Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 34 (2007) 1899–1909
computers & operations research
www.elsevier.com/locate/cor

An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups

Ching-Jong Liao*, Hsiao-Chien Juan

Department of Industrial Management, National Taiwan University of Science and Technology, 43 Keelung Road, Section 4, Taipei, 10607 Taiwan

Available online 30 August 2005

Abstract

In many real-world production systems, it requires an explicit consideration of sequence-dependent setup times when scheduling jobs. As for the scheduling criterion, the weighted tardiness is always regarded as one of the most important criteria in practical systems. While the importance of the weighted tardiness problem with sequence-dependent setup times has been recognized, the problem has received little attention in the scheduling literature. In this paper, we present an ant colony optimization (ACO) algorithm for such a problem in a single-machine environment. The proposed ACO algorithm has several features, including introducing a new parameter for the initial pheromone trail and adjusting the timing of applying local search, among others. The proposed algorithm is experimented on the benchmark problem instances and shows its advantage over existing algorithms. As a further investigation, the algorithm is applied to the unweighted version of the problem. Experimental results show that it is very competitive with the existing best-performing algorithms.

Keywords: Scheduling; Ant colony optimization; Weighted tardiness; Sequence-dependent setups

1. Introduction

The operations scheduling problems have been studied for over five decades. Most of these studies either ignored setup times or assumed them to be independent of job sequence [1]. However, an explicit consideration of sequence-dependent setup times (SDST) is usually required in many practical industrial

* Corresponding author. Fax: +886 2 27376344.
E-mail address: cjl@im.ntust.edu.tw (C.-J. Liao).

0305-0548/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2005.07.020

situations, such as in the printing, plastics, aluminum, textile, and chemical industries [2,3]. As Wortman [4] indicates, the inadequate treatment on SDST will hinder the competitive advantage.

On the other hand, a survey of US manufacturing practices indicates that meeting due dates is the single most important scheduling criterion [5]. Among the due-date criteria, the weighted tardiness is the most flexible one as it can be used to differentiate between customers.

While the importance of the weighted tardiness problem with SDST has been recognized, the problem has received little attention in the scheduling literature, mainly because of its complexity. This inspires us to develop a heuristic to obtain a near-optimal solution for this practical problem in the single-machine environment. It is noted that the single-machine problem does not necessarily involve only one machine; a complicated machine environment with a single bottleneck may be treated as a single machine problem.

We now give a formal description of the problem. We have n jobs which are all available for processing at time zero on a continuously available single machine. The machine can process only one job at a time. Associated with each job j is the required processing time (p_j), due date (d_j), and weight (w_j). In addition, there is a setup time (s_{ij}) incurred when job j follows job i immediately in the processing sequence. Let Q be a sequence of the jobs, $Q = [Q(0), Q(1), \ldots, Q(n)]$, where $Q(k)$ is the index of the kth job in the sequence and $Q(0)$ is a dummy job representing the starting setup of the machine. The completion time of $Q(k)$ is $C_{Q(k)} = \sum_{l=1}^{k} \{s_{Q(l-1)Q(l)} + p_{Q(l)}\}$, the tardiness of $Q(k)$ is $T_{Q(k)} = \max\{C_{Q(k)} - d_{Q(k)}, 0\}$, and the (total) weighted tardiness for sequence Q is $WT_Q = \sum_{k=1}^{n} w_{Q(k)} T_{Q(k)}$. The objective of the problem is to find a sequence with minimum weighted tardiness of jobs. Using the three-field notation, this problem can be denoted by $1/s_{ij}/\sum w_j T_j$ and its unweighted version by $1/s_{ij}/\sum T_j$.

Lawler et al. [6] show that the $1//\sum w_j T_j$ problem is strongly NP-hard. Since the incorporation of setup times complicates the problem, the $1/s_{ij}/\sum w_j T_j$ problem is also strongly NP-hard. The unweighted version $1/s_{ij}/\sum T_j$ is strongly NP-hard because $1/s_{ij}/C_{\max}$ is strongly NP-hard [7, p. 79] and $C_{\max}$ reduces to $\sum T_j$ in the complexity hierarchy of objective functions [7, p. 27]. For such problems, there is a need to develop heuristics for obtaining a near-optimal solution within a reasonable computation time.

