A Variant of Simulated Annealing to Solve Unrestricted Identical Parallel Machine Scheduling Problems

Claudia Gatica[1], Susana Esquivel[1] and Guillermo Leguizamón[1],

[1] LIDIC
Universidad Nacional de San Luis
Ejército de Los Andes 950 - Local 106
San Luis, Argentina
Telephone: (0266) 4420823
Fax: (0266) 4430224

{crgatica,esquivel,legui}@ unsl.edu.ar

Abstract. In this paper we propose a modification to the Simulated Annealing (SA) basic algorithm that includes an additional local search cycle after finishing every Metropolis cycle. The added search finishes when it improves the current solution or after a predefined number of tries. We applied the algorithm to minimize the Maximum Tardiness objective for the Unrestricted Parallel Identical Machines Scheduling Problem for which no benchmark have been found in the literature. In previous studies we found, by using Genetic Algorithms, solutions for some adapted instances corresponding to Weighted Tardiness problem taken from the OR-Library. The aim of this work is to find improved solutions (if possible) to be considered as the new benchmark values and make them available to the community interested in scheduling problems. Evidence of the improvement obtained with proposed approach is also provided.

Keywords: Unrestricted Parallel Identical Machines Scheduling Problem, Simulating Annealing. Maximum Tardiness.

1 Introduction

The schedule of activities is a decision process that has an important role in production and multiprocessor systems, manufacturing and information environments, and transportation distribution[17]. In particular, this paper considers the unrestricted identical parallel machine scheduling problem in which the maximum tardiness has to be minimized. Objectives such as the completion time of the last job to leave the system, known as Makespan (C_{max}), is one the more important objective function to be optimized, because it usually implies high resource utilization. In different systems of real world it is also usual stress minimization of the due-date based objectives as Maximum Tardiness (T_{max}) among others. Branch and Bound and other partial enumeration based methods, which guarantee exact solutions, are prohibitively time consuming even with only 20 jobs. The parallel machine environment has been

studied for several years due to its importance both academic and industrial. The scheduling literature provides a set of dispatching rules and heuristics. Different metaheuristics have been used to solve scheduling problems. For example, the population-based metaheuristics such as Evolutionary Algorithms and Ant Colony Optimization [2], [4], [11]. The trajectory-based heuristics have also been applied to solve these types of problems. In [13] VNS was used to solve the makespan in uniform parallel machine scheduling problem with release dates. In other related work [1] the authors applied an Iterated Local Search metaheuristic to solve the unrestricted parallel machine with unequal ready time problem. In [18] VNS with an efficient search mechanism, is proposed to solve the problem of maximum C_{max} in unrelated parallel machine scheduling. A comparative study [19] was conducted between SA and GRASP to solve the problem of maximum C_{max} in single machine scheduling, there SA outperforms GRASP. A hybrid approach is addressed in [5] which integrates features of Tabu Search (TS), SA, and VNS to solve a parallel machine problem with total tardiness objective. Another hybrid approach is presented in [14] where the authors combine TS with VNS in a way that the TS algorithm is embedded into VNS acting as a local search operator for parallel machine scheduling problem. In [6], [7], and [8] the authors face the same problem with an approach involving Evolutionary Algorithms with multirecombination and insertion of specific knowledge of the problem.

The rest of this paper is organized as follows. The next section presents the scheduling problem. After this, in section 3, the proposed algorithm is described. Section 4 explains the experimental design. In section 5 the results are shown and discussed. Finally, in section 6 we present our conclusions and outline our future work.

2 Scheduling Problem

The problem we are facing 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; 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 readily be computed. The problem is to find a processing order of the jobs with minimum objective values. The objective to be minimized is:

$$\text{Maximum Tardiness: } T_{max} = max_j (T_j) \quad (1)$$

This problem is NP-hard when $2 \leq m \leq n$ [17].

