Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 1030–1046
computers & operations research
www.elsevier.com/locate/cor

Ant colony optimization combined with taboo search for the job shop scheduling problem

Kuo-Ling Huang, Ching-Jong Liao*

Department of Industrial Management, National Taiwan University of Science and Technology, 43 Keelung Road, Section 4, Taipei 106, Taiwan

Available online 22 August 2006

Abstract

In this paper, we present a hybrid algorithm combining ant colony optimization algorithm with the taboo search algorithm for the classical job shop scheduling problem. Instead of using the conventional construction approach to construct feasible schedules, the proposed ant colony optimization algorithm employs a novel decomposition method inspired by the shifting bottleneck procedure, and a mechanism of occasional reoptimizations of partial schedules. Besides, a taboo search algorithm is embedded to improve the solution quality. We run the proposed algorithm on 101 benchmark instances and obtain competitive results and a new best upper bound for one open benchmark instance is found.

Keywords: Ant colony optimization; Taboo search; Job shop scheduling; Makespan

1. Introduction

The problem that we address in this paper arises in the context of the classical job shop scheduling problem (JSSP). In the JSSP, a set of jobs has to be processed on several machines subject to both conjunctive and disjunctive constraints, and the objective is to minimize the makespan. This problem is NP-hard in the strong sense [1]; not until 1988 (publication year) could a relatively small instance with 10 jobs and 10 machines be solved to optimality.

Due to the complexity of the JSSP, it is unrealistic to solve even a medium-sized problem by using a time-consuming optimization algorithm, such as branch and bound schemes or integer programming [2]. Therefore, metaheuristics such as taboo search (TS) [3–6], genetic algorithms [7], and simulated annealing [8,43] have been studied much in recent years (also see review papers [9–11] for details).

However, each metaheuristic has its own strength and weakness. Therefore, much research has tried to develop hybrid algorithms expecting to achieve complementarity, which improves the effectiveness and efficiency. Those previous experiments have showed that the effectiveness and efficiency of hybrid algorithms are often better than those of more simplistic ones [12–16].

In this paper, we propose a hybrid algorithm for the JSSP. Following the concept of the shifting bottleneck (SB) procedure [17], the proposed algorithm combines ant colony optimization (ACO) with a TS algorithm. Due to its exploration and information learning abilities, ACO is expected to provide an appropriate initial schedule, which can thereby be enhanced by TS iteratively.

* Corresponding author. Fax: +886 2 2737 6344.
E-mail address: cjl@im.ntust.edu.tw (C.-J. Liao).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.07.003

The remainder of this paper is organized as follows. In the next section, the JSSP is formulated mathematically. In Section 3, the proposed ACO framework is introduced and analyzed. Then, we describe the local search methods and give the implementation details in Section 4. Finally, computational results for benchmark instances are provided and the proposed algorithm is compared with some of the best-performing ones.

2. Problem definition and notation

In the JSSP, a finite set of jobs is processed on a finite set of machines. Each job follows a predefined machine order and has a deterministic processing time. Each machine can process at most one job at a time, which cannot be interrupted until its completion. A feasible schedule of the JSSP is to build a permutation of jobs for each machine. The objective is to find a feasible schedule that minimizes the makespan.

The JSSP can mathematically be defined as follows. There are a set M of machines, a set J of jobs, and a set O of operations, where $\sigma_m^j (\sigma_m^j \in O)$ represents a specific operation for job j on machine m. Let $\sigma_m^j \prec \sigma_k^j$ be the processing order restriction, i.e., σ_k^j cannot be processed before the completion of σ_m^j. Let $\Pi(m)$ denote the permutation of jobs on machine m $(m = 1, \ldots, |M|)$, where $\Pi(m, j)(j = 1, \ldots, |J|)$ is the element of $\Pi(m)$ processed in position j. Hence, a feasible schedule of JSSP is defined by $\Pi = \{\Pi(1), \Pi(2), \ldots, \Pi(|M|)\}$.

The JSSP can be represented by the disjunctive graph $G = \{V, A, E\}$ given below [18]:

$$V = \{O \cup \text{source}, \text{sink}\},$$

$$\begin{aligned} A = {} & \{(\sigma_m^j, \sigma_k^j) | \sigma_m^j, \sigma_k^j \in O, \sigma_m^j \prec \sigma_k^j\} \\ & \cup \{(\text{source}, \sigma_k^j) | \sigma_k^j \in O, \nexists \sigma_m^j \in O \wedge \sigma_m^j \prec \sigma_k^j\} \\ & \cup \{(\sigma_m^j, \text{sink}) | \sigma_m^j \in O, \nexists \sigma_k^j \in O \wedge \sigma_m^j \prec \sigma_k^j\}, \end{aligned}$$

$$E = \{\{\sigma_m^i, \sigma_m^j\} | \sigma_m^i, \sigma_m^j \in O\}.$$

V is the set of operations, where the source and the sink are dummy operations, being the representative of the start and end operations of a schedule. A is the set of conjunctive arcs (directed arcs) connecting consecutive operations of the same job, and E is the set of disjunctive arcs (edges) that connects operations on the same machine. All arcs emanating from an operation have as length the processing time of the operation. In addition, the arcs emanating the source have length zero. A feasible schedule corresponds to orienting disjunctive arcs to conjunctive arcs such that the resulting directed graph is acyclic. Given a feasible schedule Π, the directed graph $G = \{V, A, E(\Pi)\}$ can be created where

$$E(\Pi) = \bigcup_{m=1}^{|M|} \bigcup_{j=2}^{|J|} (\Pi(m, j-1), \Pi(m, j)).$$

Note that each operation in the disjunctive graph has at most two predecessors and two successors. We now introduce the following additional notation to be used in this paper:

$PT(\sigma_m^j)$	the processing time of σ_m^j.
$MP(\sigma_m^j)$	the predecessor of σ_m^j that processes on machine m.
$MS(\sigma_m^j)$	the successor of σ_m^j that processes on machine m.
$JP(\sigma_m^j)$	the predecessor of σ_m^j that belongs to the same job j.
$JS(\sigma_m^j)$	the successor of σ_m^j that belongs to the same job j.
$F(\sigma_m^j)$	the longest path from source to σ_m^j.
$B(\sigma_m^j)$	the longest path from σ_m^j to sink.
$suc(\sigma_m^j)$	the successor sets of σ_m^j.
$pre(\sigma_m^j)$	the predecessor sets of σ_m^j.
$\pi(m)$	the processing priority index of machine m.
$C_{\max}(\Pi)$	the makespan value of feasible schedule Π.

3. Ant colony optimization

ACO, one of the metaheuristics dedicated to discrete optimization problems, is inspired by the foraging behavior of real ants which can be stated as follows [19]. Real ants are capable of finding the shortest path from a food source to their nest without using any visual cue. Instead, they communicate information about the food source via depositing a chemical substance, called pheromone, on the paths. The following ants are attracted by the pheromone. Since the shorter paths have higher traffic densities, these paths can accumulate higher proportion of pheromone. Hence, the probability of ants following these shorter paths would be higher than that of those following the longer ones.

ACO has been successfully applied to a large number of combinatorial optimization problems, such as the traveling salesman problem [20] and the vehicle routing problem [21], for which ACO was shown to be very competitive to other metaheuristics. Also, ACO has been applied successfully to scheduling problems such as single machine problems [22–24] and flow shop problems [25,26].

For the JSSP, Colorni et al. [27] first applied ACO to tackle the JSSP but obtained relatively uncompetitive computational results. Later, Blum and Sampels [28] developed a state-of-the-art ACO approach to tackle the general shop scheduling problem (called group shop scheduling problem), including the JSSP and the open shop scheduling problem. Their approach uses a strong non-delay guidance for constructing schedules with a newly developed pheromone model, where pheromone values are assigned to pairs of related operations (i.e., belonging to the same group), and applies a steepest descent local search to improve the constructed schedule. In addition, they extended the well-known neighborhood structure for the JSSP proposed by Nowicki and Smutnicki [5] to the group shop scheduling problem. Recently, Blum [29] developed a new competitive hybrid algorithm combining ACO algorithm with beam search for the open shop scheduling problem.

3.1. The proposed algorithm (ACOFT)

We now describe the framework of our proposed hybrid algorithm, called ACO combined with fast TS (ACOFT) algorithm. ACOFT is inspired by the SB procedure [17]; therefore, we need to introduce the SB procedure before presenting ACOFT. The SB procedure can be characterized as the following steps: subproblem identification, bottleneck selection, subproblem solution, and schedule reoptimization [10]. In the SB procedure each of the unscheduled machines is considered as a single machine problem (SMP), and the critical machine (the one with the maximum lateness) is treated as a bottleneck machine that should be scheduled first. ACOFT constructs schedules based on the similar concept, but employs a simple rule to identify the bottleneck machine and replaces the essential yet time-consuming step, schedule reoptimization, with the proximate optimality principle (POP) [18] to reduce the computational times.

