International Journal of Production Management and Engineering

doi:10.4995/ijpme.2016.5780

Received 2016-05-15 Accepted: 2016-06-09

Hybrid genetic algorithms: solutions in realistic dynamic and setup dependent job-shop scheduling problems

Rogério M. Branco[a]*, Antônio S. Coelho[b1], Sérgio F. Mayerle[b2]

[a] Instituto Federal do Rio Grande do Sul – IFRS Campus Rio Grande, Rua Alfredo Huch, 475, Centro, CEP 96201-460, Rio Grande, RS, Brasil.

[b] Universidade Federal de Santa Catarina, Caixa Postal 476, Campus Universitário UFSC - Trindade, CEP 88040-900, Florianópolis, SC, Brasil.

[a] *rogerio.branco@gmail.com*

[b1] *a.s.coelho@ufsc.br*

[b2] *sergio.mayerle@ufsc.br*

Abstract: This paper discusses the application of heuristic-based evolutionary technique in search for solutions concerning the dynamic job-shop scheduling problems with dependent setup times and alternate routes. With a combinatorial nature, these problems belong to an NP-hard class, with an aggravated condition when in realistic, dynamic and therefore, more complex cases than the traditional static ones. The proposed genetic algorithm executes two important functions: choose the routes using dispatching rules when forming each individual from a defined set of available machines and, also make the scheduling for each of these individuals created. The chromosome codifies a route, or the selected machines, and also an order to process the operations. In essence , each individual needs to be decoded by the scheduler to evaluate its time of completion, so the fitness function of the genetic algorithm, applying the modified Giffler and Thomson's algorithm, obtains a scheduling of the selected routes in a given planning horizon. The scheduler considers the preparation time between operations on the machines and can manage operations exchange respecting the route and the order given by the chromosome. The best results in the evolutionary process are individuals with routes and processing orders optimized for this type of problem.

Key words: Genetic algorithms, Dispatching rules, Realistic job-shop scheduling.

1. Introduction

The growing competition from companies , arising from the globalized market, reinforces their attention to quality and productivity, focusing in the relationships in the supply chain and flexibility, increasing the efficiency of manufacturing.

In this context, the Flexible Manufacturing System (FMS) combines high flexibility, productivity and low levels of stock: characteristics that accept the alternative routes of production and make it more agile and robust in face of failures. So, if a machine breaks during a task, a reschedule to find an alternate route is done to finish this job, respecting due dates already planned (Porter *et al.,* 1999; Chan, 2003).

In respect to manufacturing systems directly involved with cells and FMS, Porter *et al.* (1999) and Matsuzaki (2004) point to the job-shop class in the production of small volumes and more variety of concurrent processes. In general, the job-shops are process-oriented production systems and obey a pre-defined sequence of processing. The scheduling problems are widely studied because they assume difficult conditions to solve in polynomial time (NP-hard), due to their combinatorial nature (allocating machines to produce parts). Also, the flexibility of alternate routes increases the combinations of resources to compose sequences and therefore, the problem complexity.

The literature presents researches involving the SDST-JSSP - Sequence Dependent Setup Times Job-Shop Problems, which are classic JSSP extensions and in which a setup time between two consecutive operations is required. These extensions make the classic cases closer to realistic situations but more complex.

In dynamic cases, another extension commonly seen is the NDD-JSSP - Non Deterministic Dynamic Job Shop Problems, which differs from classical because the process does not start at time 0, with random characteristics over starting times and thus, close to realistic cases.

Regarding this perspective, this paper propounds combined heuristics techniques and genetic algorithms to solve the combination of these two kind of problems, called NDD-SDST-JSSP. The scheduling algorithm contemplates the goals of shorter processing time and delivery due dates, late start times, variations in processing times of tasks and dependent setup times. Also, there's another one that forms routes and their internal sequencing, integrated in the GA's evaluation function.

2. The JSSP problem

The scheduling problems belong to the NP-hard problems and exact methods are applied only for relatively small examples of the problem (Araújo, 2006). Furthermore, real problems have additional details which involve more combinations than the classics (Herrmann *et al.,* 1995).

The classical JSSP is a group of n jobs to be processed into a set of m machines. Each task has a number of operations and a technological sequence of process. These operations require an uninterrupted processing time over a designed machine. Therefore, it is a time-completion problem that satisfies the constraints: the goal is the minor total completion time - makespan (Vazquez and Whitley, 2000).

In the SDST-JSSP, there is a setup time between consecutive operations in the same machine. Thus, once the operation O_{jv} leaves the machine M_v, before the O_{kv} process starts, a setup time $S_{oiv,okv}$ is added (Gonzales *et al.,* 2005).

3. Methods to solve JSSPs

In the literature, there is a great diversity of methods applied to solve the JSSPs. Jain and Meeran (1998) cite Johnson (1954) as one of the firsts significant works in the theory of scheduling, which aimed to minimize the makespan.

