Experimental Genetic Operators Analysis for the Multi-objective Permutation Flowshop

Carlos A. Brizuela and Rodrigo Aceves

Computer Science Department, CICESE Research Center
Km 107 Carr. Tijuana-Ensenada, Ensenada, B.C., México
{cbrizuel, raceves}@cicese.mx, +52–646–175–0500

Abstract. The aim of this paper is to show the influence of genetic operators such as crossover and mutation on the performance of a genetic algorithm (GA). The GA is applied to the multi-objective permutation flowshop problem. To achieve our goal an experimental study of a set of crossover and mutation operators is presented. A measure related to the dominance relations of different non-dominated sets, generated by different algorithms, is proposed so as to decide which algorithm is the best. The main conclusion is that there is a crossover operator having the best average performance on a very specific set of instances, and under a very specific criterion. Explaining the reason why a given operator is better than others remains an open problem.

1 Introduction

Optimization problems coming from the real world are, in most cases, multi-objective (MO) in nature. The lack of efficient methodologies to tackle these problems makes them attractive both theoretically and practically.

Among these real-world MO problems, scheduling (a combinatorial one) seems to be one of the most challenging. In a real scheduling problem we are interested not only in minimizing the latest completion time (makespan) but also in minimizing the total time all jobs exceed their respective due dates as well as the total time the jobs remain in the shop (work-in-process). A few results on MO scheduling with a single-objective-like approaches reveal that there is much to do in this research area.

On the other hand, the available methodologies in genetic algorithms (GA's) have been focused more on function optimization rather than in combinatorial optimization problems (COP's). In order to fairly compare the performance of two given algorithms some metrics have been proposed. Recent results [17], [21] have shown that these metrics may mislead conclusions on the relative performance of algorithms. This situation forces us to find appropriate procedures to fairly compare two non-dominated fronts, at least in terms of dominance relations.

The remainder of the paper is organized as follows. Section 2 states the problem we are dealing with. Section 3 reviews some available results for this problem. Section 4 introduces the global algorithm used in the operators study. Section 5

explains the experimental setup and introduces the proposed performance measure. Section 6 shows the experimental results. Finally, section 7 presents the conclusions of this work.

2 Problem Statement

Let $\mathbf{K}_n$ be the set of the n first natural numbers, i.e. $\mathbf{K}_n = \{1, 2, \cdots, n\}$. The permutation flow-shop problem consists of a set $\mathbf{K}_n$ of jobs ($n > 1$) that must be processed in a set of machines $\mathbf{K}_m$ ($m > 1$). Each job $j \in \mathbf{K}_n$ has m operations. Each operation O_{kj}, representing the k-th operation of job j, has an associated processing time p_{kj}. Each machine must finish the operation once it is started to be processed (no preemption allowed). No machine can process more than one operation at the same time. No operation can be processed by more than one machine at a time. Each job j is assigned a readiness time r_j, and due date d_j. All jobs must have the same routing through all machines. The goal is to find a permutation of jobs that minimizes a given objective function (since the order of machines is fixed).

In order to understand the objective functions we want to optimize we need to set up some notation first. Let us denote the starting time of operation O_{kj} by s_{kj}. Define the permutations $\pi = \{\pi(1), \pi(2), \cdots, \pi(n)\}$ as a solution to the problem.

With this notation a feasible solution must hold the following conditions:

$$s_{kj} > r_j \quad \forall k \in \mathbf{K}_m, j \in \mathbf{K}_n \ , \tag{1}$$

$$s_{k\pi(j)} + p_{k\pi(j)} \leq s_{(k+1)\pi(j)} \forall k \in \mathbf{K}_{m-1}, j \in \mathbf{K}_n \ . \tag{2}$$

All pairs of operations, of consecutive jobs, processed by the same machine k must satisfy:

$$\begin{gathered} s_{k\pi(j)} + p_{k\pi(j)} \leq s_{k\pi(j+1)} \\ \text{for every machine } k \in \mathbf{K}_m, \text{ and every job } j \in \mathbf{K}_{n-1} \ . \end{gathered} \tag{3}$$

Now we are in position of defining the objective functions. First we consider the makespan, which is the completion time of the latest job, i.e.

$$f_1 = c_{max} = s_{m\pi(n)} + p_{m\pi(n)} \ . \tag{4}$$

The mean flow time, representing the average time the jobs remain in the shop, is the second objective.

$$f_2 = \overline{fl} = (1/n) \sum_{j=1}^{n} fl_j \ , \tag{5}$$

where $fl_j = \{s_{mj} + p_{mj}\} - r_j$, i.e. the time job j spends in the shop after its release. The third objective is the mean tardiness, i.e.

$$f_3 = \overline{T} = (1/n) \sum_{j=1}^{n} T_j \ , \tag{6}$$

where $T_j = \max\{0, L_j\}$, and $L_j = s_{mj} + t_{mj} - d_j$.