The proposed ACOFT can be briefly sketched as follows. First, we identify the bottleneck machine among all the unscheduled machines. Each artificial ant constructs a permutation of jobs on the selected machine by using the state transition rule and then deposits an amount of pheromone by applying the local pheromone update rule. Occasionally, the POP is utilized to reoptimize the partial schedule. The above steps are repeated until a feasible schedule is established. Once all artificial ants have constructed their own schedules, the best one is improved by the embedded local search algorithm. Then the pheromone trails are modified again by applying the global pheromone update rule. The steps are iterated until a stopping criterion is satisfied. The details of each phase in ACOFT are addressed in what follows; the local search algorithm is to be elaborated in Section 4.

3.2. Initialization phase

To reduce the computational effort, we identify the bottleneck machine by applying a simple static rule, called total machine loading (TML) rule, which can be computed in advance. TML is defined as follows:

$$\pi(m) = \sum_{j=1}^{|J|} PT(\sigma_m^j) \quad \forall m = 1, \ldots, |M|,$$

where $\pi(m)$ is the TML ranking index of machine m. In this phase, a pheromone level τ_0 is initialized for all the trails, where τ_0 is a relatively small quantity.

3.3. Construction phase

3.3.1. Definition of pheromone trails for the JSSP

Before applying ACO, an important issue is to define the pheromone trails. We do not use the conventional construction approach [27,28,30], where in each construction step an operation can only be selected if all its predecessors are already in the partial schedule. Instead, we employ a decomposition method to construct a schedule, which is inspired by the SB procedure. In fact, the decomposition methods are usually used for the JSSP, e.g., Dorndorf et al. [31].

In our approach, an $|M| \times |J|$ JSSP is decomposed into $|M|$ separate SMPs, for each of which an ant constructs a permutation of jobs step by step until all the machines have been scheduled. Hence, we define $|M|$ pheromone matrices with size $|J| \times |J|$ for the related machines. Each of the pheromone matrices is defined by using the absolute position interpretation of pheromone trails (i.e., the pheromone trail associated with the assignment of an operation to a position), which is commonly applied in SMPs and generates better results [22,23].

3.3.2. State transition rule

In the construction phase, each artificial ant selects an unscheduled machine m with the highest TML level first. Then, it chooses the next operation from among a visibility set $O_V (O_V \subseteq O_m)$ instead of directly from O_m to guarantee feasibility; the method for determining O_V will be elaborated in Section 3.3.3. To select the next operation σ_m^j, we apply the following probability state transition rule:

$$\sigma = \begin{cases} \max\limits_{\sigma_m^j \in O_V} \{[\tau_m(p, j)] \cdot [\eta(\sigma_m^j)]^\beta\} & \text{if } q \leqslant q_0, \\ \phi & \text{otherwise,} \end{cases} \tag{1}$$

where $\tau_m(p, j)$ is the pheromone trail associated with assigning job j to position p with relative pheromone matrix m, and $\eta(\sigma_m^j)$ is the greedy heuristic desirability of σ_m^j. The parameter q_0 $(0 \leqslant q_0 \leqslant 1)$ determines the relative frequency between exploitation and exploration, and parameter β determines the influence of the heuristic information. Furthermore, ϕ is a random variable which gives the probability of assigning candidate job i to position p on machine m. The value of ϕ is determined by the probability distribution given below:

$$\Pr(\sigma_m^j) = \begin{cases} \dfrac{[\tau_m(p, j)] \cdot [\eta(\sigma_m^j)]^\beta}{\sum_{\sigma_m^i \in O_V} [\tau_m(p, i)] \cdot [\eta(\sigma_m^i)]^\beta} & \text{if } \sigma_m^j \in O_V, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The state transition rule resulting from Eqs. (1) and (2) is called pseudo-random-proportional rule and was introduced in [19]. This rule favors the choice of trails with higher pheromone levels.

3.3.3. Delayed precedence constraints

In the SB procedure, when several machines have been scheduled, the operations on a given unscheduled machine may be subject to a special type of precedence constraint. It may be that an operation that has to be processed on a particular machine can only be processed after certain other related operations (belonging to the same machine) have completed their processing. ACOFT uses a similar construction concept and therefore encounters the same problem. To guarantee feasibility, we implement the delayed precedence constraints described by Dauzère-Pérès and Lasserre [32] and Balas et al. [33] as follows. Each time before scheduling a machine m, we use the depth-first search method to obtain $suc(\sigma_m^j)$ for all $\sigma_m^j \in O_m$. If there exists an operation σ_m^i ($\sigma_m^i \in O_m$ and $\sigma_m^i \neq \sigma_m^j$) belonging to $suc(\sigma_m^j)$, we generate a delayed precedence constraint $(\sigma_m^j \prec \sigma_m^i)$. Thus, only if σ_m^j is processed can σ_m^i be added to O_V.

3.3.4. Greedy heuristic rule

While applying the state transition rule, the following two greedy heuristics are experimented for the heuristic information $\eta(\sigma_m^j)$:

(1) *Most work remaining* (MWR): This static heuristic selects the operation belonging to the job with the most remaining processing time.

(2) *Time remaining* (TR): This dynamic heuristic selects the operation σ_m^j with the longest path between σ_m^j and the dummy sink operation, (i.e., $\eta(\sigma_m^j) = B(\sigma_m^j)$).

An obvious difference between the static and dynamic greedy heuristics is that the static one needs to be computed in advance, whereas the dynamic one has to be computed at run-time, which is time-consuming. Moreover, to ensure that the constructed schedule is an active schedule, we use the insertion technique proposed by Aiex et al. [18]. Each time when the artificial ant chooses σ_m^j, we test if the operation can be inserted as early as possible without delaying any other scheduled operations on machine m.

3.3.5. Local update pheromone rule

After an artificial ant has completed a permutation of the executed machine m, the corresponding pheromone matrix is updated by applying the local pheromone update rule as follows:

$$\tau_m(p, j) := (1 - \rho) \cdot \tau_m(p, j) + \rho \cdot \tau_0 \quad \forall (p, j) \in \Pi(m),$$

where τ_0 is the initial pheromone level and ρ $(0 < \rho < 1)$ is the pheromone evaporating parameter. The effect of the local pheromone update rule is to make the choice of putting job j in position p on the machine less desirable for other ants to achieve diversification. Consequently, this mechanism favors the exploration of different schedules. Experimentally, without using this rule all the ants would search in a narrow neighborhood of the best previous schedule.

3.3.6. Proximate optimality principle

The POP, introduced by Fleurent and Glover [34], is implemented for the JSSP by Binato et al. [35]. In general, POP states that good solutions of partial schedules with k operations are close to good solutions of partial schedules with $k + 1$ operations.

In ACOFT, POP is utilized to reoptimize partial schedules. Our preliminary experiments show that the use of POP can improve the constructed schedules but costs additional computational time. To strike a balance between effectiveness and efficiency empirically, POP is therefore not executed after solving every SMP but only executed when 25%, 50%, and 75% of the total number of operations have been scheduled (i.e., three times in total). A detailed implementation of POP will be described in Section 4.5.

3.4. Local search phase

In ACO, the generated schedules by artificial ants may be so coarse that they should be improved by some complementary local search method [10,11]. The reason that the earliest application of ACO to the JSSP generates unsatisfactory results may be due to the lack of an appropriate local search.

To overcome this shortcoming, ACOFT tries to combine ACO with a powerful TS algorithm, called fast TS algorithm (FT or TSAB), that was proposed by Nowicki and Smutnicki [5]. However, a good initial schedule for FT is very important but relatively difficult to obtain. In this regard, we try to employ ACO, which has excellent exploration and information learning abilities, to provide an appropriate initial schedule for FT.

To make FT more efficient, we further modify its makespan calculation, the most time-consuming step. A detailed implementation of the modified makespan calculation will be given in Section 4.

3.5. Global pheromone update phase

This phase is performed after all ants have completed their schedules. In order to make the search more directed, the global pheromone update rule is intended to provide a larger amount of pheromone to better schedules.

