Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Computers & Industrial Engineering 47 (2004) 181–193
computers & industrial engineering
www.elsevier.com/locate/dsw

Application of ant colony optimization for no-wait flowshop scheduling problem to minimize the total completion time☆

S.J. Shyu[a], B.M.T. Lin[b,*], P.Y. Yin[c]

[a]*Department of Computer Science, Ming Chuan University, Kwei-Shan, Tao-Yuan 333, Taiwan, ROC*
[b]*Department of Information and Finance Management, National Chiao Tung University, Hsin-Chu 300, Taiwan, ROC*
[c]*Department of Information Management, National Chi-Nan University, Pu-Li, Nan-Tou 545, Taiwan, ROC*

Received 30 January 2002; accepted 15 June 2004
Available online 11 September 2004

Abstract

Ant colony optimization (ACO) is a meta-heuristic proposed to derive approximate solutions for computationally hard problems by emulating the natural behaviors of ants. In the literature, several successful applications have been reported for graph-based optimization problems, such as vehicle routing problems and traveling salesman problems. In this paper, we propose an application of the ACO to a two-machine flowshop scheduling problem. In the flowshop, no intermediate storage is available between two machines and each operation demands a setup time on the machines. The problem seeks to compose a schedule that minimizes the total completion time. We first present a transformation of the scheduling problem into a graph-based model. An ACO algorithm is then developed with several specific features incorporated. A series of computational experiments is conducted by comparing our algorithm with previous heuristic algorithms. Numerical results evince that the ACO algorithm exhibits impressive performances with small error ratios. The results in the meantime demonstrate the success of ACO's applications to the scheduling problem of interest.

Keywords: Production; Scheduling; Meta-heuristics; Ant colony optimization

1. Introduction

In the operations research literature, many elegant mathematical models and solution methods have been developed to cope with real-world problems. Although some exact approaches, such as branch-and-bound algorithms and dynamic programs, have been proposed to solve the problems *optimally*,

☆ The authors of this research were partially supported by the National Science Council of the ROC under grants 91-2213-E-130-002, 91-2416-H-260-001 and 90-2213-E-130-006, respectively.
* Corresponding author. Tel.: +886-3-5131472; fax: +886-3-5729915.
E-mail address: bmtlin@mail.nctu.edu.tw (B.M.T. Lin).

0360-8352/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2004.06.006

the curse of dimensionality of numerous combinatorial problems hinders the search for optimal solutions in the solution space. Seeking approximate solutions that are feasible or good enough in an acceptable time therefore shapes a viable alternative for the decision makers. Following this orientation, several meta-heuristics, such as genetic algorithms (Goldberg, 1989; Holland, 1975), tabu search (Glover, 1990) and simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983), have been proposed to deal with the computationally intractable problems. Ant colony optimization (hereafter, the acronym ACO will be used through this paper) is a new meta-heuristic developed for composing approximate solutions. The ant algorithm was first proposed by Colorni, Dorigo, and Maniezzo (1991) and has been receiving extensive attention due to its successful applications to many combinatorial optimization problems (Dorigo, Di Caro, & Gambardella, 1999). Like genetic algorithm and simulated annealing approaches, the ant algorithms also foster its solution strategy through use of nature metaphors. The ACO is based upon the behaviors of ants that they exhibit when looking for a path to the advantage of their colony. Unlike simulated annealing or tabu search, in which a single agent is deployed for a single beam session, ACO and genetic algorithms use multiple agents, each of which has its individual decision made based upon collective memory or knowledge. Some basics of the ACO will be presented in later sections.