Thus, we have the following MO problem:

$$\begin{aligned} &\text{Minimize } (f_1, f_2, f_3) \\ &\text{subject to } (1) - (3) \ . \end{aligned} \tag{7}$$

This problem, considering only the makespan (C_{max}) as the objective function, was proven to be NP-hard [11]. This implies that the problem given by (7) is at least as difficult as the makespan problem.

2.1 Tradeoff among objectives

To show that "conflict" among objectives exists, it is necessary to define what "conflict" means. We give some definitions in order to understand why our problem should be treated as a multi-objective one.

Definition 1. The *Ideal Point or Ideal Vector* is defined as the point $\mathbf{z}^*$ composed of the best attainable objective values. This is,

$z_j^* = \max\{f_j(\mathbf{x}) | \mathbf{x} \in A\}$, A is the set (without constraints) of all possible $\mathbf{x}$, $j \in \mathbf{K}_q$, where $q > 1$ is the number of objective functions.

Definition 2. *Two objectives are in conflict* if the Euclidean distance from the ideal point to the set of best values of feasible solutions is different from zero.

Definition 3. *Three or more objectives are in conflict* if they are in conflict pairwise.

To show that the objectives are in conflict we are going to construct a counter-example instance where we can enumerate all solutions and see that there is not a single solution with all its objective values being better than or equal to the respective objective values of the other solutions. This implies that the distance from the set of non-dominated solutions to the ideal point is positive.

Table 1 presents an instance of the PFSP with three jobs and two machines. All solutions for this problem are enumerated in Table 2, here we can see that solution 312 has the best makespan value, 231 the best tardiness, and 213 the best mean flow time. The ideal point $\mathbf{z}^*$ for this instance is given by $\mathbf{z}^* = (44, 16, 87)$. Therefore, the Euclidean distance from the set of values in Table 2 to the ideal point is positive.

It is not known whether or not, in randomly generated PFSP instances, it is enough to consider only two objectives, as the third one, in the Pareto front, will have the same behavior as one of the first two. We did not perform any experimental study regarding the answer for this question because of the required computational effort. However, in the experiments section we will see that for all generated instances there are two objective functions which are not in conflict. It will be very interesting to know which instances will have three conflicting objectives and which will not.

Table 1. Problem data for a 3-jobs 2-machines PFSP

Job (j)	t_{j1}	t_{j2}	Due Date (d_j)
1	17	8	30
2	5	4	15
3	15	11	30

Table 2. Counter-example for the PFSP

Π	f_1	f_2	f_3
321	45	30	101
312	44	39	110
231	45	16	105
213	48	18	87
132	47	45	115
123	48	32	102

3 Genetic Algorithm Approach

Surveys on the existing GA's methodologies for multi-objective optimization problems can be found in [7], [8], [10], and references therein. Almost any application uses the methodologies described in these documents.

The application of GA's to MO scheduling problems has been rather scarce. Two interesting ideas are those presented in [18] and [2].

In [18] the scheduling of identical parallel machines, considering as objective functions the maximum flow time among machines and a non-linear function of the tardiness and earliness of jobs, is presented. In [2] a natural extension of NSGA [19] is presented and applied to flow-shop and job-shop scheduling problems. Another, totally different approach is that presented by Isibuchi and Murata [15]. They use a local search strategy after the genetic operations without considering non-dominance properties of solutions. Their method is applied to the MO flow-shop problem. Basseur et al. [3] also presents a local search idea to deal with this problem. Research on how intensive the local search should be, for a very specific instances of the permutation flowshop problem, is presented by Ishibuchi et al. [16].

In the literature, the main idea when solving MO scheduling problems is to apply the existing GA's methodologies to the problem to solve. However, there are no traces of studies on how adequate these methodologies may be. Again, the lack of a fair methodology for comparing the results does not help to improve this situation.

Brizuela et al. [6] present a question related to the existence of a superior combination of genetic operators in terms of non-dominance relations, and whether

or not this superiority will persist over some set of instances for a given flowshop problem. The work presented here try to answer this question.

The next section presents the algorithm we use to compare the performance of different operators.

4 The Algorithm

A standard GA for MO [19] is used with some modifications. The fitness share and dummy fitness assignment are standard. The main difference is in the replacement procedure, where an elitist replacement is used.

The specific NSGA we use here as a framework is stated as follows.

Algorithm 1. Multi-objective GA.

Step 1. Set $r = 0$. Generate an initial population $POP[r]$ of g individuals.
Step 2. Classify the individuals according to a non-dominance relation. Assign a dummy fitness to each individual.
Step 3. Modify the dummy fitness by fitness sharing.
Step 4. Set i=1.
Step 5. Use RWS to select two individuals for crossover according to their dummy fitness. Perform crossover with probability p_c.
Step 6. Perform mutation of individual i with probability p_m.
Step 7. Set $i = i + 1$. If $i = g$ then go to Step 8 otherwise go to Step 5.
Step 8. Set $r = r + 1$. Construct the new generation $POP[r]$ of g individuals. If $r = r_{max}$ then STOP; otherwise go to STEP 2.

