Multiprocessor System Scheduling with Precedence and Resources Constraints by Ant Colony System

Shih-Tang Lo[1], Ruey-Maw Chen[2], Chien-Jen Wang[3] and Chung-Lun Wu[4]
[1,3,]Department of Information Management, Kun-Shan University
[2]Department of Computer Science and Information Engineering, National Chin-yi Institute of Technology
[4]Department of electronic Engineering, National Chin-yi Institute of Technology
[1]*edwardlo@mail.ksu.edu.tw*, [2]*raymond@chinyi.ncit.edu.tw*, [3]*cjw@mail.ksu.edu.tw*
[4]*arashilen@yahoo.com.tw*

ABSTRACT

An ant colony optimization (ACO) approach for the precedence and resource constrained multiprocessor scheduling problem is presented and evaluated. The algorithm employs distributed agents which mimic the way of ants to find a feasible solution to complete jobs. Traditionally, the multiprocessor scheduling problems only consider the precedence constraint and finding the minimum of maximum complete time. In ACO, a set of ant-like agents or software ants solve the problem under consideration through a cooperative effort. A two-dimensional grid assigns jobs on processors scheme is proposed. This scheme is a time dependent structure in scheduling the jobs. In the exploration of the search space, a delay solution generation rules and dynamic rules are proposed, which are not only capable to escape the local optimal solution, but can get the feedback by previously constructed solutions. Simulation results reveal that developed ant colony system algorithm ensures an appropriate approach of solving multiprocessor system scheduling problems with precedence and resources constrains.

1: Introductions

In general, job scheduling problems are seen as involving to execute a set of jobs satisfying a given type of constraints and optimizing a given criterion [1]. Scheduling has many applications in commercial, industrial and academic fields, such as communications, production planning, project management, process scheduling in operating systems and class arrangement. The multiprocessor scheduling problems only consider the precedence constraint and finding the minimum of maximum complete time. An ACO approach for the precedence and resources constrained multiprocessor scheduling problem is presented and evaluated.

In the multiprocessor scheduling problem, given programs (tasks) with precedence relation within tasks are to be scheduled in a given multiprocessor system. The Genetic Algorithm (GA) is the most popular and widely used techniques for several flavors of the multiprocessor scheduling problem [2], [3]. Hou et al. develop an efficient method, the height value of each job, based on a GA to solve the multiprocessor scheduling problem [2]. Correa et al. proposed a new combined approach, where a genetic algorithm is improved with the introduction of some knowledge about the scheduling problem [4]. Zomaya et al. investigates an alternative paradigm, based on GA, to efficiently solve the parallel processor scheduling problem [5]. All above works are considered for multiprocessor scheduling only without containing the resource constraints. Oh and Wu carried out through a multi-objective GA to minimize processors required and the total tardiness of tasks [6]. GA generates good quality of output schedules, however, their scheduling times are usually much higher than the heuristic-based techniques [7] [8].

In ACO, a set of ant-like agents or software ants solve the problem under consideration through a cooperative effort. This effort is mediated by exchanging information on the problem structure the agents concurrently collect while stochastically building solutions [9]. Ant colony system (ACS) algorithm is applied to TSP that the ACS outperforms other nature-inspired algorithms such as simulated annealing and evolutionary computation [10]. Besten et al. present an application using ACO to slove the single machine total weighted tardiness problem [11]. Yuvraj et al. use ACO to solve the problem of scheduling in flowshop with sequence-dependent setup times of jobs [12]. The two ant-colony optimization algorithms were proposed for solving the permutation flowshop scheduling problem [13]. Merkle et al., present 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 [14]. Similarly, we propose a modified ACO algorithm for scheduling, in which a set of concurrent distributed agents collectively discover a feasible solution. Two suggested rules are applied in ACO algorithm to explore the search space and to find sound solutions. They are delay solution generation rules and dynamic rules.

In this work, a multiprocessor schedule problem with resource and precedence constraints is introduced to find

a near optimal solution while the scheduling time is restricted, now to be referred as resource-constrained multiprocessor scheduling problems (RMPSP). [17] A series of studies has been done using HNN and mean field annealing. A typical CHNN scheme was applied to the multiprocessor scheduling problems. All of these works assumed that there was no resource constraint, or each resource type had only one resource available and each job needed at most one resource for each type [15-17].

2: Ant colony system and scheduling problem

2.1 Ant colony system

Ant Colony Optimization algorithms could be good alternatives to existing algorithms for hard combinatorial optimization problems [18]. ACO mimics the behavior of foraging ants. Ants deposit pheromones along the paths they move. The pheromone level deposited on a particular path increases with the number of ants passing through that path. Ants use pheromones to communicate and cooperate with each another to identify shorter paths to the food source. Dorigo et al. proposed the first ACO algorithm [19] to solve the well-known traveling salesman problem (TSP), which method then evolved into the ant colony system (ACS) [20] as depicted in Figure 1.

```
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
```

Figure 1 ACS for TSP.

