

Initial populations tests for genetic algorithm flowshop scheduling problems solving with a special blocking

Christophe Sauvey, Nathalie Sauer

▶ **To cite this version:**

Christophe Sauvey, Nathalie Sauer. Initial populations tests for genetic algorithm flowshop scheduling problems solving with a special blocking. 13th IFAC Symposium on Information Control problems in manufacturing, Jun 2009, Moscou, Russia. pp.1961-1966, 10.3182/20090603-3-RU-2001.00330 . inria-00600379

HAL Id: inria-00600379
https://inria.hal.science/inria-00600379

Submitted on 6 Feb 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Preprints of the 13th IFAC Symposium on Information Control Problems in Manufacturing, Moscow, Russia, June 3 - 5, 2009

Initial populations tests for genetic algorithm flowshop scheduling problems solving with a special blocking

C. Sauvey*. N. Sauer.**

**LGIPM, University Paul Verlaine-Metz, Ile du Saulcy, Metz, F-57000 France (Tel: (+33)(0)3-8282-0608; e-mail: sauvey@ univ-metz.fr).*
*** LGIPM, University Paul Verlaine-Metz, COSTEAM, INRIA Nancy-Grand Est Metz, France (e-mail: sauer@univ-metz. fr)*

Abstract: In this paper, we consider a flowshop scheduling problem with a special blocking *RCb* (Release when Completing Blocking). This flexible production system is prevalent in some industrial environments. We propose genetic algorithms for solving these flowshop problems and different initial populations are tested to find the best adapted. Then, different parameters sets are tested in order to find the best for further genetic algorithms populations' selection. Best parameters set and results are endly discussed.

Keywords: Scheduling, Flowshop, Blocking, Genetic Algorithms, Initial populations.

1. INTRODUCTION

Flowshop scheduling problem (FSP) is a scheduling problem with a strong engineering background, in which jobs are processed by a series of machines in exactly the same order. It is one of the most widely studied scheduling problems. Most of the literature deals with the "permutation flowshop scheduling problem (PFSP)", whereby each job is processed by the same set of machines in the same order, in which the buffer between machines is considered to be of infinite capacity.

In a real industrial environment, if there is not enough space for carrying stock after one job has finished its operation on one machine and the following machine is not yet available, it is said to be "blocked". In the situation of classical blocking which we label here as "Release when Starting Blocking (*RSb*)", the machine remains blocked until this job starts on the next machine in the routing. Hall and Sriskandarajah (1996) showed that the flowshop scheduling problem with three machines and the classical blocking constraint is a strongly NP-complete problem.

In this paper, we are interested in a flowshop problem with particular blocking which exists specifically in certain industrial situations like waste treatment plants or aeronautics parts fabrications. This blocking, labelled as "Release when Completing Blocking (*RCb*)" has been introduced for the first time by Dauzère-Pérès *et al.* (2000). For flowshop with blocking *RCb*, the machine stays blocked until one job has finished its operation on the next machine and has left for the third one. The complexity of flowshop problem with blocking *RCb* has been proved to be polynomial when $m \leq 3$ and NP-hard when $m \geq 5$ (Martinez et al., 2006). The complexity of this problem while when $m = 4$ remains unsolved. This problem has been solved with a simulated annealing by Martinez et al. (2005) and with a metaheuristic Electromagnetism-like mechanism by Yuan and Sauer (2007).

In this paper, we focus on the application of a genetic algorithm (GA) to solve the considered problem. We are both interested in searching whether GA are well adapted to flowshop problems and whether a "good" initial population reinforce this trend. GA have been applied to different scheduling problems, see for example Della Croce et al. (1995) and Gonçalves et al. (2005) for job-shop problem; Chen et al. (1995), Reeves (1995), Murata et al. (1996) and Iyera and Saxena (2004), for flow-shop environment; Caraffa et al. (2001) and Wang et al. (2006) for blocking flow-shop; Oguz and Ercan (2005), for a hybrid flow-shop problem.

