Using an Enhanced Ant Colony System to Solve Resource-Constrained Project Scheduling Problem

Ruey-Maw Chen[†] *and Shih-Tang Lo*[††]

[†]Department of Computer Science and Information Engineering, National Chin-yi Institute of Technology
Taichung 411, Taiwan, ROC
[††]Department of Information Management, Kun-Shan University, Tainan 710, Taiwan, ROC
[††]Department of Engineering Science, National Cheng-Kung University, Tainan 701, Taiwan, ROC

Summary
This study presents and evaluates a modified ant colony optimization (ACO) approach for the resource-constrained project scheduling problems. A modified ant colony system is proposed to solve the resource-constrained scheduling problems. A two-dimensional matrix is proposed in this study for scheduling activities with time, and it has a parallel scheme for solving project scheduling problems. There are two designed heuristic is proposed. The dynamic rule is designed to modify the latest starting time of activities 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 project scheduling problems with resource constraints.

Key words:
Ant colony optimization, scheduling, project management

Introduction

In a project scheduling problem, given programs (tasks) of a project with precedence relation and resource requirements constraint are scheduled, such that the program's execution time is minimized. In practice, project management considers the precedence and resource requirements constraint and finding the minimum of maximum complete time, called the resource-constrained project scheduling problem (RCPSP) [1]. The traveling salesman problem (TSP) is a typical NP-complete problem, for which obtaining an optimal solution for a tour with a minimum distance is quite time consuming. 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. Most scheduling problems are confirmed to be NP complete problems. Many different schemes have been presented for solving scheduling problems. In this study, an ACO approach for the precedence and resource constrained projected scheduling problem is presented and evaluated.
The rest of this study is organized as follows. Section 2 reviews the related work for the resource-constrained project scheduling problems. Section 3 depicted the ant colony system and 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. Related work

Early research was concentrated mainly on the formulation and solution of the problem as a mathematical model; such as integer programming, branch-and-bound method and dynamic programming [2]. But these approaches can only be useful for small problems. The heuristic methods can solve the small problem. The heuristic methods are usually computationally efficient, but easily trap into local optimal solution and no guarantee that they will find optimal solutions. Recently, metaheuristics have been applied, such as tabu search, simulated annealing, genetic algorithm and ant colony optimization. Icmeli and Erenguc developed a tabu search method for RCPSP with the objective of optimizing the net income of the project [3]. Pinson et al. apply tabu search for RCPSP, based on a neighborhood search [4]. Baar et al. developed two tabu search algorithms: the first relies on elimination of critical arcs and list scheduling techniques, while the second relies on schedule schemes, where neighbors are generated by placing activities in parallel or deleting a parallelity relation [5]. Lee and Kim used a sequence of priority values for each task of a project as a representation scheme and apply simulated annealing, tabu search, and a genetic

algorithm to manipulate the priority sequence [6]. Tasks of a project are scheduled according to their priority values. They report better results than conventional heuristics. Naphade et al. employ problem space search, which is a local search metaheuristic that proved effective for a variety of combinatorial optimization problems [7]. They report extremely encouraging results comparable in performance to the branch and bound algorithm of Demuelemeester and Herroelen [8].

2.1 Project scheduling problems using genetic algorithm

The Genetic Algorithm (GA) is the most popular and widely used technique for project scheduling problem. Crossover, mutation and selection operators are applied to create for the new generation of schedules and find the solution with GA. GA's were first developed by Holland and De Jong and are based on the mechanics of natural selection in biological systems[9] [10]. Portmann use GA integrated with other search techniques, and analyses the effects of various GA characteristics, such as the encoding, crossover, and mutation operators on the performance of GA's in scheduling [11]. GA generates a high quality of output schedules in project scheduling problems, 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.

2.2 Ant system for project scheduling problems

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 [12]. 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. RCPSP using ACO algorithms has recently been studied [13][14][15]. The ACO has been successfully applied to RCPSP, combined with the different heuristics, easily obtaining a near-optimal solution. Dorigo and Gambardella first applied the ant colony system (ACS) to solve TSP [16]. 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 [17]. Besten et al. presented an application of the ACO meta-heuristic to the single machine total weighted tardiness problem [18]. Gajpal et al. adopted ACO to solve the problem of scheduling in flowshop with sequence-dependent setup times of jobs [19]. Rajendran and Ziegler developed two ant-colony optimization algorithms for solving the permutation flowshop scheduling problem [20]. Merkle et al. presented an ACO approach for RCPSP, which is a schedule problem to find the minimum *makespan* with resource and precedence constraints [15]. These studies indicate that ACO can work successfully in many different scheduling applications about combination problems. This work attempts to find optimal or near-optimal solutions to project schedule problems with resource and precedence constraints under restricted scheduling times. 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. Therefore, the concept can be adopted to solve project scheduling, job-shop, flow-shop, open-shop problems and grid computing problems.