Scheduling heuristics can be broadly classified into two categories: the constructive type and the improvement type. In the literature, the best constructive-type heuristic for the $1/s_{ij}/\sum w_j T_j$ problem is apparent tardiness cost with setups (ATCS), proposed by Lee et al. [8]. Like other constructive-type heuristics, ATCS can derive a feasible solution quickly but the solution quality is usually unsatisfactory, especially for large-sized problems. On the other hand, the improvement-type heuristic can produce better solutions but with much more computational efforts. For the $1/s_{ij}/\sum w_j T_j$ problem, Cicirello [9] develops four different improvement-type heuristics, including limited discrepancy search (LDS), heuristic-biased stochastic sampling (HBSS), value-biased stochastic sampling (VBSS), and hill-climbing using VBSS (VBSS-HC), to obtain solutions for a set of 120 benchmark problem instances each with 60 jobs. Recently, an simulated annealing (SA) algorithm has been used to update 27 such instances in the benchmark library. To the best of our knowledge, the work of Cicirello [9] is the only research that develops improvement-type heuristics for the $1/s_{ij}/\sum w_j T_j$ problem. The importance of the problem in real-world production systems and its computational complexity justify us to challenge the problem using a recent metaheuristic, the ant colony optimization (ACO). On the other hand, there exist several heuristics of the improvement type for the unweighted problem $1/s_{ij}/\sum T_j$ [10–12].

The remainder of this paper is organized as follows. In Section 2, we briefly describe the background of ACO heuristic and then give the detailed steps of the proposed ACO algorithm. Computational experiments are conducted and reported in Section 3. Finally, conclusions are given in Section 4.

2. The proposed ACO algorithm

ACO is one of the metaheuristics for discrete optimization. It has been successfully applied to a large number of combinatorial optimization problems, including traveling salesman problems (e.g., [13], vehicle routing problems (e.g., [14]), and quadratic assignment problems (e.g., [15]), which have shown competitiveness with other metaheuristics. ACO has also been used successfully in solving scheduling problems on single machines (e.g., [16–18] and flow shops (e.g., [19,20]).

ACO is inspired by the foraging behavior of real ants, which can be described as follows [21]. Real ants have the ability to find the shortest path from a food source to their nest without using visual cues. Instead, they communicate information about the food source by producing a chemical substance, called pheromone, laid on their paths. Since the shorter paths have a higher traffic density, these paths can accumulate a higher amount of pheromone. Hence, the probability of ants following these shorter paths will be higher than the longer ones.

The ACO algorithm proposed in this paper is basically the ACS version [13], but it introduces the feature of a minimum pheromone value from "max-min-ant system" (MMAS) [22]. Other elements of MMAS are not applied because there are no significant effects for the studied problem. To make the algorithm more efficient and effective, we employ two distinctive features, along with three other elements used in some ACO algorithms. Before elaborating on these features, we present our ACO algorithm as follows:

Step 1: *Pheromone initialization*. Let the initial pheromone trail $\tau_0 = K/(nWT_{\mathrm{ATCS}})$, where K is a parameter, n is the problem size, and WT_{ATCS} is the weighted tardiness by applying the dispatching rule of ATCS (to be elaborated in Step 2.1).

Step 2: *Main loop*. In the main loop, each of the m ants constructs a sequence of n jobs. This loop is executed for Itemax = 1000 iterations or 50 consecutive iterations with no improvement, depending on which criterion is satisfied first. The latter criterion is used to save computation time for the case of premature convergence.