3 The Proposed SA Algorithm

In a previous study [3] we work on the same problem but we address it with different local search metaheuristics: SA, VNS, Iterated Local Search (ILS) and Greedy Random Adaptive Search Procedure (GRASP). This comparative study showed that the best algorithm was SA although it was only able to improve benchmark values in ten instances (see Table 1). For reasons of space only the results obtained for m = 5 and n= 100 are showed. From the results obtained we assumed that the algorithm lacked of higher exploration capacity. With the main idea of overcoming these difficulties, we design a variant of the SA algorithm.

I	Bench	ILS	GRASP	VNS	SA
1	548	587	597	**547**	**542**
6	1594	1594	**1581**	**1572**	**1567**
11	2551	2577	2626	2552	**2539**
19	3703	3756	3784	3717	3718
21	5187	5193	5232	5197	**5177**
26	84	148	407	101	**70**
31	1134	1160	1366	1145	1135
36	2069	2128	2360	2091	**2061**
41	3651	**3631**	3821	**3621**	**3607**
46	4439	4475	4599	4443	4440
56	617	725	1104	655	**609**
61	1582	1779	2453	1705	**1580**
66	2360	2483	2870	2427	**2359**
71	3786	3924	4413	3862	3791
86	1194	1455	2281	1393	1194
91	2204	2427	2953	2412	2222
96	3185	3256	3780	3216	3187
111	1365	1846	3216	1781	1458
116	2222	2537	3055	2457	2266
121	2999	3407	3890	3286	3099

Table 1: Best values achieved by each metaheuristic

The pseudo-code of the proposed SA algorithm is given in Algorithm 1. The search processes of our algorithm is divided into two stages, based on the equilibrium condition as follows: SA starts with a high initial temperature (IT = 14256), it generates a random initial solution, and it initializes the counter to the equilibrium condition, which is achieved with the length of the Markov chain (LMC = 9716), which represents a constant number of search steps that are performed without updating the temperature (T). The justification for the initial value of temperature (IT), the length of the Markov chain (LMC) as the selection of operators (op1 and op2) is given in subsection 4.2 Then, depending on the condition of equilibrium, the search process is divided into two stages. In the first stage, the solutions are generated through the perturbation operator (op1 = scramble) (step: 7) and in the second stage, once the equilibrium condition is reached, and before updating the temperature (step: 16) it applies an extra exploration procedure called Explore (step: 15) which is described in Algorithm 2. Algorithms 1 and 2 show schematically the search process performed SA.

Algorithm 1 SA Algorithm including a call to an exploration procedure.

```
1: c = 0  {Used for the equilibrium condition}
2: s = s0 {Initial solution}
3: T = T0 {Starting temperature}
4: repeat
5:    repeat
6:          c  = c + 1
7:          Generate a solution s0 applying a
            perturbation operator (op1)
8:          ΔE = f(s0) – f(s)
9:          if ΔE ≤ 0 then
10:                 s = s0
11:         else
12:                 Accept s0 with a probability e^(-ΔE/T)
13:         end if
14:   until c == Markov-chain-length
15:   s0 = Explore(s)
16:   Update (T) {Geometric temperature update}
17:   c = 0
18:   ΔE = f(s0) – f(s)
19:   if ΔE ≤ 0 then
20:        s = s0
21:   else
22:        Accept s0 with a probability e^(-ΔE/T)
23:   end if
24: until Stopping Criteria
25: return s
```

Algorithm 2 Explore(s): the exploration procedure.

```
1: Input: s solution from SA, tries is the number of attempts
2: i = 1
3: while i ≤ tries do
4:  Generate a solution s0 applying a perturbation
    operator (op2)
5:  if f(s0) < f(s) then
6:   s = s0
7:   return s
8:  else
9:   i = i + 1
10: end if
11: end while
12: return s
```

The function Explore performs (i = 1, ... , *tries*) attempts to find a solution s_0 that improves s, as follows: generates a solution s_0 by applying a perturbation operator (*op2* = 4-*opt*). If f(s_0) < f(s), s is replaced by s_0 and Explore returns s, otherwise, another attempt is made by (steps : 4−6).