The problem under study in this paper is a two-machine flowshop scheduling problem to minimize the total completion time, i.e. the sum of completion times of all jobs, under no-wait constraints. This objective is a measurement that is commonly adopted to reflect the average flow time of all orders in the manufacturing or service shops. Since Johnson's seminal work concerning the minimization of makespan (Johnson, 1954), flowshop scheduling has been one of the most extensively studied topics in the scheduling literature (Reisman, Kumar, & Motwani, 1997). No-wait constraints arise from such situations as steel melting and molding where interruptions between operations on successive machines are not allowed. To minimize the makespan, Gilmore and Gomory (1964) proposed a polynomial time solution algorithm by transforming the scheduling problem into the traveling salesman problem (TSP) with specific structures. Under the same transformation scheme, the no-wait scheduling problem to minimize the total completion time is equivalent to the cumulative TSP, which is known to be strongly NP-hard (Garey & Johnson, 1979; Sahni & Gonzalez, 1976). The cumulative TSP seeks to find a Hamiltonian cycle such that the sum of distances from the source to all distinct cities is minimized. This problem also appears in different forms in different industrial applications (Busch, 1991; Simchi-Levi & Berman, 1991; Tsitsiklis, 1992). Fischetti, Laporte, and Martello (1993) formulated an integer programming model and developed lower bounds for the development of branch-and-bound algorithms. Their exact algorithm can solve problems with 35 jobs. They also proposed an approximate method based upon the 3-Opt heuristic (Lin, 1965). van Eijl (1995) proposed a mixed integer programming model and performed computational experiments for the cumulative TSP with time windows. For the problems in the form of flowshops, Rajendran and Chaudhuri (1990) proposed and tested some heuristic procedures for the manufacturing systems that contain two or more stages of machines. Aldowaisan and Allahverdi (1998) and Aldowaisan (2001) study a new variant where setup times separated from machine operations are considered. Setup times are required for such applications as tool-changeover for different product types. They proposed dominance rules and several heuristic procedures, based upon some preference rules, to deliver approximate schedules. In this study, we develop two ACO algorithms and compare the solution qualities with that produced by the heuristic procedures of Aldowaisan and Allahverdi, Aldowaisan (2001) and Fischetti et al. (1993). For a comprehensive survey on flowshop scheduling problems with no-wait or blocking constraints and their applications, the reader is referred to Hall and Sriskandarajah (1996). Scheduling problems involving setup cost/time have been receiving considerable attention since the past decade. The papers Allahverdi et al. (1999), Cheng, Gupta, and Wang (2000), and Potts and Kovalyov (2000) are the most recent reviews in this area.

The rest of this paper is organized as follows. In Section 2, we give a formal description of the scheduling problem and a graph-based representation to facilitate the development of ACO systems. Fundamental concepts of ACO systems will be given in Section 3. The features that we adopt and develop to be incorporated into the ACO will also be presented. Section 4 is dedicated to the design, implementation and results of our computational experiments. We implement the proposed ant algorithms and previous heuristic procedures and compare the quality of the produced solutions. Concluding remarks and discussions are given in Section 5.

2. Problem formulation and graph-based representation

In this section, we first describe the problem definition and introduce the notations that will be used throughout this paper. Then, a graph-based model for the scheduling will be formulated. The mathematical equivalence between the scheduling problem and the graph model will be illustrated through a numerical example.

In the problem of concern, there is a set of jobs $N=\{1,2,3,\ldots,n\}$ available from time zero onwards for processing in a two-machine no-wait flow-shop. Each job has two operations to perform on the two machines with a setup time required before each operation starts. The processing time of job i on machine k, $1\leq i\leq n$, $k=1$ or 2, is denoted by p_{ik}, and the setup time is both job- and machine-dependent and is denoted by s_{ik}, $1\leq i\leq n$, $k=1$ or 2. If the setup of a job on the second machine is completed, then its second operation must be started on the second machine immediately when the first operation is finished on the first machine. The setup for the second operation is *anticipatory*, i.e. it can be started no matter whether the first operation is finished or not. The completion time, C_i, of job i is the time when both operations of the job are finished. The objective function we study in this paper is to find a schedule (or permutation) of the jobs such that the total completion time, i.e. the sum of completion times of all jobs, is minimized. For the given job set, $Z(S)$ denotes the total completion time of schedule S. To illustrate the problem definition, we consider the three jobs given in Table 1 and two example schedules $S_1=123$ and $S_2=213$. Gantt charts for the two schedules are shown in Fig. 1. As schedule S_2 has a better utilization of both machines, it attains a better total completion time.

Following the standard three-field notation adopted by Graham, Lawler, Lenstra, and Rinnooy Kan (1979), we use $F2/nwt, setup/\Sigma C_i$ to represent the scheduling problem. The first field specifies our machine environment, a two-machine flowshop. The second field indicates the no-wait constraints and setup requirement. The last field represents the objective function to be optimized.