A graph $G=(V,E)$ comprises a set of nodes (vertex) $V=\{v_1,v_2,.....,v_n\}$ and a set of edges $E=\{(i,j)|\ v_i, v_j \in V\}$. Normally, each edge (i, j) is associated with one value representing a distance or cost. Each ant establishes a feasible solution to the TSP by repeatedly applying a 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 when $q \leqq q_0$ based on Eq. (1).

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

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 c_{ij}. $J_k(i)$ denote the set of nodes that 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

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

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 currently best tour. The rule is defined as,

$$\tau(i,j) \leftarrow (1-\rho)\tau(i,j) + \rho\Delta\tau(i,j) \qquad (3)$$

where $0<\rho<1$ is a parameter representing the local pheromone evaporation rate, and $\Delta\tau(i, j)=\tau_0$, 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 the best solution. The global updating rule enhances the edges involved in the globally best tour and is defined as,

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

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

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

L_{gb} represents the global best tour at current iteration.

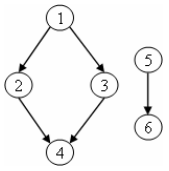
The ACO approach has been applied recently to scheduling problems, such as job-shop, flow-shop, and single machine tardiness problems. For the pheromone matrix (τ_{ij}), job j is the ith job on the machine. The ants directly use the value of τ_{ij} and heuristic function to estimate the desirability of placing job j as the ith job on the machine when computing a new solution [10][14][21][22][23]). In this work, a different pheromone matrix (τ_{tj}) is employed, which the element τ_{tj}, denoted the pheromone value of job j is processing at time t on one machine. The τ_{tj} is similar to τ_{ij} which is suitable for a dynamic environment.

2.2 Scheduling problem

Suppose there are N jobs and M identical machines. First, a job cannot be segmented (preemptive) and no job migration. There are precedence relations between the jobs, and setup time is set to zero from one job switch to next job. The precedence relations between the jobs can

be represented by an acyclic activity-on-node network. Q is a set of h resource types. $R_i \geqq 0$ is the resource capacity for resources of type $i \in Q$, Every job j has a duration p_j and resource requirements $r_{j1} \ldots\ldots r_{jh}$, where r_{ji} is the requirement for a resource of type i when job j is processed [14].

In Figure 2, there are 6 jobs and 4 resources type on two machines case. At each time, the total resource allocation is not more than the total available resource for each type resources. The two dimension matrix ($N*T$) is employed in this work, which is to represent the scheduling result, $V_{tj}=1$ if job j is processed at time t, otherwise $V_{tj}=0$. Each V_{tj} is associated with one τ_{tj} and η_{tj}.



Job #	Process time	Resource require 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

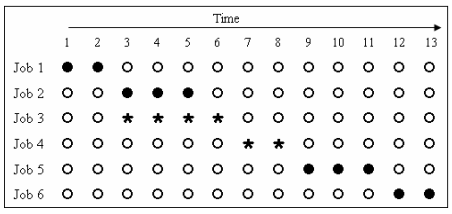


Figure 2. Simulation cases for 6 jobs with precedence and resource constraint and one solution matrix.

3: ACO algorithm for RMPSP

Figure 3 lists the scheduling algorithm for the scheduling problem, which combing the dynamic rule and delay solution generation rule is named as DDACS.

```
1.  Initialize
2.  Loop
3.    Each ant k is positioned on a starting node
4.      Loop
5.        Initialize t=1, m=0, C =∅ and Jk(t=0)
6.        Loop
7.          compute m the have scheduled job in time t
8.          while Jk(t) ≠∅ and m≦M do
9.              Select one job j ∈ Jk(t) with state transition rule
10.             if job j at time t under constraints 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 ∪ {j} and Jk(t) = Jk(t) -{j}
15.                     applied local update rule
16.         end while
17.         t=t+1
18.         add the eligible job to set Jk(t) with in-degree=0
19.       Until   | C | =N
20.     Until all ants have built a complete solution
21.     A dynamic rule to adjust the Lj
22.     A global updating rule is applied with the best solution
23. Until End_condition is reached
```

Figure 3. DDACS Algorithm.