Following Algorithm 1, the acceptance criteria is applied (steps: 8 − 13 and 18 - 23). The search process ends when it reaches a maximum number of evaluations (step: 25).

In our implementation, the representation of the solutions is a permutation of integers in the range 1... *n,* which represent the job indexes.

The initial solution is a integer permutation randomly generated as follows: from 1 to n, for each index i generates a random number between i and n. This process checks that the solution is a valid representation, i.e. it is a permutation without repetition.

4 Experimental Design

4.1 Instances for the Unrestricted Parallel Identical Machines Scheduling Problem

Unlike other scheduling problems as Flow Shop or Job Shop, after an intensive search in the literature we could not find significant benchmarks for the problem we worked on. With the purpose of creating our own benchmarks, we extracted value pairs (p_j, d_j) based on selected data corresponding to Weighted Tardiness problem taken from the OR Library [10]. The values p_j and d_j correspond to the processing time and due date, respectively. These data were taken from problem sizes of 40 and 100 jobs. For each problem size, twenty instances were selected, each one with the same identification number although they were not the same problem, i.e., we had a problem numbered 1 with 40 jobs, another 1 with 100 jobs, and so on.

#I	*m*=2, *n*=40		*m*=5, *n*=40	
	DR	**MCMP-SE**	**DR**	**MCMP-SE**
1	235 (EDD)	**216**	284 (SLACK)	**230**
6	599 (SLACK)	**595**	652 (SLACK)	**606**
11	1060 (EDD)	**998**	1130 (SLACK)	**1016**
19	1628 (EDD)	**1624**	1700 (SLACK)	**1639**
21	1660 (SLACK)	**1634**	1720 (SLACK)	**1647**
26	55 (EDD)	**35**	100 (SLACK)	**61**
31	494 (EDD)	**474**	644 (SLACK)	**546**
36	869 (SLACK)	**852**	984 (SLACK)	**887**
41	1280 (EDD)	**1271**	1340 (EDD)	**1317**
46	1240 (EDD)	**1195**	1310 (SLACK)	**1235**
56	247 (SLACK)	**229**	318 (SLACK)	**252**
61	604 (EDD)	**604**	737 (SLACK)	**669**
66	1090 (SLACK)	**1071**	1240 (SLACK)	**1129**
71	1280 (EDD)	**1254**	1330 (SLACK)	**1272**
86	493 (SLACK)	**457**	589 (SLACK)	**508**
91	896 (EDD)	**874**	1040 (EDD)	**955**
96	1537 (EDD)	**1531**	1690 (SLACK)	**1607**
111	659 (EDD)	**621**	794 (SLACK)	**689**
116	650 (SLACK)	**627**	810 (SLACK)	**695**
121	1430 (EDD)	**1377**	1580 (SLACK)	**1469**

Table 2: Obtained values for 2 - 5 machines and 40 jobs

The numbers of the instances are not consecutive because each one was selected randomly from different groups. The tardiness factor is harder for those with the highest identification number.

These instances are available on request (email: crgatica@unsl.edu.ar). In a previous work [7], those value pairs were used as input for different dispatching rules (SPT: Shorted Processing Time first, EDD: Earliest Due Date first, SLACK: Least Slack, HODG Algorithm, and R&M: Rachamadugu and Morton Heuristic) provided by PARSIFAL [17], a Software Package provided by Morton and Pentico, and a Multi Crossover Multi Parent Genetic Algorithm (MCMP-SE) with insertion of knowledge [8]. The results obtained are showed in Table 1 (cases *m*=2, *n*=40 and *m*=5, *n*=40) and Table 2 (cases *m*=2, *n*=100 and *m*=5, *n*=100).