Step 2.1: *Constructing a job sequence by each ant*. A set of artificial ants is initially created. Each ant starts with an empty sequence and then successively appends an unscheduled job to the partial sequence until a feasible solution is constructed (i.e., all jobs are scheduled). In choosing the next job j to be appended at the current position i, the ant applies the following state transition rule:

$$j = \begin{cases} \arg\max\limits_{u \in U} \{[\tau_t(i,u)][\eta(i,u)]^{\beta}\} & \text{if } q \leqslant q_0, \\ S & \text{otherwise,} \end{cases}$$

where $\tau_t(i,u)$ is the pheromone trail associated with the assignment of job u to position i at time t, U is the set of unscheduled jobs, q is a random number uniformly distributed in [0, 1], and q_0 is a parameter ($0 \leqslant q_0 \leqslant 1$) which determines the relative importance of exploitation versus exploration. If $q \leqslant q_0$, the unscheduled job j with maximum value is placed at position i (exploitation); otherwise, a job is chosen according to S (biased exploration). The random variable S is selected according to the probability:

$$p(i,j) = \frac{[\tau_t(i,j)][\eta(i,j)]^{\beta}}{\sum_{u \in U}[\tau_t(i,u)][\eta(i,u)]^{\beta}}.$$

The parameter $\eta(i,j)$ is the heuristic desirability of assigning job j to position i, and β is the parameter which determines the relative importance of the heuristic information. In our algorithm, we use the dispatching rule of ATCS as the heuristic desirability $\eta(i,j)$. The ATCS rule combines the weighted

shortest processing time (WSPT) rule, the minimum slack (MS) rule, and the shortest setup time (SST) rule, each represented by a term, in a single ranking index [7]. This rule assigns jobs in non-increasing order of $I_j(t, v)$ (i.e., set $\eta(i, j) = I_j(t, v)$) given by

$$I_j(t, v) = \frac{w_j}{p_j} \exp\left[-\frac{\max(d_j - p_j - t, 0)}{k_1 \bar{p}}\right] \exp\left[-\frac{s_{vj}}{k_2 \bar{s}}\right],$$

where t denotes the current time, w_j, p_j, d_j are the weight, processing time, and due date of job j, respectively, v is the index of the job at position $i - 1$, $\bar{p}$ is the average processing time, $\bar{s}$ is the average setup time, k_1 is the due date-related parameter, and k_2 is the setup time-related scaling parameter.

Step 2.2: *Local update of pheromone trail*. To avoid premature convergence, a local trail update is performed. The update reduces the pheromone amount for adding a new job so as to discourage the following ants from choosing the same job to place at the same position. This is achieved by the following local updating rule:

$$\tau_t(i, j) = (1 - \rho)\tau_t(i, j) + \rho\tau_0,$$

where ρ $(0 < \rho \leqslant 1)$.

Step 2.3: *Local search*. The local search in our algorithm is a combination of the interchange (IT) and the insert neighborhood (IS). The IT considers exchanges of jobs placed at the *i*th and *j*th positions while the IS inserts the job from the *i*th position at the *j*th position. We use two of its variants, IT/IS and IS/IT, depending on which is implemented first. In our algorithm, the choice of IT or IS is determined randomly. Moreover, the local search is applied whenever a better sequence is found during an iteration and it is not executed for those iterations with no improvement.

Step 2.4: *Global update of pheromone trail*. The global updating rule is applied after each ant has completed a feasible solution (i.e., an iteration). Following the rule, the pheromone trail is added to the path of the incumbent global best solution, i.e., the best solution found so far. If job j is placed at position i in the global best solution during iteration t, then

$$\tau_{t+1}(i, j) = (1 - \alpha)\tau_t(i, j) + \alpha\Delta\tau_t(i, j),$$

where α $(0 < \alpha \leqslant 1)$ is a parameter representing the evaporation of pheromone. The amount $\Delta\tau_t(i, j) = 1/WT^*$, where WT^* is the weight tardiness of the global best solution. In order to prevent the solution from falling into a local optimum that results from the pheromone evaporating to zero, we introduce a lower bound to the pheromone trail value by letting $\tau_t(i, j) = (1/5)\tau_0$.

The above ACO algorithm has two distinctive features that makes it competitive with other algorithms:

1. Introducing a new parameter for the initial pheromone trail.
2. Adjusting the timing of applying local search.

We first discuss the impact of introducing a new parameter for the initial pheromone trail τ_0. To incorporate the heuristic information into the initial pheromone trail, most ACO algorithms set $\tau_0 = 1/(nL_H)$, where L_H is the objective value of a solution obtained either randomly or through some simple heuristic [13,17–19]. However, our experimental analyses show that this setting of τ_0 results in a premature convergence for $1/s_{ij}/\sum w_j T_j$ mainly because the value is too small. We thus introduce a new parameter K for τ_0, i.e., $\tau_0 = K/(nL_H)$. Detailed experimental results are given in Section 3.