The procedures involved at each step of this algorithm are explained in the following subsections.

4.1 Decoding

Each individual is represented by a string of integers representing job numbers to be scheduled (a permutation of job numbers). In this representation individual r looks like:

$$\mathbf{i}_r = (i_1^{(r)} i_2^{(r)} \cdots i_n^{(r)}), \quad r = 1, 2, \cdots, g \quad ,$$

where $i_k^{(r)} \in \mathbf{K}_n$.

The schedule construction method for this individual is as follows:

1) Enumerate all machines in $\mathbf{K}_m$ from 1 to m.
2) Select the first job ($i_1^{(r)}$) of $\mathbf{i}_r$ and route it from the first machine (machine 1) to the last (machine m).
3) Select iteratively the second, third, $\cdots$, n-th job and route them through the machines in the same machine sequence adopted for the first job $i_1^{(r)}$ (machines 1 to m). This must be done without violating the restrictions imposed in (1) to (3).

4.2 Selection and Replacement

The selection operator we use here is standard to GA's, like those proposed elsewhere [14]. Two selection processes are distinguished here.

Selection for mating - *SELECTION* (Step 5). This is the way we choose two individuals to undergo reproduction (crossover and mutation). In our algorithm roulette wheel selection (RWS) is used. This selection procedure works based on the dummy fitness function assigned to each individual. The way to compute the dummy fitness (Step 2) and the way to do the fitness sharing (Step 3) are standard (see [10], pages 147-148).

Selection after reproduction - *REPLACEMENT* (Step 8). This is the way to choose individuals to form the new generation from a set given by all parents and all offsprings. In this paper, the best elements are selected from the pool of parents and offsprings.

To define "the best," all individuals (parents and offsprings) are sorted in fronts. Then the first g individuals are copied from the first front (the nondominated front) following the next fronts until g individuals are obtained. After this the g individuals are ordered according to the following rules: in each front, individuals with better makespans have higher priority followed by individuals with higher tardiness, and finally by the mean flow time. After sorting all individuals repeated ones are replaced by randomly selected individuals from the pool of parents and offsprings. Notice that this does not avoid having repeated individuals in the new generation. The average running time of the above procedure is O(nlgn), the average running time of Quicksort. The selection of makespan in the ordering is irrelevant to the overall algortihm. This ordering is only to have an efficient algorithm for replacing repeated individuals.

4.3 Crossover

The crossover operators described here are the most frequently used in the literature.

OBX. This is the well known order-based crossover (see [13], page 239) proposed by Syswerda for the TSP. The position of some genes corresponding to parent 1 are preserved in the offspring and the not yet copied elements in parent 2 are copied in the order they appear in this parent. For this crossover a random mask is generated as shown in Fig. 1 (left side).

PPX. Precedence Preservative Crossover [5]. A subset of precedence relations of the parents genes are preserved in the offspring. Fig. 1 (center) shows how this operator works. The 1's in the mask indicate that genes from parent 1 are to be copied and the 0's indicate that genes from parent 2 are to be copied, in the order they appear from left to right.

OSX. One Segment crossover. This is similar to the well known two point crossover ([13], page 408) where also two random points are selected. An example of how this operator work is illustrated in Fig. 1 (right), here genes from loci 1 to s_1 of parent 1 are copied to loci 1 to s_1 in the offspring, and loci $s_1 + 1$ to s_2 (in the offspring) are copied from parent 2 starting at locust 1 and copying

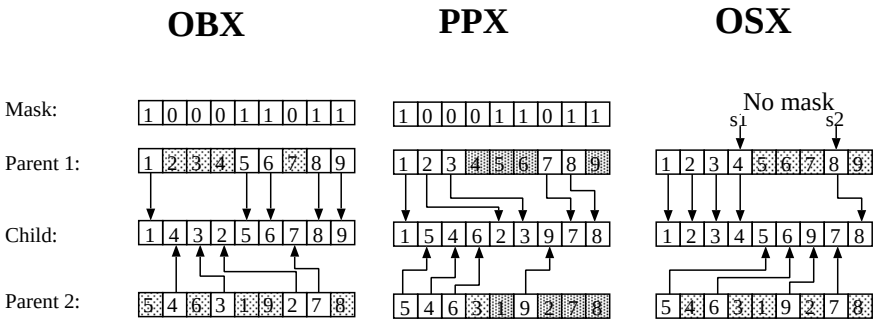

Fig. 1. Example of OBX, PPX, and OSX for nine jobs. In OSX the point s_1 indicates that all genes of P1 from loci 1 to s_1 are going to be copied in the offspring and from loci $s_1 + 1$ to s_2, in the offspring, genes come from P2. The rest will come from P1 copied from left to right considering only those that were not copied from P1 or P2

all those genes that have not been copied from parent 1. From loci $s_2 + 1$ to n, genes are copied from parent 1 considering only genes that were not copied into the offspring.