#I	*m*=2, *n*=100		*m*=5, *n*=100	
	DR	**MCMP-SE**	**DR**	**MCMP-SE**
1	562 (EDD)	**536**	590 (SLACK)	**548**
6	1550 (EDD)	**1544**	1680 (SLACK)	**1594**
11	2560 (EDD)	**2516**	2620 (SLACK)	**2551**
19	3690 (SLACK)	**3679**	3720 (SLACK)	**3703**
21	5150 (EDD)	**5143**	5240 (SLACK)	**5187**
26	60 (R&M)	**21**	168 (SLACK)	**84**
31	1110 (SLACK)	**1092**	1180 (SLACK)	**1134**
36	**2040** (SLACK)	2041	2120 (SLACK)	2069
41	3590 (EDD)	**3576**	3710 (SLACK)	**3651**
46	4420 (EDD)	**4396**	4580 (SLACK)	**4439**
56	582 (HODG)	**556**	670 (SLACK)	**617**
61	1560 (EDD)	**1549**	1630 (SLACK)	**1582**
66	2360 (EDD)	**2313**	2440 (SLACK)	**2360**
71	3780 (EDD)	**3741**	3820 (SLACK)	**3786**
86	1200 (EDD)	**1153**	1240 (SLACK)	**1194**
91	2180 (SLACK)	**2132**	2230 (EDD)	**2204**
96	3110 (SLACK)	**3093**	3250 (SLACK)	**3185**
111	5340 (WLPT)	**1325**	1420 (SLACK)	**1365**
116	2200 (EDD)	**2164**	2320 (SLACK)	**2222**
121	2940 (EDD)	**2934**	3060 (SLACK)	**2999**

Tabla 3: Obtained values for 2 - 5 machines and 100 jobs

In both Tables, #I indicates the instance identification and DR stands for Dispatching Rules. In the case of the dispatching rules, the displayed values correspond to the best obtained by the different rules used, whose names are enclosed in brackets. Bold values from both tables are considered as benchmarks in the present work.

4.2 Parameter Settings

In this subsection we describe the method used to determine the set of appropriate parameter values for our metaheuristic. There are different ways to do this, but can distinguish two main groups of techniques: one, when the sample used is formed with extreme values of the design space (*no space-filling*) or otherwise, when data values correspond to the interior of the design space (*space-filling*) [21]. The latter approach is the one we choose because it assumes that the interior of the design space can meet important characteristics of the true design model. For the generation of the samples we use the method Latin Hypercube Design (LHD) which generates random points within the design space. For the SA algorithm and Explore function their relevant parameters and corresponding application ranges were determined. They are indicated in Table 4. We use five different operators of disturbance or movement: n swaps (1), 2-opt (2), 4-opt (3), shift (4) and scramble (5). A detailed description of these operators can be found in [22]. Then LHD was employed using 20 design points which resulted in 20 different parameter configurations, this task was performed using the statistical tool R [20]. The resulting points sampling are shown in Table 5.

LMC=Length Markov chain	[1000, 10000]
CR=Cooling Rate	[0.5, 1.0]
IT=Initial Temperature	[10000, 100000]
OP1=Pertubation Operator of SA	[1, 5]
OP2=Pertubation Operator of Explore	[1, 5]

NT=Number of Tries	[10, 20]

Table 4: Parameter Ranges

Ultimately, we perform 20 experiments. Each experiment consisted of 50 runs of the algorithm SA, each run with 300,000 evaluations of the objective function for each of the 20 instances of 100 jobs and 5 machines. For the statistical study we use a software tool proposed by [12]. Such is called CONTROLTEST and automatically applies various statistical tests, one of which is the Friedman test [15] and other post-hoc procedures [16]. Resulting from the application on the median values of the runs of different configurations allowed us obtain the Average Ranking of Friedman Test and so, we were able to establish that the best performers were the c_4 and c_8 configurations (See Table 5, in column RF, such corresponds to Average Ranking of Friedman test) and also we can conclude that there are not statistical significant differences between them because the corresponding adjusted *p*-values did not give values less than 0.05, see Table 6. The only difference of the behaviour of SA with the specified parameter setup for *c4* and *c8* (and the reason of selection of *c8* configuration) was the lowest number of evaluations used by SA to achieve the best values. For reasons of space, the tables showing these results are not given here.

