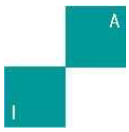
Inteligencia Artificial 46(2010), 84-95
doi: 10.4114/ia.v14i46.1550



INTELIGENCIA ARTIFICIAL

http://erevista.aepia.org/

An ACO approach for the Parallel Machines Scheduling Problem

Claudia Ruth Gatica, Susana Cecilia Esquivel, Guillermo Mario Leguizamón

Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC)
Universidad Nacional de San Luis.
Ejército de los Andes 950, (5700) San Luis, Argentina.
crgatica,esquivel,legui@unsl.edu.ar

Abstract The parallel machines scheduling problem (PMSP) comprises the allocation of jobs on the resources of the systems, i.e., a group of machines in parallel. The basic model consists of m identical machines and n jobs. The jobs are assigned according to resource availability following some allocation rule. In this work, we apply the Ant Colony Optimization (ACO) metaheuristic which includes four different specific heuristics in the solution construction process to solve unrestricted PMSP for the minimization of the *Maximum Tardiness* (T_{max}) objective. We also present a comparison of previous results obtained by a simple Genetic Algorithm (GAs), and an evidence of an improved performance of the ACO metaheuristic on this particular scheduling problem.

Keywords: Parallel Machine Scheduling, Maximum Tardiness, Ant Colony Optimization Algorithms, Specific Heuristic Problem Information.

1 Introduction

The unrestricted PMSP is considered in this paper. The problem consists of scheduling n jobs on m identical parallel machines (P_m) to minimize the T_{max}. There are no constraints in the assignement of the jobs to the machines, therefore the problem is described by ($P_m||T_{max}$). This problem belongs to more basic model of PMSP which is NP-hard, even when $m = 2$ [19]. The PMSP are representative of many real world problems. In such systems it is usual to deal with minimizations of the objectives based on the due dates, such as the T_{max}. In the scheduling literature there are several methods of resolutions for PMSP. In [18], [19] a set of dispatching rules and heuristics are presented. In [16] the PMSP was solved with approximation algorithms based on linear and integer programming. In [7] a memetic algorithm was developed, in [6] a column generation based on exact descomposition algorithm was presented. Evolutionary Algorithms (EAs) with multirecombination methods [10], [11], and knowledge insertion into problem [12] have been successfully applied to solve PMSP. Similarly, the ACO metaheuristic has also been applied to solve scheduling problems, such as [8] for Travel Salesman Problem, [4] and [5] for Single Machine Total Tardiness Problem, [13] for Job Shop Scheduling Problem, [3] for Scheduling Task Graphs, [21] Flow Shop Scheduling Problem, [22] for Parallel Machine Shop, [2] for PMSP with setup times, and [20] for PMSP with precedence constraints. All works mentioned above were tackled with different problems, different instances and objective functions, except ([10], [11], and [12]).

In this work we propose an Ant Colony System (ACS) [8], an advanced algorithm derived from the ACO metaheuristic, to applied unrestricted PMSP to minimize the T_{max}. It is important to note that the

ISSN: 1988-3064(on-line)

pheromone trails and heuristic information are the driving forces in ant algorithms to efficiently explore the search space. In the particular case of the heuristic information, different rules can be incorporated according to the problem under consideration. For the scheduling problem studied in this work, the ACS was implemented by considering different specific heuristics, which were based on the following rules: Earliest Due Date (EDD), Shortest Processing Time (SPT), Longest Processing Time (LPT), and Least Slack (SLACK). In addition, we compared the results with results obtained by simple Genetic Algorithms belonging to previous works by Ferretti et al. [10].

The remainder of the paper is organized as follows. Section 2 introduces the Parallel Machines Scheduling Problem. In Section 3, the Ant Colony Optimization metaheuristic and ACS algorithm for PMSP are presented. The experimental design is described in Section 4. In Section 5, the results are showed. Section 6 provides our conclusions.

2 Parallel Machines Scheduling Problem

The formal notation used in the literature [19] for the scheduling problem that we are dealing is a triplet: $(P_m \mid\mid T_{max})$. The first field describes the machine environment P_m, the second one contains the restrictions, we note that our problem is unrestricted, therefore this field is empty, and the third one provides the objective function T_{max}. This scheduling problem can be stated as follows: there are n jobs to be processed without interruption on some of the m identical machines belonging to the system P_m; each machine can process not more than one job at a time. Job *j (j=1,2,...n)* is made available for the processing at time zero, it requires an uninterrupted positive processing time p_j on a machine and it has a due date d_j by which it should ideally be finished. For a given processing order of the jobs (schedule), the earliest completion time C_j and the maximum delay time $T_j = \{C_j - d_j, 0\}$ of the job j can be easily estimated. The problem consists in finding a optimum schedule objective value. The objective to be minimized is:

$$MaximumTardiness : T_{max} = max_j(T_j)$$

The problems related to the due dates have received considerable attention from a practical and theorical point of view, besides, they have considered as NP-Hard when $2 \leq m \leq n$, check the literature [19].