Most research in this phase uses the elitist strategy, which permits only the global best sequence of the current iteration to deposit pheromone trails; however, this strategy may not be the best rule (see [36] for more details). Our experimental evidence showed that the elitist strategy may not be suitable for the JSSP. The elitist strategy may suit only for small-sized instances; for relatively large ones (e.g., 30 jobs, 20 machines, and 600 operations), updating only the best schedule usually results in poor convergence, implying that there is a need to adjust the parameter of the pheromone evaporation rate. In this regard, we propose to update the pheromone trails from a series of good schedules.

Once the current best schedule is improved in the local search phase, the new best one is stored in a queue, called global pheromone update queue, which will be elaborated in Section 4.3.

The definition of the global pheromone update rule is given as follows:

$$\tau_m(p, j) := (1 - \alpha) \cdot \tau_m(p, j) + \alpha \cdot \Delta\tau_m(p, j),$$

where

$$\Delta\tau_m(p, j) = \begin{cases} \left(\dfrac{OptValue}{C_{\max}(\Pi)}\right)^R & \text{if } (p, j) \in \Pi(m) \subset \text{global pheromone update queue,} \\ 0 & \text{otherwise.} \end{cases}$$

In the above equation, $\Delta\tau_m(p, j)$ is the amount of pheromone level added to $\tau_m(p, j)$ by the artificial ants, and parameter α $(0 < \alpha \leqslant 1)$ is the pheromone evaporation rate, which is used to avoid unlimited accumulation of pheromone and enable the algorithm to vanish worse pheromone information. R is set at $|J|$ in order to discriminate among the schedules in the global pheromone update queue and *OptValue* is the optimal makespan (or the best upper bound). In this study all the instances we solved are the famous benchmark instances that have been solved for several years and therefore the best makespan can be obtained from the web sites. However, if an instance has not been solved before, one may use some other heuristics such as SB procedure to obtain an upper bound. Our experiments show that there is no significant difference if one uses an upper bound obtained by SB procedure instead of the optimal makespan (or best upper bound).

4. Implementations of local search algorithm

In this section, we discuss the local search algorithms employed in ACOFT, including FT and POP.

4.1. TS algorithm

TS algorithm, proposed by Glover, is a successful local search algorithm for solving combinatorial optimization problems [37,38], and can be sketched as follows. TS starts from an initial solution. In each iteration, a move is performed to the best neighboring solution although its quality is not better than the current one. To prevent cycling, a short-term memory, called taboo list, is employed for recording a fixed number of recent moves. The use of a taboo list avoids returning to a local minimum that has been visited before. In addition, another memory structure called long-term memory is proposed, expecting to improve the diversification.

Many researchers such as Taillard [6], Dell'Amico and Trubian [3], Grabowski and Wodecki [4], and Nowicki and Smutnicki [5] applied TS to the JSSP and showed its superiority over many other metaheuristics. In this study, we employ the FT or TSAB, proposed by Nowicki and Smutnicki [5], to improve the constructed schedules. Following is the introduction of FT, and readers are recommended to refer to [5] for details.

4.2. The fast taboo algorithm

FT can be characterized by the following components: initial schedule, neighborhood definition, short-term memory (taboo list), long-term memory, and stopping criterion. In the following is the detailed implementation of FT.

The neighborhood of FT can be defined as follows. Consider a given critical path and divide it into several blocks, each of which contains at least two operations processed on the same machine. The neighborhood consists of all schedules that can be obtained from the current one by swapping the first (last) two operations in the last (first) block and in the others by swapping the first and last two successive operations.

To improve diversification, once a better schedule is found, the schedule, set of moves (neighborhood of that schedule except for the move being applied), and the taboo list are stored in a queue, namely long-term memory. When the search is terminated according to the stopping criterion of the short-term memory, the schedule, set of moves, and taboo list located in the last position in the long-term memory are retrieved to serve as a new starting point of the local search.

In ACOFT, the taboo list length MaxTL is set initially to the magical number 7 [37,38] and increases in dependence of the accumulated number of iterations of the ACO algorithm. The value of MaxTL is increased by one after every quarter of the total number of iterations, except for the last quarter (i.e., MaxTL changes between 7 and 10, inclusively). Let MaxLM be the maximum length of the long-term memory and MaxTabooIter be the stopping criterion of short-term

memory. Obviously, increasing the sizes of MaxLM and MaxTabooIter may yield a higher probability for getting a better schedule but requires more computational effort. As a trade off, they are determined experimentally.

Furthermore, to prevent cycling, FT exercises a cyclic test function to judge if there exists a cyclic period $\delta(1 \leqslant \delta \leqslant \mathrm{Max}\delta)$ by detecting repetitions of makespan values in constant time. If there exists some period that repeats MaxCycle times, then we stop the search and retrieve the element located in the last position in the long-term memory to serve as a new starting point of the local search.

4.3. Long-term memory and global pheromone update queue

The proposed global pheromone update queue has a data structure similar to the long-term memory. Therefore, once a better schedule is found, ACOFT immediately updates not only the long-term memory, but also the global update pheromone queue. Experimentally, the length of the global pheromone update queue is set at $\lfloor |J|/3+|M|/2 \rfloor$, implying that the larger the instance sizes are, the more schedules the global pheromone update queue has. This makes the use of pheromones more effective.

4.4. The modified makespan calculation

Despite producing high quality results in a reasonable computation time, the FT in ACOFT, however, has to be executed in each iteration, causing a heavy computational effort. The proposed ACOFT attempts to accelerate the standard FT by using a modified makespan calculation.

We begin by analyzing the time complexity of FT. In each iteration, two dynamic programming formulae are applied to calculate the longest path from operation $\sigma(\sigma \in O)$ to sink and source as follows:

$$F(\sigma) = \max\{F(MP(\sigma)) + PT(MP(\sigma)),\ F(JP(\sigma)) + PT(JP(\sigma))\}, \tag{3}$$

$$B(\sigma) = \max\{B(MS(\sigma)),\ B(JS(\sigma))\} + PT(\sigma), \tag{4}$$

where $F(\text{source}) = B(\text{sink}) = 0$, and the makespan can be calculated in time $O(|O|)$. However, a lower bound developed by Taillard [39] can be used to reject worse moves in constant time. Supposing that a move (σ_m^i, σ_m^j) (the old operations order) is applied, the lower bound of the makespan for the new critical path is calculated by the following equation:

$$LB = \max\{F'(\sigma_m^j) + B'(\sigma_m^j),\ F'(\sigma_m^i) + B'(\sigma_m^i)\},$$

where

$$F'(\sigma_m^j) = \max\{F(MP(\sigma_m^i)) + PT(MP(\sigma_m^i)),\ F(JP(\sigma_m^j)) + PT(JP(\sigma_m^j))\},$$

$$F'(\sigma_m^i) = \max\{F'(\sigma_m^j) + PT(\sigma_m^j),\ F(JP(\sigma_m^i)) + PT(JP(\sigma_m^i))\},$$

$$B'(\sigma_m^i) = \max\{B(MS(\sigma_m^j)) + PT(\sigma_m^i),\ B(JS(\sigma_m^i)) + PT(\sigma_m^i)\},$$

$$B'(\sigma_m^j) = \max\{B'(\sigma_m^i) + PT(\sigma_m^j),\ B(JS(\sigma_m^j)) + PT(\sigma_m^j)\}.$$

If LB is larger than the previous makespan, then it must be the new value of the critical path; on the contrary, if it is less than one, we should recalculate the makespan value.

Once a best neighbor is selected (i.e., from one iteration to the next), the values of $F(\sigma)$ $(\sigma \in \{suc(\sigma_m^j) \cup \sigma_m^j\})$ and $B(\sigma)$ $(\sigma \in \{pre(\sigma_m^i) \cup \sigma_m^i\})$ can be updated by, respectively, applying (3) and (4), taking at most $O(|O|)$ time. In comparison with the standard FT, the computational speed was accelerated by a factor of about two times.

4.5. POP with partial disjunctive graph

In ACOFT, POP local search is based on FT but it works without long-term memory. As stated earlier, POP is only executed a quarter of the total operations have been scheduled, except for the last one. The length of POP taboo list (MaxPOPTL) is set at 4 initially and increased by one each time after reaching the stopping criterion (i.e., MaxPOPTL changes between 4 and 6, inclusively). Another parameter MaxPOPIter, which specifies the maximum number of iterations without improving the best partial schedule obtained so far, is much less than MaxTabooIter in order to reduce the computational load.

