Available online at www.sciencedirect.com

ScienceDirect

Expert Systems with Applications 34 (2008) 2071–2081

Expert Systems with Applications

www.elsevier.com/locate/eswa

Multiprocessor system scheduling with precedence and resource constraints using an enhanced ant colony system

Shih-Tang Lo [a], Ruey-Maw Chen [b], Yueh-Min Huang [a,*], Chung-Lun Wu [c]

[a] *Department of Engineering Science, National Cheng-Kung University, Tainan 701, Taiwan, ROC*
[b] *Department of Computer Science and Information Engineering, National Chin-yi Institute of Technology, Taichung 411, Taiwan, ROC*
[c] *Department of Electronic Engineering, National Chin-yi Institute of Technology, Taichung 411, Taiwan, ROC*

Abstract

This study presents and evaluates a modified ant colony optimization (ACO) approach for the precedence and resource-constrained multiprocessor scheduling problems. A modified ant colony system is proposed to solve the scheduling problems. A two-dimensional matrix is proposed in this study for assigning jobs on processors, and it has a time-dependency relation structure. The dynamic rule is designed to modify the latest starting time of jobs and hence the heuristic function. In exploration of the search solution space, this investigation proposes a delay solution generation rule to escape the local optimal solution. Simulation results demonstrate that the proposed modified ant colony system algorithm provides an effective and efficient approach for solving multiprocessor system scheduling problems with resource constraints.

Keywords: Ant colony optimization; Scheduling; Multiprocessor

1. Introduction

1.1. Scheduling problem

Job scheduling problems are typically considered to involve executing a set of jobs satisfying given constraints and optimizing given criteria. Jobs are assigned timing constraints such as ready time, due date, and a processing time (Cardeira & Mammeri, 1996). There are also some other constraints, like setup time between two jobs, job precedence, and resource requirements. Scheduling has many applications in commercial, industrial and academic fields, including avionics, communications, signal processing, routing, industrial control, operations research, production planning, project management, process scheduling in operating systems, class arrangement and grid computing. Many different schemes have been presented for solving scheduling problems. In practice, multiprocessor scheduling problems only consider the precedence constraint and finding the minimum of maximum complete time. In this study, an ACO approach for the precedence and resource-constrained multiprocessor scheduling problem is presented and evaluated.

1.2. Multiprocessor scheduling problems using genetic algorithm

In a multiprocessor scheduling problem, given programs (tasks) with precedence relation within tasks are scheduled in a given multiprocessor system, such that the program's execution time is minimized. Such problems are the same as the task-scheduling problem. This problem is extremely difficult to solve precisely, and many heuristic methods for finding a suboptimal or optimal schedule exist. Most scheduling problems are confirmed to be NP-complete problems. The traveling salesman problem (TSP) is a typical NP-complete problem, for which obtaining an optimal

* Corresponding author.
E-mail addresses: edwardlo@mail.ksu.edu.tw (S.-T. Lo), raymond@mail.ncit.edu.tw (R.-M. Chen), huang@mail.ncku.edu.tw (Y.-M. Huang), arashilen@yahoo.com.tw (C.-L. Wu).

solution for a tour with a minimum distance is quite time-consuming. Liu and Leyland pioneered real-time scheduling algorithms for mono-job or scheduling of independent and periodic tasks (Liu & Layland, 1973). The genetic algorithm (GA) is the most popular and widely used technique for several kinds of multiprocessor scheduling problem (Correa, Ferreira, & Rebreyend, 1996; Hou, Ansari, & Ren, 1994). Crossover, mutation and selection operators are applied to create for the new generation of schedules and find the solution with GA. Hou et al. (1994) developed an efficient method, the height value of each job in graph, based on a genetic algorithm to solve the multiprocessor scheduling problem. Correa, Ferreira, and Rebreyend (1999) proposed a novel combined approach, in which a genetic algorithm is enhanced with the introduction of some knowledge about the scheduling problem represented by the use of a list heuristic in the crossover and mutation genetic operations. Zomaya, Ward, and Macey (1999) studied an alternative paradigm, based on genetic algorithms, to solve the parallel processor scheduling problem efficiently without the need to apply any restrictive problem-specific assumptions. The above studies focused only on multiprocessor scheduling without resource constraints. Some works focus on finding the minimum processors or the heterogeneous processors scheduling problems. Oh and Wu (2004) presented a multi-objective genetic algorithm, which aims to minimize the number of processors required and the total tardiness of tasks. Topcuoglu, Hariri, and Wu (2002) presented two novel scheduling algorithms for a bounded number of heterogeneous processors, aiming to meet high performance and fast scheduling time simultaneously.

A GA generates a high quality of output schedules in homogeneous or heterogeneous systems, but the scheduling times are generally much higher than with the heuristic-based schemes. Additionally, several control parameters in a genetic algorithm need to be determined appropriately. Hence, GA along with simulated annealing (SA) and local search methods, called guided random search techniques, have been presented (Kwok, Ahmad, & Gu, 1996; Topcuoglu et al., 2002; Wu, Shu, & Gu, 1997).

1.3. Ant system for job scheduling problems

In ACO, a set of ant-like agents or software ants solve the problem under consideration cooperatively. This effort is mediated by exchanging information based on the problem structure collected concurrently by the agents, while building solutions stochastically. Similarly, an ACO scheduling algorithm, consisting of concurrent distributed agents, which discovers a feasible solution, is presented. ACO is a class of constructive meta-heuristic algorithms that share the common approach of building a solution on the basis of information provided by both a standard constructive heuristic function and previously constructed solutions (Maniezzo & Carbonaro, 1999). Dorigo and Gambardella (1997) first applied the ant colony system (ACS) to solve TSP. Simulation results indicate that ACS outperforms other nature-inspired algorithms, such as simulated annealing and evolutionary computation. Applications of the ACO algorithm are also involved in solving job shop scheduling problems (Pierucci, Brandani, & Sogaro, 1996). Besten, Sttzle, and Dorigo (2000) presented an application of the ACO meta-heuristic to the single machine total weighted tardiness problem. Gajpal, Rajendran, and Ziegler (2004) adopted ACO to solve the problem of scheduling in flowshop with sequence-dependent setup times of jobs. Rajendran and Ziegler (2004) developed two ant colony optimization algorithms for solving the permutation flowshop scheduling problem. Merkle, Middendorf, and Schmeck (2002) presented an ACO approach for the resource-constrained project scheduling problem (RCPSP), which is a schedule problem to find the minimum *makespan* with resource and precedence constraints. These studies indicate that ACO can work successfully in many different scheduling applications about combination problems.

