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, 119991 Moscow GSP-1, Russia. Tel, fax: +007 495 1356238. Email: Alexandr.Lazarev@ksu.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, 119991 Moscow GSP-1, Russia. Tel, fax: +007 495 1356238. Email: axel73@mail.ru

Abstract

We propose a hybrid algorithm to deal with the NP-hard single-machine scheduling problem to minimize the total job tardiness based on the Ant Colony Optimization (ACO) meta-heuristic, in conjunction with four well-known elimination rules for the problem. The hybrid algorithm has the same run time as that of ACO. We conduct 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, Metaheuristics

Introduction

We are given a set of n independent jobs N 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 as $1||\sum T_j$, which has been shown to be NP-hard in the ordinary sense (Du, Leung, 1990). Lawler (Lawler, 1977) presented an $O(n^4 \sum p_j)$ time dynamic programming algorithm for the problem. Szwarc et al. (Szwarc et al., 2001) constructed state-of-the-art algorithms to handle the special instances of the problem discussed in (Potts, Wassenhove, 1982) for $n \leq 500$. It was shown in (Croce et al., 2004) that all known constructive and decomposition heuristics for non-paradoxical instances of the problem can yield arbitrarily bad approximation ratios (Szwarc et al., 2001).

In this paper we propose a hybrid algorithm based on the Ant Colony Optimization (ACO) meta-heuristic by Bauer et al. (Bauer et al., 1999), in conjunction with the four well-known Elimination Rules 1-4 for $1||\sum T_j$ introduced in (Szwarc et al.,1999; Lazarev et al. 2004; Chang et al.,1995). We conduct 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. We test both algorithms under three cases of $1||\sum T_j$, namely the special instances of Potts and Van Wassenhove (Potts, Wassenhove, 1982), the case B-1 of Lazarev et al. (Lazarev et al. 2004), and the canonical instances of Du and Leung (Du, Leung, 1990).

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 $d_1 \leq d_2 \leq \ldots \leq d_n$, and 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 increasing order of their processing times.

We denote by $I = \langle \{p_j, d_j\}_{j \in N}, t_0 \rangle$ an instance with the job set N, in which the jobs have processing times p_j, due dates d_j, and a starting time

t_0. Let j^* denote the job with the largest processing time in N, i.e.,

$$j^* = \arg\max_{j \in N} \{d_j : p_j = \max_{i \in N} p_i\}$$

.

We consider a subset of jobs $N' \subseteq N$ that must be processed from time $t' \geq t_0$. Let $N' = \{1, 2, \ldots, n'\}$. Define the set $L(N', t')$, i.e., a position list, of all the indices $k \geq j^*$ such that

(a) $t' + \sum_{j=1}^{k} p_j < d_{k+1}$ (Elimination Rule 1 (Szwarc et al.,2001; Lazarev et al. 2004)), and

(b) $d_j + p_j \leq t' + \sum_{j=1}^{k} p_j$, for all $j = \overline{j^*(N') + 1, k}$
(Elimination Rules 2 and 3 (Szwarc et al.,2001; Lazarev et al. 2004)),

where $d_{n'+1} := +\infty$.

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

Proposition 1 (Lazarev et al. 2004) *For all instances $\langle \{p_j, d_j\}_{j \in N}, t_0 \rangle$, the set $L(N, t_0)$ is not empty.*

Proposition 2 (Lazarev et al. 2004) *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 $k \in L(N, t_0)$.*

Let $N = (j_1, \ldots, j_n)$, 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.

Proposition 3 (Elimination Rule 4) (Szwarc et al.,1999; Chang et al.,1995) *Delete k from the position list $L(N', t')$, if $|L(N', t')| > 1$, and $(F(\pi^k) > F(\pi^{k+1})$ or $F(\pi^k) \geq F(\pi^i)$ for some $j^* \leq i < k)$.*

Now we present Algorithm A, an exact solution algorithm for $1||\sum T_j$, which is based on Elimination Rules 1-4.

