Adding problem-specific knowledge in Evolutionary Algorithms to Solve W-T Scheduling Problems

de San Pedro M., Pandolfi D., Villagra A., Lasso M.,Vilanova G.
Proyecto UNPA-29/B032[1]
División Tecnología
Unidad Académica Caleta Olivia
Universidad Nacional de La Patagonia Austral
Ruta 3 Acceso Norte s/n
(9011) Caleta Olivia – Santa Cruz - Argentina
e-mail: { edesanpedro, dpandolfi, avillagra, mlasso}@uaco.unpa.edu.ar; gvilanov@satlink.com
Phone/Fax : +54 0297 4854888

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

In a production system it is usual to stress minimum tardiness to achieve higher client satisfaction. According to the client relevance, job processing costs and requirements, and various other considerations, a weight is assigned to each job. An important, non-trivial, problem is to minimize weighted tardiness.

Evolutionary algorithms (EAs) have been proved as efficient tools to solve scheduling problems. Latest improvements in EAs have been developed by means of multirecombination, a method which allows multiple exchange of genetic material between individuals of the mating pool.

As EAs are blind search methods this paper proposes to insert problem-specific-knowledge by recombining potential solutions (individuals of the evolving population) with seeds, which are solutions provided by other heuristics specifically intended to solve the scheduling problem under study.

In this work we describe two main approaches where seeds are inserted either in the initial population or as a part of every mating pool during evolution. Both methods were contrasted for a set of problem instances extracted from the OR-Library. An outline of the weighted tardiness problem in a single machine environment, details of implementation and results are discussed.

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

1. Introduction

Single-machine scheduling problems are of interest as the basis to more complex machine environments. In particular the Total Weigthed Tardiness problem ($1||\sum w_j T_j$) [3, 14], a NP-hard problem, is a generalization of the Total Tardiness problem ($1||\sum T$). It is important to have heuristics providing reasonably good schedules with tolerable computational effort because Branch and Bound and other partial enumeration based methods are prohibitively time consuming even with only 20 jobs.
Among other heuristics [11], evolutionary algorithms (EAs) have been successfully applied to solve scheduling problems [16,17]. Current trends in evolutionary algorithms make use of multiparent [4, 5, 6] and multirecombinative approaches [7, 8, 9]. The latter we called, *multiple-crossovers-on-multiple-parents* (MCMP). Instead of applying crossover once on a pair of parents this feature applies n_1 crossover operations on a set of n_2 parents. In order to improve the balance between exploration and exploitation in the search process a variant called MCMP-SRI [12], recombines a breeding individual (stud) by repeatedly mating individuals that randomly immigrates to a mating pool. Under this approach the random immigrants incorporate exploration (making unnecessary the use of mutation operations) and the multi-mating operation with the stud incorporates exploitation to the search process. .
In this work MCMP-SRSI, a novel variant, which considers the inclusion of a stud-breeding individual in a pool of random and seed-immigrant parents [18], is presented. Here, the seeds generated by conventional heuristics, and continuously present, introduce the problem-specific knowledge.
Two approaches are considered. The first approach generates the initial population with neighbours of three selected seeds and in subsequent generations applies MCMP-SRI. The second one applies MCMP-SRSI. Next sections describe the weighted tardiness scheduling problem, the alternative ways to insert *problem-specific knowledge* and discuss the results obtained.

2. The weighted tardiness scheduling problem

The single-machine total weighted tardiness problem [14] can be stated as follows: n jobs are to be processed without interruption on a single machine that can handle no more than one job at a time. Job j (j = 1,...,n) becomes available for processing at time zero, requires an uninterrupted positive processing time *pj* on the machine, has a positive weight *wj*, and a due date *dj* by which it should ideally be finished. For a given processing order of the jobs, the earliest completion time *Cj* and the tardiness $Tj = \max\{Cj-dj,0\}$ of job *j* can readily be computed. The problem is to find a processing order of the jobs with minimum total weighted tardiness

$$\sum_{j=1}^{n} w_j T_j$$

Even with this simple formulation, this model leads to an optimization problem that is NP-Hard [14].

3. Multirecombination of random and seed immigrants with the stud

Multiple Crossovers per Couple (MCPC) [7], [8] and Multiple Crossovers on Multiple Parents (MCMP) [9] are multirecombination methods, which improve EAs performance by reinforcing and balancing exploration and exploitation in the search process. In particular, MCMP is an extension of MCPC where the multiparent approach of Eiben [4], [5], [6] is introduced. Results obtained in diverse single and multiobjective optimization problems indicated that the searching space is efficiently exploited by the multiple application of crossovers and efficiently explored by the greater number of samples provided by the multiple parents.