This paper is organized as follows. The precise definition of the problem is given in section 2. Details of GA are given in section 3. In section 4, four different initial populations are tested on the same conditions to find the best adapted one. Then, in section 5, we describe the way we came to the optimal parameters for the genetic algorithm with different trials on the same problem. Computational results and performance evaluation are presented in section 6. Finally, the last section concludes the paper and gives some perspectives to our work.

2. PROBLEM DESCRIPTION

A flowshop scheduling problem is considered where N jobs, each composed of M operations, must be processed non-pre-emptively on M machines. All jobs need to be processed in the same order on machines (i.e. routing does not differ from one job to another). No intermediate buffer space is available, and a machine can only execute one job at any time. Objective function consists in minimizing completion time, also called makespan.

Buffer space lack between machines implies that a machine which has completed the process of a given job can not release the job immediately. In classical blocking *RSb*, a machine remains blocked by a job until this job starts on next machine in the routing. An example with four machines with classical blocking constraints *RSb* is shown in Fig. 1. Time period when a machine remains blocked by a job until its operation on the downstream machine is finished is called blocking time ; dead time is a set period of time that a machine goes into a standby mode while the job to be processed has not yet finished its operation on the former machine.

In the particular type of blocking constraints *RCb* considered in this paper, the machine remains blocked by a job until its operation on the downstream machine is finished and it leaves the machine (see Fig. 2).

In regard to classical flowshop problems, *RCb* blocking induces supplementary blocking time, which we have defined as differential blocking time. In this paper, we define two different definitions of differential blocking time whether it comes before occurrence of J_k on machine M_r, or after.

Differential blocking time "after" job J_k on machine M_r is due to insufficient execution time of preceding job in the routing on machine M_{r+2}. Differential blocking time "before" job J_k on machine M_r is due to insufficient execution time of job J_k on machine M_{r-1} in front of execution time of preceding job in the routing on machine M_{r+1}.

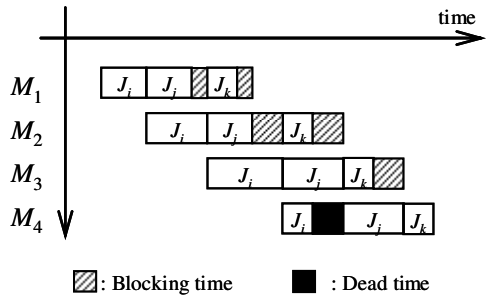


Fig. 1. Four machines problem with classical blocking *RSb*.

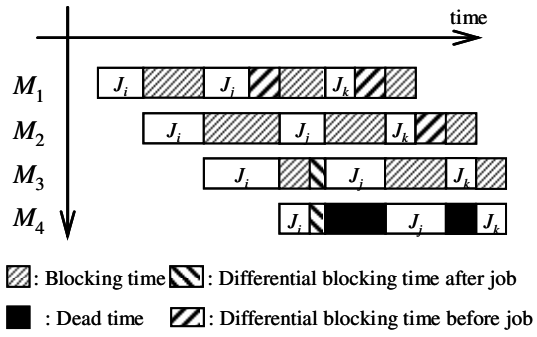


Fig. 2. Four machines problem with particular blocking *RCb*.

RCb constraint is practical in some industrial environments, such as waste treatment industry and aeronautics part fabrication (Martinez, 2005). Considering waste treatment industries as an example, a company receives different types of industrial and farm waste to be treated. The waste are brought by trucks and unloaded into silos. These silos are considered as the parallel machines of the first step. The cargo is then treated by one and only one machine, which is considered to be the machines of the second step. As the products are processed from silo to mixer, the silo cannot be available until the process on the mixer of the totality of waste is finished.

3. GENETIC ALGORITHM HEURISTIC (GA)

3.1 Genetic Algorithm Description.

Genetic algorithms are optimisation algorithms based on genetic derived techniques and nature evolution mechanisms (crossing, muting, selections, etc.) and belong to evolutionary algorithms class. Use of genetic algorithms on combinatory problems solving has come with John Holland's works on adaptive systems in 1962 (Holland, 1962). Since this date, this optimisation method has proved its efficiency and its fields of application have ever widely spread (Holger and Stützle, 2005), (Siarry and Michalewicz, 2007). The mechanism consists in evolving, from an initial population, a set of spatial points towards one or more optimal points of the optimisation problem treated.