3. Ant colony system and scheduling problem

3.1 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 [21][22]. 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.

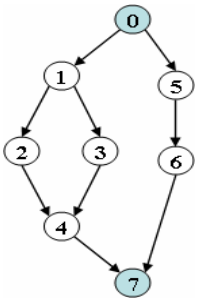
The ACO algorithm has recently been applied to scheduling problems, such as job-shop, flow-shop, and single machine tardiness problems [1][16][23][24]. Traditionally, the pheromone matrix $\tau=[\tau_{ij}]$, where the pheromone is added to an element τ_{ij} of the pheromone matrix, finds a good solution 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 [15]. Bauer et al. proposed ACO algorithms using a conventional pheromone matrix $[\tau_{ij}]$ to solve the single machine total tardiness problem and the flow-shop problem [23]. This study adopts a modified pheromone matrix $[\tau_{tj}]$, in which the element τ_{tj}, denoting the pheromone value of activity j

is processed at time t on a specific machine. Restated, the two-dimensional grid for scheduling activities at certain time is a time-dependent relation structure. The element τ_{tj} is similar to τ_{ij}, which is designed to suit a dynamic environment. The number of resource is changed by the time; the proposed scheme is easy to reschedule the remained activities from the time of resource changed.

3.2 Scheduling problem

The formal assumptions of the scheduling problem domain are introduced in advance. Suppose that there are N activities in a project. Based on these assumptions, let J= {0,1,…,N,N+1 } denote the set of activities, and activities cannot be both segmented and preemptive. Activity 0 is the only start activity that has no predecessor and activity N+1 is the only end activity that has no successor, and no resource requirements and has no processing time. Q denotes a set of renewable resource totals of k types, and $R_i \geqq 0$ is the resource quantity for resource type i, $i \in Q$. The availability of each resource type i, $i \in Q$ in each time period is R_i units. Each activity j, $j \in J$, has a duration p_j and resource requirements $r_{j,1}$…… r_{jq}, where $r_{j,i}$ denotes the requirement for a resource type i when processing activity j. The value of $r_{j,i}$ does not change with time [15]. There are precedence relations between the activities, and transmission time is assumed to zero from one activity switch to the next activity. The precedence relations between the activities can be represented by an acyclic activity-on-vertex (AOV) network. A network $G=(V,E)$ comprises a set of activities (vertex) $V=\{v_1,v_2,\ldots..,v_n\}$ and a set of edges $E=\{(i,j)|\ v_i\ ,\ v_j \in V\ \}$ representing the precedence relation. A schedule of a project is mapping a set of activities to meet the activity precedence relations and resource requirements.

Figure 2 shows a basic example of the problem domain studied, including a precedence graph and resources constraint with 6 activities and 4 resource types. 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 *activity* 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 one (V_{tj}=1) if activity 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.



(a)

Activity #	*Process time*	*Resource require* R_1=3,R_2=3,R_3=2,R_4=1
Activity 1	2	2 2 1 1
Activity 2	3	3 0 1 1
Activity 3	4	0 2 1 0
Activity 4	2	2 2 2 1
Activity 5	3	1 2 2 1
Activity 6	2	1 1 0 0

(b)

Figure 1. Simulation cases for 6 activities with precedence and resource constraints

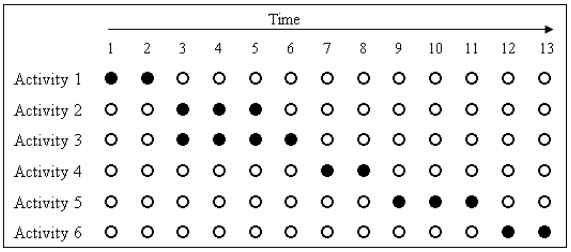


Figure 2. Simulation result with one solution matrix.

