A Comparative Study of Crossover Operators for Genetic Algorithms to Solve the Job Shop Scheduling Problem

JORGE MAGALHÃES-MENDES
Department of Civil Engineering
CIDEM
School of Engineering – Polytechnic of Porto
Rua Dr. António Bernardino de Almeida, 431 – 4200-072 Porto
PORTUGAL
jjm@isep.ipp.pt

Abstract: - Genetic algorithms (GA) are wide class of global optimization methods. Many genetic algorithms have been applied to solve combinatorial optimization problems. One of the problems in using genetic algorithms is the choice of crossover operator. The aim of this paper is to show the influence of genetic crossover operators on the performance of a genetic algorithm. The GA is applied to the job shop scheduling problem (JSSP). To achieve this aim an experimental study of a set of crossover operators is presented. The experimental study is based on a decision support system (DSS). To compare the abilities of different crossover operators, the DSS was designed giving all the operators the same opportunities. The genetic crossover operators are tested on a set of standard instances taken from the literature. The makespan is the measure used to evaluate the genetic crossover operators. The main conclusion is that there is a crossover operator having the best average performance on a specific set of solved instances.

Key-Words: - Scheduling, Genetic Algorithms, Crossover Operators, Optimization, Operations Research, JSSP.

1 Introduction and Background

Scheduling is one of the most critical issues in the planning and managing of manufacturing process. The difficulty of finding the optimal schedule depends on the shop environment, the process constrainsts and the performance indicator. One of the most difficult problems in this area is the job shop scheduling problem (JSSP), Qing-dao-er-ji and Wang [38].

The JSSP may be described as follows: given *n* jobs, each job must be processed on *m* machines. Each operation uses one of the *m* machines for a fixed duration. Each machine can process at most one operation at a time and once an operation initiates processing on a given machine it must complete processing on that machine without interruption. A schedule is a complete set of operations (*n* x *m*), to be processed on different machines, in a given order. The problem is to find a schedule of minimal time to complete all jobs.

The JSSP is considered as a particularly hard combinatorial optimization problem, Lawler et al. [10].

Exact methods (Giffler and Thompson [21], Carlier and Pinson [22, 23], Brucker et al. [24], Williamson et al. [25]) have been successful in solving small instances. Problems of dimension 15×15 are still considered to be beyond the reach of today's exact methods.

Many approximate methods have been developed in the last two decades to solve the JSSP, such as:

1. *Simulated annealing* (SA) - Lourenço [12];
2. *Tabu search* (TS) (Nowicki and Smutnicki [9, 31], Pezzela and Merelli [7], Zhang et al. [11, 35]),
3. Evolutionary algorithm (EA) techniques like *genetic algorithms* (GA) - Aarts et al. [15], Croce et al. [17], Dorndorf et al. [18], Wang and Zheng [19], Essafi et al. [3], Gonçalves et al. [4], Hasan et al. [27], Mendes [30], Qing-dao-er-ji and Wang [38], Choi and Park [39], Chiu et al. [40, 41]);
4. EA related techniques like *particle swarm optimization* (PSO) - Sha and Hsu [6];
5. *Greedy randomized adaptive search procedure* (GRASP) - Aiex et al. [16] and Binato et al. [20];
6. Others metaheuristics - Rego and Duarte [8] proposed a filter-and-fan approach based on the *shifting bottleneck procedure* (SBP) and Pardalos et al. [32].

Evolutionary algorithms often perform well approximating solutions to all types of problems. The most popular type of EA is the GA.

The GA seeks the solution of a problem in the form of strings of numbers (traditionally binary, although the best representations are usually those that reflect something about the problem being solved), by applying operators such as crossover and mutation.

The performance of a GA is dependent on the genetic operators in general and on the type of crossover operator, in particular. During the evolution process by a GA, if the selected chromosomes are identical, some of the crossover operators have failed to create offspring that are different from their parents. Effective crossover in a GA is achieved through establishing the optimum relationship between the crossover and the search problem itself [28].

Some literature focuses on the genetic operators, e.g., Sywerda [33] and De Jong and Spears [34] compared the various crossover operators, particularly the numbers of crossover points. A relevant result is that sometimes the final output would be better if the number of crossover points was increased.

Besides empirical analysis, substantial efforts have been invested in comparing, from theoretical perspectives, between mutation and crossover as well as between the various crossover operators [34]. However, these theories are not general enough to allow for predicting when to apply or what type of crossover operator to employ [28].

Given the current state of knowledge is justified, therefore, additional research work that allows to obtain new results.

This paper describes an experimental study of a set of genetic crossover operators. These crossover operators are applied on a hybrid genetic algorithm. This hybrid genetic algorithm works with a local search using the Monte Carlo method [30]. The aim of this study was to validate empirically the most appropriate crossover operator for solving the job shop scheduling problem.

The remainder of the paper is organized as follows. In section 2 is defined the problem. In section 3 is referred the decision support system. In section 4 we refer the advantages of using genetic algorithms in combinatorial nature problems.

In section 5 is described the genetic algorithm. In section 6 are presented and discussed the results of the experimental study. Finally, conclusion and remarks for further works are given in Section 7.

2 Problem Definition

There are a set of jobs $J = \{1,\ldots, n\}$, a set of machines $M = \{1,..., m\}$, and a set of operations $O = \{o_0, o_1, \ldots, o_{ji},\ldots o_{nm}, o_{nm+1}\}$. Set O contains all the operations of each job. Each job has m operations. Each machine can process at most one operation at time.

The JSSP is to find a schedule which minimizes the makespan (C_{max}), that is, the finish time of the last operation completed in the schedule, taking into account the precedence constraints.