4.4 Mutation

The mutation operators that we use here are also standard (see [12]). The following mutation operators are used (Step 6).

INSERT. Two loci (s_1, s_2) are randomly selected if $s_1 < s_2$ then the gene corresponding to s_1 is placed on s_2 and all genes from $s_1 + 1$ to s_2 are shifted one position towards s_1. If $s_1 > s_2$ then the shift operation is performed towards s_2 as it is depicted in Fig. 2 (left side).

SWAP. Two loci are randomly selected and their genes interchanged. An example of this is shown in Fig. 2 (center).

SWITCH. A single interchange of two adjacent genes is performed. The locus to swap is randomly selected. Once this locus is selected the corresponding gene is interchanged with its immediate successor to the right, if the last gene (locus n) is selected then this is interchanged with the first one (locus 1). An example is shown in Fig. 2 (right side).

5 Performance Measures and Experimental Setup

Many performance measures for MO algorithms have been proposed (for a survey see [8]-[10]), these metrics try to identify three different characteristics [20] of the attained non-dominated front: *i*) how far the non-dominated front is from the true Pareto front, *ii*) how widely extended the set of solutions is, in the non-dominated front, and *iii*) how evenly distributed the solutions are, in the non-dominated front. Researchers have already noticed that, for many problems, the

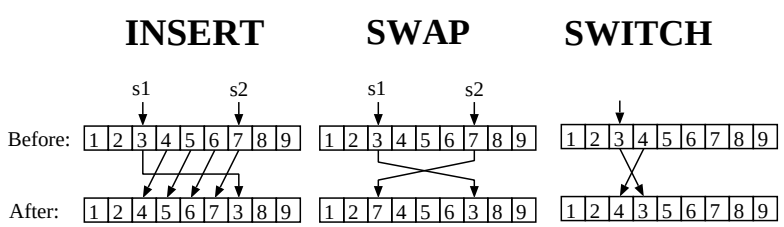

Fig. 2. INSERT, SWAP, and SWITCH mutation operators for a 9-jobs flowshop problem

proposed metrics are not "compatible" with the non-dominance relation among two different non-dominated fronts [17], this is especially true in combinatorial problems. For this kind of problems, properties such as expansion or evenly distributed may not be important. It will depend, of course, on the way the true Pareto front is extended and distributed. All the difficulties mentioned above motivate us to directly work with the nondominance relations between the fronts of the algorithms we want to compare.

In our approach, the non-dominated front to be used for comparisons, is obtained by applying the following procedure.

For each run i of algorithm j denote the set of solutions in the last generation as S_j^i. Calculate $Out(j)$ =NDS($\bigcup_{i=1}^{Numruns} S_j^i$), where NDS($X$) is an operator obtaining the set of non-dominated solutions from the input set X. Let $ns = |Out(j)|$, na = number of algorithms to compare, and let $ss(r, j)$ denote element r of $Out(j)$. Take each solution of $Out(j)$ and compare it against all other solutions in the sets $Out(p)$ for $p = 1, 2, \cdots, na$ and $p \neq j$. For each pair of solutions belonging to different sets we can obtain the following results: i) no dominance relation can be established between $ss(r, j)$ and $ss(k, p)$ ($ss(r, j) \succ\prec ss(k, p)$), ii) solution $ss(r, j)$ is dominated by solution $ss(k, p)$ ($ss(r, j) \prec ss(k, p)$), iii) solution $ss(r, j)$ dominates solution $ss(k, p)$ ($ss(r, j) \succ ss(k, p)$), and iv) the solutions are the same ($ss(r, j) = ss(k, p)$).

We propose as measures counting the number of times each of these situations happen, i.e. m_j^1 counts the number of times a solution in $Out(j)$ can not establish a dominance relation with solutions in the other sets, m_j^2 counts the number of solutions in other sets that dominate solutions in $Out(j)$, m_j^3 counts the number of times solutions in $Out(j)$ dominates solutions in $Out(p)$, finally m_j^4 counts the number of times solution $ss(r, j)$ has copies of itself in other sets.

5.1 Statistical Considerations

Since the previous experiments are taken over the whole set of solutions generated in different runs, we need to know if this behavior statistically holds when considering the output of randomly generated runs. In order to verify this, we compare the set of non-dominated solutions of each run and compute the statistics for each of the four measures (m_j^1,m_j^2,m_j^3, and m_j^4). For computing

these statistics we randomly select, without replacement, NDS(S_j^i) uniformly on $i \in \{1, \cdots, NumRuns\}$ for each $j \in \{1, \cdots, na\}$ until we have $NumRuns$ nondominated fronts to compare. The average for each measure m_j^i is computed over $NumRuns$ different samples.

The experiment is performed for each combination of crossover and mutation. Three different crossover operators and three different mutation operators are considered (see subsections 4.3 and 4.4).