Conf.	LMC	CR	IT	OP_1	OP_2	NT	RF
c1	1287	0,79391	86906	4	3	17	14,025
c2	6455	0,50691	57118	2	1	11	5,425
c3	2809	0,58518	93290	2	4	15	3,275
c4	**8258**	**0,66540**	**84705**	**5**	**2**	**16**	**1,775**
c5	3358	0,54591	30000	2	5	18	3,575
c6	4554	0,81334	59801	3	2	14	8,15
c7	4681	0,56859	69300	4	4	16	13,525
c8	**9716**	**0,61812**	**14256**	**5**	**3**	**11**	**1,575**
c9	8745	0,95200	54194	2	4	12	19,825
C10	5806	0,97936	42010	4	2	10	17,775
C11	3721	0,70923	20184	3	3	20	8,025
C12	7727	0,87080	67559	1	3	14	16,175
C13	1894	0,89262	14825	3	2	17	7,95
C14	9199	0,68212	73597	4	3	12	12,95
C15	6071	0,75536	36234	2	4	15	5,525
C16	7246	0,93239	37356	3	2	18	18,625
C17	7903	0,83626	80125	2	5	13	17,1
C18	5348	0,64777	24275	4	4	19	12,65
C19	2336	0,74463	99708	1	1	13	11,875
C20	2615	0,92429	46197	3	2	19	10,2

Table 5: Parameter Configurations

config.	p-Bonf	p-Holm	p-Hoch	p-Homm
c4	1,05E+00	5,27E-01	5,27E-01	5,27E-01

Table 6: Adjusted *p*-values

4.3 Final Optimization Experiments

For each scenario, 50 runs were executed, each one with 600,000 objective function evaluations. In each experiment we calculate the following metrics:

1) **Best:** The best value found in each run.

2) **Median:** Is the median objective value obtained from the best found individuals throughout all runs.

3) **SD of Median:** The standard deviation of median objective value is the square root of its variance.

4) **Miter:** Is the mean of iterations where the best value was obtained.

5) **SD of Miter:** The standard deviation of mean of iterations in each run is the square root of its variance.

All the experiments reported in this work were run on a sub-cluster conformed by 1 CPUs of 64 bits, processor Intel Q9550 Quad Core 2.83GHz, with 4GB DDR3 1333Mz of memory, 500 Gb SATA and 2 TB SATA hard disks, Asus P5Q3 motherboard and 11 CPUs of 64 bits each with processor Intel Q9550 Quad Core 2.83GHz, 4GB DDR3 1333Mz memory, 160 Gb SATA hard disk and Asus P5Q3 motherboard.

5 Results and Discussion

For all cases studied, Table 7 synthesizes the best values of the objective function found by SA. In Table 7 entries marked in bold indicate that SA improved the benchmark value while entries in italic show that SA reached benchmark. For the case of 40 jobs and 2 machines, in almost all instances the benchmark values were achieved, except in instances 6, 26, and 116 where the algorithm was able to find smaller values.