Several other studies have followed him, where the variety of techniques involved and the forms to modelling these problems greatly increased over these nearly six decades.

Following, some methods applied to solve the JSSP can be viewed, without the intention to terminate the discussion about the subject, but only listing the most expressive techniques quite evident in the literature which belong to this state of the art.

3.1. Exact and aproximative methods

Several strategies are presented in the literature to solve the classic JSSP. In decision problems as these ones, the Critical Path Method - CPM is one of the most mentioned. Also, formulations involving Linear Programming (LP), Integer Linear Programming (ILP) or Mixed Integer Programming (MIP) are used. Furthermore, the enumerative methods such as Branch and Bound (BB) are strong highlights and the dynamic programming is evidenced in the optimal solution of the classic JSSPs.

In general, many simplifications are required to problems in order to find solutions, and they still are little adaptable to variations in size, where applications are restricted to a few small problems (Wall, 1996).

It is not difficult to imagine that for real JSSPs, more complex than the classic ones, these implementation strategies will be aggravated.

So, a non exact method is now treated, due to its potential in solving JSSPs: the metaheuristics. They are able to search solutions, consisting in the application, at each step, of a subordinate heuristic, which has to be modeled for each specific problem.

For them, the principle adopted to explore the solution in the space search can be local or populational. In the first case the operation is performed by means of movement applied to each step on the current solution, generating another promising approach in its vicinity, like the Tabu Search or the Simulated Annealing techniques. Already, the methods based on population search, are to maintain a set of good solutions and combine them in order to try to produce even better solutions. Classic examples are the Genetic and also, the Memetic Algorithms. The following will be arranged some more detailed operating information of these metaheuristics, focusing on the problem dealt with in this work.

3.2. Tabu search

This metaheuristic is considered an iterative global optimization technique. Having originated from the search for integer programming problem solving, it was later extended to almost all combinatorial problems (Goldbarg and Luna, 2000).

In general, the Tabu Search (Tabu Search - TS) is a procedure that restricts the search and tries to find optimal solutions, storing the search history in memory. It prohibits (tabu) movements in the neighbourhood with certain attributes, in order to guide the search process as well as solutions (based on available information) have double or are similar to previously stored solutions / obtained.

The work of Jain and Meeran (1998) takes the TS as one of the most efficient search for good solutions in classical job-shop systems. They note that methods like branch and bound, if combined, show improvements in search, yet with greater computational cost. Like most local search strategies, TS requires many parameters that must be carefully adjusted. Considering the imminent application of differences in actual cases studied, this may be a difficult barrier to be overcome.

3.3. Simulated Annealing

Belonging to the random-guided search techniques, Simulated Annealing - SA presents random components, but also employ current status information to guide the search of the solution of the problem studied. It is a local search method that accepts worsening movements to escape from local optima.

It is based on an analogy with thermodynamics, simulating the cooling of a heated set of atoms. For the use of SA should be defined a priori, a method for generating an initial solution s, a method for generating the surrounding solutions S (neighborhood structure) and an objective function f(s) to be optimized (Mauri and Lorena, 2006).

Some contributions to neighborhood functions for JSSP were showed by Jain and Meeran (1998). Basically they consist of the reversal of processing orders from a pair of adjacent critical operations for the same machine. This method of SA proposed appears to be quite robust to the JSSP, but Jain and Meeran (1998) mentioned that the results are, also, poor.

Only when incorporated into other techniques (eg .: genetic algorithm) is that the quality of the results is improved. The authors also mention the excessive consumption of computational time for good solutions can be found, and the high dependence of the parameters to the algorithm's nature. It adds that slow colds also potentiate the best results, but also generate a considerable computational time consumption.

3.4. Evolutionary algorithms

The Evolutionary Algorithms (EAs) are heuristic search techniques based on natural selection mechanisms, computationally simulating the environments which use the principles of evolution and heredity. They operate with a population of solutions (chromosomes, or individuals), applying selection techniques guided by the ability of each one and subsequently genetic operators such as reproduction and mutation act on them, generating new individuals, new solutions.

According to Linden (2008), there are several proposed computational models based on the concept of simulation of evolution through selection and breeding and mutation operators, all dependent on each individual's fitness in their species and the environment in which it's inserted. Barboza (2005) cite some of these methods, as the Evolutionary Strategies (EE), Genetic Programming (PG), Classifier Systems (CS) and Transgenetic Computational (CT), among others, saying that the most widespread and researched is the Genetic Algorithm (GA), given their flexibility and effectiveness in performing global search in different environments.

4. Applied methods

In general, the solution method proposed for NDD-SDST-JSSP is a combination of dispatching rules, the genetic and the modified GT algorithms (Giffler and Thompson, 1960). Thus, several considerations were made, starting from the chromosome coding, application of genetic operators and evaluation of individuals.