3.3 Modified ACO algorithm for RCPSP

Figure 3 displays the steps of the scheduling algorithm for the resource-constrained project 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 activity 0 at time 0. At each stage, a set of all eligible activities $J_k(t)$, comprising all candidates for successors at time t. The initial activities in $J_k(0)$ have in-degree=0 which refers to the number of eligible activities at time 0. The following activities selected from $J_k(t)$ are applied until resource constraints is not satisfied. An activity is selected by the ant from $J_k(t)$ if it satisfies resource constraints, C and $J_k(t)$ are then updated later, where C denotes the set of already scheduled activities. The algorithm runs until a stopping criterion is met, e.g., a certain number of generations have been performed.

To define the scheduling algorithm concisely, we define two auxiliary times: the *earliest* and *latest* start times of an activity. First, the earliest start time of activity j (E_j) is the length of the longest path from an entry activity to activity j. The latest start time of activity j.(L_j) is defined to be the latest time at which activity j may start, such that activity j and all successors of activity j have a chance to complete as soon as possible.

$$E_j = \begin{cases} 0 & , \text{if } \neg\exists i : (i,j) \in E \\ \max\limits_{i \in Pre(j)} \{E_i + p_j\} & , \text{otherwise} \end{cases} \quad (1)$$

$$L_j = \begin{cases} E_j + p_j & , \text{if } \neg\exists i : (j,i) \in E \\ \min\limits_{i \in succ(j)} \{L_i - p_j\} & , \text{otherwise} \end{cases} \quad (2)$$

where the $pre(j)$ ($succ(j)$) is the set of immediate predecessors (successors) of activity j, if $(i,j) \in E$. The earliest and latest start times of an activity is applied in η function determination. The L_j is used to build the initial solution. The state transition rules are governed by Eq. (3) and (4). The next activity j is chosen from $J_k(t)$ when $q \leqq q_0$, which flavors the choices for the next activity with the highest pheromone times heuristic value, where the η function is defined in Eq. (5).

$$j = \arg \max_{l \in J_k(t)} \left\{ [\tau(t,l)]^{\alpha} \cdot [\eta(t,l)]^{\beta} \right\} \quad (3)$$

If $q > q_0$, then activity j is randomly selected from $J_k(t)$ according to the probability distribution given by Eq. (4).

$$P_k(t,j) = \begin{cases} \dfrac{[\tau(t,j)]^{\alpha} \cdot [\eta(t,j)]^{\beta}}{\sum_{l \in J_k(t)} [\tau(t,l)]^{\alpha} \cdot [\eta(t,l)]^{\beta}} & , j \in J_k(t) \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

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 activity j, $j \in J$ in Eq. (1) and (2). The E_j and L_j are initially computed under no resource considerations, hence there is a conservative value for each activity. The L_j from the best solution of all ants is adopted in every iteration. Finally, the L_j is changed dynamically in the current iteration according to dynamic rules given in Section 3.3.3. Equation (5) shows the η function of the modified ACO.

$$\eta(t,j) = \begin{cases} \dfrac{1}{(d_j + 1) \times \sqrt[c]{p_j}} & , \text{if } E_j \leq t < L_j \\ \dfrac{1}{(2 - d_j / c_1) \times \sqrt[c]{p_j}} & , \text{if } t \geq L_j \end{cases} , j \in J_k(t) \quad (5)$$

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

Equation (5) demonstrates that activity j with the shortest process time (shortest p_j) and nearest to L_j (minimum d_j) obtains the highest η value. Activity j with the highest probability ($P_k(t,j)$) is selected from $J_k(t)$ at time t. Hence, the activity with minimum d_j and shortest p_j is first when $E_j \leqq t < L_j$, or the activity with maximum d_j and shortest p_j is first when $t \geqq L_j$.

1. Initialize
2. Loop
3. Each ant k is positioned on a starting node
4. Loop
5. Initialize $t=1$, $C = \varnothing$ and $J_k(t=0)$
6. Loop
7. Select one activity $j \in J_k(t)$ with *state transition rule*
8. while $j \neq$ null do
9. if activity j at time t under resource constraint satisfied
10. if the *delay rule* is activated then delay activity j
11. else schedule activity j at time t
12. $C = C \cup \{j\}$ and $J_k(t) = J_k(t) - \{j\}$
13. applied *local update rule*
14. Select next activity $j \in J_k(t)$ with state transition rule
15. end while
16. $t = t+1$
17. add the eligible activity to set $J_k(t)$ with current in-degree=0
18. Until $|C| = N$
19. Until all ants have built a complete solution
20. A *dynamic rule* to adjust the L_j
21. A *global pheromone updating rule* is applied with the best solution
22. Until End_condition is reached

Figure 3. DDACS Algorithm.

