) and presented in the classic 1992 book (Holland, 1992). However, all the improved GAs present in the specialized literature maintains as the main sequence of steps the one originally presented by Holland (1992) in Algorithm 1.

Algorithm 1: A Genetic Algorithm Pseudocode.

-
- (1) $it = 0$;
 - (2) Generate initial population:
 $\Omega_0 = \{C_1, C_2, \dots, C_n\}$
 - (3) Evaluate the fitness of the initial population
 - (4) Repeat
 - (5) Select individuals for crossover
 - (6) Apply crossover operator
 - (7) Apply mutation operator
 - (8) Evaluate new individuals
 - (9) Generate a new population: Ω_{it+1}
 - (10) $it = it + 1$;
 - (11) Until Termination criterion is satisfied
-

The generation of a new population, as done in step (9) of Algorithm 1, generally takes into account some own insertion technique so that only the best individuals from the iteration are maintained and do not change the size of the population.

2.3 Transgenic Operator

In order to simulate the biological advances of genetic engineering, Amaral and Hruschka (2014) proposed the use of transgenic technique in GA. The concept of transgenics is to transfer, from one generation to another, genes that probably describe a good feature. For example, vitamin supplementation of maize is used in developing countries to avoid that the population, usually with food habits based mainly on cereals, suffers from lack of vitamins. This supplementation can be done with the use of transgenic planting (Naqvi et al., 2009). The addition of this concept to GA occurs in the form of an operator, as represented in Algorithm 2.

Algorithm 2: A Transgenic Genetic Algorithm Pseudo-Code.

-
- (1) $it = 0$;
 - (2) Generate initial population:
 $\Omega_0 = \{C_1, C_2, \dots, C_n\}$
 - (3) Evaluate the fitness of the initial population
 - (4) Repeat
 - (5) Select individuals for crossover
 - (6) Apply crossover operator
 - (7) Apply mutation operator
 - (8) Evaluate new individuals
 - (9) Generate a new population: $\Omega_{it+\frac{1}{2}}$
 - (10) Apply transgenic operator
 - (11) Evaluate modified individuals
 - (12) Generate modified population: Ω_{it+1}
 - (13) $it = it + 1$;
 - (14) Until Termination criterion is satisfied
-

In Algorithm 2, two inserts of individuals are carried out: one in the step (9) and another in the step (12). However, only the population Ω_{it+1} is maintained in the process, since the population $\Omega_{it+\frac{1}{2}}$ is an intermediate population, from which the transgenic individuals are made in step (10). In this way, the population Ω_{it+1} is formed by the individuals of $\Omega_{it+\frac{1}{2}}$ together with the transgenic ones.

The transgenic operator codification is described in detail in the next section.

3 THE PROPOSED GENETIC ALGORITHM MODEL FOR JSSP

3.1 Chromosome Codification

In this study, we will follow Morandin et al. (2008a) approach to treating a subproblem within the JSSP, which is to get the best sequence of products that must be obeyed as a priority list when starting product processing on each of the machines in their respective script. Thus, the metaheuristic developed here should not obtain an "optimal sequence of operations", but rather an "optimal sequence of n jobs".

Thus, the chromosome (C) of the proposed method is formulated according to Equation (3):

$$C = ([P_{i_1}, R_{i_1, j_1}], [P_{i_2}, R_{i_2, j_2}], \dots, [P_{i_n}, R_{i_n, j_n}]), \quad (3)$$

in which, $i_k \in \{1, 2, \dots, n\}$, $i_{k-1} \neq i_k$ and $R_{i_k, j_k} \in \mathcal{R}_{i_k}$, $\forall k \in \{1, 2, \dots, n\}$.

In this way, the problem chromosome is formed by the genes $g_i = [P_i, R_{i, j(i)}]$ and hence C represents a product processing order in JSSP.

Figure 1 shows an example of chromosome and some scripts for this modeling applied to a 3×3 JSSP.

$C =$		
$P_1, R_{1,1}$	$P_2, R_{2,1}$	$P_3, R_{3,2}$
$R_{1,1} = (M_1, M_2, M_3)$	$R_{3,2} = (M_2, M_1)$	
$R_{2,1} = (M_1, M_3)$		

Figure 1: Example of chromosome and scripts.

Thus, in the example shown in Figure 1, the list of operations that the processing of jobs P_1 , P_2 and P_3 must obey, following scripts $R_{1,1}$, $R_{2,1}$ and $R_{3,2}$ respectively and according to the priority sequence defined on chromosome C , would be the next:

- 1) P_1 processing starts at M_1 ;
- 2) P_2 processing starts at M_1 as soon as it is vacant;
- 3) P_3 processing starts at M_2 ;
- 4) P_1 processing starts at M_2 as soon as it is vacant;
- 5) P_2 processing at M_3 begins;
- 6) P_3 processing starts at M_1 as soon as it is vacant;
- 7) P_1 processing starts at M_3 as soon as it is vacant.

