Applying Ant Colony Optimization (ACO) algorithm to dynamic job shop scheduling problems

Rong Zhou, Heow Pueh Lee*
and Andrew Y.C. Nee

Department of Mechanical Engineering,
National University of Singapore,
9 Engineering Drive 1, 117576, Singapore
E-mail: mpeleehp@nus.edu.sg
*Corresponding author

Abstract: Ant Colony Optimization (ACO) is applied to two dynamic job scheduling problems, which have the same mean total workload but different dynamic levels and disturbing severity. Its performances are statistically analysed and the effects of its adaptation mechanism and parameters such as the minimal number of iterations and the size of searching ants are studied. The results show that ACO can perform effectively in both cases; the adaptation mechanism can significantly improve the performance of ACO when disturbances are not severe; increasing the size of iterations and ants per iteration does not necessarily improve the overall performance of ACO.

Keywords: dynamic job shop scheduling; Ant Colony Optimization; ACO.

Reference to this paper should be made as follows: Zhou, R., Lee, H.P. and Nee, A.Y.C. (2008) 'Applying Ant Colony Optimization (ACO) algorithm to dynamic job shop scheduling problems', *Int. J. Manufacturing Research*, Vol. 3, No. 3, pp.301–320.

Biographical notes: R. Zhou is a PhD student in the Department of Mechanical Engineering, National University of Singapore. She received a BE in Mechanical Engineering in 1994 from the South China University of Technology. Her research interests are in the areas of production scheduling, artificial intelligence, multi-agent systems and simulation, and distributed manufacturing systems.

H.P. Lee is an Associate Professor in the Department of Mechanical Engineering, National University of Singapore, as well as the Deputy Executive Director for Research, Institute of High Performance Computing. He received his BA and MA from Cambridge, his ME from the National University of Singapore as well as a MS and PhD from Stanford University. His research interests are in biomechanics and nanomechanics.

A.Y.C. Nee is a Professor of Manufacturing Engineering, Department of Mechanical Engineering, National University of Singapore. He received his PhD and DEng from the University of Manchester and UMIST, respectively. His research interests are in computer applications to tool, die, fixture design and planning, distributed manufacturing systems, virtual and augmented reality applications in manufacturing. He is a fellow of CIRP and an elected fellow of the Society of Manufacturing Engineers (USA), both in 1990. He had held appointments as Head of the Department of Mechanical Engineering, Dean of Faculty of Engineering and currently, he is the Director of Research of NUS.

1 Introduction

A job shop manufacturing system is specifically designed to simultaneously process different types of products in one shop floor. Jobs have their own technical orders and a pre-determined processing time for each one of their operations. They route in the shop floor to be processed on machines till all of their operations are completed. A Job Shop Scheduling Problem (JSSP) refers to the static problem where optimal schedules are searched for a given set of jobs. It is generally NP-hard (Garey and Johnson, 1979). In the real world, a JSSP becomes dynamic when jobs arrive continuously and it thus has an additional complexity. Continuing with production in the face of dynamic events, while optimising the system performance, remains a challenge.

Industry has generally combined pragmatic approaches like Manufacturing Resource Planning (MRP II) or Enterprise Resource Planning (ERP) for medium-term production planning and then dispatching rules for short-term operational resource allocation. Dispatching rules are widely adopted for their robustness. However, they do not guarantee the realisation of the full potential of a shop floor since their decision making is based on local and current conditions. As computer capability is improved greatly, scheduling systems using algorithms, especially metaheuristic algorithms, have continuously been a research topic for providing optimised solutions. For example, the Genetic Algorithm (GA) has been applied to dynamic JSSPs by Lin et al. (1997) and others.

Ant Colony Optimization (ACO) is another metaheuristic proposed by Dorigo et al. (1999) and it has been applied to JSSP by Colorni et al. (1994) and van der Zwaan and Marques (1999). However, its applications in dynamic JSSP are limited. The goal of this study is to apply ACO to two dynamic JSSPs exploring its unique property of seeking solutions through the adaptation of its pheromone matrix. The inspiration is from the phenomenon that ants will not go back to their nest to restart searching a new route when the existing one is not available; instead, they search for another shortest route based on the current pheromone information.

This study aims to analyse the performances of a basic version of ACO on two dynamic JSSPs. The importance of its adaptation mechanism, which is applied in the procedure of updating the pheromone matrix, is also studied. Furthermore, the effects of two important parameters: the minimal number of iterations and the size of searching ants per iteration are investigated.

The outline of the paper is as follows: in Section 2, a literature review of ACO on scheduling related problems is given; in Section 3, the analysis of dynamic scheduling problem is given; in Section 4, the application of ACO to dynamic JSSP is described. In Section 5, the design of experiments is provided and computational results are given in Section 6. Finally, conclusions and further studies are discussed in Section 7.

2 Literature review

ACO is a general term for ant-based algorithms used for solving NP-hard combinatorial Optimisation problems. Its introduction can be found in Dorigo et al. (1996, 1999), Dorigo and Gambardella (1997a, 1997b), Dorigo and Di Caro (1999) and Dorigo and Stützle (2004).

ACO was first introduced to solve JSSP by Colorni et al. (1994) and it was successful in finding solutions within 10% of the optima for both instances of 10×10 and 10×15 JSSPs (Dorigo et al., 1996). However, despite showing the viability of the approach, the computational results are not competitive with the state-of-the-art algorithms (Stützle and Dorigo, 1999). Then it was applied to three benchmark JSSPs by van der Zwaan and Marques (1999). The schedules were within 8% and 26%, respectively, of the best known optima for the $10/10/G/C_{max}$ Muth-Thompson (10×10 job-shop scheduling problem: ten machines and ten jobs) and the $20/10/G/C_{max}$ Lawrence problems, which are the classic JSSPs for optimising makespan and are known to have optimal solutions recorded in OR-Library. The authors considered the results promising since the tests were only partially executed with an iteration number of 2000. The study also presented the importance of parameter settings.

ACO has also been applied to dynamic JSSPs. The main concern here is about the updating of the problem graph and the pheromone matrix, which are the main procedures taking computational space and time. Thus, the strategies to modify the pheromone matrix become the main topic. Vogel et al. (2002) proposed a continuously operating the Ant Algorithm, which could easily adapt to sudden changes in the production system. A Position-Operation-Pheromone-matrix and an allocation table are maintained. Pheromone values were reset whenever there was a change. The dynamic ACO was tested on the data collected from a real-world practice for two months and was compared to a manual, a priority-rule and the GA approaches. The result generated by ACO was only inferior to the GA approach for the same problem.

