2nd International Conference on Engineering Optimization
September 6 - 9, 2010, Lisbon, Portugal

Solving the Job Shop Problem with a random keys genetic algorithm with instance parameters

José António Oliveira[1], Luís Dias[1], Guilherme Pereira[1]

[1] DPS, University of Minho, Braga, Portugal, {zan, lsd, gui}@dps.uminho.pt

Abstract
In this work is presented a genetic algorithm for the Job Shop Scheduling Problem (JSSP). The genetic algorithm is based in random keys chromosome that is very easy to implement and allows using conventional genetic operators for combinatorial optimization problems.
JSSP is a classic combinatorial optimization problem and is also a NP-hard problem. In the JSSP each job is formed by a set of operations that has to be processed in a set of machines. Each job has a technological definition that determines a specific order to process the job's operations, and it is also necessary to guarantee that there is no overlap, in time, in the processing of operations in the same machine; and that there is no overlap, in time, in the processing of operations of the same job. The objective of the JSSP is to conclude the processing of all jobs as soon as possible, this is, to minimize the makespan. The JSSP represents several real situations of planning and for that it is a very important problem. Recently, the load operations in a warehouse were modeled by a JSSP with recirculation.
The use of exact algorithms for the Job Shop is still limited to instances of small size. The alternative to solve the Real-World Job Shop Scheduling Problem is the use of heuristic procedures. Genetic Algorithms is a well known heuristic technique and largely used on the engineering field of solving optimization problems. This genetic algorithm includes specific knowledge of the problem to improve its efficiency. It is used a constructive algorithm based in Giffler-Thompson's algorithm to generate active plans. The constructive algorithm reads the chromosome and decides which operation is scheduled next. The first population generation is based on the instances parameters. This option increases the effectiveness of the genetic algorithm.
The genetic algorithm is tested by using some benchmark problems and is presented computational results.
Keywords: Metaheuristics, Job Shop Scheduling, Genetic Algorithm; Random keys.

1. Introduction

Some Combinatorial Optimization Problems are very hard to solve and therefore require using heuristic procedures. One of them is the Job Shop Scheduling Problem (JSSP). The use of exact methods to solve the JSSP is limited to the instances of small size. According to Zhang et al. [1] the Branch and Bound methods do not solve instances larger than 250 operations in a reasonable time. As stated in Liu et al. [2] in practical manufacturing environments the scale of job shop scheduling problems could be much larger. They exemplify that in some big textile factories, where the number of jobs may be up to 1,000.
The heuristic methods have become very popular and have gained much success in solving job shop scheduling problems. In the last twenty years a huge quantity of papers was published presenting several metaheuristic methods. From Simulated Annealing [3] to Particle Swarm Optimization [4], there are several variants of the same method class. Very popular between the researchers is the Evolutionary Algorithms [5,6,7,8,9,10,11,12,13]. In 1996, Vaessens et al. [14] stated a goal for the Job Shop Problem: to achieve an average error of less than two percent within 1,000 seconds total computation time. In this work the authors presented the Genetic Algorithms as the less effective metaheuristic to solve the JSSP. A possibility to increase the efficiency and the effectiveness is an algorithm that includes specific knowledge of the problem. Several works include some specific local search for the JSSP that is based in the critical path on a disjunctive graph to model the JSSP. For a long period, the Nowicki and Smutnicki's tabu search method [15] was seen as the most effective and efficient method for JSSP. In 2005, the authors presented a new version of a tabu search for the JSSP.
Presented in this work is a new idea for improving the effectiveness and efficiency in an Evolutionary Algorithm to solve the JSSP. Since a difficulty with the JSSP is the change from instance to instance, a procedure is implemented to take knowledge from the instance and transfer it to the Population Generation parameters.
The paper is organized as follows. An introductory section defines the JSSP problem and its representation with the use of a disjunctive graph, in terms of solutions sets. A central section describes the methodology. An experimental section describes the performance of our algorithm and the most interesting results are explained. Some conclusions and a discussion on future work end the paper.

2. The Job Shop Scheduling Problem

The JSSP represents several real situations of planning and for that reason it is a very important problem. The JSSP is an important practical problem in the fields of production management and manufacturing engineering. The applications of JSSP can be found in production planning, project resource management, distributed or parallel computing, and many other related fields. According to Lin et al. [17], a large number of small to medium companies still operate as job shops.

Throughout the years in the vast bibliography of the JSSP, variations were being presented to the classic model that allowed studying specific real cases. For instance, Kimbrel and Sviridrenko [18] present the particular high-multiplicity JSSP that arises in the integrated circuit fabrication. Also, Oliveira [13] modeled the load operations in a warehouse by a JSSP with recirculation. As Yang et al. [19] say, the JSSP is a hard combinatorial optimization problem and computationally challenging. As they pointed out, "efficient methods for arranging production and scheduling are very important for increasing production efficiency, reducing cost and improving product quality".

In the Job Shop Scheduling Problem (JSSP) each job is formed by a set of operations that has to be processed in a set of machines. Each job has a technological definition that determines a specific order to process the job's operations, and it is also necessary to guarantee that there is no overlap, in time, in the processing of operations in the same machine; and that there is no overlap, in time, in the processing of operations of the same job. The objective of the JSSP is to conclude the processing of all jobs as soon as possible, that is, to minimize the makespan.