Procedure ProcL (N, t)

0. There exists 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 $j^* \in N$;

3. Construct the set $L(N, t)$ for job j^*;

4. FOR ALL $k \in L(N, t)$ **DO:**

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

5. π^*:=arg $\min\limits_{k \in L(N,t)} \{F(\pi_k, t)\}$;

6. RETURN π^*.

Algorithm A

π^*:=**ProcL**(N, t_0).

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

We present the ACO algorithm of Bauer et al. (Bauer et al., 1999) 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 future pheromone values. 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 (Bauer et al., 1999), where the instances of (Potts, Wassenhove, 1982) for $n = 50$ and 100 were tested. For $n = 50$, ACO generated an optimal solution for 609 out of 625 instances. The relative error was less than 0.08%. For $n = 100$, all 125 instances were solved optimally.

It is easy to show that the run 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 run time of $O(n^3)$, but the number of times local search is applied is unknown. So ACO has a run time no less than $O(mn^3)$. In practice, the run 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., 1999), in conjunction with Elimination Rules 1-4 (Szwarc et al.,1999; Lazarev et al. 2004; Chang et al.,1995).

In Algorithm H, each ant 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)

0. There exists 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 $j^* \in N$;

3. Construct the set $L(N,t)$ for job j^*;

4. 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 $\pi^i = (j_1, \ldots, j_{m-1}, j_{m+1}, \ldots, j_i, j^*, j_{i+1}, \ldots, j_n)$, $j^* = j_m, m < i$;

5. Chose $k \in L(N,t)$ randomly according to probability ρ_{kj*};

6. 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};

7. 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, 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 found schedule. Else,

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

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, we invoke the 2-opt strategy.

It is easy to show that the run 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 run time of $O(n^3)$, but the number of times it is invoked is unknown. So Algorithm H has a run time no less than $O(mn^3)$. In other words, the run times of Algorithm H and ACO are comparable.

4 Computational results for instances of (Potts, Wassenhove, 1982)

In this section we present 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 in (Potts, Wassenhove, 1982).

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.

When $F(\pi_{EDD}) = 0$, we did not consider any instance because in this case both Algorithm H and ACO would yield the optimal solutions.

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 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 found total tardiness value F_{ACO} 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.

Table 1.

n	Instances	not opt. ACO	not opt H.	rel. ACO	rel. H.	Ants ACO	Ants H.
4	2500	0	0	0	0	1.0404	1.004
5	2500	0	0	0	0	1.0616	1.0216
6	2500	0	0	0	0	1.0908	1.036
7	2500	0	0	0	0	1.1384	1.0612
8	2500	0	0	0	0	1.1968	1.0588
9	2500	0	0	0	0	1.1456	1.1044
10	2500	0	0	0	0	1.2576	1.1228
11	2500	0	0	0	0	1.2364	1.126
12	2500	0	0	0	0	1.2672	1.1484
13	2500	0	0	0	0	1.2968	1.2296
14	2500	0	0	0	0	1.3704	1.2464
15	2500	0	0	0	0	1.3576	1.2744
16	2500	0	0	0	0	1.4324	1.3928
17	2500	0	0	0	0	1.4376	1.3196
18	2500	0	0	0	0	1.4656	1.3216
19	2500	2	0	0.22	0	1.6164	1.4004
20	2500	1	0	0.58	0	1.6064	1.4204

Continuation of the Table 1.