4.1. The priority dispatching rules

The most popular heuristic techniques applied to scheduling problems, the Priority Dispatching Rules (PDR´s), have demonstrated their importance in several works and even today, are still widely used in combined methods. Some examples are the work of Singh, Mehta and Jain (2006), El-Bouri and Shah (2006), Branco (2010), among others.

Such importance lies in the easy implementation and low computational cost required. In general, the procedure is to choose a set of operations, not scheduled yet. According to a criterion of choice, a set formed by operations that can be processed in a specific machine will have one of them selected by this adopted criterion, which will be inserted into the scheduling.

According to the tests, are used the following know rules from the literature:

- *RND (random):* Rule based in random uniformly distributed variable;
- *SPT (shortest process time):* gives greater priority to the task that presents smaller processing time.
- *S/RPT (slack per remaining processing time):* gives priority to operations based on the composite index by the ratio of the delivery date, subtracts the task completion and remaining processing time.

There are several other rules, such as SRPT, LTWK and SPT/TWK, commented by Chiang and Fu (2006). Extensions of SPT incorporate, under combined conditions, other goals, also with good efficiency and late operations.

Jain and Meeran (1998) apud Chang *et al.* (1996) show a study evaluating 42 PDRs applied in an integer linear programming model, where the SPT rule showed the best performance.

Regarding work interests, it is important to consider the due time when implementing processes, given the characteristics of demand oriented to orders/requests that the processes are subject to, but without forgetting the relevant conditions concerning flow time in processes, that leads to combined rules. Thus, the aim is to combine some simpler rules to promote better results in low computational time. Therefore, SPT_q (less time needed to complete a process) is selected in Equation 1 and S/RPT, in Equation 2, motivated by the success of the first, in a wide variety of jobs and, for the latter, considering the time needed to complete the ongoing processes.

$$iSPT_q = \sum_{q=j}^{mi} piq \qquad (1)$$

$$iSPRT = \frac{d_i - t - \sum_{q=j}^{mi} piq}{\sum_{q=j}^{mi} piq} = \frac{d_i - t - iSPT_q}{iSPT_q} \qquad (2)$$

where:

d_i = job_i due date;

pij = processing time of the operation j in job $i_;$

t = current time;

m_i = number of operations remaining to finish job_i;

$\sum_{q=j}^{mi} piq$ = $iSPT_q$ = process time remaining (job_i);

Both indexes are inversely proportional to the priority value, i.e., higher priority to lower value and lower priority to higher values, so the algorithm is built to prioritize operations with lower rates.

The iCHR is given by the equations below:

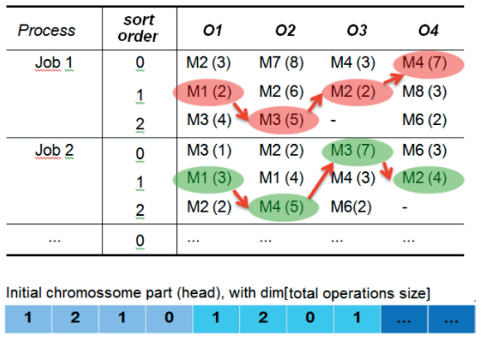
$$iCHR = iSPT_q . \; iSPRT = iSPT_q . \frac{d_i - t - iSPT_q}{iSPT_q} \qquad (3)$$

$$iCHR = d_i - t - iSPT_a \qquad (4)$$

4.2. The genetic algorithm

Solving a wide variety of problems in class NP - complete, Evolutionary Algorithms (EAs) make heuristic search techniques based on natural mechanisms of selection, simulating computational environments based on these principles of evolution and heredity (Goldberg, 1989).

The proposed structure to chromosome has two known parts: head and body. The "head" contains the information of the route and the "body" will act in the operation sequence, both with the same dimensions and, for each operation, the locus contains a machine index. The Figure 1 shows the relationship between the operations and machines available to process them and the structure of the "head" of the chromosome.



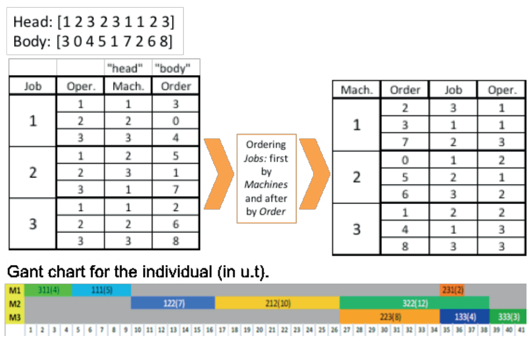
Process	sort order	O1	O2	O3	O4
Job 1	0	M2 (3)	M7 (8)	M4 (3)	M4 (7)
	1	M1 (2)	M2 (6)	M2 (2)	M8 (3)
	2	M3 (4)	M3 (5)	-	M6 (2)
Job 2	0	M3 (1)	M2 (2)	M3 (7)	M6 (3)
	1	M1 (3)	M1 (4)	M4 (3)	M2 (4)
	2	M2 (2)	M4 (5)	M6(2)	-
...	0	...	...	...	...