3.2 Opportunity of GA to Flowshop problems solving.

Genetic algorithms seem to be very well adapted to solve scheduling problem, particularly flowshop problems because of their structure, directly and canonically translating an individual into a possible solution. Indeed, the individuals' genes number is directly equal to the number of jobs to be scheduled, and for example the 6-genes-individual (6-2-3-1-5-4) directly represents the possible scheduling solution ($J_6 - J_2 - J_3 - J_1 - J_5 - J_4$). When no ordering constraints have to be respected, this canonical application between a sequence and an individuate can be taken and we can apply the genetic algorithm synoptic presented in figure 3.

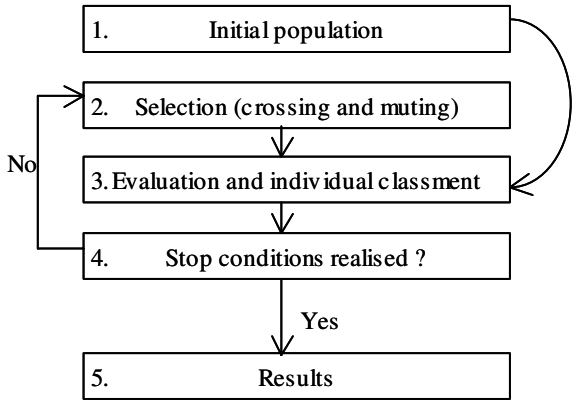


Fig. 3. Genetic algorithm presentation.

An initial population, corresponding to a solutions set, is randomly generated and directly evaluated. Evaluation consists in our problem to calculate each individual's

makespan to sort them. Then, a selection is operated on individuals in order to determinate which individuals are likely to give the best possible results. Simple crossing and mutation operators have been developed. This process is similar to natural selection process, where most adapted individuals win reproduction competition whereas less adapted die before reproduction.

In our genetic algorithm, we give possibility to impose best individuals keeping percentage (*pc_best*), new randomly generated people introduction percentage (*pc_new*) and crossing percentage (*pc_cross*). Crossing step has for goal to recombine two parents with producing children who inherits some of parental characteristics. Individuals selected for crossing are taken among preceding generation's best individuals and new randomly generated people. Remaining percentage of the created generation is created by mutation. Mutation is a random alteration of an individual's genes. In our algorithm, we select for mutation among best individuals, in order to see whether they ever improve their good results.

As represented on fig.3, loop composed of steps 2 and 3 is repeated as long as step 4 allows to. To perform our simulations, we chose to stop when the population does not evolves quickly enough any more or stops evolving. The population is then homogeneous and we can hope that it is near to optimum(s). The criteria we retained to stop our simulations is the counter of times the best value was identical (*ctbvi*).

In order to accelerate problem solving, the idea of giving very quickly a first solution to each problem would give initial populations quite adapted. We hope that an initial population already adapted to the treated problem is better than a normal one.

4. BEST INITIAL POPULATION SELECTION

Presented works had been performed to see different initial populations effect on final results of genetic algorithm solving.

In order to determine whether an initial population "correctly chosen" at the first step of genetic algorithm would give better results, we proposed to generate four different initial populations. They have been chosen for their dispersion, their adaptability to the problem, their pre-selection among a crowd, or no-specifically-chosen (random population).

Population A: The first population has been randomly generated, without any other specification.

Population B: It has been generated in order to spread the population the most widely over solutions space. The norm that we used for this dispersion is the square of the canonical distance between two individuals. The square root usually taken with this norm has been avoided in order to merely treat integers.

$$\mathrm{Dist}(IndA, IndB) = \sum_{i=1}^{nb_genes} (IndA_i - IndB_i)^2$$

where:

– $IndA_i$ (resp. $IndB_i$) is the i^{th} gene of individual A (resp. B);
– *nb_genes* is the number of individuals in a population.