The problem of makespan minimization with no-wait constraints can be formulated as the well-known TSP. Although, in general, the TSP is strongly NP-hard, the graph derived from an input instance of $F2/nwt/C_{\max}$ exhibits special structures that lead to the existence of polynomial-time solution algorithms. With the attempt to minimize the total completion time, the graph-based model is also viable

Table 1
A set of three jobs

Job	1	2	3
p_{i1}	5	3	2
p_{i2}	4	4	2
s_{i1}	2	1	3
s_{i2}	3	3	1

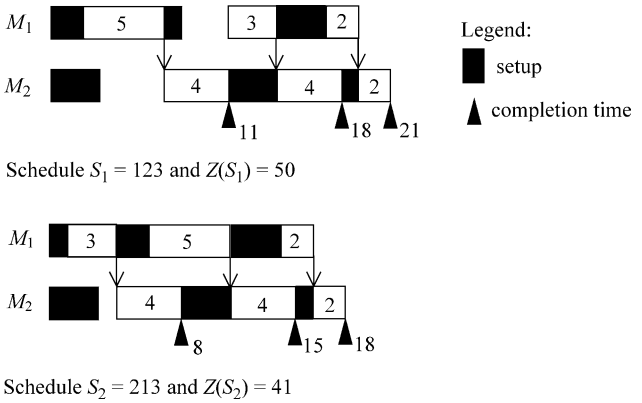

Fig. 1. Two sample schedules under no-wait constraints.

except that the problem turns out to be so-called cumulative TSP, which is equivalent to the strongly NP-hard problem of finding a cumulative Hamiltonian path (Sahni & Gonzalez, 1976). With setup considerations on both machines as a generalization of the original scheduling problem, we can reshape the transformation as follows. For the given set of n jobs, we construct a weighted directed graph $G(V, E)$ with $n+1$ nodes, $V=\{0,1,2,\ldots,n\}$ and $E=V\times V$. In graph G, node i in $V-\{0\}$ corresponds to job i in N. A weighting function w: $E\rightarrow \mathbf{R}^{+}$ on the edges is defined as:

$$\text{For } 1\leq i\leq n, w(0,i) = \max\{s_{i1}+p_{i1}, s_{i2}\}+p_{i2}; \tag{1}$$

$$\text{For } 1\leq i,j\leq n, w(i,j) = \begin{cases} (s_{j1}+p_{j1}-p_{i2})+p_{j2}, & \text{if } s_{j1}+p_{j1}\geq p_{i2}+s_{j2}; \\ s_{j2}+p_{j2}, & \text{otherwise;} \end{cases} \tag{2}$$

$$\text{For } 1\leq i\leq n, w(i,0) = \infty; \tag{3}$$

$$\text{For } 0\leq i\leq n, w(i,i) = \infty; \tag{4}$$

In the above formulation, Eq. (1) stands for the completion time of the job scheduled first. Eq. (2) defines the offset of completion times between job i and job j given that job j is the immediate successor of job i. Eq. (3) prevents any traverse into node 0, while Eq. (4) avoids self-loops recurred on nodes. With the weighted graph, the problem seeks a tour, starting from node 0, that traverses each node once and exactly once so that the cumulative total weight is minimum. With the above set of three jobs, we have the corresponding weighted graph with four nodes as shown in Fig. 2.

In the weighted graph, tour 0-1-2-3 has an accumulated weight $=11+(11+7)+(11+7+3)=50$, while tour 0-2-1-3 has an accumulated weight $=8+(8+7)+(8+7+3)=41$. The results correspond to the total completion times that we have derived for schedules $S_1=123$ and $S_2=213$.

3. Ant colony system

In this section, we first briefly review the basic principles of the ACO. The specific features of the ACO we develop for solving the *F*2/*nwt, setup*/ΣC_i problem will follow.

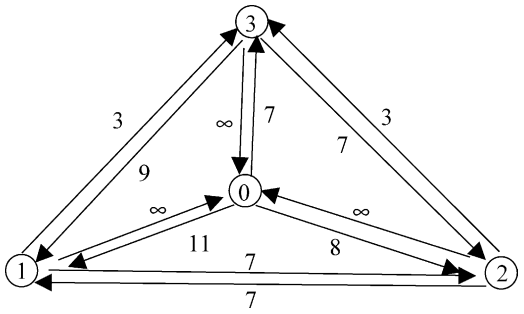

Fig. 2. Graphical representation for *F2/nwt, setup*/C_{max}.