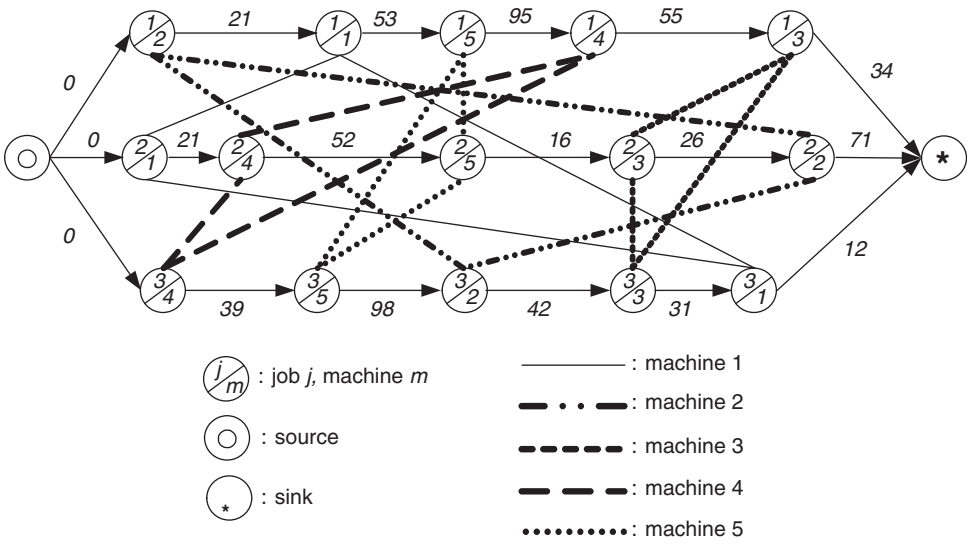

Fig. 1. An example of five machines, three jobs, and 15 operations.

Furthermore, to accelerate the procedure, we substitute the general disjunctive graph (GDG) by a partial disjunctive graph (PDG) in our POP. Let O_S denote the set of operations already scheduled, where suffix S is the accumulated number of executed machines. Then the PDG $G_p = \{V, A, E(\Pi)\}$ is given below:

$$V = \{O_S \cup \text{source} \cup \text{sink}\},$$

$$\begin{aligned} A = {} & \{(\sigma_m^j, \sigma_k^j) | \sigma_m^j, \sigma_k^j \in O_S, \sigma_m^j \prec \sigma_k^j\} \\ & \cup \{(\text{source}, \sigma_k^j) | \sigma_k^j \in O_S, \nexists \sigma_m^j \in O_S \wedge \sigma_m^j \prec \sigma_k^j\} \\ & \cup \{(\sigma_m^j, \text{sink}) | \sigma_m^j \in O_S, \nexists \sigma_k^j \in O_S \wedge \sigma_m^j \prec \sigma_k^j\}, \end{aligned}$$

$$E(\Pi) = \bigcup_{m=1}^{S} \bigcup_{j=2}^{|J|} (\Pi(\pi(m), j-1), \Pi(\pi(m), j)).$$

Considering only the scheduled operations in the PDG implies that when applying (3) and (4), we need not only omit the unscheduled operations along with their adjacent arcs, but also connect the remaining adjacent scheduled operations belonging to the same job with dummy arcs. Note that there exist no edges in the PDG, and the weights of the dummy arcs can be calculated in advance.

We now use a numerical example to describe how the PDG works. Consider a JSSP with five machines and three jobs as depicted in Fig. 1. Given the following partial schedule $\Pi = \{\Pi(2), \Pi(4), \Pi(5)\}$, where

$$\begin{aligned} \Pi(2) &= \{\sigma_2^1, \sigma_2^3, \sigma_2^2\}, \\ \Pi(4) &= \{\sigma_4^3, \sigma_4^2, \sigma_4^1\}, \\ \Pi(5) &= \{\sigma_5^3, \sigma_5^2, \sigma_5^1\} \end{aligned}$$

a GDG can be illustrated in Fig. 2, which has exactly one critical path (source, σ_4^3, σ_5^3, σ_5^2, σ_5^1, σ_4^1, σ_3^1, sink) with length 337. Now, we apply the PDG to modify Fig. 2 and illustrate it in Fig. 3, where the unscheduled operations and their adjacent arcs represented by dotted lines are omitted. In addition, five dummy arcs are added to connect the scheduled operations, where their weights are added accordingly. When applying the modified makespan calculation, the time complexity of the PDG is reduced to $O(|O_S|)$.

Note that the notation $JP(\sigma_j^m)$, $JS(\sigma_i^m)$, and $PT(\sigma_j^m)$ in the proposed modified makespan calculation should be updated slightly if their adjacent arcs are replaced by the dummy ones. For instance, since arc (σ_1^2, σ_1^1), operation σ_1^1, and arc

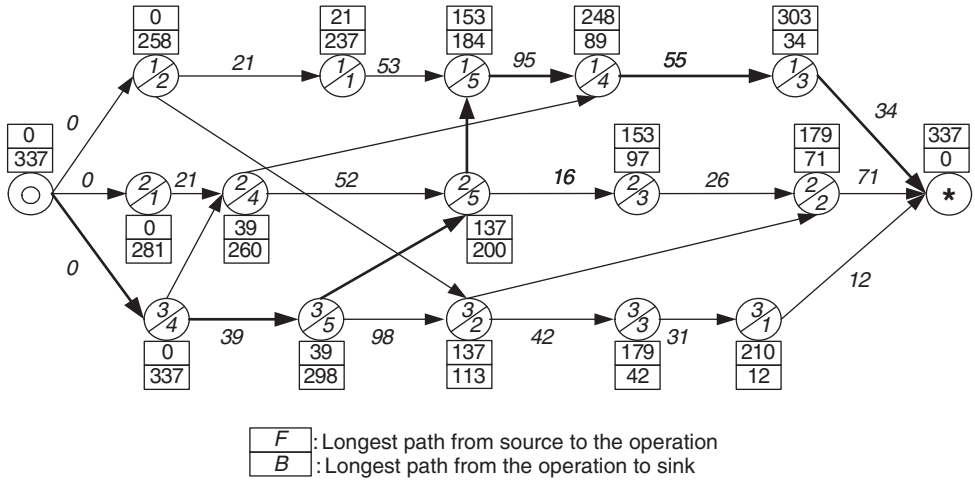

Fig. 2. The general disjunctive graph with three machines being scheduled.

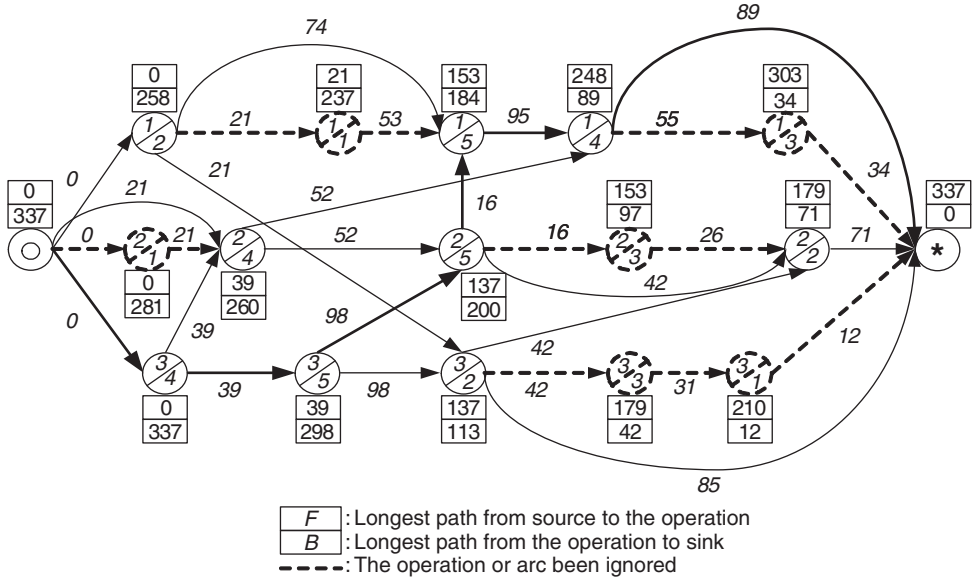

Fig. 3. The proposed partial disjunctive graph of Fig. 2.

(σ_1^1, σ_1^5) are omitted, the job successor $JS(\sigma_1^2) = \sigma_1^5$ and the processing time $PT(\sigma_1^2)$, depending on its successor, may equal either 21 (weight of the original arc) or 74 (weight of the dummy arc). Suppose now that we want to apply the move (σ_5^3, σ_5^2) (the old operations order). The lower bound can then be calculated as follows:

$$F'(\sigma_5^2) = \max\{0, 39 + 52\} = 91,$$
$$F'(\sigma_5^3) = \max\{91 + 16, 0 + 39\} = 107,$$
$$B'(\sigma_5^3) = \max\{84 + 98, \underline{0} + \underline{183}\} = 282,$$
$$B'(\sigma_5^2) = \max\{282 + 16, \underline{0} + \underline{113}\} = 298,$$
$$LB = \max\{107 + 282, 91 + 298\} = 389,$$

where the modified numbers are underlined. Since the new lower bound is larger than the previous makespan in Fig. 3 (337), the new longest path must pass through σ_5^3 and σ_5^2 with makespan 389.