The classical JSSP model considers a set of n jobs, and a set of m machines. Each job consists of a set of m operations (one operation on each machine), among which precedence relations exist that define a single order of processing. Each operation is processed in one machine only, during p units of time. The instant when the job is concluded is C. All operations are concluded at Cmax (called makespan).

For the convenience of the representation, operations are numbered consecutively from 1 to $N = n \times m$, in which N is the total number of operations. The classic model considers that all the jobs are processed once in every machine, and the total number of operations is $n \times m$. In a more general model, a job can have a number of operations different from m (number of machines). The case in which a job is processed more than once in the same machine, is called a job shop model with recirculation [12]. Scheduling problems occur wherever a number of tasks have to be performed with limited resources.

The computational and practical significance of the JSSP have motivated the attention of researchers for the last several decades. Yang et al. [19] enumerate the existing approaches for the JSSP that include exact methods such as branch-and-bound and dynamic programming, approximate and heuristic methods such as dispatching priority rules, shifting bottleneck approach, and Lagrangian relaxation. The authors associate the development of artificial intelligence techniques with the rise in many metaheuristic methods that have been applied to the JSSP, such as simulated annealing, tabu search, genetic algorithm, ant colony optimization, particle swarm optimization, and artificial immune system.

Over the years a lot of research has been made into this problem, particularly with genetic algorithms. An important issue in the genetic algorithms is the efficiency. This paper presents the inclusion of a "new" initial population that a generation procedure takes into account at the instance of the problem. The aim of this procedure is to improve the efficiency and the effectiveness of the genetic algorithm.

The JSSP is modeled mathematically. Blazewicz et al. [20] refer to the development of different formulations for this problem. The first one arises in 1959. One of the most used was presented by Adams et al. in 1988, and is based in the disjunctive graph. We address details of this formulation [20].

Roy and Sussman [21] modulated the JSSP using a graph, $G = (V, C \cup D)$, which they designated as a disjunctive graph. The set of nodes is formed by all N operations and by a start node s and an end node e. The set of arcs is formed by two subsets. The conjunctive arcs set C, and the disjunctive arcs set D. Figure 1 presents the disjunctive graph of an example with four jobs and four machines. Each job is represented by a row of nodes. In each node is represented the operation's index and the machine where the operations are processed. The nodes (operations) of the same job are connected by a conjunctive arc, and represent the precedence relations that exist between them. For each pair of operations that are processed in the same machine (belonging to a different job) there exits one disjunctive arc. This arc is a non-oriented arc, and is represented by a hatch line. If an operation i is performed before operation j, the arc is oriented form i to j. All operations that are processed in the same machine form a sub-graph clique of disjunctive arcs.

The sequencing based on the disjunctive graph consists (for all machines) in the definition of a processing order between all operations that are processed by that same machine. A schedule is valid if the resulting oriented graph is acyclic. The longest path length is also designated by a critical path of the acyclic graph and is equal to the value of $C_{\max}$.

A lot of research has been focused on obtaining and improving solutions for the JSSP. For a review and

comparison, we refer the reader to Blazewicz et al. [20], Cheng et al. [26], Jain and Meeran [22], and to Vaessens et al. [14].

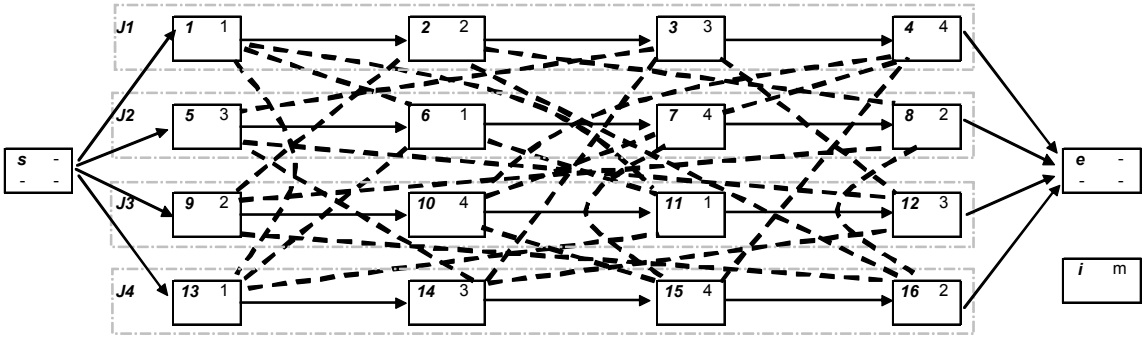


Figure 1: Disjunctive graph

The solutions (schedules) for the JSSP can be classified in 3 sets: semi-actives, actives, non-delayed. These solutions obey relations of inclusion that are illustrated in the Venn diagram that is shown in Figure2. In relation to the optimal solution of the problem (minimization of $C_{\max}$), it is known that it is an active schedule but not necessarily a non-delayed schedule.

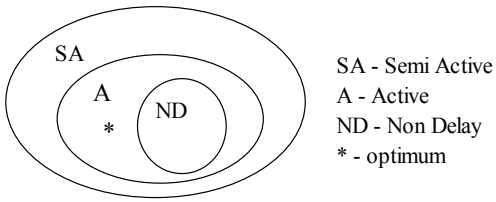


Figure 2: Type of schedules

The JSSP solution consists of obtaining an orientation of the disjunctive arcs in such a way that we can achieve the minimum critical path. A critical block of operations is defined as a set with the maximum successive operations that belong to the critical path and that are processed in that same machine. In the literature there are some neighbourhood schemes introduced based on the reorientation of some disjunctive arcs of the critical path. Among the already existing ones we emphasize the scheme proposed by Nowicki and Smutnicki [15], because it is simple and limits the exchanges to the operations of extremities of the blocks. For additional explanations on the neighbourhood schemes based on blocks, we refer the reader to Jain et al. [23].