Once one activity j is selected according to the state transition rule Eq. (3) and (4), then V_{tj}=1, $t \in [S_j, f_j]$, where $S_j = t$ and $f_j = t + p_j - 1$. S_j (f_j) is the starting (finish) process time of activity 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 activity has high η value ($>1/2$) when the time $t > L_j$, and low η value ($<1/2$) when $t < L_j$ for activities in $J_k(t)$. An activity with a η value of 0 is not a member of $J_k(t)$. The η value of an activity is close to 1/2 when $L_j = t$. However, the η value of an activity is always between 0 and 1.

3.3.1 Local Update Rule

The pheromones τ_{tj} are updated by the local updating rule after an ant has built one RCPSP 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 activity j at time t, and then modify their pheromone levels.

$$\tau(t,j)_{new} = (1-\rho)\cdot\tau(t,j) + \rho\cdot\Delta\tau(t,j),\ t = S_j \qquad (6)$$

where $0<\rho<1$ denotes the evaporation rate as an input parameter, where activity 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 activity j has a lower probability of being chosen by another ant at time t. In Figure 2, activity $1 \in J_k(1)$ is the first activity 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 activities 2 and 3 are selected, the related τ value (τ_{32} and τ_{33}) is decreased. Figure 4 shows another feasible solution for the next ant, while activity 5 is the first activity to be assigned with the highest τ value. Such a solution has the smallest *makespan*, i.e. *makespan*=11. The local update rule adopted to select another activity is a strategy to avoid being trapped in a local maximum (or minimum).

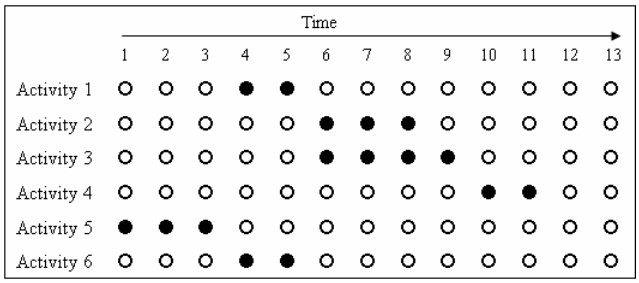


Figure 4. Simulation result of another ant after the previous ant using the local update rule.

3.3.2 Global update rule

After all ants have built all feasible schedules, the global update rule, Eq. (7), 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. (7). This is an elitist strategy that leads ants to search near the best-found solution.

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

where
$$\Delta\tau_{gb}(t,j) = \begin{cases} \Delta ms, & \text{if } V_{tj} = 1 \\ 0, & \text{if } V_{tj} = 0 \end{cases} \qquad (7)$$

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

where $0<\delta<1$ denotes a parameter representing the global pheromone evaporation rate, and 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 activity 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 activity, pheromone is added when an activity is being processed in the activity schedule list of the best solution obtained in the current generation. Otherwise, the pheromone is evaporated if V_{tj} =0.

3.3.3 Dynamic rule

In the beginning, one activity's earliest starting and latest starting time is computed by the critical path without considering the resource constraints. The *makespan* is first assumed to be equal to the critical path length at first iteration. Owing to the resource constraints, the *makespan* may be larger than the critical path length in the solution. The L_j is initially determined under no resource considerations. That is, the value of L_j may be increased along with the *makespan*. A schedule may contain some activities that start to run behind L_j, while considering the resource constraints The original L_j is a conservative value, while the η function is computed based on the latest starting time, as shown in Eq. (5). If the L_j can not reflect the actual latest starting time, then the L_j is an excessively conservative value for the state transition rule. The ants will choice the activity under the state transition rule with η value and pheromone τ value. Therefore, a rule is designed to refine the latest starting time by feedback of the best solution found in current iteration. This rule is called a "*dynamic*" rule. If the activity is processed before the L_j, then the L_j does not need to be extended later. For those activities that have been processed later than the L_j,

the new L_j is simply 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 20 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:

$$L_j = \begin{cases} L_j \ , \text{ if } E_j < S_j \le L_j \\ S_j \ , \text{ if } L_j < S_j \end{cases} \quad (8)$$

3.3.4 Delay solution generation rule

The delay solution generation rule (called the delay rule for short) is indicated in step 10 of DDACS. This rule enables some activities to be assigned later on purpose to escape the local optimal solution. The delayed activity 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. (9). The delay is not allowed to later than L_j, hence the S_j of activity j cannot be greater than L_j. This rule is enable one activity can be processed later to let the other activities be processed ahead to yield different solution under the resource constraints. For instance, if one activity is selected to schedule which require many resources at time t, then some other activities requiring same resources are prohibited from being processed for some time. Accordingly, these activities result in a larger *makespan* than the optimal solution.