Table 1
The setting values of ACOFT parameters

MaxAnt $= \lfloor \lvert M \rvert /3 \rfloor$	MaxPOPIter $= 25$
MaxIter $= 320$	MaxPOPTL $\in \{4, 5, 6\}$
$\tau_0 = 10$	MaxTabooIter $= 3000$
$\beta = 1$	MaxLM $= 3$
$\rho = 0.1$	MaxTL $\in \{7, \ldots, 10\}$
max $\delta = 30$	MaxCycle $= 6$
$q_0 \in \{0.9^{a}, 0.65^{b}\}$	$\alpha \in \{0.85^{a}, 0.7^{b}\}$

[a]The general parameter set of the algorithm.
[b]The parameter set when falling into local optimum.

Table 2
Computational comparison between the partial disjunctive graph (PDG) and general disjunctive graphs (GDG)

Instance	$\lvert J \rvert \times \lvert M \rvert$	$\overline{T}$(PDG)	$\overline{T}$(GDG)	Improvement (times)
LA36	15×15	45.372	102.279	2.254
TA01	15×15	44.161	96.875	2.194
TA11	20×15	109.914	228.194	2.095
TA21	20×20	232.375	474.681	2.042

5. Computational results

A computational study was conducted to evaluate the efficiency and effectiveness of our proposed ACOFT algorithm, which was coded in C++ and run on a PC with an AMD XP-1800+ (1533 MHz) CPU and 768 MB RAM under Windows XP using Microsoft Visual C++ 6.0.

Choosing appropriate parameters is time-consuming and in general they depend on the particular instance. Thus, some of the parameters in our approach are adjusted dynamically as follows: when ACOFT generates the same makespan value during successive two iterations, the algorithm is probably falling into a local minimum, and hence, we lower parameters q_0 (utilization of exploration) and α (global pheromone evaporation rate) to give the algorithm a higher probability to escape from the local minimum. Table 1 lists all the parameter values used in our experiments.

ACOFT was tested on 101 benchmark instances of different sizes which can be downloaded from OR-Library web site (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html). The best known lower and upper bounds of Taillard's instances are provided on Taillard's web site (http://ina2.eivd.ch/collaborateurs/etd/default.htm). All the tested instances are classified into the following three classes:

(a) Nine instances of three different sizes ($\lvert J \rvert \times \lvert M \rvert = 15 \times 10, 20 \times 10, 15 \times 15$) denoted by (LA24, LA25, LA27, LA29, and LA36–LA40) due to Lawrence [39], where instance LA29 is still open.
(b) Five instances denoted by (ORB1–ORB5) due to Applegate and Cook [40], two instances (FT10, FT20) due to Fisher and Thompson [41], and five instances (ABZ5–ABZ9) due to Adams et al. [17]. Instances ABZ8 and ABZ9 are still open.
(c) Eighty instances of eight different sizes denoted by (TA01–TA80) due to Taillard [42]. These instances are solved only for 48 out of 80 instances.

Before evaluating the performance of ACOFT, we experimented with ACOFT without the local search phase (FT) for both general and PDGs for comparing their computational efficiency. All the parameters are set equal, including the random seeds, to make a fair comparison. The experiment was carried out on four instances (namely LA36, TA01, TA11, and TA21) with different sizes, for each of which the algorithm ACOFT was run five times. Table 2 indicates that using the PDG can accelerate the computational speed about two times.

Table 3
Comparisons of ACOFT-MWR, ACOFT-RAND, FT-LTM, and FT-STM with the same computational times

Instance	OPT or (LB, UB)	$\overline{T}$	ACOFT-MWR			ACOFT-RAND			FT-LTM			FT-STM		
			$C^*_{\max}$	$\overline{C}_{\max}$	RE	$C^*_{\max}$	$\overline{C}_{\max}$	RE	$C^*_{\max}$	$\overline{C}_{\max}$	RE	$C^*_{\max}$	$\overline{C}_{\max}$	RE
LA29[a]	(1142, 1152)	1442.7	1158	1165.6	1.40	1164	1167.7	1.93	1164	1168.2	1.93	1168	1180.5	2.28
LA40[a]	1222	941.4	1224	1226.4	0.16	1224	1228.4	0.16	1224	1229.0	0.16	1228	1239.5	0.49
ABZ07[a]	656	1438.6	658	663.2	0.30	666	673.6	1.52	665	669.2	1.37	666	671.4	1.52
ABZ08[a]	(645, 665)	1634.3	670	670.8	3.88	670	674.5	3.88	674	678.5	4.50	679	684.9	5.27
ABZ09[a]	(661, 679)	1792.9	683	687.4	3.33	687	688.3	3.93	688	691.4	4.08	691	699.4	4.54
TA01[b]	1231	1154.2	1231	1232.1	0.00	1231	1234.3	0.00	1234	1238.2	0.24	1242	1249.0	0.89
TA11[b]	(1323, 1361)	1604.2	1367	1371.9	3.33	1367	1371.2	3.33	1379	1383.6	4.23	1391	1408.5	5.14
TA21[b]	(1539, 1644)	2640.3	1650	1661.5	7.21	1647	1658.5	7.02	1658	1668.1	7.73	1688	1716.5	9.68
TA32[b]	(1774, 1796)	3547.7	1822	1838.4	2.71	1830	1842.0	3.16	1839	1852.2	3.66	1843	1858.7	3.89
TA41[b]	(1859, 2018)	4790.1	2038	2051.8	9.63	2047	2059.4	10.1	2044	2064.9	9.95	2083	2108.9	12.05
MRE					3.19			3.50			3.79			4.58

[a]The instance runs for 15 replications.
[b]The instance runs for 10 replications.

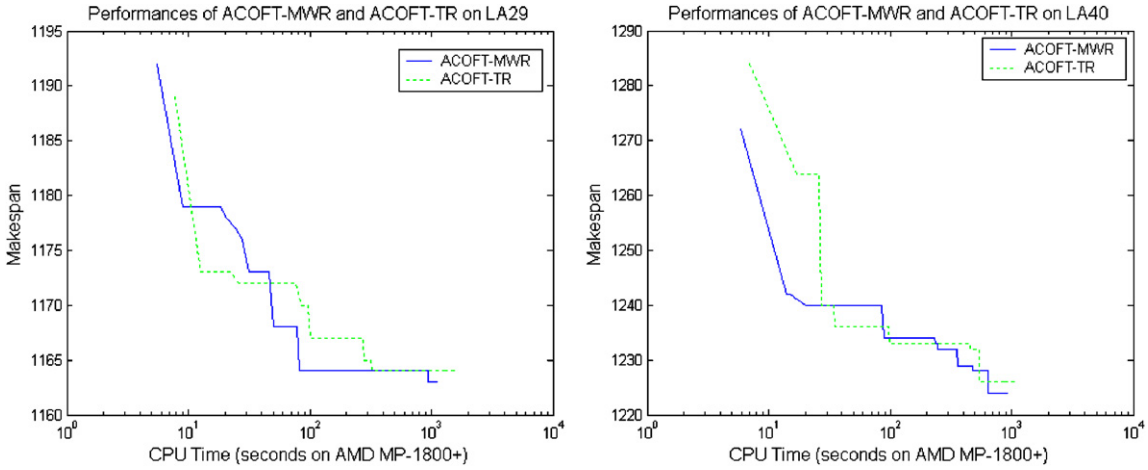

Fig. 4. The performances of ACOFT-MWR and ACOFT-TR on LA29 and LA40.