1.4. Artificial neural networks

Hopfield and Tank first adopted artificial neural networks, called Hopfield neural networks (HNN), to solve optimization problems. In the HNN (Hopfield & Tank, 1985), the state input information from a community of neurons is received to determine the neuron output state information. Each neuron exchanges information with other neurons in the network. These neurons apply this information to move the network cooperatively, thus achieving convergence. A competitive Hopfield neural network (CHNN) utilizes a competitive learning mechanism to update the neuron states in the Hopfield neural network. In our previous work, a multi-constraint schedule problem for a multiprocessor system was solved by HNN (Huang & Chen, 1999). Chen et al. also presented a modified neural network to solve the multiprocessor scheduling problem with inequality constraints (Chen, Lo, & Huang, 2007).

A series of studies has been conducted using HNN and mean field annealing. These schemes are adopted for multiprocessor scheduling problems, and a modified cooling schedule has been developed to accelerate the convergence rate for the problem investigated (Chen & Huang, 1998). A typical CHNN scheme has also been applied to the same problem (Chen & Huang, 2001). Most of these works concentrate on the specific multiprocessor scheduling situations in which each resource type has one resource available.

1.5. Resource-constrained multiprocessor scheduling problems

This work attempts to find optimal or near-optimal solutions to multiprocessor schedule problems with resource and precedence constraints and restricted scheduling times, known as resource-constrained multiprocessor scheduling problems, and denoted by RCMPSP in this study. The traditional multiprocessor scheduling problems only consider the precedence constraint, and do not address the resource requirement problem while executing. However, RCMPSP has some processors available in scheduling problems, not only with the precedence relation between jobs, but also with the resource requirement constraints. This work not only meets the job precedence and resource requirement, but also minimizes the *makespan*. The proposed algorithm enables fast optimal or near-optimal solutions to be found, and is useful in industrial environments where computational resources and time are restricted.

RCPSP using ACO algorithms has recently been studied (Brucker, Drexel, Möhring, Neumann, & Pesch, 1999; Herroelen, Reyck, & Demeulemeester, 1998; Merkle et al., 2002). These scheduling problems have been demonstrated to be NP-hard. The ACO has been successfully applied to RCPSP, combined with the different heuristics, easily obtaining a near-optimal solution. The RCMPSP is a general scheduling problem. Therefore, the concept can be adopted to solve project scheduling, job-shop, flow-shop, open-shop problems and grid computing problems. Hence, the RCMPSP is of interest in this investigation.

The rest of this study is organized as follows. Section 2 reviews the ant colony system in the TSP problem and the constraints of RCMPSP. Section 3 describes the proposed ACO algorithm, which combines the delay solution generation rule and dynamic rule for the scheduling problem. The simulation examples and experimental results are presented in Section 4. Conclusions and discussions are given in Section 5.

2. Ant colony system and scheduling problem

2.1. Ant colony system

The ACO algorithm has been demonstrated to be an effective means of solving complex combinatorial optimization problems. In ACO, the positive feedback of pheromone deposits on arcs comprising more optimal node-arc tours (paths), allows the next cycle (iteration) to progress toward an optimal solution (Stützle & Hoos, 2000). ACO mimics the behavior of foraging ants. Ants deposit pheromones on the paths that they move along. The pheromone level deposited on a particular path increases with the number of ants passing along it. Ants adopt pheromones to communicate and cooperate with each another in order to identify the shortest paths to a destination. ACO is applied to the TSP first, since it enables an efficient evolution toward quality sub/optimal solutions. Dorigo et al. proposed the ACO algorithm to solve the well-known TSP, which evolved into the ant colony system (ACS) as shown in Fig. 1 (Dorigo & Gambardella, 1997; Dorigo, Maniezzo, & Colorni, 1996).

A graph $G = (V,E)$ comprises a set of nodes (vertex) $V = \{v_1, v_2, \ldots, v_n\}$ and a set of edges $E = \{(i,j)|v_i, v_j \in V\}$.

```
Initialize
Loop
  Each ant is positioned on a starting node
  Loop
    Each ant applies a state transition rule to incrementally build a
      solution and a local pheromone update rule
  Until all ants have built a complete solution
  A global pheromone updating rule is applied
Until End_condition is reached
```

Fig. 1. ACS for TSP.

Normally, each edge (i,j) is associated with one value representing a distance or cost. Each ant establishes a complete tour in a graph (i.e., a feasible solution to the TSP) by repeatedly applying a stochastic greedy rule (the state transition rule) to choose nodes to visit. Ants choose the next node to visit using a combination of heuristic and pheromone information. Ant k at node v_i selects the next node v_j to move based on Eq. (1) when $q \leqq q_0$:

$$[\tau(i,j)]^{\alpha}[\eta(i,j)]^{\beta} = \max_{v_l \in J_k(i)} \left\{ [\tau(i,l)]^{\alpha}[\eta(i,l)]^{\beta} \right\} \tag{1}$$

where q is a random number uniformly distributed in $[0,1]$, and $0 \leqq q_0 \leqq 1$ is an predetermined parameter that determines the relative importance of exploitation versus exploration. $\tau(i,j)$ denotes the pheromone level on edge (i,j). And $\eta(i,j)$ represents a heuristic function defined as the reciprocal of cost. $J_k(i)$ denotes the set of nodes to be visited by ant k at node v_i, and parameter α, β determines the relative importance between the pheromone level and the edge cost.

If $q > q_0$, v_j is randomly selected from $J_k(i)$ according to the probability distribution given by the following equation:

$$p_k(i,j) = \begin{cases} \frac{[\tau(i,j)]^{\alpha}[\eta(i,j)]^{\beta}}{\sum_{v_l \in J_k(i)} [\tau(i,l)]^{\alpha}[\eta(i,l)]^{\beta}}, & \text{if } j \in J_k(i), \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

After an ant has completed its tour, the pheromones on the edges of that tour are updated using the local updating rule. The ACS uses the following local updating rule to prevent succeeding ants from searching in the neighborhood of the current best tour. The rule is defined as

$$\tau(i,j) = (1-\rho)\tau(i,j) + \rho\Delta\tau(i,j), \tag{3}$$

where ρ $(0 < \rho < 1)$ is a parameter representing the local pheromone evaporation rate, and $\Delta\tau(i,j) = \tau_0$ is the initial pheromone level.

Once all the ants have completed their tours, the pheromones on all edges of the graph are updated using the global updating rule. The ACS uses the global updating rule to accelerate searching for the best solution. The global updating rule enhances the edges discovered in the globally best tour and is defined as

$$\tau(i,j) = (1-\delta)\tau(i,j) + \delta\tau_{gb}(i,j), \tag{4}$$

where δ $(0 < \delta < 1)$ is a parameter representing the global pheromone evaporation rate, and

$$\tau_{gb}(i,j) = \begin{cases} L_{gb}^{-1}, & \text{if edge}(i,j) \in \text{the global best tour}, \\ 0, & \text{otherwise}, \end{cases} \tag{5}$$

where L_{gb} represents the globally best tour in the current iteration.