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

The "delay" rule deliberately delays an eligible activity, as shown in Eq. (9). This rule enables an undiscovered solution to be found. The delay time is defined as follows.

$$delaytime = \begin{cases} q \times (L_j - t) \ , \text{ if } q > q_1 \text{ and } t \le L_j \\ 0 \ , \text{ otherwise} \end{cases} \quad (9)$$

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 rule in Eqs. (3) and (4) 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 activities is decreased as the iteration increase.

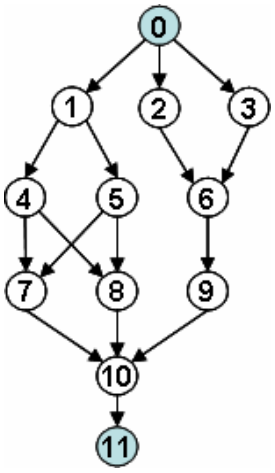
The DDACS combines the above described rules to explore the search space of a feasible solution. The following simulations indicate these rules are suitable for resource-constrained project scheduling problems.

4. Experimental simulations

The simulations involved different sets of scheduling problems with different activities, from 10 to 30 activities, then later with 30 to120 activities. 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, τ_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.

Figure 5 shows the 10 activities case with the given precedence and resource constraints. Figures 6 and 7 indicate the scheduling results. Figures 6 show the results of the no-delay rule used in the modified ACO. Figures 7 display scheduling results of using the delay rule in the algorithm.

Figure 8 show the corresponding optimal schedule obtained using the proposed ant approach. The allocation profiles of three types of resources are reflectively described in the schedule.



(a)

Activity#	*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

(b)

Figure 5. Simulation cases for 10 activities with precedence and resource constraint.