As for the second feature, we note that in conventional ACO algorithms the local search is executed for the global best solution once within each iteration even for those with no improvement. In our algorithm, the local search is applied whenever a better solution is found during an iteration. Hence, the local search may be applied more than once or completely unused in an iteration. The computational experiments given in Section 3 show that our approach can save the computation time consistently as many as four times without deteriorating the solution quality. The main reason is that the local search in our approach has a higher probability of generating a better solution because the search is performed on a less unexplored space.

In addition to the two features, there are also some useful elements which have been used in other ACO algorithms being employed in our proposed algorithm. These elements include

1. A lower bound for the pheromone trail. Stützle and Hoos [22] develop the so-called MMAS which introduces upper and lower bounds to the values of pheromone trail. Based on our experiments, imposing a lower bound to the pheromone trail value improves the solution significantly, but there is no significant effect with an upper bound. Thus, only a lower bound is introduced in our algorithm.
2. A local search combining IT and IS. In our algorithm, the local search is a combination of IT and IS [17]. We use two of its variants, IT/IS and IS/IT, depending on which is implemented first. The choice of IT/IS or IS/IT is determined randomly in our algorithm.
3. The job-to-position definition of pheromone trail. In general, there are two ways to define the pheromone trail in scheduling: job-to-job [18] and job-to-position definitions. Based on our experimental analyses, the job-to-position definition is more efficient than the job-to-job one for the $1/s_{ij}/\sum w_j T_j$ problem and its unweighted version.

3. Computational experiments

To verify the performance of the algorithm, two sets of computational experiments were conducted: one was for $1/s_{ij}/\sum w_j T_j$ and the other was for its unweighted version $1/s_{ij}/\sum T_j$. The algorithm was coded in C++ and implemented on a Pentium IV 2.8 GHz PC.

In the first set of experiments (for $1/s_{ij}/\sum w_j T_j$), the proposed ACO was tested on the 120 benchmark problem instances provided by Cicirello [9], which can be obtained at http://www.ozone.ri.cmu.edu/benchmarks/bestknown.txt. The problem instances are characterized by three factors (i.e., due-date tightness δ, due-date range R, and setup time severity ζ) and generated by the following parameters: $\delta = \{0.3, 0.6, 0.9\}$, $R = \{0.25, 0.75\}$, and $\zeta = \{0.25, 0.75\}$. For each combination, 10 problem instances each with 60 jobs were generated. The best-known solutions to these instances were established by applying each of the following improvement-type heuristics: LDS, HBSS, VBSS, and VBSS-HC. According to the benchmark library, 27 such instances have been updated recently by a simulated annealing algorithm.

To determine the best values of parameters, a series of pilot experiments were conducted. The best values for our problem are as follows: Itemax $= 1000$, $m = 30$, $\alpha = 0.1$, $\beta = 0.5$, $\rho = 0.1$, $q_0 = 0.9$.

We now evaluate the impact of adding a new parameter K for the initial pheromone trail. To make a clear comparison, we temporarily remove the local search from the algorithm; all the other experiments in this paper were done with local search. Table 1 gives the computational results for 10 arbitrarily chosen Cicirello's instances where each one was run five times. Although there is no single best value of K for all the problem instances, good results can be obtained by setting $K = 20$ for the problem. It can be

Table 1
Impact of introducing a parameter $K = 20$ for the initial pheromone trail

Problem	Average			Best		
	$K = 1$	$K = 20$	% to $K = 1$	$K = 1$	$K = 20$	% to $K = 1$
71	179892	172487	−4.1	174341	164671	−5.5
72	71694	69761	−2.7	69787	69657	−0.2
73	47322	45809	−3.2	46772	43242	−7.5
74	61158	49032	−19.8	59211	47809	−19.3
75	43518	39251	−9.8	43484	37291	−14.2
76	97201	72494	−25.4	88887	68361	−23.1
77	61302	52809	−13.9	58902	51940	−11.8
78	37598	34675	−7.8	37309	30274	−18.9
79	146437	134360	−8.2	142718	132398	−7.2
80	62990	45816	−27.3	58601	40266	−31.3