3.1. Basics of ACO

The ACO was first proposed by Colorni et al. (1991) as a meta-heuristic scheme for finding near-optimal solutions. It has been successfully used to solve many complex problems, such as TSPs (Colorni, Dorigo, Maffioli, Maniezzo, Righini, & Trubian, 1996; Dorigo, Maniezzo, & Colorni, 1996; Dorigo & Gambardella, 1997; Stutzle & Dorigo, 1999), quadratic assignment problems (Gambardella, Taillard, & Dorigo, 1999), vehicle routing problems (Bullnheimer, 1999; Bullnheimer, Hartl, & Strauss, 2001), and production scheduling problem (Colorni, Dorigo, Maniezzo, & Trubian, 1994; Shyu, Yin, Lin, & Haouari, 2003; Shyu, Yin & Lin, 2004), just to name a few.

The ACO simulates the behaviors of real ants moving on a weighted connected graph and is able to solve many complex combinatorial optimization problems. The basic algorithm of the ACO introduced by Dorigo et al. (1999) and Dorigo et al. (1996) is outlined as the following:

Algorithm Ant_Colony_Optimization

1. Initialize.
   Represent the underlying problem by a weighted connected graph.
   Set initial pheromone for every edge.
2. Repeat
   2.1. For each ant do
      Randomly select a starting node.
      Repeat
         Move to the next node according to a node transition rule.
      Until a complete tour is fulfilled.
   2.2. For each edge do
      Update the pheromone intensity using a pheromone updating rule.
   Until the stopping criterion is satisfied.
3. Output the global best tour.

3.2. Specific features of the ACO for F2/nwt, setup/ΣC_i

3.2.1. Initialization

The initialization step of the ACO includes two parts: the problem's graph representation and the initial pheromone setting. First, as described in the previous section, the underlying problem is represented by a weighted directed graph $G(V, E)$, each edge (i, j) has an associated weight $w(i, j)$. Thus the problem's objective is reduced to that of the cumulative TSP. Second, unlike the strategy used by Dorigo et al. (1999), where every edge is given a constant quantity of initial pheromone, our proposed method initializes the pheromone on edges from the results obtained by a greedy heuristic algorithm. The greedy heuristic positions one ant on the first node, then the ant moves to the next unvisited node such that the connecting edge has the minimal weight. The process is repeated until every node is visited. To obtain a better initialization of pheromone intensity, the greedy heuristic is applied n times by letting every node be the starting node. In the world of real ants, shorter paths will retain more pheromone; analogously, in the ACO, the paths corresponding to better solutions should receive more pheromone and become more attractive. Let W_k be the cumulated weight of the complete tour fulfilled by ant k, which has selected node k as its starting node. The edge (i, j) is initialized by a quantity of pheromone, τ_{ij}, defined by

$$\tau_{ij} = \sum_{k=1}^{n} \Delta\tau_{ij}^{k}, \tag{5}$$

where

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{W_k}, & \text{if edge } (i,j) \text{ belongs to the tour performed by ant } k; \\ 0, & \text{otherwise;} \end{cases} \tag{6}$$

and Q is a given constant. The proposed pheromone initialization scheme can provide a certain degree of guidance from scratch and thus will speed up the convergence of the running sessions of the ACO.

3.2.2. Node transition rule

During the running session, the ants moving on the graph travel from node to node through the edges. Because no node can be visited twice, we put the nodes that are already visited in a tabu list and mark them as inaccessible to prohibit the ants from visiting any node more than once. For the k-th ant on node i, the selection of next node j to follow is probabilistically determined according to the node transition probability,

$$\phi_{ij}^{k} = \begin{cases} \dfrac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{h \notin tabu_k}(\tau_{ih})^{\alpha}(\eta_{ih})^{\beta}}, & \text{if } j \notin tabu_k; \\ 0, & \text{otherwise;} \end{cases} \tag{7}$$

where τ_{ij} is the current pheromone intensity on edge (i, j), η_{ij} is the value of visibility, defined by $1/w(i, j)$, on edge (i, j), α and β are parameters controlling the relative importance of global and local preferences for edge (i, j), and $tabu_k$ indicates the current set of nodes inaccessible from ant k. The pheromone is a sort of long-term memory that records previous experiences of the ants about the global

preference for edge (i, j) according to the quality of the solutions found. On the other hand, the visibility is a sort of short-term memory that reflects the local preference for edge (i, j). The visibility is simply determined in a greedy fashion taking into account the local information about the weight $w(i, j)$ on edge (i, j) only. Therefore, the probabilistic node transition rule is a trade-off between the degree of intensification and diversification of the solution searching process in the solution space.