The applications of ACO in the dynamic Travelling Salesman Problem (TSP) using the adaptation mechanism of the pheromone matrix (Angus and Hendtlass, 2002; Guntsch and Middendorf, 2001; Guntsch et al., 2001; Guntsch and Middendorf, 2002) also inspired the current study. The result of Angus and Hendtlass (2002) indicates that a new route found through adaptation is significantly faster than those found through starting searching all over again. The experimental results of Guntsch and Middendorf (2002) also showed that the ACO keeping previous information performs superior than the approach of restarting, all over again, the procedure upon dynamic events.

In summary, the applications of ACO in dynamic JSSP are few and it is not clear how ACO performs in dynamic JSSP. The promising findings in dynamic TSPs inspire the current studies.

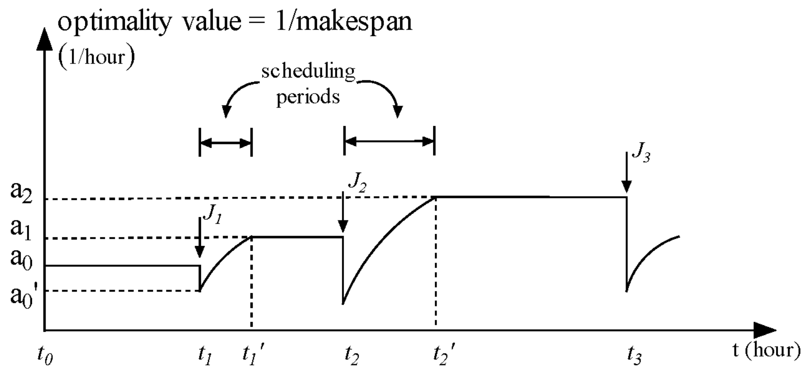
3 Analysis of dynamic scheduling problem

The dynamism of a scheduling problem is usually treated following the approach of a rolling time horizon (Raman and Talbot, 1993) where a scheduling problem consisting of all known jobs is solved. When a new job arrives at time *t*, the part of the solution consisting of operations already started before *t* is fixed and a new problem is constructed, consisting of the backlog to be starting after time *t*, plus all the operations from the newly arrived job. The dynamic problem is thus decomposed into a series of static intermediate problems over time (Branke, 2002). The following analysis is based on following assumption.

- the performance objective is makespan, which is the 'length' of the schedule, or an interval between the time at which the schedule begins and the time at which the schedule ends
- the unexpected incoming job is the only type of dynamic event in the job shop
- the length of processing time is counted in hours.

The optimality values of intermediate schedules over time in a dynamic environment can be illustrated in Figure 1, with the optimality value formulated as 1/*makespan* so that the minimum makespan means the maximal optimality. In the figure, a schedule with an optimality value of a_0 has been executed from time t_0 to t_1, when a new job J_1 arrives. The optimality of the current schedule immediately drops to a_0' if job J_1 is simply put at the end of the schedule. Then a rescheduling procedure is triggered to form an intermediate problem with the backlog operations and all of the operations from job J_1 assuming that the scheduling period allowed is $[t_1, t_1']$. A new schedule with an optimality value of a_1 is generated and executed from t_1' till the second job J_2 arrives at time t_2, where a similar procedure repeats.

Figure 1 The optimality values of schedules over time in a dynamic environment



The optimality of a schedule for a given performance measure found in the time intervals of $[t_1, t_1']$ or $[t_2, t_2']$ can be affected by the following factors:

- the length of the computational time slot
- the size of an intermediate problem
- the quality of the scheduling algorithm
- dynamic scheduling strategies.

3.1 The length of a computational time slot

A computational time slot refers to the time span that can be allowed for searching a new intermediate schedule. Its length is problem-dependent; for example, the computational time slot for the intermediate problem caused by job J_1 can be decided by its travelling time from the reception area to its first workcentre. The length may proportionally affect the optimality of an intermediate schedule.

3.2 The size of an intermediate problem

Given the same length of a computation time slot, a smaller scheduling problem implies lower computational cost and a better solution and vice versa. A schedule minimising makespan may have a better opportunity to complete more operations before an interruption occurs. Thus, the resulting intermediate problem can have a smaller size and hence a better chance to find a good schedule, which facilitates the generation of another good schedule in the following disturbed moment. On the contrary, a greater intermediate problem may have less opportunity to find a good schedule. The poor schedule, in turn, may produce a larger intermediate problem at the next disturbing moment. As the procedure goes on, the overall performance of the scheduling system may deteriorate.

3.3 The quality of a scheduling algorithm

A good scheduling algorithm should generate a timely and satisfactory schedule to guide production. Information adaptation can speed up the procedure of finding a new optimum as mentioned in Section 2. The idea is to generate a schedule, not from scratch, but to exploit the optimal information kept in the current solution and quickly find a good solution for the modified problem. This adaptation also has the advantage of maintaining similarity between two continual schedules, which is preferred in real life applications.

3.4 Dynamic scheduling strategies

Dynamic scheduling strategies involve choosing the scheduling frequency or employing partial scheduling. Scheduling frequency refers to how often the rescheduling procedure is triggered. It can be event-driven, periodic or performance-driven. The event-driven approach triggers a rescheduling procedure whenever an event occurs; the periodic approach triggers the rescheduling procedure according to a pre-set time period; the performance-driven approach uses performance values of current production system as the initiator of the rescheduling procedure. These approaches essentially solve different dynamic scheduling problems where the last two alter the original problem by postponing the reactions to interrupters.

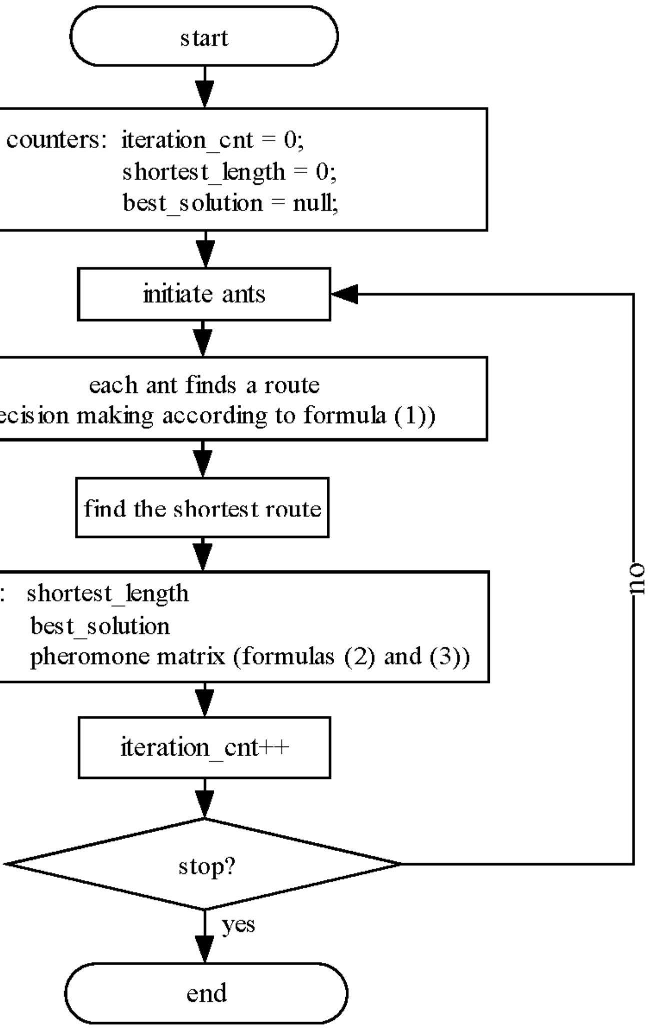
Partial scheduling considers only a partial set of jobs from an intermediate problem in a computing interval in order to cover the next estimated execution period. This approach is inspired by the fact that a schedule may not have an opportunity to be fully executed before dynamic interrupts; thus, there is no need to include the operations that may not be processed before those interruptions. Partial scheduling can save computational costs but its solutions may lack a wide view.