Let $O = \{o_0, o_1, \ldots, o_{ji},\ldots o_{nm}, o_{nm+1}\}$denote the set of all operations (n x m) to be scheduled and $M = \{1,..., m\}$ the set of machines. The operations o_0 and o_{nm+1} are dummy, have no duration and represent the initial and final operations. The operations are interrelated by two kinds of constraints:

- First, the precedence constraints, which force each operation o_{ji} to be scheduled after all predecessor operations are completed $\boldsymbol{P}_{ji}$;
- Second, operation o_{ji} can only be scheduled if the machine it requires is idle. Further, let d_{ji} denote the (fixed) duration (processing time) of operation o_{ji}.

Let $\boldsymbol{F}_{ji}$ represent the finish time of operation o_{ji}. A schedule can be represented by a vector of finish times ($F_{11,}, F_{ji}, \ldots, F_{nm+1}$).

Let $\boldsymbol{A}(t)$ be the set of operations being processed at time t, and let $\boldsymbol{r}_{ji} = 1$ if operation o_{ji} requires machine m to be processed and $\boldsymbol{r}_{ji} = 0$ otherwise.

The conceptual model of the JSSP can be described the following way:

$$\textbf{Min } \boldsymbol{F}_{nm+1} \tag{1}$$

subject to:

$$\boldsymbol{F}_{kl} \le \boldsymbol{F}_{ji} - \boldsymbol{d}_{ji} \qquad \boldsymbol{j} = 1,\ldots,\boldsymbol{n} \; ; \; \boldsymbol{i} = 1,\ldots,\boldsymbol{m} \; ; \; \boldsymbol{kl} \in \boldsymbol{P}_{ji} \tag{2}$$

$$\sum_{\boldsymbol{ji} \in \boldsymbol{A}(\boldsymbol{t})} \boldsymbol{r}_{ji} \le 1 \qquad \boldsymbol{i} \in \boldsymbol{M} \; ; \; \boldsymbol{t} \ge 0 \tag{3}$$

$$\boldsymbol{F}_{ji} \ge 0 \qquad \boldsymbol{j} = 1,\ldots,\boldsymbol{n} \; ; \quad \boldsymbol{i} = 1,\ldots,\boldsymbol{m} \tag{4}$$

The objective function (1) minimizes the finish time of operation o_{nm+1} (the last operation), and therefore minimizes the *makespan*. Constraints (2) impose the precedence relations between operations and constraints (3) state that one machine can only process one operation at a time. Finally (4) forces the finish times to be non-negative.

3 A decision support system (DSS)

A DSS combines a genetic algorithm, a schedule generator scheme (SGS) and a local search procedure, consisting in the following steps, see Figure 1:

- **Step1**: Combines a genetic algorithm with a schedule generation scheme

(SGS). This SGS generates parameterized active schedules. This step allows to obtain a schedule for each chromosome;

- **Step2**: This step makes use of a local search procedure that attempts to improve the solution obtained previously.

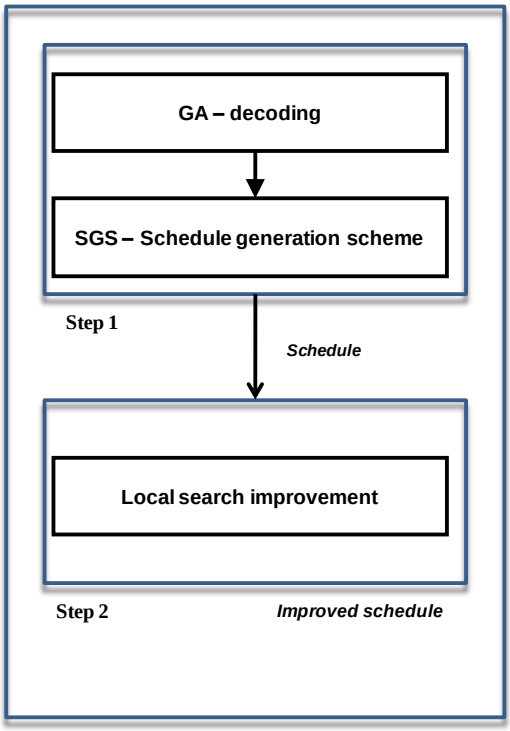


Fig.1 – Architecture of the approach.

The DSS allows the user choose the crossover operator:

- GA-MC-SPc: Single-Point crossover or simple crossover;
- GA-MC-TPc: Two-Point crossover;
- GA-MC-Uc: Uniform crossover or discrete crossover;
- GA-MC-Fc: Flat crossover.

The DSS was developed using the Visual Basic for Applications (VBA) from Microsoft and the Gantt Time Package V3.21 [30].

4 Real-coded evolutionary algorithms

The evolutionary algorithms are an interdisciplinary research area comprising several paradigms inspired by Darwinian principle of evolution.

The current stage of research considers, among others, the following paradigms: genetic algorithms, genetic programming, evolutionary strategies, neuroevolution and differential evolution.

The genetic algorithms have been applied successfully in several areas, such as bioinformatics, computational science, engineering, economics, chemistry, manufacturing, mathematics and physics.

A real-coded GA is adopted in this article. Compared with the binary-code GA, the real-coded GA has several distinct advantages, which can be summarized as follows, Y.-Z. Luo et al. [36]:

- It is more convenient for the real-coded GA to denote large scale numbers and search in large scope, and thus the computation complexity is amended and the computation efficiency is improved;
- The solution precision of the real-coded GA is much higher than that of the binary-coded GA;
- As the design variables are coded by floating numbers in classical optimization algorithms, the real-coded GA is more convenient for combination with classical optimization algorithms.

5 Genetic Algorithms

Genetic algorithms (GA) are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search [1].