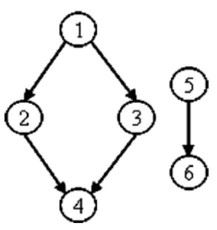
The ACO algorithm has recently been applied to scheduling problems, such as job-shop, flow-shop, and single machine tardiness problems (Bauer et al. 1999; Dorigo & Gambardella, 1997; Iredi, Merkle, & Middendorf, 2001; Merkle & Middendorf, 2001). Traditionally, the pheromone matrix $\tau = [\tau_{ij}]$, where the pheromone is added to an element τ_{ij} of the pheromone matrix, finds a good solution where job j is the ith job on the machine. The following ants of the next generation directly use the value of τ_{ij} and heuristic function to estimate the desirability of placing job j as the ith job on the machine when obtaining a new solution (Merkle et al., 2002). Bauer, Bullnheimer, Hartl, and Strauss (1999) proposed ACO algorithms using a conventional pheromone matrix $[\tau_{ij}]$ to solve the single machine total tardiness problem and the flow-shop problem. This study adopts a modified pheromone matrix $[\tau_{tj}]$, in which the element τ_{tj}, denoting the pheromone value of job j is processed at time t on a specific machine. Restated, the two-dimensional grid for assigning jobs on processors is a time-dependent relation structure for scheduling jobs. The element τ_{tj} is similar to τ_{ij}, which is designed to suit a dynamic environment.

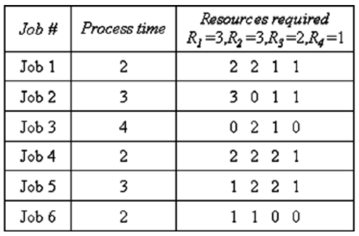
2.2. Scheduling problem

Most scheduling problems focus on minimizing either the maximum complete time (*makespan*) or the tardiness. However, maximizing performance, and minimizing *makespan* and scheduling time are the major issues in multiprocessor scheduling problems. Examples of such problems include scheduling jobs onto a fixed set of machines in a manufacturing plant, scheduling aircraft takeoffs and landings onto one or more landing strips, and scheduling meeting rooms for multiple events of varying size and length. Solving multiprocessor scheduling problems containing precedence and resource constraints is similar to the above examples, and is the major concern in this study. This work investigates a job scheduling problem involving non-preemptive multitasking with processing time, precedence and resource constraints conditions. However, meta-heuristic methods such as genetic algorithms, artificial neural networks, and simulated annealing are time-consuming. However, the ACO has been demonstrated to have a fast convergence rate in many applications. A modified ACO has been built to solve defined scheduling problems. The major difference between RCMPSP and RCPSP is that RCMPSP has a special resource type – processors (or machines), and one processor can only process one job (activity) at a time. The studied multiprocessor system comprises a number of identical (homogeneous) processors. In other words, this study focuses on homogeneous processors only, while the number of processors can be limited or unlimited.

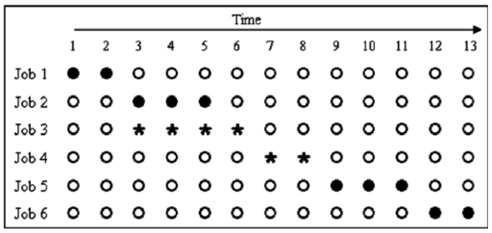
The formal assumptions of the scheduling problem domain are introduced in advance. Suppose that there are N jobs and M machines in a scheduling system. First, a job cannot be both segmented and preemptive. Second, a job cannot be assigned to different machines, which implies that no job migration is allowed between machines. Based on these assumptions, a set of job schedules is sought. Let $J = \{1,\ldots,N\}$ denote the set of jobs, and $m = \{1,\ldots,M\}$ represent the set of machines. Q denotes a set of resource totals of g types, and $R_i \geqq 0$ is the resource quantity for resource type i, $i \in Q$. Each job j, $j \in J$, has a duration p_j and resource requirements $r_{j,1,\ldots,r_{jg}}$, where $r_{j,i}$ denotes the requirement for a resource type i when processing job j. The value of $r_{j,i}$ does not change with time (Merkle et al., 2002). There are precedence relations between the jobs, and setup time is assumed to zero from one job switch to the next job. The precedence relations between the jobs can be represented by an acyclic activity-on-vertex (AOV) network. A job schedule list is mapping a set of tasks to a set of processors to meet the job precedence relations and resource requirements.

Fig. 2 shows a basic example of the problem domain studied, including a precedence graph and resources constraint with six jobs and four resource types on two machines. The total utilized resources can not be more than the total available resources for each resource type at a certain time. A two-dimensional matrix $(T \times N)$ is adopted to denote the scheduling result. The axes of the matrix are *job* and *time,* as denoted by j and t, respectively. The state of a coordinate is represented by V_{tj}.The value of V_{tj} is set to





Job #	Process time	Resources required $R_1=3, R_2=3, R_3=2, R_4=1$
Job 1	2	2 2 1 1
Job 2	3	3 0 1 1
Job 3	4	0 2 1 0
Job 4	2	2 2 2 1
Job 5	3	1 2 2 1
Job 6	2	1 1 0 0



Time

	1	2	3	4	5	6	7	8	9	10	11	12	13
Job 1	●	●	○	○	○	○	○	○	○	○	○	○	○
Job 2	○	○	●	●	●	○	○	○	○	○	○	○	○
Job 3	○	○	*	*	*	*	○	○	○	○	○	○	○
Job 4	○	○	○	○	○	○	*	*	○	○	○	○	○
Job 5	○	○	○	○	○	○	○	○	●	●	●	○	○
Job 6	○	○	○	○	○	○	○	○	○	○	○	●	●

Fig. 2. Simulation cases for six jobs with precedence and resource constraints and one solution matrix.

one ($V_{tj} = 1$) if job j is processed at time t, otherwise $V_{tj} = 0$. Every V_{tj} is associated with one τ_{tj} and one η_{tj}. Thus, unlike other approaches, the τ_{tj} and η_{tj} in the proposed approach is time-dependent.