4 ACO applied to dynamic job shop scheduling problem

4.1 ACO algorithm

The flow chart of the ACO algorithm is given in Figure 2. The basic idea is to repetitively initiate a set of ants to walk in a common environment (problem graph). Each ant walks through all of those operations (nodes) one by one and thus forms a route, which can be interpreted as schedules and its length can represent the value of some performance measures, like makespan, in this study. The goal of each ant is to minimise this value.

Figure 2 The flow chart of ACO algorithm



A walking ant leaves behind itself, on its route, some amount of pheromone, which changes the global environment. The probability for an ant to choose its next node is directed by both the pheromone amount on the route and the distance from its current location to the targeted one. Ant *i* chooses the next node according to the State Transition Rule in formula (1) (Dorigo et al., 1996).

$$p_{ij}(h) = \frac{(\tau_{ij}(h))^{\alpha}\left[1/d_{ij}\right]^{\beta}}{\sum_{j\in\text{allowed-nodes}}(\tau_{ij}(h))^{\alpha}\left[1/d_{ij}\right]^{\beta}}. \tag{1}$$

h: iteration index

τ_{ij}: quantity of pheromone on the edge

d_{ij}: heuristic distance between nodes i and j

p_{ij}: probability to travel from node i to node j.

The parameters α and β tune the relative importance of the pheromone and the heuristic distance in decision making. The heuristic distance d_{ij} in this study is the sum of the travelling time between the current workcentre to the target one and the processing time of the operation in the target workcentre. The environment is represented by a pheromone matrix, which is updated by the best solution at each iteration. The updating can be described in formulae (2) and (3) (Dorigo et al., 1996).

$$\tau_{ij}(h+1) = (1-\rho)\cdot\tau_{ij}(h) + \Delta\tau_{ij}(h+1). \tag{2}$$

$$\Delta\tau_{ij}(h+1) = \begin{cases} \dfrac{Q}{f_{\text{evaluation}}(\text{best\_so\_far})} \\ 0, \quad \text{otherwise} \end{cases}. \tag{3}$$

ρ: evaporation coefficient

Q: quality of pheromone per unity of distance.

h is the iteration index to indicate the number of iterations that sets of ants have been initiated. ρ is the evaporation coefficient, which can be a real number between 0 and 1.0. Pheromones on all edges evaporate at the rate of ρ so as to diversify the searching procedure into a bigger solution space or out of local optima. Meanwhile, the best solution kept so far also maintains some good features worthy of further exploration. This information can be used to intensify certain searching areas through strengthening the pheromones on all edges of the best route by an amount of $\Delta\tau_{ij}(h + 1)$. The abilities to diversify and intensify its searching areas provide ACO the opportunity to find optimal solutions for combinatorial problems within reasonable times. Q is an adjustable constant representing the quantity of pheromone per unity of distance and the choice of its value is problem-dependent.

4.2 *ACO for job shop scheduling problems*

Each job in a classical JSSP comprises several operations to be processed on different machines. Generally, their technique orders and the processing times are represented in a technical matrix T and a processing time matrix P, respectively. Each row of T indicates the order of machines that all the operations of one job will visit while each row of P indicates the processing times that all those operations will take on their processing machines. Simple examples of them are given as follows.

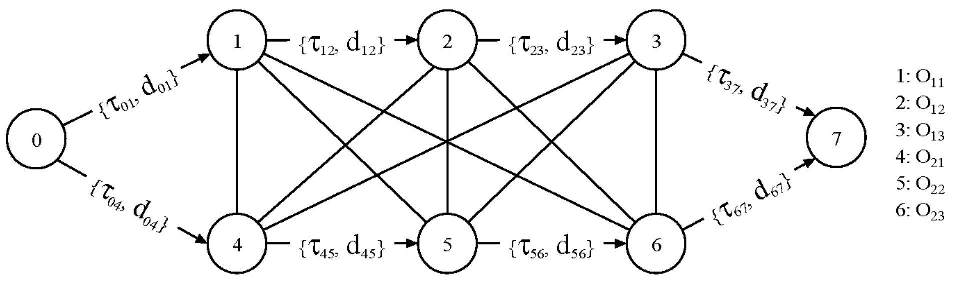
Figure 3 presents a technical matrix and a processing matrix of a JSSP with two jobs and three machines. The first job has three operations O_{11}, O_{12}, and O_{13} that will be processed on machines $M1$, $M2$ and $M3$, in that order, and its three operations need processing times of $t(O_{11})$, $t(O_{12})$, and $t(O_{13})$, respectively, to be processed.

Figure 3 The technical matrix T and the processing matrix P for a 2 × 3 JSSP

$$T=\begin{bmatrix} M1 & M2 & M3 \\ M3 & M1 & M2 \end{bmatrix} \qquad P=\begin{bmatrix} t(O_{11}) & t(O_{12}) & t(O_{13}) \\ t(O_{21}) & t(O_{22}) & t(O_{23}) \end{bmatrix}$$

The JSSP above can be represented as a graph (Figure 4). Nodes 1–6 represent operations O_{11}, O_{12}, to O_{13}, and O_{21}, O_{22}, to O_{23}. They are connected by horizontal and directional edges reflecting the precedence constraints given in matrix T and bi-directional edges, which indicate that there are no ordering constraints among connected operations. Dummy nodes 0 and 7, representing the source and the sink of the graph, are the starting and the ending points of routing. They are connected by directional edges to the first and the last operations of all jobs, respectively.