It is noteworthy that, by construction, the feasibility of this modeling is maintained, as presented by Morandin et al. (2008a).

3.2 Fitness Function

The objective function of this work is the time taken to process the products of the JSSP, according to the configuration given in the input chromosome. Thus, the definition of this function is given in Equation (4):

$$\text{fit}(C) = \text{MKS}, \quad (4)$$

in which, MKS is the makespan value of configuration C , defined in Equation (1).

Thus, the goal of the GA developed here is to find the C configuration that has the lowest possible makespan value.

3.3 Transgenic Operator Codification

3.3.1 Transgenic Operator

Let us suppose at this stage of the study that a set of genes that we know carry "good features" is given, since in this subsection we describe how the transgenic operator works given a set of better genes. In the next subsection we describe a most relevant gene selection technique that we use in this paper to define which genes should be transferred in the transgenic operator.

We propose, in a preliminary way, that the genes set for use in the transgenic operator are the index genes $\mathcal{K} = \{K_1, K_2, \dots, K_{N_{\text{Trans}}}\}$, so that the elements of \mathcal{K} are N_{Trans} index of genes $g_i = [P_i, R_{i, j}]$ on the set $\{1, 2, \dots, n\}$, where n is the number of genes in a chromosome and N_{Trans} is the number of genes to be transferred in the operator.

In order to control the reduction of population diversity, we propose the use of $N_{\text{Trans}} \leq \sqrt{n}$, since if genes are replicated in large quantities, transgenic individuals may present endogenous phenomena. In this work, we take $N_{\text{Trans}} = \text{ceil}(\sqrt{n})$.

In order for the concept of transgenesis to be maintained, we propose to transfer genes from a model individual, which is the individual with the best fitness, to the worst individuals. That is, in each generation t , we take the best individual c^* and transfer its genes, whose indices belong to \mathcal{K} , to the n_{Trans} worst individuals of the same generation.

Thus, the transgenic individual will have the index genes belonging to \mathcal{K} in the same coordinates (positions) in which they are arranged in the best individual. And the remaining genes will occupy the remaining coordinates respecting the order in which they were in the individual before suffering transgenics. An operation scheme of the transgenic

operator is shown in Figure 2, given a set of indices \mathcal{K} , a subject c^* considered as an individual with good features, and an individual c that will receive transgenic genes, it is possible to construct a transgenic individual \hat{c} .

Thus, the transgenic operator is modelled according to the function $\text{trans}_{c^*}(\cdot)$ presented in Equation (5).

$$\begin{aligned} \text{trans}_{c^*}(C) &= \text{trans}_{c^*}([P_{i_1}, R_{i_1, j_1}], \dots, [P_{i_n}, R_{i_n, j_n}]) \\ &= \text{trans}_{c^*}(g_{i_1}, \dots, g_{i_n}) \\ &= (g_{i_1}, \dots, g_{K_1}^*, \dots, g_{K_2}^*, \dots, g_{K_{N_{\text{Trans}}}}^*, \dots, g_{i_n}), \end{aligned} \quad (5)$$

in which, $c^* = (g_{i_1}^*, g_{i_2}^*, \dots, g_{i_n}^*)$ is the best individual in the current generation.

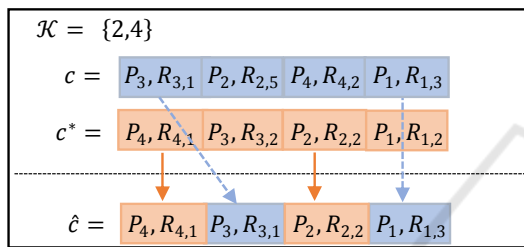


Figure 2: Transgenic operator application example.

In this way, the transgenic operator is defined in detail in Algorithm 3 below.

Algorithm 3: Transgenic operator.

Input:	n_{Trans}	Number of individuals to get transgenics
	Ω_t	Current population
	\mathcal{K}	Index of genes to be transferred
(1)	$c^* :=$ Individual with best fitness on Ω_t	
(2)	$\{w_1, w_2, \dots, w_{n_{\text{Trans}}}\} :=$ The n_{Trans} worst individuals in Ω_t	
(3)	For $i = 1$ to n_{Trans} do:	
(4)	$\hat{w}_i := \text{trans}_{c^*}(w_i)$	
(5)	End	
Output:	$\{\hat{w}_1, \hat{w}_2, \dots, \hat{w}_{n_{\text{Trans}}}\}$	Transgenic individuals

It is worth noting that the chromosomes generated by the transgenic operator have their feasibility guaranteed by Equation (5), that is, the operator is also feasible by construction, since the genes (jobs) chosen to be transferred by transgene (set \mathcal{K}) will already have their guaranteed position in the chromosomes generated, which is the position established by the best individual in the population, while the genes that do not belong to the set \mathcal{K} will

receive the position in which they present themselves in the worst individuals.