3. Methodology

In this work we adopted a method based on genetic algorithms. This technique's simplicity to model more complex problems and its easy integration with other optimization methods were factors that were considered for its choice. The algorithm proposed was conceived to solve the classical JSSP, but it is possible to use the same method to solve other variants of the JSSP.

One of the features that differentiates conventional genetic algorithms is the fact that the algorithm does not deal directly with the problem's solutions, but with a solution representation, the chromosome. The algorithm manipulations are done over the representation and not directly over the solution [24].

Traditionally, genetic algorithms used bit string chromosomes. These chromosomes consisted of only '0s' and '1s'. Modern genetic algorithms more often use problem-specific chromosomes, as the balance between flexibility and raw efficiency tends away from the latter, and with evidence that use of real-valued chromosomes often outperformed bit string chromosomes anyway. Another alternative is the Gray code that is a binary numeral system where two successive values differ in only one digit [24].

The permutation code was adequate to permutation problems. In this kind of representation, the chromosome is a literal of the operations sequence on the machines. In the classical JSSP case, the chromosome is composed by m sub-chromosomes, one for each machine, each one composed by n genes, one for each operation [13]. The *i* gene of the sub-chromosome corresponds to the operation processed in *i* place in the corresponding machine. The allele identifies the operation's index in the disjunctive graph [13].

Nevertheless, in this work, the random key code presented by Bean [25] is used. As Gonçalves et al. state, the important feature of random keys is that all offspring formed by crossover are feasible solutions, when it is used as a constructive procedure based on the available operations to schedule and the priority is given by the random key allele. Through the dynamics of the genetic algorithm, the system learns the relationship between random key vectors and solutions with good objective function values. Another advantage of the random key representation is the possibility of using the conventional genetic operators. This characteristic allows the use of the genetic algorithm with other optimization problems, adapting only a few routines related with the problem.
A chromosome represents a solution to the problem and is encoded as a vector of random keys (random numbers). In this work, according to Cheng et al. [26], the problem representation is indeed a mix from priority rule-based representation and random keys representation.
The solutions are decoded by an algorithm, which is based on Giffler and Thompson's algorithm [27]. While the Giffler and Thompson's algorithm can generate all the active plans, the constructor algorithm only generates the plan in agreement with the chromosome. As advantages of this strategy, we have pointed out minor dimension of solution space, which includes the optimum solution and the fact that it does not produce impossible or disinteresting solutions from the optimization point of view. On the other hand, since the dimensions between the representation space and the solution space are very different, this option can represent a problem because two chromosomes can represent the same solution.
The constructive algorithm has N stages and in each stage an operation is scheduled. To assist the algorithm's presentation, consider the following notation existing in stage t:

P_t - the partial schedule of the $(t-1)$ scheduled operations;

S_t - the set of operations schedulable at stage t, i.e. all the operations that must precede those in S_t are in P_t;

σ_k - the earliest time that operation o_k in S_t could be started;

ϕ_k - the earliest time that operation o_k in S_t could be finished, that is $\phi_k = \sigma_k + p_k$;

M^* - the selected machine where $\phi^* = \min_{o_k \in S_k} \{\phi_k\}$;

S_t^* - the conflict set formed by $o_j \in S_t$ as processed in M^* and $\sigma_j < \phi^*$.

o_j^* - the selected operation to be scheduled at stage t

The constructor algorithm of solutions is presented in a format, similar to the one used by Cheng et al. [26] to present the Giffler and Thompson algorithm [27].

Algorithm 1: Constructive algorithm

Step 1	Let $t=1$ with P_1 being null. S_1 will be the set of all operations with no predecessors; in other words those that are first in their job.
Step 2	Find $\phi^* = \min_{o_k \in S_t} \{\phi_k\}$ and identify M^*. If there is a choice for M^*, choose arbitrarily. Form S_t^*.
Step 3	Select operation o_j^* in S_t^*, with the greatest allele value. Otherwise, go to *Step 4*.
Step 4	Move to next stage by (1) adding o_j^* to P_t, so creating P_{t+1}; (2) deleting o_j^* from S_t and creating S_{t+1} by adding to S_t the operation that directly follows o_j^* in its job (unless o_j completes its job); (3) incrementing t by 1.
Step 5	If there are any operations left unscheduled $(t < N)$, go to *Step 2*. Otherwise, stop.

In *Step 3* instead of use a priority dispatching rule, the information given by the chromosome is used. If the maximum allele value is equal for two or more operations, one is chosen randomly.

3.1. First generation

In the JSSP problem it is possible to calculate for each operation the remaining time for completion of the job. In the disjunctive graph this time is referred to as "tail". This value represents the path length from the operation and node *e*. It is easy to admit that the operations with large tail must be sequenced first because they could define the makespan. Indeed, there exists the dispatching rule that is used in practice and sequence operations in first place

that belongs to the job with Most Work Remaining (MWR).
Consider the following example presented in Table 1, with four jobs (*J1, J2, J3, J4*) and four machines (m1, m2, m3, m4). In this instance there are sixteen operations. Table 1 presents this instance. The operations are numbered sequentially and represented by index *i*. The first four operations belong to the job *J*1. The processing time of each operation is given in the row *p*. Row *m* identifies the machine where the operations must be performed. The tail row shows the remaining time to complete the job after the processing of the operation. For instance, the tail of operation 1 is 67 and is equal to the processing time of operations 2, 3 and, 4, and that is respectively 30, 16, and 21.