Figure 4 The graph representing a 2 × 3 JSSP



Each edge is associated with a pair of values $\{t_{ij}, d_{ij}\}$, representing the values of the pheromone on it and the heuristic distance between the two nodes it connects. The value of d_{ij} can be easily looked up from matrix P while the value for t_{ij} should be found in the pheromone matrix, which is updated by the ants who found best solutions (Figure 4). An example of the pheromone matrix for the previous JSSP is shown in Figure 5, which records the pheromone values of all the edges.

The first row of Figure 5 gives the pheromone values of the edges starting from node 0 to the rest six nodes (The pheromones of edges end at nodes 7 are not necessary to be included). Only $\tau_{01} = 0.1$and $\tau_{04} = 0.1$ exist since node 0 can only reach node 1 and node 4. Others are initiated to be 0. Similarly, the second row gives the pheromones of the edges starting from node 1. τ_{10}, τ_{11} and τ_{13} do not exist and are thus initiated as 0. The updating of the pheromone matrix takes the majority of the computational effort due to the dominant size of the pheromone matrix $(n \times m + 1)^2$, where n and m are the sizes of jobs and machines, respectively. As each ant walks through all the nodes in the matrix, the computational complexity is $O(s \times u \times (n \times m)^2)$, where s is the size of iterations and u is the number of ants per iteration.

Ant i cannot guarantee to find a feasible route for a JSSP before it is equipped with three lists: scheduled operation list (S_i), accessible operation list (A_i), and non-accessible operation list (NA_i). List S_i includes the nodes that have been visited by ant i; A_i stores the currently accessible nodes; NA_i stores the rest of the unvisited nodes. The size of S_i increases as ant i proceeds in the graph. Finally, the ordered nodes in list S_i form a complete route, which is a schedule for the JSSP.

Figure 5 An example of the pheromone matrix for a 2 × 3 JSSP

	N_0	N_1	N_2	N_3	N_4	N_5	N_6
N_0	0	0.1	0	0	0.1	0	0
N_1	0	0	0.16	0	0.18	0.19	0.20
N_2	0	0	0	0.18	0.19	0.20	0.21
N_3	0	0	0	0	0.20	0.21	0.22
N_4	0	0.16	0.15	0.14	0	0.22	0
N_5	0	0.17	0.16	0.15	0	0	0.24
N_6	0	0.18	0.17	0.16	0	0	0

4.2.1 ACO for job shop scheduling problem with parallel machines

It is assumed in the current study that there can be an arbitrary number of machines in one workcentre. Thus, a list M_{ij} recording the earliest available times of all machines in workcentre j has to be maintained by ant i. For example, a $M_{23} = \{1.0,\ 1.3,\ 2.1\}$ represents the earliest available times of all three machines in workcentre 3 kept by ant 2. Machine 1 is available from time 1.0; machine 2 is from time 1.3; and machine 3 is from time 2.1. M_{23} turns out to be $M_{23} = \{1.8,\ 1.3,\ 2.1\}$ if an operation with a processing time of 0.8 is allocated to machine 1.

The rule to choose a machine from among several available machines is based on the times that machines become available. In this study, the machine with the earliest available time has the highest priority to be chosen, assuming all the machines in one workcentre are identical. A machine should be chosen randomly if more than one machine has the same earliest available time. This approach avoids the prolonged idleness of some machines.

4.2.2 ACO in a dynamic job shop scheduling environment

- Updating an intermediate JSSP

At each rescheduling moment, an intermediate JSSP has to be updated before the ACO algorithm can be executed. The updating of a pheromone matrix involves updating of nodes and pheromone values. Updating nodes has two aspects: deleting the nodes representing completed or processing operations and adding the nodes representing all the operations of the new job. For example, a new job with three operations O_{31}, O_{32} and O_{33} arrives at the job shop at the moment that node 1 is completed and node 4 is on processing. The updating of nodes includes deleting all the cells related to node 1, as well as adding in the three new nodes (Figure 6).

The cells related to nodes 1 include those from the third column and the third row while the cells related to node 4 include those from the sixth column and the sixth row. All of them are shaded in Figure 6(a) and need to be deleted. Then three new nodes representing three operations of the new job are added to both the ends of the row and the column surrounded by black borders in Figure 6(b); all of the new cells are initiated with appropriate values. Finally, the nodes are re-numbered according to the updated order and a new pheromone matrix is generated in Figure 6(c).

Updating pheromone values of a new pheromone matrix can be with or without the adaptation mechanism. In the former case, the pheromone values on all edge are re-initiated while in the latter case only the new edges are initiated and the others remain unchanged. For example, the adaptation mechanism is presented in Figure 6, where only new edges within the frame of Figure 6(b) are initiated and the others remain unchanged. In this way, some Optimisation information in the previous problem can be kept and a new schedule is sought based on it.

Figure 6 Updating the pheromone matrix upon the reception of a new job: (a) deleting the cells related to nodes 1 and 4; (b) adding in three nodes 7, 8, 9 and (c) the updated pheromone matrix

	N_0	N_1	N_2	N_3	N_4	N_5	N_6
N_0	0	0.1	0.1	0.1	0.1	0.1	0.1
N_1	0	0	0.16	0	0.18	0.19	0.20
N_2	0	0	0	0.18	0.19	0.20	0.21
N_3	0	0	0	0	0.20	0.21	0.22
N_4	0	0.16	0.15	0.14	0	0.22	0
N_5	0	0.17	0.16	0.15	0	0	0.24
N_6	0	0.18	0.17	0.16	0	0	0

(a)

	N_0	N_2	N_3	N_5	N_6	N_7	N_8	N_9
N_0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1
N_2	0	0	0.18	0.20	0.21	0.1	0.1	0.1
N_3	0	0	0	0.21	0.22	0.1	0.1	0.1
N_5	0	0.16	0.15	0	0.24	0.1	0.1	0.1
N_6	0	0.17	0.16	0	0	0.1	0.1	0.1
N_7	0	0.1	0.1	0.1	0.1	0	0.1	0
N_8	0	0.1	0.1	0.1	0.1	0	0	0.1
N_9	0	0.1	0.1	0.1	0.1	0	0	0

(b)

	N_0	N_1	N_2	N_3	N_4	N_5	N_6
N_0	0	0.1	0.1	0.1	0.1	0.1	0.1
N_1	0	0	0.16	0	0.18	0.19	0.20
N_2	0	0	0	0.18	0.19	0.20	0.21
N_3	0	0	0	0	0.20	0.21	0.22
N_4	0	0.16	0.15	0.14	0	0.22	0
N_5	0	0.17	0.16	0.15	0	0	0.24
N_6	0	0.18	0.17	0.16	0	0	0

(c)

- Parameters in the dynamic environment

Increasing the values of *s* and *u* increases the computational time according to $O(s \times u \times (n \times m)^2)$. Thus both values of *s* and *u* have to be constrained as the timeslot for solving each intermediate JSSP is always limited in a dynamic environment.