A further extension of MCMP is known as MCMP-SRI [12], [13]. This approach considered the mating of an evolved individual (the stud) with random immigrants. In our new variant (MCMP-SRSI), the process for creating offspring is performed as follows. From the old population the stud, is selected by means of proportional selection and inserted in the mating pool together with three seed immigrants, created by means of three different heuristics. The number of n_2 parents in the mating pool is completed with randomly created individuals (random immigrants). The stud mates every other parent, the couples undergo partial mapped crossover (PMX) and 2* n_2 offspring are created. The best of these 2* n_2 offspring is stored in a temporary children pool. The crossover operation is repeated n_1 times, for different cut points each time, until the children pool is completed. Finally, the best offspring created from n_2 parents and n_1 crossover is inserted in the new population.

4. Introducing problem-specific knowledge

Dispatching heuristics are methods that allow deciding which job should be inserted into the system. To do this, a rule assigns an index to every job and the one with the highest priority is selected. There are different heuristics [11] for the Total Weighted Tardiness problem. Some of them are earliest due date first (EDD), weighted shortest processing time first (WSPT), Montagne, R&M and Covert, whose principal property is not only the quality of the results, but also to give an ordering of the jobs (schedule) close to the optimal sequence. Unlike EAs, they were specifically conceived to solve this problem and consequently their inherent problem-specific knowledge provides to the evolutionary algorithm with genetic information of the promising areas for exploitation. For the problem under study we chose the following heuristics:

Rachamadagu and Morton Heuristic (R&M). This heuristic provides a schedule according to the following expression.

$$p_j = (w_j / p_j)[\exp\{-(S_j)^+ / kp_{av}\}]$$

with $Sj = [dj - (pj + Ch)]$ is the slack of job j at time Ch, where Ch is the total processing time of the jobs already scheduled, k is a parameter of the method (usually k =2.0) and p_{av} is the average processing time of jobs competing for top priority. In the R&M heuristic, also called the *Apparent Tardiness Cost* heuristic, jobs are scheduled one at a time and every time a machine becomes free a ranking index is computed for each remaining job. The job with the highest-ranking index is then selected to be processed next.

The Covert rule. This heuristic provides a schedule according to the following expression.

$$p_j = (w_j / p_j)\{1-(S_j)^+ / kp_j\}^+$$

Under this heuristic the WSPT rule is modified by a slack factor, and processing times different than the job being considered are not taking into account. The Covert rule works, in the case of a single resource similar to R&M.

Modified R&M heuristic. Here a logarithmic function of the slack of the job considered and the average processing times of remaining jobs are used,

$$p_j = (w_j / p_j)[\ln\{-(S_j)^+ / kp_{av}\}]$$

The idea of inserting seeds (good solutions) in evolutionary processes was originally implemented by Reeves [16] as an alternative way to introduce problem-specific knowledge to the algorithm. In his approach Reeves, inserted one seed provided by a non evolutionary heuristic, only once in the initial population, expecting that its genetic material were occasionally exchanged by means of the selection mechanism.
This paper proposes two different approaches for the insertion of knowledge in the evolutionary process. In the first approach this knowledge is inserted in the initial population, while in the second approach this knowledge is continuously inserted by mean of seeds introduced during the execution of the recombination process (mating pool).

First approach
In the first approach the initial population is generated as the neighbourhood of seeds provided from a robust heuristic and subsequently MCMP-SRI is applied. As initial solutions had goods features, genotypic as well as phenotypic, this permits to start the search in a promissory subspace allowing evolutionary algorithms to converge towards the optimum quicker than if the population were randomly generated.
In this kind of scheduling problems, solutions (individuals of a population) are usually permutations of job identifiers, which represent schedules. To generate the neighbourhood of the seeds the operator used must start by a solution represented by a permutation and generate a new solution that differ a little from de original. Special attention must be paid in this process because new solutions must remain valid. We selected *adjacency* and *inversion* as neighbourhood operators.
The adjacency operator creates a new neighbour by changing the original schedule as follows. From a random selected position a job is interchanged with anyone of its adjacent jobs, either at left or right, with the same probability. The maximum number of different neighbours that could be generated by this operator is l (if l is the number of jobs for a schedule). Since there are three seeds the total number of neighbours that will be generated is $3*l$. To generate the neighbourhood with the adjacency operator the following steps should be accomplished:

Step 1: Insert the three selected seeds (R&M, Covert and Modified R&M) in the population.
Step 2: Select randomly the seed to which the adjacency operator will be applied.
Step 3: Determine a position (job) in the seed to be interchanged. The schedule (list of jobs) should be considered circular.
Step 4: Generate a neighbour by interchanging the selected job with the one at its left or right position.
Step 5: Insert the neighbour in the population and return to step 2 to generate new neighbours until the population is completed.