	n=40				*n*=100			
	m=2		*m*=5		*m*=2		*m*=5	
#I	Bench	Best	Bench	Best	Bench	Best	Bench	Best
1	**216**	*216*	**230**	**229**	**536**	*536*	**548**	**539**
6	**595**	**594**	**606**	**604**	**1544**	*1544*	**1594**	**1569**
11	**998**	*998*	**1016**	*1016*	**2516**	*2516*	**2551**	**2544**
19	**1624**	*1624*	**1639**	*1639*	**3679**	*3679*	**3703**	3708
21	**1634**	*1634*	**1647**	*1647*	**5143**	*5143*	**5187**	**5177**
26	**35**	**27**	**61**	**55**	**21**	*21*	**84**	**70**
31	**474**	*474*	**546**	**542**	**1092**	*1092*	**1134**	**1125**
36	**852**	*852*	**887**	**885**	**2041**	**2037**	**2069**	**2061**
41	**1271**	*1271*	**1317**	**1313**	**3576**	*3576*	**3651**	**3607**
46	**1195**	*1195*	**1235**	**1227**	**4396**	*4396*	**4439**	*4439*
56	**229**	*229*	**252**	**244**	**556**	*556*	**617**	**606**
61	**604**	*604*	**669**	**651**	**1549**	*1549*	**1582**	**1580**
66	**1071**	*1071*	**1129**	**1128**	**2313**	*2313*	**2360**	**2355**
71	**1254**	*1254*	**1272**	**1266**	**3741**	*3741*	**3786**	3791
86	**457**	*457*	**508**	**507**	**1153**	*1153*	**1194**	*1194*
91	**874**	*874*	**955**	**947**	**2132**	*2132*	**2204**	**2199**
96	**1531**	*1531*	**1607**	**1597**	**3093**	*3093*	**3185**	3187
111	**621**	*621*	**689**	**665**	**1325**	1331	**1365**	1397
116	**627**	**619**	**695**	**661**	**2164**	*2164*	**2222**	2264
121	**1377**	*1377*	**1469**	*1469*	**2934**	2939	**2999**	3089

Table 7: *Bench* and *Best* values found

For the case of 40 jobs and 5 machines SA in four instances (11, 19, 21, 121) obtained the same value as the benchmark. In all other instances found better values. Furthermore, in the scenario of 100 jobs and 2 machines, SA obtains a value less than the benchmark in instance 36. In the case of instance 121, the proposed algorithm does not reach the benchmark value but by a little difference; in all the remaining instances reaches the benchmark values. In the last case analyzed, 100 jobs and 5 machines, SA improves the benchmark values in 12 instances (1, 6, 11, 21, 26, 31, 36, 41, 56, 61, 66 and 91). In two instances, 46 and 86 matches the benchmark. It reaches values close to benchmark in instances 19, 71, and 96; but the values obtained in the instances 111, 116, and 121 are further away from the known values. Previously observed behaviours allow us to assume that SA behaves fairly well for problems that

involve more machines because it improves or reaches the known values of the objective function. In the case of 2 machines, it reaches in most instances the benchmark values and also produces some improvements. Since the true optimal values are unknown, we may not conclude categorically if the number of machines makes the problem harder or if we do not improve the benchmark is because these are the true optimum.

6 Conclusion

The parallel machine environment has been studied for several years due to its importance both academic and industrial. Unlike other scheduling problems we could not find significant benchmarks for the problem of our interest, so in previous works we created our own instances, for 40 and 100 jobs, extracting data from the OR-Library corresponding to Weighted Tardiness and then we adapt them for the T_{max} problem. The main objective of our work was propose an improved version of SA with additional exploration capabilities in order to find new benchmark values (when possible) on the 20 instances analyzed in each case. This objective was achieved for several considered scenarios, the improved version of SA found new benchmark values. These results encourage us to continue with our research in two main directions: *a*) discuss alternatives regarding the combination of trajectory-based metaheuristics (e.g., SA with VNS or GRASP and also SA with population-based metaheuristics), and *b*) increase the quantity of instances to be considered, by adapting instances of the Weighted Tardiness Problems available in the OR-Library in order to obtain an extended set of instances for future research.

Acknowledgments. The authors would like to thank to the Universidad Nacional de San Luis for its continuous support.