In the next subsection, a method for determining the most significant genes is presented. In other words, we have established how to construct the set \mathcal{K} so that the indices of the genes selected to be transferred by the transgenic operator transfer good features to the affected individuals.

3.3.2 Method to Find More Significant Genes

A necessary step to perform the Genetic Algorithm with Transgenic Operator is to determine which genes are the most significant to the problem, ie, which genes will be transmitted from the best individual to other individuals in the population in order to direct the worst individuals to the best solutions.

In this work we propose a method to determine the most significant genes, this method uses as a principle of weighted average and simulation, the latter being the concept of applying repeatedly the metaheuristic developed in the JSSP scenario we wish to solve before actually applying the algorithm to a given problem.

Specifically, the elaborated method simulates the behavior of GA with Transgenic operator, in which each gene individually must be simulated as the most significant through the Transgenic operator and such simulations are performed over N_G generations. In each simulation, the difference between the fitness value of an individual prior to the use of the Transgenic operator and after the use of the Transgenic operator is saved over this generations, so that it is possible to evaluate some improvements in individuals by applying certain genes in the Transgenic stage.

In each simulation performed, the improvement of the modified individuals is stored during the N_G generations of the Transgenics process. Improvement is measured by the difference between an individual's fitness value before using Transgenics and after using Transgenics. Thus, in a simulation that the gene $g_i = [P_i, R_{i,j}]$ is the most significant, a vector $v_i \in \mathbb{R}^{N_G \times 1}$ storing the average improvement between the individual fitness values before and after transgenic application is created. At the end of all generations of the fixed gene simulation, the difference matrix G is defined, as shown in Equation (6) below.

$$G = \begin{bmatrix} | & | & | \\ v_1 & \dots & v_n \\ | & | & | \end{bmatrix}. \quad (6)$$

After the simulation and generation step of the improvement matrix G , a weighted average is used to calculate which genes are most significant. For each column v_i the measures of mean μ , standard deviation σ and largest difference λ are calculated. These measurements are included in the calculation of the weighted average W , given by Equation (7), which is a vector score containing a coordinate for each gene in the chromosome. The higher the score of a gene in W , the more significant we consider it.

$$W(g_i) = \frac{\sigma(v_i) + 2 \cdot \mu(v_i) + 3 \cdot \lambda(v_i)}{6}. \quad (7)$$

All measures used in the weighted average calculation were chosen for certain purposes. Standard deviation σ was applied to verify the convergence speed of the possible improvements obtained. Mean μ was applied to assess whether a given gene influences a high or low improvement over all generations. Largest difference λ was applied by determining which genes contribute to great instantaneous improvement, representing the influence of a given gene on a generation. The weight values used for each statistical property used in the Weighted Average Measurement presented were obtained through evaluations of a grid search.

3.4 Basic Operators

The genetic algorithm we developed is based on the genetic algorithm of Morandin et al. (2008a) used in the resolution of JSSPs of size 9×9 and the main contribution of this work is the adaptation of the transgenic operator for use in JSSP and the evaluation of this in larger problems. In this way, all the standard operators of our GA, such as crossover and mutation, are the same operators described by Morandin et al. (2008a). For the selection operator, we use the technique of roulette wheel and for insertion operator we use elitism.

4 RESULTS AND DISCUSSION

4.1 Experimental Settings and Benchmark Specification

As this work was developed using a modeling similar to the one used by Morandin et al. (2008a), Morandin et al. (2008b) and Kato et al. (2010), so the evaluations are performed on a similar benchmark used by the authors of the referred works, which

consists of specialized scenarios for the used modeling. In this way, the scenarios evaluated are of similar or more complex dimensions to the dimensions of the most commonly used state-of-the-art base instance configurations, such as Lawrence (1984), which were not used in this paper because it has only one script per job and fixed size, thus presenting great distinction to the experiments performed in the works used for comparison. However, the scenarios evaluated in this experiment have dimensions equivalent to those in Lawrence's bases.

The method described in this paper was evaluated in a specific job shop scheduling problem of size 9×9 , which consists of a problem of $n = 9$ jobs and $m = 9$ machines, which was found and detailed by Kato et al. (2009). Two more complex JSSP scenarios were generated to test the scope of the method and its direction in the search space. A specific job shop scheduling problem of size 20×8 and a job shop scheduling problem of size 100×40 , which were generated following the same rules as the 9×9 scenario.