Population dispersion coefficient is defined as the minimal distance value between two population individuals. With this definition, two individuals can be responsible of the whole population dispersion coefficient. In order to correctly spread the population over the solutions space, we developed the algorithm 1 to obtain an important dispersion coefficient.

Algorithm 1.
Give an initial population
Repeat
  Calculate all the distances between all individuates of the population
  Take off the population one of the individuals (they are 2 at least) which have the minimal distance.
  Take off the population one of the most distant individuals.
  Nb-Iter = 0
  Do
    Randomly generate two new individuals.
    Calculate the minimal distance between all population individuates and new individuals.
    Nb-Iter = *Nb-Iter* +1
  While ((minimal distance ≤ population dispersion coefficient) and (*Nb-Iter* < *Nb-Iter-Max*))
  Memorise the new population
Until minimal distance between two individuals is sufficient enough.

With this algorithm, we are sure to make always population involve in spreading sense, precisely because of the stopping condition in the "do…while loop". At the next loop turn, the "≤" gives insurance that population dispersion coefficient will not be obtained with the two just generated individuals, but with two others. Thus, we can afford that each loop turn is efficient, and that global population dispersion increases.

Population C: It has been generated with a special heuristic, specially adapted for each specific problem treated. The developed heuristic for population C starts from the observation of Fig. 2. The algorithm is the following:

Algorithm 2.
C_{max} = Infini
For i = 1 to N do
  Take J_i as the first job of this new sequence σ_i.
  Repeat
    Research the best job to put behind. The best job is the one which gives the least sum of differential blocking times when it is put just after the last job in the partial sequence σ_i.
    Put this job at the end of the partial sequence σ_i.
  Until (all the jobs have been placed in the sequence)
  C_{max1} = makespan of this new sequence σ_i
  If ($C_{max1} < C_{max}$) then
    $\sigma^* = \sigma_i$
    $C_{max} = C_{max1}$
  EndIf
EndFor

Thus, N solutions σ_i are constructed, each beginning with a different job, because it is very difficult to determine the best beginning job. Population C is the result of this heuristic. Results obtained with best individuate σ^* are presented on column 4 of Table 7 under the name HCS1.

Population D: To generate population D, for each treated problem, a ten times bigger population is randomly triggered, from which the ten best percents are retained as initial population D.

To evaluate heuristics characteristics, we dispose of a hundred problems benchmark and their associated lower bounds or optimal results given in (Martinez, 2005). We have considered two configurations of problems: 20 jobs and 5 machines and 50 jobs and 5 machines. For each configuration, 100 problems have been considered. Three trials have been performed for each initial population type (A, B, C and D) for each *ctbvi* value and the twelve results for all of the hundred problems have been calculated. Mean values are presented in Tables 1 and 2.

Table 1 and Table 2 contain for each initial population the relative performance [(GAmakespan – *LB*)/*LB**100], where *LB* is the lower bound given by Martinez (2005). These percentages are given for three values of *ctbvi* (counter of times the best value was identical) and for parameters determined in paragraph 5.

Table 1. Results of different types of population for problems with 20 jobs and 5 machines

Ctbvi	Pop. A	Pop. B	Pop. C	Pop. D
50	8.58%	**8.57%**	8.58%	8.58%
100	8.10%	**8.06%**	8.11%	8.11%
200	**7.77%**	7.83%	7.83%	7.80%

Table 2. Results of different types of population for problems with 50 jobs and 5 machines

Ctbvi	Pop. A	Pop. B	Pop. C	Pop. D
50	11.84%	11.71%	11.43%	**11.40%**
100	**9.99%**	10.02%	10.04%	10.07%
200	9.05%	**9.04%**	9.22%	9.15%

More than confirmation that solution quality improves with *ctbvi*, this table proves that initial population is not helpful at improving genetic algorithm solutions quality in identical working conditions.

5. BEST SELECTION PARAMETERS SET FOR GA

The selection program we developed let possibility of choosing three parameters, defining all the population selection for the next generation: best individuals keeping percentage (*pc_best*), new individuals introducing percentage (*pc_new*) and crossing among best and new individuals percentage (*pc_cross*). When the sum of these three terms is lower than 1, difference corresponds to mutation among best individuals percentage.