Initial chromossome part (head), with dim[total operations size]

1	2	1	0	1	2	0	1	...	...

Figure 1. Example of relation between the original table of operations and the chromossome structure – phenotype×genotype (*source: own*).

Also, the "body" contains whole alleles, not repeated, in the interval ai=[0,total operations-1] and, in the scheduling, it indexes the order of operations, as the Figure 2 shows.



Head: [1 2 3 2 3 1 1 2 3]
Body: [3 0 4 5 1 7 2 6 8]

		"head"	"body"
Job	Oper.	Mach.	Order
1	1	1	3
	2	2	0
	3	3	4
2	1	2	5
	2	3	1
	3	1	7
3	1	1	2
	2	2	6
	3	3	8

Ordering *Jobs:* first by *Machines* and after by *Order*

Mach.	Order	Job	Oper.
1	2	3	1
	3	1	1
	7	2	3
2	0	1	2
	5	2	1
	6	3	2
3	1	2	2
	4	1	3
	8	3	3

Gant chart for the individual (in u.t).

Figure 2. Example of scheduling process from a given chromosome – phenotype×genotype (*source: own*).

4.3. Creating the initial population

Since the "head" must be built first, this is made using random numbers in the range of $[0,nm_{ij}-1]$, where nm_{ij} is the number of available machines to process operation j of the process i. The Figure 1 above shows this construction part, where resource allocation starts building the "body", an ordered 0 to nm_{ij-1} array.

4.4. Evaluation of the population

The function corresponding to the evaluation of the population, individual by individual, is the fitness function. Each of these values are the quantification of its adaptations. In other words, it means to apply the modified GT algorithm to make the active scheduling for each chromosome. The time completion can be the objective function of the search. Where: n = number of tasks; o_{ik} = operation k of *job* i; t_{ik} = time able to start operation k of *job* i; p_{ik} = processing time of the operation k belonging to the *job* i.

As follows, the modified GT algorithm schedulings:

Modified GT Algorithm:

Step 1: Place the first schedulable operation of each task (of the active planning horizon) in the set of candidate operations $C=\{o_{i1}|1\leq i\leq n\}$;

Step 2: Choose an operation o' of C, with earliest completion time;

Step 3: Determine the machine M', in which o' must be processed and thus build the set G (the conflict set of M'), consisting of all operations of C to be executed in M';

Step 4: Remove operations that do not start before o' finish, $G=\{o_{ik}\in G|t_{ik}<t'+p'\}$

Step 5: Run the sub- algorithm to select an operation o^*_{ik} of G;

Step 6: Remove o^*_{ik} from C, where $C=C\setminus\{o_{ik}\}$

Step 7: Insert the operation o^*_{ik} in the schedule and calculate start time;

Step 8: Insert the successor operation of the o^*_{ik} in the set C (if any);

Step 9: If $C\neq\varnothing$, go to Step 2, if not END.

End of sub-algorithm.

And, to calculate the step 5 (chosing the O^{*}_{ik} of G), the sub-algorithm is now presented:

Sub-algorithm Step 5

```
Create CR: set of operations using M', unscheduled, and with previous scheduled;

If $G \cap Cr \neq \{\}$ then
    Choose operation $G \cap Cr$ with higher priority index (RHP's)
Else

    Create PG: set of processes belonging to the G operations;

    Create CrG: subset of Cr, with the operations $\in$ to the processes of PG;

    If $CrG \neq \{\}$ then

        Find O: operation of CrG with higher priority index;

        Find J*: process that contains the task O;

        Find O*: operation of G belonging to J*;

    Else

        Find O*: oper. of G with low priority;

    End if;

End if;
```

End of sub-algorithm.

For the M' machine, if the operation is to be the one sequenced by thex "body" of the chromosome and the one with highest priority index, it will also be the operation O*, i.e., the candidate operation elected to be scheduled. Otherwise, if the sequence does not match with the job-shop problem, conflicts may occur.

Because there are two conflicting interests, both of them without the creation of unfeasible individuals must be considered. The goal is to find, in G, an operation that most closely matches to that suggested by the chromosome's "body", i.e., to the machine M', it is tried to schedule an operation of G which belongs to the same process of the highest indexed operation from Cr.

If no intersection of G with the operation indicated by the sequencing for M' exists, then the one with less priority is selected, relaxing in some choice criterion.

4.5. Selecting individuals

The selection process must list individuals, which will be part of the reproduction. The selection method adopted was proposed by Mayerle (1994), which consists in a ordered stochastic selection, having, in maximization, individuals in decreasing order according to their fitness, as follows).

$$Sel(R) = \left\{ r_j \in R \,\middle|\, j = m + 1 - \left\lceil \frac{-1 + \sqrt{4 \cdot rnd(m^2 + m)}}{2} \right\rceil \right\} \quad (5)$$

Where: R is the set of the *m* individuals; r_j is the j-th chromosome; *rnd* is random uniformly distributed $\in [0,1)$; $\lceil x \rceil$ is the smallest integer greater than x.