Table 4
The development of the solution quality of ACOFT-MWR and ACOFT-TR on LA40 after 10, 100, and 1000 s

Seconds	ACOFT-MWR			ACOFT-TR		
	$C^*_{\max}$	$\overline{C}_{\max}$	$C^{**}_{\max}$	$C^*_{\max}$	$\overline{C}_{\max}$	$C^{**}_{\max}$
10	1242	1263.5	1278	1258	1270.1	1283
100	1224	1238.6	1242	1227	1241.7	1244
1000	1224	1226.4	1228	1224	1228.4	1232

Now we demonstrate that whether the developed pheromone model can help at all in generating the good schedules. One may treat our approach as a repeated restart FT that runs several times and takes the best result; however, our experiments show that the pheromones are indeed inessential. Without this mechanism, the generated initial schedules for FT would become poor; thus, FT may waste lots of computational efforts on converging. This will be demonstrated

Table 5
Comparison with other algorithms for instances class (a)

Instance	OPT or (LB, UB)	SB-GA		TSAB (FT)		SB-RGLS2		TSSB		SHKT-720min		ACOFT-MWR				ACOFT-TR			
		$C^*_{\max}$	RE	$C^*_{\max}$	RE	$C^*_{\max}$	RE	$C^*_{\max}$	RE	$C^*_{\max}$	RE	$C^*_{\max}$	$\overline{C}_{\max}$	$\overline{T}$	RE	$C^*_{\max}$	$\overline{C}_{\max}$	$\overline{T}$	RE
LA24	935	957	2.353	939	0.428	935	0.000	938	0.321	938	0.321	935	938.4	764.6	0.000	935	938.4	693.9	0.000
LA25	977	1007	3.071	977	0.000	977	0.000	979	0.205	977	0.000	977	977.0	39.3	0.000	979	977.0	56.6	0.000
LA27	1235	1269	2.753	1236	0.081	1235	0.000	1235	0.000	1238	0.243	1235	1236.6	385.6	0.000	1235	1237.2	1035.6	0.000
LA29	(1142, 1152)	1210	5.954	1160	1.576	1164	1.926	1168	2.277	1161	1.664	1158	1165.6	1442.7	1.401	1163	1165.4	1533.1	1.839
LA36	1268	1317	3.864	1268	0.000	1268	0.000	1268	0.000	1268	0.000	1268	1268.0	36.6	0.000	1268	1268.0	123.5	0.000
LA37	1397	1446	3.508	1407	0.716	1397	0.000	1411	1.002	1397	0.000	1397	1398.8	879.6	0.000	1397	1400.1	1127.6	0.000
LA38	1196	1241	3.763	1196	0.000	1196	0.000	1201	0.418	1196	0.000	1196	1196.0	55.4	0.000	1196	1196.0	58.1	0.000
LA39	1233	1277	3.569	1233	0.000	1233	0.000	1240	0.568	1233	0.000	1233	1233.0	65.7	0.000	1233	1233.0	161.1	0.000
LA40	1222	1252	2.455	1229	0.573	1224	0.164	1233	0.900	1224	0.164	1224	1226.4	941.4	0.164	1224	1225.9	1151.4	0.164
MRE			3.477		0.375		0.232		0.632		0.266				0.174				0.223

Table 6
Comparison with other algorithms for instances class (b)

Instance	$\|J\| \times \|M\|$	OPT or (LB, UB)	GPPR		SHKT-720min		TSSB		ACOFT-MWR				ACOFT-TR			
			$C^*_{\max}$	RE	$C^*_{\max}$	RE	$C^*_{\max}$	RE	$C^*_{\max}$	$\overline{C}_{\max}$	$\overline{T}$	RE	$C^*_{\max}$	$\overline{C}_{\max}$	$\overline{T}$	RE
ORB1	10×10	1059	1059	0.000	1059	0.000	1064	0.472	1059	1059.0	56.6	0.000	1059	1059.0	327.5	0.000
ORB2	10×10	888	888	0.000	888	0.000	890	0.225	888	888.5	569.3	0.000	888	888.2	166.4	0.000
ORB3	10×10	1005	1005	0.000	1005	0.000	1013	0.796	1005	1007.4	403.7	0.000	1005	1005.0	24.7	0.000
ORB4	10×10	1005	1005	0.000	1005	0.000	1013	0.796	1005	1005.0	17.9	0.000	1005	1008.4	862.6	0.000
ORB5	10×10	887	887	0.000	887	0.000	887	0.000	887	888.9	670.3	0.000	887	888.7	808.1	0.000
FT10	10×10	930	930	0.000	930	0.000	930	0.000	930	930.0	64.6	0.000	930	930.0	118.2	0.000
FT20	20×5	1165	1165	0.000	1165	0.000	1165	0.000	1165	1165.0	46.0	0.000	1165	1165.0	22.3	0.000
ABZ5	10×10	1234	1234	0.000	1234	0.000	1234	0.000	1234	1235.8	501.9	0.000	1234	1235.8	832.4	0.000
ABZ6	10×10	943	943	0.000	943	0.000	943	0.000	943	943.0	199.3	0.000	943	943.0	238.2	0.000
ABZ7	20×15	656	692	5.488	661	0.762	666	1.524	658	663.2	1438.6	0.305	660	663.9	2993.6	0.601
ABZ8	20×15	(645, 665)	705	9.302	672	4.186	678	5.116	670	670.8	1634.3	3.876	671	673.7	3374.6	4.031
ABZ9	20×15	(661, 679)	740	11.95	687	3.933	693	4.841	683	687.4	1792.9	3.327	685	689.2	3621.8	3.631
MRE				2.228		0.740		1.148				0.626				0.689

Table 7
Comparison with other algorithms for instances class (c)

Instance	$\lvert J\rvert \times \lvert M\rvert$	TSSB		BV-Best		TSGW		ACOFT-MWR		ACOFT-TR	
		MRE	$\overline{T}$	MRE	$\overline{T}$	MRE	$\overline{T}$	MRE	$\overline{T}$	MRE	$\overline{T}$
TA01–TA10	15 × 15	0.450	2175	0.173	1498	0.272	7.58	0.082	1109.7	0.057	1404.6
TA11–TA20	20 × 15	3.473	2526	3.018	4559	3.866	7.12	2.698	1553.4	2.745	2891.5
TA12–TA30	20 × 20	6.500	34 910	6.098	6850	6.314	20.45	5.750	2319.5	5.738	3673.1
TA31–TA40	30 × 15	1.921	14 133	0.795	8491	1.757	20.04	0.975	3016.4	0.948	5202.4
TA41–TA50	30 × 20	6.043	11 512	5.204	16 018	5.820	9.93	5.114	4703.0	4.844	9989.2
TA51–TA60	50 × 15	0.019	421	0.000	196	0.008	4.09	0.000	173.6	0.000	1393.7
TA61–TA70	50 × 20	0.396	6342	0.112	2689	0.364	9.20	0.021	2637.1	0.049	5633.7
TA71–TA80	100 × 20	0.000	231	0.000	851	0.000	6.50	0.000	267.4	0.000	739.2
MRE		2.350		1.925		2.218		1.830		1.798	
No. OPT		31		35		34		38		40	

in the following experiment, all of which were run for the same computational times, depending on the average computational time needed for ACOFT-MWR with 320 iterations.

1. ACOFT-MWR: ACOFT with the MWR heuristic rule is run with pheromones.
2. ACOFT-RAND: ACOFT is run without pheromones (i.e., we set $\tau_m(p, j)$ for all m, p, and j to be constant values). To generate different initial schedules in each iteration, we set $q_0 = 0$ (employs exploration only). This corresponds to some type of randomized restart mechanism for the FT search algorithm.
3. FT-LTM: The FT search algorithm is run alone with the backtracking feature. To guarantee that the FT-LTM can run until meeting the stopping criteria, we let MaxTabooIter = 30 000 and MaxLM = 300.
4. FT-STM: The FT search algorithm is run alone without the backtracking feature. To guarantee that the FT-STM can run until meeting the stopping criteria, we let MaxTabooIter be a very large integer number.

Note that the initial schedules for FT-LTM and FT-STM are obtained from ACO. Table 3 lists the computational results, including the best value of the makespan from the replications ($C^*_{\max}$), the relative error of this best makespan (RE; a percentage by which the solution obtained is above the optimum value (OPT) if it is known or the best lower bound value (LB)), the mean relative error (MRE), the average makespan ($\overline{C}_{\max}$), the average computational time ($\overline{T}$), and the number of replications for each instance. Clearly, it is shown that algorithms ACOFT-RAND, FT-LTM, and FT-STM with the same computational times cannot obtain relative better results, which demonstrates that the good performance of ACOFT is not merely because of on large computational times.

In the following experiments, we examined two different heuristic rules on ACOFT, namely ACOFT-MWR and ACOFT-TR. Instances (a) and (b) are run for 15 replications, and instances (c), which are more difficult and time-consuming, are run for 10 replications. The values of $C^*_{\max}$, RE, $\overline{C}_{\max}$, $\overline{T}$, and MRE for ACOFT-MWR and ACOFT-TR are listed in Tables 3, 5, 6, and 8. The performances of ACOFT-TR and ACOFT-MWR are illustrated in Fig. 4 (runs for one replication), and the development of the solution quality of the algorithms in terms of $C^*_{\max}$, $\overline{C}_{\max}$, and $C^{**}_{\max}$ (the worst solution) after 10, 100, and 1000 s are also listed in Table 4 (runs for 15 replications).