The value of $s_{\min}$ decides the minimal sets of ants that can be initiated. Its role is to guarantee a minimal computational timeslot for each intermediate JSSP. The value of $s_{\max}$ decides the maximal sets of ants that can be initiated. Its role is to avoid over-enhancement of pheromone values on some edges. The variable *s* within $[s_{\min}, s_{\max}]$ can improve the quality of an intermediate schedule as much as possible in the current test bed where the rescheduling procedure and the event action for a new arrival job (*e*) run independently on different computational threads. For example, if *e* arrives before $s_{\min}$ $s_{\min}$ is satisfied in the previous intermediate JSSP, its execution will be delayed until $s_{\min}$ is completed; otherwise, the event action of *e* can be immediately executed. Meanwhile, more iterations are allowed to improve the solution if the rescheduling procedure is not stopped by *e* and *s* is not greater than $s_{\max}$. The number of ants (*u*) per iteration is also adjustable and its effects will be investigated in experiments.

5 Experimental design

In this study, a scheduler is combined with the existing job shop, which is simulated as a discrete event system (Zhou et al., 2008) and is responsible for generating new schedules to direct the operation in the simulated job shop. It is assumed that the reception of a new job will trigger a rescheduling procedure to find a full schedule within the computational timeslot, and makespan is the only performance measure. The best-so-far schedule is then dispatched to be executed in all workcentres. The rescheduling procedure repeats until the preset stop criteria are met.

5.1 Experimental environments

The studied dynamic job shop is shown in Figure 7 and it has five workcentres and one receiving or shipping station. The numbers of machines in workcentres 1 to 5 are 4, 2, 5, 3, and 2, respectively. The machines in the same workcentre are assumed identical. Transportation times (hours) between any two workcentres are given in Table 1, which are added to with the processing time to form the heuristic value indicated in Section 4.1.

Figure 7 Layout of the job shop (see online version for colours)

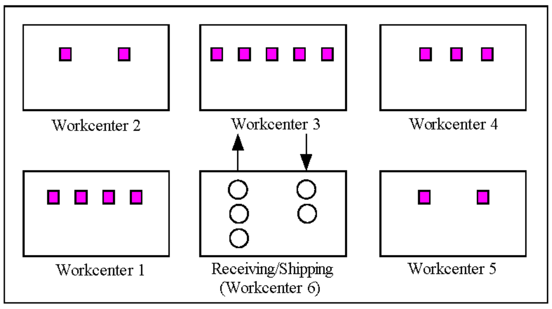
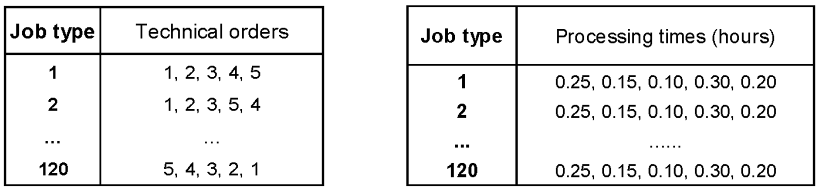


Table 1 Transportation times between workcentres (hours)

Workcentre	*1*	*2*	*3*	*4*	*5*	*6*
1	0	0.01	0.01	0.02	0.02	0.01
2	0.01	0	0.01	0.02	0.02	0.01
3	0.01	0.01	0	0.01	0.01	0.01
4	0.02	0.02	0.01	0	0.01	0.01
5	0.02	0.02	0.01	0.01	0	0.01
6	0.01	0.01	0.01	0.01	0.01	0

New jobs first arrive at the receiving/shipping station (workcentre 6) and travel through workcentres according to their technical orders and finally leave the system from the receiving/shipping station. There are totally 5! = 120 types of jobs and each type of job occurs with a probability of 1/5! and the total processing time for each job is one hour. The technical matrix and the processing time matrix are given in Figure 8.

Figure 8 The technical matrix and the processing time matrix



Job type	Technical orders
1	1, 2, 3, 4, 5
2	1, 2, 3, 5, 4
...	...
120	5, 4, 3, 2, 1

Job type	Processing times (hours)
1	0.25, 0.15, 0.10, 0.30, 0.20
2	0.25, 0.15, 0.10, 0.30, 0.20
...	......
120	0.25, 0.15, 0.10, 0.30, 0.20

The parameters of the ACO algorithm are $\alpha = 10.0$, $\beta = 10.0$, $\rho = 0.01$, $Q = 1.0$, and $\tau = 0.5$ tuned by van der Zwaan and Marques (1999) to solve several JSSP benchmarks. They are adopted here as each intermediate JSSP is similar to those benchmarks. It is also assumed that the computation timeslot decided by $s_{\min}$ is within the time constraint in realistic applications. The following are the default values: $s_{\min} = 25$, $s_{\max} = 100$, and $u = 10$.

5.2 Experimental variables

Jobs arrive at the shop floor with inter-arrival times that are independent exponential random variables. The mean job inter-arrival time and the lot size are two problem variables that decide the utilisations of workcentres. Two levels of job-arrival frequencies with the same mean size of total jobs are tested. In problem 1, jobs arrive one by one with the mean job inter-arrival time of nine jobs per hour. In problem 2, jobs are released in lots and arrive one lot per hour with nine jobs per lot. Jobs in one lot can be different types and will be processed job by job. In both problems, the type of a job is randomly decided so that each one of the 120 types has an equal chance of being chosen. Thus, the mean total processing time demanded from each workcentre is the same.

The size of jobs in a lot decides the severity that an underlying scheduling problem is disturbed. For example, there are 16 unprocessed operations when a lot of new jobs are released to the shop floor. The size of operations for the new intermediate JSSP is increased to 22 if there is only one job (with six operations) in the lot. Thus, the operations in the previous JSSP take about 73% (16/22) of the total operations in the new

JSSP. However, they take only 57% (16/28) if one more job (also with six operations) is included in the lot. Obviously, the underlying problem is changed more severely by a large lot than a small one.

The simulation for each problem runs five replications for 200 simulation hours (totally about 1800 jobs). Only steady-state performance is measured and the average values of five replications are listed for all performance measures.

5.3 Performance measures

Many performance measures are also recorded as the results of dynamic scheduling using ACO. Those include machine utilisation and other inventory measures such as average Work-in-Process (WIP), average/maximal number in each queue, average daily throughput, average time of jobs spending in the job shop, and average total time of jobs in queues. Furthermore, the average/maximal size of operations per iteration for ACO is also measured for performance evaluation.

Machine utilisation refers to the rate of the machine busy time to the whole experimental steady state period. Average WIP refers to the average size of jobs during the experimental period. Average daily throughput refers to the average number of completed jobs per day (eight hours).

