A parallel metaheuristics for the single machine total weighted tardiness problem with sequence-dependent setup times

Wojciech Bożejko
Wrocław University of Technology
Institute of Computer Engineering, Control and Robotics
Janiszewskiego 11-17, 50-372 Wrocław, Poland
wojciech.bozejko@pwr.wroc.pl

Mieczysław Wodecki
University of Wrocław
Institute of Computer Science
Joliot-Curie 15, 50-383 Wrocław, Poland
mwd@ii.uni.wroc.pl

In the paper we propose a parallel population-based metaheuristics for solving a single machine scheduling problem with total weighted tardiness criterion and sequence-dependent setup times. It is represented by $1|s_{ij}|\sum w_iT_i$ in literature and it belongs to the strongly NP-hard class. Calculations on the representative group of benchmark instances were done and results were compared with the best known from literature. Obtained solutions were better than the benchmark ones in many instances.

Keywords: Meta-heuristic Search.

1 Introduction

The single machine total weighted tardiness problem with sequence-dependent setup times is denoted in literature as $1|s_{ij}|\sum w_iT_i$ and it is strongly NP-hard, because $1||\sum w_iT_i$ (with $s_{ij} = 0$) is strongly NP-hard (see Lenstra, Rinnoy Kan and Brucker (1977)). To date the best construction heuristics for this problem has been the Apparent Tardiness Cost with Setups (ATCS – Lee, Bhaskaran and Pinedo (1997)). Many metaheuristics have also been proposed. Tan et al. (2000) presented a comparison of four methods of solving the considered problem: Branch and Bound, Genetic Search, random-start pair-wise interchange and Simulated Annealing. Gagné, Price and Gravel (2002) compared an Ant Colony Optimization algorithm with other heuristics. Cicirello and Smith (2005) proposed benchmarks for the single machine total tardiness problem with sequence-dependent setups by generated 120 instances and applied stochastic sampling approaches: Limited Discrepancy Search (LDS), Heuristic-Biased Stochastic Sampling (HBSS), Value Biased Stochastic Sampling (VBSS), Value Biased Stochastic Sampling seeded Hill-Climber (VBSS-HS) and Simulated Annealing. The best goal function value obtained by their approaches was published in literature and presented at http://www.ozone.ri.cmu.edu/benchmarks.html as the upper bounds of the benchmark problems. This upper bounds was next improved by Cicirello (2006) by Genetic Algorithm, Lin and Ying (2006) by Tabu Search, Simulated Annealing and Genetic Algorithm, and Liao and Juan (2007) by an Ant Optimization.

In this paper we propose a method by which we have obtained the new better upper bound values. It is based on the idea which we have introduced in the paper [1].

2 Definition of the problem

Let $N = \{1, 2, \ldots, n\}$ be a set of n jobs which have to be processed, without an interruption, on one machine. This machine can process at the most one job in any time. For a job i ($i = 1, 2, \ldots, n$), let p_i,w_i,d_i be: a *time of executing*, a *weight of the cost function* and a *deadline*. Let s_{ij} be a setup time which represents a time is needed to prepare the machine for executing a job j after finishing executing a job i. Additionally s_{0i} is a time which is needed to prepare a machine for executing the first job i (at the beginning of the machine work). If a sequence of job's executing is determined and C_i (i =1,2, ..., n) is a time of finishing executing a job i, then $T_i = \max\{0, C_i - d_i\}$ we call a *tardiness*, and $f_i(C_i) = w_i T_i$ *a cost of tardiness* of a job i. The considered problem consists in determining such a sequence of executing of jobs which minimizes a *sum of costs of tardiness*, i.e. $\sum w_i T_i$.