The method provides less selective pressure then Monte Carlo's selection (roulette), and also allows the best individuals to have more chance of crossing than the less able ones with the merit of recovering the super-individual effect, which will possibly exist due to the elitist strategy regarding the population formation.

4.6. Strategy to form a population

The population is created considering four different formation processes: cloning, random formation, greedy formation and reproduction. Because there is a sorting process, the best are cloned to the current population, but in a small fraction of the total population. Other small fraction is designed to individuals generated by the original algorithm of the first population. The fourth way of formation consists in generating a very small population part using dispatching rules (DRs) widely discussed in the literature, as: SPT, S/PRT, CHR and RND (shortest processing time, slack per remaining time, combined heuristic rule and random, respectively). The user also defines the number of individuals to compete for reproduction. This strategy is based on Gonçalves *et al.* (2005), whose intention is to avoid a premature convergence of the population.

4.7. Genetic operators applied to GA

The operators applied in the GA are: crossover, mutation and cloning.

Starting with the crossover operator, the main operator of genetic algorithm, perpetuates the characteristics of the fittest individuals through the exchange of parents' information, passing it to the offspring individuals. The application is different to the "head" and to the "body". To the first, their alleles are copied to the offsprings. For this, the uniform crossover is applied, based on a binary mask formed by 0/1 digits.

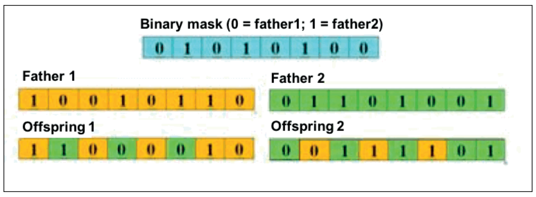


Figure 3. Example of the "mask" used into the uniform crossover operator (*source: own*).

In crossover process, a mask allele "zero" means, to the offspring, that the gene's donor is father 1. Otherwise, the donor is father 2. To another offspring, the reversal of the mask is required before the process. No harmonization is required.

After, individuals are subjected to the second phase of the crossing, now using order-based operators. This is necessary due to the desire to keep the sequence proposed by the parents as faithful as possible. The operator now is the PPX (precedence preservative crossover), acting in the chromosomes' "bodies" and using the masks involved in the previous step. Inheriting from his father 1 all the genes situated in his respective locus of the "zero" allele mask, it starts to complete the sequence based on the father 2. If the allele obtained from the father 2 already exists in some locus of offspring 1, it sought the next, until the gap can be completed.

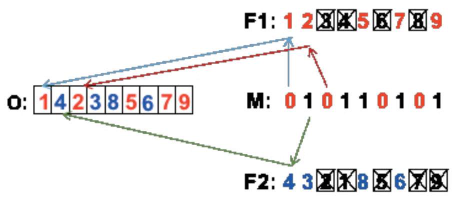


Figure 4. Example of the PPX crossover operator used to from a new offspring (*source: own*).

This operator was applied based on the observations of Gonçalves *et al.* (2005), such operators produce good effects when applied to schedules, instead of the traditional one or two cut points.

Moreover, during the crossover process, the mutation operator will actuate changing the value sampled by another one, randomly generated. In phenotype terms, it's a new route, meaning a new machine assignment to that operation.

In the "body", the mutation will exchange genes between two randomly selected points, forcing task sequence changes.

5. Applied tests

With interest in the analisys of how the algorithm solves the proposed problems types and their characteristics, the original data from Chan (2003) and Kumar *et al.* (2003) (Appendix I and II, respectively) were applied in the tests. Although, some modified problems proposed by Araujo (2006) were, also applied.

5.1. From simple routes or classic JSSPs

In these cases, the combined dispatching rule presented in Equation 4 contributes to form the route with specific criteria. This CHR is choosen based in a previous survey of Branco (2010) among others, when it is used in classic JSSPs with good results to these problems which have the characteristic of just one route to choose. Thus, the CHR is also applied to the NDD-SDST-JSSP presented in this work.

5.2. Expansion of alternative routes

In order to evaluate the algorithm ability with alternative routes combinations, Araujo (2006) expanded the original Chan (2003) problem (data in Appendix 1), introducing another machine and so, turning some task processes more flexible.

The average makespan with the proposed conjuncture was reduced, at minimum values from 931 ut to 786 ut, with average computational time close to 20 s using a dual core cpu with 2 Gbytes of RAM. It was expected, since the increase of machine number also increases the number of routes and thus the processing options. Araujo (2006), observing it has raised the issue that, since it increases the amount of routes and thus reduces the total processing time, reduces, at the same time, the frequency of use of machines.