Table 5 reports the best computational results obtained by Dorndorf and Pesch (SB-GA) [13], Nowicki and Smutnicki [5] (FT), Balas and Vazacopoulos [22] (SB-RGLS2), Pezzella and Merelli [14] (TSSB), and Schultz et al. [8] (SHKT-720min). Overall, ACOFT-MWR outperforms all the other novel algorithms in terms of the best solution quality.

Table 6 shows the comparison of our algorithms with other novel heuristic algorithms proposed by Aiex et al. [18] (GPPR), Schultz et al. [8] (SHKT-720min), Pezzella and Merelli [14] (TSSB) on instances (b). It is observed from Table 6 that the proposed algorithms produce MREs of 0.626% (ACOFT-MWR) and 0.689% (ACOFT-TR), both of which exhibit the competition with other novel algorithms.

The comparison in Tables 5 and 6 is done to assess the peak performance of our algorithms with respect to the solution quality of the solutions reachable when compared to the state-of-the-art algorithms for the JSSP. These tables only list the computational times of our algorithms but omit those of others because it is not easy to compare the

Table 8
Results by ACOFT for instances class (c)

Instance	OPT or (LB, UB)	ACOFT-MWR				ACOFT-TR			
		$C^*_{\max}$	$\overline{C}_{\max}$	RE	$\overline{T}$	$C^*_{\max}$	$\overline{C}_{\max}$	RE	$\overline{T}$
TA01	1231	1231	1232.1	0.000	1154.2	1231	1233.4	0.000	1531.4
TA02	1244	1244	1244.2	0.000	343.9	1244	1244.2	0.000	685.2
TA03	1218	1220	1221.3	0.164	1368.8	1220	1223.3	0.164	1833.7
TA04	1175	1175	1180.3	0.000	1029.9	1175	1180.1	0.000	1186.2
TA05	1224	1229	1237.5	0.408	1242.0	1228	1236.9	0.327	1492.6
TA06	1238	1240	1244.7	0.162	1431.5	1238	1242.6	0.000	1549.1
TA07	1227	1228	1231.2	0.081	1236.6	1228	1232.0	0.081	1687.0
TA08	1217	1217	1221.1	0.000	1030.1	1217	1220.2	0.000	968.4
TA09	1274	1274	1281.3	0.000	1258.6	1274	1279.5	0.000	1694.2
TA10	1241	1241	1248.3	0.000	901.4	1241	1246.2	0.000	1418.2
TA11	(1323, 1361)	1367	1371.9	3.326	1604.2	1365	1374.5	3.175	3122.7
TA12	(1351, 1367)	1374	1380.4	1.702	1577.0	1374	1387.2	1.702	2786.7
TA13	(1282, 1342)	1349	1361.1	5.226	1638.9	1350	1359.2	5.304	2902.3
TA14	1345	1345	1345.0	0.000	888.0	1345	1345.0	0.000	514.2
TA15	(1304, 1340)	1342	1359.1	2.914	1794.3	1350	1362.6	3.528	3174.3
TA16	(1302, 1360)	1362	1374.7	4.608	1284.2	1362	1370.1	4.608	3313.0
TA17	1462	1473	1482.5	0.752	1591.1	1470	1489.4	0.547	3620.5
TA18	(1369, 1396)	1403	1415.0	2.484	1733.2	1404	1417.2	2.557	3102.6
TA19	(1297, 1335)	1341	1352.6	3.392	1592.2	1341	1355.5	3.392	3045.8
TA20	(1318, 1351)	1352	1357.4	2.580	1630.8	1353	1359.1	2.656	3332.7
TA21	(1539, 1644)	1650	1661.5	7.212	2640.3	1647	1653.1	7.018	4158.4
TA22	(1511, 1600)	1601	1629.5	5.956	2346.7	1601	1620.6	5.956	3586.4
TA23	(1472, 1557)	1558	1570.1	5.842	2394.2	1560	1572.4	5.978	4175.7
TA24	(1602, 1647)	1648	1665.4	2.871	2036.6	1652	1659.5	3.121	3320.2
TA25	(1504, 1595)	1599	1608.5	6.316	2089.3	1597	1611.0	6.184	3654.3
TA26	(1539, 1645)	1655	1664.3	7.537	2178.6	1652	1669.7	7.342	3178.8
TA27	(1616, 1680)	1687	1694.6	4.394	2412.0	1686	1699.3	4.332	3523.8
TA28	(1591, 1614)	1618	1628.0	1.697	2233.2	1618	1625.4	1.697	3804.8
TA29	(1514, 1625)	1629	1637.2	7.596	2516.9	1627	1344.0	7.464	3324.9
TA30	(1473, 1584)	1592	1629.5	8.079	2347.4	1595	1617.2	8.282	4003.5
TA31	1764	1766	1769.0	0.113	2880.0	1764	1771.1	0.000	5155.3
TA32	(1774, 1796)	1822	1838.4	2.706	3547.7	1819	1836.5	2.537	5622.9
TA33	(1778, 1793)	1805	1813.9	1.519	3418.4	1808	1822.4	1.687	5829.6
TA34	(1828, 1829)	1832	1839.4	0.219	3900.6	1831	1850.7	0.164	5758.1
TA35	2007	2007	2007.3	0.000	240.4	2007	2007.0	0.000	799.3
TA36	1819	1823	1839.4	0.220	3030.5	1819	1840.9	0.000	5355.4
TA37	(1771, 1778)	1793	1798.6	1.242	3657.3	1791	1804.2	1.129	5548.6
TA38	1673	1677	1689.4	0.239	3188.2	1677	1691.3	0.239	6351.4
TA39	1795	1795	1805.2	0.000	2692.5	1797	1812.5	0.111	5547.6
TA40	(1631, 1674)	1688	1713.0	3.495	3608.5	1690	1711.4	3.617	6055.9
TA41	(1859, 2018)	2038	2051.8	9.629	4790.1	2015[a]	2041.3	8.392	10 860.4
TA42	(1867, 1956)	1961	1972.1	5.035	4343.0	1963	1982.1	5.142	9422.3
TA43	(1809, 1859)	1875	1899.3	3.648	4703.2	1872	1893.0	3.483	9792.9
TA44	(1927, 1984)	2011	2023.2	4.359	4419.7	2002	2019.3	3.892	10 007.8
TA45	(1997, 2000)	2001	2015.6	0.200	3736.5	2000	2021.7	0.150	11 207.2
TA46	(1940, 2021)	2047	2071.0	5.515	4990.8	2037	2066.9	5.000	9180.5
TA47	(1789, 1903)	1927	1949.7	7.714	4921.6	1928	1947.5	7.770	10 139.9
TA48	(1912, 1952)	1968	1989.3	2.929	4803.8	1967	1984.6	2.877	9354.1
TA49	(1915, 1968)	1989	2012.0	3.864	5119.2	1984	2007.5	3.603	10 159.5
TA50	(1807, 1926)	1956	1963.3	8.246	5201.9	1954	1972.6	8.135	9767.2
TA62	2869	2875	2897.2	0.209	14 842.7	2883	2912.4	0.488	31 547.9
TA67	2825	2825	2825.5	0.000	13 690.6	2825	2825.2	0.000	28 926.1
MRE				1.830				1.798	

[a]Indicates the new upper bound.

algorithms experimented on different machines. Overall, the results indicate that with the suggested algorithms very high quality solutions are obtained, often better than those reported for other algorithms. However, this comes at the cost of increased computational times.

Table 7 gives the comparison of ACOFT-MWR and ACOFT-TR with a novel heuristic algorithm proposed by Grabowski and Wodecki [4] (TSGW) and with other famous hybrid algorithms, TSSB and BV-best, on instances (c) in terms of MRE, $\overline{T}$, and numbers of optimal solutions obtained (No. OPT). Note that BV-best indicates the best results from all those SB-GLS series, provided by Balas and Vazacopoulus [22]. In addition, TSSB was coded in C and experimented on a Pentium 133 MHz PC, TSGW was coded in C + + and experimented on a Pentium 333 MHz PC, and SB-GLS was coded in C and experimented on a SUN SPARC-330 workstation. In this experiment, we obviously give much more computation effort than was allowed in TSSB, TSGW, and SB-GLS series; however, ACOFT-TR could obtain MRE of 1.798% and 40 optimal solutions that are superior to all the others.