The general schema of GA may be illustrated as follows (Fig. 2).

```
procedure GENETIC-ALGORITHM

Generate initial population P0;
 Evaluate population P0;
 Initialize generation counter g ←0;

 While stopping criteria not satisfied repeat
    Select some elements from Pg to copy into Pg+1;
    Crossover some elements of Pg and put into Pg+1;
    Mutate some elements of Pg and put into Pg+1;
    Evaluate some elements of Pg and put into Pg+1;
    Increment generation counter: g ← g+1;
  End while

End GENETIC-ALGORITHM;
```

Fig.2 - Pseudo-code of a genetic algorithm.

First of all, an initial population of potential solutions (individuals) is generated randomly. A selection procedure based on a fitness function enables to choose the individuals candidate for reproduction. The reproduction consists in recombining two individuals by the crossover

operator, possibly followed by a mutation of the offspring. Therefore, from the initial population a new generation is obtained. From this new generation, a second new generation is produced by the same process and so on. The stop criterion is normally based on the number of generations.

5.1 Decoding

The genetic algorithm uses a random key alphabet which is comprised of real random numbers between 0 and 1.

A chromosome represents a solution to the problem and is encoded as a vector of random keys (random numbers).

Each solution chromosome is made of 2*n* genes where *n* is the number of operations:

$$Chromosome = (gene_1, .., gene_n, \ gene_{n+1}, \ldots, gene_{2n})$$

The first half $(gene_1, .., gene_n)$ corresponds to the potential solution to the optimization problem, that is, one gene for each activity. The second half $(gene_{n+1}, .., gene2_n)$ defines the vector where each gene has an associated delay time.

The priority decoding expression used the following expression:

$$\boldsymbol{PRIORITY_j} = \boldsymbol{gene_j}$$
$$\boldsymbol{Delay_j} = \boldsymbol{gene_{n+j}} \times 1.5 \times \boldsymbol{mdur}, \qquad \boldsymbol{j} = 1, \ldots, \boldsymbol{n}.$$

where ***mdur*** is the maximum duration of all operations. The factor 1.5 was chosen as a result of a careful fine-tuning experimental phase.

5.2 Initial population

The initial populations are generated randomly. The quality of this population is poor and one way to improve it is to incorporate some chromosomes generated by priority rules.

In this paper are selected the priority rules GRPW (greatest rank positional weight) and SPT (shortest processing time) to improve some chromosomes of the initial population.

5.3 Crossover operators

The genetic algorithms typically use the following types of operators:

- **Selection**: Operator for selecting individuals for reproduction according to their fitness;
- **Crossover** (Sexual Recombination): Operator of merging the genetic information of two individuals. In many respects the effectiveness of crossover is depended on coding;
- **Mutation** (Asexual): In real evolution, the genetic material can by changed randomly by erroneous reproduction or other deformations of genes, e.g. by gamma radiation. In genetic algorithms, mutation realized as a random deformation of alleles with a certain probability, Shaparov [37].

Usually, there are two means of modifying genetic material: a recombination operation that could be understood as some kind of crossover and mutation.

In this study we compared the following crossover operators:

- Single-point Crossover or simple crossover;
- Two-point Crossover;
- Uniform Crossover or discrete crossover;
- Flat Crossover.

The reasons by which these crossover operators were chosen are:

- Single-point crossover: is a simple and often-used method for genetic algorithms;
- N-point crossover. Instead of only one, N breaking points are chosen randomly. Every second section is swapped. Among this class, two-point crossover is particularly important;
- Usually flat or discrete crossovers are applied in real-coded genetic algorithm (Shaparov [37]).

5.3.1 Single-point crossover

After reproduction, crossover may proceed in two steps. First, members of the newly reproduced chromosomes in the mating pool are mated at random. Second, each pair of chromosomes undergoes crossover as follows: an integer position *k* along the chromosome is selected uniformly at random between 1 and the chromosome length *l*. Two new chromosomes are created swapping all the genes between *k*+1 and *l* [1], see Fig. 3.

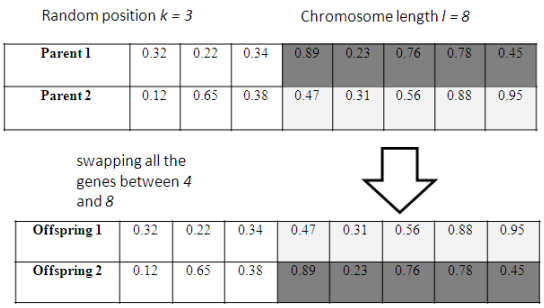
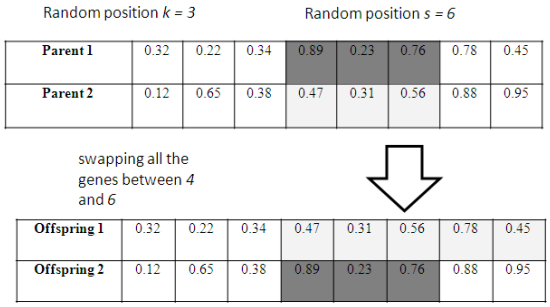


Fig.3 – Single-point crossover operator example.

5.3.2 Two-point crossover

Two-point crossover is very similar to single-point crossover except that two cut-points are randomly generated instead of one, see Fig. 4.



Parent 1	0.32	0.22	0.34	0.89	0.23	0.76	0.78	0.45
Parent 2	0.12	0.65	0.38	0.47	0.31	0.56	0.88	0.95
Offspring 1	0.32	0.22	0.34	0.47	0.31	0.56	0.78	0.45
Offspring 2	0.12	0.65	0.38	0.89	0.23	0.76	0.88	0.95

Fig.4 – Two-point crossover operator example.

5.3.3 Uniform crossover

In uniform crossover, a value of the first parent's gene is assigned to the first offspring and the value of the second parent's gene is to the second offspring with a probability value p_c.

With probability p_c the value of the first parent's gene is assigned to the second offspring and the value of the second parent's gene is assigned to the first offspring, see Fig. 5.