2.1 Conventional Heuristics to Scheduling Problems

Dispatching heuristics assign a priority index to every job in a waiting queue. The one with the highest priority is selected to be processed next. There are different heuristics (e.g., [19] and [18]) for the above mentioned problem whose principal property is not only the quality of the results but also give a schedule closest the optimal sequence. The following dispatching heuristics were selected to determine priorities, and they were used to build schedules by the Ant Colony System:

- *EDD* (Earliest Due Date first): the job with earliest due date is selected first and the final scheduled jobs are ordered satisfying:
$$d_1 \leq d_2 \leq ... \leq d_n.$$
- *SPT* (Shortest Processing Time first): the job with shortest processing time is selected first and the final scheduled jobs are ordered satisfying:
$$p_1 \leq p_2 \leq ... \leq p_n.$$
- *LPT* (Largest Processing Time first): the job with largest processing time is selected first and the final scheduled jobs are ordered satisfying:
$$p_n \leq p_{n-1} \leq ... \leq p_1.$$
- *SLACK* (Least slack): the job with the smallest difference between due date and processing time is selected first and the final scheduled jobs are ordered satisfying:
$$d_1 - p_1 \leq d_2 - p_2 \leq ... \leq d_n - p_n.$$

3 Ant Colony Optimization

Ant Colony Optimization (ACO) is a metaheuristic in which a base process is the *solutions construction*, (see in the bibliography [9]). It manages a colony of ants that concurrently and asynchronously visit adjacent states of the considered problem by moving through neighbor nodes of the construction graph G. They move by applying a stochastic local decision policy, denoted by P_{ij}, which is descripted in section 3.1, it makes use of pheromone trails, denoted by (τ_{ij}) and heuristic information, denoted by (η_{ij}). In this way, ants increasingly build solutions to the optimization problem. Once an ant has built a solution, or while the solution is being built, the ant evaluates the (partial) solution that will be used by the *update pheromones* procedure to decide how much pheromone to deposit. *Update pheromones* this is the process by which the pheromone trails are modified. The trails value can either increase, as ants deposit pheromone on the components or connections they use, or decrease, due to pheromone evaporation. From a practical point of view, the deposit of new pheromone increases the probability for those components/connections, that were either used by many ants or that were used by at least one ant, and which produced a very good solution, to be used again by future ants. On the other hand, pheromone evaporation implements a useful form of *forgetting*: it avoids premature convergence of the algorithm toward a suboptimal region, therefore favoring the exploration of new areas of the search space.

The Ant Colony System (ACS), is an ACO algorithm introduced by Dorigo and Gambardella [8]. It uses a modified rule when an ant chooses the next travel node. It uses a best-so-far pheromone update rule but applies pheromone evaporation only to the trail that belongs to the solution components that are in best-so-far solution. It also uses a local pheromone update rule to decrease the pheromone values on visited solution components, in order-to encourage exploration. A general outline of the ACS is presented in Algorithm 1.

Algorithm 1 Pseudo-code for ACS

```
Initialize
for c=1 to Cycles-Number do
  for k=1 to Ants-Number do
    Construct-Ant-Solution (Local Update Pheromone)
    Save-Best-Solution
    Rank-Solution
    Global-Update-Pheromone
    Reallocation-Ants
  end for
end for
Print-Best-Solution
```

3.1 ACS for $(P_m||T_{max})$

This section presents the description of the main components of the implemented ACS.

1. *Construction graph:* The ants perform random *walks* in a *construction graph* and these walks represent feasible solutions of the underlying combinatorial optimization problem. To construct a feasible solution the artificial ants successively choose jobs to be appended to the actual subsequence, until all jobs are scheduled. Each ant decides independently of each other which job j should be the i-th job in the sequence, and each ant generates a complete solution. A walk consists of several "node-to-node" movements and these movements are performed on the basis of *transition probabilities*. The transition probability P_{ij} that job j be selected to be processed on position i in the sequence is formally given by:

$$P_{ij} = \begin{cases} \frac{\tau_{ij}\eta_{ij}^{\beta}}{\Sigma\tau_{ih}\eta_{ih}^{\beta}} & j\epsilon\Omega \\ 0 & otherwise \end{cases} \quad (1)$$

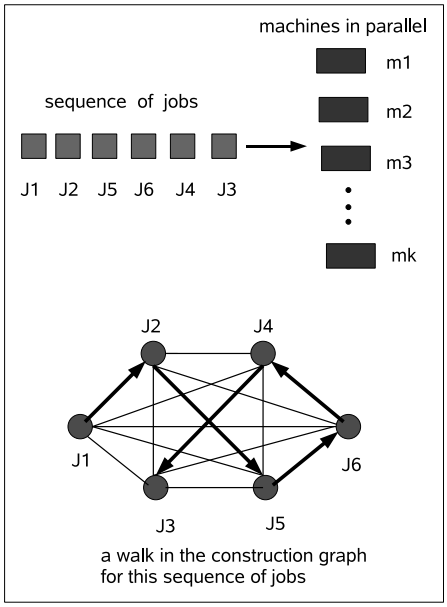


Figure 1: *Example of a construction graph for 6 jobs in m_k machines in parallel.*

where Ω is the set of non-scheduled jobs, and h belongs to Ω, η_{ij} is the specific-problem heuristic information, and τ_{ij} the pheromone trails.

For example, for an instance problem of 6 jobs and m_k machines, the construction graph can be seen as in Figure 1. The nodes in the graph represent jobs whereas the edges represent the possible walks that the ants can follow. The solution $(J_1, J_2, J_5, J_6, J_4, J_3)$ is a walk in the graph represented by boldface edges and it starts in J_1 node.

2. The formulas of the local and global pheromone update are:

   (a) Local Update Rule

   $$\tau_{ij} = (1 - \rho_{local})\tau_{ij} + \rho_{local}\tau_0 \tag{2}$$

   where ρ_{local} is a weight between $0 \leq \rho_{local} \leq 1$, and τ_0 is a constant value.

   (b) Global Update Rule

   $$\tau_{ij}(t+1) = (1 - \rho_{global})\tau_{ij}(t) + \rho_{global}\Delta\tau^b_{ij}(t+1) \tag{3}$$

   where $\forall(i,j)\epsilon T^b$ and ρ_{global} is a weight between $0 \leq \rho_{global} \leq 1$, Δ is $1/fitness$, b is the index of the best-so-far solution, and T^b is the path of b.