In this section, we describe the way we came to genetic algorithm optimal parameters set with different trials on the same problem.

5.1 Selection parameters set for a 20 genes population.

We performed simulations in order to determine the best parameters set in terms of solution accuracy when all other parameters (problem, stop conditions) where equals. We performed simulations first on a 20 jobs, 5 machines set of 100 problems. For each problem solving, GA has been launched three times per initial population (A, B, C or D). Each population treated was a 100 20-genes-individual's population. Average results given on tables are the relative performance compared to lower bound available in (Martinez, 2005). Computational times are not given because they are always inferior to one second.

For the first simulation round, *pc_best* varied between the four values 0.05, 0.10, 0.15 and 0.20; *pc_new* varied between 0.00, 0.10, 0.20 and 0.30 and *pc_cross* varied between 0.10, 0.20, 0.30, 0.40 and 0.50.

We chose to present the problem with 20 jobs and 5 machines with these 80 parameters combinations in order to find the best. The comparison has been made on 3 different stopping conditions (*ctbvi* = 50, 100 and 200, *ctbvi* stands for Counter of Times when Best Value is Identical). Best parameters sets are identical on the three respective tests. Results presented on Table 3 have been calculated for *ctbvi* = 200. On lines are given *pc_best*/*pc_new* couples and on rows *pc_cross* values. On each case is given the average of all the twelve values obtained with three trials for all of four initial populations. For the 20 genes population, best parameters set found is (*pc_best* = 0,15; *pc_new* = 0,10; *pc_cross* = 0,50) for the three tested stopping conditions (*ctbvi* = 50, 100 and 200).

Table 3. Simulation results for problems with 20 jobs and 5 machines and *ctbvi*=200.

First round		*pc_cross*				
pc_best	*pc_new*	**0.10**	**0.20**	**0.30**	**0.40**	**0.50**
0.05	**0.00**	9.4	9.8	10.0	9.6	9.6
	0.10	8.7	8.4	8.1	8.0	**7.9**
	0.20	8.8	8.5	8.3	8.2	8.1
	0.30	8.9	8.7	8.5	8.3	8.2
0.10	**0.00**	8.8	8.7	9.0	9.1	9.1
	0.10	8.4	8.2	8.1	8.0	**7.9**
	0.20	8.4	8.3	8.3	8.2	8.4
	0.30	8.8	8.4	8.3	8.1	8.0
0.15	**0.00**	9.1	9.2	9.3	9.4	9.3
	0.10	8.4	8.1	8.0	7.9	**7.9**
	0.20	8.3	8.2	8.0	8.0	**7.9**
	0.30	8.5	8.3	8.2	8.1	8.1
0.20	**0.00**	8.4	8.4	8.6	8.8	9.0
	0.10	8.4	8.3	8.2	8.3	8.2
	0.20	8.5	8.3	8.2	8.1	8.0
	0.30	8.4	8.2	8.1	8.0	8.0

For *pc_best*, there are three local optima around the values (*pc_best* = 0.10; 0.15 and 0.20). We can clearly see on Table 3 that best results are obtained on border of definition domain. This is the reason why we propose these 3 values for the second round of simulations (Table 4).

Results are very bad with *pc_new* = 0.00. Optimal parameters set contains *pc_new* = 0.10. In all other parameters sets, occurrence of *pc_new* = 0.10 makes a local optima for results. For these three reasons, we propose 0.05 and 0.15 as new possible values of *pc_new*.

As the best value for *pc_cross* is on definition domain border, we propose all the possible values between 0.50, 0.60, 0.70 and 0.80 in function of the sum of other percentages. Indeed, sum of all percentages, mutation percentage included, must be equal to one. So, for example, for the couple of values (*pc_best* = 0.15; *pc_new* = 0.10), 0.70 is the maximal possible value for *pc_cross*, mutation percentage being consequently equal to 0.05.

Second simulation round results are presented on Table 4. They confirmed first simulation round and precise that best parameters set is endly (*pc_best* = 0.15; *pc_new* = 0.15; *pc_cross* = 0.60 or 0.70).