n	Instances	not opt. ACO	not opt H.	rel. ACO	rel. H.	Ants ACO	Ants H.
21	2500	0	0	0	0	1.5892	1.4232
22	2500	1	0	0.16	0	1.626	1.4844
23	2500	0	0	0	0	1.6572	1.5184
24	2500	0	0	0	0	1.6992	1.5568
25	2500	0	0	0	0	1.8128	1.5504
26	2500	0	0	0	0	1.7736	1.584
27	2500	0	0	0	0	1.88	1.6828
28	2500	2	0	0.09	0	1.9704	1.6688
29	2500	0	0	0	0	1.9172	1.7872
30	2500	0	0	0	0	1.9744	1.7268
31	2500	0	0	0	0	1.9656	1.8656
32	2500	0	0	0	0	2.1688	1.8788
33	2500	0	0	0	0	2.214	1.844
34	2500	1	0	0.15	0	2.2212	1.9568
35	2500	0	0	0	0	2.3272	2.1152
36	2500	0	1	0	0.04	2.2332	2.154
37	2500	1	1	0.38	0.01	2.4796	2.102
38	2500	0	0	0	0	2.2696	2.1172
39	2500	0	0	0	0	2.576	2.1044
40	2500	1	0	0.04	0	2.6036	2.2424
41	2500	0	0	0	0	2.552	2.2704
42	2500	1	1	0.05	0.01	2.7888	2.4092
43	2500	1	1	0.07	0.06	2.7316	2.3656
44	2500	3	0	0.04	0	2.8464	2.3784
45	2500	2	0	0.68	0	2.9736	2.4728
46	2500	1	0	0.03	0	3.1624	2.4088
47	2500	2	0	0.01	0	3.248	2.5152
48	2500	9	0	0.56	0	3.4516	2.5196
49	2500	3	1	0.15	0.08	3.4252	2.7
50	2500	9	1	0.35	0.29	3.716	2.6336
51	2500	8	0	0.22	0	3.8412	2.7768
52	2500	4	1	0.04	0.07	3.5816	2.86
53	2500	4	2	0.03	0.42	3.8948	2.9668
54	2500	9	3	0.1	0.29	4.0324	2.9924

The termination of the Table 1.

n	Instances	not opt. ACO	not opt H.	rel. ACO	rel. H.	Ants ACO	Ants H.
55	2500	8	2	0.11	0.06	4.1048	3.0496
56	2500	9	1	0.83	0.01	4.2916	3.0064
57	2500	7	0	0.23	0	4.1568	3.158
58	2500	14	0	0.17	0	4.71	3.3724
59	2500	14	4	0.24	0.1	4.81	3.3372
60	2500	11	1	0.22	0.01	4.7268	3.4224
61	2500	18	2	1.26	0.02	5.3032	3.5216
62	2500	10	2	0.26	0.01	5.0964	3.5032
63	2500	17	7	0.16	0.08	5.3016	3.5728
64	2500	15	6	0.57	0.46	5.2388	3.6504
65	2500	18	7	0.1	0.14	5.548	3.6604
66	2500	17	11	0.15	0.14	5.4288	3.8552
67	2500	17	7	0.83	0.1	6.1068	4.1016
68	2500	25	4	0.2	0.08	6.3864	3.7252
69	2500	18	6	0.12	0.1	6.1912	4.0796
70	2500	33	4	0.23	0.05	6.974	3.8672
100	617	36	0	0.31	0	27.35	4.66

The first and second columns record the number of jobs n and the number of instances considered, respectively. The third and fourth 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 5 and 6, 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.

The results show that both ACO and Algorithm H could produce the optimal solutions 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 (Lazarev et al.,2004)

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

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

It was reported in (Lazarev et al.,2004) that this case is the "hardest" for Algorithm A, i.e., it requires frequent invocation of Elimination Rules 1-4. Instances of the case (1) were also called "hard" instances in (Croce et al.,2004). This case has been shown to be NP-hard in the ordinary sense (Lazarev,Gafarov,2006). It has been shown that the exact algorithms proposed in (Szwarc et al.,1999; Lazarev et al. 2004; Chang et al.,1995) for this case each have a run 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 $[A, A + p_n]$, where $A \in [0, \sum p_j - p_n]$.

The experimental approach discussed in section 4 was applied to treat the instances in this section. We used the exact Algorithm B-1 modified for integer instances to obtain the optimal solutions, which has a run time of $O(n \sum p_j)$. The results are presented in Table 2.