3. For selecting the next component j, ACS uses the next formula:

   $$j = \begin{cases} arg.max\{\tau_{ij}\eta^{\beta}_{ij}\} & q \leq q_0 \\ P_{ij} & otherwise \end{cases} \tag{4}$$

   where q is a random variable uniformly distributed in $[0, 1]$, q_0 is a parameter between $0 \leq q_0 \leq 1$, and P_{ij} is the probability item selection given in equation 1.

4. *The objective function:*

   $$Min : T_{max} = max_j(T_j)$$

   where $T_j = \{C_j$ - $d_j,\ 0\}$ is the maximum delay time, d_j is the due date, and C_j is the earliest completion time of the j job.

5. *Specific information heuristics* η

   (a) *Earliest Due Date (EDD)* based heuristic, where jobs are sorted and scheduled according to ascending due dates.

   $$\eta_{EDD} = 1/d_j$$

   (b) *Shortest Processing Time (SPT)* based heuristic, where jobs are sorted and scheduled according to ascending shortest processing time.

   $$\eta_{SPT} = 1/p_j$$

   (c) *Largest Processing Time (LPT)* based heuristic, where jobs are sorted and scheduled according to ascending largest processing time.

   $$\eta_{LPT} = p_j$$

   (d) *Least Slack (SLACK)* based heuristic, where jobs are sorted and scheduled according to ascending smallest difference between due date and processing time.

   $$\eta_{SLACK} = 1/(d_j - p_j)$$

To implement the ACS with different heuristics we use the Mallba project [1]. It is an integrated way to develop a skeleton library for combinatorial optimization that includes exact, heuristic, and hybrid methods. The skeletons are based on the separation of two concepts: the problem to be solved and the general resolution method to be used. The skeletons can be seen as generic templates that only need to be instanced with the characteristics of the problem in order to solve it. All the features related to the method of selected generic resolution and their interaction with the problem itself, are implemented by the skeleton, while the particular characteristics of the problem must be provided by the user.

4 Experimental Design

As it is not easy to find published benchmarks for the unrestricted parallel machines scheduling problems we worked on, we built our own test with proper data, based on selected data corresponding to weighed tardiness problems taken from the OR-Library [17]. In such library, 125 test instances are available for each problem size $n = 40$, $n = 50$ and $n = 100$, where n is the number of jobs.

For problems of size 40, 20 instances are selected and for problems of size 100, 20 instances are selected as well, each instance with the same identification number, although they are not the same problem. That is to say that we have a problem numbered 1 with 40 jobs and another one numbered 1 with 100 jobs, and so on. The numbers of the problems are not consecutive because each one was selected randomly from different groups. The tardiness factor is harder for those with the highest identification number.

The instances were randomly generated in the OR-Library as follows: For each job j ($j = 1, ..., n$), an integer processing time $p\{j\}$ was generated from the uniform distribution $[1, 100]$ and integer processing weight $w\{j\}$ was generated from the uniform distribution $[1, 10]$. Instance classes of varying hardness were generated by using different uniform distributions for generating the due dates. For a given relative range of due dates RDD ($RDD = 0.2, 0.4, 0.6, 0.8, 1.0$) and a given average tardiness factor TF ($TF = 0.2, 0.4, 0.6, 0.8, 1.0$), an integer due date $d(j)$ for each job j was randomly generated from the uniform distribution $[P(1 - TF - RDD/2), P(1 - TF + RDD/2)]$, where $P = SUM\{j = 1, ..., n\}\ p(j)$. Five instances were generated for each of the 25 pairs of values of RDD and TF, yielding 125 instances for each value of n.

These data were the input for dispatching rules and conventional heuristics, and we used PARSIFAL [18], a software package provided by Morton and Pentico to evaluate the instances problem by means of different heuristics, as for example *EDD* and *SPT* used in this work. In this way, we got the benchmarks (known optimal values) for problem instances for the T_{max} objective function.

To evaluate and compare the *ACS* algorithms (each with different heuristic information), and compare them with a simple Genetic Algorithm, the following relevant performance variables were chosen from previous works [10]:

Parameters	
Names	*Values*
number of runs	30
number of steps	1000
colony size	140
q_0	0,9
t_0	0,5

Table 1: *Parameter values.*

Parameters				
Names	*EDD*	*SLACK*	*SPT*	*LPT*
β	5	5	10	5
ρ_{local}	0,05	0,05	0,02	0,05
ρ_{global}	0,9	0,9	0,5	0,5

Table 2: *Parameter values chosen of experiments.*

- *Ebest* $= ((best\ value - opt\text{-}val)/opt\text{-}val) * 100$: it is the percentage error of the best found solution when compared with the known or estimated (upper bound) optimum value *opt-val*. It gives a measure on how far the best solution is from that *opt-val*. When this value is negative, it means that the *opt-val* has been improved.
- *Mean Ebest (MEbest)*: it is the mean value of *Ebest* throughout all runs.
- *Mean Best ($\mu Best$)*: it is the mean objective value obtained from the best found solutions throughout all runs.
- *Hit Ratio*: it is the percentage of runs where the ACS reaches or improves the known or estimated optimum value.

The initial phase of the experiments consisted in stablishing some best parameter values for the ACS, and some others were set from the related literature [15], and they are presented in Tables 1 and 2.

We used the same maximum number of evaluations in all the experiments. We took a maximum number of $140,000$ evaluations, and used *elitism*. We performed several experiments with T_{max}, for three systems of unrestricted parallel machines scheduling problems:

- 20 instances of 40 jobs and $m = 2$ machines.
- 20 instances of 40 jobs and $m = 5$ machines.
- 20 instances of 100 jobs and $m = 5$ machines.