Table 4. Simulation results for problems with 20 jobs and 5 machines and *ctbvi*=200.

Second round		pc_cross			
pc_best	pc_new	0.50	0.60	0.70	0.80
0.10	**0.05**	7.9	7.9	7.8	7.9
	0.10	7.9	7.9	8.0	8.1
	0.15	7.9	8.0	8.1	///
	0.20	8.3	8.1	8.1	///
0.15	**0.05**	7.9	7.9	7.9	7.9
	0.10	7.9	7.9	**7.8**	///
	0.15	7.9	**7.8**	**7.8**	///
	0.20	8.0	7.9	///	///
0.20	**0.05**	8.0	8.0	8.0	///
	0.10	7.9	7.9	7.9	///
	0.15	7.9	7.9	///	///
	0.20	8.0	8.1	///	///

5.2 Selection Parameters set for a 50 genes population.

We performed then the same 3 x 80 simulations on a 50 jobs, 5 machines set of 100 problems. The population treated was then a 100 50-genes-individual's population. Genes number variation could be expected to change the best parameters set.

Results of first simulations round are quite similar to 20-genes simulations results (Table 5). For the 50 genes population, the best parameters set is (*pc_best* = 0.10; *pc_new* = 0.10; *pc_cross* = 0.50) for the three tested stopping conditions (*ctbvi* = 50, 100 and 200). Even if optimal *pc_best* value comes from 0.15 to 0.10, the optimal genetic algorithm parameters set turns around the same values for a 20 or a 50 genes problem. On Tables 5 and 6, computational time is given between brackets (in seconds) behind results. We can observe its variation in terms of *pc_new* and *pc_cross*.

We investigated towards higher *pc_cross* because 9.5 values on first round investigation domain border are rounded with similar results and because optimal parameters set had been obtained in this area for a 20-genes population. Second range of simulations has been performed with exactly the same values for *pc_best*, *pc_new* and *pc_cross*. Table 6 presents these results.

Results of this second range of simulations clearly confirm our first analysis. The best parameters set for a 50-genes problem is (*pc_best* = 0.10; *pc_new* = 0.10; *pc_cross* = 0.70) (Table 6). We can also notice that optimal results are obtained with a reasonable computational time compared to maximum.

Table 5. Simulation results for problems with 50 jobs and 5 machines and *ctbvi*=200.

First round		*pc_cross*				
pc_best	*pc_new*	0.10	0.20	0.30	0.40	0.50
0.05	**0.00**	12.4(0)	12.3(0)	12.2(0)	12.3(0)	12.5(0)
	0.10	11.5(0)	10.4(1)	10.0(1)	9.6(2)	9.7(2)
	0.20	13.1(1)	12.1(1)	11.4(2)	11.2(2)	10.6(2)
	0.30	11.0(1)	10.7(2)	10.6(2)	10.0(3)	10.1(3)
0.10	**0.00**	11.7(0)	11.7(0)	11.7(0)	11.7(0)	11.7(0)
	0.10	10.6(0)	10.2(1)	9.9(1)	9.9(1)	**9.5(2)**
	0.20	10.6(1)	10.2(1)	10.0(1)	10.0(2)	**9.5(2)**
	0.30	10.6(1)	10.4(2)	10.2(2)	9.8(2)	9.8(3)
0.15	**0.00**	12.6(0)	12.0(0)	11.9(0)	11.8(0)	11.7(0)
	0.10	10.8(0)	10.3(0)	11.2(0)	9.8(1)	9.8(1)
	0.20	10.6(1)	10.2(1)	9.9(1)	9.8(1)	9.7(2)
	0.30	10.5(1)	10.3(1)	10.1(2)	10.0(2)	9.8(3)
0.20	**0.00**	11.4(0)	11.4(0)	11.4(0)	11.7(0)	12.0(0)
	0.10	10.9(0)	10.3(0)	10.0(1)	9.8(1)	9.6(1)
	0.20	10.5(1)	10.5(1)	9.9(1)	9.7(2)	9.6(2)
	0.30	10.8(1)	10.3(1)	9.9(2)	9.7(2)	10.9(3)

Table 6 Simulation results for problems with 50 jobs and 5 machines and *ctbvi*=200.