Table 8 lists the computational results for instances (c), which include 80 instances, where TA51–TA80 are relatively large-sized instances but easy to solve (instances with $|J|$ much large than $|M|$ are easy), except for TA62 and TA67. Hence we just list the computational results of TA01–TA50, TA62, and TA67 and omit the easier ones. For these instances, it is observed that ACOFT-TR performs better than ACOFT-MWR. In particular, it is noted that ACOFT-TR found a new best upper bound 2015 for instance TA41.

6. Conclusions

In this paper, we have proposed a hybrid algorithm combining ACO with fast taboo for minimizing the makespan in the JSSP. To improve over a straightforward ACO algorithm, we have defined a specific pheromone trail definition inspired from SB and tested a dynamic greedy heuristic. This new way of defining the pheromone trails can also be applied in other shop scheduling problems. Besides, we have exercised the proximate optimality principle (POP) local search with a proposed partial disjunctive graph (PDG) to offer better searching guidance for artificial ants. Moreover, the traditional FT is accelerated by a modified makespan calculation.

To combine ACO with fast taboo, we have proposed a global pheromone update queue with the stores of a series of good schedules to update the pheromone trails diversely and thereby the history information can be utilized more effectively. Also, to offer robustness of ACOFT, the parameters of evaporation rate and relative proportion between the exploitation and exploration are adjusted dynamically, making the escape from a local minimum easy.

The proposed algorithm has been tested on 101 benchmark instances and compared with other algorithms. In sum, the solutions ACOFT can yield are often the same or slightly better than reported for best-performing algorithms for the JSSP; in particular, ACOFT has improved the best upper bound on one open benchmark instance (TA41). However, these results are obtained using higher computation times that used in most of the publications.

Acknowledgment

The authors are most grateful to the constructive comments of the anonymous referees that improved the presentation of the paper. This work is partially supported by the National Science Council, Taiwan, Republic of China, under grant number NSC 92-2213-E-011-058.

References

[1] Garey MR, Johnson DS. Computers and Intractability: a guide to the theory of NP-completeness. San Francisco, CA: Freeman and Company; 1979.
[2] Carlier J, Pinson E. An algorithm for solving the job-shop problem. Management Science 1989;35:164–76.
[3] Dell'Amico M, Trubian M. Applying tabu search to the job shop scheduling problem. Annals of Operations Research 1993;41:231–52.
[4] Grabowski J, Wodecki M. A very fast tabu search algorithm for the job shop problem. In: Rego C, Alidaee B, editors. Metaheuristic optimization via memory and evolution. Dordrecht: Kluwer Academic Publishers; 2004.
[5] Nowicki E, Smutnicki C. A fast tabu search algorithm for the job shop problem. Management Science 1996;42:797–813.
[6] Taillard ÉD. Parallel taboo search techniques for the job shop scheduling problem. ORSA Journal on Computing 1994;6:108–17.
[7] Croce FD, Tadei R, Volta G. A genetic algorithm for the job shop problem. Computers and Operations Research 1995;22:15–24.
[8] Schultz SR, Hodgson TJ, King RE. On solving the classic job shop makespan problem by minimizing Lmax. Raleigh, NC: Department of Industrial Engineering, North Carolina State University; 2004.
[9] Blazewicz J, Domschke W, Pesch E. The job shop scheduling problem. European Journal of Operational Research 1996;93:1–33.

[10] Jain AS, Meeran S. Deterministic job-shop scheduling: past, present and future. European Journal of Operational Research 1999;113:390–434.
[11] Vaessens RJM, Aarts EHL, Lenstra JK. Job shop scheduling by local search. INFORMS Journal on Computing 1996;8:302–17.
[12] Balas E, Vazacopoulos A. Guided local search with shifting bottleneck for job shop scheduling. Management Science 1998;44:262–75.
[13] Dorndorf U, Pesch E. Evolution based learning in a job shop scheduling environment. Computers and Operations Research 1995;22:25–40.
[14] Pezzella F, Merelli E. A tabu search method guided by shifting bottleneck for the job shop scheduling problem. European Journal of Operational Research 2000;120:297–310.
[15] Wang L, Zheng DZ. An effective optimization strategy for job shop scheduling problems. Computers and Operations Research 2001;28: 585–96.
[16] Yamada T, Nakano R. Job-shop scheduling by simulated annealing combined with deterministic local search. In: Meta-heuristics: theory and applications. Dordrecht: Kluwer Academic Publishers; 1996. p. 237–48.
[17] Adams J, Balas E, Zawack D. The shifting bottleneck procedure for job shop scheduling. Management Science 1988;34:391–401.
[18] Aiex RM, Binato S, Resende MGC. Parallel GRASP with path-relinking for job shop scheduling. Parallel Computing 2003;29:393–430.
[19] Dorigo M, Stützle T. Ant colony optimization. Cambridge, MA: MIT Press; 2004.
[20] 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.
[21] Bullnheimer B, Hartl RF, Strauss C. An improved ant system algorithm for the vehicle routing problem. Annals of Operations Research 1999;89:319–28.
[22] Bauer A, Bullnheimer B, Hartl RF, Strauss C. An ant colony optimization approach for the single machine total tardiness problem. In: Proceedings of the 1999 congress on evolutionary computation. New York: IEEE Press; 1999. p. 1445–50.
[23] den Besten M, Stützle T, Dorigo M. Ant colony optimization for the total weighted tardiness problem. In: Proceeding of PPSN VI, sixth international conference on parallel problem solving from nature. Lecture Notes in Computer Science 2000;1917:611–20.
[24] 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.
[25] Stützle T. An ant approach for the flow shop problem. Proceeding of EUFIT '98: sixth European congress on intelligent techniques and soft computing, vol. 3, 1998. p. 1560–4.
[26] 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.
[27] Colorni A, Dorigo M, Maniezzo V, Trubian M. Ant system for job shop scheduling. Belgian Journal of Operations Research 1994;34:39–53.
[28] Blum C, Sampels M. An ant colony optimization algorithm for shop scheduling problems. Journal of Mathematical Modelling and Algorithms 2004;3:285–308.
[29] Blum C. Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling. Computers and Operations Research 2005;32:1565–91.
[30] Blum C, Sampels M. Ant colony optimization for FOP shop scheduling: a case study on different pheromone representations. In: Proceedings of the 2002 congress on evolutionary computation (CEC'02), vol. 2. Los Alamitos, CA: IEEE Computer Society Press; 2002. p. 1558–63.
[31] Dorndorf U, Pesch E, Phan-Huy T. Constraint propagation and problem decomposition: a preprocessing procedure for the job shop problem. Annals of Operations Research 2002;115:125–45.
[32] Dauzère-Pérès S, Lasserre JB. A modified shifting bottleneck procedure for job-shop scheduling. International Journal of Production Research 1993;31:923–32.
[33] Balas E, Lenstra JK, Vazacopoulos A. The one-machine problem with delayed precedence constraints and its use in job shop scheduling. Management Science 1995;41:94–109.
[34] Fleurent C, Glover F. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. INFORMS Journal on Computing 1991;11:198–204.
[35] Binato S, Hery WJ, Loewenstern D, Resende MGC. A GRASP for job shop scheduling. In: Ribeiro CC, Hansen P, editors. Essays and surveys on metaheuristics. Dordrecht: Kluwer Academic Publishers; 2001. p. 59–79.
[36] Blum C. ACO applied to group shop scheduling: a case study on intensification and diversification. In: Dorigo M, Di Caro G, Sampels M, editors. Proceedings of ANTS 2002—from ant colonies to artificial ants: third international workshop on ant algorithms, Lecture notes in computer science, vol. 2463. Berlin, Germany: Springer; 2002. p. 14–27.
[37] Glover F. Tabu search (Part I). ORSA Journal on Computing 1989;1:190–206.
[38] Glover F. Tabu search (Part II). ORSA Journal on Computing 1990;2:4–32.
[39] Lawrence S. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement). Pittsburgh, PA: Graduate School of Industrial Administration, Carnegie Mellon University; 1984.
[40] Applegate D, Cook W. A computational study of the job-shop scheduling problem. ORSA Journal on Computing 1991;3:149–56.
[41] Fisher H, Thompson GL. Probabilistic learning combinations of local job-shop scheduling rules. In: Muth JM, Thompson GL, editors. Industrial scheduling. Englewood Cliffs, NJ, Chichester, UK: Prentice-Hill; 1963.
[42] Taillard ÉD. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64:108–17.
[43] Van Laarhoven PJN, Aarts EHL, Lenstra JK. Job shop scheduling by simulated annealing. Operations Research 1992;40:113–25.