In this paper, we adopt a two-level probabilistic strategy developed by Dorigo et al. (1999) for the ACO to determine the next node to transit to. In this strategy, the algorithm first draws a random number from the interval [0, 1]. Depending on a threshold parameter λ, the algorithm triggers either *exploitation* or *biased exploration* strategy. In exploitation (where the random number generated is smaller than or equal to λ), ant k will select from the accessible neighbors the node with the largest value of ϕ_{ij}^k. In biased exploration (where the random number generated is greater than λ), the probability of selecting node j out of the accessible neighborhood of node i is given in Eq. (7). Therefore, another random number is generated from the interval [0, 1] to determine, in roulette wheels, which accessible node to visit next.

3.2.3. Pheromone updating rule

Once all of the ants have completed their tours (which is called a cycle), the intensity of pheromone on each edge is updated by the pheromone updating rule,

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \sum_{k=1}^{n} \Delta\tau_{ij}^k, \tag{8}$$

where ρ is the evaporation rate of previous pheromone intensity and is chosen between 0 and 1, n is the number of distributed ants and also the number of jobs, and $\Delta\tau_{ij}$ is the pheromone currently laid by ant k and is calculated by Eq. (6). In Dorigo and Gambardella (1997), a local pheromone updating policy is deployed. That is, an ant leaves reinforcement pheromone immediately when it travels through an edge. It is a natural implication of the real world. From the experiences attained from our preliminary experiments, incorporating the local update rule however did not provide significant improvements. Therefore, we use global updating rules only in our setting.

3.2.4. Hybridized with a local optimizer

Since the quality of the solutions delivered by local heuristics heavily depends on the initialization setting, there is a trend to embed local heuristics within meta-heuristics so that the meta-heuristics could provide good initializations for the local heuristics to start with (Dorigo et al., 1996). In our development, the best tour attained at the end of each cycle will be further improved by deploying a local optimizer. The local optimizer tentatively swaps each pair of nodes on the best tour and then picks up the node-pair that will be the most beneficial, i.e. will result in the best improvement. The greedy optimization process continues until no further improvement is attainable. The pheromone on the edges of the newly obtained best tour is therefore further reinforced by a quantity of Q/W_{best}, where W_{best} is the accumulated weight of the best tour. Then, the execution of another new cycle starts over.

3.2.5. Stopping criterion

The stopping criterion of the ACO could be a maximum number of iterative cycles, specified CPU time limit, or maximum number of cycles between two improvements of the global best solution. In this paper, we use a given number of iterative cycles as the stopping criterion. Although, the convergence

concept is commonly adopted in meta-heuristics, such as in simulated annealing, to determine if the iteration process should stop or not, we found it does not necessarily boost the performance in our ACO implementation. Therefore, in our experiment setting, the algorithm will terminate when a given number of cycles have been executed.

4. Computational experiments

In this section, we describe the computational experiments we used in order to evaluate the effectiveness of the application of the ACO to no-wait flowshop scheduling problems. Aldowaisan and Allahverdi (1998) and Aldowaisan (2001) developed several heuristic algorithms for deriving approximate solutions in a reasonable time. The approach exerted in these algorithms is first determining job sequences based upon different preference rules and then applying dominance properties to improve the schedules. We implemented all of the algorithms and for convenience take the best solution as the output. We call the aggregate version as *Algorithm AA*. Our ant algorithm is denoted as *Algorithm ACO*. And, *Algorithm ACO_Loc* stands for the variant with a local optimizer. To consider the problem from the aspect of cumulative TSP, we also implemented the heuristic proposed by Fischetti et al. (1993). This heuristic is denoted by Algorithm 3-OPT to reflects its basic 3-OPT operation.