Table 2.

n	Instances	not opt. ACO	not opt H.	rel. ACO	rel. H.	Ants ACO	Ants H.
4	1000	0	0	0	0	1.038	1.005
5	1000	0	0	0	0	1.071	1.034
6	1000	0	0	0	0	1.159	1.073
7	1000	1	0	0.67	0	1.38	1.044
8	1000	0	0	0	0	1.311	1.058
9	1000	0	0	0	0	1.401	1.137
10	1000	0	0	0	0	1.418	1.088
11	1000	0	0	0	0	1.584	1.091
12	1000	0	0	0	0	1.478	1.247
13	1000	0	0	0	0	1.495	1.201
14	1000	0	0	0	0	1.441	1.207
15	1000	0	0	0	0	1.621	1.241
16	1000	0	0	0	0	1.501	1.279
17	1000	0	0	0	0	1.45	1.271
18	1000	0	0	0	0	1.526	1.805
19	1000	0	0	0	0	1.511	1.36
20	1000	0	0	0	0	1.448	1.252
21	1000	0	0	0	0	1.457	1.457
22	1000	0	0	0	0	1.448	1.373
23	1000	0	1	0	0	1.481	1.761
24	1000	0	0	0	0	1.446	1.644
25	1000	0	2	0	0	1.337	1.696
26	1000	0	3	0	0.01	1.381	1.871
27	1000	0	1	0	0.01	1.429	1.707
28	1000	0	1	0	0	1.532	1.8
29	1000	0	3	0	0	1.423	1.815
30	1000	0	2	0	0	1.311	2.027
31	1000	0	4	0	0.01	1.354	1.929
32	1000	0	2	0	0	1.343	1.98
33	1000	0	3	0	0	1.379	2.005
34	1000	0	2	0	0	1.166	1.764

Continuation of the Table 2.

n	Instances	not opt. ACO	not opt H.	rel. ACO	rel. H.	Ants ACO	Ants H.
35	1000	0	7	0	0	1.287	2.435
36	1000	0	4	0	0	1.288	1.894
37	1000	0	5	0	0	1.237	2.102
38	1000	0	5	0	0	1.266	2.027
39	1000	0	5	0	0	1.216	2.115
40	1000	0	4	0	0	1.187	2.043
41	1000	0	3	0	0	1.241	1.868
42	1000	0	6	0	0	1.212	2.162
43	1000	0	4	0	0	1.287	2.042
44	1000	0	4	0	0	1.335	1.861
45	1000	0	2	0	0	1.304	2.149
46	1000	0	3	0	0	1.239	1.895
47	1000	0	4	0	0	1.224	1.847
48	1000	0	5	0	0	1.251	2.298
49	1000	0	3	0	0	1.264	2.179
50	1000	0	1	0	0	1.168	1.712
51	1000	0	0	0	0	1.251	1.332
52	1000	0	5	0	0	1.22	1.82
53	1000	0	6	0	0	1.213	1.995
54	1000	0	2	0	0	1.189	1.59
55	1000	0	1	0	0	1.139	1.639
56	1000	0	5	0	0	1.107	2.075
57	1000	0	5	0	0	1.18	2.049
58	1000	0	4	0	0	1.208	2.175
59	1000	0	0	0	0	1.218	1.424
60	1000	0	5	0	0	1.114	2.076
61	1000	0	4	0	0	1.15	1.773
62	1000	0	6	0	0	1.123	2.154
63	1000	0	4	0	0	1.114	1.909
64	1000	0	0	0	0	1.137	1.207
65	1000	0	4	0	0	1.112	1.854
66	1000	0	4	0	0	1.237	1.798
67	1000	0	1	0	0	1.132	1.57
68	1000	0	1	0	0	1.098	1.412

The termination of the Table 2.