3. Modified ACO algorithm for RCMPSP

Fig. 3 displays the steps of the scheduling algorithm for the resource-constrained multiprocessor scheduling problem by modified ACS, which combines the dynamic rule and delay solution generation rule, and is called a dynamic and delay ant colony system (DDACS). DDACS begins with a partial schedule containing no jobs at time 0. At each stage, a set of all eligible jobs $J_k(t)$, comprising all candidates for successors at time t. The initial jobs in $J_k(0)$ have in-degree = 0 which refers to the number of eligible jobs at time 0. The following jobs selected from $J_k\ (t)$ are applied until $m = M$ or not satisfying resource constraints. A job is selected by the ant from $J_k(t)$ if it satisfies resource and processor constraints. The processor constraint defines the most M jobs that can be assigned to M processors. After allocating one job to one processor so as to satisfy the resource constraints, C and $J_k(t)$ are then updated, where C denotes the set of already scheduled jobs. The algorithm runs until a stopping criterion is met, e.g., a certain number of generations have been performed or the average quality of the solutions found by the ants of a generation is unchanged for several generations.

The state transition rules are governed by Eqs. (6) and (7) (Park et al., 1994). The next job j is chosen from $J_k(t)$ when $q \leqq q_0$, which flavors the choices for the next job with the highest pheromone times heuristic value, where the η function is defined in Eq. (8):

$$j = \arg\max_{l \in J_k(t)} \left\{ [\tau(t,l)]^{\alpha} \cdot [\eta(t,l)]^{\beta} \right\} \tag{6}$$

If $q > q_0$, then job j is randomly selected from $J_k(t)$ according to the probability distribution given by the following equation:

$$P_k(t,j) = \begin{cases} \frac{[\tau(t,j)]^{\alpha} \cdot [\eta(t,j)]^{\beta}}{\sum\limits_{l \in J_k(t)} [\tau(t,l)]^{\alpha} \cdot [\eta(t,l)]^{\beta}}, & j \in J_k(t), \\ 0, & \text{otherwise}, \end{cases} \tag{7}$$

where α, β denote the parameters correlating to the importance of the pheromone and heuristic, respectively. Concerning heuristics, this study adopts the adaptations of priority heuristics known as the critical path method to determine the earliest/latest starting process time; E_j/L_j for job j, $j \in J$. The E_j and L_j are initially computed under no resource considerations, hence there is a conservative value for each job. However, the actual starting process time of some jobs is behind the L_j when involving resource

```
1.  Initialize
2.  Loop
3.    Each ant $k$ is positioned on a starting node
4.    Loop
5.      Initialize $t=1$, $m=0$, $C=\varnothing$ and $J_k(t=0)$
6.      Loop
7.        compute $m$ that has scheduled jobs at time $t$
8.        Select one job $j \in J_k(t)$ with state transition rule
9.        while $j \neq$ null and $m<M$ do
10.           if job $j$ at time $t$ under resource constraint satisfied
11.             if the delay rule is activated then delay job $j$
12.             else schedule job $j$ at time $t$
13.                 $m=m+1$
14.                 $C = C \cup \{ j \}$ and $J_k(t) = J_k(t) - \{ j \}$
15.                 applied local update rule
16.           Select next job $j \in J_k(t)$ which not been selected with state transition rule
17.       end while
18.       $t=t+1$
19.       add the eligible job to set $J_k(t)$ with current in-degree=0
20.     Until $|C|=N$
21.   Until all ants have built a complete solution
22.   A dynamic rule to adjust the $L_j$
23.   A global pheromone updating rule is applied with the best solution
24. Until End_condition is reached
```

Fig. 3. DDACS algorithm.

constraints. The L_j is used in the η function to build the initial solution. The L_j from the best solution of all ants is adopted in every iteration. Finally, the L_j is changed dynamically in the coming iteration according to dynamic rules given in Section 3.3. Eq. (8) shows the η function of the modified ACO:

$$\eta(t,j) = \begin{cases} \frac{1}{(d_j+1)\times\sqrt[c]{p_j+1}}, & \text{if } E_j \leqslant t < L_j, \\ \frac{1}{(2-d_j/c_1)\times\sqrt[c]{p_j+1}}, & \text{if } t \geqslant L_j, \end{cases} ,j \in J_k(t) \tag{8}$$

where $d_j = |L_j - t|$ and c, c_1 are large enough constant values.

Eq. (8) demonstrates that job j with the shortest process time (shortest p_j) and nearest to L_j (minimum d_j) obtains the highest η value. Job j with the highest probability ($P_k(t,j)$) is selected from $J_k(t)$ at time t. Hence, the job with minimum d_j and shortest p_j is first when $E_j \leqq t < L_j$, or the job with maximum d_j and shortest p_j is maximum first when $t \geqq L_j$.Once one job j is selected according to the state transition rule Eq. (6), then $V_{tj} = 1$, $t \in [S_j,f_j]$, where $S_j = t$ and $f_j = t + p_j - 1$. Restated, S_j (f_j) is the starting (finish) process time of job j in the current solution. Thus this setting ensures that the non-preemptive requirement is satisfied. Restated, V_{tj} is set to one during the time period of S_j to f_j. An unassigned job has high η value (>1/2) when the time $t > L_j$, and low η value (<1/2) when $t < L_j$ for jobs in $J_k(t)$. A job with a η value of 0 is not a member of $J_k(t)$. The η value of a job is close to 1/2 when $L_j = t$. However, the η value of a job is always between 0 and 1.

3.1. Local update rule

The pheromones τ_{tj} are updated by the local updating rule after an ant has built one RCMPSP solution. The modified ACS adopts the following local updating rule to prevent succeeding ants from searching in the neighborhood of the current schedule of the current ant. The ants select job j at time t, and then modify their pheromone levels:

$$\tau(t,j)_{new} = (1-\rho)\cdot\tau(t,j) + \rho\cdot\Delta\tau(t,j), \quad t = S_j, \tag{9}$$

where $0 < \rho < 1$ denotes the evaporation rate as an input parameter, where job j progresses from S_j to f_j. $\Delta\tau(t,j) = \tau_0$ is set in the proposed ACO method. If the pheromone τ_{tj} is set to a low value, then job j has a lower probability of being chosen by another ant at time t.

In Fig. 2, job $1 \in J_k(1)$ is the first job in the schedule at time 1, where $J_k(1) = \{1,5\}$. Thus, τ_{11} evaporates some pheromone lower than τ_{15}, according to the local update rule. At time 3, $J_k(3) = \{2,3,5\}$. If jobs 2 and 3 are selected, the related τ value (τ_{32} and τ_{33}) is decreased. Fig. 4 shows another feasible solution for the next ant, while job 5 is the first job to be assigned to the processor with the highest τ value. Such a solution has the smallest *makespan*, i.e. *makespan* = 11. The local update rule adopted to select another job is a strategy to avoid being trapped in a local maximum (or minimum).