Table 1: Example of a JSSP 4x4

	J1				***J2***				***J3***				***J4***			
i	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***
p	16	30	16	21	16	15	3	6	3	3	11	4	10	13	22	14
m	1	2	3	4	3	1	4	2	2	4	1	3	1	3	4	2
tail	67	37	21	0	24	9	6	0	18	15	4	0	49	36	14	0

Figure 3 presents a partial disjunctive graph of the example above. In each node the index of operation *i*, the machine *m*, the processing time *p,* and the tail are represented. For convenience, Figure 3 presents only the disjunctive arcs set of machine 1.

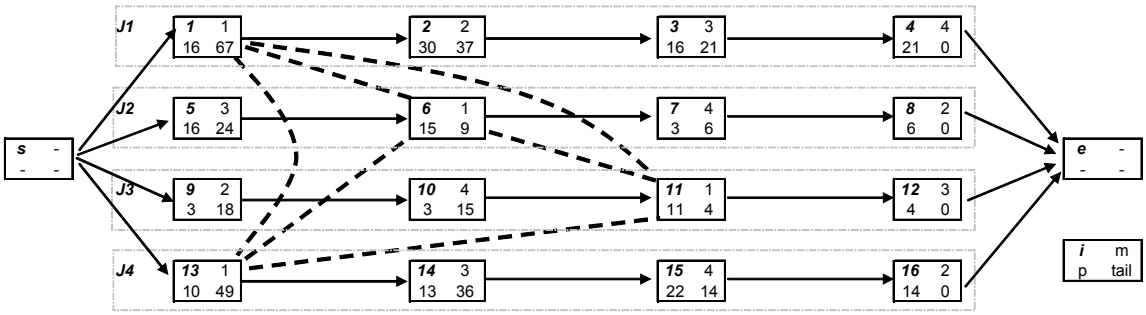


Figure 3: Partial disjunctive graph of example of Table 1.

Applying the MWR rule to this instance the schedule is given in Figure 4. For this small example, this MWR solution is the optimal solution.

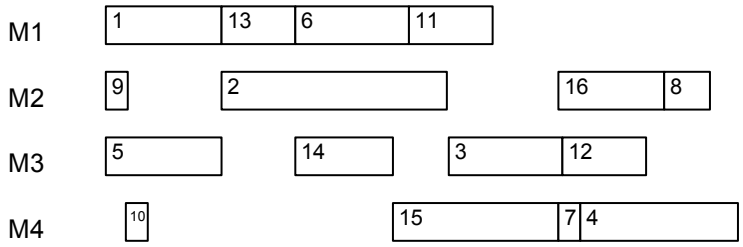


Figure 4: Gantt chart of MWR solution.

In the chosen random key representation, a chromosome is a vector of *N* genes that are real numbers between 0 and 1. The representation has one gene for each operation. Table 2 presents two chromosomes for a JSSP of four jobs and four machines. The second chromosome (chrms2) was generated randomly, while the first chromosome (chrms1) represents the information that is obtained dividing in the example above the tail value per maximum tail value (67 in the example). Applying the constructive algorithm and considering the chromosome chrms1, the solution of Figure 4 is obtained. Attending to this property we implement a procedure to generate the first population that includes some knowledge about the instance in the form of the tail of each operation.

Table 2: Chromosomes for a JSSP 4x4

	J1				J2				J3				J4			
i	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
chrms1	1,00	0,55	0,31	0,00	0,36	0,13	0,09	0,00	0,27	0,22	0,06	0,00	0,73	0,54	0,21	0,00
chrms2	0,04	0,84	0,62	0,16	0,46	0,15	0,47	0,13	0,39	0,60	0,33	0,97	0,23	0,04	0,66	0,33

To generate L individuals in a population the following expression at Eq.(1) is used to generate the allele value for each gene j:

$$allele_j = \frac{randbetween\left(tail_j; tail_j + i \times GAP\right)}{\max_j(tail) + i \times GAP} \quad , \quad i = 0,\ldots,L-1 \quad , \quad j = 1,\ldots,N \tag{1}$$

3.2. Considerations about GAP

The *GAP* value allows obtaining at the end of population a quite random chromosome. The higher the *GAP, the* more random is the population. Lower *GAP* gives a population close to the MWR dispatch rule.

For a population of 50 individuals, the graphic presented in Figure 5 presents some statistics of the allele values for each gene (16 operations / genes). The average value, maximum value, minimum value and also the tail value of each operation is presented.