The platform of our experiments is a personal computer with a Pentium-III 1.6 Hz CPU and 256 MB RAM. The programs are coded in Java. Job processing times, p_{ij}'s, were randomly selected from the interval [0, 100]. To contrast potential impacts that setup times may have on the effectiveness of the algorithms, we drew the values of setup time, s, randomly from different intervals [0, 10], [0, 50] and [0, 100]. The experiments consist of two parts. In part one, *Algorithms AA, ACO, ACO_Loc*, 3-*OPT* and a simple branch-and-bound algorithm were implemented. The problem size n is set as 6, 8, 10 and 12. For each combination of n and s, ten input instances were generated. The numerical results are averaged through each ten instances. In the ant algorithms, we set the number of ants as 6, number of iterations as 200, evaporation rate ρ as 0.1, exploitation/biased-selection parameter λ as 0.1, parameters α and β for transition rules as 2 and 1, respectively.

The numerical results are summarized and tabulated in Table 2. The column *BaB_Time* contains the average time required for the branch-and-bound algorithm to produce optimal solutions; *Ini_sol* indicates the computational results about the simple heuristic that produces initial solutions for the ant algorithms to initialize pheromone intensity. We keep track of the average objective value and average elapsed running time delivered by each method. The average ratios are calculated by $(Z_H - Z^*)/Z^* \times 100\%$, where Z_H is the total completion time given by method H, and Z^* is the optimal value given by the branch-and-bound algorithm. The columns entitled '#*Opt*' record the number of optimal solutions encountered by the algorithms through the ten input instances. From the results, we can find that the heuristic based upon 3-OPT for cumulative the TSP demonstrates better performances for these small-size instances. The two ant algorithms consistently outperform *Algorithm AA*. However, the numbers of optimal solutions that all the methods have encountered decrease as the problem size grows.

The next issue of interest is the scalability of ant algorithms, i.e. to determine if the ant colony algorithms are capable of producing quality solutions in a reasonable time for large-scale instances. In the experimental setting, the problem sizes are 50, 100, 150, 200 and 250. As the execution of the branch-and-bound algorithm is quite time-consuming for large-scale problems, we deploy *Algorithms*

Table 2
Experimental results for small scale problems

N	*s*	*BaB_-Time*	*Opt_-Sol*	Initial		*ACO*			*ACO_Loc*			3-*OPT*			*AA*		
				Solution	Error (%)	Solution	Error (%)	#Opt	Solution	Error (%)	#Opt	Solution	Error (%)	#Opt	Solution	Error (%)	#Opt
8	10	0.012	2171.3	4580.2	110.94	2251.3	3.68	2	2193.8	1.04	3	2187.1	0.73	6	2345.9	8.04	1
	50	0.016	3122.9	6522.6	108.86	3182.9	1.92	1	3141.3	0.59	5	3129.8	0.22	7	3241.6	3.80	1
	100	0.015	3968.5	8254.2	107.99	4082.4	2.87	2	3976.2	0.19	6	3976.0	0.19	7	4089.1	3.04	0
10	10	0.868	3226.0	6759.4	109.53	3330.5	3.24	0	3245.9	0.62	5	3228.4	0.07	4	3388.3	5.03	0
	50	1.091	4006.3	8298.6	107.14	4120.2	2.84	0	4050.2	1.10	3	4014.1	0.19	7	4239.5	5.82	0
	100	1.551	5710.9	12,042.0	110.86	6005.4	5.16	0	5759.8	0.86	3	5729.9	0.33	4	6020.4	5.42	0
12	10	106.031	4687.7	9805.2	109.17	4878.2	4.06	0	4719.4	0.68	0	4706.9	0.41	3	4903.6	4.61	0
	50	131.058	5884.3	12,445.8	111.51	6162.7	4.73	0	5927.5	0.73	3	5916.7	0.55	3	6154.9	4.60	0
	100	147.858	8251.6	17,322.4	109.93	8623.3	4.50	0	8297.6	0.56	2	8290.4	0.47	2	8872.7	7.53	0