6 Computational results and analysis

The results are given in Tables 2–7 where each table records two sets of experimental results separated by the sign of '/'. The corresponding experimental conditions are given in the first row of table and also separated by the sign of '/'. For example, Table 1 records the experimental results of ACO with and without the pheromone adaptation mechanism. This condition is given in the first row of Table 1 as "ACO (with/without pheromone adaptation); the results of ACO with pheromone adaptation are recorded before '/' while the results of ACO without pheromone adaptation are recorded after '/'.

Some general observations can be found as follows. Firstly, workcentres 2 and 5 are bottlenecks. Secondly, the machine utilisation is inversely proportional to the size of machines in its workcentre. The above two results are in accordance with the facts that workcentres 2 and 5 have the smallest numbers of machines (2) while they have the same workload as the rest of the workcentres. Thirdly, the improvement in the average daily throughput and the machine utilisation also implies the decreasing in the average number of jobs in a queue, the maximal number of jobs in a queue, the job average time in the system, the job average total waiting times in queues, the maximal size of WIP, and the maximal/average size of operations of intermediate problems, which reflects the overall performance of ACO as analysed in Section 3.2.

6.1 ACO performance analysis

The performance evaluations of ACO in two dynamic JSSPs are listed in Tables 2 and 3. The 200 hourly throughputs of the five replications for both problems with the adaptation mechanism are plotted in Figures 9 and 10 where the moving average $\overline{Y}_i(20)$ with a window of 20 (Law and Kelton, 2000) is used. A warming up period of $l = 20$ hours (about 180 jobs) can be obtained.

Figure 9 Moving average ($w = 20$) of hourly throughputs of problem 1 with adaptation

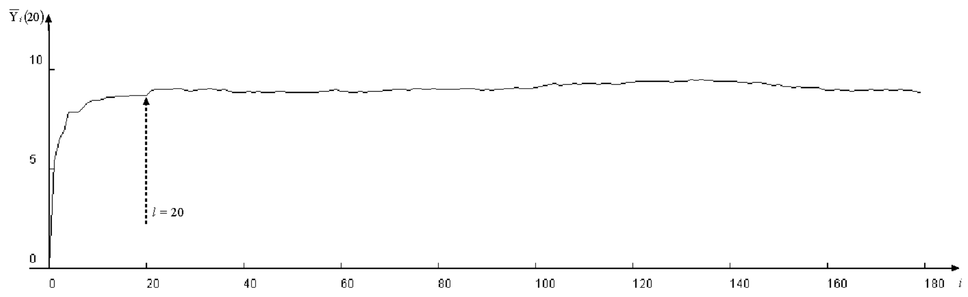


Figure 10 Moving average ($w = 20$) of hourly throughputs of problem 2 with adaptation

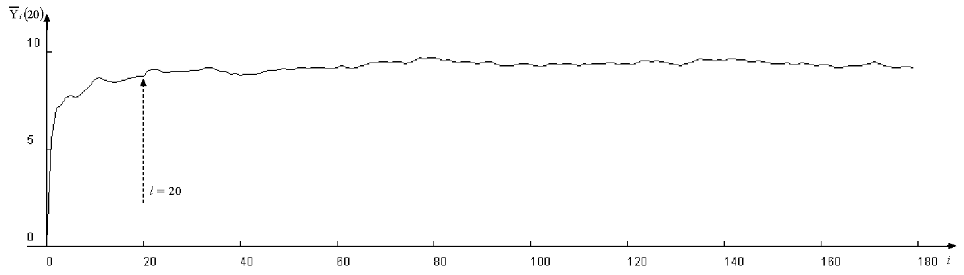


Table 2 Effectiveness and adaptation of ACO – problem 1

Mean job inter-arrival time: 1/9 hour, 1 job/lot			*Number of machines: 4, 2, 5, 3, 2*		
ACO (with/without pheromone adaptation) (10 ants)			*Simulation time: 200 hours*		
120 types of jobs (randomly)			*Warming up time: 20 hours*		
Performance measure	*1*	*2*	*3*	*4*	*5*
Machine utilisation (workcentre)	0.404/ 0.421	0.902/ 0.830	0.355/ 0.334	0.564/ 0.563	0.916/ 0.839
Average number in queue (workcentre)	0.703/ 5.764	5.558/ 36.115	0.404/ 3.316	1.297/ 14.793	5.838/ 35.726
Maximum number in queue(workcentre)	7.0(10)/ 20.4(33)	21.2(28)/ 104.4(140)	5.2(6)/ 14.0(24)	9.4(12)/ 41.4(70)	21.0(31)/ 77.6(123)
Average daily throughput (shop floor)			72.258/64.871		
Average time in system (shop floor)			2.524/11.022		
Average total time in queues (shop floor)			1.448/9.946		
Maximal size of WIP (shop floor)			48.8(69)/216(380)		
Maximal size of operations			138.8(198)/734.4(1335)		
Average size of operations			59.8/285.8		

Then 90% confidence intervals for the steady-state mean daily throughputs of the two problems are constructed as $72.258 \pm t_{9,0.95}\sqrt{0.46/5}$ (or [72.09, 72.43]) for Problem 1 and $73.973 \pm t_{9,0.95}\sqrt{2.13/5}$ (or [73.19, 74.75]) for Problem 2. The results show that ACO can perform well in both dynamic JSSPs to meet the expected daily throughput of 72 jobs as the mean inter-arrival time of jobs is 1/9 hour and there are 8 hours per day.

Table 3 Effectiveness and adaptation of ACO – problem 2

Mean job inter-arrival time: 1/9 hour, 1 job/lot			*Number of machines: 4, 2, 5, 3, 2*		
ACO ($s_{\min}$ = 25/40 iterations) (10 ants)			*Simulation time: 200 hours*		
120 types of jobs (randomly)			*Warming up time: 20 hours*		
Performance measure	*1*	*2*	*3*	*4*	*5*
Machine utilisation (workcentre)	0.438/ 0.462	0.922/ 0.924	0.364/ 0.361	0.617/ 0.617	0.935/ 0.935
Average number in queue (workcentre)	3.482/ 3.488	27.839/ 29.382	2.420/ 2.445	5.564/ 5.523	30.195/ 29.774
Maximum number in queue (workcentre)	15.8(17)/ 17.6(21)	70.0(84)/ 80.4(86)	15.2(18)/ 15.2(17)	21.0(28)/ 22.4(26)	69.8(87)/ 71.4(91)
Average daily throughput (shop floor)			73.973/73.929		
Average time in system (shop floor)			8.426/8.545		
Average total time in queues (shop floor)			7.350/7.469		
Maximal size of WIP (shop floor)			151.4(178)/152.6(178)		
Maximal size of operations			565.8(669)/569.4(683)		
Average size of operations			275.4(364)/278.8		

6.2 The effects of the ACO adaptation mechanism

The performance comparisons of ACO with/without adaptation in both problems are also listed in tables in Tables 2 and 3. The daily throughputs drop from 72.258 to 64.871 in Problem 1 and from 73.973 to 73.929 in Problem 2 when the adaptation mechanism is first applied and then removed. The change is significant in Problem 1 and minor in Problem 2.

The results indicate that the adaptation mechanism has a great effect in the situation where disturbances are not severe as in Problem 1 and has little effect in the situation where disturbances are severe as in Problem 2. The observation can be explained as follows. In Problem 1, jobs arrive one by one and neighbouring intermediate JSSPs are not markedly different. A good solution can be found through the adaptation mechanism within a given computational timeslot. However, in Problem 2, there would be no much difference between the pheromone matrices with and without the adaptation mechanism since the underlying problem can be dramatically changed by a large lot.

6.3 The effects of the number of minimal iterations

The results given in Tables 4 and 5 show that increasing the s_{min} deteriorates the performance of ACO in both problems, especially in Problem 1 (72.258 for $s_{min} = 25$ and 64.693 for $s_{min} = 40$), when both problems adopt the adaptation mechanism. This seems against the initial expectation that increasing the number of iterations can increase the optimality of an intermediate schedule and thus improve the performance of the overall performance of ACO.

Table 4 Increase the minimal number of iterations – Problem 1

Mean job inter-arrival time: 1/9 hour, 1 job/lot			*Number of machines: 4, 2, 5, 3, 2*		
ACO (s_{min} = 25/40 iterations) (10 ants)			*Simulation time: 200 hours*		
120 types of jobs (randomly)			*Warming up time: 20 hours*		
Performance measure	*1*	*2*	*3*	*4*	*5*
Machine utilisation (workcentre)	0.404/ 0.419	0.902/ 0.826	0.355/ 0.332	0.564/ 0.560	0.916/ 0.835
Average number in queue (workcenre)	0.703/ 6.040	5.558/ 37.344	0.404/ 3.113	1.297/ 15.989	5.838/ 37.085
Maximum number in queue (workcentre)	7.0(10)/ 24.4(35)	21.2(28)/ 78.2(140)	5.2(6)/ 13.4(27)	9.4(12)/ 45.4(75)	21.0(31)/ 75.4(117)
Average daily throughput (shop floor)	72.258/64.693				
Average time in system (shop floor)	2.524/11.458				
Average total time in queues (shop floor)	1.448/10.382				
Maximal size of WIP (shop floor)	48.8(69)/222(383)				
Maximal size of ACO operations	138.8(198)/778.2(1362)				
Average size of ACO operations	59.8/300.2				

This phenomenon could be explained as follows. The pheromone values of certain edges are increased too much as the result of increasing s_{min} and the over-strengthened pheromone matrix becomes too rigid to be developed further to find a new good solution. Thus, the scheduler can only produce a worse intermediate schedule each time, especially in a highly dynamic environment where the computational time is limited.

6.4 The effects of changing the number of ants per iteration

The results testing the effects of changing the number of ants per iteration are given in Tables 2, 3, 6 and 7, which show that the overall performance of ACO gets worse as the size of ants per iteration increases. For example, with other problem parameters unchanged, the average daily throughput decreases from 72.258, 68.364, to 65.502 in Problem 1 and from 73.973, 72.978, to 71.502 in Problem 2 when the size of ants per iteration increases from 10, 20, to 40.

Table 5 Increase the minimal number of iterations – Problem 2

Mean job inter-arrival time: 1/9 hour, 1 job/lot			*Number of machines: 4, 2, 5, 3, 2*		
ACO ($s_{\min}$ = 25/40 iterations) (10 ants)			*Simulation time: 200 hours*		
120 types of jobs (randomly)			*Warming up time: 20 hours*		
Performance measure	*1*	*2*	*3*	*4*	*5*
Machine utilisation (workcentre)	0.438/ 0.459	0.922/ 0.918	0.364/ 0.334	0.617/ 0.530	0.935/ 0.930
Average number in queue (workcentre)	3.482/ 4.640	27.839/ 31.667	2.420/ 3.436	5.564/ 8.295	30.195/ 32.961
Maximum number in queue (workcentre)	15.8(17)/ 19.0(30)	70.0(84)/ 73.6(90)	15.2(18)/ 16.4(24)	21.0(28.0)/ 25.2(37)	69.8(87)/ 73.6(86)
Average daily throughput (shop floor)	73.973/73.138				
Average time in system (shop floor)	8.426/9.657				
Average total time in queues (shop floor)	7.350/8.581				
Maximal size of WIP (shop floor)	151.4(178)/164(194)				
Maximal size of ACO operations	565.8(669)/598.6(687)				
Average size of ACO operations	275.4(364)/302.2(413)				

The phenomenon can be explained as follows. A schedule with a smaller makespan is more likely found by ants with increased number; subsequently, a greater pheromone value is added on related edges. The optimality found in this schedule is an advantage if it can be fully realised without disturbance. Otherwise, it could be a disadvantage for causing over-strengthened edges. Similar to the situation in Section 6.3, the pheromone matrix may become rigid in capturing new information introduced by new jobs and fail to give good schedules for the following intermediate problems. Thus, increasing the number of ant per iteration may lead to an inferior overall performance in a dynamic environment. For both cases, $u = 10$ seems working well.

Table 6 Increase the number of ants per iteration – Problem 1

Mean job inter-arrival time: 1/1 hour, 9 jobs/lot			*Number of machines: 4, 2, 5, 3, 2*		
ACO (u = 20/40)			*Simulation time: 200 hours*		
120 types of jobs (randomly)			*Warming up time: 20 hours*		
Performance measure	*1*	*2*	*3*	*4*	*5*
Machine utilisation (workcentre)	0.421/ 0.423	0.873/ 0.838	0.348/ 0.335	0.550/ 0.566	0.884/ 0.848
Average number in queue (workcentre)	4.666/ 5.209	34.264/ 32.949	3.112/ 2.937	12.153/14. 593	34.237/ 32.734

Table 6 Increase the number of ants per iteration – Problem (continued)

Mean job inter-arrival time: 1/1 hour, 9 jobs/lot			*Number of machines: 4, 2, 5, 3, 2*		
ACO (u = 20/40)			*Simulation time: 200 hours*		
120 types of jobs (randomly)			*Warming up time: 20 hours*		
Performance measure	*1*	*2*	*3*	*4*	*5*
Maximum number in queue(workcentre)	18.2(34)/ 22.2(34)	75.2(143)/ 70(138)	13.2(22)/ 11.8(21)	36.2(79)/ 41.2(83)	72.8(122)/ 68.6(116)
Average daily throughput (shop floor)			68.364/65.502		
Average time in system (shop floor)			9.999/10.232		
Average total time in queues (shop floor)			8.923/9.156		
Maximal size of WIP (shop floor)			189.8(389)/200.2(382)		
Maximal size of ACO operations			626.6(1320)/674.2(1332)		
Average size of ACO operations			275.4(529)/262.6(531)		

