THE ROLE OF DIFFERENT CROSSOVER METHODS WHEN SOLVING THE OPEN SHOP SCHEDULING PROBLEM VIA A SIMPLE EVOLUTIONARY APPROACH

Labarere I., Beraudo V., Salto C., Alfonso H.
Proyecto UNLPAM-09/F015[1]
Departamento de Informática - Facultad de Ingeniería
Universidad Nacional de La Pampa
Calle 110 esq. 9
(6360) General Pico – La Pampa – Rep. Argentina
e-mail: { saltoc, alfonsoh }@ing.unlpam.edu.ar
Phone: +54 2302 422780/422372, Ext. 6302

Gallard R.
Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC)[2]
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106
(5700) - San Luis -Argentina
e-mail: rgallard@unsl.edu.ar
Phone: +54 2652 420823
Fax : +54 2652 430224

ABSTRACT

The Open Shop Scheduling Problem (OSSP) is one of the most interesting, complexes and not frequently approached scheduling problems. Due to its intractability with other techniques, in this work we present an evolutionary approach to provide approximate solutions.

One of the most important points in an Evolutionary Algorithm is to determine how to represent individuals of the evolving population and then to decide suitable genetic operators. In this work, we use permutations as chromosomes. Dealing with permutations requires appropriate crossover operators to ensure feasible offspring. Usual operators are *partially-mapped*, *order*, *cycle and one-cut-point* crossover. The goal is to determine which is the most adequate for facing the OSSP with a simple evolutionary algorithm. Several known instances have been considered for testing in order to evaluate the algorithm behavior.

Keywords: Open Shop Scheduling, Evolutionary Computation, Crossover.

[1] The Research Group is supported by the Universidad Nacional de La Pampa.
[2] The LIDIC is supported by the Universidad Nacional de San Luis and the ANPCYT (National Agency to Promote Science and Technology).

1. Introduction

Evolutionary algorithms (EAs) can be used as techniques to solve problems inspired in natural evolution [4]. In an EA, a data structure, representing a feasible solution to a problem, is defined. Each possible data set admissible for that structure would be a potential solution to the problem. An EA behave as a search method, where the solutions to the problem are able to reproduce between them, combining characteristics and generating new solutions.

EAs have proved their ability to solve difficult scheduling problems. One of the most frequent models taken from real life is known as *open shop*. An Open Shop Scheduling Problem (OSSP) involves a collection of m machines and a collection of n jobs. Jobs can follow routes, which are open and arbitrarily decided by the scheduler. Each job consists of a set of operations, sometimes called *tasks* [8, 6]. Each machine can process at most one operation at a time and each job can be processed by at most one machine at any given time. The order in which the jobs are processed on the machine, can be chosen arbitrarily; but two or more tasks from the same job cannot be processed on the same machine. The objective in our OSSP is to determine a feasible combination of the machine and job orders, a schedule, which minimizes the overall finishing time, also known as the *makespan*. The makespan cannot be less than the maximum workload of each machine or the total processing time needed of n jobs.

Consider an OSSP where there are two machines and n jobs, denoted as $02||Cmax$ [6]. The makespan has to be minimized. Job j may be processed first on machine 1 and then on machine 2 or vice versa; the decision maker may determine the routes. With only two jobs, it can be easily verified that there are only two possible schedules and both have the same makespan. If $n > 3$ the problem belongs to the class of *NP-hard* problems.

The OSSP has many applications, specially in the manufacturing world and in industry. The common example is that of an automotive repair shop [10]. In such a shop, a typical job might involve the operations "spray-paint", and "change-tyres" to be performed on the same vehicle. These operations cannot usually be performed concurrently (especially if the stations at which these operations are performed are in different places, for instance), but can be performed in any order. Also it is usually true that different stations (i.e. "machines") can concurrently process operations from different jobs (e.g. involving different vehicles). If the operations in a job must be performed in some fixed order, then this becomes a Job Shop Scheduling Problem (JSSP).

In this work, we study the performance of evolutionary algorithms using different crossover operators, which are suitable for permutation representation.

The work is organized as follows. Section 2 presents a detailed problem description. Sections 3 and 4 describe the adopted representation and the crossovers methods implemented. Section 5 gives details on the experiments and section 6 discusses the results reached. Finally, conclusions and future works are presented.