5 Analysis of Results

In tables 3, 4, and 5, we show the results obtained by two heuristics, η_{EDD} and η_{SLACK} of ACS algorithm because they had the best performance for the three problems studied, and also we show the results obtained by simple genetic algorithms *GAs* reported in [10].

In tables 3 and 4, we can see the $\mu Best$ average value obtained by heuristic η_{EDD} were the best minimun value, but if we look at the instances one to one, minimum values (the bold values), some belong to *GA* and some to *ACS*. The highest average values of *Hit Ratio* were for the η_{EDD} and η_{SLACK} heuristics. However, the *Hit Ratio* is zero or near zero for the instance numbers 66, 91, 111, 116, and 121. The some observation is true for *GA* algorithm.

In table 5 the minimun $\mu Best$ average value was obtained by *GA*, but the best *Hit Ratio* averages were for η_{EDD} and η_{SLACK} heuristics.

Maximun Tardiness: n=40 y m=2							
Inst.	*Opt.*	*GA*		*ACS* (η_{EDD})		*ACS* (η_{SLACK})	
N	*val.*	$\mu Best$	*HRatio*	$\mu Best$	*HRatio*	$\mu Best$	*HRatio*
1	235	225.73	9	**216.0**	100	**216.0**	100
6	599	614.7	17	**594.0**	100	**594.0**	100
11	1060	1018.1	93	**998.0**	100	**998.0**	100
19	1628	1635.93	20	**1624.0**	100	**1624.0**	100
21	1660	**1639.67**	100	**1646.2**	90	1688.5	20
26	55	66	60	**27.0**	100	**27.0**	100
31	494	527.47	7	**474.0**	100	**474.0**	100
36	869	920.23	7	**852.0**	100	**852.0**	100
41	1280	1421.8	7	**1271.0**	100	1271.1	100
46	1240	**1224.0**	80	1230.1	80	1381.8	0.0
56	247	293.17	0.0	**229.0**	100	**229.0**	100
61	602	762.73	0.0	604.1	0.0	**604.0**	0.0
66	1090	1244.43	0.0	**1236.3**	0.0	1272.2	0.0
71	1280	**1289.27**	53	**1346.0**	10	1398.8	0.0
86	493	535.43	17	**457.0**	100	**457.0**	100
91	896	**1014.07**	3	**1164.4**	0.0	1192.1	0.0
96	1537	**1592.03**	13	1630.6	30	**1615.6**	0.0
111	659	**956.97**	0.0	1014.6	0.0	1206.7	0.0
116	650	**799.3**	0.0	900.3	20	1635.2	0.0
121	1430	**1486.07**	13	1676.6	0.0	1954.0	0.0
AVG	**900,2**	**963.36**	**29.0**	**959.56**	**61.5**	**1034.55**	**51.0**

Table 3: *The Opt-val are obtained by Parsifal, the $\mu Best$ and HitRatio are obtained by simple genetic algorithm GA and ACS algorithms using η_{EDD} and η_{SLACK} specific information heuristics.*

Such heuristics use the due date d_j values in their formulas, as does the objective function T_{max}. We could say that good performance of *ACS* algorithm which uses these heuristics, were due to the existing relationship between the specific information used by the search process and the objective function to be minimized.

However, for the three problems studied, the minimum best $\mu Best$ values (the bold values) for instances with higher identification numbers were the ones obtained by *GA* algorithm.

We also campare the results of *ACS* algorithms with different information heuristics, η_{EDD}, η_{SLACK}, η_{SPT} and η_{LPT}, by mean of statistic test. We use the *EBest* values of the 30 independent runs to do the analysis of variance between four ACS algorithms. We apply the Kruskal-Wallis [14] one-way analysis when the values (the sample) do not have a normal distribution (determined by Kolmogorov-Smirnov test), and we apply Anova test when the values have a normal distribution. The Kruskal-Wallis and Anova test, both return the *p-value* for the null hypotesis for all samples. If the *p-values* is zero or near zero, that suggests that at least one sample is significantly different (or statistically significant) than the other samples. Usually, if *p-values* are less than 0.05, we declare that the results are statistically significant

For the three problems the *p-values* were zero in all instances, this means that all comparisons are significantly different. However, if we use the Tukey method, we can do multiple comparison two by two, and this way we can determine between what experimental conditions there are significant differences.

Tables 6 and 7 show the results of Tukey method, which are confidence intervals that compose the upper bound, estimated value and lower bound for each pair of ACS algorithms, and this for each instance. According to Tukey method, the differences between means in which the confidence interval that encompasses the lower and upper limits do not contain the value 0 are statistically significant.

In tables bold confidence intervals belong to the algorithms that do not have difference significant. In table 6, the $ACS(\eta_{EDD})$ is not different to $ACS(\eta_{SLACK})$ for instances 1, 6, 26 and 41. Similarly the $ACS(\eta_{EDD})$ is not different to $ACS(\eta_{SPT})$ for instances 21. In table 7, the $ACS(\eta_{EDD})$ is not different to $ACS(\eta_{SLACK})$ for instances 61, 71, 111 and 41. Also $ACS(\eta_{SLACK})$ is not different to $ACS(\eta_{SPT})$ for instance 66, and to $ACS(\eta_{LPT})$ for instance 91. For instance 56 only are different $ACS(\eta_{EDD})$ and $ACS(\eta_{SLACK})$, and for instance 86 only are different $ACS(\eta_{EDD})$ and $ACS(\eta_{SLACK})$, similarly $ACS(\eta_{SPT})$ and $ACS(\eta_{LPT})$. The rest of the comparison of algorithms have difference significant.