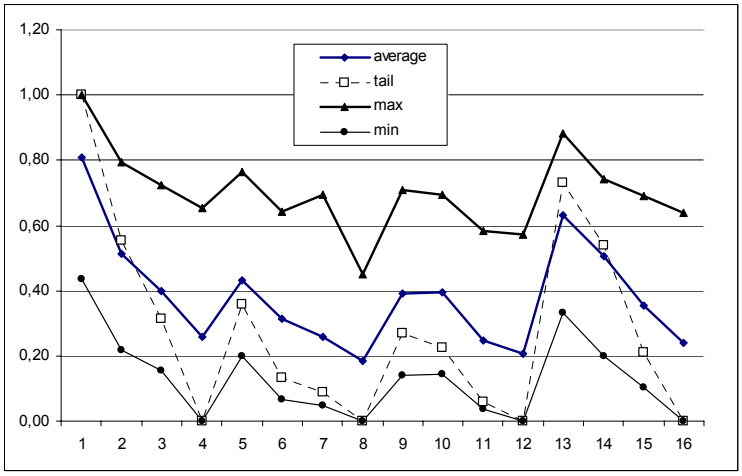
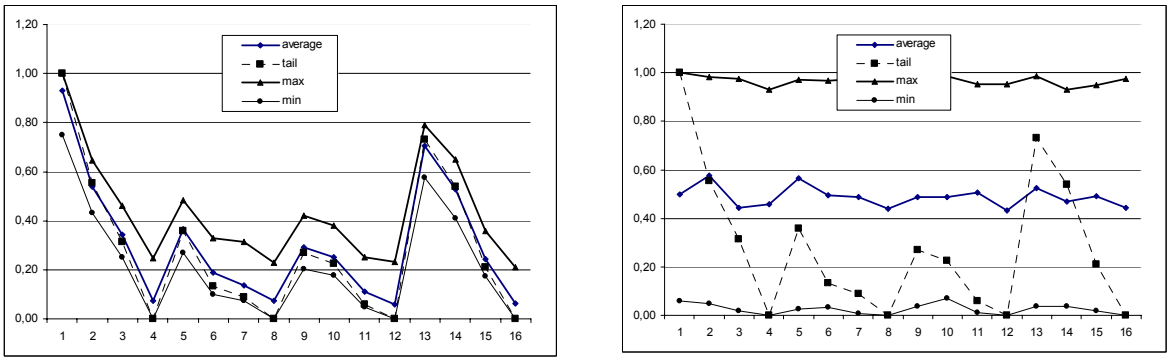


Figure 5: Statistics of allele's values.

Figure 6 shows the *GAP* effect in the statistics of the allele values for the 16 operations. Figure 6 a) shows the statistics when a small *GAP* is used. The values are very close to the tail of each operation. Figure 6 b) shows the statistics for a high *GAP*. The range of variation for the alleles is very similar for all genes, so the population is "more random".



a) b)

Figure 6: The *GAP* effect on statistics of the 16 genes.

Figure 7 presents the alleles' values of four genes when using a *GAP*=5. These genes belong to the same machine because they represent the priority of operation 1, 6, 11, and 13. The tails of the operations are respectively 67, 9, 4, and 49. It is possible to verify that for the first chromosomes the alleles' values translate the priority give by the tail of each operation. It is also possible to verify that the last individuals have quite random alleles. It is possible to see that in some individuals the operation 13 (Gene13) and operation 6 (Gene6) have a priority greater than the operation 1 (Gene1). This situation allows a first sequence of these operations before operation 1, which is the operation with the greatest tail.

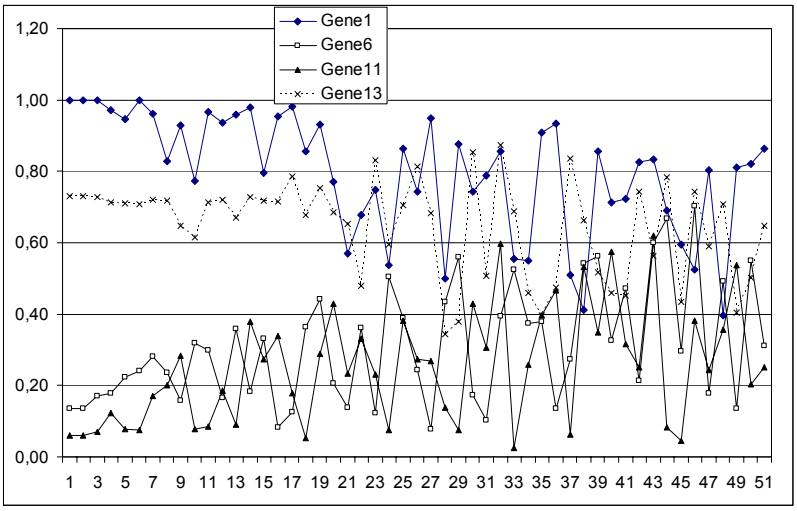
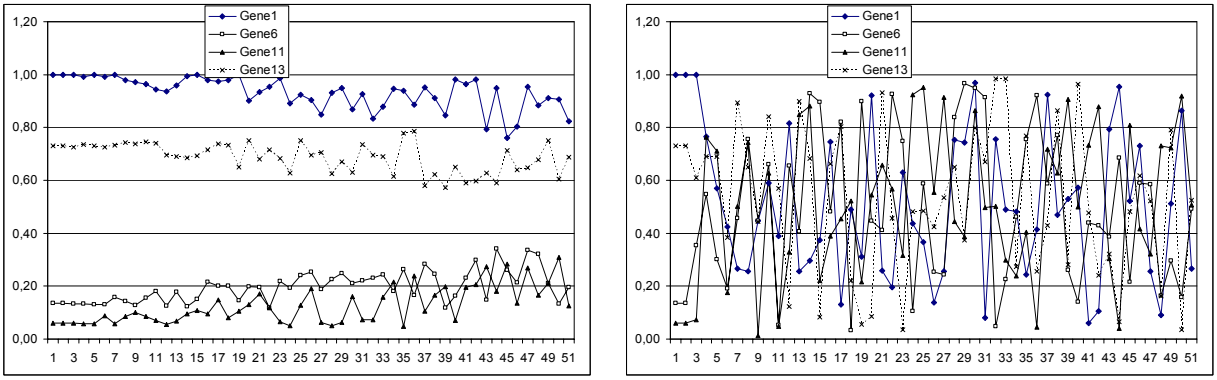


Figure 7: The allele generation.