Table 7 Increase the number of ants per iteration – Problem 2

Mean job inter-arrival time: 1/1 hour, 9 jobs/lot			*Number of machines: 4, 2, 5, 3, 2*		
ACO (u = 20/40)			*Simulation time: 200 hours*		
120 types of jobs (randomly)			*Warming up time: 20 hours*		
Performance measure	*1*	*2*	*3*	*4*	*5*
Machine utilisation (workcentre)	0.434/ 0.453	0.916/ 0.903	0.319/ 0.355	0.614/ 0.605	0.929/ 0.914
Average number in queue (workcentre)	4.940/ 9.140	31.686/ 37.849	3.372/ 6.538	8.251/ 16.221	33.826/ 37.819
Maximum number in queue (workcentre)	19.8(32)/ 30.0(41)	73.4(84)/ 77.2(97)	17.6(23)/ 27.4(36)	25.2(36)/ 38.8(45)	72.4(87)/ 75.6(94)
Average daily throughput (shop floor)			72.978/71.502		
Average time in system (shop floor)			9.759/12.349		
Average total time in queues (shop floor)			8.683/11.273		
Maximal size of WIP (shop floor)			166.6(213)/177.6(282)		
Maximal size of ACO operations			599.8(679)/718.2(963)		
Average size of ACO operations			305.4(443)/358.6(507)		

7 Conclusions and further studies

A basic version of ACO has been applied to two dynamic JSSPs with the same workloads but in different dynamic levels and disturbing severity. The experimental results show:

- the ACO performs effectively in both cases
- the adaptation mechanism of the ACO does have a great effect in the situation where disturbances are slight but it has little effect in the situation where disturbances are severe
- improving the optimality of immediate schedules but reducing the flexibility of the pheromone matrix will lead to an inferior long-term performance.

Further studies include applying a more advanced version of ACO to dynamic JSSP, exploring better ways to updating the pheromone matrix dynamically, and comparing the performance of ACO with other approaches in systematically designed scenarios.

References

Angus, D. and Hendtlass, T. (2002) *Ant Colony Optimization Applied to a Dynamically Changing Problem*, IEA/AIE, LNAI 2358, pp.618–627.

Branke, J. (2004) *Evolutionary Optimization in Dynamic Environments*, Kluwer Academic Publishers, Boston.

Colorni, A., Dorigo, M., Maniezzo, V. and Trubian, M. (1994) 'Ant system for Job-shop scheduling', *JORBEL-Belgian Journal of Operations Research Statistics and Computer Science*, Vol. 34, No. 1, pp.39–53.

Dorigo, M. and Di Caro, G. (1999) 'The Ant Colony Optimization Meta-Heuristic', in Corne, D., Dorigo, M. and Glover, F. (Eds.): *New Ideas in Optimization*, McGraw-Hill, pp.11–32.

Dorigo, M. and Gambardella, L.M. (1997a) 'Ant colonies for the traveling salesman problem', *BioSystems*, Vol. 43, No. 2, pp.73–81.

Dorigo, M. and Gambardella, L.M. (1997b) 'Ant colony System: a cooperative learning approach to the traveling salesman problem', *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 4, pp.317–365.

Dorigo, M. and Stützle, T. (2004) *Ant Colony Optimization*, A Bradford Book, The MIT Press, Cambridge, Massachusetts, London, England.

Dorigo, M., Di Caro, G. and Gambardella, L.M. (1999) 'Ant algorithms for discrete optimization', *Artificial Life*, Vol. 5, No. 3, pp.137–172.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996) 'The Ant system: optimization by a colony of cooperating agents', *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, Vol. 26, No. 1, pp.1–13.

Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: An Introduction to the Theory of NP-Completeness*, Freeman W.H., San Francisco.

Guntsch, M. and Middendorf, M. (2001) 'Pheromone modification strategies for ant algorithms applied to dynamic TSP', in Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R. and Tijink, H. (Eds.): *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*, Number 2037 in Lecture Notes in Computer Science, Springer-Verlag, pp.213–222.

Guntsch, M., Middendorf, M. and Schmeck, H. (2001) 'An ant colony optimization approach to dynamic TSP', in Spector, L., Goodman, E.D., Wu, A., Langdon, W.B. and Voigt, H.M. (Eds.): *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, Morgan Kaufmann Publishers, pp.860–867.

Guntsch, M. and Middendorf, M. (2002) ‘Applying population based ACO to dynamic optimization problems, in Ant algorithms’, *Proceedings of Third International Workshop ANTS 2002*, Vol. 2463, of LNCS, pp.111–122.

Law, A.M. and Kelton, W.D. (2000) *Simulation Modeling and Analysis*, McGraw-Hill, Boston, MA.

Lin, S-C., Goodman, E.D. and Punch, W.F. (1997) ‘A genetic algorithm approach to dynamic job shop scheduling problems’, in Back, T. (Ed.): *Proceedings of the 7th International Conference on Genetic Algorithm*, Morgan Kaufmann, San Francisco, CA, pp.481–488.

Raman, N. and Talbot, F.B. (1993) ‘The job shop tardiness problem: a decomposition approach’, *European Journal of Operational Research*, Vol. 69, pp.187–199.

Stützle, T. and Dorigo, M. (1999) ‘ACO algorithms for the traveling salesman problem’, |in Miettinen, K., Mäkälä, M.M., Neittaanmäki, P. and Périaux, J. (Eds.): *In Evolutionary Algorithms in Engineering and Computer Science*, John Wiley & Sons, Chichester, UK, pp.163–183.

Vogel, A., Fischer, M., Jaehn, H. And Teich, T. (2002) ‘Real-world shop floor scheduling by ant colony optimization’, in Dorigo, M., Middendorf, M. and Stützle, T. (Eds.): *ANTS 2002*, LNCS 2463, pp.268–273.

Zhou, R., Lee, H.P. and Nee, A.Y.C. (2008) ‘Simulating the generic job shop as a multi-agent system’, *Special Issue of International Journal of Intelligent Systems Technologies and Applications (IJISTA)*, Vol. 4, Nos. 1–2, pp.5–33.

van der Zwaan, S. and Marques, C. (1999) ‘Ant colony optimization for job shop scheduling’, *Proceedings of the 3rd’Workshop on Genetic Algorithms and Artificial Life (GAAL’99))*, 22 April, LaSEEB – IST, IST, Lisbon.

Website

OR-Library, www.ms.ic.ac.uk/info.html