Table 3
Numerical results for large scale problems

n	*s*	*ACO_Loc*		*ACO*			3-*OPT*		*AA*	
		Solution	Time	Solution	Time	Deviation (%)	Solution	Deviation (%)	Solution	Deviation (%)
50	10	63,976.5	0.748	68,324.1	0.454	6.80	63,663.5	−0.49	72,138.3	12.76
	50	87,384.2	0.693	91,773.5	0.44	5.02	87,280.4	−0.12	96,264.4	10.16
	100	122,130.1	0.690	127,506.5	0.569	4.40	121,510.4	−0.51	134,833.8	10.40
100	10	240,672.1	6.620	258,508.2	1.982	7.41	249,430.8	3.64	281,698.9	17.05
	50	346,841.5	5.826	364,489.9	2.220	5.09	354,968.7	2.34	388,539.7	12.02
	100	479,551.1	6.069	499,564.1	1.939	4.17	489,288.8	2.03	530,893.8	10.71
150	10	542,957.9	31.246	583,537.2	5.408	7.47	569,500.2	4.89	632,878.0	16.56
	50	773,577.8	28.461	814,150.0	4.719	5.24	799,684.7	3.37	877,652.5	13.45
	100	1,074,551.2	27.228	1,114,050.5	4.310	3.68	1,097,094.7	2.10	1,197,940.8	11.48
200	10	95,9106.1	81.878	1,046,035.3	11.216	9.06	1,021,875.5	6.54	1,136,283.2	18.47
	50	1,396,873.0	80.055	1,473,009.7	10.415	5.45	1,449,308.5	3.75	1,566,054.0	12.11
	100	2,402,063.5	88.968	2,460,088.6	11.987	2.42	2,439,362.4	1.55	2,674,909.3	11.36
250	10	1,483,044.7	204.685	1,579,907.4	15.672	6.53	1,563,153.1	5.40	1,775,079.8	19.69
	50	2,218,980.6	244.672	2,354,657.7	12.237	6.11	2,317,802.0	4.45	2,546,022.9	14.74
	100	3,015,346.1	257.931	3,149,833.4	12.908	4.46	3,104,329.1	2.95	3,377,454.4	12.01

AA, 3-*OPT*, *ACO*, and *ACO_Loc* only in the second part. Ten sets of jobs were also generated for each combination of n and s. Preliminary experiments showed that it is time-consuming for *Algorithm* 3-*OPT* to reach a local optimum. It is mainly due to a neighborhood of $O(n^3)$. Therefore, we used the time elapsed by *Algorithm ACO_Loc* as a limit of execution time for 3-*OPT*. For comparisons of solution values, the output produced by *Algorithm ACO_Loc* was deployed as the basic metrics. Therefore, we kept track of deviations of results of the other three algorithms. The numerical results shown in Table 3 include the objective values and execution times, in seconds, of the algorithms. The execution of *Algorithm AA* is relatively fast, so its elapsed time is not shown in the table. From the numerical results, we readily find that *Algorithm* 3-*OPT* performs well when the number of jobs is 50. Its performance is however rather diminished by as problem size becomes larger. As for ant algorithms, we see that *Algorithm ACO_Loc* consistently delivers quality solutions more than *Algorithm AA*. Of course, *Algorithm ACO_Loc* takes more time in composing solutions. If we resort to *Algorithm ACO* without incorporating local optimizer, the ant colony system still provides better solutions than the heuristics developed by Aldowaisan and Allahverdi (1998) and Aldowaisan (2001). Different from simple rules of thumb, *Algorithms ACO* and *ACO_Loc* are both meta-heuristics and thus take a relatively longer time converging to a steady state to report a solution. It is reasonable for decision makers to spend a few seconds or minutes using *Algorithms ACO* and *ACO_Loc* to derive quality solutions.

5. Concluding remarks

In this paper, we have considered the no-wait two-machine flowshop problem of optimizing the total completion time, which is known to be strongly NP-hard. To adapt the ACO for solving the scheduling problem, we have considered a transformation of the original problem into a graph-based model. We have developed two versions of the ACO system, i.e. either with or without local optimizers when a complete tour in the graph is fulfilled. Computational experiments have been designed and performed to demonstrate the potential applicability of the ACO system to the scheduling problem. Numerical results have shown that the ACO algorithms outperform previous algorithms. In the mean time, the outcome has also revealed that the proposed ACO algorithms are effective and robust in dealing with the scheduling under study.

Although the applications to dealing with complex combinatorial problems so far have demonstrated the effectiveness and robustness of the ACO, there is still considerable room for further development. For example, incorporating different styles of convergence, pheromone updating rules and colony relationship are worthy of further research. Another direction of potential interest may be in the study of non-graph-based models.

Acknowledgements

The authors are grateful to the anonymous referees for their constructive comments that have greatly improved the presentation of this paper.

References

Aldowaisan, T. (2001). A new heuristic and dominance relations for no-wait flowshops with setups. *Computers and Operations Research*, *28*, 563–584.