Specifically, the times at which jobs take to be processed by machines were randomly generated following a uniform distribution within the time range [400,500]. For each product, 2 to 5 scripts were randomly generated, containing 5 to 7 machines per script, as detailed in Table 1. Each of these scenarios was generated before all evaluations were performed. Thus, the metaheuristics compared here acted on the same JSSP scenarios, so that the tests were as honest as possible.

Table 1: Benchmark specification.

	Scenario 1	Scenario 2	Scenario 3
Jobs	9	20	100
Machines	9	8	40
Production time	[400,500]	[400,500]	[400,500]
Scripts per job	2	{2,3,4,5}	{2,3,4,5}
Machines per script	{5,6,7}	{5,6,7}	{5,6,7}

The number of evaluations was set to 35 for each technique in order to use the non-parametric Wilcoxon tests (Veldhuizen and Lamont, 2000) to determine if our method presents competitive results to the compared metaheuristics. In addition, the statistical properties presented by the tests can be viewed in box plots visualizations.

All the algorithms tested were coded in Matlab software. All tests were run on a notebook with i7 processor and 16GB RAM.

4.2 Results of the Proposed Algorithm and Comparisons with Other Works

In order to compare the efficiency of the proposed method (GA-Trans), we implemented some metaheuristics already successfully used in JSSP: GA (Morandin et al., 2008a); Adaptive GA (AGA) (Morandin et al., 2008b); and Ant Colony Optimization (ACO) (Kato et al., 2010).

We try to follow as closely as possible the settings presented in each work for honest results. However, the configurations of our method are more similar to GA and AGA metaheuristics configurations, which makes sense, since these algorithms differ only in the use of specific operators. Thus, the configurations of the GA, AGA and GA-Trans techniques are presented in Table 2 and the configurations used by the ACO are set out in Table 3.

Table 2: State-of-the-art configuration on GAs.

	GA	AGA	GA-Trans
Number of chromosomes	30	30	30
Crossover Rate	0.8	0.8	0.8
Mutation Rate	0.05	0.05	0.05
Iterations	200	200	200
n_{Trans}	-	-	12
Stopping Criterion	Iterations	Iterations	Iterations

Table 3: State-of-the-art configuration on ACO.

Number of Ants	50
α	1
β	2
T_{max}	10
T_{min}	0.25
Evaporation	0.02
Iterations	75
Stopping Criterion	Iterations

In addition, we apply the methodology described in section 3.3.2 to determine which genes should be used (\mathcal{K}) in the transgenic operator. The genes obtained are shown in Table 4.

Table 4: More significant genes.

Scenario	Genes
9×9	[2,4,7]
20×8	[1,4,10,13,18]
100×40	[5,21,33,49,53,68,72,80,93,100]

4.2.1 Results Comparison

Scenario 9×9 . The first scenario evaluated is the used by Morandin et al. (2008a), Morandin et al. (2008b), Kato et al. (2009) and Kato et al. (2010). The 35 tests are show in Table 5. So, the maximum value obtained by each technique is colored red and the minimum value is colored blue.

Table 5: Results of 35 Tests.

	ACO	GA	AGA	GA-Trans
1	4632	4698	5015	4670
2	4669	4936	4677	4944
3	4977	4752	4673	4632
4	4945	4956	4640	4635
5	4929	4694	4741	4691
6	4872	4996	4718	4929
7	4746	4917	4944	4848
8	4754	4981	4688	4632
9	4693	4954	4989	4688
10	4901	5051	4632	4848
11	4968	4848	4659	4635
12	4736	4991	4934	5019
13	4688	4705	4921	4688
14	4956	4718	4956	4635
15	4895	4988	4925	4929
16	4734	4725	4945	4635
17	4788	5042	4880	4860
18	4782	4945	4848	4632
19	4704	4934	4956	4688
20	4899	5068	4945	4677
21	4752	4891	4848	5051
22	4785	4669	4951	4670
23	4752	4951	4919	4776
24	4929	4958	5006	4656
25	4860	4848	4968	4691
26	4763	4710	4693	4718
27	4688	4925	4705	4670
28	4642	4898	4984	4979
29	4688	4848	4759	4635
30	4909	4945	4944	4635
31	4693	4848	4735	4670
32	4692	4679	4635	4654
33	4898	5168	4929	4929
34	4646	4705	4956	4635
35	4946	4901	4632	4677

Analyzing Table 5 and Table 6, it can be concluded that the proposed technique presented, on

average, makespan values that are smaller than the other techniques. In addition, GA-Trans presented the smallest makespan (4632) when considering the 35 evaluations conducted. In addition, GA-Trans presents as the most often occurring value the value 4635, which is a makespan value that is less than the minimum value presented by the GA and a value that is very close to the minimum makespan presented by the other methods.

Table 6: Statistical Measures.