Let Π be a set of permutations of elements from the set N. For a permutation $\pi \in \Pi$ by

$$F(\pi) = \sum_{i=1}^{n} f_{\pi(i)}(C_{\pi(i)}),$$

we represent a *cost of permutation* π (i.e. a sum of costs of tardiness, when jobs are executed in a sequence determined by a permutation π), where $C_{\pi(i)} = \sum_{j=1}^{i} (s_{\pi(j-1)\pi(j)} + p_{\pi(j)})$ and $\pi(0) = 0$. The considered problem consists in determining a permutation $\pi \in \Pi$ which has a minimal sum of costs of tardiness.

3 Parallel Population-Based Metaheuristics

We present a method belonging to the population-based approaches which consists in determining and researching the local minima. This (heuristic) method is based on the following observation. If there are the same elements in some positions in several solutions, which are local minima, then these elements can be in the same position in the optimal solution.

Since we propose this method for solving problems in which a solution is a permutation, so in the next part of the paper we identify these two terms.

The basic idea is to start with an initial population (any subset of the solution space). Next, for each element of the population, a local optimization algorithm is applied (e.g. descending search algorithm or a metaheuristics) to determine a local minimum. In this way we obtain a set of permutations – local minima. If there is an element which is in the same position in several permutations, then it is fixed in this position in the permutation, and other positions and elements of permutations are still free. A new population (a set of permutations) is generated by drawing free elements in free positions (because there are fixed elements in fixed positions). After determining a set of local minima (for the new population) we can increase the number of fixed elements. To prevent from finishing the algorithm's work after executing some number of iterations (when all positions are fixed and there is nothing left to draw), in each iteration "the oldest" fixed elements are set as free.

For the parallel version of the algorithm a global model of parallelization has been proposed. This model executes multiple population training metaheuristics synchronizing populations in each iteration, i.e. common global table of the fixed elements and positions is used for each processor. In every iteration the average number $nr(a, l)$ of permutations (for all subpopulations) in which there is an element a in a position l is computed.

Let Π be a set of all permutations of elements from the set $N = \{1, 2, \ldots, n\}$ and the function:

$$F : \Pi \to R^+ \cup \{0\}.$$

We consider a problem which consists in determining optimal permutation $\pi_{opt} \in \Pi$. We use the following notation: π^* – sub-optimal permutation determined by the algorithm, η – number of elements in the population, P^i – population in the iteration i of the algorithm, $P^i = \{\pi_1, \pi_2, \ ... \ , \pi_\eta\}$, $LocalOpt(\pi)$ – local optimization procedure to determine local minimum where π is a starting solution, LM^i – a set of local minima in iteration i, $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \ ... \ , \hat{\pi}_\eta\}$, $\hat{\pi}_j = LocalOpt(\pi_j)$, $\pi_j \in P^i, \ j = 1, 2, \ ... \ , \eta$, FS^i – a set of fixed elements and position in permutations of population P^i, $FixSet(LM^i, FS^i)$ – a procedure which determines a set of fixed elements and positions in the next iteration of the population-based algorithm, $FS^{i+1} = FixSet(LM^i, FS^i)$, $NewPopul(FS^i)$ – a procedure which generates a new population in the next iteration of algorithm, $P^{i+1} = NewPopul(FS^i)$.

In any permutation $\pi \in P^i$ positions and elements which belong to the set FS^i (in iteration i) we call *fixed*; other elements and positions we call *free*.

The algorithm begins by creating an initial population P^0 (and it can be created randomly). We set a sub-optimal solution π^* as the best element of the population P^0,

$$F(\pi^*) = min\{F(\beta) : \ \beta \in P^0\}.$$

A new population of iteration $i + 1$ (a set P^{i+1}) is generated as follows: for a current population P^{i+1} a set of local minima LM^i is determined (for each element $\pi \in P^i$ executing procedure $LocalOpt(\pi)$). Elements which are in the same positions in local minima are established (procedure $FixSet(LM^i, FS^i)$), and a set of fixed elements and positions FS^{i+1} is generated. Each permutation of the new population P^{i+1} contains the fixed elements (in fixed positions) from the set FS^{i+1}. Free elements are randomly drawn in the remaining free positions of permutation.
If permutation $\beta \in LM^i$ exists and $F(\beta) < F(\pi^*)$, then we update π^* ($\pi^* \leftarrow \beta$). The algorithm finishes (*Stop Criterion*) after executing the *Max_iter* iterations.

The general structure of the parallel population-based heuristic algorithm for the permutation optimization problem for each processor is given below. The $FixSet$ and $NewPopul$ algorithms are described in the further part of the paper.

Algorithm 1. Parallel Population-Based Metaheuristics (ParPBM)
Initialization:
$P^0 \leftarrow \{\pi_1, \pi_2, \ldots, \pi_\eta\}$; *starting populations for each of nrtasks populations*
$\pi^* \leftarrow$ the best element of the population P^0;
$i \leftarrow 0$;
$FS^0 \leftarrow \emptyset$;
parfor $p = 1..nrtasks$ **do**
  repeat
    For each process p determine sets of local minima
      $LM_p^i \leftarrow \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$, where $\hat{\pi}_j \leftarrow LocalOpt(\pi_j), \pi_j \in P_p^i$;
    for $j \leftarrow 1$ **to** η **do** **if** $F(\hat{\pi}_j) < F(\pi^*)$ **then** $\pi^* \leftarrow \hat{\pi}_j$;
    Determine sets $FS^{i+1} \leftarrow FixSet(LM^i, FS^i)$;
    Generate new populations $P^{i+1} \leftarrow NewPopul(FS^i)$;
    $i \leftarrow i + 1$;
  until $i > max\_iter$;
end parfor.

Local optimization (*LocalOpt* procedure.) A fast method based on the local improvement is applied to determine the local minima. The method begins with an initial solution π^0. In

each iteration for the current solution π^i the neighborhood $\mathcal{N}(\pi^i)$ is determined. Next, from the neighborhood it is chosen the best element π^{i+1} is chosen which is the current solution in the next iteration.

In the implementation of the *LocalOpt* procedure a very quick *descent search* algorithm is applied. The neighborhood is generated by swap moves (s-moves). Local search procedure does not preserve the positions of fixed elements to obtain better (closer to optimal) values of the local minima.

A set of fixed elements and position (*FixSet* procedure). The set FS^i (in iteration i) includes quadruples (a, l, α, φ), where a is an element of the set $N = \{1, 2, \ldots, n\}$, l is a position in the permutation ($1 \leq l \leq n$) and α, φ are attributes of a pair (a, l). A parameter α means "*adaptation*" and decides on inserting to the set, and φ – "*age*" – decides on deleting from the set. Parameter φ enables to set free a fixed element after making a number of iterations of the algorithm. However, a parameter α determines a fraction of local minima, in which an element a is in position l.
Both of these parameters are described in a further part of this chapter. The maximal number of elements in the set FS^i is n. If the quadruple (a, l, α, φ) belongs to the set FS^i, then there is an element a in the position l in each permutation from the population P^i.

In each iteration of the algorithm, after determining local minima (*LocalOpt* procedure), a new set $FS^{i+1} = FS^i$ is established. Next, a $FixSet(LM^i, FS^i)$ procedure is invoked in which the following operations are executed:

(1) fixing the new elements,

(2) erasing the oldest,

(3) modifying of the age of each element.

There are two functions of acceptance Φ and Γ connected with the operations of inserting and deleting. Function Φ is determined by an auto-tune function. Function Γ is fixed experimentally as a constant.

Fixing elements. Let $P^i = \{\pi_1, \pi_2, \ldots, \pi_\eta\}$ be a population of η elements in the iteration i. For each permutation $\pi_j \in P^i$, applying the local search algorithm ($LocalOpt(\pi_j)$ procedure), a set of local minima $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$ is determined. For any permutation $\hat{\pi}_j = (\hat{\pi}_j(1), \hat{\pi}_j(2), \ \ldots, \hat{\pi}_j(n)), \quad j = 1, 2, \ \ldots \eta$, let be $nr(a, l) = \left|\{\hat{\pi}_j \in LM^i : \quad \hat{\pi}_j(l) = a\}\right|$. It is a number of permutations from the set LM^i in which the element a is in the position l.
If $a \in N$ is a free element (i.e. $a \notin FS^i$) and $\alpha = \frac{nr(a,l)}{\eta} \geq \Phi(i)$, then the element a is fixed in the position l; we assume $\varphi = 1$ and the quadruple (a, l, α, φ) is inserted to the set of fixed elements and positions, it means $FS^{i+1} \leftarrow FS^i \cup \{(a, l, \alpha, \varphi)\}$.

Auto-tune of the acceptance level Φ. Acceptance function Φ is defined so that

$$\forall i, \ \ 0 < \Phi(i) \leq 1.$$

It is possible that no element is acceptable to be fixed in an iteration, or too many elements (i.e. a half) are fixed in one iteration. To prevent from it an auto-tune procedure for Φ value is proposed. In each iteration i, if

$$\max_{a,l \in \{1,2,\ldots,n\}} \frac{nr(a,l)}{\eta} < \Phi(i) \quad \text{or} \quad \max_{a,l \in \{1,2,\ldots,n\}} \frac{nr(a,l)}{\eta} \gg \Phi(i)$$

therefore, $\Phi(i)$ value is fixed as

$$\Phi(i) \leftarrow \max_{a,l \in \{1,2,\ldots,n\}} \frac{nr(a,l)}{\eta} - \epsilon,$$

where ϵ is a small constant, e.g. $\epsilon = 0.05$. In this way the value of $\Phi(i)$ is adopted to the problem instance specificity. In this implementation of the algorithm $\Phi(0) = 0.7$ as a starting value.

Deleting elements. Each fixed element is released after executing some number of iterations to makes possible testing a plenty of local minima. In this implementation function $\Gamma(i)$ is defined as a constant equals 2, so each element of the set FS^i is deleted after executing 2 iterations.

In the i-th iteration of the ParPBM algorithm a set of fixed elements and positions FS^{i+1} is determined by using the following algorithm.

Algorithm 2. FixSet(LM^i, FS^i)
Input: a set of local minima LM^i;
Output: a set of fixed elements and position FS^i;
for all $k, j \in N$ **do** $nr(k,j) \leftarrow 0$; **end for**;
for j:=1 **to** η **do**
for a:=1 **to** n **do**
$nr(a, \hat{\pi}_j(a)) \leftarrow nr(a, \hat{\pi}_j(a)) + 1$;
end for;
end for;
for a:=1 **to** n **do**
if (a is a free job) **then**
choose l such, that $nr(a,l) = \max_{1 \le s \le n} \{nr(a,s)\}$;
if $nr(a,l) \ge \Phi(i)$ **then**
fix an element a on the position l
(i.e. $FS^{i+1} \leftarrow FS^i \cup \{(a,l,\alpha,\varphi)\}$, where $\alpha = \frac{nr(a,l)}{\eta}$ and $\varphi = 0$;)
end for;
{*Increasing age of each fixed element and setting free the old ones*}
for a:=1 **to** n **do**
$\varphi(a) \leftarrow \varphi(a) + 1$;
if $\varphi(a) > \Gamma(i)$ **then** set free an element a
(i.e. $FS^{i+1} \leftarrow FS^{i+1} \setminus \{(a,l,\alpha,\varphi)\}$);
end for.

Procedure *NewPopul*. Let a quadruple $(a,l,\alpha,\varphi) \in FS^{i+1}$. Therefore, in each permutation of a new population P^{i+1} there exists an element a in a position l. Randomly drawn free elements will be inserted in remaining (free) positions. A function *random* generates an element of the set FE from the uniform distribution. Computational complexity of the *NewPopul* algorithm is linear. Population P^{i+1} is generated as follows:

Algorithm 3. New Population (*NewPopul*(FS_{i+1}))

$P^{i+1} \leftarrow \emptyset$;
Determine a set of free elements:
$FE \leftarrow \{a \in N : \ \neg \exists\, (a, l, \alpha, \varphi) \in FS^{i+1}\}$
and a set of free positions:
$FP \leftarrow \{l : \ \neg \exists\, (a, l, \alpha, \varphi) \in FS^{i+1}\}$;
for $j \leftarrow$ **to** η **do**
for every $(a, l, \alpha, \varphi) \in FS^{i+1}$ **do** $\pi_j(l) \leftarrow a$; **end for**;
for $s \leftarrow 1$ **to** n **do**
if $s \in FP$ **then**
$w \leftarrow random(FE)$ **and** $FE \leftarrow FE \backslash \{w\}$;
$\pi_j(s) \leftarrow w$;
end for;
$P_{i+1} \leftarrow P_{i+1} \cup \{\pi_j\}$.
end for.

4 Computational experiments

Parallel population-based metaheuristics was implemented in C++ language with the MPI library (MPICH) and it was tested on the cluster of 8 computers with Intel Celeron 2.4GHz processors and 240MB RAM memory working under Windows 2000 operating system connected by Ethernet 100mbit/s. Computational experiments were done to compare the obtained results with the benchmarks from literature (Cicirello and Smith (2005), represented by a in the tables) and the newest obtained results for this single machine problem (Liao and Juan (2007), ACO, represented by b; Lin and Ying (2006): SA, GA, TS, represented by c, d, e, respectively; and SA-TABU, f (on line).

The following tuning parameters was used: $max\_iter = 520$, threshold $\epsilon = 0.05$ for Φ adjusting, population size per each processor $\eta = 200$, maximal age $\Gamma(i) = 2$.

In the Table 1 results of computational experiments for the problem $1|s_{ij}|\sum w_i T_i$ are presented with the new upper bounds marked. Average percentage deviation δ of the obtained solution value F_{ParPBM} to benchmark ones F_{bench} from Cicirello and Smith (2005) is also computed: $\delta = \frac{F_{ParPBM} - F_{bench}}{F_{bench}} \cdot 100\%$. Benchmark values F_{bench} are not the best knows, but all the authors are comparing to them, so the average value of δ is rational measurement of the algorithm efficiency. As we can see it was possible to find 18 new upper bounds of the optimal cost function for the benchmark instances. The time consumed by all processors took 7 minutes in average.

Table 2 presents the average percentage deviation from the benchmark solutions of the newest approaches to the concerned problem compared to sequential implementation of the ParPBM (for one processor). It was on the level of $-10,05\%$ with average time 20 seconds and was better than earlier proposed approaches to this problem with comparable times of execution (the authors of SA^c, GA^d and TS^e announce 27 seconds in average on Pentium IV 1.4GHz).

5 Conclusion

We have discussed a new approach to the permutation optimization problems based on the parallel population-based metaheuristic algorithm. Usage of the population with fixed features of local optima makes the performance of the method much better than the iterative improvement approaches, such as in tabu search, simulated annealing as well as classical genetic algorithms.

Table 1: Results of computational experiments for the problem $1|s_{ij}|\sum w_i T_i$. The new upper bounds are marked by bold font.

Nr	F_{best}	F_{ParPBM}	$\delta(\%)$	Nr	F_{best}	F_{ParPBM}	$\delta(\%)$
1	684^{d}	696	-28,83%	61	76357^{f}	76373	-4,40%
2	5082^{e}	5367	-17,29%	62	44769^{f}	44869	-6,25%
3	1792^{c}	**1782**	-24,11%	63	75317^{c}	76146	-3,39%
4	6526^{d}	6615	-20,41%	64	92572^{c}	92860	-3,65%
5	4662^{d}	4774	-14,84%	65	126907^{f}	128593	-4,66%
6	5788^{b}	7500	-9,02%	66	59717^{f}	59852	-6,56%
7	3693^{d}	3765	-13,39%	67	29390^{d}	29394	-15,77%
8	142^{d}	153	-53,21%	68	22148^{e}	22528	-14,68%
9	6264^{f}	6628	-12,77%	69	64632^{b}	71534	-5,14%
10	2021^{c}	2099	-14,36%	70	75102^{d}	75801	-6,65%
11	3867^{d}	4452	-15,41%	71	150053^{f}	151866	-5,81%
12	0^{a}	0	0,00%	72	46903^{e}	50171	-11,88%
13	5685^{d}	5797	-5,69%	73	29408^{c}	30529	-16,28%
14	3045^{d}	3205	-18,68%	74	33375^{e}	34357	-10,28%
15	1458^{d}	1788	-38,66%	75	21863^{e}	23244	-24,97%
16	4940^{d}	**4939**	-26,40%	76	55055^{c}	56198	-16,81%
17	204^{c}	**194**	-58,01%	77	33303^{f}	35932	-11,41%
18	1610^{d}	1723	-31,46%	78	20632^{f}	22256	-11,35%
19	36^{f}	**0**	-100,00%	79	119099^{f}	121085	-3,77%
20	2967^{d}	3273	-21,94%	80	20161^{f}	23138	-27,34%
21	0^{a}	0	0,00%	81	384996^{d}	385528	-0,42%
22	0^{a}	0	0,00%	82	410458^{f}	**410432**	-0,74%
23	0^{a}	0	0,00%	83	459817^{f}	460459	-1,20%
24	1063^{d}	**1060**	-40,82%	84	330384^{c}	330837	-0,25%
25	0^{a}	0	0,00%	85	555106^{c}	556891	-0,30%
26	0^{a}	0	0,00%	86	364075^{f}	365924	0,04%
27	0^{d}	0	-100,00%	87	399214^{f}	401482	-0,38%
28	0^{d}	0	-100,00%	88	434948^{d}	436394	-0,11%
29	0^{a}	0	0,00%	89	410966^{c}	**410909**	-1,44%
30	165^{c}	**159**	-72,35%	90	402233^{d}	403601	-0,82%
31	0^{a}	0	0,00%	91	344988^{e}	346111	-0,31%
32	0^{a}	0	0,00%	92	364593^{f}	**362823**	-0,81%
33	0^{a}	0	0,00%	93	410462^{a}	413534	0,75%
34	0^{a}	0	0,00%	94	335210^{f}	335542	-0,23%
35	0^{a}	0	0,00%	95	519364^{f}	523309	-0,87%
36	0^{a}	0	0,00%	96	461484^{b}	462961	-0,31%
37	755^{c}	986	-59,04%	97	413109^{e}	417890	-0,57%
38	0^{a}	0	0,00%	98	532519^{a}	530958	-0,29%
39	0^{a}	0	0,00%	99	370084^{b}	373734	-0,28%
40	0^{a}	0	0,00%	100	439944^{d}	**437403**	-1,01%
41	71186^{e}	71418	-2,40%	101	353408^{e}	**353141**	-0,75%
42	58199^{e}	59377	-4,01%	102	493889^{e}	494300	-0,37%
43	147211^{c}	147721	-1,51%	103	379913^{b}	**379195**	-0,26%
44	35648^{c}	35742	-7,71%	104	358334^{c}	358741	-0,90%
45	59307^{d}	59572	-5,08%	105	450808^{c}	**450806**	-1,22%
46	35320^{e}	35909	-5,48%	106	455849^{d}	457420	-0,54%
47	73972^{f}	74024	-4,10%	107	352855^{f}	**352821**	-1,07%
48	65164^{c}	65943	-4,32%	108	462737^{e}	463753	-0,93%
49	79055^{e}	79224	-5,85%	109	413205^{c}	413569	-0,54%
50	32743^{f}	33201	-8,37%	110	419481^{e}	420152	-0,27%
51	52163^{f}	**51854**	-11,47%	111	347233^{b}	349805	-0,26%
52	99200^{d}	102688	-2,54%	112	373238^{b}	376036	-0,37%
53	91302^{c}	92111	-3,50%	113	261239^{d}	**261001**	-0,84%
54	122968^{f}	124708	0,93%	114	470327^{b}	472780	-0,09%
55	69571^{f}	71657	-6,17%	115	459194^{b}	464415	0,91%
56	78960^{d}	79756	-9,80%	116	527459^{b}	537799	-0,45%
57	69320^{c}	**68489**	-2,73%	117	512028^{f}	**508415**	-1,96%
58	48081^{e}	48637	-12,40%	118	352118^{b}	357087	-0,14%
59	55396^{c}	56376	-4,54%	119	579462^{e}	584046	0,02%
60	68176^{f}	69103	-5,76%	120	398590^{b}	402422	0,68%
				Average			**-10,28**

Table 2: Average improvement rates (%) of the SA, GA and TS compared to the proposed ParPBM (sequential version) approach. Standard deviation of the ParPBM results σ^{PHM} over 10 runs.

Problem set	SA[c]	GA[d]	TS[e]	ParPBM	σ^{ParPBM}
1 to 10	20.00	22.83	19.12	20.82	3.47
11 to 20	20.89	27.60	18.46	31.63	33.62
21 to 30	30.39	30.93	29.18	29.79	31.85
31 to 40	6.86	6.42	5.81	5.90	3.59
41 to 50	5.21	5.65	5.33	4.83	2.02
51 to 60	5.29	5.65	4.44	5.01	6.11
61 to 70	7.25	6.56	7.25	7.06	3.80
71 to 80	15.39	15.02	16.32	13.89	5.80
81 to 90	0.66	0.56	0.56	0.46	0.40
91 to 100	-0.47	-0.50	-0.11	0.37	0.64
101 to 110	0.60	0.24	0.64	0.68	0.45
111 to 120	-0.23	-0.44	-0.23	0.19	1.01
Average	9.32	9.97	8.90	10.05	7.73

References

[1] Bożejko W. and M. Wodecki (2006). Evolutionary Heuristics for Hard Permutational Optimization Problems, International Journal of Computational Intelligence Research, Vol. 2, Issue 2, Research India Publications, 151-158.

[2] Cicirello V.A. and S.F. Smith. (2005). Enhancing stochastic search performance by value-based randomization of heuristics, Journal of Heuristics, 11, 5-34.

[3] Cicirello V.A. (2006). Non-Wrapping Order Crossover: An Order Preserving Crossover Operator that Respect Absolute Position, 8th Annual Genetic and Evolutionary Computation Conference GECCO 2006, ACM Press, 1125-1131.

[4] Gagné C., W.L. Price and M. Gravel (2002). Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times, Journal of the Operational Research Society, 53, 895-906.

[5] Lee Y.H., K. Bhaskaran and M. Pinedo. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups, IIE Transactions, 29, 45-52.

[6] Lenstra J.K., A.G.H. Rinnoy Kan and P. Brucker. (1977). Complexity of Machine Scheduling Problems, Annals of Discrete Mathematics, 1, 343-362.

[7] Liao C.-J. and H. C. Juan. (2007) An ant opimization for single-machine tardiness shceduling with sequence-dependent setups, Computers & Operations Research, 34, 1899-1909.

[8] Lin S.-W. and K.-C. Ying (2006), Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics, International Journal of Advanced Manufacturing Technology (on line), DOI 10.1007/s00170-006-0693-1.

[9] Lin S.-W. and K.-C. Ying, Hybrid simulated annealing and tabu search for solving single-machine tardiness problems with sequence-dependent setup times, (on line) http://pc33.mis.hfu.edu.tw/smswtsds/data/

[10] Tan K.C., R. Narasimban, P.A. Rubin and G.L. Ragatz. (2000). A comparison on four methods for minimizing total tardiness on a single procesor with sequence dependent setup times, Omega, 28, 313-326.