5.2 Benchmark Generation

In order to generate a set of benchmark instances we take one of the problem instances in the library maintained by Beasly [4] and apply a procedure proposed by Armentano and Ronconi [1] for generating due-dates. The authors in [1] propose a way to systematically generate controlled due dates, following a certain structure. The procedure is based on an estimation of a lower bound for the makespan (f_1). This lower bound is defined as P, then due dates are randomly and uniformly generated for each job in the range of $P(1 - T - R/2)$ to $P(1 - T + R/2)$, where T and R are parameters for generating different scenarios of tardiness and due date ranges, respectively.

6 Results

The results on the experiments previously described are presented here. The main conclusion drawn from these results is that the crossover operator denominated OBX outperforms all other studied operators in terms of the dominance measures we propose in this work.

Table 3. Dominance relations, mean spacing (S), and ONVG for non-dominated fronts generated by nine algorithms. Instance 6 (75 jobs 20 machines)

Operator	(m_j^4)	(m_j^3)	(m_j^2)	(m_j^1)	(S)	(ONVG)
Algo 1	0.00	49.22	33.04	17.74	17.98	23.86
Algo 2	0.00	47.49	34.21	18.30	18.86	18.62
Algo 3	0.00	2.92	88.36	8.72	30.54	32.96
Algo 4	0.00	87.42	1.29	11.29	19.40	10.70
Algo 5	0.00	80.14	4.42	15.44	15.01	10.02
Algo 6	0.00	69.05	8.29	22.66	21.25	13.66
Algo 7	0.00	48.22	30.14	21.64	17.97	9.88
Algo 8	0.00	43.14	34.53	22.33	36.35	9.40
Algo 9	0.00	11.06	76.76	12.18	28.03	17.56

The maximum number of generations is 2000. The total number of runs for each algorithm is 50. The crossover rate was fixed to 1.0, the mutation rate to

0.1, and the population size to 100. Ten instances of 75 jobs and 20 machines are studied. The 75-jobs and 20-machines was taken from the benchmark library maintained by Beasly [4]. The due-dates were generated by setting T and R to 0.4 and 1.2, respectively. The operator combinations to study are denominated as follows: OSX-INSERT is Algo 1, OSX-SWAP (Algo 2), OSX-SWITCH (Algo 3), OBX-INSERT (Algo 4), OBX-SWAP (Algo 5), OBX-SWITCH (Algo 6), PPX-INSERT (Algo 7), PPX-SWAP (Algo 8), and PPX-SWITCH is Algo 9.

All generated instances have their processing times as in the benchmark instance in [4], the only difference is in the due-dates (the original problem in [4] considers no due dates).

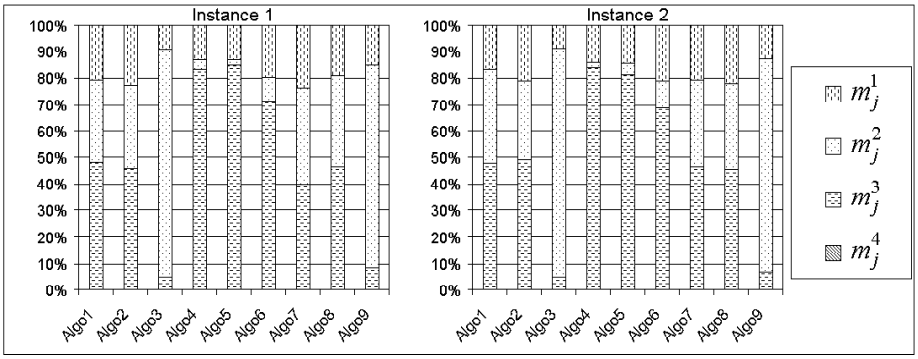

Fig. 3. Performance measures (m_j^i, $j \in \{1, \cdots, na\}, i \in \{1, \cdots, 4\}$) for the 75 jobs 20 machines flowshop problem. Instances 1 and 2

Table 3 presents the results for a single instance and 50 runs, it is clear to see that the combinations OBX with INSERT (Algo 4) and OBX with SWAP (Algo 5) have a small number of solutions dominated by the other combinations (less than 4.5%) and the highest percentage of domination, over 80% each. Table 3 also presents results for the Spacing (S) and the Overall Nondominated Vector Generation (ONVG) as they are defined in [20]. In this table we can see that the ONVG measure is not a good indicator for the performance of algorithms. ONVG indicates that Algo 3 is the best one, but considering dominance relations it is one of the worst ($m_3^2 = 88.36\%$). Figures 3 to 7 show the performance measure for each combination of operators, and for each of the 10 different generated instances.