Table 2
Effect of timing for applying the local search

Problem	Average		Best		Time (s)		
	Conv.	New	Conv.	New	Conv.	New	%
71	157328+	160022	150521	157382	120.25	30.99	25.8
72	58011	57669+	56364	56273+	122.62	32.11	26.2
73	35989+	36203	34932+	35108	121.31	31.45	25.9
74	37267	37012+	34508+	34964	121.52	31.80	26.2
75	34305	32013+	32990	29878+	118.66	31.42	26.5
76	68225	67936+	67084	65317+	126.05	33.02	26.2
77	40113+	40539	37247+	37896	121.89	33.14	27.2
78	28987	25998+	27308	25213+	123.52	31.84	25.8
79	126553	125293+	123905	123408+	125.92	32.59	25.9
80	28488+	29033	27401+	27796	130.30	34.30	26.3

Conv.: the conventional approach; New: the new approach used in our algorithm.

observed from Table 1 that adding a new parameter $K = 20$ can significantly improve the solutions. The experiments were rerun with local search and the same value ($K = 20$) was found to be suitable.

Another preliminary experiment was conducted to evaluate the timing of applying the local search. As noted earlier, in conventional ACO algorithms, the local search is executed once for every iteration but our algorithm applies the local search whenever a better solution is found. Table 2 gives the computational results for 10 arbitrarily chosen Cicirello's instances where each one was run five times. In the experiment, the only termination rule is set by letting Itemax = 1000. It is seen from the table that the two approaches result in a similar solution quality, but our approach requires only about 25% of the computation time as compared to the conventional approach.

Table 3
Comparison of the solutions from the proposed ACO_{LJ} with the best-known solutions

Prob.	Best-known	ACO_{LJ}	Prob.	Best-known	ACO_{LJ}	Prob.	Best-known	ACO_{LJ}	Prob.	Best-known	ACO_{LJ}	Prob.	Best-known	ACO_{LJ}
1	978	894^{+}	25	0	0	49	84143	81530^{+}	73	36465	34082^{+}	97	420287	419370^{+}
2	6489	6307^{+}	26	0	0	50	36235	35507^{+}	74	38292	33725^{+}	98	532519	533106^{-}
3	2348	2003^{+}	27	229	137^{+}	51	58574	55794^{+}	75	30980	27248^{+}	99	374781	370080^{+}
4	8311	8003^{+}	28	72	19^{+}	52	105367	105203^{+}	76	67553	66847^{+}	100	441888	441794^{+}
5	5606	5215^{+}	29	0	0	53	95452	96218^{-}	77	40558	37257^{+}	101	355822	355372^{+}
6	8244	5788^{+}	30	575	372^{+}	54	123558	124132^{-}	78	25105	24795^{+}	102	496131	495980^{+}
7	4347	4150^{+}	31	0	0	55	76368	74469^{+}	79	125824	122051^{+}	103	380170	379913^{+}
8	327	159^{+}	32	0	0	56	88420	87474^{+}	80	31844	26470^{+}	104	362008	360756^{+}
9	7598	7490^{+}	33	0	0	57	70414	67447^{+}	81	387148	387866^{-}	105	456364	454890^{+}
10	2451	2345^{+}	34	0	0	58	55522	52752^{+}	82	413488	413181^{+}	106	459925	459615^{+}
11	5263	5093^{+}	35	0	0	59	59060	56902^{+}	83	466070	464443^{+}	107	356645	354097^{+}
12	0	0	36	0	0	60	73328	72600^{+}	84	331659	330714^{+}	108	468111	466063^{+}
13	6147	5962^{+}	37	2407	2078^{+}	61	79884	80343^{-}	85	558556	562083^{-}	109	415817	414896^{+}
14	3941	4035^{-}	38	0	0	62	47860	46466^{+}	86	365783	365199^{+}	110	421282	421060^{+}
15	2915	2823^{+}	39	0	0	63	78822	78081^{+}	87	403016	401535^{+}	111	350723	347233^{+}
16	6711	6153^{+}	40	0	0	64	96378	95113^{+}	88	436855	436925^{-}	112	377418	373238^{+}
17	462	443^{+}	41	73176	73578^{-}	65	134881	132078^{+}	89	416916	412359^{+}	113	263200	262367^{+}
18	2514	2059^{+}	42	61859	60914^{+}	66	64054	63278^{+}	90	406939	404105^{+}	114	473197	470327^{+}
19	279	265^{+}	43	149990	149670^{+}	67	34899	32315^{+}	91	347175	345421^{+}	115	460225	459194^{+}
20	4193	4204^{-}	44	38726	37390^{+}	68	26404	26366^{+}	92	365779	365217^{+}	116	540231	527459^{+}
21	0	0	45	62760	62535^{+}	69	75414	64632^{+}	93	410462	412986^{-}	117	518579	512286^{+}
22	0	0	46	37992	38779^{-}	70	81200	81356^{-}	94	336299	335550^{+}	118	357575	352118^{+}
23	0	0	47	77189	76011^{+}	71	161233	156272^{+}	95	527909	526916^{+}	119	583947	584052^{-}
24	1791	1551^{+}	48	68920	68852^{+}	72	56934	54849^{+}	96	464403	461484^{+}	120	399700	398590^{+}

$^{+}$The proposed ACO_{LJ} is better.
$^{-}$The proposed ACO_{LJ} is worse.

We now present the formal experimental results for $1/s_{ij}/\sum w_j T_j$. Table 3 shows the comparison between the solutions from our ACO algorithm, denoted by ACO_{LJ} hereafter, and the best-known solutions to the benchmark instances. Our ACO_{LJ} was run 10 times and the best solution was selected. For the 104 instances with non-zero weighted tardiness, ACO_{LJ} has produced a better solution in 90 such instances (86%). For those with zero weighted tardiness, ACO_{LJ} has also obtined an optimal solution. The average computation time for each run is only 4.99 s with 16 instances taking more than 10 seconds. The comparison of computation times cannot be made because they are not provided in the benchmark library.

These favorable computational results encourage us to apply our ACO_{LJ} algorithm to the unweighted problem $1/s_{ij}/\sum T_i$. Our ACO_{LJ} can be applied to $1/s_{ij}/\sum T_i$ by simply setting all weights equal to 1. Thus, in the second set of the experiments, we compare ACO_{LJ} with three best-performing algorithms, RSPI, ACO_{GPJ}, and Tabu-VNS for $1/s_{ij}/\sum T_i$. The RSPI algorithm is a genetic algorithm proposed by Rubin and Ragatz [11], the ACO_{GPG} algorithm is an ACO algorithm developed by Gagné et al. [18], and the Tabu-VNS algorithm is a hybrid tabu search/variable neighborhood search algorithm developed by Gagnè et al. [12]. Although ACO_{GPG} is also an ACO algorithm, it is rather different from our proposed

Table 4
Comparison of the proposed ACO_{LJ} algorithm with two best-performing algorithms for the unweighted problem (small-and medium-sized problems)

Problem	RSPI	ACO_{GPG}	ACO_{LJ}
Prob401	90*	90*	90*
Prob402	0*	0*	0*
Prob403	3418*	3418*	3418*
Prob404	1067*	1067*	1067*
Prob405	0*	0*	0*
Prob406	0*	0*	0*
Prob407	1861*	1861*	1861*
Prob408	5660*	5660*	5660*
Prob501	266	261+	263
Prob502	0*	0*	0*
Prob503	3497	3497	3497
Prob504	0*	0*	0*
Prob505	0*	0*	0*
Prob506	0*	0*	0*
Prob507	7225+	7268	7225+
Prob508	1915+	1945	1915+
Prob601	36	16	14+
Prob602	0*	0*	0*
Prob603	17792	17685	17654+
Prob604	19238	19213	19092+
Prob605	273	247	240+
Prob606	0*	0*	0*
Prob607	13048	13088	13010+
Prob608	4733	4733	4732+
Prob701	118	103+	103+
Prob702	0*	0*	0*
Prob703	26745	26663	26568+
Prob704	15415	15495	15409+
Prob705	254	222	219+
Prob706	0*	0*	0*
Prob707	24218	24017	23931+
Prob708	23158	23351	23028+

*Indicates optimal solution.
+The best performance among the three algorithms (ties for all are not indicated).