Aldowaisan, T., & Allahverdi, A. (1998). Total flow time in no-wait flowshops with separated setup times. *Computers and Operations Research*, *25*, 757–765.

Allahverdi, A., Gupta, J. N. D., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, *27*, 219–239.

Bullnheimer, B., (1999). Ant Colony Optimization in Vehicle Routing. Doctoral thesis, University of Vienna, Austria.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (2001). An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, *89*, 319–328.

Busch, I.K., (1991). Vehicle Routing on Acylic Networks. PhD Dissertation. Johnsons Hopkins University, Baltimore, MD.

Cheng, T. C. E., Gupta, J. N. D., & Wang, G. (2000). A review of flowshop scheduling research with setup times. *Productions and Operations Management*, *9*, 262–282.

Colorni, A., Dorigo, M., Maffioli, F., Maniezzo, F., Righini, G., & Trubian, M. (1996). Heuristics from nature for hard combinatorial problems. *International Transactions on Operations Research*, *3*, 1–21.

Colorni, A., Dorigo, M., & Maniezzo, V. (1991). In F. J. Varela, & P. Bourgine, *Distributed optimization by ant colonies*. *Proceedings of the First European Conference on Artificial Life, Paris*.

Colorni, A., Dorigo, M., Maniezzo, M., & Trubian, M. (1994). Ant system for job-shop scheduling. *JORBEL—Belgium Journal of Operational Research, Statistics and Computer Science*, *34*, 39–53.

Dorigo, M., Di Caro, G., & Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial Life*, *5*, 137–172.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computing*, *1*, 53–66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics: B*, *26*, 29–41.

Fischetti, M., Laporte, G., & Martello, S. (1993). The delivery man problem and cumulative matroids. *Operations Research*, *41*, 1055–1064.

Gambardella, L. M., Taillard, E., & Dorigo, M. (1999). Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, *50*, 167–176.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freedman.

Gilmore, P. C., & Gomory, R. E. (1964). Sequencing on a one-state variable machine: a solvable case of the traveling salesman problem. *Operations Research*, *12*, 655–679.

Glover, F. (1990). Tabu search: a tutorial. *Interfaces*, *20*, 74–94.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Learning*. New York: Addison Wesley.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, *5*, 287–326.

Hall, N. G., & Sriskandarajah, C. (1996). A survey of machine scheduling problems with locking and no-wait in process. *Operations Research*, *44*, 510–525.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, *1*, 61–67.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell Systems Technology Journal*, *44*, 2245–2269.

Potts, C. N., & Kovalyov, M. K. (2000). Scheduling with batching: a review. *European Journal of Operational Research*, *120*, 228–249.

Rajendran, C., & Chaudhuri, D. (1990). Heuristic algorithms for continuous flow-shop problem. *Naval Research Logistics*, *37*, 695–705.

Reisman, A., Kumar, A., & Motwani, J. (1997). Flowshop scheduling/sequencing research: a statistical review of the literature, 1952–1994. *IEEE Transactions on Engineering Management*, *44*, 316–329.

Sahni, S., & Gonzalez, T. (1976). P-complete approximation problems. *Journal of the ACM*, *23*, 555–565.

Shyu, S. J., Yin, P. Y., Lin, B. M. T., & Haouari, M. (2003). Ant-tree: an ant colony optimization approach to the generalized minimum spanning tree problem. *Journal of Experimental and Theoretical Artificial Intelligence*, *15*, 103–112.

Shyu, S. J., Yin, P. Y., & Lin, B. M. T. (2004). An ant colony optimization algorithm for the minimum weight vertex cover problem. *Annals of Operations Research*, *131*, 283–304.

Simchi-Levi, D., & Berman, O. (1991). Minimizing the total flow time of n jobs on a network. *IIE Transactions*, *23*, 236–244.

Stutzle, T., & Dorigo, M. (1999). ACO algorithms for the traveling salesman problem. In K. Miettinen, M. Makela, P. Neittaanmaki, & J. Periaux (Eds.), *Evolutionary Algorithms in Engineering and Computer Science*. New York: Wiley.

Tsitsiklis, J. (1992). Special cases of traveling salesman and repairman problems with time windows. *Networks*, *22*, 263–282.

van Eijl, C. A. (1995). *A Polyhedral Approach to the Delivery Man Problem. Technical Report 95-19, Department of Mathematics and Computer Science*. The Netherlands: Eindhoven University of Technology.