	ACO	GA	AGA	GA-Trans
Mean	4797.4	4881.2	4838.5	4741.7
Standard Deviation	110.3	130.1	134.3	131.3
Minimum Value	4632	4669	4632	4632
Maximum Value	4977	5168	5015	5051
Mode	4688	4848	4956	4635
Average of time (s)	6.62	1.12	1.66	1.53

With respect to the average time of the GA-Trans, it can be affirmed that it is very competitive to GA-like methods and usually takes only 23,11% of the time spent by the ACO, as can be observed in more detail in Figure 3.

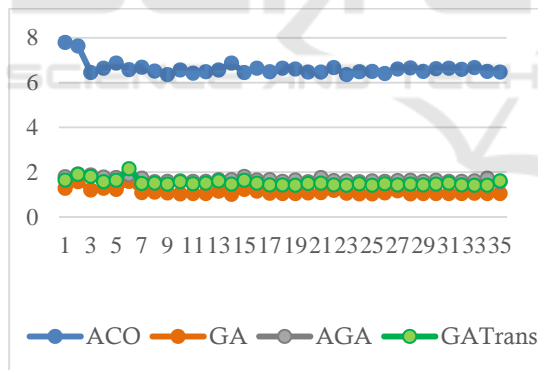


Figure 3: Time taken in 35 tests for each technique.

Although GA-Trans does not present the smallest worst makespan, it can be observed in the box plot of Figure 4 that it is a discrepant value of the technique. In fact, we can see that GA-Trans is the method that usually presents the best results in comparison with the other techniques.

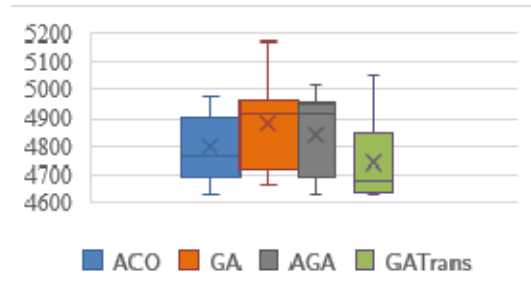


Figure 4: Box plot of the methods' results.

Convergence graphs were constructed of the proposed algorithm and the three algorithms tested, to evaluate the evolution of the entire population of the algorithm when iterations of the method are advanced. Specifically, all methods were used to generate a solution, and all individuals involved in the process were evaluated. In this way, graphically, the y-axis shows the average of all of the makespan values reached by the individuals (chromosomes or ants) of a population along the iterations of each technique, and the x-axis shows its number of iterations (generation). The purpose of these graphs is to demonstrate how fast or slow the algorithm is in finding an optimal solution. As seen in the convergence graphs of the algorithm, the GA with the transgenic operator shown in Figure 5, which is being directed through the application of transgenics of the most significant genes, has a convergence that requires fewer iterations if compared to a simple GA, to an Adaptive GA or to an ACO. With this finding, we note the advantage in a faster convergence that a GA with a transgenic operator can offer. Furthermore, according to the graph, there is no consensus among ACO ants in the observed evaluation, since the ants that find the minimum makespan do not significantly change the mean of the whole population.

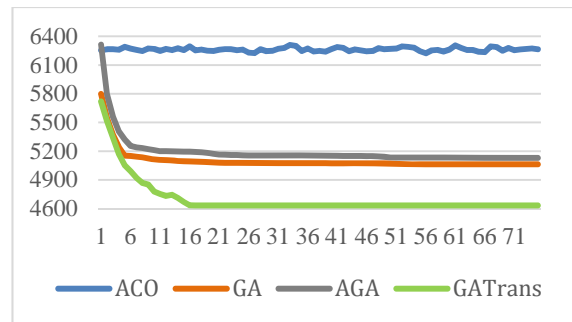


Figure 5: Mean fitness function of a chromosome or ant population over 75 iterations of each method.

The Wilcoxon test is used to infer if two samples come from the same distribution and, if they are not, the test also classifies which sample is composed of statistically lower values. Thus, we use this statistical test to decide whether GA-Trans is statistically equivalent to some other metaheuristic evaluated. As observed in Table 7, assuming as initial hypothesis (H_0) that GA-Trans is equal to some other metaheuristic, the Wilcoxon test presents zero or almost zero p -values (0.0026 and 0.0072), which means that the initial hypothesis must be discarded as it is very unlikely. Similarly, assuming that GA-Trans has makespan values statistically lower than the values presented by other metaheuristics, the Wilcoxon test presents p -values equal to or very close to 1, which guarantees that GA-Trans presents statistically lower values.

Table 7: Wilcoxon test.