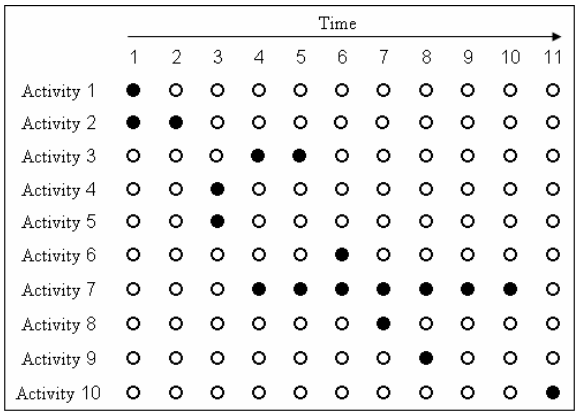


Figure 6. The solution matrix with no delay rule

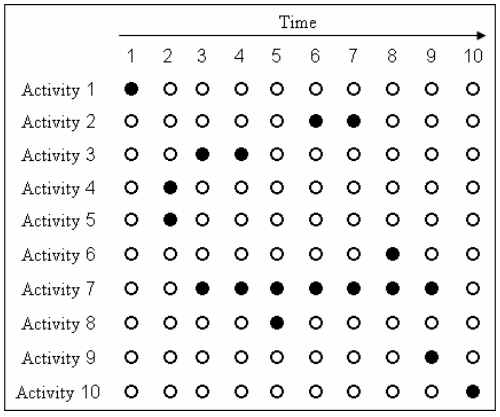


Figure 7. The solution matrix with delay rule

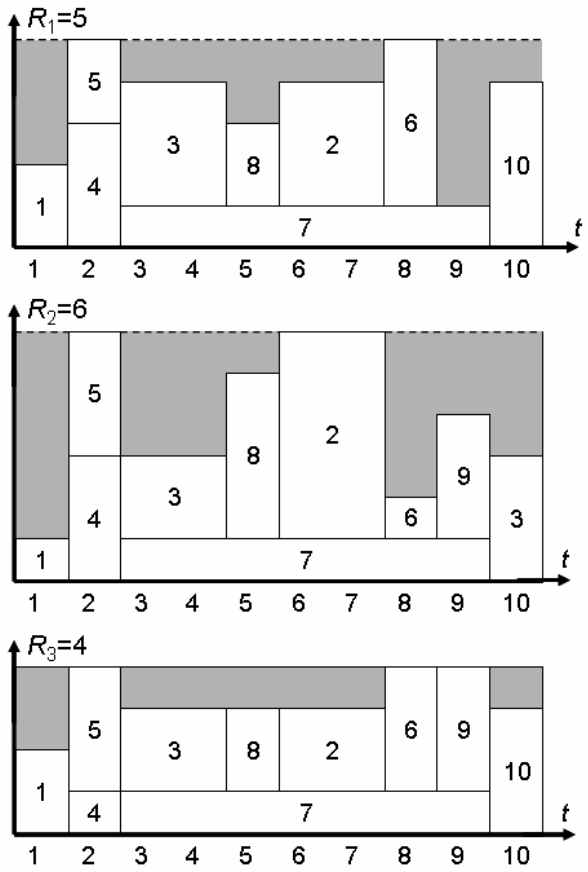


Figure 8. Resource allocation for one simulation result

The *makespan* of the optimal solution case is 10; thus the proposed approach with the delay rule can obtain the optimal solution in less than 10 iterations.

The following simulation cases are PSLIB cases. The PSLIB library has cases with 30 to 120 activities; each case is with at least 480 instances. Thus, suggested DDACS 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 show the simulation results (activity 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 activities to 120 activities. All simulations were run on a Pentinum4 2.8GHz PC using the C language.

Table 1. Execution summary of one instance for 30 to 120 activities.

activity	MS_avg	MS_best	MS_worst	MS_stdevp	Execution time
30	44.2	43	45	0.53	0.3
60	96.3	95	98	0.9078	1.02
90	121.8	120	123	0.6353	2.51
120	92.3	91	94	1.0450	4.83

Figure 9 illustrates the simulation results of PSLIB for cases of 30 activities with 480 instances. Simulation results indicate that the DDACS found more optimal solutions for PSLIB problems.

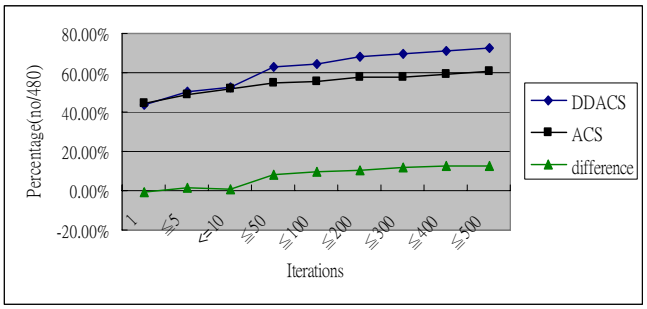


Figure 9. Probability of finding optimal solutions comparison with ACS and DDACS.

Figure 10 shows the difference between computed *makespan* and optimal *makespan* for cases of 30 activities 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. 10.

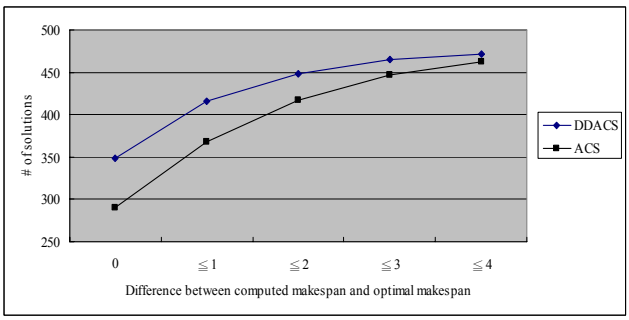


Figure 10. The number of near optimal solutions for 480 different instances with 500 iterations.

The proposed is suitable to solve the changing the available resource, if the resource is changed at time *t* while the schedule is under processing, then the DDACS can run immediately for the remainder unfinished activity. The reschedule process is under new resource circumstance. Figure 11 show the resource R_2 is changed from 6 to 5 at time 6. The activity 2, 6, 7, 9 and 10 are rescheduled at this time to satisfy the resource constraints. The two possible simulation results are showed in Fig. 12, the Figure 12-(b) is the solution by using the DDACS with delay rule.