Second round		pc_cross			
pc_best	pc_new	0.50	0.60	0.70	0.80
0.10	**0.05**	9.8 (1)	9.6 (1)	9.2 (2)	9.2 (2)
	0.10	9.7 (2)	9.3 (2)	**9.1 (2)**	9.9 (3)
	0.15	9.4 (2)	9.3 (2)	9.5 (3)	///
	0.20	9.5 (2)	9.6 (3)	9.8 (4)	///
0.15	**0.05**	9.7 (1)	9.5 (1)	9.7 (1)	10.0 (2)
	0.10	9.5 (1)	9.4 (2)	9.5 (2)	///
	0.15	9.5 (2)	9.3 (2)	9.7 (3)	///
	0.20	9.5 (2)	9.4 (3)	///	///
0.20	**0.05**	10.4 (1)	10.0 (1)	9.9 (1)	///
	0.10	9.6 (1)	9.6 (2)	10.0 (2)	///
	0.15	9.5 (2)	9.4 (2)	///	///
	0.20	9.8 (2)	10.0 (3)	///	///

5.3 Conclusion.

After having performed simulations on 20-genes and 50-genes individuals' populations, we have succeeded in finding best set of parameters for each of them. Results show that, for our particular problem, best efficiency is obtained with (*pc_best* = 0.10; *pc_new* = 0.10; *pc_cross* = 0.70) in all of two cases. This is the reason why we will continue to develop our genetic algorithm with this particular parameters set. Global results of genetic algorithm on all benchmark problems are presented on Table 7 under the GA row.

6. COMPARISON OF EXPERIMENTAL RESULTS

In this paper, we tested the GA heuristic with population C and optimal parameters set on the same benchmarks used by Martinez (2004, 2005). As genetic algorithm is a stochastic algorithm, it has been launched 6 times per problem with *ctbvi*=5000. Best results out of 6 tries and total computational time are presented in Table 7.

We compared the results with those of heuristic NEH (Nawaz, Enscore and Ham, 1983) adapted to flowshop problem with blocking *RCb* by Martinez et al. (2005), referred to here as heuristic (MNEH), Simulated Annealing (SA) with the best neighbourhoods given in Martinez et al. (2004) and ElectroMagnetism like optimisation heuristic (EM) tested by Yuan and Sauer (2007). The computational results are given in Table 7.

In the 3rd column, we show relative performance of the algorithm MNEH compared to the optimal. This column is directly comparable to column 4 where population C solutions are presented. Columns 5 to 6 contain relative performance of Simulated Annealing compared to optimal solutions (for problems with 5 to 10 jobs) and to lower bound (for problems with 20 to 50 jobs). Average time (in second) of this method is reported in column 6. EM results and time are presented on column 7 and 8. Columns 9 and 10 present respectively GA heuristic results and execution time. Table 7 proofs that genetic algorithms are well adapted to flow shop problems, especially for big-size problems, because obtained solutions for 50 jobs are very good in front of results obtained with simulated annealing. Even if electromagnetism like heuristic is quite efficient, genetic algorithm give good results in a shorter time.

Table 7. Computational results comparison with average computation times and *ctbvi*=5000 for GA.

N	M	MNEH	HCS1	SA		EM		GA	
				%	Time	%	Time	%	Time
5	5	0.99	5.34	**0**	0.1	**0**	2	**0**	0.05
	10	0.69	5.4	**0**	0.2	**0**	3.2	**0**	1
	20	0.38	3.95	**0**	0.3	**0**	4.0	**0**	1.63
10	5	2.64	6.49	0.39	6.1	**0**	3.0	**0**	1.03
	10	2.38	8.21	0.25	18	**0**	8.0	**0**	2.04
	20	2.61	7.71	0.13	39	**0**	17	**0**	3.9
20	5	9.19	11.48	7.16	364	6.09	117	**5.49**	31
	10	14.73	19.95	12.41	364	**12.03**	136	12.43	46
	20	16.29	25.33	14.41	364	**13.99**	146	16.71	86
50	5	9.48	9.62	7.48	305	9.49	388	**5.83**	198
	10	19.45	19.3	16.68	1398	**12.93**	389	14.77	223
	20	24.71	26.57	22.20	4828	22.72	483	**20,85**	376