ACO_{LJ}. Not only are the two features and three elements discussed above different (e.g., ACO_{GPG} uses a job-to-job pheromone definition), but ACO_{GPG} has its own feature (e.g., using look-ahead information in the transition rule).

To test the small-and medium-sized problems (with 15–45 jobs), we use the 32 test problem instances provided by Rubin and Ragatz [11] which can be obtained at http://mgt.bus.msu.edu/datafiles.htm. Table 4

Table 5
Comparison of the proposed ACO_{LJ} with two best-performing algorithms for the unweighted problem (large-sized problems)

Problem	ACO_{GPG}	Tabu-VNS	ACO_{LJ}
Prob551	212	185	183+
Prob552	0	0	0
Prob553	40828	40644+	40676
Prob554	15091	14711	14684+
Prob555	0	0	0
Prob556	0	0	0
Prob557	36489	35841+	36420
Prob558	20624	19872+	19888
Prob651	295	268+	268+
Prob652	0	0	0
Prob653	57779	57602	57584+
Prob654	34468	34466	34306+
Prob655	13	2+	7
Prob656	0	0	0
Prob657	56246	55080+	55389
Prob658	29308	27187+	27208
Prob751	263	241+	241+
Prob752	0	0	0
Prob753	78211	77739	77663+
Prob754	35826	35709	35630+
Prob755	0	0	0
Prob756	0	0	0
Prob757	61513	59763+	60108
Prob758	40277	38789	38704+
Prob851	453	384+	455
Prob852	0	0	0
Prob853	98540	97880+	98443
Prob854	80693	80122	79553+
Prob855	333	283+	324
Prob856	0	0	0
Prob857	89654	87244+	87504
Prob858	77919	75533	75506+

+The best performance among the three algorithms (ties for all not indicated).

shows the comparison among RSPI, ACO_{GPG}, and ACO_{LJ} (the result of Tabu-VNS is not available for the small-sized problems), where ACO_{GPG} gives the best solution from 20 runs and ACO_{LJ} gives the best solution from 10 runs. All the three algorithms find the optimal solutions for those 16 instances whose optimal solutions are known. For the remaining 16 instances, the three algorithms (RSPI, ACO_{GPG}, and ACO_{LJ}) find the best solutions for 2 (13%), 3 (19%), and 15 (94%) such instances, respectively; three instances end in a tie.

To test the large-sized problems (with 55–85 jobs), we use the test problem instances provided by Gagné et al. [18] and make the comparison among ACO_{GPG}, Tabu-VNS, and ACO_{LJ}. Both ACO_{GPG} and ACO_{LJ} give their best solutions from 20 runs. To make our ACO_{LJ} algorithm more efficient and effective, some of its parameters were fine-tuned: $\beta = 5$, $q_0 = 0.7$, and $K = 5$. It can be observed from Table 5 that all the three algorithms find optimal solutions for those 10 instances whose optimal solutions are known (i.e., with zero tardiness). For the remaining 22 instances, Tabu-VNS is the best for 11 ($11/20 = 55\%$) instances and ACO_{LJ} is the best for 9 ($9/20 = 45\%$) instances; 2 such cases end in a tie. Moreover, our ACO_{LJ} has updated 3 instances (Prob551, Prob654, Prob753) with unknown optimal solutions, which are provided at http://wwwdim.uqac.ca/~c3gagne/home_fichiers/ProbOrdo.htm. The comparison of computation times is difficult to make because the computation time of Tabu-VNS is not reported in its paper. We simply note that the average computation time of each run for our ACO_{LJ} is 24.17 s for all the 32 instances.

4. Conclusions

In this paper, we have proposed an ACO algorithm for minimizing the weighted tardiness with sequence-dependent setup times on a single machine. The developed algorithm has two distinctive features, including a new parameter for the initial pheromone trail and the change of timing for applying local search. These features, along with other elements, make the algorithm very effective and efficient. The algorithm not only updates 86% of the benchmark instances for the weighted tardiness problem but also it is very competitive with the existing best-performing metaheuristics including GA and Tabu-VNS. Moreover, the algorithm shows an excellent performance in computation time.