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

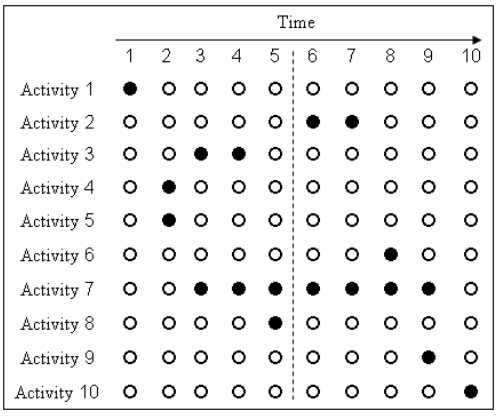
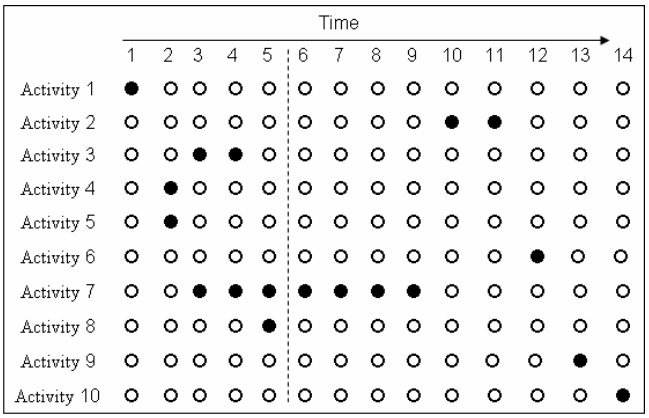
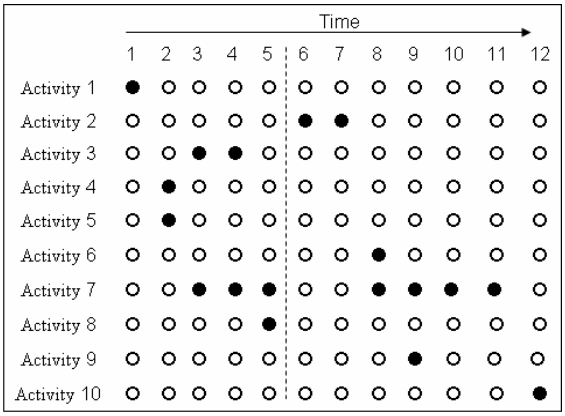


Figure 11. The simulation results for resource changed



(a)



(b)

Figure 12. The simulation results for resource changed

5. Conclusions and discussion

This study presents a modified ACO approach named DDACS for a multi-constraint (precedence and resource constraints) project scheduling problem. A two dimension (time and activity) matrix graph is adopted to solve the scheduling problem. The proposed DDACS algorithm modifies the latest starting time of each activity in the dynamic rule for each iteration. The latest starting time of an activity is used in the heuristic influence, as listed in Eq. (5). 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 or near schedule of the resource-constrained project scheduling problems. 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. Restated, the execution time of DDACS is linear proportional to ant number and matrix size. Changing of available resources is an important consideration in the resource-constrained project scheduling problem. However, the proposed DDACS method is an adaptable scheme for such variable resource situations. This work focuses on investigating the resource-constrained project scheduling problems. However, the more complex conditions, such as set-up time between activities or reschedule cost should be further studied. Meanwhile, a dynamic situation can be further studied, with emergency activities arriving at a certain time. 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

1. Merkle, D. & Middendorf, M. (2001). A new approach to solve permutation scheduling problems with ant colony optimization. In Proceedings of the Evo. Workshops 2001, number 2037 in Lecture Notes in Computer Science, pages 484–494. Springer Verlag.
2. A genetic algorithm approach to a general category project scheduling problem Ozdamar, L.; Systems, Man and Cybernetics, Part C, IEEE Transactions on Volume 29, Issue 1, Feb. 1999 Page(s):44 – 59
3. O. Icmeli and S. S. Erenguc, "A tabu search procedure for the resourceconstrained project scheduling problem with discounted cash flows," *Comput. Oper. Res.*, vol. 21, pp. 841–853, 1994.
4. E. Pinson, C. Prins, and F. Rullier, "Using tabu search for solving the resource-constrained project scheduling problem," in *Proc. 4th Int. Workshop Project Management and Scheduling*, Leuven, Belgium, 1993, pp. 102–106.
5. T. Baar, P. Brucker, and S. Knust, "Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem," in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I. Osman, and C. Roucarion, Eds. Norwell, MA: Kluwer, 1998, pp. 1–18.
6. J. K. Lee and Y. D. Kim, "Search heuristics for resource-constrained project scheduling," *J. Oper. Res. Soc.*, vol. 47, pp. 678–689, 1996.
7. K. Naphade, S.Wu, and R. Storer, "Problem space search algorithms for resource-constrained project scheduling," *Ann. Oper. Res.*, vol. 70, pp. 307–326, 1997.
8. E. Demeulemeester andW. Herroelen, "A branch-and-bound procedure for the multiple resource-constrained project scheduling problem,"*Manage. Sci.*, vol. 38, pp. 1083–1818, 1992.
9. J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michigan Press, 1975.
10. K. A. De Jong, "Adaptive system design: A genetic approach," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-10, pp. 566–574, 1980.
11. M. C. Portmann, "Genetic algorithms and scheduling: A state of the art and some propositions," in *Proc. Workshop Production Planning Contr.*, Mons, 1996.
12. Maniezzo, V. & Carbonaro, A. (1999). Ant Colony Optimization: an Overview. Proceedings of MIC'99, III Metaheuristics International Conference, Brazil
13. Brucker, P., Drexel, A., Möhring, R. H., Neumann, K., & Pesch, E. (1999). Resource-constraint project scheduling:

Notation, classification, models, and methods. Eur. J. Oper. Res., vol. 112, no. 1, pp. 3–41.
14. Herroelen, W., B. Reyck, D. & Demeulemeester, E. (1998). Resource-constrained project scheduling: A survey of recent developments. Comput. Oper. Res., vol. 13, no. 4, pp. 279–302.
15. Merkle, D. Middendorf, M. & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. IEEE Transactions on Evolutionary Computation, Vol. 6, Issue: 4 On page(s): 333- 346
16. 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.
17. Pierucci, P., Brandani, E. R., & Sogaro, A. (1996). An industrial application of an on-line data reconciliation and optimization problem. Computers & Chemical Engineering, Volume 20, Pages S1539-S1544.
18. Besten, M. D., Sttzle, T. & Dorigo, M. (2000). Ant Colony Optimization for the Total Weighted Tardiness Problem. Berlin, Germany: Springer-Verlag, vol. 1917, Lecture Notes in Computer Science, pp. 611-620.
19. 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, Volume 155, Issue 2, Pages 426-438
20. Rajendran, C. & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize *makespan*/total flowtime of jobs. European Journal of Operational Research, Volume 155, Issue 2, Pages 426-438
21. Stützle, T., Hoos, H. H. (2000). MAX-MIN Ant system. Future Generation Computer Systems, v.16 n.9, p.889-914.
22. 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.
23. Bauer, A., Bullnheimer, B., Hartl, R. F. & Strauss, C. (1999). An ant colony optimization approach for the single machine total tardiness problem. In Proc. 1999 Congr. Evolutionary Computation, 1999, pp. 1445-1450.
24. Iredi,S., Merkle, D., & Middendorf, M. (2001). Bi-Criterion Optimization with Multi Colony Ant Algorithms. In First International Conference on Evolutionary Multi-Criterion Optimization (EMO'01)., vol. 1993, Lecture Notes in Computer Science, pp. 359–372. Springer-Verlag.

Ruey-Maw Chen, he was born at Tainan, Taiwan, R.O.C. in 1960. He received the B. S., the M. S. and the PhD degree in engineering science from National Cheng Kung University of Taiwan R.O.C. in 1983, 1985 and 2000, respectively.

From 1985 to 1994 he was a senior engineer on avionics system design at Chung Shan Institute of Science and Technology (CSIST). Since 1994, he is a technical staff at Chinyi Institute of Technology. Since 2002, he has been with the Department of Computer Science and Information Engineering, National Chinyi Institute of Technology, where he is an assistant professor. His research interests include scheduling, and neural networks, computer networks.

Shih-Tang Lo, he was born at Hsinchu, Taiwan, R.O.C. in 1965. He received the B. S in computer science from Soochow University in 1987, the M. S. in Information Engineering from Tamkang University in 1989 and now is Phd student in engineering science, National Cheng Kung University of Taiwan R.O.C.

From 1991 until now, he is an instructor of department of information management, Kun-Shan University. His research interests include project scheduling, neural networks, genetic algorithm and ant colony optimization.