Figure 8 shows the effect of *GAP* on the allele generation for the genes of machine 1. Figure 8 a) shows the alleles values when used as small *GAP*. The values are very close to the tail of each operation. Figure 8 b) shows the statistics for a high *GAP*. The range of variation for the alleles is very similar for all genes, so these allow "any" sequence to schedule these four operations in machine 1.



a) b)

Figure 8: The effect of *GAP* on allele generation.

3.2. The algorithm structure

The genetic algorithm has a very simple structure and can be represented in the Algorithm 2. It begins with population generation and her evaluation. Attending to the fitness of the chromosomes the individuals are selected to be parents. The crossover is applied and it generates a new temporary population that also is evaluated. Comparing the fitness of the new elements and of their progenitors the former population is updated.

The Uniform Crossover (UX) is used this work. This genetic operator uses a new sequence of random numbers and swaps both progenitors' alleles if the random key is greater than a prefixed value. Table 3 illustrates the UX's application on two parents (prnt1, prnt2), and swaps alleles if the random key is greater or equal than 0.75. The genes 3, 4 and 16 are changed and it originates two descendants (dscndt1, dscndt2). Descendant 1 is similar to

parent 1, because it has about 75% of genes of this parent.

Algorithm 2: Genetic algorithm

begin
$P \leftarrow$ GenerateInitialPopulation()
Evaluate(P)
while termination conditions not meet **do**
    $P' \leftarrow$ Recombine(P) //UX
    Evaluate(P')
    $P \leftarrow$ Select($P \cup P'$)
end while

Table 3: The UX crossover

	J1				*J2*				*J3*				*J4*			
i	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***
prnt1	0,89	0,49	0,24	0,03	0,41	0,11	0,24	0,12	0,33	0,30	0,27	0,24	0,80	0,53	0,28	0,18
prnt2	0,83	0,41	0,40	0,04	0,29	0,35	0,38	0,01	0,42	0,32	0,28	0,28	0,72	0,61	0,44	0,13
randkey	0,64	0,72	0,75	0,83	0,26	0,56	0,28	0,31	0,09	0,11	0,37	0,25	0,23	0,05	0,40	0,76
dscndt1	0,89	0,49	0,40	0,04	0,41	0,11	0,24	0,12	0,33	0,30	0,27	0,24	0,80	0,53	0,28	0,13
dscndt2	0,83	0,41	0,24	0,03	0,29	0,35	0,38	0,01	0,42	0,32	0,28	0,28	0,72	0,61	0,44	0,18

4. Computational experiments

Beasley [28] developed an OR-Library and it is a collection of test data sets for a variety of Operations Research (OR) problems. The benchmark problems are taken from this OR-Library. Several researchers have addressed the importance of solving Job Shop Scheduling Problems, which will help in solving real world problems in industries and in scheduling their jobs perfectly.