5.3. Non-Deterministic Dynamic characteristics

For the tests involving NDD-JSSP (realistic) simulation routines in the original algorithm were introduced with the aim to generate random variables, and the simulated times are: instant start operation processing, transport time between machines, operation processing time in the machine. All these times are crucial for defining the end time of each operation, in each job.

The simulation of turbulent environment scheduling consists in to generate random variables with defined average and standard deviation. The average is equal to the original time and standard deviation will be between 5, 10 and 20%.

These indeterminisms created during the scheduling, they vary setup times (dependent), processing times, start and traveling time are important to evaluate the behavior of the solution NDD-JJSP proposed, since it reacts at the time when the change is detected, adapting itself.

Considering the objective function and the Chan (2003) original problem, the minimum and average data obtained for each case are: 960 and 990, 1013 and 1045, 995 and 1022, respectively, for the turbulence simulations with 5, 10 and 20%. It is noticed that they vary little from one class to another, according to the disturbances generated, which presents good scheduling in dynamic environments, as can be viewed in Table 1.

Table 1. Results of mean makespans from the solved problems, considering disturbances of 0, 5, 10 and 20% and 1 and 5% mutations rates.

	Mean makespan					
	Kumar		Chan		Chan (expand.)	
Disturb	1%	5%	1%	5%	1%	5%
0%	385	383	944	949	842	830
5%	431	432	997	996	859	840
10%	457	459	1049	1051	899	909
20%	527	523	1168	1167	1048	1032

In a non turbulent environment and, the makespan obtained by Chan (2003) is close to 1000 ut, Araujo (2006) a mean of 1070 ut and this work reached 931/944 of minimal/mean values. The same performance analysis is done to the Kumar *et al.* (2003) problem (Appendix 2), where the closeness of the makespans results is also observed, which were: 381, 424, 501 ut. The adopted conjuncture is robust with regard to schedules in turbulent environments. Araujo (2006) has obtained 410 ut for the same demand of pieces. Comparing and considering no disturbances, this work got a 350/383 ut of minimum/mean makespans values, meaning good response even with disturbances.

5.4. Test of dependent setup times

To evaluate the dependent setup times in the JSSPs, the problem in Table 2, is approached with the goal to analise the solutions to this classic problem with and without setup times.

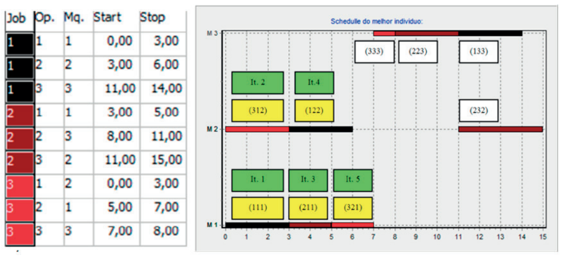
Table 2. Data from a randomly formed 3x3 JSSP (*source: Yamada and Nakano* (*1997*)).

Job	Oper.	Machine	Order	T_{START}	T_{STOPl}	P_{ij}
3	3	3	0	0	1	1
3	1	2	1	0	3	3
1	1	1	2	0	3	3
2	3	2	3	0	4	4
2	2	3	4	0	3	3
1	3	3	5	0	3	3
1	2	2	6	0	3	3
2	1	1	7	0	2	2
3	2	1	8	0	2	2

Unique of each chromosome, a processing order is established for each operation ("body"), as the machines that will process each task.

This test shows how the scheduler creates a agenda from a given chromosome and a table of dependent setup times among operations, where the scheduler can obtain a result as the Figure 5, with 15 ut to be finished.

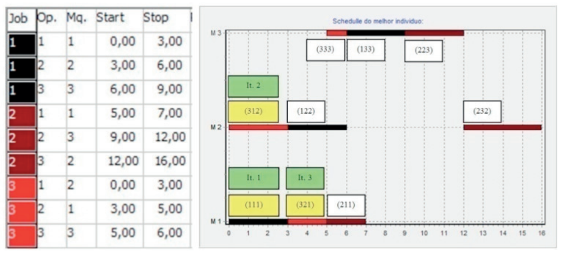
With a setup dependent time between operations (1,1,1) and (2,1,1), in the assigned sequence of the machine 1 (iteration 3), a second scheduling can be obtained (Figure 6).



Job	Op.	Mq.	Start	Stop
1	1	1	0,00	3,00
1	2	2	3,00	6,00
1	3	3	11,00	14,00
2	1	1	3,00	5,00
2	2	3	8,00	11,00
2	3	2	11,00	15,00
3	1	2	0,00	3,00
3	2	1	5,00	7,00
3	3	3	7,00	8,00

Figure 5. Example of solution of Yamada and Nakano 3×3 JSSP (*source: own*).