n	Instances	not opt. ACO	not opt H.	rel. ACO	rel. H.	Ants ACO	Ants H.
69	1000	0	4	0	0	1.12	1.912
70	1000	0	6	0	0	1.076	1.904
71	1000	0	5	0	0	1.105	1.907
72	1000	0	4	0	0	1.123	1.765
73	1000	0	3	0	0	1.084	1.589
74	1000	0	2	0	0	1.11	1.527
75	1000	0	4	0	0	1.122	1.656
76	1000	0	4	0	0	1.122	1.688
77	1000	0	0	0	0	1.177	1.382
78	1000	0	2	0	0	1.088	1.532
79	1000	0	6	0	0	1.122	2.114
80	1000	0	6	0	0	1.104	1.97
81	1000	0	3	0	0	1.103	1.553
82	1000	0	2	0	0	1.103	1.602
83	1000	0	2	0	0	1.18	1.653
84	1000	0	2	0	0	1.08	1.603
85	1000	0	3	0	0	1.111	1.555
86	1000	0	1	0	0	1.149	1.534
87	1000	0	0	0	0	1.11	1.415
88	1000	0	2	0	0	1.123	1.401
89	1000	0	1	0	0	1.087	1.484
90	1000	0	3	0	0	1.086	1.596
91	1000	0	4	0	0	1.083	1.76
92	1000	0	4	0	0	1.094	1.936
93	1000	0	2	0	0	1.097	1.519
94	1000	0	2	0	0	1.096	1.463
95	1000	0	2	0	0	1.093	1.459
96	1000	0	5	0	0	1.095	1.963
97	1000	0	0	0	0	1.079	1.134
98	1000	0	0	0	0	1.125	1.238
99	1000	0	1	0	0	1.073	1.275
100	1000	0	1	0	0	1.068	1.525

In the table there are cases where the number of instances for which an algorithm could not find an optimal solution is not equal to 0, yet the corresponding relative error is 0, e.g., $n = 23$. This is because the relative errors obtained for such cases were very small, which became 0 on conversion into percentages.

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%. Overall, both algorithms required fewer than 3 ants 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 (Du,Leung,1990)

In this section we consider another NP-hard case, known as the canonical DL-instances (Du,Leung,1990), of the problem $1||\sum T_j$. It has also been shown that the exact algorithms presented in (Szwarc et al.,1999; Lazarev et al. 2004; Chang et al.,1995) for the canonical DL-instances each have a run 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 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 for canonical-DL instances to obtain the optimal solutions for the instances considered, which has a run 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 $3n + 1 = 13, 16, \ldots, 121$, and we considered 50 instances. Both Algorithm H and ACO were applied to deal with the instances. The results are presented in Table 3.

Table 3.

n	Instances	not opt. ACO	not opt H.	rel. ACO	rel. H.	Ants ACO	Ants H.
13	50	0	0	0	0	1.96	1.94
16	50	1	0	0	0	4.92	3.4
19	50	2	3	0	0	10.3	11.64
22	50	2	2	0	0	7.66	8.22
25	50	4	4	0	0	16.28	16.44
28	50	6	4	0	0	19.2	15.24
31	50	0	3	0	0	8.16	15.66
34	50	1	1	0	0	8.8	9.44
37	50	1	0	0	0	7.12	7.58
40	50	0	0	0	0	5.88	4.74
43	50	0	0	0	0	4.14	5.76
46	50	0	0	0	0	3.32	4.48
49	50	0	0	0	0	4.52	4.76
52	50	0	0	0	0	3.28	4.48
55	50	0	0	0	0	3.36	4.26
58	50	0	0	0	0	3.82	4.58
61	50	0	0	0	0	3.04	4.36
64	50	0	0	0	0	3.48	3.46

The termination of the Table 3.