H_0	p -Value	Confidence Level
GA-Trans=GA	0	95%
GA-Trans<GA	1	95%
GA-Trans=GA	0.0026	95%
GA-Trans<GA	0.9987	95%
GA-Trans=GA	0.0072	95%
GA-Trans<GA	0.9965	95%

Scenario 20 × 8. The parameter settings of the methods were the same as the previous test, Table 2-4. In Table 8, some statistical measures extracted from the 35 evaluations on the 20 × 8 scenario are presented.

Table 8: Statistical Measures.

	ACO	GA	AGA	GA-Trans
Mean	7164.8	7252.3	7193.6	6799.0
Standard Deviation	76.3	169.3	198.1	197.7
Minimum Value	6870	6829	6788	6301
Maximum Value	7342	7584	7583	7146
Mode	–	7254	7221	6688
Average of time (s)	17.35	2.07	3.26	3.01

As we can see in Table 8, the results obtained in the 20 × 8 scenario by the proposed technique are, on average, smaller than the values presented by the other techniques. In addition, GA-Trans presented the smallest makespan (6301) of all of the techniques discussed when considering the 35 evaluations conducted in each. In addition, GA-Trans presents as the most often occurring value the value 6688, which

is a makespan value that is less than the minimum value presented by the ACO, GA and AGA.

With respect to the average time of the GA-Trans, it can be affirmed that it is very competitive to GA-like methods and usually takes only 17,34% of the time spent by the ACO, which makes the time spent by the ACO a noncompetitive time, so we present in Figure 6 a comparison between the times spent by the compared GA-like techniques.

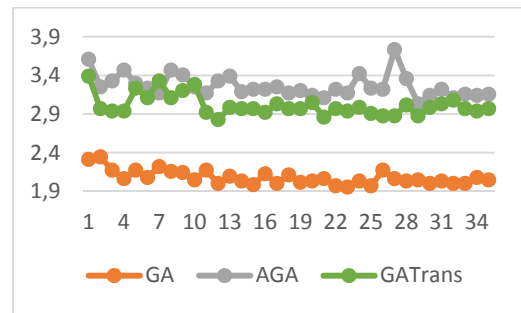


Figure 6: Time taken in 35 tests for each technique.

It can be observed in the box plot of Figure 7 that GA-Trans presents significantly better results than the other methods. In fact, the increase in the complexity of the scenario increased the differences presented by the comparative techniques and made GA-Trans stand out among the others. Using the Wilcoxon test would be redundant in stating that GA-Trans gives the best results.

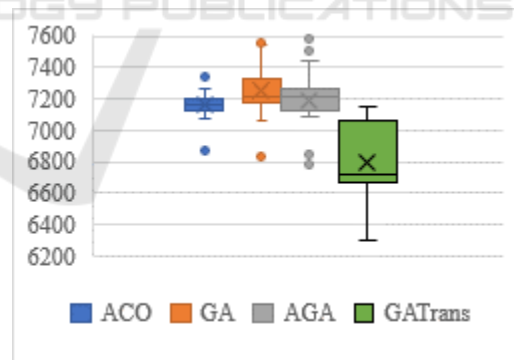


Figure 7: Box plot of the methods' results.

Scenario 100 × 40. The parameter settings of the methods were the same as the previous test, Table 2-4. Similar to the other scenarios, as we can see in Table 9, GA-Trans presented even more promising measures than in the other less complex scenarios. In the case, GA-Trans presented the smallest makespan (7348) of all of the techniques discussed when considering the 35 evaluations conducted in each. In addition, GA-Trans presents as the most often occurring value the value 8016, which is a makespan

value that is less than the minimum value presented by the GA and AGA.

Table 9: Statistical Measures.

	ACO	GA	AGA	GA-Trans
Mean	8682.7	8677.3	8757.8	7830.1
Standard Deviation	350.0	164.7	192.0	254.3
Minimum Value	7643	8274	8423	7348
Maximum Value	9162	9105	9162	8569
Mode	8793	8667	–	8016
Average of time (s)	115.37	13.91	22.93	19.14

With respect to the average time of the GA-Trans, it can be affirmed that it is very competitive to GA-like methods and usually takes only 16.59% of the time spent by the ACO, as it can be observed in more detail in Figure 8.

In Figure 9 it is possible to observe that GA-Trans maintained its good search direction, achieving much better results than the compared methods. Moreover, the difference between the compared techniques is so great in these last two scenarios that the use of the Wilcoxon test has become redundant, as the box plot show that GA-Trans statistically gets lower makespan values than the other techniques addressed.