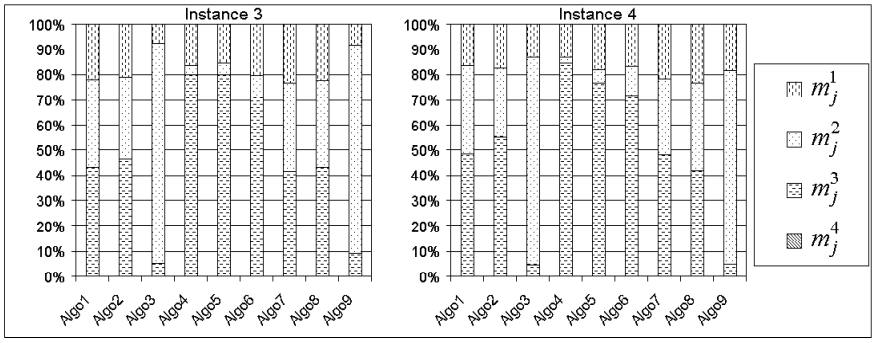

Fig. 4. Performance measures (m_j^i, $j \in \{1, \cdots, na\}, i \in \{1, \cdots, 4\}$) for the 75 jobs 20 machines flowshop problem. Instances 3 and 4

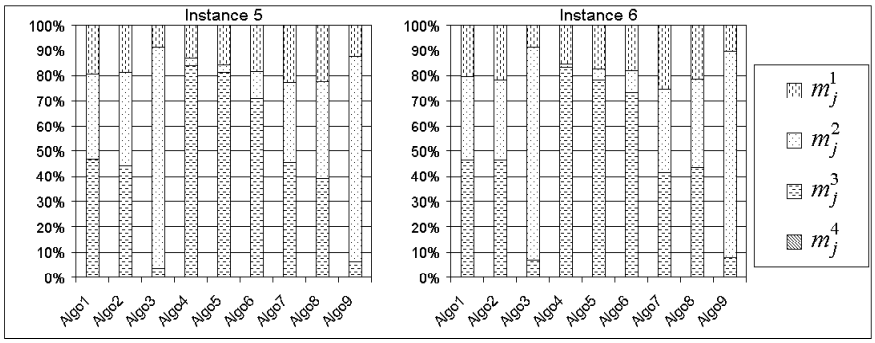

Fig. 5. Performance measures (m_j^i, $j \in \{1, \cdots, na\}, i \in \{1, \cdots, 4\}$) for the 75 jobs 20 machines flowshop problem. Instances 5 and 6

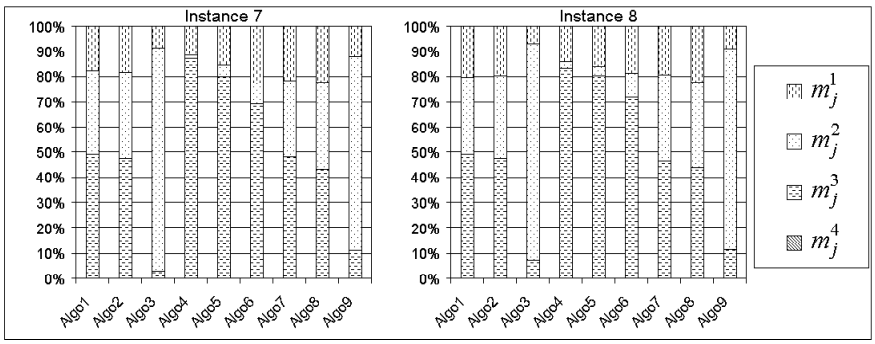

Fig. 6. Performance measures (m_j^i, $j \in \{1, \cdots, na\}, i \in \{1, \cdots, 4\}$) for the 75 jobs 20 machines flowshop problem. Instances 7 and 8

Figures 8 to 9 show the projections in two axes of the overall nondominated front for Algo 1, Algo 4, and Algo 5 (instance 6). It is easy to see that Algo 4 and 5 clearly outperforms Algo 1, in every projection. Figure 9 shows that the mean flow time (f_2) and the tardiness (f_3) are not in conflict. This is because of the range of generated due dates.

Figures 10 to 11 show the same two axes projection of three randomly selected non dominated fronts (one for each algorithm) of a single run, and for a particular problem instance (Instance 6). Again, Algo 4 and 5 outperform Algo 1.

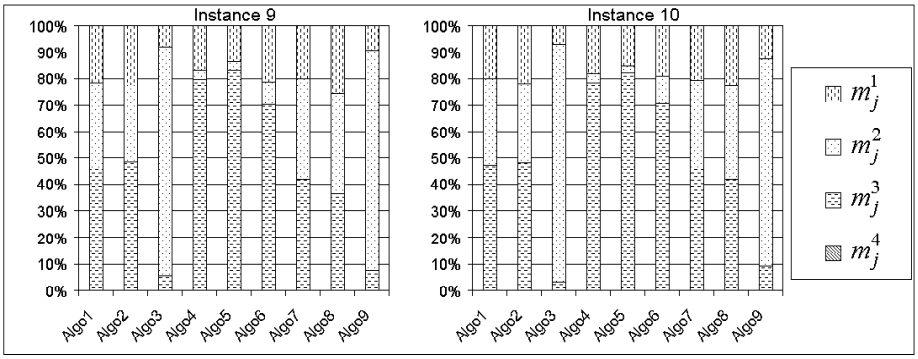

Fig. 7. Performance measures (m_j^i, $j \in \{1, \cdots, na\}, i \in \{1, \cdots, 4\}$) for the 75 jobs 20 machines flowshop problem. Instances 9 and 10