Figure 2 show the results of the Tukey test to compare *ACS* (η_{EDD}) and *ACS* (η_{SLACK}) for some difficult instances of the third problem.

Maximun Tardiness: n=40 y m=5							
Inst.	*Opt.*	*GA*		*ACS* (η_{EDD})		*ACS* (η_{SLACK})	
N	*val.*	$\mu Best$	*HRatio*	$\mu Best$	*HRatio*	$\mu Best$	*HRatio*
1	284	283.67	50	242.8	100	**232.5**	100
6	652	647.2	67	621.5	100	**608.0**	100
11	1130	1059.83	100	1030.9	100	**1019.9**	100
19	1700	1666.4	97	1656.0	100	**1648.0**	100
21	1720	**1681.37**	90	1734.6	10	1730.5	30
26	100	127.77	13	79.5	100	**71.1**	100
31	644	613.2	77	**560.2**	100	**560.2**	100
36	984	1001.8	37	919.3	100	**908.1**	100
41	1340	1446.07	3	**1316.9**	100	1326.0	100
46	1310	**1270.43**	100	1346.6	0.0	1340.8	0.0
56	318	403.97	0.0	263.7	100	**255.1**	100
61	737	896.37	0.0	717.9	100	**684.7**	100
66	1240	**1363.9**	0.0	1385.7	0.0	1429.1	0.0
71	1330	**1352.97**	10	1416.5	10	1464.9	0.0
86	589	624.0	23	**541.6**	100	564.4	100
91	1040	**1112.8**	17	1239.3	0.0	1308.2	0.0
96	1690	**1699.27**	50	1731.8	20	1766.1	10
111	699	**1036.73**	0.0	1131.1	0.0	1239.8	0.0
116	672	**943.57**	0.0	1050.7	0.0	1588.0	0.0
121	1580	**1639.67**	20	1778.9	0.0	1893.0	0.0
AVG	**987.95**	**1043.55**	**38.0**	**1038.275**	**57.0**	**1081.92**	**57.0**

Table 4: *The Opt-val are obtained by Parsifal. the $\mu Best$ and HitRatio are obtained by simple genetic algorithm GA and ACS algorithms using η_{EDD} and η_{SLACK} specific information heuristics.*

6 Conclusions

We present four versions of ACS algorithm, each implemented with a different heuristic: 1) η_{EDD} based on Earliest Due Date rule, 2) η_{SPT} based on Shortest Processing Time, 3) η_{LPT} based on Largest Processing Time, and 4) η_{SLACK} based on Least Slack. The experiments were performed considering 20 instances for each proposed system, with the purpose of minimizing T_{max}.

The results show that ACS (η_{EDD}) and ACS (η_{SLACK}) were those who had a better behavior. For most simple instances (1 to 61) obtained competitive results both versions, being the ACS (η_{EDD}) outperformed by ACS (η_{SLACK}). However for more complex instances (from 61 onwards) ACS (η_{EDD}) was who had the best achievements, but none became achieve, in any run, the benchmark values.

The results also were comparable with those obtained by a simple GA, but we can not say that the same about the results reported by other advanced GAs [10].

Clearly, more work must be done because none of the ACS algorithms achieve the optimum known to all the problem discussed.

We are actually working on the construction of an ACS hybridized with local search mechanism in order to be able to improve the solutions forward by the ACS algorithm.

Acknowledgements

The authors gratefully acknowledge constant financial support from the University and $ANPCYT$.

References

[1] Enrique Alba, Gabriel Luque, Jose Garcia Nieto, Guillermo Ordónez, and Guillermo Leguizamón. MALLBA: A software library to design efficient optimisation algorithms. *International Journal of Innovative Computing and Applications*, 1(1):74–85, 2007.

[2] Jean Paul Arnaout, Rami Musa, and Ghaith Rabadi. Ant colony optimazation algorithm to parallel machines scheduling problem with setups. *IEEE Conference of on Automation Science and Engineering*, pages 578–582, 2008.

[3] Markus Bank and Udo Honing. An aco-based approach for scheduling task graphs with communication costs. *International Conference on Parallel Processing*, pages 623–629, 2005.

Maximun Tardiness: n=100 y m=5							
Inst.	*Opt.*	*GA*		*ACS* (η_{EDD})		*ACS* (η_{SLACK})	
N	*val.*	$\mu Best$	*HRatio*	$\mu Best$	*HRatio*	$\mu Best$	*HRatio*
1	590	742.67	3	555.3	100	**547.53**	100
6	1680	1676.1	57	1588.17	100	**1583.40**	100
11	2620	2686.03	7	2569.6	100	**2556.13**	100
19	3720	3889.10	0.0	3720.3	40	**3704.87**	100
21	5240	5330.9	0.0	**5311.9**	0.0	5495.93	0.0
26	168	499.87	0.0	129.2	100	**110.30**	100
31	1180	1515.30	0.0	1167.67	100	**1139.57**	100
36	2120	2462.23	0.0	2092.33	100	**2072.13**	100
41	3710	4087.43	0.0	3646.27	100	**3634.37**	100
46	4580	**4694.37**	7	4762.17	0.0	5188.67	0.0
56	670	1465.93	0.0	658.33	80	**613.0**	100
61	1630	2620.43	0.0	1646.27	13.33	**1631.03**	46.67
66	2440	3276.7	0.0	**3102.97**	0.0	3474.13	0.0
71	3820	**4436.1**	0.0	4499.4	0.0	4612.17	0.0
86	1240	2695.53	0.0	**1271.43**	0.0	1457.57	0.0
91	2230	**3096.47**	0.0	3285.33	0.0	3643.63	0.0
96	3250	**3933.7**	0.0	4040.2	0.0	4561.57	0.0
111	1420	3214.43	0.0	**3014.23**	0.0	3075.23	0.0
116	2320	**3390.9**	0.0	3632.87	0.0	4441.80	0.0
121	3060	4192.17	0.0	**4077.67**	0.0	4867.33	0.0
AVG	**2384.4**	**2595.28**	**3.0**	**2738.56**	**41.67**	**2920.52**	**47.33**