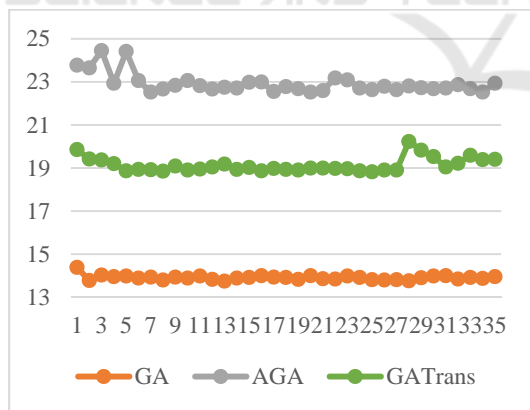


Figure 8: Time taken in 35 tests for each technique.

With the obtained results it is possible to verify that the proposed method obtains good results in scenarios of less complexity and the results are more evident in the ones of greater complexity. Besides, the proposal was able to find better makespan results in a competitive processing time.

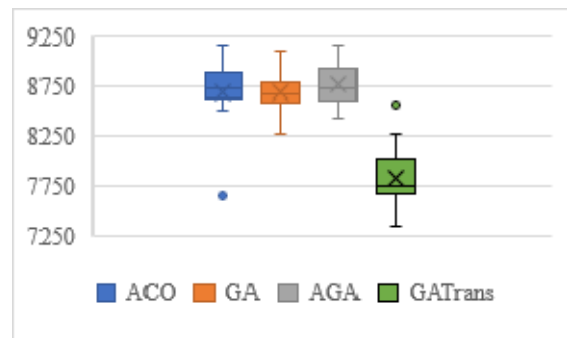


Figure 9: Box plot of the methods' results.

5 CONCLUSION

The objective of the paper was to develop an alternative version of the transgenic operator proposed by Amaral and Hruschka (2014) to reduce the makespan in job shop scheduling problem. The proposal was evaluated, and the results obtained were compared to other approaches proposed in related work (Morandin et al., 2008a; Morandin et al., 2008b; Kato et al., 2010), using as an evaluation criteria the minimization of the makespan value and the time to obtain the response. The Transgenic operator proposed in this work is an altered and adapted version of the original operator proposed by Amaral and Hruschka (2014), in this way it is a new operator that was inspired by the original operator with the focus of obtaining a better performance in the specific problem. Some changes were proposed in the original operator with the purpose of a simplification of the method and a better use in the JSSP. Therefore, the fundamental contribution of this work is this new operator that translates into efficiency in the optimization of the JSSP and also the significant improvement of the results obtained in a scenario present in the literature with a competitive time.

The first JSSP scenario that was evaluated (9 × 9), comparing the makespan values obtained, there was a tendency of improvement of the proposed algorithm in 82.86% of the cases in comparison to the results obtained with the GA and in 71.43% with the AGA. In comparison with the ACO technique, there was a tendency to improve the proposed algorithm in 65.71% of the cases. The mean execution time of the proposed algorithm was 1.53 seconds, while the mean time spent by the GA was 1.12 seconds; the Adaptive GA used 1.66 seconds, and the ACO used 6.62 seconds, i.e., there was an increase of 36.6% when the proposed technique was compared with GA, and there was a reduction in the

processing time of the proposed algorithm of 7.83% with respect to the AGA and a reduction of 76.89% with respect to the ACO. In addition, by comparing the values of the makespan obtained for the problem addressed, it is possible to conclude by means of the Wilcoxon statistical test, with 95% confidence, that the proposed method will have better results than the results obtained by the GA, Adaptive GA and ACO.

With respect to the last two evaluated scenarios, the box plot visualization showed that the GA-Trans technique presented much better results than the other techniques approached, statistically outperforming them and it was useful to confirm the versatility of the proposed method.

The genetic algorithm with a transgenic operator is promising in solving the JSSP. Thus, it is convenient that in future studies, the proposed algorithm is applied in problems similar to the JSSP, since the GA with transgenic operator obtained more significant results when compared to other metaheuristics. In this way, it is possible to work equivalently when applied to other combinatorial problems. It would also be interesting to study possible alternative techniques to determine the most significant genes that are passed in the transgenics.