The inversion operator generates a new neighbour by changing the original schedule as follows. The new neighbour is created by random selection of two positions and exchanging them. The maximum number the neighbours that could be generated by this operator is $l * (l-1)/ 2$ (if l is the number of jobs for a schedule). Since there are three seeds then the total number of neighbours that could be generated is ($l * (l-1)/2) * 3$). To generate a neighbourhood with the inversion operator the following steps should be accomplished:
:

Step 1: Insert the three selected seeds (R&M, Covert and Modified R&M) in the population.
Step 2: Select randomly the seed to which the inversion operator will be applied.
Step 3: Determinate two position in the seed and interchange the jobs.
Step 4: Insert the neighbour in the population and return to step 2 to generate new neighbours until the population is completed.

Second approach
In the second approach MCMP-SRSI is used. Here the problem-specific knowledge is continuously inserted in the mating pool by means of the three selected seeds, which are always present. So, in the mating pool the seeds replace three of the random immigrants. In order to avoid premature convergence, adjacency and inversion are used as mutation operators.

5. Experimental tests and results

The following algorithms where designed:

MCMP-SRI-AD: Initial population is built as the neighbourhood of three seeds by means of the adjacency operator.
MCMP-SRI-IN: Initial population is built as the neighbourhood of three seeds by means of the inversion operator.
MCMP-SRSI-AD: Randomized initial population. Seeds continuously remain in the mating pool. Adjacency is used as a mutation operator.
MCMP-SRSI-IN: Randomized initial population. Seeds continuously remain in the mating pool. Inversion is used as a mutation operator.

All evolutionary algorithms were tested for selected instances from OR-library benchmarks [1,2]. To compare the algorithms, the following relevant performance variables were chosen:

Ebest = ((best value - opt_val)/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 on how far the best individual is from that opt_val.
Hit Ratio. Denotes the percentage of runs where the algorithm finds the best benchmark value (optimum or lower bound) . Its value is 1 when the best benchmark value is reached in every run.
Gbest. It is the generation where the best individual was found.

The best parameter setting was determined under each algorithm after a series of initial trials, as follows:

Approach	max_gen	Pop_size	Pc	Pm	n1	n2
MCPC-SRSI-AD	200	100	0.65	0.05	14	16
MCPC-SRSI-IN	200	100	0.65	0.05	14	16
MCPC-SRI-AD	200	100	0.65	0.00	14	16
MCPC-SRI-IN	200	100	0.65	0.00	14	16

The following tables summarize minimum, mean and general average values for the performance variables through all selected instances for the 40 and 50 job problem sizes.

In table 1 results show that MCMP-SRI-AD hits the best benchmark published for 40 and 50 jobs problems size with an average hit ratio of 61 % and 41 %, respectively. Regarding *Ebest*, its average value is 0.34% and 0.24% for 40 and 50 jobs, respectively. The number of generations, *Gbest*, required to find the best individual is 36.06 and 65.35 in average for 40 and 50 jobs problems size, respectively.

Table 1. Performance variables values under MCMP-SRI-AD, for the 40 and 50 jobs instances.

Instance K	Best Benchm.	Min WT	Mean Ebest	Hit Ratio	Mean Gbest	Instance K	Best Benchm.	Min WT	Mean Ebest	Hit Ratio	Mean Gbest
wt40-1	913	913	0.00	1.0	1.0	**wt50-1**	2134	2134	0.00	1.00	11.60
wt40-6	6955	6955	0.00	1.0	4.4	**wt50-6**	26276	26415	0.99	0.00	123.30
wt40-11	17465	17465	0.03	0.9	1.1	**Wt50-14**	51785	51887	0.22	0.00	73.80
wt40-19	77122	77132	0.09	0.0	97.9	**Wt50-19**	89299	89474	0.20	0.00	57.70
wt40-21	77774	77774	0.01	0.1	36.7	**Wt50-21**	214546	214567	0.02	0.00	77.50
wt40-26	108	108	0.00	1.0	1.0	**Wt50-26**	2	2	0.00	1.00	1.00
wt40-31	6575	6575	0.00	1.0	1.0	**Wt50-31**	9934	10015	2.54	0.00	113.90
wt40-41	57640	57907	0.66	0.0	115.7	**Wt50-44**	123893	123893	0.01	0.70	39.20
wt40-46	64451	64455	0.04	0.0	1.0	**Wt50-46**	157505	157505	0.00	0.30	43.10
wt40-51	0	0	0.00	1.0	1.0	**Wt50-51**	0	0	0.00	1.00	1.00
wt40-56	2099	2099	5.66	0.2	68.4	**Wt50-56**	1258	1258	0.04	0.90	34.60
wt40-66	65386	65386	0.02	0.1	75.3	**Wt50-66**	76878	76887	0.05	0.00	126.40
wt40-71	90486	90486	0.00	1.0	1.0	**Wt50-71**	150580	150580	0.08	0.10	115.80
wt40-76	0	0	0.00	1.0	1.0	**Wt50-76**	0	0	0.00	1.00	1.00
wt40-91	47683	47683	0.01	0.7	110.8	**Wt50-91**	89298	89490	0.25	0.00	124.00
wt40-96	126048	126048	0.00	1.0	1.0	**Wt50-96**	177909	177912	0.04	0.00	133.90
wt40-101	0	0	0.00	1.0	1.0	**Wt50-101**	0	0	0.00	1.00	1.00
wt40-106	0	0	0.00	1.0	1.0	**Wt50-106**	0	0	0.00	1.00	1.00
wt40-116	46770	46839	0.26	0.0	90.1	**Wt50-116**	35727	35727	0.15	0.10	120.00
wt40-121	122266	122266	0.03	0.1	110.8	**Wt50-121**	78315	78448	0.22	0.00	107.10
Mean Avg			**0.34**	**0.61**	**36.06**	**Mean Avg**			**0.24**	**0.41**	**65.35**