Parent 1	0.32	0.22	0.34	0.89	0.23	0.76	0.78	0.45
Parent 2	0.12	0.65	0.38	0.47	0.31	0.56	0.88	0.95
Random Number	0.32	0.22	0.34	0.89	0.23	0.76	0.78	0.45
Prob. Cross = 0.7	< 0.7	< 0.7	< 0.7	> 0.7	< 0.7	> 0.7	> 0.7	< 0.7
Offspring 1	0.32	0.22	0.34	0.89	0.23	0.76	0.78	0.45
Offspring 2	0.12	0.65	0.38	0.47	0.31	0.56	0.88	0.95

Fig.5 – Uniform crossover operator example.

This work used a probability p_c of 0.7. An example of this operator has been used successfully in Gonçalves et al. [4].

5.3.4 Flat crossover

Consider the following parents:

*Parent*1 = $(x_{1,1}, \ldots, x_{1,n})$
*Parent*2 = $(x_{2,1}, \ldots, x_{2,n})$

and a vector of random values r = $(r_1, \ldots, r_n)$.

The *offspring* 1 = $(x^1_1, \ldots, x^1_n)$ is computed as a vector of linear combinations in the following way (for all $i = 1, \ldots, n$):

$$x_i^1 = r_i\, x_{1,i} + (1 - r_i)\, x_{2,i} \quad , \qquad i = 1, \ldots, n \tag{5}$$

Second offspring is computed analogously, see Fig.6.

Parent 1	0.32	0.22	0.34	0.89	0.23	0.76	0.78	0.45
Parent 2	0.12	0.65	0.38	0.47	0.31	0.56	0.88	0.95
Random Number 1	0.32	0.22	0.34	0.89	0.23	0.76	0.78	0.45
Offspring 1	0.18	0.56	0.37	0.84	0.29	0.71	0.80	0.73
Parent 1	0.32	0.22	0.34	0.89	0.23	0.76	0.78	0.45
Parent 2	0.12	0.65	0.38	0.47	0.31	0.56	0.88	0.95
Random Number 2	0.16	0.34	0.92	0.54	0.65	0.76	0.98	0.32
Offspring 2	0.15	0.50	0.34	0.70	0.26	0.71	0.78	0.79

Fig.6 – Flat crossover operator example.

5.4 Random mutation

The mutation operator preserves diversification in the search. The mutation operator chosen was the random mutation. This operator is applied to each offspring in the population with a predetermined probability. For a randomly chosen gene *i* of an individual $(gene_1, .., gene_n, gene_{n+1}, \ldots, gene_{2n})$, the allele $gene_i$ is replaced by a randomly chosen value from a interval]0, 1[.

We assume that the probability of the mutation in this work is 0.1%. With 1000 genes positions we should expect 1000 x 0.001 = 1 genes to undergo mutation for this probability value.

5.5 Configuration of the experiments

All results reported for GA-MC were obtained with the same parameters. Though there is no straightforward way to configure the parameters of a genetic algorithm, we obtained good results with values:

- **Population size:** 2 × number of operations for each problem;
- **Initial population**: 1% chromosomes calculated by priority rules GRPW and SPT;
- **Top** (best): 1% from the previous population chromosomes are copied to the next generation;
- **Mutation rate:** 0.1%;
- **Termination criterion:** Maximum number of generations.

5.6 Evolutionary strategy

To breed good solutions, the random key vector population is operated upon by a genetic algorithm.

There are many variations of genetic algorithms obtained by altering the reproduction, crossover, and mutation operators.

Reproduction is a process in which individual (chromosome) is copied according to their fitness values (*makespan*).

Reproduction is accomplished by first copying some of the best individuals from one generation to the next, in what is called an elitist strategy.

In this paper the fitness proportionate selection, also known as roulette-wheel selection, is the genetic operator for selecting potentially useful solutions for reproduction. The characteristic of the roulette wheel selection is stochastic sampling.

The fitness value is used to associate a probability of selection with each individual chromosome. If f_i is the fitness of individual i in the population, its probability of being selected is,

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \quad , \qquad i = 1,\ldots,n \tag{6}$$

A roulette wheel model is established to represent the survival probabilities for all the individuals in the population.

Fig. 7 shows this evolutionary strategy with the operators selection, recombination (or crossover) and mutation.

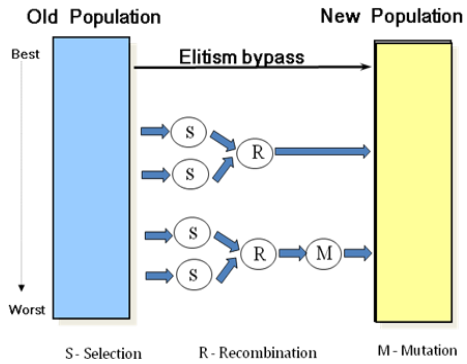


Fig.7 – Evolutionary strategy.

This evolutionary strategy was applied to the resource constrained project scheduling problem with a good performance, see Mendes [2, 26].

6 Numerical experiments

To evaluate the performance of each crossover operator, we considered the following classes of problems, taken from the literature:

- ABZ5, ABZ6, ABZ7, ABZ8 and ABZ9 proposed by Adams et al. [10];
- FT6, FT 10 and FT20 originally proposed by Fisher and Thompson [13];
- LA1-LA40 proposed by Lawrence [14];
- ORB1-ORB10, proposed by Applegate and Cook [29].

Tables 1, 2, 3 and 4 summarize the experimental results. It lists number of jobs, number of operations, instance, best known solution (BKS), GA-MC-SPc, GA-MC-TPc, GA-MC-Uc and GA-MC-Fc. The last row of each table shows the value of the Average Relative Deviation (ARD).