It is observed that the depend setup time made the scheduler choose, in its 3rd iteration, the operation (3,2,1), instead of (2,1,1). Being dependent on a predecessor operation, the processing time will increase from 2 ut to 5 ut, leading the the completion time to increase from 5 ut to 8 ut



Job	Op.	Mq.	Start	Stop
1	1	1	0,00	3,00
1	2	2	3,00	6,00
1	3	3	6,00	9,00
2	1	1	5,00	7,00
2	2	3	9,00	12,00
2	3	2	12,00	16,00
3	1	2	0,00	3,00
3	2	1	3,00	5,00
3	3	3	5,00	6,00

Figure 6. Example of solution of Yamada, Nakano 3×3 JSSP considering dependent setup time (*source: own*).

The scheduler has taken the best choice, considering de cost to accept, as in the Figure 3, the (2,1,1) after the (1,1,1). This choice saved 1 ut in this special case, so the goal was achieved when the scheduler did this exchange of tasks aimed at performing the best schedule, avoiding the time costs of the operations for the same order established by the same chromosome.

6. Discussions of results

This paper proposes a scheduling technique using combined heuristic rule (CHR) in schedules based in a modified version of the GT algorithm. The focus is to promote efficient scheduling in realistic job-shop problems (no determinism in times of operations), and, at the same time, consider the dependent setup times between operations in the same machine.

With the implemented algorithms in object-pascal language was possible to observe the general behavior in turbulent environments (varying time of operations process), with good scheduling capabilities without exceeding the proposed completion times. Also the good results found when compared to those obtained by other authors made further encouraging the implementation in future decision-making process.

The heuristic rule CHR was efficient in the test results, which usually get good solutions with time quite satisfactory, probably due to components based on rules: SPT and S/PRT. Regardless, the adopted situation for the genetic algorithm, which inserts individuals formed by other heuristics, could “dope” at low rates the population that was being built at each iteration.

The super-individual absence was important to the good performance of the GA, as a result of the selection proposed by Mayerle (1994).

With focus on problems with multiple route options, as the problems posed by Kumar *et al.* (2003) and Chan (2003), algorithms for route selection (using coding “head/body” of the chromosome and equipment tables available on the shop floor, as well as their processing times for each operation and setup times) and scheduling (inserted into the GA as the fitness function based on the modified algorithm GT), the results reached makespan, in some cases, better than the original solution problems. The expectations have not changed when the results were contrasted with those obtained by Araujo (2006), solved by other circumstances, also based on genetic algorithms.

Considering the problems demonstrates the proposal ability to organize, for a given planning horizon, the tasks to be scheduled, whether belonging to a physical or virtual manufacturing cell, which may form an alternate route, also considering dependent setup times.

About dependent setup times, the tests can show good results, exchanging operations in order to find best completion times, scapping from the costs imposed by the times among operations, respecting the order predefined by the chromosome.

References

Araujo, L. O. (2006). *Método de Programação de Sistemas de Manufatura do Tipo Job Shop Dinâmico Não Determinístico.* Tese (doutorado), Universidade de São Paulo, São Paulo.

Barboza, A. O. (2005). *Simulação e técnicas da computação evolucionária aplicadas à problemas de programação linear inteira mista*, Tese (doutorado), Universidade Tecnológica Federal do Paraná. Paraná.

Branco, R. M. (2010). *Agendamento de tarefas em sistemas de manufatura job-shop realista com demanda por encomenda: solução por algoritmo genético*, Tese (doutorado), Universidade Federal de Santa Catarina.

Chan, F. T. S. (2003). Effects of dispatching and routeing decisions on the performance of flexible manufacturing system. *International Journal of Advanced Manufacturing Technology*, 21(5), 328-338. doi:10.1007/s001700300038

Chiang, T-C., Fu, L-C. (2006). Using Dispatching Rules for Job Shop Scheduling with Due Date-based Objectives, *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, Orlando, Florida. doi:10.1109/ROBOT.2006.1641909

El-Bouri, A., Shah, P. (2006). A neural network for dispatching rule selection in a job shop. *The International Journal of Advanced Manufacturing Technology*, 31(3), 342-349. doi:10.1007/s00170-005-0190-y

Giffler, B., Thompson, G. (1960). Algorithms for solving production scheduling problems. *Operations Research,* 8(4), 487-503. doi:10.1287/opre.8.4.487

Goldbarg, M. C., Luna, H. P. L. (2000). *Otimização Combinatória e Programação Linear*. 2 Ed. Editora Campus.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. US: New York

Gonçalves, J. F., de Magalhães Mendes, J. J., Resende, M. G. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European journal of operational research*, 167(1), 77-95. doi:10.1016/j.ejor.2004.03.012

Herrmann, J. W., Lee, C.-Y., Hinchman, J. (1995). Global job shop scheduling with a genetic algorithm. *Production and Operations Management*, 4(1), 30-45. doi:10.1111/j.1937-5956.1995.tb00039.x

Jain, A. S., Meeran, S. (1998). *A state-of-the-art review of job-shop scheduling techniques*, Technical Report, Department of Applied Physics, Electronics and Mechanical Engineering, University of Dundee, Scotland.