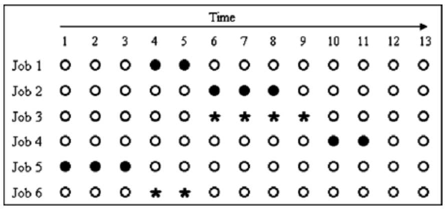


Fig. 4. Simulation result of another ant after the previous ant using the local update rule.

3.2. Global update rule

After all ants have built all feasible schedules, the global update rule, Eq. (10), is used to increase the pheromone τ_{tj} by applying the best solution so far. For all τ_{tj}, the pheromone is increased by the global update rate if $V_{tj} = 1$, where $t = S_j$, and is otherwise evaporated by global pheromone evaporation rate, as shown in Eq. (10). This is an elitist strategy that leads ants to search near the best-found solution:

$$\tau(t,j)_{\text{new}} = (1-\delta)\cdot\tau(t,j) + \delta\cdot\Delta\tau_{gb}(t,j), \quad t = S_j \tag{10}$$

where $0 < \delta < 1$ denotes a parameter representing the global pheromone evaporation rate, and

$$\Delta\tau_{gb}(t,j) = \begin{cases} \Delta ms, & \text{if } V_{tj} = 1, \\ 0, & \text{if } V_{tj} = 0 \end{cases} \quad \text{and}$$

$$\Delta ms = \frac{1 + \max\{0, ms_{\text{old}} - ms_{gb}\}}{ms_{gb}} \tag{11}$$

where $\Delta\tau_{gb}(t,j)$ is computed by the best schedule in the current iterations, and the amount of pheromone added is $\delta\Delta\tau_{gb}(t,j)$ when job j is assigned to run in time period $[S_j,f_j]$. The ms_{old} and ms_{gb} denote the *makespan* of the best schedule in the previous and current iterations, respectively. For each job, pheromone is added when a job is being processed in the job schedule list of the best solution obtained in the current generation. Otherwise, the pheromone is evaporated if $V_{tj} = 0$.

3.3. Dynamic rule

The studied multiprocessor scheduling system with resource constraints combines the RCPSP and task assignment. One job's earliest starting and latest starting time is computed by the critical path. The *makespan* is first assumed to be equal to the critical path length without considering the resource constraints. Owing to the resource constraints, the *makespan* may be larger than the critical path in the optimal solution. That is, the value of L_j increases along with the *makespan*. A schedule may contain some jobs that start to run behind L_j, which is a conservative value and is initially determined under no resource considerations, while the η function is based on the latest starting time, as shown in Eq. (8). If the L_j cannot reflect the actual latest starting time, then the L_j is an excessively conservative value for the state transition rule. Therefore, a rule is designed to refine the latest starting time by feedback of the best solution found in each iteration. This rule is called a "dynamic" rule. If the job is processed before the L_j, then the L_j does not need to be extended later. For those jobs that have been processed later than the L_j, the new L_j is replaced by the S_j. Restated, this replacement is used to acquire the most accurate value for L_j. This rule is adopted in step 22 of the DDACS procedure in Fig. 3. The accuracy of estimation of the η function value rises as the accuracy of L_j increases. The L_j dynamic adjustment rule is defined as follows:

$$L_j = \begin{cases} L_j, & \text{if } E_j < S_j \leqslant L_j, \\ S_j, & \text{if } L_j < S_j. \end{cases} \tag{12}$$

3.4. Delay solution generation rule

The delay solution generation rule (called the delay rule for short) is indicated in step 11 of DDACS. This rule enables some jobs to be assigned later on purpose to escape the local optimal solution. The delayed job is excluded from in $J_k(t)$ for a certain delay length, which is a uniform distribution of $[0, L_j - t]$ as demonstrated in Eq. (13). The delay is not later than L_j, hence theS_j of job j cannot be greater than L_j. One job can be processed later to let the other jobs be processed ahead to yield global optimal solution under the resource constraints. For instance, if one job requiring many resources at time t is selected and added into C, then some other jobs requiring same resources are prohibited from being processed for some time. Accordingly, these jobs result in a larger *makespan* than the optimal solution.

Fig. 6 depicts an example of this situation. Based on the proposed method without delay strategy, two jobs in $J_k(1) = \{1,2,3\}$ are schedule to run when two processors exist at $t = 1$. Suppose that jobs 1 and 2 are assigned for processing, and $J_k(2) = \{3,4,5\}$ at time 2. In this case, job 2 needs two R_3 resources and job 5 needs three R_3 resources. The total amount of R_3 resources available is 4, which is not sufficient for processing jobs 2 and 5 concurrently at $t = 2$. The solution is never optimal if job 2 is scheduled at $t = 1$. If jobs 2 and 3 are delayed to process, then the other jobs (jobs 4 and 5) can be executed earlier, and the successor jobs of jobs 4 and 5 can start to run as soon as possible. Hence, an optimal solution is obtained, since job 7 is a critical path job and processed earlier. The comparison between cases without delay strategy and with delay strategy is shown in Figs. 6 and 7.

The "delay" rule deliberately delays an eligible job, as shown in Eq. (13). This rule enables an undiscovered solution to be found. The delay time is defined as follows:

$$\text{delay time} = \begin{cases} q \times (L_j - t), & \text{if } q > q_1 \quad \text{and} \quad t \leqslant L_j \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where q is a random number uniformly distributed in [0,1]. The q_1 $(0 < q_1 < 1)$ is a predetermined parameter that determines the probability of changing the influence on the decisions of the ants. The rules in Eqs. (6) and (7) are adopted when $q \leqq q_1$. Otherwise, this delay strategy is applied when $q > q_1$ and $t < L_j$. The q_1 value increases along with the iteration. The q_1 value is close to one after certain iterations. Restated, the possibility of delaying jobs is decreased as the iteration increase.

The DDACS combines the above described rules to explore the search space of a feasible solution. The following simulations indicate these rules are suitable for resource-constrained multiprocessor problems.

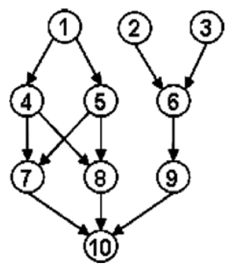
4. Experimental simulations

The simulations involved different sets of scheduling problems with different jobs, from 10 to 30 jobs on different processors, then later with 30–120 jobs. All simulation cases assumed that three or four different resource types were available and 10 ants were adopted. The simulations used various set of weighting factors. The q_0 value was set in the range of [0.8, 0.95]. The initial q_1 value was set in the [0.7, 0.95] range. Other settings were: iteration$_{\max} = 1000$, $\tau_0 = 0.01$, $c = 10$ and $c_1 = 50$. Moreover, $\delta = 0.1$, $\rho = 0.1$, $\alpha = 1$, $\beta = 1$, $q_0 = 0.9$ and $q_1 = 0.95$ were set in the simulation, if no other values are mentioned.