The ARD is calculated in the following way:

$$RE = \sum_{i=1}^{NIS} \frac{C_{\max_i} - BKS_i}{BKS_i} \tag{7}$$

$$ARD = \frac{RE}{NIS} \tag{8}$$

where NIS is number of instances solved.

Problem	*BKS*	*GA-MC-SPc*	*GA-MC-TPc*	*GA-MC-Uc*	*GA-MC-Fc*
ABZ5 (10 x 10)	1234	1234	1234	1234	1234
ABZ6 (10 x 10)	943	943	943	943	943
ABZ7 (15 x 20)	656	688	695	690	693
ABZ8 (15 x 20)	645	704	708	702	701
ABZ9 (15 x 20)	661	715	720	724	721
% ARD		4,44%	4,93%	4,71%	4,68%

Table 1: Experimental results for instances ABZ5-ABZ9.

Problem	*BKS*	*GA-MC-SPc*	*GA-MC-TPc*	*GA-MC-Uc*	*GA-MC-Fc*
FT06 (6 x 6)	55	55	55	55	55
FT10 (10 x 10)	930	930	930	930	930
FT20 (20 x 5)	1165	1165	1165	1165	1165
% ARD		0,00%	0,00%	0,00%	0,00%

Table 2: Experimental results for instances FT06, FT10 and FT20.

In most of the problems, the single crossover operator performs better than the others and is the operator with lowest ARD, see Tables 1, 2, 3, 4 and 5.

Unlike the algorithm GA-MC-SPc that always has the best ARD, the algorithm GA-MC-SPc has not always the second best (lowest) ARD. Indeed, the algorithm GA-MC-Fc for ABZ problems (Table 1) obtains the second lowest ARD.

This result can lead to considerer that the operator Fc is second best for larger problems, i.e.,

problems greater than 10x10. For problems larger than 10x10, we obtain the following values of ARD: 1.51% (GA-MC-SPc), 1.89% (GA-MC-TPc), 1.84% (GA-MC-Uc) and 2.13% (GA-MC-Fc), see Table 5.

Globally the second best value for the ARD is obtained by algorithm GA-MC-Uc and the third best value by the algorithm GA-MC-TPc, see Table 6.

With these results, for these types of problems, the single-crossover performed the best results, followed by the uniform crossover operator.

Problem	*BKS*	*GA-MC-SPc*	*GA-MC-TPc*	*GA-MC-Uc*	*GA-MC-Fc*
LA01 (10 x 5)	666	666	666	666	666
LA02 (10 x 5)	655	655	655	655	655
LA03 (10 x 5)	597	597	597	597	597
LA04 (10 x 5)	590	590	590	590	590
LA05 (10 x 5)	593	593	593	593	593
LA06 (15 x 5)	926	926	926	926	926
LA07 (15 x 5)	890	890	890	890	890
LA08 (15 x 5)	863	863	863	863	863
LA09 (15 x 5)	951	951	951	951	951
LA10 (15 x 5)	958	958	958	958	958
LA11 (20 x 5)	1222	1222	1222	1222	1222
LA12 (20 x 5)	1039	1039	1039	1039	1039
LA13 (20 x 5)	1150	1150	1150	1150	1150
LA14 (20 x 5)	1292	1292	1292	1292	1292
LA15 (20 x 5)	1207	1207	1207	1207	1207
LA16 (10 x 10)	945	945	946	946	946
LA17 (10 x 10)	784	784	784	784	784
LA18 (10 x 10)	848	848	848	848	848
LA19 (10 x 10)	842	842	842	842	842
LA20 (10 x 10)	902	902	902	902	902
LA21 (15 x 10)	1046	1054	1052	1058	1059
LA22 (15 x 10)	927	932	935	932	935
LA23 (15 x 10)	1032	1032	1032	1032	1032
LA24 (15 x 10)	935	944	949	944	955
LA25 (15 x 10)	977	985	984	989	993
LA26 (20 x 10)	1218	1218	1218	1218	1218
LA27 (20 x 10)	1235	1252	1252	1266	1273
LA28 (20 x 10)	1216	1231	1234	1228	1248
LA29 (20 x 10)	1157	1180	1213	1193	1209
LA30 (20 x 10)	1355	1355	1355	1355	1355
LA31 (30 x 10)	1784	1784	1784	1784	1784
LA32 (30 x 10)	1850	1850	1850	1850	1850
LA33 (30 x 10)	1719	1719	1719	1719	1719
LA34 (30 x 10)	1721	1721	1721	1721	1721
LA35 (30 x 10)	1888	1888	1888	1888	1888
LA36 (15 x 15)	1268	1278	1282	1284	1284
LA37 (15 x 15)	1397	1408	1411	1422	1407
LA38 (15 x 15)	1196	1213	1219	1232	1247
LA39 (15 x 15)	1233	1244	1250	1250	1249
LA40 (15 x 15)	1222	1233	1246	1233	1245
% ARD		0,31%	0,47%	0,47%	0,64%

Table 3: Experimental results for instances LA01-LA40.

Tables 1, 2, 3 and 4 shows that the GA-MC-SPc is able to find the best know solution for 2 instances in the set ABZ (40%), 3 instances in the set FT (100%), 28 in the set LA (70%) and 9 in the set of ORB (90%).

We must think why the SPc operator shows better results. A strong possibility relates to the theory of *building blocks* (Goldberg [1]). This operator has only a cut, so it is more likely to not destroy blocks. Instead, the operator of two points cut large blocks and operator's uniform and flat ignores its existence.

This can be a very important result in choosing the crossover operator when using genetic algorithms, particularly in job shop.