Table 4: Computational results

	Average of 5 runs Iteration nº										Best of 5 runs Iteration nº									
Instance	**1**	**10**	**50**	**100**	**500**	**1000**	**2000**	**3000**	**4000**	**5000**	**1**	**10**	**50**	**100**	**500**	**1000**	**2000**	**3000**	**4000**	**5000**
la21-tail	1300,3	1263,5	1229,2	1211,3	1173,5	1164,0	1141,3	1135,6	1133,6	1132,0	1279	1238	1206	1176	1153	1141	1118	1118	1118	1118
la21-rand	1319,3	1274,5	1241,3	1227,7	1190,5	1182,2	1168,3	1161,8	1149,4	1144,0	1288	1240	1208	1208	1176	1168	1140	1158	1118	1118
la02-tail	761,0	699,0	697,0	677,0	672,0	665,0	665,0	662,0	662,0	662,0	761	699	697	677	672	665	665	662	662	662
la02-rand	774,0	726,0	708,0	676,0	665,0	665,0	665,0	665,0	665,0	665,0	774	726	708	676	665	665	665	665	665	665
la19-tail	957,7	934,8	907,2	905,5	882,0	874,0	865,0	860,0	856,8	854,8	909	909	892	892	872	868	848	848	848	848
la19-rand	971,3	940,0	915,8	901,0	884,2	875,7	866,3	859,7	858,3	856,8	956	917	901	885	876	867	857	857	852	850
la24-tail	1184,0	1129,8	1103,0	1091,5	1061,2	1055,7	1033,7	1025,5	1021,3	1017,5	1164	1095	1092	1080	1049	1042	1024	1018	1009	1009
la24-rand	1201,7	1135,0	1102,7	1086,0	1062,0	1055,0	1042,5	1037,3	1029,8	1027,0	1179	1115	1055	1055	1035	1035	1018	1018	1018	1014
la25-tail	1225,2	1165,5	1138,5	1127,0	1089,7	1070,7	1053,3	1032,7	1031,8	1029,5	1193	1123	1122	1108	1073	1053	1033	1010	1010	1010
la25-rand	1247,5	1189,3	1160,2	1143,3	1093,5	1077,8	1060,5	1049,3	1042,5	1037,5	1196	1155	1148	1123	1088	1064	1045	1042	1032	1027
la27-tail	1589,3	1541,0	1500,8	1499,0	1445,0	1438,8	1434,5	1424,8	1421,5	1418,5	1475	1475	1475	1475	1425	1422	1415	1413	1413	1413
la27-rand	1622,3	1573,2	1524,8	1510,2	1485,7	1465,8	1448,0	1439,3	1435,5	1427,3	1598	1552	1498	1495	1468	1452	1426	1426	1426	1401
la29-tail	1524,7	1473,3	1437,3	1420,0	1369,3	1352,5	1335,2	1320,3	1311,5	1311,0	1499	1433	1418	1392	1340	1340	1317	1300	1292	1292
la29-rand	1566,8	1503,7	1450,3	1423,5	1386,8	1376,8	1348,3	1344,5	1336,0	1325,8	1525	1486	1419	1392	1369	1363	1335	1335	1325	1303
la30-tail	1637,0	1579,2	1541,2	1535,3	1493,8	1477,5	1460,5	1449,8	1447,8	1447,7	1609	1548	1528	1527	1476	1467	1437	1434	1434	1434
la30-rand	1638,2	1610,2	1564,3	1549,8	1509,8	1495,7	1480,3	1477,3	1468,8	1463,3	1604	1571	1549	1533	1495	1470	1466	1466	1452	1443
la36-tail	1552,2	1483,2	1437,7	1420,0	1372,5	1343,5	1338,2	1329,5	1318,5	1314,7	1498	1450	1413	1392	1353	1316	1307	1307	1295	1289
la36-rand	1583,2	1500,0	1451,0	1417,0	1384,5	1368,2	1356,5	1346,5	1337,2	1330,3	1536	1471	1406	1392	1362	1340	1340	1321	1314	1314
la37-tail	1687,8	1635,8	1608,7	1588,5	1547,3	1542,3	1526,2	1517,0	1506,0	1505,0	1673	1622	1592	1571	1520	1520	1498	1492	1492	1492
la37-rand	1683,0	1649,0	1599,5	1587,5	1564,3	1553,8	1534,8	1521,7	1520,2	1517,2	1647	1602	1590	1564	1554	1535	1514	1500	1500	1500
la38-tail	1499,8	1447,8	1420,2	1410,2	1359,8	1348,8	1334,2	1312,5	1307,3	1304,8	1468	1429	1406	1403	1328	1328	1328	1296	1296	1288
la38-rand	1505,3	1466,5	1416,5	1392,2	1369,3	1347,2	1332,7	1318,5	1314,2	1311,7	1465	1443	1400	1363	1345	1328	1307	1290	1290	1290
la39-tail	1546,5	1501,2	1457,0	1435,8	1402,0	1368,3	1356,3	1347,8	1343,7	1338,7	1512	1462	1439	1410	1390	1348	1337	1337	1337	1329
la39-rand	1588,8	1491,3	1463,8	1444,2	1400,0	1390,7	1366,3	1361,7	1351,5	1350,2	1554	1466	1430	1410	1389	1367	1336	1336	1336	1336
la40-tail	1498,7	1447,7	1419,3	1401,0	1370,2	1363,0	1345,7	1335,3	1329,2	1325,7	1439	1411	1401	1374	1345	1345	1337	1319	1311	1311
la40-rand	1522,8	1460,2	1438,3	1418,3	1387,3	1366,2	1358,2	1352,2	1349,0	1346,7	1488	1443	1416	1409	1374	1339	1339	1339	1339	1333
mt10-tail	1126,2	1081,2	1060,3	1030,5	1002,3	984,8	969,3	959,5	952,7	949,3	1098	1063	1042	1019	975	972	955	951	940	940
mt10-rand	1143,0	1080,8	1048,2	1031,5	998,3	981,3	966,2	959,8	958,6	956,8	1117	1062	1023	1019	987	964	958	955	955	955

Table 4 presents the computational results. Fourteen instances were chosen from Beasley [28]. These instances

represent some of the hardest to solve and they were also chosen for others researchers [7,11,13,14].
For each instance were made five runs considering the tail information on the initial population, and also were made five runs with an initial population generated randomly. In each run the algorithm performs 5000 iterations. Table 4 shows the average value and the best value of five runs. Each column shows the values in different iterations of the algorithm. In the computation experiments it was only used a value for the *GAP*.
Globally in all experiments the use of tails' information of the instance produces in average 1% better results. This advantage is more evident in the initial iterations, so for some instances it was obtained solutions about 3% better. In terms of best solution the use of tails' information is relevant on the initial iterations. In some instances the use of tails information produces a solution about 8% better than using a random initial population.

5. Conclusions

This paper compares two schemes to generate the initial solution in an evolutionary algorithm to solve the Job Shop Scheduling Problem. The conventional random generation is compared with a new method based on instance's information. It is possible to establish different priorities to the operations related with the remaining processing time to complete the jobs. Using this type of priority was implemented a procedure to generate an initial population. The proposed algorithm uses a representation based on random keys.
The experimental computation performed gives better results when the information of the instance is used, mainly in the first iterations. This is a possibility to increase the effectiveness of a method when it is not possible to spend more time to compute the solutions. This process is very simple and autonomous. Also it is possible to parameterize the use of the instance's information.
The further work consists to apply this strategy to others problems. The first tentative will be the Project Scheduling Problem with Resource Constraints, once that is a generalization of Job Shop Scheduling Problem.
Since the random keys representation has no Lemarkin property is our intent to apply the same scheme of initial generation in other types of representation for the Job Shop Scheduling Problem.