Fig. 5 shows the simplest case. This case involves 10 jobs, two or three processors with the given precedence and resource constraints. Figs. 6 and 7 indicate the scheduling results for two processors. Figs. 8 and 9 display the simulation results of three processors. Figs. 6 and 8 show the results of the no-delay rule used in the modified ACO. Figs. 7 and 9 display scheduling results of using the delay rule in the algorithm.

The *makespan* of the optimal solution in the two processors case is ten; thus the proposed approach with the delay rule can obtain the optimal solution in less than 10 iterations. Meanwhile, the simulation results of using three processors indicate that more processors do not improve the *makespan*, due to the resource constraints and because the jobs would be completely finished with two processors.

The following simulation cases are PSLIB cases, which are one special problem case of the studied RCMPSP. The PSLIB library has cases with 30–120 jobs; each case



JOB#	Process time	Resources required $R_1=5, R_2=6, R_3=4$
1	1	2 1 2
2	2	3 5 2
3	2	1 2 2
4	1	3 3 1
5	1	2 3 3
6	1	1 1 3
7	7	1 1 1
8	1	1 4 2
9	1	0 3 3
10	1	1 2 3

Fig. 5. Simulation cases for 10 jobs with precedence and resource constraint.

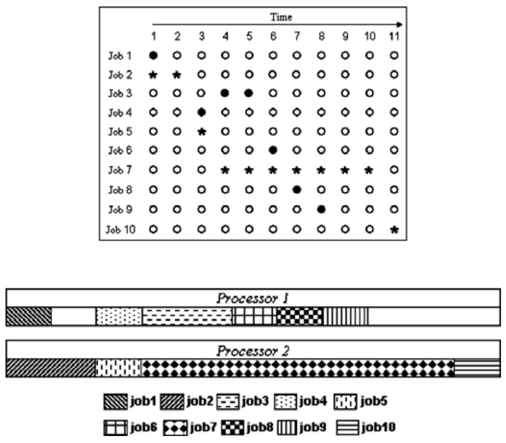


Fig. 6. The solution matrix and Gantt chart with no delay rule for two processors.

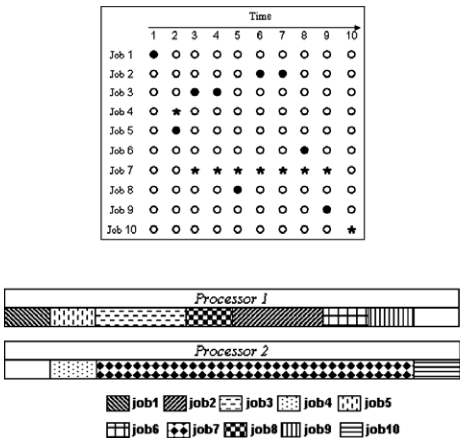


Fig. 7. The solution matrix and Gantt chart with delay rule for two processors.

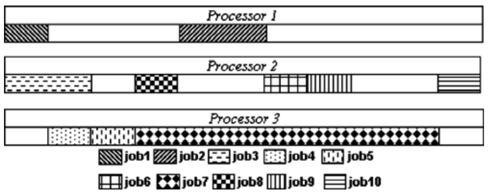


Fig. 8. The simulation result by Gantt chart with no delay rule for three processors.

is with at least 480 instances. The PSLIB includes project scheduling problems with no processor constraints. This work studies RCMPSP which has resources constraints. Restated, the PSLIB problems are special cases of

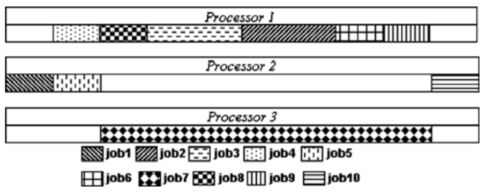


Fig. 9. The simulation result by Gantt chart with delay rule for three processors.

resource-constrained multiprocessor problems that assume the processors are unlimited. Thus, suggested DDACS can be directly applied to solve the PSLIB problems. The following simulations were used to test the proposed DDACS to check whether the optimal solutions can be found as listed in PSLIB. The other purpose of solving PSLIB using DDACS is to verify the designed dynamic rule and delay solution generation rule in obtaining a near-optimal (optimal) solution. The following simulation results indicate that the DDACS finds a near-optimal (optimal) solution under a certain number of iterations.

Table 1 shows the simulation results (job number, processor number, average, best, worst *makespan*, standard deviation of *makespan* and execution time) by proposed scheme. Each case was simulated 10 times; each simulation was set to run for 20 iterations. These simulation cases were set from 30 jobs to 120 jobs with intervals of 30 jobs and with different numbers of processors. All simulations were run on a Pentinum4 2.8 GHz PC using the C language.

Fig. 10 illustrates the simulation results of PSLIB for cases of 30 jobs with 480 instances. Simulation results indicate that the DDACS found more optimal solutions for PSLIB problems.

Fig. 11 shows the difference between computed *makespan* and optimal *makespan* for cases of 30 jobs with 480 instances. The total number of near-optimal solutions with dynamic and delay solution generation rules were greater than that obtained when no rule was employed. For instance, DDACS obtained about 93.3% (=448/480) cases with near-optimal solutions, in which the difference between the computed *makespan* and optimal *makespan* was no more than 2, as in Fig. 11c.

The q_1 value indicates the probability of not adopting the delay rule. Fig. 12 shows the numbers of optimal solutions found for different no delay probabilities q_1 with different q_0 values. Simulation results reveal that more optimal solutions found as high probability q_1 was set. Restated, a low delay probability (higher q_1 value) is suggested.

Table 1
Execution summary of one instance for 30–120 jobs

job	Processor	MS_avg	MS_best	MS_worst	MS_stdevp	Execution time
j30	4	44.2	43	45	0.5386	0.29
j60	5	96.3	95	98	0.9078	1.01
j90	6	121.8	120	123	0.6353	2.52
j120	11	92.3	91	94	1.0450	4.83

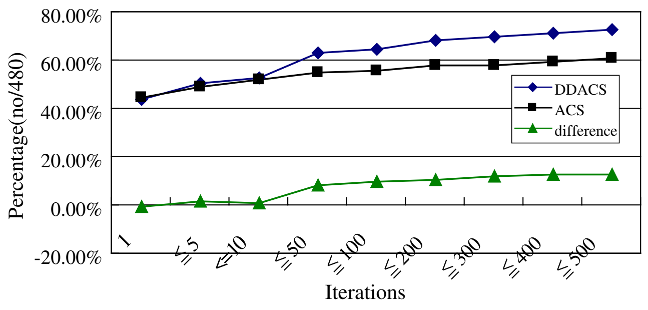


Fig. 10. Probability of finding optimal solutions comparison with ACS and DDACS.