Problem	*BKS*	*GA-MC-SPc*	*GA-MC-TPc*	*GA-MC-Uc*	*GA-MC-Fc*
ORB1 (10 x 10)	1059	1059	1059	1059	1059
ORB2 (10 x 10)	888	888	888	888	888
ORB3 (10 x 10)	1005	1005	1021	1020	1020
ORB4 (10 x 10)	1005	1005	1011	1011	1011
ORB5 (10 x 10)	887	887	889	889	889
ORB6 (10 x 10)	1010	1013	1013	1013	1013
ORB7 (10 x 10)	397	397	397	397	397
ORB8 (10 x 10)	899	899	899	899	899
ORB9 (10 x 10)	934	934	934	934	934
ORB10 (10 x 10)	944	944	944	944	944
% ARD		0,03%	0,27%	0,26%	0,28%

Table 4: Experimental results for instances ORB1-ORB10.

Problem	*BKS*	*GA-MC-SPc*	*GA-MC-TPc*	*GA-MC-Uc*	*GA-MC-Fc*
LA21 (15 x 10)	1046	1054	1052	1058	1059
LA22 (15 x 10)	927	932	935	932	935
LA23 (15 x 10)	1032	1032	1032	1032	1032
LA24 (15 x 10)	935	944	949	944	955
LA25 (15 x 10)	977	985	984	989	993
LA26 (20 x 10)	1218	1218	1218	1218	1218
LA27 (20 x 10)	1235	1252	1252	1266	1273
LA28 (20 x 10)	1216	1231	1234	1228	1248
LA29 (20 x 10)	1157	1180	1213	1193	1209
LA30 (20 x 10)	1355	1355	1355	1355	1355
LA31 (30 x 10)	1784	1784	1784	1784	1784
LA32 (30 x 10)	1850	1850	1850	1850	1850
LA33 (30 x 10)	1719	1719	1719	1719	1719
LA34 (30 x 10)	1721	1721	1721	1721	1721
LA35 (30 x 10)	1888	1888	1888	1888	1888
LA36 (15 x 15)	1268	1278	1282	1284	1284
LA37 (15 x 15)	1397	1408	1411	1422	1407
LA38 (15 x 15)	1196	1213	1219	1232	1247
LA39 (15 x 15)	1233	1244	1250	1250	1249
LA40 (15 x 15)	1222	1233	1246	1233	1245
ABZ7 (15 x 20)	656	688	695	690	693
ABZ8 (15 x 20)	645	704	708	702	701
ABZ9 (15 x 20)	661	715	720	724	721
% ARD		1,51%	1,89%	1,84%	2,13%

Table 5: Average relative deviation for problems greater than 10x10.

Additionally, the GA-MC-SPc results are compared against the most recent work in the literature.

The performance of GA-MC-SPc was evaluated on a standard set of 50 benchmark instances belonging to two classical sets known LA from

Lawrence [14] and ORB from Applegate and Cook [29].

Problem	*BKS*	*GA-MC-SPc*	*GA-MC-TPc*	*GA-MC-Uc*	*GA-MC-Fc*
LA01	666	666	666	666	666
LA02	655	655	655	655	655
LA03	597	597	597	597	597
LA04	590	590	590	590	590
LA05	593	593	593	593	593
LA06	926	926	926	926	926
LA07	890	890	890	890	890
LA08	863	863	863	863	863
LA09	951	951	951	951	951
LA10	958	958	958	958	958
LA11	1222	1222	1222	1222	1222
LA12	1039	1039	1039	1039	1039
LA13	1150	1150	1150	1150	1150
LA14	1292	1292	1292	1292	1292
LA15	1207	1207	1207	1207	1207
LA16	945	945	946	946	946
LA17	784	784	784	784	784
LA18	848	848	848	848	848
LA19	842	842	842	842	842
LA20	902	902	902	902	902
LA21	1046	1054	1052	1058	1059
LA22	927	932	935	932	935
LA23	1032	1032	1032	1032	1032
LA24	935	944	949	944	955
LA25	977	985	984	989	993
LA26	1218	1218	1218	1218	1218
LA27	1235	1252	1252	1266	1273
LA28	1216	1231	1234	1228	1248
LA29	1157	1180	1213	1193	1209
LA30	1355	1355	1355	1355	1355
LA31	1784	1784	1784	1784	1784
LA32	1850	1850	1850	1850	1850
LA33	1719	1719	1719	1719	1719
LA34	1721	1721	1721	1721	1721
LA35	1888	1888	1888	1888	1888
LA36	1268	1278	1282	1284	1284
LA37	1397	1408	1411	1422	1407
LA38	1196	1213	1219	1232	1247
LA39	1233	1244	1250	1250	1249
LA40	1222	1233	1246	1233	1245
ABZ5	1234	1234	1234	1234	1234
ABZ6	943	943	943	943	943
ABZ7	656	688	695	690	693
ABZ8	645	704	708	702	701
ABZ9	661	715	720	724	721
FT06	55	55	55	55	55
FT10	930	930	930	930	930
FT20	1165	1165	1165	1165	1165
ORB1	1059	1059	1059	1059	1059
ORB2	888	888	888	888	888
ORB3	1005	1005	1021	1020	1020
ORB4	1005	1005	1011	1011	1011
ORB5	887	887	889	889	889
ORB6	1010	1013	1013	1013	1013
ORB7	397	397	397	397	397
ORB8	899	899	899	899	899
ORB9	934	934	934	934	934
ORB10	944	944	944	944	944
% ARD		0,60%	0,80%	0,78%	0,89%

Table 6: Average relative deviation with all problems.

As different authors used different number of problems, the comparison is based only on those problems that authors considered. Note that the authors have different approaches to solve the JSSP problem.

We compare the obtained results with the aforementioned state of art approaches:

- Qing-dao-er-ji and Wang [38];
- Rego and Duarte [8];
- Hasan et al. [27];
- Gonçalves et al. [4];
- Aarts et al. [15];
- Dorndorf et al. [18];
- Binato et al. [20];
- Nowicki and Smutnicki [31];
- Sha and Hsu [6];
- Pardalos et al. [32];
- Adams et al. [10];
- Zhang et al. [35].