Table 2. Performance variables values under MCMP-SRI-IN, for the 40 and 50 jobs instances.

Instance K	Best Benchm.	Min WT	Mean Ebest	Hit Ratio	Mean Gbest	Instance K	Best Benchm.	Min WT	Mean Ebest	Hit Ratio	Mean Gbest
wt40-1	913	913	0.00	1.0	1.0	**Wt50-1**	2134	2134	0.00	1.00	12.20
wt40-6	6955	6955	0.00	1.0	37.5	**Wt50-6**	26276	26415	1.13	0.00	138.00
wt40-11	17465	17465	0.00	1.0	2.4	**Wt50-14**	51785	51887	0.23	0.00	96.40
wt40-19	77122	77132	0.10	0.0	145.0	**Wt50-19**	89299	89474	0.21	0.00	75.90
wt40-21	77774	77774	0.01	0.4	41.1	**Wt50-21**	214546	214566	0.02	0.00	92.70
wt40-26	108	108	0.00	1.0	1.0	**Wt50-26**	2	2	0.00	1.00	1.00
wt40-31	6575	6575	0.00	1.0	1.0	**Wt50-31**	9934	10015	3.44	0.00	78.60
wt40-41	57640	57876	0.68	0.0	137.6	**Wt50-44**	123893	123893	0.03	0.20	50.70
wt40-46	64451	64451	0.04	0.1	23.4	**Wt50-46**	157505	157505	0.00	0.50	90.50
wt40-51	0	0	0.00	1.0	1.0	**Wt50-51**	0	0	0.00	1.00	1.00
wt40-56	2099	2099	5.23	0.4	100.3	**Wt50-56**	1258	1258	0.00	1.00	42.60
wt40-66	65386	65386	0.05	1.0	89.4	**Wt50-66**	76878	76906	0.08	0.00	86.50
wt40-71	90486	90486	0.00	1.0	1.0	**Wt50-71**	150580	150604	0.09	0.00	124.60
wt40-76	0	0	0.00	1.0	1.0	**Wt50-76**	0	0	0.00	1.00	1.00
wt40-91	47683	47683	0.03	0.6	146.6	**Wt50-91**	89298	89525	0.28	0.00	87.30
wt40-96	126048	126048	0.00	1.0	1.0	**Wt50-96**	177909	177909	0.05	0.20	108.50
wt40-101	0	0	0.00	1.0	1.0	**Wt50-101**	0	0	0.00	1.00	1.00
wt40-106	0	0	0.00	1.0	1.0	**Wt50-106**	0	0	0.00	1.00	1.00
wt40-116	46770	46802	0.21	0.0	117.4	**Wt50-116**	35727	35757	0.29	0.00	117.60
wt40-121	122266	122281	0.10	0.0	80.1	**Wt50-121**	78315	78448	0.22	0.00	83.60
Mean Avg			**0.32**	**0.68**	**46.49**	**Mean Avg**			**0.30**	**0.40**	**64.54**

In table 2 we can see that MCMP-SRI-IN hits the best benchmark published for 40 and 50 jobs problems size with an average hit ratio of 68 % and 40 %, respectively. Concerning *Ebest*, its average value is 0.32% and 0.30% for 40 and 50 jobs, respectively. The number of generations, *Gbest*, required to find the best individual is 46.49 and 64.54 in average for 40 and 50 jobs problems size, respectively.

Table 3. Performance variables values under MCMP-SRSI-AD, for the 40 and 50 jobs instances.

Instance K	Best Benchm.	Min WT	Mean Ebest	Hit Ratio	Mean Gbest	Instance K	Best Benchm.	Min WT	Mean Ebest	Hit Ratio	Mean Gbest
wt40-1	913	913	0.00	1.0	1.0	**wt50-1**	2134	2134	0.00	1.00	7.80
wt40-6	6955	6955	0.00	1.0	2.7	**wt50-6**	26276	26276	0.36	0.30	83.50
wt40-11	17465	17465	0.00	1.0	3.5	**Wt50-14**	51785	51785	0.00	1.00	19.10
wt40-19	77122	77122	0.00	1.0	52.4	**Wt50-19**	89299	89299	0.00	1.00	24.30
wt40-21	77774	77774	0.00	1.0	8.7	**Wt50-21**	214546	214546	0.00	0.20	47.30
wt40-26	108	108	0.00	1.0	1.0	**Wt50-26**	2	2	0.00	1.00	1.00
wt40-31	6575	6575	0.00	1.0	1.0	**Wt50-31**	9934	9934	0.37	0.80	108.50
wt40-41	57640	57876	0.41	0.0	20.4	**Wt50-44**	123893	123893	0.00	1.00	3.80
wt40-46	64451	64451	0.00	1.0	7.9	**Wt50-46**	157505	157505	0.00	1.00	2.80
wt40-51	0	0	0.00	1.0	1.0	**Wt50-51**	0	0	0.00	1.00	1.00
wt40-56	2099	2099	4.36	0.5	47.5	**Wt50-56**	1258	1258	0.00	1.00	16.40
wt40-66	65386	65386	0.00	0.9	17.6	**Wt50-66**	76878	76878	0.00	1.00	95.10
wt40-71	90486	90486	0.00	1.0	1.4	**Wt50-71**	150580	150580	0.00	1.00	33.50
wt40-76	0	0	0.00	1.0	1.0	**Wt50-76**	0	0	0.00	1.00	1.00
wt40-91	47683	47683	0.00	1.0	20.3	**Wt50-91**	89298	89448	0.17	0.00	73.90
wt40-96	126048	126048	0.00	1.0	1.3	**Wt50-96**	177909	177909	0.00	1.00	74.70
wt40-101	0	0	0.00	1.0	1.0	**Wt50-101**	0	0	0.00	1.00	1.00
wt40-106	0	0	0.00	1.0	1.0	**Wt50-106**	0	0	0.00	1.00	1.00
wt40-116	46770	46770	0.00	0.7	145.5	**Wt50-116**	35727	35727	0.00	1.00	9.00
wt40-121	122266	122266	0.00	1.0	52.3	**Wt50-121**	78315	78315	0.04	0.40	53.50
Mean Avg			**0.24**	**0.91**	**19.43**	**Mean Avg**			**0.05**	**0.84**	**32.91**

Table 4. Performance variables values under MCMP-SRSI-IN, for the 40 and 50 jobs instances.

Instance K	Best Benchm.	Min WT	Mean Ebest	Hit Ratio	Mean Gbest	Instance K	Best Benchm.	Min WT	Mean Ebest	Hit Ratio	Mean Gbest
wt40-1	913	913	0.00	1.0	1.0	**wt50-1**	2134	2134	0.00	1.00	9.80
wt40-6	6955	6955	0.00	1.0	2.3	**wt50-6**	26276	26276	0.51	0.10	114.40
wt40-11	17465	17465	0.00	1.0	3.5	**wt50-11**	51785	51785	0.00	1.00	15.70
wt40-19	77122	77122	0.00	1.0	66.7	**wt50-19**	89299	89299	0.00	1.00	25.20
wt40-21	77774	77774	0.00	1.0	15.6	**wt50-21**	214546	214555	0.01	0.00	30.70
wt40-26	108	108	0.00	1.0	1.0	**wt50-26**	2	2	0.00	1.00	1.00
wt40-31	6575	6575	0.00	1.0	1.0	**wt50-31**	9934	9934	0.95	0.40	140.00
wt40-41	57640	57876	0.41	0.0	19.0	**wt50-41**	123893	123893	0.00	1.00	4.30
wt40-46	64451	64451	0.00	1.0	12.6	**wt50-46**	157505	157505	0.00	1.00	2.60
wt40-51	0	0	0.00	1.0	1.0	**wt50-51**	0	0	0.00	1.00	1.00
wt40-56	2099	2099	6.97	0.2	35.5	**wt50-56**	1258	1258	0.00	1.00	23.30
wt40-66	65386	65386	0.00	0.9	19.8	**wt50-66**	76878	76878	0.00	0.80	91.10
wt40-71	90486	90486	0.00	1.0	1.7	**wt50-71**	150580	150580	0.00	1.00	56.60
wt40-76	0	0	0.00	1.0	1.0	**wt50-76**	0	0	0.00	1.00	1.00
wt40-91	47683	47683	0.00	1.0	22.7	**wt50-91**	89298	89448	0.17	0.00	85.50
wt40-96	126048	126048	0.00	1.0	1.3	**wt50-96**	177909	177909	0.00	0.90	59.80
wt40-101	0	0	0.00	1.0	1.0	**wt50-101**	0	0	0.00	1.00	1.00
wt40-106	0	0	0.00	1.0	1.0	**wt50-106**	0	0	0.00	1.00	1.00
Wt40-116	46770	46770	0.00	1.0	121.3	**wt50-116**	35727	35727	0.00	1.00	10.90
Wt40-121	122266	122266	0.00	1.0	41.9	**wt50-121**	78315	78315	0.02	0.70	67.90
Mean Avg			**0.37**	**0.91**	**18.55**	**Mean Avg**			**0.08**	**0.80**	**37.14**

In table 3 results show that MCMP-SRSI-AD hits the best benchmark published for 40 and 50 jobs problems size with an average hit ratio of 91 % and 84 %, respectively. About *Ebest*, its average value is 0.24% and 0.05% for 40 and 50 jobs, respectively. The number of generations, *Gbest*, required to find the best individual is 19.43 and 32.91 in average for 40 and 50 jobs problems size, respectively.

Table 4 shows that MCMP-SRSI-IN hits the best benchmark published for 40 and 50 jobs problems size with an average hit ratio of 91 % and 80 %, respectively. Regarding *Ebest*, its average value is 0.37% and 0.08% for 40 and 50 jobs, respectively. The number of generations, *Gbest*, required to find the best individual is 18.55 and 37.14 in average for 40 and 50 jobs problems size, respectively.

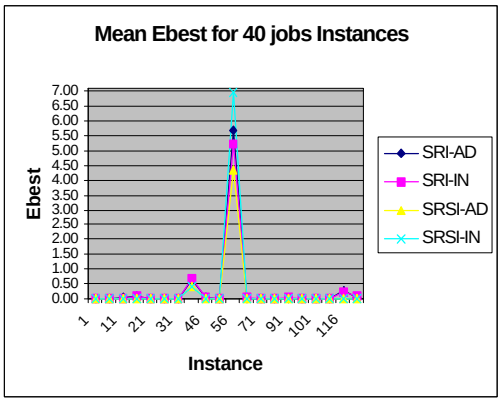
Table 5. Average performance variables values under each approach, for the 40 and 50 job problem sizes.

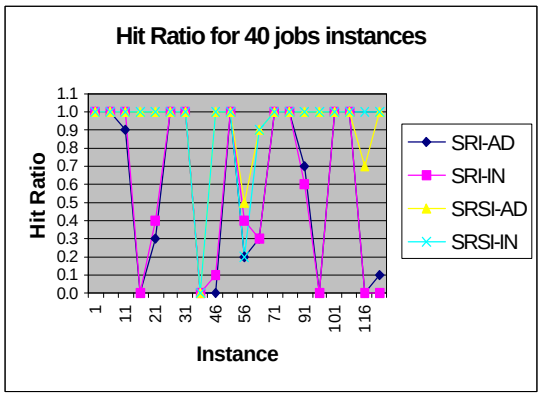
Approach	Av. Ebest		Av. Gbest		Av. Hit Ratio	
	40 jobs	50 jobs	40 jobs	50 jobs	40 jobs	50 jobs
MCPC-SRI-AD	0.34	0.24	36.0	65.4	61	41
MCPC-SRI-IN	0.32	0.30	46.5	64.5	68	40
MCPC-SRSI-AD	**0.24**	**0.05**	19.4	**32.9**	**91**	**84**
MCPC-SRSI-IN	0.37	0.08	**18.6**	37.4	**91**	80

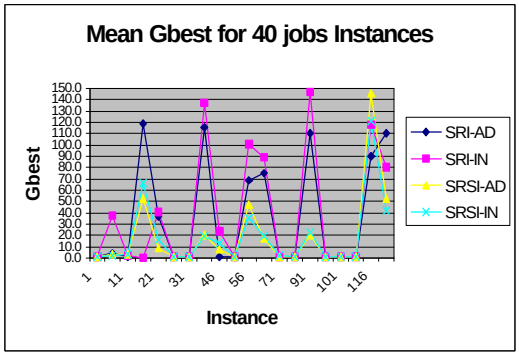
From table 5 we conclude that MCMP-SRSI-AD is the best performer for both problem sizes with showing low *Gbest* average values.

Figures 1 and 2 , show the behaviour of each algorithm on each problem size.

Figure 1. Values of the performance variables obtained under both evolutionary approaches for the 40 jobs instances, (a) Mean Ebest, (b) Mean Gbest, (c) Hit Ratio.







If we concentrate on the 40 jobs problem size we can observe that mean *Ebest* for any approach show a maximum for the wt40-56 instance, ranging from 4% to 7%, while for the remaining instances the

mean percentile error is smaller than 1%. In general MCMP-SRSI show smaller or equal mean percentile error than MCMP-SRI in all instances.

Concerning Hit Ratio, MCMP-SRSI approaches do not reach the benchmark only in one instance (wt40-41), while they reaches the benchmark, in every run, for at least 16 of the 20 instances.
On the other hand MCMP-SRI approaches do not reach the benchmark in four instances (wt40-41), while they reaches the benchmark, in every run, for at most 11 of the 20 instances.

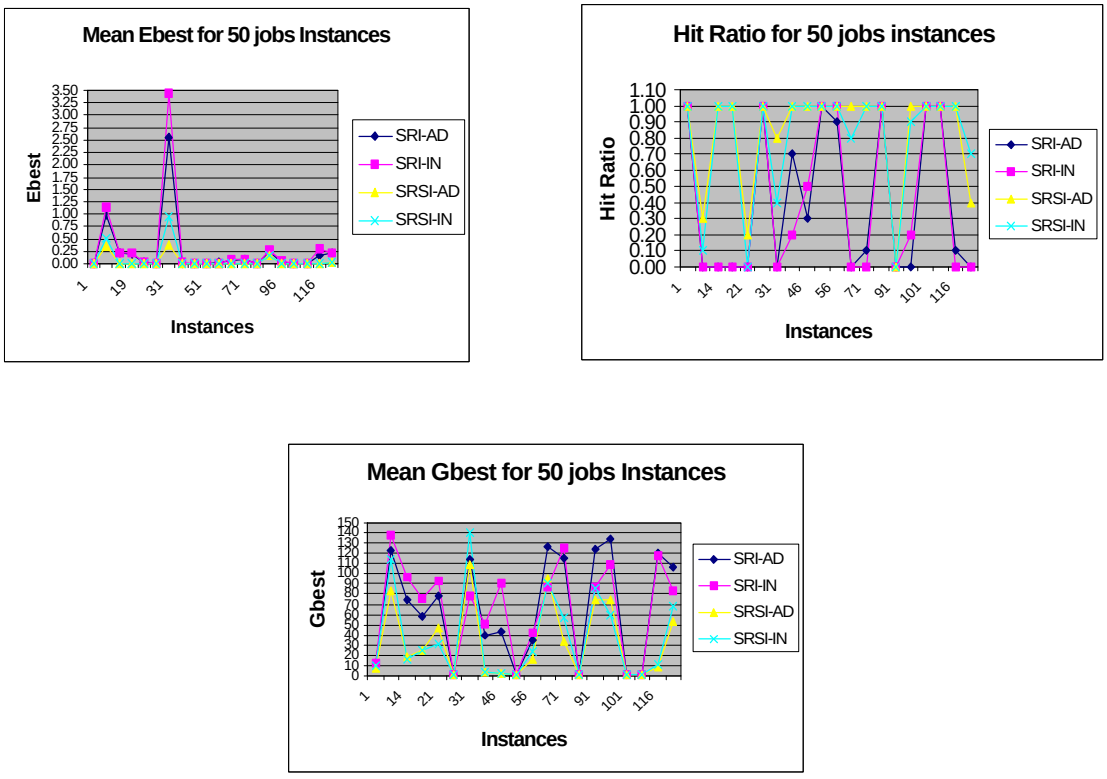
Regarding *Gbest* we can observe that MCMP-SRSI approaches require smaller computational effort than those based in MCMP-SRI.

When looking at the 50 jobs problem size we can observe that for the algorithms based in MCMP-SRSI mean *Ebest* values show a maximum smaller than 1%. In general MCMP-SRSI show smaller or equal mean percentile error than MCMP-SRI in all instances.

Concerning Hit Ratio, MCMP-SRSI approaches do not reach the benchmark only in two s (wt40-41), while they reaches the benchmark, in every run, for at least 15 of the 20 instances.
On the other hand MCMP-SRI approaches do not reach the benchmark in 10 instances (wt50-21 and wt50-91), while they reaches the benchmark, in every run, for at most 7 of the 20 instances.

Regarding *Gbest* we can see that MCMP-SRSI approaches require smaller computational effort than those based in MCMP-SRI.

Figure 2. Values of the performance variables obtained under both evolutionary approaches for the 50 jobs instances, (a) Mean Ebest, (b) Mean Gbest, (c) Hit Ratio.