REFERENCES

- Amaral, L. R., & Hruschka Jr, E. R. (2014). Transgenic: An evolutionary algorithm operator. *Neurocomputing*, 127, 104-113.
- Antonio, L. M., & Coello, C. A. C. (2017). Coevolutionary multiobjective evolutionary algorithms: Survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 22(6), 851-865.
- Asadzadeh, L. (2015). A local search genetic algorithm for the job shop scheduling problem with intelligent agents. *Computers & Industrial Engineering*, 85, 376-383.
- Dao, T. K., Pan, T. S., & Pan, J. S. (2018). Parallel bat algorithm for optimizing makespan in job shop scheduling problems. *Journal of Intelligent Manufacturing*, 29(2), 451-462.
- Guo, P., Wang, X., & Han, Y. (2010). The enhanced genetic algorithms for the optimization design. In *2010 3rd International Conference on Biomedical Engineering and Informatics* (Vol. 7, pp. 2990-2994). IEEE.
- Hasan, S. K., Sarker, R., & Essam, D. (2010). Evolutionary scheduling with rescheduling option for sudden machine breakdowns. In *IEEE Congress on Evolutionary Computation* (pp. 1-8). IEEE.
- Holland, J. (1975). *Adaptation in natural and artificial systems: an introductory analysis with application to biology*. Control and artificial intelligence.
- Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.
- Hosseinabadi, A. A. R., Vahidi, J., Saemi, B., Sangaiah, A. K., & Elhoseny, M. (2019). Extended genetic algorithm for solving open-shop scheduling problem. *Soft computing*, 23(13), 5099-5116.
- Kato, E. R., Morandin, O., & Fonseca, M. A. S. (2009). Ant colony optimization algorithm for reactive production scheduling problem in the job shop system. In *2009 IEEE International Conference on Systems, Man and Cybernetics* (pp. 2199-2204). IEEE.
- Kato, E. R., Morandin, O., & Fonseca, M. A. S. (2010). A Max-Min Ant System modeling approach for production scheduling in a FMS. In *2010 IEEE International Conference on Systems, Man and Cybernetics* (pp. 3977-3982). IEEE.
- Kazemi, A., Mohamed, A., Shareef, H., & Zayandehroodi, H. (2012). An Improved Power Quality Monitor Placement Method Using MVR Model and Combine Cp and Rp Statistical Indices. *Journal of Electrical Review*, 88, 205-209.
- Kundakcı, N., & Kulak, O. (2016). Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem. *Computers & Industrial Engineering*, 96, 31-51.
- Kurdi, M. (2016). An effective new island model genetic algorithm for job shop scheduling problem. *Computers & operations research*, 67, 132-142.
- Lawrence, S. (1984). *Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement)*. Graduate School of Industrial Administration, Carnegie-Mellon University.
- Lu, H., Shi, J., Fei, Z., Zhou, Q., & Mao, K. (2018a). Analysis of the similarities and differences of job-based scheduling problems. *European Journal of Operational Research*, 270(3), 809-825.
- Lu, P. H., Wu, M. C., Tan, H., Peng, Y. H., & Chen, C. F. (2018b). A genetic algorithm embedded with a concise chromosome representation for distributed and flexible job-shop scheduling problems. *Journal of Intelligent Manufacturing*, 29(1), 19-34.
- Morandin, O., Kato, E. R., Deriz, A. C., & Sanches, D. S. (2008a). A search method using genetic algorithm for production reactive scheduling of manufacturing systems. In *2008 IEEE International Symposium on Industrial Electronics* (pp. 1843-1848). IEEE.
- Morandin, O., Sanches, D. S., Deriz, A. C., Kato, E. R. R., & Tsunaki, R. H. (2008b). An adaptive genetic algorithm based approach for production reactive scheduling of manufacturing systems. In *2008 34th Annual Conference of IEEE Industrial Electronics* (pp. 1461-1466). IEEE.
- Naqvi, S., Zhu, C., Farre, G., Ramessar, K., Bassie, L., Breitenbach, J., ... & Christou, P. (2009). Transgenic multivitamin corn through biofortification of endosperm with three vitamins representing three distinct metabolic pathways. *Proceedings of the National Academy of Sciences*, 106(19), 7762-7767.
- Nguyen, S., Zhang, M., & Tan, K. C. (2018). Adaptive charting genetic programming for dynamic flexible job shop scheduling. In *Proceedings of the Genetic and*

- Evolutionary Computation Conference* (pp. 1159-1166). ACM.
- Nie, L., Gao, L., Li, P., & Li, X. (2013). A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. *Journal of Intelligent Manufacturing*, 24(4), 763-774.
- Öztop, H., Tasgetiren, M. F., Eliiyi, D. T., & Pan, Q. K. (2018). Iterated greedy algorithms for the hybrid flowshop scheduling with total flow time minimization. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 379-385). ACM.
- Peng, C., Fang, Y., Lou, P., & Yan, J. (2018). Analysis of double-resource flexible job shop scheduling problem based on genetic algorithm. In *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)* (pp. 1-6). IEEE.
- Romero, M. A. F., García, E. A. R., Ponsich, A., & Gutiérrez, R. A. M. (2018, July). A heuristic algorithm based on tabu search for the solution of flexible job shop scheduling problems with lot streaming. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 285-292). ACM.
- Veldhuizen, D. A. V., & Lamont, G. B. (2000). On measuring multiobjective evolutionary algorithm performance. In *Proceedings of the 2000 Congress on Evolutionary Computation*. CEC00 (Cat. No. 00TH8512) (Vol. 1, pp. 204-211). IEEE.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS