A Hybrid Algorithm for the Single-machine Total Tardiness Problem

T. C. E. Cheng [a], A. A. Lazarev [b] [c], E. R. Gafarov [d]

[a] Department of Logistics, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong. Email: lgtcheng@polyu.edu.hk

[b] Corresponding author. Computing Centre of the Russian Academy of Sciences, Vavilov st. 40, 119333 Moscow GSP-1, Russia. Tel, fax: +007 495 1356238. Email: alaz@ccas.ru, jobmath@mail.ru

[c] Partially supported by DAAD (Deutsche Akademische Austauschdienst): A0206237/Ref. 325 and by Russian funding support of scientific school N - 5833.2006.1

[d] Computing Centre of the Russian Academy of Sciences, Vavilov st. 40, 119333 Moscow GSP-1, Russia. Tel, fax: +007 495 1356238. Email: axel73@mail.ru

Abstract

We propose a hybrid algorithm based on the Ant Colony Optimization (ACO) meta-heuristic, in conjunction with four well-known elimination rules, to tackle the NP-hard single-machine scheduling problem to minimize the total job tardiness. The hybrid algorithm has the same running time as that of ACO. We conducted extensive computational experiments to test the performance of the hybrid algorithm and ACO. The computational results show that the hybrid algorithm can produce optimal or near-optimal solutions quickly, and its performance compares favourably with that of ACO for handling standard instances of the problem.

Keywords: Scheduling, Meta-heuristics

Introduction

We are given a set N of n independent jobs that must be processed on a single machine. Preemption of the jobs is not allowed. The machine can handle only one job at a time. All the jobs are assumed to be available for processing at time 0. For each job j, $j \in N$, a processing time $p_j > 0$ and a due date d_j are given. A schedule π is uniquely determined by a permutation of the elements of N. Define $T_j(\pi) = \max\{0, c_j(\pi) - d_j\}$ as the tardiness of job j under schedule π, where $c_j(\pi)$ is the completion time of job j under

schedule π. We seek to find an optimal schedule π^* that minimizes the total job tardiness, i.e., $F(\pi) = \sum_{j=1}^{n} T_j(\pi)$. The problem is denoted by $1||\sum T_j$ and is known to be NP-hard in the ordinary sense [1]. Lawler [2] presented an $O(n^4 \sum p_j)$ time dynamic programming algorithm for the problem. Szwarc et al. [3, 4] designed enumeration algorithms to handle the special instances of the problem discussed in [5] for $n \leq 500$. It was shown in [6] that all known constructive and decomposition heuristics for non-paradoxical instances of the problem can yield arbitrarily bad approximation ratios.

In this paper we propose a hybrid algorithm based on the Ant Colony Optimization (ACO) meta-heuristic by Bauer et al. [7], in conjunction with the four well-known Elimination Rules 1-4 for $1||\sum T_j$ introduced in [4, 8, 9]. We conducted comprehensive computational experiments to compare the performance of the hybrid algorithm and ACO with respect to the following measures: percentage of time that an optimal solution is found, relative error of the solution found, and number of iterations needed to find an optimal solution, where one ant in ACO is equivalent to one iteration of the solution procedure. We tested both algorithms using three special cases of $1||\sum T_j$, namely the special instances of Potts and van Wassenhove [5], the case B-1 of Lazarev [8], and the canonical instances of Du and Leung [1].

The paper is organized as follows. We introduce in the next section Elimination Rules 1-4, and an exact algorithm that solves the problem optimally. In the following section, we present the ACO algorithm. We then discuss our hybrid algorithm, and present the results of the computational experiments in Sections 3-6. We conclude the findings in the final section.

1 An exact solution algorithm

Without loss of generality, let $d_1 \leq d_2 \leq \ldots \leq d_n$; if $d_k = d_{k+1}$ then $p_k \leq p_{k+1}$. In other words, the jobs are first sequenced in the earliest due date (EDD) order, and if there is a tie, then the jobs are sequenced in the shortest processing time (SPT) order, i.e., for jobs that have the same due dates, they are sequenced in nondecreasing order of their processing times.

We denote by $I = \langle \{p_j, d_j\}_{j \in N}, t \rangle$ an instance of the problem $1||\sum T_j$ with the job set N and parameters $\{p_j, d_j\}_{j \in N}$ that must be processed from the start time t. Let $(j_1 \rightarrow j_2)_\pi$ denote that job j_1 precedes job j_2 under schedule π. Let j^* denote the job with the largest processing time in N.

Let $N = \{1, 2, \ldots, n\} = \{j_1, \ldots, j_n\}$, where $d_{j_1} \leq \ldots \leq d_{j_n}$. We denote

by $\pi^k = (j_1, \ldots, j_{m-1}, j_{m+1}, \ldots, j_k, j^*, j_{k+1}, \ldots, j_n)$, $j^* = j_m$, $m < k$, the modified EDD sequence, where job j^* is moved from its original position m to position k.

We consider a subset of jobs $N' \subseteq N$ that must be processed from time $t' \geq t_0$. Let $N' = \{1, 2, \ldots, n'\}$. Construct a position list $L(N', t')$ for the job j^*, the job with the largest processing time in N', as follows:

0. $L(N', t') := \{j^*, j^* + 1, \ldots, n'\}$; $d_{n'+1} := +\infty$;

1. FOR $k := j^* + 1$ **TO** n' **DO**

IF $t' + \sum_{j=1}^{k} p_j \geq d_{k+1}$ (Elimination Rule 1 [4, 8])
THEN $L(N', t') := L(N', t') \setminus \{k\}$;

FOR $i := j^* + 1$ **TO** n' **DO**

IF $d_i + p_i > t' + \sum_{j=1}^{k} p_j$ (Elimination Rules 2 and 3 [4, 8])
THEN $L(N', t') := L(N', t') \setminus \{k\}$.

FOR $i := j^*$ **TO** k **DO**

IF $F(\pi^k) > F(\pi^{k+1})$ or $F(\pi^k) \geq F(\pi^i)$ (Elimination Rule 4 [9, 10]) **THEN** $L(N', t') := L(N', t') \setminus \{k\}$.

Proposition 1 [8] *For all instances* $\langle \{p_j, d_j\}_{j \in N}, t_0 \rangle$, *the list* $L(N, t_0)$ *is not empty.*

Proposition 2 [8] *For all instances* $\langle \{p_j, d_j\}_{j \in N}, t_0 \rangle$, *there exists an optimal schedule* π^* *such that* $(j \to j^*)_{\pi^*}$ *for all* $j \in \{1, 2, \ldots, k\} \setminus \{j^*\}$ *and* $(j^* \to j)_{\pi^*}$ *for all* $j \in \{k+1, \ldots, n\}$ *for some* $k \in L(N, t_0)$.

Now we present Algorithm A by Lazarev [8], an exact solution algorithm for $1||\sum T_j$, which is based on Elimination Rules 1-4. Szwarc et al. [3] and Chang et al. [9] have applied similar ideas for constructing Algorithm A in designing algorithms for $1||\sum T_j$.

Procedure ProcL(N, t)

Input: *An instance* $\langle \{p_j, d_j\}_{j \in N}, t \rangle$ *with the job set* $N = \{j_1, j_2, \ldots, j_n\}$ *and start time* t*, where* $d_{j_1} \leq d_{j_2} \leq \ldots \leq d_{j_n}$*;*

1. **IF** $N = \emptyset$ **THEN** π^*:= empty schedule, **GOTO 6**;
2. Find $j^* \in N$; construct the list $L(N, t)$ for job j^*;
3. **FOR ALL** $k \in L(N, t)$ **DO:**

   $\pi_k := (\mathbf{ProcL}(N', t'), j^*, \mathbf{ProcL}(N'', t''))$, where
   $N' := \{j_1, \ldots, j_k\} \setminus \{j^*\}$, $t' := t$,
   $N'' := \{j_{k+1}, \ldots, j_n\}$, $t'' := t + \sum_{i=1}^{k} p_{j_i}$;
4. π^*:=arg $\min_{k \in L(N,t)} \{F(\pi_k, t)\}$;
5. **RETURN** π^*.

Algorithm A

π^*:=**ProcL**(N, t_0).

It is known that Algorithm A has an exponential running time for several cases of the problem $1||\sum T_j$ [10]. The running time of Algorithm A for instances of the problem with integer parameters is $O(n^4 \sum p_j)$ [2, 8].

2 Ant Colony Optimization for $1||\sum T_j$

We present the *ACO* algorithm by Bauer et al. [7] in this section. In each generation, each of the m ants constructs one solution. An ant selects the jobs in the order in which they appear in a schedule. For the selection of a job, the ant uses both heuristic and pheromone information. The heuristic information, denoted by η_{ij}, and the pheromone information, denoted by τ_{ij}, is an indicator of how good it seems to place job j in position i of the schedule. With probability q_0, where $0 < q_0 < 1$ is a parameter of the algorithm, the ant chooses the next job j from the set S of jobs that have not been scheduled so far that maximizes $[\tau_{ij}]^\alpha [\eta_{ij}]^\beta$, where α and β are constants that determine the relative influence of the pheromone values and the heuristic values, respectively, on the decision of the ant. With probability

$1-q_0$, the ant selects the next job j according to the probability distribution determined by

$$p_{ij} = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{h \in S} [\tau_{ih}]^\alpha [\eta_{ih}]^\beta}.$$

The heuristic values η_{ij} are computed according to the modified due date (MDD) rule, i.e., $\eta_{ij} = \frac{1}{\max\{T+p_j, d_j\}}$, where T is the total processing time of all the jobs that have already been scheduled.

After an ant has selected the next job j, a local pheromone update is performed at element $(i; j)$ of the pheromone matrix according to $\tau_{ij} := (1-\rho)\tau_{ij} + \rho\tau_0$ for some constant ρ, $0 < \rho < 1$, where $\tau_0 = \frac{1}{mT_{EDD}}$, and T_{EDD} is the total tardiness of the schedule that is obtained when the jobs are ordered according to the EDD rule. The value τ_0 is also used to initialize the elements of the pheromone matrix.

After each ant has constructed a solution, the solution is further improved with a 2-opt strategy, i.e., a local search procedure with pairwise swapping of jobs. The 2-opt strategy considers possible swaps between all pairs of jobs in the constructed sequence.

The best solution found so far is then used to update the pheromone matrix. But before doing so, some old pheromone values will decay according to $\tau_{ij} := (1-\rho)\tau_{ij}$. The reason is that old pheromone values should not have too strong an influence on the future. Then, for every job j in the schedule of the best solution found so far, some amount of pheromone is added to element $(i; j)$ of the pheromone matrix, where i is the position of job j in the schedule. The amount of pheromone added is ρ/T^*, where T^* is the total tardiness of the best found schedule, i.e., $\tau_{ij} := \tau_{ij} + \rho/T^*$. The algorithm stops when some stopping criterion is met, e.g., a certain number of generations has been reached or the best found solution has not changed for several generations.

Computational results of the ACO algorithm were presented in [7], where the instances from [5] for $n = 50$ and 100 were tested. For $n = 50$, ACO generated an optimal solution for 609 out of the 625 tested instances. The relative error was less than 0.08%. For $n = 100$, all 125 tested instances were solved optimally.

It is easy to show that the running time of ACO without local search is $O(mn^2)$. For each i (there is a total of n positions), job j is chosen in $O(n)$ time. Local search has a running time of $O(n^3)$, but the number of times local search is applied is unknown. So ACO has a running time no less than $O(mn^3)$. In practice, the running time of ACO does not exceed $O(mn^2)$.

3 A hybrid algorithm

We present in this section Algorithm H, a hybrid algorithm based on the *ACO* meta-heuristic by Bauer et al. [7], in conjunction with Elimination Rules 1-4 [4, 8, 9, 10].

In Algorithm H, each ant (i.e., each iteration) executes a modified version of Algorithm A, where the current job j^* is randomly placed in position $k \in L(N,t)$.

Procedure ProcL modified(N,t)

Input: *An instance* $\langle \{p_j, d_j\}_{j\in N}, t\rangle$ *with the job set* $N = \{j_1, j_2, \ldots, j_n\}$ *and start time* t*, where* $d_{j_1} \le d_{j_2} \le \ldots \le d_{j_n}$*;*

1. **IF** $N = \emptyset$ **THEN** π^*:= empty schedule, **GOTO 6**;
2. Find $j^* \in N$; construct the list $L(N,t)$ for job j^*;
3. Compute the array of probabilities for each $i \in L(N,t)$:

$$\rho_{ij*} = \frac{\tau_{ij*}/F(\pi^i)}{\sum_{h\in L(N,t)} \tau_{hj*}/F(\pi^h)},$$

   where $\pi^i = (j_1, \ldots, j_{m-1}, j_{m+1}, \ldots, j_i, j^*, j_{i+1}, \ldots, j_n),\ j^* = j_m, m < i$;
4. Choose $k \in L(N,t)$ randomly according to probability ρ_{kj*};
5. Update the local trail:

$$\tau_{kj*} := (1-\rho)\tau_{kj*} + \rho\tau_0,$$

   where $\tau_0 = 1/(mT_{EDD})$, and T_{EDD} is the total tardiness of schedule π_{EDD};
6. **RETURN**

   π^* :=(**ProcL**(N',t'), j^*,**ProcL**(N'',t'')), where
   $N' := \{j_1, \ldots, j_k\} \setminus \{j^*\}$, $t' := t$,
   $N'' := \{j_{k+1}, \ldots, j_n\}$, $t'' := t + \sum_{i=1}^{k} p_{j_i}$.

Upon completing each iteration (recalling that 1 ant = 1 iteration), we update the "global trail" τ_{ij} according to

$$\tau_{ij} := (1-\rho)\tau_{ij} + \rho/T^*,$$

if job j is placed in position i of the best schedule found. Otherwise,

$$\tau_{ij} := (1-\rho)\tau_{ij},$$

where $\rho \in [0,1]$ is a parameter of the algorithm, and T^* is the total tardiness of the best found schedule. After each iteration (1 ant = 1 iteration), we invoke the 2-opt strategy.

It is easy to show that the running time of Algorithm H without local search is $O(mn^2)$. For each j^* (there are a total n jobs), position k is chosen in $O(n)$ time. Local search has a running time of $O(n^3)$, but the number of times it is invoked is unknown. So Algorithm H has a running time no less than $O(mn^3)$. In other words, the running times of Algorithms H and ACO are comparable.

4 Computational results for instances of Potts and van Wassenhove

In this section we present the computational results of applying ACO and Algorithm H to deal with instances of the problem $1||\sum T_j$ comprising $n = 4, \ldots, 70, 100$ jobs that are generated using the schema given in [5].

The instances were generated as follows: for each job j, a processing time $p_j \in Z$ was randomly chosen from the uniform distribution $[1, 100]$, and a due date from the uniform distribution

$$[\sum_{j=1}^{n} p_j(1 - TF - RDD/2), \sum_{j=1}^{n} p_j(1 - TF + RDD/2)],$$

where TF is the tightness factor and RDD is the relative due date. Both of the values TF and RDD were taken from the set $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. For each combination of (TF, RDD), we generated 100 instances, i.e., a total of 2,500 instances were generated for each n.

We did not consider trivial instances where $F(\pi_{EDD}) = 0$. We used the following parameter settings: $\alpha = 1$, $\beta = 2$, and $\rho = 0.1$. For the heuristic information η_{ij}, we used the MDD rule.

For each instance, the exact Algorithm A returned an optimal value F_{opt}. In applying ACO, ants were allowed to continue to run when the optimal solution was not found. The number of ants was constrained by $m \leq 100$. ACO could run up to 10 times for each instance when the optimal solution was not obtained. The best total tardiness value F_{ACO} found was recorded, and the relative error $\frac{F_{ACO}-F_{opt}}{F_{opt}}$ was computed. The same experimental approach was taken to test Algorithm H.

In this way, we obtained computational results to compare the performance of ACO and Algorithm H with respect to the following measures: percentage of time that an optimal solution is found, relative error of the solution found, and number of iterations needed to find an optimal solution. The results are presented in Table 1.

The first column records the number of jobs n. The second and third columns show the number of instances for which ACO and Algorithm H could not find an optimal solution, respectively. The relative errors of ACO and Algorithm H are shown in columns 4 and 5, respectively, while the average numbers of iterations needed to solve the instances by ACO and Algorithm H are shown in the last two columns, respectively.

For $n = 100$, we considered 617 instances, whereas for each n, $4 \leq n \leq 70$, we considered 2,500 instances. In the table only rows for which the number of non-optimal solutions greater than zero are shown.

The results show that both ACO and Algorithm H could produce an optimal solution for more than 99% of the instances. Algorithm H could not find an optimal solution for less than 0.44% of the total number of instances considered, and its relative error was less than 0.46%. On the other hand, the relative error of ACO was up to 1.26% for $n = 61$, and the number of instances for which ACO could not optimally solve was greater than 1% of the instances considered for $n = 70$. We thus expect that the superiority of the performance of Algorithm H over ACO will become more significant as n grows.

5 Computational results for instances of case B-1

In this section we present the computational results of applying ACO and Algorithm H to tackle instances of a special case B-1 of the problem $1||\sum T_j$.

For this case, we have

$$\begin{cases} p_1 \geq p_2 \geq \ldots \geq p_n, \\ d_1 \leq d_2 \leq \ldots \leq d_n, \\ d_n - d_1 \leq p_n. \end{cases} \quad (1)$$

It was reported in [11] that this case is the "hardest" for Algorithm A, i.e., it requires frequent execution of Elimination Rules 1-4. Instances of this case were also called "hard" instances in [6]. This case has been shown to be NP-hard in the ordinary sense [12]. It has been shown that the exact algorithms proposed in [4, 8, 9, 10] for this case each have a running time of $O(2^{\frac{n}{2}})$.

We tested instances with $n = 4, \ldots, 100$ jobs. For each n, we considered 1,000 instances of case B-1. The values of p_j were randomly sampled from the uniform distribution $[1, 500]$, while the due dates d_j were randomly chosen from the uniform distribution $[X, X + p_n]$, where $X \in [0, \sum p_j - p_n]$.

The experimental approach discussed in Section 4 was applied to treat the instances in this section. We used the exact Algorithm B-1 [8] modified for integer instances to obtain the optimal solutions, which has a running time of $O(n \sum p_j)$. The results are presented in Table 2. In the table only rows for which the number of non-optimal solutions is greater than zero and the relative error is greater than 0.01% are shown.

The results show that ACO found the optimal solutions for all of the instances considered, except for $n = 7$, while Algorithm H found the optimal solutions for 99% of the instances. However, the relative error of Algorithm H was no larger than 0.01%. In general, both algorithms required fewer than 3 ants (iterations) to produce the optimal solutions. Therefore, we may conclude that the performance of Algorithm H is only marginally inferior to that of ACO.

6 Computational results for canonical DL-instances

In this section we consider another NP-hard case, known as the canonical DL-instances [1], of the problem $1||\sum T_j$. It has also been shown that the exact algorithms presented in [4, 8, 9] for the canonical DL-instances each have a running time of $O(2^{\frac{n}{2}})$.

First, consider the Even-Odd Partition (EOP) problem: Given a set of $2n^*$ positive integers $B = \{b_1, b_2, \ldots, b_{2n^*}\}$, where $b_i \geq b_{i+1}$, $i = 1, 2, \ldots, 2n^* - 1$, is there a partition of B into two subsets B_1 and B_2 such that $\sum_{b_i \in B_1} b_i = \sum_{b_i \in B_2} b_i$, and such that for each $i = 1, \ldots, n$, B_1 (and hence B_2) contains exactly one number of $\{b_{2i-1}, b_{2i}\}$?

We generated instances of EOP for $n^* = 4, \ldots, 40$. Let $\delta_i = b_{2i-1} - b_{2i}$, $i = 1, \ldots, n^*$. The values of δ_i were randomly chosen from the uniform distribution $[1, 50]$. For each n^* and each set of δ_i values generated, we constructed an instance of EOP as follows: $b_{2n^*} := 1$, $b_{2n^*-1} := b_{2n^*} + \delta_n^*$, $b_{2i} := b_{2i+1} + 1$, $b_{2i-1} := b_{2i} + \delta_i$, $i := 1, \ldots, n^* - 1$.

We then converted the EOP instance to a canonical DL-instance for each n^*, with the job set $N = \{V_1, V_2 \ldots, V_{2n^*}, W_1, W_2, \ldots, W_{n^*+1}\}$, where $|N| = 3n^* + 1$. Let $b = (4n^* + 1)\delta$. Denote $\delta = \frac{1}{2}\sum_{i=1}^{n^*}(b_{2i-1} - b_{2i})$. Let $a_{2i-1} = b_{2i-1} + (9n^{*2} + 3n^* - i + 1)\delta + 5n^*(b_1 - b_{2n^*})$ and $a_{2i} = b_{2i} + (9n^{*2} + 3n - i + 1)\delta + 5n(b_1 - b_{2n^*})$, $i = 1, \ldots, n^*$. Define the job due dates and processing times as follows:

$$p_{V_i} = a_i, \qquad i = 1, 2, \ldots, 2n^*;$$

$$p_{W_i} = b, \qquad i = 1, 2, \ldots, n^* + 1;$$

$$d_{V_i} = \begin{cases} (j-1)b + \delta + (a_2 + a_4 + \ldots + a_{2i}), & \text{if } i = 2j - 1, \\ d_{V_{2j-1}} + 2(n^* - j + 1)(a_{2j-1} - a_{2j}), & \text{if } i = 2j, j = 1, 2, \ldots, n^*; \end{cases}$$

$$d_{W_i} = \begin{cases} ib + (a_2 + a_4 + \ldots + a_{2i}), & \text{if } i = 1, 2, \ldots, n^*, \\ d_{W_{n^*}} + \delta + b, & \text{if } i = n^* + 1. \end{cases}$$

For this case we used the exact pseudo-polynomial Algorithm B-1 canonical [12] for canonical-DL instances to obtain the optimal solutions for the instances considered, which has a running time of $O(n^*\delta)$. The experimental settings followed those discussed in Section 4. For each n^*, where $n^* = 4, \ldots, 40$, the number of jobs was $n = 3n^* + 1 = 13, 16, \ldots, 121$, and we considered 50 instances. Both Algorithm H and ACO were applied to deal with the instances. The results are presented in Table 3. In the table only rows for which the number of non-optimal solutions is greater than zero are shown.

The performance of both algorithms was largely comparable. It is noted that when $3n^* + 1 = 25$ or 28, the number of instances for which the algorithms could not find an optimal solution was greater than 10% of the

instances tested. But the relative errors were all less than 0.01%, and the number of iterations required to obtain the optimal solutions were all fewer than 20.

It can be assumed that the chance of finding an optimal canonical DL-schedule is approximately $O(1/2^{n^*})$ [1]. This is because for each pair of V_{2i-1} and V_{2i}, $i = n^*, \ldots, 1$, there exist two orders with almost identical probabilities: V_{2i-1} is processed in position $2i-1$ and V_{2i} in position $3n^* + 1 - (i-1)$ of an optimal schedule, and vice versa.

We repeated the experiments without the 2-opt strategy for both algorithms. The results are shown in Table 4. In the table only rows for which the number of non-optimal solutions is greater than zero are shown.

The results show that both algorithms achieved a "good" performance only with the aid of local search. But the number of local search executed may be exponential. For $3n^* + 1 \geq 40$, none of the solutions obtained by both algorithms was optimal.

Conclusions

Our computational results show that Algorithm H performs better than ACO for the instances generated by the schema of [5]. For 99.5% of the instances considered for this case, Algorithm H found the optimal solutions. The relative error was less than 0.5%, and the average number of iterations needed was fewer than 5 (i.e., 5 ants).

For the "hard" instances of case B-1, Algorithm H performs marginally inferior to ACO. But Algorithm H found the optimal solutions for 99% of the instances considered, and its relative error was no larger than 0.01%.

For the NP-hard case of [1], both ACO and Algorithm H perform comparably and could achieve a "good" performance only with the aid of local search.

Acknowledgments *We are grateful to the Editor and the anonymous referees for their constructive comments on earlier versions of this paper. This research is part of a project funded by the Russian Science Support Foundation.*

References

[1] J. Du and J.Y.-T. Leung. Minimizing total tardiness on one processor is NP-hard. Mathematics of Operations Research 1990; 15: 483–495.

[2] E.L. Lawler. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics 1977; 1: 331–342.

[3] W. Szwarc, F. Della Croce and A. Grosso. Solution of the single machine total tardiness problem. Journal of Scheduling 1999; 2: 55–71.

[4] W. Szwarc, A. Grosso and F. Della Croce. Algorithmic paradoxes of the single machine total tardiness problem. Journal of Scheduling 2001; 4: 93-104.

[5] C.N. Potts and L.N. van Wassenhove. A decomposition algorithm for the single machine total tardiness problem. Operations Research Letters 1982; 1: 177–182.

[6] F. Della Croce, A. Grosso and V. Paschos. Lower bounds on the approximation ratios of leading heuristics for the single-machine total tardiness problem. Journal of Scheduling 2004; 7: 85–91.

[7] A. Bauer, B. Bullnheimer, R.F. Hartl and C. Strauss. Minimizing total tardiness on a single machine using Ant Colony Optimization. Proceedings of the 1999 Congress on Evolutionary Computation (CEC99), Washington, D.C., USA, 6-9 July, 1999; 1445–1450.

[8] A.A. Lazarev. Solution of the NP-hard total tardiness minimization problem in scheduling theory. Computational Mathematics and Mathematical Physics 2007; 47: 1039-1049.

[9] S. Chang, Q. Lu, G. Tang and W. Yu. On decomposition of total tardiness problem. Operation Research Letters 1995; 17: 221–229.

[10] A.A. Lazarev and E.R. Gafarov. Scheduling Theory: The Total Tardiness Problem. Computing Centre of the Russian Academy of Sciences, Russia, 2006 (in Russian).

[11] A.A. Lazarev, A. Kvaratskhelia and A. Tchernykh. Solution algorithms for the total tardiness scheduling problem on a single machine. Workshop Proceedings of the ENC'04 International Conference 2004; 474–480.

[12] E.R. Gafarov and A.A. Lazarev. A special case of the single-machine total tardiness problem is NP-hard. Journal of Computer and Systems Sciences International 2006; 45: 450–458.

Table 1. Computational results for instances of Potts and van Wassenhove

n	not opt. ACO	not opt $H.$	rel. ACO	rel. $H.$	Ants ACO	Ants $H.$
19	2	0	0.22	0	1.6164	1.4004
20	1	0	0.58	0	1.6064	1.4204
22	1	0	0.16	0	1.626	1.4844
28	2	0	0.09	0	1.9704	1.6688
34	1	0	0.15	0	2.2212	1.9568
36	0	1	0	0.04	2.2332	2.154
37	1	1	0.38	0.01	2.4796	2.102
40	1	0	0.04	0	2.6036	2.2424
42	1	1	0.05	0.01	2.7888	2.4092
43	1	1	0.07	0.06	2.7316	2.3656
44	3	0	0.04	0	2.8464	2.3784
45	2	0	0.68	0	2.9736	2.4728
46	1	0	0.03	0	3.1624	2.4088
47	2	0	0.01	0	3.248	2.5152
48	9	0	0.56	0	3.4516	2.5196
49	3	1	0.15	0.08	3.4252	2.7
50	9	1	0.35	0.29	3.716	2.6336
51	8	0	0.22	0	3.8412	2.7768
52	4	1	0.04	0.07	3.5816	2.86
53	4	2	0.03	0,42	3.8948	2.9668
54	9	3	0.1	0.29	4,0324	2,9924
55	8	2	0.11	0.06	4.1048	3.0496
56	9	1	0.83	0.01	4,2916	3.0064
57	7	0	0.23	0	4.1568	3.158
58	14	0	0.17	0	4.71	3.3724
59	14	4	0.24	0.1	4.81	3.3372
60	11	1	0.22	0.01	4.7268	3.4224
61	18	2	1.26	0.02	5.3032	3.5216
62	10	2	0.26	0.01	5.0964	3.5032
63	17	7	0.16	0.08	5.3016	3.5728
64	15	6	0.57	0.46	5.2388	3.6504
65	18	7	0.1	0.14	5.548	3.6604
100	36	0	0.31	0	27.35	4.66

Table 2. Computational results for instances of case B-1

n	not opt. ACO	not opt $H.$	rel. ACO	rel. $H.$	Ants ACO	Ants $H.$
7	1	0	0.67	0	1.38	1.044
26	0	3	0	0.01	1.381	1.871
27	0	1	0	0.01	1.429	1.707
31	0	4	0	0.01	1.354	1.929

Table 3. Computational results for canonical DL-instances

n	not opt. ACO	not opt $H.$	rel. ACO	rel. $H.$	Ants ACO	Ants $H.$
16	1	0	<0.01	0	4.92	3.4
19	2	3	<0.01	<0.01	10.3	11.64
22	2	2	<0.01	<0.01	7.66	8.22
25	4	4	<0.01	<0.01	16.28	16.44
28	6	4	<0.01	<0.01	19.2	15.24
31	0	3	0	<0.01	8.16	15.66
34	1	1	<0.01	<0.01	8.8	9.44
37	1	0	<0.01	0	7.12	7.58

Table 4. Computational results for canonical DL-instances (without local search)

n	not opt. ACO	not opt $H.$	rel. ACO	rel. $H.$	Ants ACO	Ants $H.$
13	50	26	13.46	0.01	100	58.2
16	50	35	0.77	0.03	100	73.82
19	50	43	3.29	2.78	100	88.06
22	50	46	2.86	2.05	100	93.52
25	50	49	2.03	1.58	100	98.02
28	50	50	2.97	2.49	100	100
31	50	49	3.48	2.02	100	98.48
34	50	49	2.85	1.67	100	98.4
37	50	49	1.71	1.41	100	98.02