6. Conclusions

Evolutionary algorithms are robust search algorithms in the sense that they provide good solutions to a broad class of problems which otherwise are computationally intractable. But their robustness has as a drawback the kind of search process they perform: a blind search that slightly addressed by the relative fitness of the solutions, completely ignores the nature of the problem. In order to improve their performance we faced two ways for inserting problem-specific-knowledge by recombining seeds in the evolutionary process. Seeds are good solutions provided by heuristics specifically designed for the problem under our concern; the weighted tardiness single-machine scheduling problem. Variants of MCMP-SRI include seeds and their neighbourhood in the initial population and then the recombination of the stud (breeding individual) and the random immigrants follows in successive generations. Variants of MCMP-SRSI, starting from a randomized initial population, continuously include the seeds in the mating pool to be recombined with the stud and the random immigrants. As indicated by the results this latter approach outperforms the former. In particular MCMP-SRSI-AD is the best performer for both problem sizes. It worth saying here, that both methods produced solutions of higher quality than those of previous works where the insertion of problemm-specific-knowledge was not considered. Further work will be dedicated to find alternative ways to guide the evolutionary search for different scheduling problems.

7. Acknowledgements

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

8. References

[1] J.E.Beasley "Common Due Date Scheduling", OR Library, http://mscmga.ms.ic.ac.uk/

[2] H.A.J.Crauwels, C.N.Potts and L.N.Van Wassenhove "Local search heuristics for the single machine total weighted tardiness scheduling problem", *Informs Journal on Computing* 10, 341-350. 1998.

[3] T.Chen and M.Gupta, "Survey of scheduling research involving due date determination decision", *European Journal of Operational Research,* vol 38, pp. 156-166, 1989.

[4] A.E.Eiben, P.E.Raué, and Z.Ruttkay, "Genetic algorithms with multi-parent recombination", *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature,* Springer-Verlag, 1994, number 866 in LNCS, pp. 78-87.

[5] A.E. Eiben, C.H.M. Van Kemenade, and J.N. Kok, "Orgy in the computer: Multi-parent reproduction in genetic algorithms". Proceedings of the *3rd European Conference on Artificial Life*, Springer-Verlag, 1995, number 929 in LNAI, pages 934-945.

[6] A.E. Eiben and. Th. Bäck, "An empirical investigation of multi-parent recombination operators in evolution strategies". Evolutionary Computation, 5(3):347-365, 1997.

[7] S. Esquivel, A. Leiva, R. Gallard, "Multiple Crossover per Couple in Genetic Algorithms", Proceedings of the *Fourth IEEE Conference on Evolutionary Computation (ICEC'97)*, Indianapolis, USA, April 1997, pp 103-106.

[8] S. Esquivel, A. Leiva, R. Gallard, "Couple Fitness Based Selection with Multiple Crossover per Couple in Genetic Algorithms". Proceedings of the *International Symposium on Engineering of Intelligent Systems (EIS´98)*, La Laguna, Tenerife, Spain, February 1998, pp 235-241.

[9] S. Esquivel, H. Leiva,.R. Gallard, "Multiple crossovers between multiple parents to improve search in evolutionary algorithms", Proceedings of the *Congress on Evolutionary Computation (IEEE)*. Washington DC, 1999, pp 1589-1594.

[10] M. Michalewicz, "*Genetic Algorithms + Data Structures = Evolution Programs"*. Third revised edition, Springer, 1996.

[11] T. Morton, D. Pentico, *"Heuristic scheduling systems"*, Wiley series in Engineering and technology management. John Wiley and Sons, INC, 1993.

[12] D. Pandolfi, G. Vilanova, M. De San Pedro, A. Villagra, "Multirecombining studs and immigrants in evolutionary algorithm to face earliness-tardiness scheduling problems". Proceedings of the *International Conference in Soft Computing*. University of Paisley, Scotland, U.K., June2001, pp.138

[13] D. Pandolfi; G. Vilanova; M. De San Pedro; A. Villagra, R. Gallard: "Solving the Single-Machine Common Due Date Problem Via Stud and Inmigrants in Evolutionary Computation", *World Multiconference on Systemics, Cybernetics and Informatics*, Orlando July 2001 pp 409-413

[14] M. Pinedo, "*Scheduling: Theory, Algorithms and System.*" First edition Prentice Hall, 1995.

[15] R.V. Rachamadugu, T.E. Morton, "Myopic heuristics for the single machine weighted tardiness problem". *GSIA*, Carnigie Mellon University, Pittsburgh, PA. 1982., Working paper 30-82-83.

[16] C. Reeves, "A genetic algorithm for flow shop sequencing", *Computers and Operations Research*, vol 22, pp5-13, 1995.

[17] Y. Tsujimura, M. Gen, E. Kubota: "Flow shop scheduling with fuzzy processing time using genetic algorithms". *The 11th Fuzzy Systems Symposium*, Okinawa,. 1995,.pp 248-252.

[18] Pandolfi D., de San Pedro M., Villagra A., Vilanova G., Gallard R. – "Multirecombining random and seed immigrants in evolutionary algorithms to solve W-T scheduling problems"- Proceedings ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business and Application (CSITeA-02), Foz Iguazu, Brazil 2002.