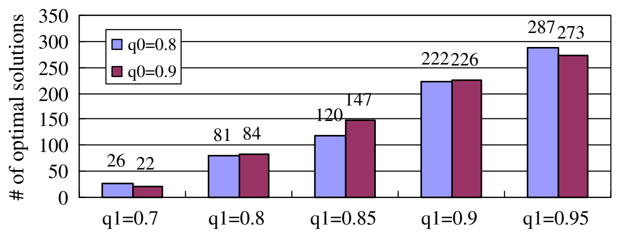


Fig. 12. The number of optimal solutions found for 30 jobs/480 different instances after 20 iterations with different delay probability.

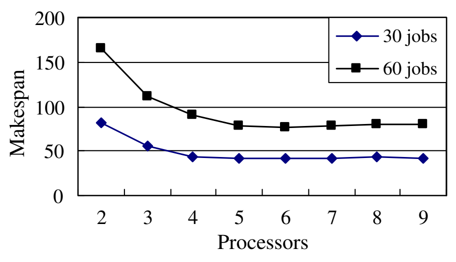


Fig. 13. The *makespan* for one 30 jobs and 60 jobs case after 100 iterations.

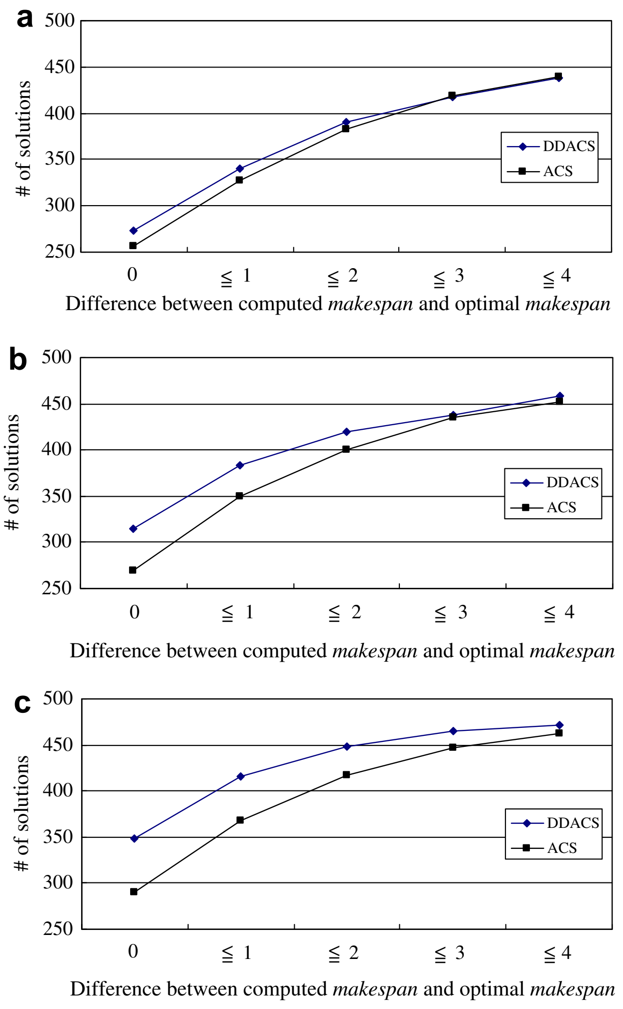


Fig. 11. The number of near-optimal solutions for 480 different instances with different iterations: (a) 20 iterations; (b) 100 iterations and (c) 500 iterations.

Fig. 13 depicts the *makespan* of the simulation results of cases with 30 and 60 jobs with different numbers of processor simulations. These two cases show that increasing the number of processors may not improve the schedule when precedence constraints are adopted. Different numbers of processors were set in the simulation. The minimum number of required processors was not considered in this study. Most of the simulations assumed that the number of processors in the multiprocessor system was sufficient. The simulation results easily reveal the requisite number of processors. The case with 60 jobs required about 5 processors, while that with 30 jobs needed 4 processors. If only 2 or 3 processors were available, then the *makespan* increased when the number of processors was not enough to process jobs simultaneously.

To better understand how α, β values affect optimal solutions obtained by proposed DDACS. Different α, β values were tested in simulation. Fig. 14 shows that a lower α value applied in DDACS is better than a high value. Restated, a lower α value yields more optimal solutions. Below, an α value that is equal to 1 and 2 under 500 iterations is shown. The best α, β values are set case by case based on the scheduling considerations of the real case. The best α and β values of this study are all set to 1.

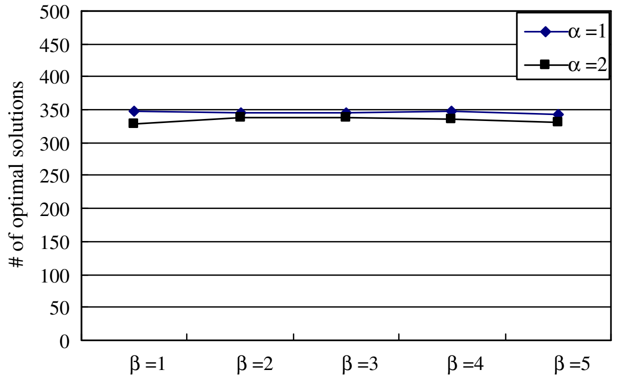


Fig. 14. The number of optimal solutions, found for one 30 jobs case after 500 iterations with different α, β values.

5. Conclusions and discussion

This study presents a modified ACO approach named DDACS for a multi-constraint (precedence and resource constraints) multiprocessor scheduling problem. A two dimension (time and job) matrix graph is adopted to represent the scheduling problem. This graph is used to resolve the minimum *makespan* schedule. The proposed DDACS algorithm modifies the latest starting time of each job in the dynamic rule for each iteration. The latest starting time of a job is used in the heuristic influence, as listed in Eq. (8). The latest starting time amendment provides an appropriate feedback to find the optimal solution. Moreover, a delay solution generation rule is applied to allow the solution to escape from the local minimum. The delay solution generation rule is a good strategy to search for a better solution, as revealed by the simulation results, as shown in Fig. 10.