Johnson, S. M., (1954). Optimal two and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1), 61-68. doi:10.1002/nav.3800010110

Kumar, R., Tiwari, M. K., Shankar, R. (2003). Scheduling of flexible manufacturing systems: an ant colony optimization approach. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 217(10), 1443-1453. doi:10.1243/095440503322617216

Linden, R. (2008). *Algoritmos genéticos*. Brasport, 2 edição. 2008.

Mauri, G. R., Lorena, L. A. N. (2006). Simulated Annealing Aplicado a um Modelo Geral do Problema de Roteirização e Programação de Veículos. *XXXVIII SBPO - Simpósio Brasileiro de Pesquisa Operacional*. Goiânia, GO.

Matsusaki, C. T. M. (2004). *Modelagem de Sistemas de Controle Distribuídos e Colaborativos De Sistemas Produtivos*. Tese (doutorado), Universidade de São Paulo, São Paulo. doi:10.11606/t.3.2004.tde-20122004-112454

Mayerle, S. F. (1994). *Um algoritmo genético para a solução do caixeiro viajante*. Tecnical Report, Departamento de Engenharia de Produção e Sistemas, UFSC.

Porter, K., Little, D., Peck, M., Rollins, R. (1999). Manufacturing classifications: relationship with production control systems. *Integrated Manufacturing Systems,* 10(3-4): 189-198. doi:10.1108/09576069910280431

Singh, A., Mehta, N. K., Jain, P. K. (2007). Multicriteria dynamic scheduling by swapping of dispatching rules. *International Journal of Advanced Manufacturing Technology*, 34(9), 988-1007. doi:10.1007/s00170-006-0674-4

Vázquez, M., Whitley, D. (2000). A comparison of Genetic Algorithms in solving the Dynamic Job Shop Scheduling Problem. *GECCO*. Available in https://docs.google.com/file/d/0B0xb4crOvCgTMXpKRnp6NFo3bTQ/edit

Wall, M. B. (1996). *A Genetic Algorithm for Resource-Constrained Scheduling*, Tesis, Departament of Mechanical Engineering, Massachusetts Institute of Technology.

Appendix I

Table I. Data from the original (without gray data) and modified problem of Chan (2003) (*source: Araujo*(*2006*)).

	Operation			
Job	1	2	3	4
1	M1 (105)	M3 (168)	M5 (70)	M2 (210)
	M2 (126)	**M4 (140)**	**M3 (140)**	**M1 (175)**
	M6 (91)	-	-	**M4 (175)**
2	M2 (140)	M3 (70)	M5 (245)	M4 (175)
	M3 (168)	M2 (112)	**M2 (266)**	**M5 (203)**
	M7 (154)	-	-	-
3	M5 (200)	M1 (125)	M4 (150)	M2 (75)
	M1 (100)	**M2 (60)**	M3 (135)	**M4 (90)**
	-	**M7 (60)**	-	-
4	M4 (150)	M2 (150)	M5 (100)	M3 (125)
	M2 (75)	**M5 (75)**	**M4 (50)**	M1 (75)
	-	-	**M6 (100)**	-
5	M1 (50)	M3 (100)	M2 (75)	M4 (150)
	M3 (40)	**M4 (150)**	**M5 (100)**	**M3 (125)**
	-	-	**M6 (125)**	-
6	M3 (175)	M2 (84)	M1 (175)	M5 (70)
	M5 (140)	**M4 (56)**	**M4 (140)**	M3 (161)
	-	**M5 (70)**	-	-
7	M4 (245)	M5 (70)	M1 (70)	M2 (105)
	M1 (266)	**M4 (126)**	M4 (105)	**M5 (170)**
	M3 (245)	-	**M7 (105)**	-
8	M5 (105)	M4 (280)	M3 (175)	M1 (140)
	M4 (70)	M5 (210)	**M2 (140)**	**M2 (56)**

Appendix II

Table II. Data of the problem proposed by Kumar *et al*. (2003) (*source: Araujo*(*2006*)).

	Operation			
Job	1	2	3	4
1	M1 (7)	M2 (3)	M1 (3)	M1 (2)
	M3 (4)	-	M3 (6)	M2 (4)
2	M1 (8)	M3 (4)	M1 (7)	M1 (8)
	M2 (12)	-	M2 (14)	M3 (4)
3	M1 (10)	M2 (2)	M1 (2)	M1 (6)
	M2 (15)	M3 (6)	M3 (4)	M2 (3)
	M3 (8)	-	-	-
4	M2 (9)	M1 (6)	M2 (7)	M1 (9)
	M3 (5)	M3 (2)	M3 (12)	M2 (6)
	-	-	-	M3 (3)
5	M1 (10)	M2 (7)	M1 (5)	M1 (4)
	M3 (15)	M3 (14)	M2 (8)	M2 (6)
	-	-	-	M3 (8)