The result is showed in Table 7.

The algorithm GA-MC-SPc gives good results when compared to the state of art. In Table 7 we can see that the algorithm proposed is among the top ten best performing.

This study limited the number of generations for all operators tested. The number of generations was limited to 400. A higher number of generations for all operators would increase the computational time. Although there is convergence of the algorithms close to 400 generations, an increase of generations in the proposed algorithm can improve its global performance.

The experiments were performed on an Intel Core 2 Duo CPU T7250 @2.00 GHz. The computational times dispended are in the range [7, 3100] seconds.

NIS	*Test problems*	*Authors*	*Algorithm*	*ARD(%)*
40	LA01-LA40	**This paper**	**GA-MC-SPc**	**0.31**
		Qing-dao-er-ji and Wang [38]	HGA(1)	0.18
		Rego and Duarte [8]	F&F-PRD	0.29
		Hasan et al. [27]	GR-SA-RA	0.97
		Gonçalves et al. [4]	P. Active	0.41
		Gonçalves et al. [4]	Non-Delay	1.20
		Gonçalves et al. [4]	Active	1.10
		Aarts et al. [15]	GLS1	2.05
		Aarts et al. [15]	GLS2	1.75
		Dorndorf et al. [18]	PGA	4.00
		Dorndorf et al. [18]	SBGA	1.25
		Binato et al. [20]	-	1.87
		Nowicki and Smutnicki [31]	i-TSAB	0.05
		Sha and Hsu [6]	HPSO	0.02
		Sha and Hsu [6]	PSO	0.37
		Pardalos et al. [32]	GES	0.00
		Adams et al. [10]	SBI	3.67
10	ORB1-ORB10	**This paper**	**GA-MC-SPc**	**0.03**
		Nowicki and Smutnicki [31]	i-TSAB	0.00
		Zhang et al. [35]	TS	0.00
		Pardalos et al. [32]	GES	0.00

Table 7: Comparison of the % deviations for the different number of problems authors considered.

7 Conclusions and remarks

This paper presents an experimental study of a set of crossover operators, namely single-crossover, two-point crossover, uniform crossover and flat crossover. The single-crossover has the best results, followed by the uniform crossover operator.

This can be a very important result in choosing the crossover operator when using genetic algorithms, particularly in job shop.

Additionally, the best crossover operator was tested on a set of 50 standard instances taken from the literature and compared with the best state-of-the-art approaches. The algorithm produced good results when compared with other approaches.

Although there is convergence of the algorithms close to 400 generations, an increase of generations in the proposed algorithm (GA-MC-SPc) can improve its global performance.

Further work could be conducted to explore the possibility of genetically correct the chromosomes supplied by the genetic algorithm to reflect the solutions obtained by the local search heuristic.

Acknowledgements

This work has been partially supported by the CIDEM - Centre for Research & Development in Mechanical Engineering. CIDEM is a unit of FCT – Portuguese Foundation for the Science and Technology.

References:

[1] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, 1989.

[2] J. Magalhães-Mendes, Project scheduling under multiple resources constraints using a genetic algorithm, *WSEAS TRANSACTIONS on BUSINESS and ECONOMICS*, Issue 11, Volume 5, November 2008, pp. 487-496.

[3] I. Essafi, Y. Mati and S.D. Pérès, A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem, *Computers & Operations Research*, Vol. 35, Issue 8, 2008, pp. 2599-2616.

[4] J.F. Gonçalves, J.M. Mendes, and M.C.G. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, Vol. 167, 2005, pp. 77-95.

[5] J.J.M. Mendes, J.F. Gonçalves and M.G.C. Resende, A random key based genetic algorithm for the resource constrained project scheduling problem. *Comput. Oper. Res.* 36, 1, 2009, pp. 92-109.

[6] D. Y. Sha and C. Hsu, A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering*, 51, 4, 2006, pp. 791-808.

[7] F. Pezzela and E. Merelli, A tabu search method guided by shifting bottleneck for the job shop scheduling problem, *European Journal of Operational Research*, Vol. 120, 2000, pp. 297-310.

[8] C. Rego and R. Duarte, A filter-and-fan approach to the job shop scheduling problem, *European Journal of Operational Research*, Vol. 194, 2009, pp. 650–662.

[9] E. Nowicki and C. Smutnicki, A Fast Taboo Search Algorithm for the Job-Shop Problem, *Management Science*, Vol. 42, No. 6, 1996, pp. 797-813.

[10] J. Adams, E. Balas and Z. Zawack, *The shifting bottleneck procedure for job shop scheduling*, Management Science, Vol. 34, 1988, pp. 391-401.

[11] C. Y. Zhang, P. Li. and Z. Guan, A very fast TS/SA algorithm for the job shop scheduling problem, *Computers & Operations Research*, Vol. 35, 2008, pp. 282-294.

[12] H.R. Lourenço, Local optimization and the job-shop scheduling problem, *European Journal of Operational Research*, Vol. 83, 1995, pp. 347-364.

[13] H. Fisher and G.L. Thompson, *Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules*, in: Industrial Scheduling, J.F. Muth and G.L. Thompson (eds.), Prentice-Hall, Englewood Cliffs, NJ, 1963, pp. 225-251.

[14] S. Lawrence, *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques*, GSIA, Carnegie Mellon University, Pittsburgh, PA, 1984.

[15] E.H.L.Aarts, P.J.M.Van Laarhoven, J.K. Lenstra and N.L.J.Ulder, A computational study of local search algorithms for job shop scheduling, *ORSA Journal on Computing*, 6, 1994, pp. 118-125.

[16] R.M.Aiex, S.Binato and M.G.C. Resende, Parallel GRASP with Path-Relinking for Job Shop Scheduling, *Parallel Computing*, Vol. 29, Issue 4, 2003, pp. 393 - 430.