n	Instances	not opt. ACO	not opt H.	rel. ACO	rel. H.	Ants ACO	Ants H.
67	50	0	0	0	0	3.22	3.48
70	50	0	0	0	0	2.26	3.1
73	50	0	0	0	0	2.46	3.36
76	50	0	0	0	0	2.96	3.22
79	50	0	0	0	0	2.22	2.52
82	50	0	0	0	0	2.94	3.24
85	50	0	0	0	0	3.34	3.72
88	50	0	0	0	0	2.8	3.56
91	50	0	0	0	0	2.64	2.6
94	50	0	0	0	0	2.7	2.8
97	50	0	0	0	0	2.7	2.9
100	50	0	0	0	0	2.46	2.68
103	50	0	0	0	0	2.48	2.52
106	50	0	0	0	0	3.08	2.46
109	50	0	0	0	0	2.44	2.2
112	50	0	0	0	0	2.18	2.22
115	50	0	0	0	0	2.08	2.12
118	50	0	0	0	0	2.02	1.96
121	50	0	0	0	0	2.18	2.56

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 $O(1/2^n)$ (Du,Leung,1990). 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 the next Table 4.

Table 4.

n	Instances	not opt. ACO	not opt H.	rel. ACO	rel. H.	Ants ACO	Ants H.
13	50	50	26	13.46	0.01	100	58.2
16	50	50	35	0.77	0.03	100	73.82
19	50	50	43	3.29	2.78	100	88.06
22	50	50	46	2.86	2.05	100	93.52
25	50	50	49	2.03	1.58	100	98.02
28	50	50	50	2.97	2.49	100	100
31	50	50	49	3.48	2.02	100	98.48
34	50	50	49	2.85	1.67	100	98.4
37	50	50	49	1.71	1.41	100	98.02
40	50	50	50	0.96	1.79	100	100
43	50	50	50	2.3	2.06	100	100
46	50	50	50	1.16	2.24	100	100
49	50	50	50	2.18	2.36	100	100
52	50	50	50	1.61	1.75	100	100
55	50	50	50	1.42	1.87	100	100
58	50	50	50	1.08	1.4	100	100
61	50	50	50	1.22	1.51	100	100
64	50	50	50	1.37	1.6	100	100
67	50	50	50	2.41	1.66	100	100
70	50	50	50	1.82	1.71	100	100
73	50	50	50	1.52	1.57	100	100
76	50	50	50	1.71	1.61	100	100
79	50	50	50	2.47	1.49	100	100
82	50	50	50	2.44	1.65	100	100
85	50	50	50	2.02	2.69	100	100
88	50	50	50	1.93	2.03	100	100
91	50	50	50	2.79	1.79	100	100
94	50	50	50	2.28	1.46	100	100
97	50	50	50	1.95	1.37	100	100
100	50	50	50	2.21	1.75	100	100

The termination of the Table 4.

n	Instances	not opt. ACO	not opt H.	rel. ACO	rel. H.	Ants ACO	Ants H.
103	50	50	50	1.48	1.74	100	100
106	50	50	50	2.05	1.56	100	100
109	50	50	50	1.78	1.4	100	100
112	50	50	50	1.97	1.76	100	100
115	50	50	50	1.76	1.95	100	100
118	50	50	50	1.79	1.98	100	100
121	50	50	50	2.27	1.38	100	100

The results show that both algorithms achieved "good" performance only with the aid of local search. But the number of local search executed may be exponential. When $3n + 1 = 40$ or more, 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 (Potts, Wassenhove, 1982). 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 (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 (Du,Leung,1990), both ACO and Algorithm H perform comparably and could achieve good performance only with the aid of local search.

References

[1] J. Du, 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, A. Grosso. Solution of the single machine total tardiness problem. Journal of Scheduling 1999; 2; 55–71.

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

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

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

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

[10] D. Merkle, M. Middendorf. An ant algorithm with a new pheromone evaluation rule for total tardiness problem. Lecture Notes in Computer Science 2000; 1803; 287–296.

[11] H. Emmons. One machine sequencing to minimize certain functions of job tardiness. Operations Research 1969; 17 701–715.

[12] A.A. Lazarev, E.R. Gafarov. Special case of the single machine total tardiness problem is NP-hard. Journal of Computer and Systems Sciences International 2006; 3; 120–128 (in Russian).