Table 5: *The Opt-val are obtained by Parsifal, the $\mu Best$ and $HitRatio$ are obtained by simple genetic algorithm GA and ACS algorithms using η_{EDD} and η_{SLACK} specific information heuristics.*

[4] A. Bauer, B. Bernd, R. Hartl, and C. Strauss. An ant colony optimization approach for the single machine total tardiness problem. *IEEE*, pages 1445–1450, 1999.

[5] A. Bauer, B. Bernd, R. Hartl, and C. Strauss. Minimizing total tardiness on a single machine using ant colony optimization. *Central European Journal of Operations Research*, 8(2):125–141, 2000.

[6] Zhi-Long Chen and Warren B. Powell. A column generation based decomposition algorithm for a parallel machines just-in-time scheduling problem. *Elseiver European of Operational Research*, 1999.

[7] Runwei Cheng and Mitsuo Gen. Parallel machines scheduling problem using memetic algorithms. *Elseiver Science*, 33(3-4):761–764, 1997.

[8] M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transaction Evolutionary Comput.*, 1(1):53–66, 1997.

[9] M. Dorigo and T. Stulzle. *Ant Colony Optimization.* Massachusetts Institute of Tecnology, 2004.

[10] E. Ferretti and S. Esquivel. A comparison of simple and multirecombinated evolutionary algorithms with and without problem specific knowledge insertion, for parrallel machines scheduling. *International Transaction on Computer Science and Engineering*, 3(1):207–221, 2005.

[11] E. Ferretti and S. Esquivel. An efficient approach of simple and multirecombinated genetic algorithms for parallel machine scheduling. In *IEEE Congress on Evolutionary Computation*, volume 2, pages 1340–1347, Scotland, UK, September 2005. IEEE Center.

[12] E. Ferretti and S. Esquivel. Knowledge insertion: An efficient approach to simple genetic algorithms for unrestricted for parallel equal machines scheduling. In *GECCO'05*, pages 1587–1588, Washington DC, USA, 2005.

[13] J. Heinonen and F. Pettersson. Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. *Applied Mathematics and Computation*, 2006.

[14] M. Hollander and D.A. Wolfe. *Nonparametric Statiscal Methods.* Wiley, 1975.

[15] Botee M. Hozefa and Eric Bonabeau. Evolving ant colony optimization. *Complex Systems*, pages 149–159, 1998.