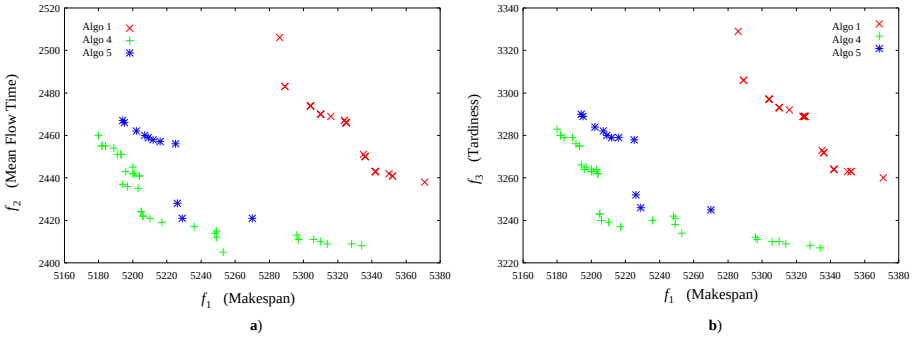

Fig. 8. Union of all nondominated solutions in the last generation ($Out(j)$) for Algo 1, 4, and 5 for Instance 6. a) Mean Flow Time and Makespan relations. b) Tardiness and Makespan relations

By looking at Figs. 8 to 11 we can see that Algo 4 outperforms Algo 1 and Algo 5. To study the significance of the difference between these algorithms,

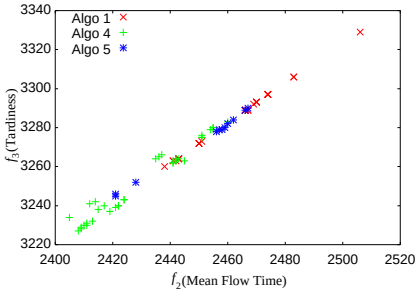

Fig. 9. Tardiness and Mean Flow Time relations. Union of all nondominated solutions in the last generation ($Out(j)$) for Algo 1, 4, and 5 for Instance 6

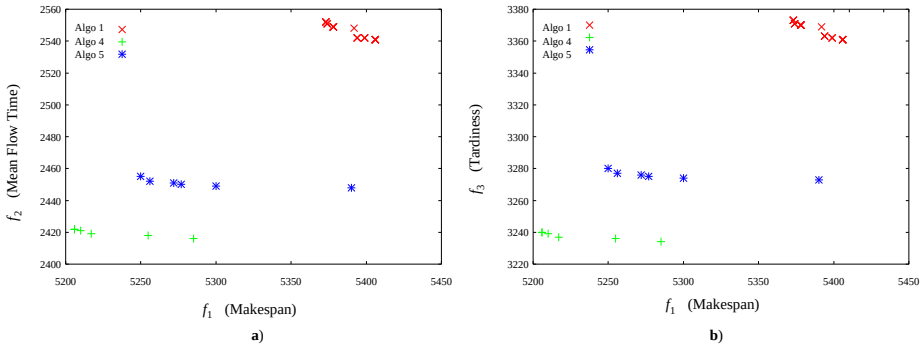

Fig. 10. Nondominated solutions in the last generation of a randomly selected run ($\text{NDS}(S_j^i)$) for Algo 1, 4, and 5 for Instance 6. a) Mean Flow Time and Makespan relations. b) Tardiness and Makespan relations

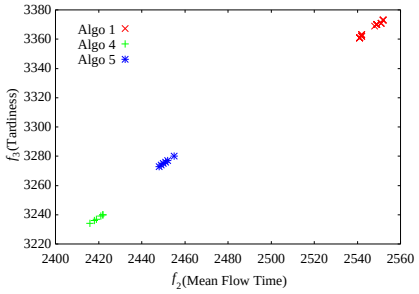

Fig. 11. Tardiness and Mean Flow Time relations. Nondominated solutions in the last generation of a randomly selected run ($\text{NDS}(S_j^i)$) for Algo 1, 4, and 5 for Instance 6

dominance relations between Algo 4 - Algo 1, and Algo 4 - Algo 5 are computed for all instances. The results for Algo 4 and Algo 5 are presented in Table 4, these algorithms uses the same crossover operator (OBX) and different muta-

Table 4. Comparison of dominance relations between Algo 4 - Algo 5, and Algo 4 - Algo 1 over all instances. Dominance relations, mean and standard deviation.