References

1. C. Chen, *An Iterated Local Search for Unrelated Parallel Machines Problem with Unequal Ready Times*, Proceedings of the IEEE International Conference on Automation and Logistics Qingdao, China September 2008.
2. C. Mihil□, A. Mih□il□, *An Evolutionary Algorithm for Uniform Parallel Machines Scheduling*, Second UKSIM European Symposium on Computer Modelling and Simulation, 978-0-7695-3325-4/08, 2008 IEEE DOI 10.1109/EMS.2008.34, 2008.
3. C. Gatica and S. Esquivel and G. Leguizamón, *Comparative Study of Trajectory Metaheuristics for the Resolution of Scheduling Problem of Unrestricted Parallel Identical Machines*, XVIII Congreso Argentino de Ciencias de la Computación, 2012.
4. C. Gatica and S. Esquivel and G. Leguizamón, *An ACO approach for the Parallel Machines Scheduling Problem*, Inteligencia Artificial 46(2010), 84-95, doi: 10.4114/ia.v14i46.1550, 2010.
5. D. Anghinolfi and M. Paolucci, *Parallel machines total tardiness scheduling with a new hybrid metaheuristic approach*, Computer Operations Res.34:3471-3490, 2007.
6. E. Ferretti and S. Esquivel, *Knowledge Insertion: An Efficient Approach to Simple Genetic Algorithms for Unrestricted for Parallel Equal Machines Scheduling*. GECCO'05, 1587-1588, 2005, Washington DC, USA.

7. E. Ferretti and S. Esquivel, *An Efficient Approach of Simple and Multirecombinated Genetic Algorithms for Parallel Machine Scheduling*, IEEE Congress on Evolutionary Computation, 1340-1347, September 2005, Scotland, UK, IEEE Centre.
8. E. Ferretti and S. Esquivel, *A Comparison of Simple and Multirecombinated Evolutionary Algorithms with and without Problem Specific Knowledge Insertion, for Parallel Machines Scheduling*, International Transaction on Computer Science and Engineering, 2005, volume 3, number 1, 207-221.
9. E. G. Talbi, *Metaheuristics from design to implementation*, by John Wiley & Sons, Canada, 2009.
10. J. E. Beasley, *OR-Library: distributing test problems by electronic mail*, Journal of the Operational Research Society 41 (11), 1990, pp 1069-1072, as mentioned on. *http://people.brunel.ac.uk/[squigle]mastjjb/jeb/info.html*.
11. J. Arnaout and R. Musa and G. Rabadi, *Ant colony optimization algorithm to parallel machine scheduling problem with setups*, 4th IEEE Conference on Automation Science and Engineering Key Bridge Marriott, Washington DC, USA August 23-26, p:578-582, 2008.
12. J. Derrac, S. Garcıa, D. Molina, F. Herrera, *A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms*, Swarm and Evolutionary Computation, 2011.
13. K. Li and B. Cheng, *Variable neighbourhood search for uniform parallel machine makespan scheduling problem with release dates*, 2010 International Symposium on Computational Intelligence and Design.
14. M. Sevaux and K. Sörensen, *VNS/TS for a parallel machine scheduling problem*, MEC-VNS: 18th Mini Euro Conference on VNS, 2005.
15. M. Friedman, *The use of ranks to avoid the assumption of normality implicit in the analysis of variance*, Journal of American Statistical Association 3 (1937) 674-701
16. M. Friedman, *A comparison of alternative test of significance for the problem of the m rankings*, Annals of Mathematical Statistics 11 (1940) 86-92.
17. M. Pinedo, *Scheduling: Theory, Algorithms and System*, Prentice Hall, 1995.
18. N. Piersman and W. van Dijk, *A local search heuristic for unrelated parallel machine scheduling with efficient neighbourhood search*, Mathematical and Computer Modelling, vol. 24, no. 9, pp. 11-19, 1996.
19. P. Sivasankaran and T. Sornakumar and R. Panneerselvam, *Design and Comparison of Simulated Annealing Algorithm and GRASP to Minimize Makespan in Single Machine Scheduling with Unrelated Parallel Machines*, Intelligent Information Management, 2010, 2, 406-416, doi:10.4236/iim.2010.27050 Published Online July 2010 (http://www.SciRP.org/journal/iim).
20. R Project, *The R Project for Statistical Computing*, http://www.rproject.org/.
21. T. Bartz-Beielstein, *Experimental Research in Evolutionary Computation*, The New Experimentalism, Springer, 2006.
22. T. Bäck, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*, Institute of Physics Publishing Bristol Philadelphia and Oxford University Press, New York, USA, 1997.