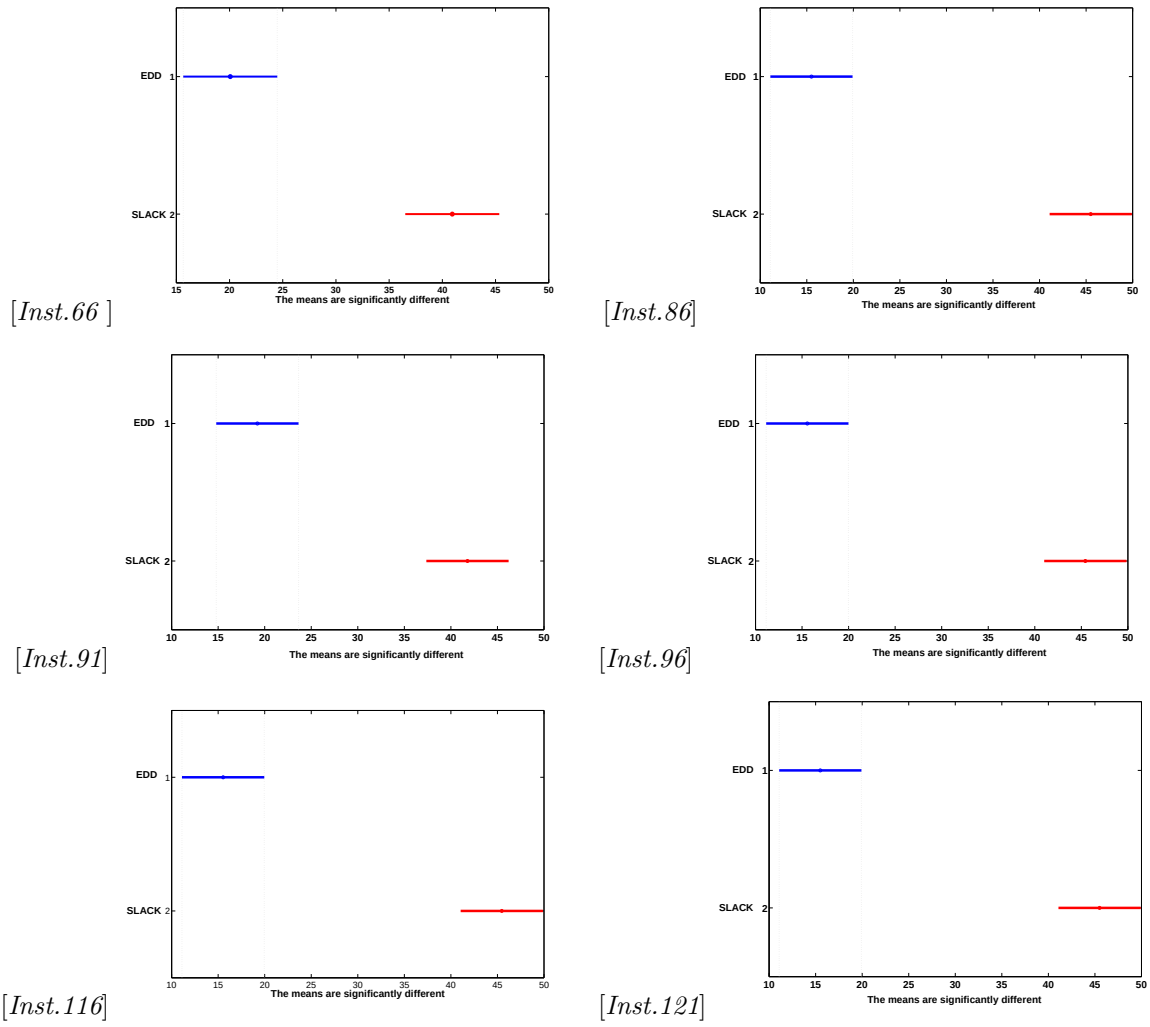


Figure 2: The results of Tukey test.

[16] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximations algorithms for scheduling unrelated parallel machines. *Springer Berlin Heidelberg*, 46(1-3):259–271, 1990.

[17] OR library Beasley J. `http://people.brunel.ac.uk/mastjjb/info.html`.

[18] T. Morton and D. Pentico. *Heuristic Scheduling Systems*. John Wiley and Sons, New York, 1993.

[19] M. Pinedo. *Scheduling: Theory, Algorithms and System*. Prentice Hall, 1995.

[20] Zhu Qiong, Gu Yichao, Zhang Gong Zhang Jie, and Chen Xuefang. An ant colony optimization model for parallel machine scheduling with human resource constraints. *Springer-Verlag Heidelberg 2010*, pages 917–926, 2009.

[21] Chandrasekharan Rajendran and Hans Ziegler. Ant colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, pages 426–438, 2004.

[22] Sankar Saravana, S.G. Ponnambalam, V. Rathinavel, and M.S. Visveshvaren. Scheduling in parallel machine schop: An ant colony optimization approach. *Industrial Technology, 2005, ICIT 2005 IEEE International Conference*, pages 276–280, 2005.

Multiple comparison of means by Tukey method for the Ebest of ACS algorithms					
Inst.	*ACS 1*	*ACS 2*	*Lower Bound*	*Estimated Value*	*Upper Bound*
1	EDD	SLACK	**-16,8646**	**6,1333**	**29,1313**
	EDD	SPT	-64,9313	-41,9333	-18,9354
	EDD	LPT	-94,9313	-71,9333	-48,9354
	SLACK	SPT	-71,0646	-48,0667	-25,0687
	SLACK	LPT	-101,0646	-78,0667	-55,0687
	SPT	LPT	-52,9979	-30	-7,0021
6	EDD	SLACK	**-0,6128**	**0,1329**	**0,8787**
	EDD	SPT	-12,6962	-11,9504	-11,2046
	EDD	LPT	-41,2596	-40,5139	-39,7681
	SLACK	SPT	-12,8291	-12,0833	-11,3376
	SLACK	LPT	-41,3926	-40,6468	-39,9011
	SPT	LPT	-29,3092	-28,5635	-27,8177
11	EDD	SLACK	0,0465	0,4071	0,7677
	EDD	SPT	-8,3148	-7,9542	-7,5936
	EDD	LPT	-26,4153	-26,0547	-25,6941
	SLACK	SPT	-8,7219	-8,3613	-8,0007
	SLACK	LPT	-26,8224	-26,4618	-26,1012
	SPT	LPT	-18,4611	-18,1005	-17,7399
19	EDD	SLACK	0,0497	0,3719	0,694
	EDD	SPT	-7,5068	-7,1846	-6,8624
	EDD	LPT	-24,9557	-24,6335	-24,3113
	SLACK	SPT	-7,8786	-7,5565	-7,2343
	SLACK	LPT	-25,3276	-25,0054	-24,6832
	SPT	LPT	-17,7711	-17,4489	-17,1267
21	EDD	SLACK	-78,1287	-55,0833	-32,0379
	EDD	SPT	**-43,5121**	**-20,4667**	**2,5787**
	EDD	LPT	-108,2287	-85,1833	-62,1379
	SLACK	SPT	11,5713	34,6167	57,6621
	SLACK	LPT	-53,1454	-30,1	-7,0546
	SPT	LPT	-87,7621	-64,7167	-41,6713
26	EDD	SLACK	**-0,0115**	**0,0032**	**0,0179**
	EDD	SPT	-0,4143	-0,3996	-0,3849
	EDD	LPT	-1,0263	-1,0116	-0,9969
	SLACK	SPT	-0,4175	-0,4028	-0,3881
	SLACK	LPT	-1,0295	-1,0148	-1,0001
	SPT	LPT	-0,6267	-0,612	-0,5973
31	EDD	SLACK	6,811	29,8667	52,9223
	EDD	SPT	-53,1223	-30,0667	-7,011
	EDD	LPT	-83,1223	-60,0667	-37,011
	SLACK	SPT	-82,989	-59,9333	-36,8777
	SLACK	LPT	-112,989	-89,9333	-66,8777
	SPT	LPT	-53,0557	-30	-6,9443
36	EDD	SLACK	0,1281	0,8648	1,6015
	EDD	SPT	-24,6313	-23,8946	-23,158
	EDD	LPT	-86,7839	-86,0472	-85,3105
	SLACK	SPT	-25,4961	-24,7594	-24,0227
	SLACK	LPT	-87,6486	-86,912	-86,1753
	SPT	LPT	-62,8892	-62,1525	-61,4158
41	EDD	SLACK	**-0,3245**	**0,2794**	**0,8833**
	EDD	SPT	-19,9139	-19,31	-18,7061
	EDD	LPT	-50,0154	-49,4115	-48,8076
	SLACK	SPT	-20,1933	-19,5894	-18,9855
	SLACK	LPT	-50,2948	-49,6909	-49,087
	SPT	LPT	-30,7054	-30,1015	-29,4976
46	EDD	SLACK	-9,8424	-9,1201	-8,3978
	EDD	SPT	-4,7405	-4,0182	-3,2959
	EDD	LPT	-14,5738	-13,8515	-13,1292
	SLACK	SPT	4,3796	5,1019	5,8242
	SLACK	LPT	-5,4538	-4,7314	-4,0091
	SPT	LPT	-10,5556	-9,8333	-9,111
56	EDD	SLACK	0,6577	3,6617	6,6657
	EDD	SPT	**-193,5959**	**-190,5919**	**187,5879**
	EDD	LPT	**-350,3672**	**-347,3633**	**344,3593**
	SLACK	SPT	**-197,2576**	**-194,2536**	**191,2496**
	SLACK	LPT	**-354,0289**	**-351,025**	**348,021**
	SPT	LPT	**-159,7753**	**-156,7714**	**153,7674**