7. CONCLUSIONS

We have tested different initial populations for a flowshop scheduling problem resolution with a genetic algorithm, under special RC blocking conditions, usually encountered in industrial problems. Four different initial populations have been tested and no real particular influence has occured. Then, for the treated problem, the best functioning point of our genetic algorithm has been found in terms of its four parameters. Our results have then be compared to former existing results, and thus have shown validity and reliability of our resolution process.

Our future works will deal with genetic algorithm amelioration on the same problems, especially in terms of population selection policy. We will particularly work on genes recurrent associations' location among best individuals. Local amelioration techniques could also benefit to genetic algorithm results. Future works will begin to verify whether these techniques are significantly improving results.

REFERENCES

Caraffa, V., Ianes, S., Bagchi, T.P. and Sriskandarajah, C. (2001). Minimizing makespan in a blocking flowshop using genetic algorithms, *International Journal Production Economics*, 70, 101–115.

Chen, C.L., Vempati, V.S. and Aljaber, N. (1995). An application of genetic algorithms for flowshop problems, *European Journal of Operational Research*, 80, 389-96.

Dauzère-Pérès, S., Pavageau, C., Sauer, N. (2000). Modélisation et résolution par PLNE d'un problème réel d'ordonnancement avec contraintes de blocage, 3ème congrès de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'2000), 216-217, Nantes, France.

Della Croce, F., Tadei, R., and Volta, G. (1995). A genetic algorithm for the job shop problem, *Computers and Operations Research*, 22, 15–24.

Gonçalves, J.F., Magalhães Mendes (de), J.J. and Resend, M.G.C. (2005). A hybrid genetic algorithm for the job shop scheduling problem, *European Journal of Operational Research*, 167(1), 77-95.

Hall, N.G. and Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process, *Operations Research*, 44, 510-525.

Holger H.H. and Stützle T. (2005). Stochastic Local Search: Foundations and Applications, Morgan Kaufmann, San Francisco, CA, USA.

Holland J.H. (1962). Outline for logical theory of adaptive systems, *Journal of the association of computing machinery*, 3, 297-314.

Iyer, S. K. and Saxena, B. (2004). Improved genetic algorithm for the permutation flowshop scheduling problem, *Computers and Operations Research*, 31, 593–606.

Ling Wang, Liang Zhang, Da-Zhong Zheng, (2006). An effective hybrid genetic algorithm for flow shop sheduling with limited buffers, *Computers & Operations Research,* 33, 2960-2971.

Martinez, S., Guèret C., and Sauer, N. (2004). Simulated-annealing for the flow-shop problem with blocking constraint, EUROXX, p. 27.

Martinez., S. (2005). Ordonnancement de systèmes de production avec contraintes de blocage, *Ph.D thesis*, IRCCyN, Nantes, France.

Martinez, S., Dauzière-Pérès, S., Guèret, C., Mati, Y. and Sauer, N. (2006). Complexity of flowshop scheduling problems with a new blocking constraint. *European Journal of Operational Research*, 169(3), 855-864.

Murata T, et al. (1996). Genetic algorithm for flowshop scheduling problems. *Computers and Industrial Engineering*, 30(4), 1061–71.

Nawaz, M., Enscore, E., and Ham I., (1983). A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. OMEGA, *The International Journal of Management Science*, 11 (1), 91-95.

Oguz, C. and Ercan, M.F. (2005). A Genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks, *Journal of Scheduling*, 8, 323–351.

Reeves, C.R. (1995). A genetic algorithm for flowshop sequencing, *Computers and Operations Research*, 22, 5-13.

Siarry P. and Michalewicz Z. (2007). Advances in Metaheuristics for Hard Optimization, Springer-Natural Computing Series.

Yuan, K. and Sauer, N. (2007). Application of EM algorithm to flowshop scheduling problems with a special blocking, ISEM'07.