The proposed DDACS scheme provides an efficient method of finding the optimal schedule of the multi-constraint multiprocessor system. However, the simulation results demonstrated some significant consequences for this study when applied to the scheduling domain. These were as follows:

1. The resource-constrained project scheduling problem is a special RCMPSP scheduling problem. Therefore, the proposed method can be applied to solve RCPSP directly without modification. Processors crashing or changing of available resources is an important consideration in multiprocessor systems. However, the proposed DDACS method is an adaptable scheme for such variable resource situations.
2. This method can be adopted to predict the minimum number of processors required in the multiprocessor system, as the results shown in Fig. 13, which indicates that increasing the number of processors to more than the required number do not improve the solution.
3. An important feature of the scheduling algorithm is its efficiency or performance, i.e., how its execution time increases with the problem size. A fast convergence rate is a significant characteristic of an ant colony system. The execution time of the DDACS algorithm is proportional to $O(N \times T \times ant)$ for one iteration instead of $O(N \times M \times T \times ant)$ if three dimensions matrix used. Restated, the execution time of DDACS is linear proportional to ant number and matrix size.

This work focuses on investigating multiprocessor system scheduling with precedence and resource constraints. The scheduling processors in this investigation are homogeneous processors. However, the more complex conditions, such as set-up time between jobs on a particular machine, and the communication cost of jobs running on different processors and heterogeneous processors, should be further studied. Meanwhile, a dynamic situation can be studied, with emergency jobs arriving at a certain time and with the changes in available resources. Moreover, the *makespan* is considered in this work, but tardiness is allowed in other scheduling problems, such as job-shop, flow-shop and industry production plans. Heuristic functions and how to generate better solutions can also be further discussed, and future research endeavors should address these issues more thoroughly.

References

Bauer, A., Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999). An ant colony optimization approach for the single machine total tardiness problem. In *Proceedings of the 1999 Congress on Evolutionary Computation, 1999* (pp. 1445–1450).

Besten, M. D., Sttzle, T., & Dorigo, M. (2000). Ant colony optimization for the total weighted tardiness problem. *Lecture notes in computer science* (Vol. 1917, pp. 611–620). Berlin, Germany: Springer-Verlag.

Brucker, P., Drexel, A., Möhring, R. H., Neumann, K., & Pesch, E. (1999). Resource-constraint project scheduling: Notation, classification, models, and methods. *European Journal of Operation Research, 112*(1), 3–41.

Cardeira C., & Mammeri, Z. (1996). Neural network versus max-flow algorithms for multi-processor real-time scheduling. Real-time systems. In *Proceedings of the Eighth Euromicro Workshop* (pp. 175–180).

Chen, R. M., & Huang, Y. M. (1998). Multiconstraint task scheduling in multiprocessor system by neural network. In *Proceedings of the IEEE 10th international conference on tools with artificial intelligence, Taipei* (pp. 288–294).

Chen, R. M., Lo, S. T., & Huang, Y. M. (2007). Combining competitive scheme with slack neurons to solve real-time job scheduling problem. *Expert Systems with Applications, 33*(1), 75–85.

Chen, R. M., & Huang, Y. M. (2001). Competitive neural network to solve scheduling problem. *Neurocomputing, 37*(1–4), 177–196.

Correa, R. C., Ferreira, A., & Rebreyend, P. (1996). Integrating list heuristics into genetic algorithms for multiprocessor scheduling. In *Parallel and distributed processing, eighth IEEE symposium* (pp. 462–469).

Correa, R. C., Ferreira, A., & Rebreyend, P. (1999). Scheduling multiprocessor tasks with genetic algorithms. *IEEE Transactions on Parallel and Distributed Systems, 10*(8), 825–837.

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

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transaction on System, Man and Cybernetics, 26*(1), 1–13.

Gajpal, Y., Rajendran, C., & Ziegler, H. (2004). An ant colony algorithm for scheduling in flowshops with sequence-dependent setup times of jobs. *European Journal of Operational Research, 155*(2), 426–438.

Herroelen, W. B., Reyck, D., & Demeulemeester, E. (1998). Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research, 13*(4), 279–302.

Hopfield, J. J., & Tank, D. W. (1985). Neural computation of decision in optimization problems. *Biological Cybernetics, 52*, 141–152.

Hou, E. S. H., Ansari, N., & Ren, Hong. (1994). A genetic algorithm for multiprocessor scheduling. Systems. *IEEE Transactions on Parallel and Distributed, 5*(2), 113–120.

Huang, Y. M., & Chen, R. M. (1999). Scheduling multiprocessor job with resource and timing constraints using neural network. *IEEE Transactions on System, Man and Cybernetics, Part B, 29*(4), 490–502.

Iredi, S., Merkle, D., & Middendorf, M. (2001). Bi-Criterion Optimization with Multi Colony Ant Algorithms. In *Proceedings of the first international conference on evolutionary multi-criterion optimization (EMO'01). Lecture notes in computer science* (Vol. 1993, pp. 359–372). Springer-Verlag.

Kwok, Y. K., Ahmad, I., & Gu, J. (1996). FAST: A low-complexity algorithm for efficient scheduling of DAGs on parallel processors. In *Proc. int'l conf. parallel processing (ICPP)* (vol. II, pp. 150–157).

Liu, C., & Layland, J. (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM, 20*(l), 46–61.

Maniezzo, V., & Carbonaro, A. (1999). Ant colony optimization: An overview. In *Proceedings of MIC'99, III metaheuristics international conference, Brazil*.

Merkle, D., & Middendorf, M. (2001). A new approach to solve permutation scheduling problems with ant colony optimization. In *Proceedings of the EvoWorkshops 2001*, *Lecture notes in computer science* (vol. 2037, pp. 484–494).

Merkle, D., Middendorf, M., & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation, 6*(4), 333–346.

Oh, J., & Wu, C. (2004). Genetic-algorithm-based real-time task scheduling with multiple goals. *Journal of Systems and Software, 71*(3), 245–258.

Park, J. G., Park, J. M., Kim, D. S., Lee, C. H., Suh, S. W., & Han, M. S. (1994). Dynamic neural network with heuristic. *IEEE International Conference on Neural Networks, 7*, 4650–4654.

Pierucci, P., Brandani, E. R., & Sogaro, A. (1996). An industrial application of an on-line data reconciliation and optimization problem. *Computers & Chemical Engineering, 20*, S1539–S1544.

Rajendran, C., & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research, 155*(2), 426–438.

Stützle, T., & Hoos, H. H. (2000). MAX–MIN ant system. *Future Generation Computer Systems, 16*(9), 889–914.

Topcuoglu, H., Hariri, S., & Wu, M. Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems Publication, 13*(3), 260–274.

Wu, M. Y., Shu, W., & Gu, J. (1997). Local search for DAG scheduling and task assignment. In *1997 international conference on parallel processing (ICPP '97)* (pp. 174–180).

Zomaya, A. Y., Ward, C., & Macey, B. (1999). Genetic scheduling for parallel processor systems: Comparative studies and performance issues. *IEEE Transactions on Parallel and Distributed Systems, 10*(8), 795–812.