Table 6: *Multiple comparison means by Tukey method of the simple instances of 100 jobs and 5 machines.*

Multiple comparison of means by Tukey method for the Ebest of ACS algorithms					
Inst.	*ACS 1*	*ACS 2*	*Lower Bound*	*Estimated Value*	*Upper Bound*
61	EDD	SLACK	**-38,5507**	**-15,5**	**7,5507**
	EDD	SPT	-75,8007	-52,75	-29,6993
	EDD	LPT	-105,8007	-82,75	-59,6993
	SLACK	SPT	-60,3007	-37,25	-14,1993
	SLACK	LPT	-90,3007	-67,25	-44,1993
	SPT	LPT	-53,0507	-30	-6,9493
66	EDD	SLACK	-17,5555	-12,7281	-7,9007
	EDD	SPT	-19,7878	-14,9604	-10,133
	EDD	LPT	-54,5842	-49,7568	-44,9294
	SLACK	SPT	**-7,0596**	**-2,2323**	**2,5951**
	SLACK	LPT	-41,8561	-37,0287	-32,2013
	SPT	LPT	-39,6238	-34,7964	-29,9691
71	EDD	SLACK	**-27,1267**	**-4,0667**	**18,9933**
	EDD	SPT	-67,8933	-44,8333	-21,7733
	EDD	LPT	-99,36	-76,3	-53,24
	SLACK	SPT	-63,8267	-40,7667	-17,7067
	SLACK	LPT	-95,2933	-72,2333	-49,1733
	SPT	LPT	-54,5267	-31,4667	-8,4067
86	EDD	SLACK	-19,4349	-14,6398	-9,8447
	EDD	SPT	**-172,1392**	**-167,3441**	**162,549**
	EDD	LPT	**-276,2629**	**-271,4678**	**266,6727**
	SLACK	SPT	**-157,4994**	**-152,7043**	**147,9092**
	SLACK	LPT	**-261,6231**	**-256,828**	**252,0329**
	SPT	LPT	-108,9188	-104,1237	-99,3286
91	EDD	SLACK	-60,9698	-37,9167	-14,8635
	EDD	SPT	-53,0865	-30,0333	-6,9802
	EDD	LPT	105,7032	-82,65	-59,5968
	SLACK	SPT	**-15,1698**	**7,8833**	**30,9365**
	SLACK	LPT	-67,7865	-44,7333	-21,6802
	SPT	LPT	-75,6698	-52,6167	-29,5635
96	EDD	SLACK	-79,9042	-56,9	-33,8958
	EDD	SPT	-50,4542	-27,45	-4,4458
	EDD	LPT	-111,1209	-88,1167	-65,1125
	SLACK	SPT	6,4458	29,45	52,4542
	SLACK	LPT	-54,2209	-31,2167	-8,2125
	SPT	LPT	-83,6709	-60,6667	-37,6625
111	EDD	SLACK	**-18,5013**	**4,5667**	**27,6346**
	EDD	SPT	-65,7846	-42,7167	-19,6487
	EDD	LPT	-95,7846	-72,7167	-49,6487
	SLACK	SPT	-70,3513	-47,2833	-24,2154
	SLACK	LPT	-100,3513	-77,2833	-54,2154
	SPT	LPT	-53,068	-30	-6,932
116	EDD	SLACK	-37,9127	-33,4224	-28,9321
	EDD	SPT	-14,7489	-10,2586	-5,7683
	EDD	LPT	-60,8769	-56,3866	-51,8963
	SLACK	SPT	18,6735	23,1638	27,6541
	SLACK	LPT	-27,4545	-22,9642	-18,4739
	SPT	LPT	-50,6183	-46,128	-41,6377
121	EDD	SLACK	-82,1816	-59,15	-36,1184
	EDD	SPT	-51,3316	-28,3	-5,2684
	EDD	LPT	-112,1816	-89,15	-66,1184
	SLACK	SPT	7,8184	30,85	53,8816
	SLACK	LPT	-53,0316	-30	-6,9684
	SPT	LPT	-83,8816	-60,85	-37,8184

Table 7: *Multiple comparison means by Tukey method of hard instances of 100 jobs and 5 machines.*