This paper presents an improvement-type algorithm for the weighted tardiness problem with sequence-dependent setup times on a single machine. This problem is important in real-world production system because the tardiness is recognized as the most important criterion and the setup time needs explicit consideration in many situations. The importance of the problem in practice justifies the researchers paying more attention to the problem and its extensions, such as in the more complicated job shop environment. With the help of recently developed metaheuristics, the researchers should be able to tackle more difficult problems in the real world.

References

[1] Allahverdi A, Gupta JND, Aldowaisan TA. A review of scheduling research involving setup considerations. Omega 1999;27:219–39.
[2] Das SR, Gupta JND, Khumawala BM. A saving index heuristic algorithm for flowshop scheduling with sequence dependent set-up times. Journal of the Operational Research Society 1995;46:365–73.
[3] Gravel M, Price WL, Gagné C. Scheduling jobs in an Alcan aluminium factory using a genetic algorithm. International Journal of Production Research 2000;38:3031–41.
[4] Wortman DB. Managing capacity: getting the most from your company's assets. Industrial Engineering 1992;24:47–9.
[5] Wisner JD, Siferd SP. A survey of US manufacturing practices in make-to-order machine shops. Production and Inventory Management Journal 1995;1:1–7.
[6] Lawler EL. A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics 1997;1:331–42.
[7] Pinedo M. Scheduling: theory algorithm and system. Englewood Cliffs, NJ: Prentice-Hall; 1995.
[8] Lee YH, Bhaskaram K, Pinedo M. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transactions 1997;29:45–52.

[9] Cicirello VA. Weighted tardiness scheduling with sequence-dependent setups: a benchmark library. Technical Report, Intelligent Coordination and Logistics Laboratory, Robotics Institute, Carnegie Mellon University, USA, 2003.
[10] Tan KC, Narasimhan R. Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach. Omega 1997;25:619–34.
[11] Rubin PA, Ragatz GL. Scheduling in a sequence-dependent setup environment with genetic search. Computers and Operations Research 1995;22:85–99.
[12] Gagné C, Gravel M, Price WL. A new hybrid Tabu-VNS metaheuristic for solving multiple objective scheduling problems. Proceedings of the Fifth Metaheuristics International Conference. Kyoto, Japan, 2003.
[13] Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1997;1:53–66.
[14] Bullnheimer B, Hartl RF, Strauss C. An improved ant system algorithm for the vehicle routing problem. Annals of Operations Research 1999;89:319–28.
[15] Gambardella LM, Taillard ÉD, Dorigo M. Ant colonies for the quadratic assignment problem. Journal of Operational Research Society 1999;50:167–76.
[16] Bauer A, Bullnheimer B, Hartl RF, Strauss C. An ant colony optimization approach for the single machine total tardiness problem. Proceedings of the 1999 Congress on Evolutionary Computation. New York: IEEE Press; 1999, p. 1445–50.
[17] den Besten M, Stützle T, Dorigo M. Ant colony optimization for the total weighted tardiness problem. Proceeding PPSN VI, Sixth International Conference Parallel Problem Solving from Nature, Lecture Notes in Computer Science, vol. 1917, Berlin: Springer; 2000. p. 611–20.
[18] Gagné C, Price WL, Gravel M. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. Journal of the Operational Research Society 2002;53:895–906.
[19] Ying GC, Liao CJ. Ant colony system for permutation flow-shop sequencing. Computers and Operations Research 2004;31:791–801.
[20] T'kindt V, Monmarché N, Tercinet F, Laügt D. An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. European Journal of Operational Research 2002;42:250–7.
[21] Dorigo M, Stützle T. The ant colony optimization metaheuristics: algorithms, applications, and advances. In: Glover F, Kochenberger G, editors. Handbook of metaheuristics, vol. 57. International Series in Operations Research & Management Science. Dordrecht: Kluwer; 2002. p. 251–85.
[22] Stützle T, Hoos HH. Max-min ant system. Future Generation Computer System 2000;16:889–914.