Algo 4 vs. Algo 5	(m_j^4)	(m_j^3)	(m_j^2)	(m_j^1)
Mean	0.00	27.26	20.12	52.62
Std. Dev.	1.00	6.87	6.55	4.63
Algo 4 vs. Algo 1				
Mean	0.00	84.61	1.42	13.97
Std. Dev.	0.00	4.84	1.80	3.30

tion operators, INSERT and SWAP, respectively. In this case we see that the domination rate produced by Algo 4 (27%) is not far away from the domination rate of Algo 5 (20%) and the nondominance is more than 50%. The lower part of Table 4 shows a different result, two different crossover operators are compared (OBX and OSX), Algo 4 - Algo 1. In this case we see that solutions in the nondominated front generated by Algo 4 dominates most of the times (84.61%) solutions in the front generated by Algo 1. This result shows the superiority of the OBX operator.

7 Conclusions

An experimental analysis of genetic operators (crossover and mutation) for a set of instances of the multi-objective permutation flowshop problem is presented. Outperformance measures, related to dominance relations of the nondominated set generated at each run, are proposed.

The main conclusion of the work is that the OBX crossover operator outperforms all others in terms of dominance relations. This outperformance is verified on a specific set of instances of the PFSP. The reason why this operator outperforms the others remains to be solved.

As future work we need to verify whether the properties for the best operator combinations still hold under different settings of the parameters T and R, different problems size, and different algorithms. The most important point will be to understand why some operators outperform others. This will help us to better understand this class of problems.

References

1. Armentano, V. A., and Ronconi, D. P.: Tabu search for total tardiness minimization in flowshop scheduling problems. Computers & Operations Research, Vol. 26 (1999) 219-235
2. Bagchi, T. P.: Multiobjective Scheduling by Genetic Algorithm. Kluwer Academic Publishers (1999)

3. Basseur, M., Seynhaeve, F., Talbi E.: Design of multi-objective evolutionary algorithms: Application to the flow-shop scheduling problem. Congress on Evolutionary Computation (2002) 459-465
4. http://mscmga.ms.ic.ac.uk/info.html
5. Bierwirth, C., Mattfeld, D. C., Kopfer, H.: On Permutation Representations for Scheduling Problems. In Proceedings of Parallel Problem Solving from Nature. Lecture Notes in Computer Science, Vol. 1141. Springer-Verlag, Berlin Heidelberg New York (1996) 310-318
6. C. Brizuela, Y. Zhao, N. Sannomiya.: Multi-Objective Flowshop: Preliminary Results. In Zitzler, E., Deb, K., Thiele, L., Coello Coello, C. A., Corne D., eds., Evolutionary Multi-Criterion Optimization, First International Conference, EMO 2001, vol. 1993 of LNCS, Berlin:Springer-Verlag (2001) 443-457
7. Coello Coello, C. A.: A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. Knowledge and Information Systems, Vol. 1, No. 3 (1999) 269-308
8. Coello Coello, C. A., Van Veldhuizen, D. A., and Lamont, G. B.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers (2002)
9. Deb, K.: Multi-objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. Evolutionary Computation, 7(3) (1999) 205-230
10. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons (2001)
11. M. R. Garey, D. S. Johnson and Ravi Sethi.: The Complexity of Flowshop and Jobshop Scheduling. Mathematics of Operations Research, Vol. 1, No. 2 (1976) 117-129
12. Gen, M. and Cheng, R.: Genetic Algorithms & Engineering Design. John Wiley & Sons (1997)
13. Gen, M. and Cheng, R.: Genetic Algorithms & Engineering Optimization. John Wiley & Sons (1997)
14. Golberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley (1989)
15. Isibuchi, H. and Murata, T.: Multi-objective Genetic Local Search Algorithm. Proceedings of the 1996 International Conference on Evolutionary Computation (1996) 119-124
16. Isibuchi, H. and Murata, T.: Multi-objective Genetic Local Search Algorithm and its application to flowshop scheduling. IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews, 28(3) (1998) 392-403
17. Knowles J. and Corne D.: On Metrics for Comparing Nondominated Sets. Proceedings of the 2002 Congress on Evolutionary Computation (2002) 711-716
18. Tamaki, H., and Nishino, E.: A Genetic Algorithm approach to multi-objective scheduling problems with regular and non-regular objective functions. Proceedings of IFAC LSS'98 (1998) 289-294
19. Srinivas, N. and Deb, K.: Multi-Objective function optimization using non-dominated sorting genetic algorithms. Evolutionary Computation, 2(3) (1995) 221-248
20. Van Veldhuizen, D. and Lamont, G.: On measuring multiobjective evolutionary algorithm performance. Proceedings of the 2000 Congress on Evolutionary Computation (2000) 204-211
21. Zitzler E., Laumanns M., Thiele L., Fonseca C. M., Grunert da Fonseca V.: Why Quality Assesment of Multiobjective Optimizers Is Difficult. Proceedings of the 2002 Genetic and Evolutionary Computation Conference (GECCO2002) (2002) 666-673