References

[1] C. Zhang, P. Li, Zn Guan and Y. Rao, A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem, *Computers & Operations Research*, 53, 313-320, 2007.
[2] M. Liu, J. Hao, C. Wu, A prediction based iterative decomposition algorithm for scheduling large-scale job shops, *Mathematical and Computer Modelling*, 47, 411–421, 2008.
[3] P. J. van Laarhoven, E. Aarts, and J. Lenstra, Job shop scheduling by simulated annealing. *Operations Research*, 40, 1, 113-125, 1992.
[4] Z. Lian, X. Gu and B. Jiao, A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan, *Applied Mathematics and Computation*, 183, 1008–1017, 2006.
[5] R. Cheng and M. Gen, A tutorial survey of job-shop scheduling problems using genetic algorithms part I: Representation, *Computers & Industrial Engineering*, 34 (4), 983–997, 1996.
[6] L. Davis, Job-shop scheduling with genetic algorithm, in: J.J. Grefenstette (Ed.), *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications*, Lawrence Erlbaum, Pittsburgh, PA, USA, 1985, 136–140.
[7] F. Della Croce, R. Tadei and G. Volta, A genetic algorithm for the job shop problem, Computers & Operations Research, 22 (1), 15–24, 1995.
[8] H.L. Fang, P. Ross, and D. Corne, A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In *Proceedings of the fifth international conference on genetic algorithms*, M. Kaufmann Publishers, 1993, 375–382.
[9] B.J. Park, H.R. Choi, and H.S. Kim, A hybrid genetic algorithm for the job shop scheduling problems, *Computers and Industrial Engineering*, 45, 597–613, 2003.
[10] J. Gao, L. Sun and M. Gen, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, *Computers & Operations Research*, 35, 2892– 2907, 2008.
[11] J.F. Gonçalves, J.J Mendes and M.G.C. Resende, A hybrid genetic algorithm for the job shop scheduling problem, *European Journal of Operational Research*, 167, 77–95, 2005.
[12] J.A. Oliveira, A genetic algorithm with a quasi-local search for the job shop problem with recirculation, *Applied Soft Computing Technologies: The Challenge of Complexity*, , Springer, Berlin / Heidelberg, 2006, 221-234, ISBN 978-3-540-31649-7, DOI 10.1007/3-540-31662-0_18.
[13] J.A. Oliveira, Scheduling the truckload operations in automatic warehouses, *European Journal of Operational Research*, 179, 3, 723-735, 2007.
[14] R. Vaessens, E. Aarts and J.K. Lenstra, Job Shop Scheduling by local search, *INFORMS Journal on Computing*, 8, 302-317, 1996.
[15] E. Nowicki, C. Smutnicki, A fast taboo search algorithm for the job-shop problem, *Management Science* 42, 797-813, 1996.

[16] E. Nowicki, C. Smutnicki, An advanced tabu search algorithm for the job shop problem, *Journal of Scheduling*, 8(2), 145–159, 2005.
[17] T.L. Lin, S.J. Horng, T.W. Kao, Y.H. Chen, R.S. Run, R.J. Chen, J.L. Lai and I.H. Kuo, An efficient job-shop scheduling algorithm based on particle swarm optimization, *Expert Systems with Applications*, 37 (3), 2629-2636, 2009.
[18] T. Kimbrel and M. Sviridenko, High-multiplicity cyclic job shop scheduling, *Operations Research Letters*, 36, 574–578, 2008.
[19] J. Yang, L. Sun, H.P. Lee, Y. Qian and Y. Liang, Clonal Selection Based Memetic Algorithm for Job Shop Scheduling Problems, *Journal of Bionic Engineering*, 5, 111-119, 2008.
[20] J. Blazewicz, W. Domschke and E. Pesch, The job shop scheduling problem: Conventional and new solution techniques, *European Journal of Operational Research*, 93, 1-33, 1996.
[21] B. Roy and B. Sussmann, Les problemes d'ordonnancement avec constraintes disjonctives, *Note DS*, No. 9 Bis, SEMA, Paris, 1964.
[22] A.S. Jain, S. Meeran, A state-of-the-art review of job-shop scheduling techniques, *European Journal of Operations Research*, 113, 390-434, 1999.
[23] A.S. Jain, B. Rangaswamy, S. Meeran, New and "Stronger" Job-Shop Neighbourhoods: A Focus on the Method of Nowicki and Smutnicki, *Journal of Heuristics*, 6, 457-480, 1996.
[24] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
[25] J.C. Bean, Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6, 154–160, 1994.
[26] R. Cheng, M. Gen, and Y. Tsujimura, A tutorial survey of job-shop scheduling problems using genetic algorithms - I. Representation, *Computers & Industrial Engineering*, 30, 983-997, 1996.
[27] B. Giffler and G.L. Thompson, Algorithms for solving production scheduling problems, *Operations Research*, 8, 487-503, 1960.
[28] J.E. Beasley, OR-Library: distributing test problems by electronic mail, *Journal of the Operational Research Society*, 41, 1069-1072, 1990.