2. The Open Shop Scheduling Problem

The OSSP consists of m machines $M_1, M_2, \ldots, M_m$ and n jobs $J_1, J_2, \ldots, J_n$. Each job J_i consists of m operations O_{ij}. The processing times are giving in an $m \times n$ matrix P where p_{ij} indicates the duration of the operation O_{ij} for the job j on the machine i, without preemption.

Operations for a job can be processed in any order, but only one at any giving time. We assume that each machine can process at most one operation at a time and each job can be processed by at most one machine at any given time. The order, in which the job is processed by the machines, can be chosen arbitrarily. The objective function to be minimized is the maximum time that is necessary to complete all jobs, or makespan.

An example follows. Table 1 describes a possible instance for the problem with 4 machines and 4 jobs. Table 2 shows a possible allocation of operations on the machines, and the corresponding Gantt diagram is showed in figure 1.

Machine	Job *J0*	Job *J1*	Job *J2*	Job *J3*
M0	34	15	38	95
M1	2	89	19	7
M2	54	70	28	34
M3	61	9	87	29

Table 1. A 4x4 benchmark problem for the OSSP.

Machine	Job	Operation	Operation Length	Start time	End time
M3	*J1*	O_{31}	9	0	9
M0	*J1*	O_{01}	15	9	24
M2	*J3*	O_{23}	34	0	34
M1	*J3*	O_{13}	7	34	41
M1	*J2*	O_{12}	19	0	19
M2	*J1*	O_{21}	70	34	104
M3	*J0*	O_{30}	61	9	70
M1	*J0*	O_{10}	2	70	72
M0	*J2*	O_{02}	38	24	62
M0	*J3*	O_{03}	95	62	157
M2	*J0*	O_{20}	54	104	158
M0	*J0*	O_{00}	34	157	191
M1	*J1*	O_{11}	89	104	193
M3	*J2*	O_{32}	87	70	157
M3	*J3*	O_{33}	29	157	186
M2	*J2*	O_{22}	28	158	186

Table 2. A schedule for the benchmark problem in table 1 with the makespan value of 193

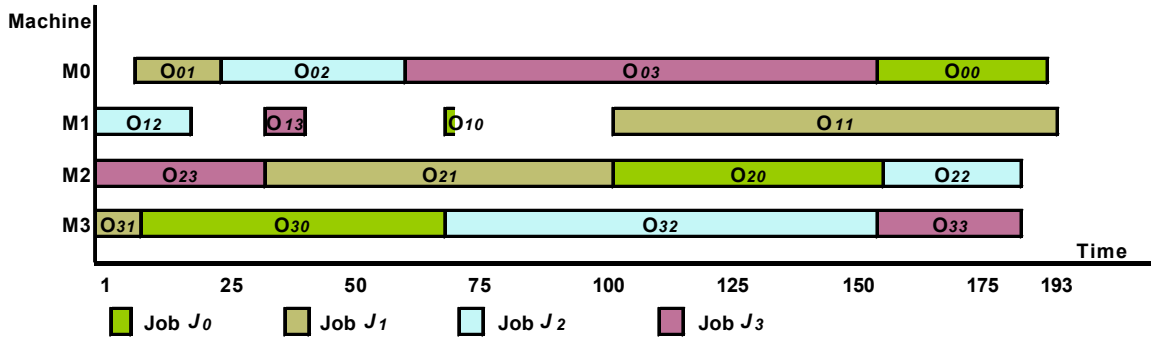


Figure 1. A schedule for the benchmark problem, OSSP 4x4, with minimum makespan.

3. Chromosome Representation

In this work we are using permutations for the chromosome representation: to each operation of a job is given a unique number. We first enumerate the operations of job J_1, then the operations of job J_2, and so on. Thus, for a problem with 4 jobs and 4 machines, the first operation of job J_1 will be given number 0, and the last operation of job J_4 will be given number 15. For individual chromosomes of the EA, we use strings of length p, where p is the total number of operations involved. A scheduling of the operations is represented by using an integer string, $y_1, y_2,\ldots, y_p$, where the value of y_i represents the operation to be scheduled next. Thus, if we have 4 jobs and 4 machines, the following string (figure 2) represents a possible schedule of the sixteen operations.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	5	13	14	9	7	2	3	8	12	0	1	6	11	15	10

Figure 2. A chromosome representation for a 4x4 instance of OSSP.

This representation is interpreted as follows. First, it schedules the operation that has number 4 (the first operation of J_1), then, the operation with number 5 (the second operation of J_1), and so on.
The procedure for calculating makespan consists in scheduling the operations with the goal of finishing an operation at the earliest possible time. This is not as obvious as it might seem. Consider the following scenario (see Figure 3): we want to schedule operation O_c from job J_c of length t on machine M_j. Our algorithm scans the operations that have been already scheduled on M_j. In case a "gap" exists between two consecutive operations, O_a and O_b, such that $y - x >= t$ (where x is the completion time of O_a and y is the start time of O_b,) then the algorithm checks if the operation O_c can be scheduled between x and y. This is done, only if no other operation from job J_c is currently being processed on some other machine between times x and y.
If no such gap exists, then operation O_c is scheduled sometime after the last operation that was processed on machine M_j.

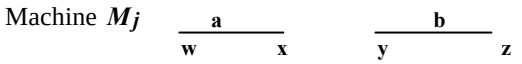


Figure 3. Scheduling operation c between operations a and b on machine *Mj.*

4. Crossover Operators

For this work we select the permutation representation and then in order to obtain feasible schedules after each recombination, adequate genetic operators are needed. They are:

- *partial-mapped crossover* (PMX). PMX was proposed by Goldberg and Lingle [2] and can be viewed as an extension of two-point crossover for binary string to permutation representation. It uses a special repairing procedure to resolve the illegitimacy caused by the simple two-point crossover. Thus the essentials of PMX are a simple two-point crossover plus a repairing procedure.
- *order crossover* (OX). OX was proposed by Davis [1]. It can be viewed as a variation of PMX with a different repairing procedure.
- *cycle crossover* (CX). CX was proposed by Oliver, Smith and Holland [5]. Essentially it can be viewed as a class of uniform crossover to permutation representation together with a repairing procedure. It takes some alleles from one of the parents and selects others from the other parent. Alleles of the first parent are selected in order to define a cycle according to the positions between the parents.
- *one-cut-point crossover (OCPX).* OCPX was proposed by Reeves [7]. It is a kind of one-point crossover for permutation representation. It defines a cut point, and then takes all the alleles from the first parent until that position and completes the offspring with the alleles according to their occurrence in the other parent.

5. Experiments

The algorithms were tested for a set of selected instances of the Open Shop Scheduling Problem that are well known in the literature [9], Taillard gives the pseudocode to generate the individual problem instances. We implemented, in Borland C, Taillard's algorithm and executed it to generate 40 benchmark problems. The generator takes as input: time seed, machine seed and the problem size, which are specified in [9]. We have worked with all the instances for small size problems (4x4 and 5x5) and the medium size problems (7x7 and 10x10). For each instance, a series of fifty runs were performed for each crossover operator (OCPX, OX, CX, PMX). The EA used proportional selection for mating. For mutation, a simple *interchange operator*, which randomly exchanges genes of the chromosome, was designed. The population size was fixed at 100 individuals for all instances. The maximum number of generations was fixed at 1000 and probabilities for crossover

and mutation were set at 0.8 and 0.1, respectively. These values were determined as the best combination of probabilities after many initial trials.
The following relevant performance variables were chosen:

- **Ebest** = (Abs(*opt_val* - best value)/*opt_val*)100
  It is the percentile error of the best found individual when compared with the known, or estimated, optimum value *opt_val*. It gives us a measure of how far the best individual is from that *opt_val*.
- **Epop** = (Abs(*opt_val*- pop mean fitness)/*opt_val*)100
  It is the percentile error of the population mean fitness when compared with *opt_val*. It tell us how far the mean fitness is from that *opt_val*.
- **#Opt** = It indicates the number of times that the known, or estimated, optimum value is obtained during several experiments made.
- **MEbest** and **MEpop** indicate the mean values for Ebest and Epop, respectively.

6. Results

Tables 3 and 4 report the results of the algorithm using different crossover methods on the 4x4 instances. In all of them, the optimum value for each instance was reached, independently of the crossover type; the exception was in the instance Mat43 under the CX, but the *Ebest* value (0.8) is very close to the optimum. Analyzing #Opt, OX obtained the biggest number of occurrences in most cases, followed by PMX, OCPX and finally CX. Now considering mean *Ebest*, best results are obtained with OX, followed by PMX, OCPX and CX.

Instance	OCPX			PMX			OX			CX		
	#opt	ebest	mebest	#opt	ebest	mebest	#opt	ebest	mebest	#opt	ebest	mebest
Mat40	11	0.0000	1.7824	6	0.0000	1.0259	10	0.0000	0.9326	8	0.0000	1.8549
Mat41	2	0.0000	1.6186	9	0.0000	1.4915	12	0.0000	1.2373	6	0.0000	2.4746
Mat42	7	0.0000	0.3838	20	0.0000	0.2214	24	0.0000	0.1919	2	0.0000	0.8044
Mat43	3	0.0000	1.2320	6	0.0000	1.3760	3	0.0000	1.1440	0	0.8000	2.1200
Mat44	6	0.0000	2.0339	12	0.0000	1.6881	12	0.0000	1.6949	4	0.0000	3.0102
Mat45	27	0.0000	1.3439	20	0.0000	1.2275	19	0.0000	1.8624	6	0.0000	3.1534
Mat46	16	0.0000	0.9055	11	0.0000	1.0547	3	0.0000	1.1244	3	0.0000	1.8806
Mat47	18	0.0000	1.0783	5	0.0000	1.7880	26	0.0000	0.7650	4	0.0000	1.8525
Mat48	11	0.0000	2.2605	8	0.0000	2.2912	17	0.0000	1.6322	9	0.0000	3.0421
Mat49	9	0.0000	2.3963	36	0.0000	0.3779	36	0.0000	0.5714	6	0.0000	2.4516

Table 3. ebest results for 4x4 instances.

Instance	OCPX		PMX		OX		CX	
	epop	mepop	epop	mepop	epop	mepop	epop	mepop
Mat40	23.4109	36.0747	47.9888	56.2099	40.7619	53.4364	5.9358	11.4604
Mat41	18.9086	36.8223	47.6049	53.3789	40.3621	50.6051	5.9629	12.6370
Mat42	18.1065	30.8318	43.5274	47.8076	38.6551	45.4717	13.8400	10.4499
Mat43	20.0173	37.6874	46.8763	53.5512	46.6567	51.7476	5.5634	11.5779
Mat44	20.8616	39.0704	49.8481	54.2613	43.9801	51.7308	4.5784	12.3258
Mat45	8.2756	37.8797	53.4804	61.3702	44.8413	59.3113	6.4487	13.9492
Mat46	9.4037	34.2632	45.6479	53.4267	32.9484	49.8518	3.6633	11.5271
Mat47	6.0330	30.6541	37.8355	62.3665	46.2057	61.2375	5.1697	10.8525
Mat48	13.5176	34.8459	53.4773	59.2208	51.4398	57.8281	5.8659	13.2079
Mat49	16.5270	33.1802	46.1708	50.5378	42.8912	50.2298	2.7374	12.0814

Table 4. epop results for 4x4 instances.

Table 4 details *Epop* results. Lower values were found with CX, varying between 2.7 and 13.8%; this indicates that the population is fairly centred on the best-found value (as we saw previously, in

only one instance the optimum value was not reached). Then, it is followed by OCPX, which shows an error ranging between 6 and 23%. Higher *Epop* values are observed with the application of OX and PMX, whose values are almost twice as much those showed by other methods.

Although OX obtained the best *Ebest* results, the population behaves better with CX in average.

Table 5 exhibits results on the 5x5 instances. Here, only both Mat50 instance under CX and Mat51 under OX reached the optimum value. Best results were achieved by OCPX, CX and OX, but no one brings out the other. Analyzing MEbest values, OCPX presents good results, followed for CX, OX and PMX.

Epop values on the 5x5 instances are described in table 6. The lowest levels of *Epop* were found with CX, whose values go from 6.2 to 11.21%; OCPX follows it with an error ranging between 32.4 and 44.3%. Remaining crossover methods exhibit *Epop* values all over 50%, but results reached for PMX are the worst ones.

Evaluating results obtained on 7x7 instance (table 7), a crossover ranking can be established considering *Ebest* results: CX, OCPX, OX and finally PMX. The latest two show remarkable higher errors. The same ranking can be observed for mean *Ebest* values.

Analyzing *Epop* values (table 8) the same crossover ranking is preserved here. Moreover it matches the same ranking presented on previous 4x4 and 5x5 instances

Tables 9 y 10 introduce results corresponding to 10x10 instances. Here, the same behaviour on both *Ebest* and *Epop* values is observed: CX in the first place, then OCPX, OX and finally PMX.

Instance	OCPX			PMX			OX			CX		
	#opt	ebest	mebest	#opt	ebest	mebest	#opt	Ebest	mebest	#opt	ebest	mebest
Mat50	0	1.6667	5.7800	0	1.0000	7.9000	0	0.3333	6.8000	1	0.0000	6.5467
Mat51	0	1.5267	6.0229	0	3.4351	8.3282	1	0.0000	6.3740	0	1.5267	6.3130
Mat52	0	3.7152	8.3529	0	6.1920	11.3870	0	4.9536	11.1022	0	3.0960	8.3653
Mat53	0	2.5806	8.2839	0	3.8710	12.6000	0	1.9355	10.8516	0	5.4839	9.4323
Mat54	0	2.7607	7.8344	0	2.4540	11.2699	0	4.9080	10.5153	0	2.7607	8.1595
Mat55	0	2.5641	7.7436	0	2.8846	11.1154	0	3.8462	8.8846	0	2.5641	7.2756
Mat56	0	2.9703	7.9274	0	3.3003	10.5215	0	2.3102	9.3663	0	1.3201	7.7624
Mat57	0	1.3333	6.3267	0	3.6667	9.8133	0	2.6667	8.0067	0	1.6667	6.5533
Mat58	0	1.1331	7.0765	0	1.1331	9.9377	0	1.6997	8.7989	0	2.2663	7.5184
Mat59	0	1.8405	7.5706	0	3.0675	10.5583	0	4.6012	9.9755	0	3.0675	7.2822

Table 5. ebest results for 5x5 instancias.

Instance	OCPX		PMX		OX		CX	
	Epop	mepop	epop	mepop	epop	mepop	epop	mepop
Mat50	35.7890	48.0589	56.4269	62.2417	53.6392	59.5873	6.8393	18.4225
Mat51	34.8742	49.0347	56.7402	64.8935	54.7649	61.0515	6.2199	17.7527
Mat52	44.2850	52.2842	61.4160	66.6979	58.8878	64.2265	9.8863	19.6888
Mat53	39.3764	54.0097	65.5028	69.4677	58.1226	66.6563	11.2184	20.7248
Mat54	32.4003	50.7764	62.7619	66.6487	56.6426	65.0553	8.2019	19.7695
Mat55	42.1066	51.6408	62.2816	67.2295	59.4266	64.1871	11.2184	18.9530
Mat56	41.2108	52.6391	60.2320	66.1630	58.5800	65.0227	9.6134	19.4897
Mat57	35.6116	48.3666	58.5249	64.5336	53.8045	62.0988	8.4083	16.7813
Mat58	36.0607	50.0923	60.8160	64.5034	54.9705	62.6435	8.0242	18.7984
Mat59	39.2151	51.5956	62.2325	65.5376	56.3927	63.6676	10.1430	18.6864

Tabla 6 . epop values for 5x5 instances.

Instance	OCPX			PMX			OX			CX		
	#opt	ebest	mebest	#opt	ebest	mebest	#opt	ebest	mebest	#opt	ebest	mebest
Mat70	0	9.5890	14.8721	0	12.1005	18.5936	0	9.3607	16.6438	0	3.8813	10.3333
Mat71	0	7.1269	15.8396	0	12.9176	20.8018	0	13.1403	18.2539	0	4.6771	11.1581
Mat72	0	9.6033	18.0835	0	11.2735	22.6347	0	14.6138	20.7641	0	5.8455	13.0939
Mat73	0	7.4946	14.9422	0	11.9914	20.1456	0	12.4197	18.4154	0	5.5675	11.0707
Mat74	0	10.2625	16.6205	0	12.1718	20.6444	0	14.5585	19.7566	0	5.7279	11.7327
Mat75	0	9.5652	17.5783	0	13.9130	23.1957	0	11.5217	20.8217	0	4.5652	13.2391
Mat76	0	9.8851	17.2368	0	14.9425	22.9379	0	13.3333	21.0069	0	4.8276	13.3149
Mat77	0	9.6244	14.5728	0	11.2676	19.0141	0	11.0329	16.9812	0	5.1643	10.9624
Mat78	0	6.5217	15.5043	0	13.2609	20.1304	0	11.7391	17.0957	0	4.1304	11.1261
Mat79	0	10.7500	17.8300	0	13.2500	21.7050	0	13.7500	20.3650	0	6.5000	11.9200

Table 7. ebest values for 7x7 instances.

Instance	OCPX		PMX		OX		CX	
	epop	mepop	epop	mepop	epop	mepop	epop	mepop
Mat70	43.3194	54.8069	61.7965	67.1995	57.3683	63.8090	10.1774	22.7061
Mat71	46.3777	57.7936	67.0472	69.7787	61.1039	67.4021	14.9775	24.1770
Mat72	47.0279	58.9272	67.7334	70.8012	64.2437	68.2968	16.5996	26.8816
Mat73	38.9990	54.0460	63.5970	67.2085	58.7923	63.9556	13.5723	24.2843
Mat74	45.6723	57.6537	65.9494	69.7041	62.1950	66.9763	13.9652	24.6742
Mat75	49.7233	58.8904	68.0571	72.1303	63.1417	68.8114	16.2404	25.7376
Mat76	50.2511	59.4891	67.1649	71.9833	60.5037	69.3969	15.3238	26.7919
Mat77	43.9033	54.2849	63.0676	67.5670	59.2106	64.7806	13.4904	23.6322
Mat78	37.8769	55.0435	63.4307	67.3368	58.8338	63.8739	15.4853	25.2915
Mat79	47.8212	59.7214	69.2195	73.2935	63.0734	69.5887	16.8440	25.3123

Table 8. epop values for 7x7 instances.

Instance	OCPX			PMX			OX			CX		
	#opt	ebest	mebest	#opt	ebest	mebest	#opt	ebest	mebest	#opt	ebest	mebest
Mat100	0	24.3411	29.1039	0	24.4961	33.4636	0	22.0155	31.0078	0	17.3643	22.8155
Mat101	0	17.8571	26.5816	0	26.3605	31.6565	0	20.5782	29.3639	0	13.9456	20.1122
Mat102	0	18.3306	26.3175	0	25.6956	31.8069	0	21.9313	28.4550	0	12.7660	20.5565
Mat103	0	21.4905	26.5858	0	26.6898	31.1612	0	19.0641	28.4125	0	14.3847	20.9324
Mat104	0	18.8768	26.8736	0	24.6490	32.1154	0	23.7129	29.7910	0	15.1326	21.7129
Mat105	0	22.3048	29.5093	0	28.6245	34.6357	0	21.1896	32.1636	0	16.9145	23.6022
Mat106	0	18.6196	26.0289	0	24.0770	30.7352	0	22.4719	28.4045	0	10.1124	19.1974
Mat107	0	20.1342	26.5973	0	23.1544	31.4228	0	17.2819	28.1275	0	13.2550	20.0839
Mat108	0	19.6639	27.9462	0	26.3866	33.3580	0	23.1933	30.8605	0	15.4622	22.8269
Mat109	0	17.9402	25.4120	0	21.7608	30.3887	0	22.0930	27.6346	0	12.2924	19.3621

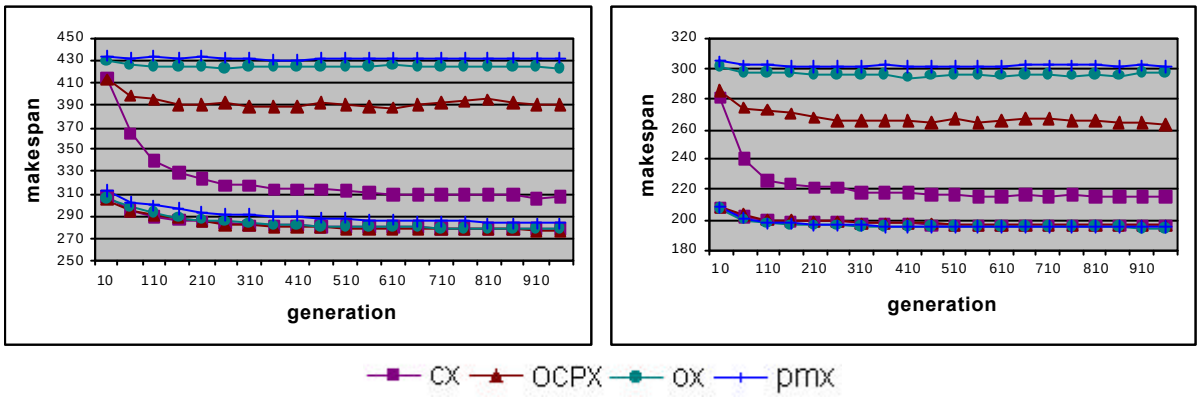
Table 9. ebest results for 10x10 instances.

Instance	OCPX		PMX		OX		CX	
	epop	Mepop	epop	mepop	epop	mepop	epop	mepop
Mat100	55.7180	62.4777	71.5070	73.8312	67.4040	71.4458	27.5660	37.5863
Mat101	50.3510	60.5000	69.6992	72.8551	65.8009	70.5147	22.7332	34.0526
Mat102	50.4205	60.1335	68.2775	71.6965	62.1060	68.6121	20.8567	34.5906
Mat103	52.5470	60.7264	67.2913	71.8538	63.8804	69.3153	27.6176	35.7719
Mat104	51.7378	61.2430	69.2007	72.5177	66.5922	70.5284	27.3043	36.8116
Mat105	55.2660	63.9792	72.6298	76.9366	70.3335	74.1149	27.6176	38.6999
Mat106	52.5901	60.4353	67.6839	70.7704	62.7713	68.5515	21.1674	33.8335
Mat107	52.8072	60.7251	69.6044	72.9874	65.6795	70.1579	22.7785	34.3411
Mat108	54.2745	62.4257	70.7354	74.3941	67.3351	72.0496	28.3389	37.1650
Mat109	49.3047	58.6597	66.5349	70.8761	64.5970	68.8694	21.4699	33.8517

Table 10. epop results for 10x10 instances.

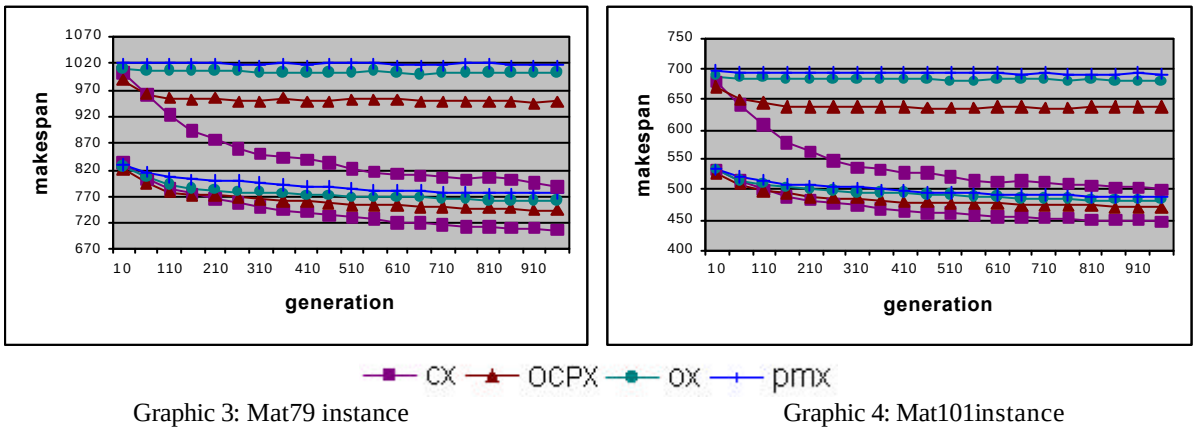
Graphics 1 to 4 show the evolution over the generations of both the best individual (lower curves) and the mean population makespan (higher curves). X axis represents the number of generations and the Y axis the makespan values. A representative matrix instance was selected for each problem size according to the complexity level. The curves are obtained by averaging the results from the fifty experiments made for each instance. Mat40 instance was the one selected from 4x4 set (graphic 1). A superposition for all the crossover methods is observed among curves representing the progress of the best value; in particular with CX, the mean value curve is very near to the best value. In all instances, the mean population makespan remains constant and quite high through the generations with PMX and OX. Using CX and arriving to generation 200, an approaching to the best makespan found is observed, such a process of improvement continues in a notorious way as soon as the complexity of the problems is increased. Now, with OCPX the biggest approach to the best value found is detected between generations 100 and 150, from this point on the mean population error does not vary so much.

In Mat40 instance (graphic 1), the evolution of the best individual is independent of the crossover method. In Mat51 (graphic 2), changes produced in the best individual are similar in CX, OX and OCPX, but in PMX the results are not so good as those showed by other crossover methods. In Mat79 and Mat101 instances, graphics 3 and 4 respectively, the curve representing the evolution of the best individual for CX is beneath others.



Graphic 1: Mat40 instance

Graphic 2: Mat51 instance



Graphic 3: Mat79 instance

Graphic 4: Mat101instance

7. Conclusions

In this work we presented a simple evolutionary algorithm as an alternative technique to solve the open shop scheduling problem. The representation of solutions selected is a permutation of

operations. For that it was necessary to consider well-designed crossover and mutation operators in order to obtain feasible offspring after each mating action. Particularly, the crossover operators selected and contrasted were those proposed for the traveling salesman problem (TSP): partial-mapped, order, cycle and one-cut point crossover. Some problem instances with different complexity level were used to evaluate the behaviour of our algorithm. Results from the application of different crossover methods were contrasted. Some observations and suggestions about the use of these algorithms were given.

Analyzing results obtained for instances with lesser complexity, we can remark that, regarding quality of results, OX provides better results than other crossover methods. When the instance complexity is increased, the algorithm using CX is the one that reaches solutions closer to the optimum value.

Independently of the instance complexity, we concluded that using CX operator levels of error in the population lower than employing the others crossover methods, were reached. However when that complexity is increased higher errors were shown.

Next steps will be oriented to develop more refined algorithms considering other representations, incorporating multiplicity characteristics that were applied to other scheduling problems.

8. Acknowledgements

We acknowledge the cooperation of the project group for providing new ideas and constructive criticisms. Also to the Universidad Nacional de San Luis, the Universidad Nacional de La Pampa, and the ANPCYT from which we receive continuous support.

9. References

[1] Davis, L., Applying adaptive algorithms to domains, in Proceeding of the International Joint Conference on Artificial Intelligence, pp. 162-164, 1985.

[2] Goldberg, D. and Lingle R., Alleles, loci and traveling salesman problem, in Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, Mj, 1985

[3] Fang, H., Ross, P. and Corne, D., A promising hybrid GA/Heuristic approach for open-shop scheduling problems, in Proceedings of 11th European Conference on Artificial Intelligence ECAI 94, pp. 590-594, 1994.

[4] Michalewicz, Z., *Genetic Algorithm + Data Structure = Evolution Programs,* 3ª ed., Springer-Verlag, New York, 1996.

[5] Oliver I.M., Smith D.J. and Holland J.R.C. A study of permutation crossover operators on the travelling salesman problem, in Proceedings of the 2nd International Conference on Genetic Algorithms, pp. 224-230, 1987.

[6] Pinedo, M, *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, 1995.

[7] Reeves C., A genetic algorithms for flow shop sequencing, in Computer and Operations Research, Vol. 22, pp, 5-13, 1995.

[8] Khuri, S. and Miryala, S.R., Genetic algorithms for solving open shop scheduling problems, in Proccedings of the 9th Portuguese Conference on Artificial Intelligence, EPIA1999, Springer, pág. 357-368, 1999.

[9] Taillard E., Benchmarks for basic scheduling problems, in European Journal of Operational Research, vol. 64, nro. 2, pág. 278-285, 1993.

[10] Gonzalez, T and Sahni, S., Open shop scheduling to minimize finish time, Journal of the Association for Computing Machinery, 23(4), 665-679, 1976.