[17] F. Croce, R. Tadei, and G. Volta, A Genetic Algorithm for the Job Shop Problem, *Computers and Operations Research*, Vol. 22(1), 1995, pp. 15-24.

[18] U. Dorndorf, and E. Pesch, Evolution Based Learning in a Job Shop Environment, *Computers and Operations Research*, Vol. 22, 1995, pp. 25-40.

[19] L. Wang, and D. Zheng, An effective hybrid optimisation strategy for job-shop scheduling problems, *Computers & Operations Research*, Vol. 28, 2001, pp. 585-596.

[20] S. Binato, W.J.Hery, D.M. Loewenstern and M.G.C.Resende, *A GRASP for Job Shop Scheduling*. In: Essays and Surveys in Metaheuristics, Ribeiro, Celso C., Hansen, Pierre (Eds.), Kluwer Academic Publishers, 2002.

[21] B. Giffler and G.L. Thompson, Algorithms for Solving Production Scheduling Problems, *Operations Research*, Vol. 8(4), 1960, pp. 487-503.

[22] J. Carlier and E. Pinson, An Algorithm for Solving the Job Shop Problem. *Management Science*, Feb, 35(29), 1989, pp.164-176.

[23]J. Carlier and E. Pinson, A practical use of Jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, Vol. 26, 1990, pp. 269-287.

[24]P. Brucker, B. Jurisch and B. Sievers, A Branch and Bound Algorithm for Job-Shop Scheduling Problem, *Discrete Applied Mathematics*, Vol. 49, 1994, pp. 105-127.

[25]D. P.Williamson, L.A. Hall, J.A. Hoogeveen, , C. A. J.Hurkens, J. K. Lenstra, S. V. Sevastjanov and D. B. Shmoys, Short Shop Schedules, *Operations Research*, 45(2), 1997, pp. 288-294.

[26]J. Magalhães-Mendes, Project scheduling using a competitive genetic algorithm. In *Proceedings of the 8th WSEAS Conference on Simulation, Modelling and Optimization* (Santander, Cantabria, Spain). J. M. de la Maza and P. L. Espí, Eds. Mathematics And Computers In Science And Engineering. WSEAS, Stevens Point, Wisconsin, 2008, pp. 39-42.

[27]S.M.K. Hasan, R. Sarker and D. Cornforth, GA with Priority Rules for Solving Job-Shop Scheduling Problems. In *Proceedings of the IEEE Congress on Evolutionary Computation CEC(2008)*, 2008, pp. 1913-1920.

[28]M. J. Varnamkhasti, L. S. Lee, M. R. Bakar, and W. J. Leong, A Genetic Algorithm with Fuzzy Crossover Operator and Probability, *Advances in Operations Research*, vol. 2012, Article ID 956498, 16 pages, 2012. doi:10.1155/2012/956498

[29]D. Applegate and W. Cook, A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2), 1991, pp. 149-156.

[30]J. Magalhães-Mendes, A genetic algorithm for the job shop scheduling with a new local search using Monte Carlo method, In *Proceedings of the 10th WSEAS international conference on Artificial intelligence, knowledge engineering and data bases* (AIKED'11), Cambridge, UK, 2011, pp. 26-31.

[31]E. Nowicki and C. Smutnicki, An Advanced Tabu Search Algorithm for the Job Shop Problem, *Journal of Scheduling*, 8, 2005, pp.145–159.

[32]P. Pardalos, O. Shylo and A. Vazacopoulos, Solving job shop scheduling problems utilizing the properties of backbone and big valley, *Computational Optimization and Applications*, 47(1), 2010, pp.1-16.

[33]G. Sywerda, Uniform crossover in genetic algorithms, in Proceedings of the *3rd International Conference on Genetic Algorithms*, 1989, pp. 2-9.

[34]K. A. De Jong and W. M. Spears, A formal analysis of the role of multi-point crossover in genetic algorithms, *Annals of Mathematics and Artificial Intelligence*, vol. 5, no. 1, 1992, pp. 1–26.

[35]C.Y. Zhang, P.G. Li, Z.L. Guan, and Y.Q. Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11), 2007, pp. 3229-3242.

[36]Y.-Z. Luo, G.-J. Tang, Z.G. Wang and H.Y. Li. Optimization of perturbed and constrained fuel-optimal impulsive rendezvous using a hybrid approach. *Engineering Optimization*, 38(8), 2006, pp. 959-973.

[37]R. R. Sharapov. *Genetic Algorithms: Basic Ideas, Variants and Analysis*, Vision Systems: Segmentation and Pattern Recognition, Goro Obinata and Ashish Dutta (Ed.), 2007, pp. 407-422, ISBN: 978-3-902613-05-9.

[38]R. Qing-doa-er-ji and Y. Wang. A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, 39(10), 2012, pp. 2291-2299.

[39]H.R. Choi and B.J. Park, Genetic Algorithm for the Integration of Process Planning and Scheduling in a Job Shop, *WSEAS TRANSACTIONS on INFORMATION SCIENCE & APPLICATIONS*, Issue 12, Volume 3, December 2006, pp. 2498-2504.

[40]H. Chiu, K. Hsieh, Y.T. Tang and W. Chien, Employing a Genetic Algorithm Based on Knowledge to Address the Job Shop Scheduling Problem, *WSEAS TRANSACTIONS on COMPUTER RESEARCH*, Issue 2, Volume 2, February 2007, pp. 327-333.

[41]H. Chiu, K. Hsieh, Y.T. Tang and C.Y. Wang, A Novel Approach to Address the Job-Shop Scheduling Problem by using a Tabu Genetic Algorithm, *WSEAS TRANSACTIONS on COMPUTER RESEARCH*, Issue 2, Volume 2, February 2007, pp. 339-345.