A set of all eligible jobs $J_k(t)$ is computed, which are all predecessors finished satisfied at time t. The initial $J_k(0)$ is with in-degree=0 which is the eligible jobs at time 0. A job is selected by the ant from $J_k(t)$ and scheduled at time t. The processor constraint is defined that there are at most M jobs can be assigned to M processors. After selecting one job to one processor which satisfies the resource constraints, then, C and $J_k(t)$ are updated, where C is the set of already scheduled jobs. The algorithm runs until some stopping criterion is met, e.g., a certain number of generations have been done. According to state transition rule given by Eq. (6) and (7), the next job j is chosen from $J_k(t)$ when $q \leqq q_0$ which flavors the choices for the next job by closing to Lj ,shortening processing time and with a large amounts of pheromone[24].

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

If $q>q_0$, job j is randomly selected from $J_k(t)$ according to the probability distribution given by Eq. (7).

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

where α, β are the parameters correlating to the importance of pheromone and heuristic, respectively. Concerning heuristics, this study uses the adaptations of priority heuristics called critical path method to compute the earliest/latest starting process time; E_j /L_j for job j. At the beginning, the E_j and L_j is under no resource consideration, it is a more conservative value for each job. The L_j is changed dynamically in coming iteration based on dynamic rule showed in section 3.1. The η function is shown in Eq. (8).

$$\eta(t,j) = \begin{cases} \dfrac{1}{(d_j+1) \times \sqrt[c]{p_j+1}}, & \text{if } E_j \leq t < L_j \\ \dfrac{d_j}{(2-d_j/c) \times \sqrt[c]{p_j+1}}, & \text{if } t \geq L_j \end{cases}, \; j \in J_k(t) \qquad (8)$$

where $d_j = |L_j - t|$ and c is a large constant value

It shows that the job j with the shortest process time and nearest L_j will get higher η value. The job j has higher probability is selected from $J_k(t)$ at time t, which the d_j is minimum first when $L_j \leqq t < L_j$, or which the d_j is maximum first when $t>L_j$. Once one job j is selected based on the state transition rule, then $V_{tj}=1$, $t \in [s_j, f_j]$, where start processing time $s_j = t$ and finish time $f_j = t+p_j -1$, which is to ensure non-preemptive requirement satisfied.

An ant has built one solution of RMPSP, the pheromones τ_{tj} are updated using the local updating rule.

$$\tau(t,j)_{new} = (1-\rho) \cdot \tau(t,j) + \rho \cdot \Delta\tau(t,j), \; t \in [s_j, f_j] \qquad (9)$$

where job j processed from s_j to f_j ,and set $\Delta\tau(t, j)=\tau_0$ The pheromone τ_{tj} is set to a lower value, and then this job j will have lower probability to be the choice for

another ant. The reason for this is that old pheromone should not have too strong an influence on the future.

Figure 4 is another feasible solution for next ant, the job 5 is the first job to assign to processor which has the higher τ value. Such solution has the smaller *makespan*, i. e. *makespan*=11. The local update rule used to select another job is a strategy to avoid trap into local maximum (or minimum).

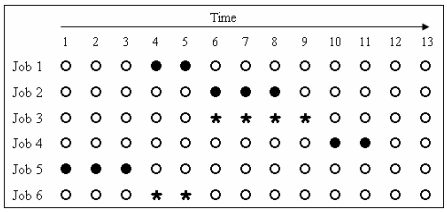


Figure 4. Simulation result of another ant after the previous ant using local update rule.

After all ants build all feasible schedules, the global update rule is used to increment the pheromone τ_{tj} by applying the best solution so far. All τ_{tj} , which the V_{tj} =1, get enhanced pheromone is updated by the global update rule, otherwise evaporating some amount of pheromone by global pheromone evaporation rate. This is an elitist strategy that leads ants to search near the best-found solution.

$$\tau(t,j)_{new} = (1-\delta)\cdot\tau(t,j) + \delta\cdot\Delta\tau_{gb}(t,j)\,,\ t\in[s_j, f_j] \quad (10)$$

$$\Delta\tau_{gb}(t,j) = \begin{cases} \Delta ms\ , & \text{if } V_{tj}=1 \\ 0\ , & \text{if } V_{tj}=0 \end{cases},\ t\in[s_j, f_j] \quad (11)$$

$$\text{and} \qquad \Delta ms = \frac{1 + ms_{old} - ms_{gb}}{ms_{gb}}$$

where $\Delta\tau_{gb}(t,j)$ is computed by the best schedule in the current iterations, the amount of pheromone added is $\delta\Delta\tau_{gb}(t,j)$. The ms_{old} and ms_{gb} is the *makespan* of the best schedule in previous and current iteration.

3.1 Dynamic rule

One job's earliest starting and latest starting time is computed by critical path. At first, it is assumed that the *makespan* is equal to critical path length without consideration of the resource constraint. Due to the resource constraint, the *makespan* may be greater than the critical path in the optimal solution. That is the critical path may be longer, then the value of *Lj* have to increase, while the η function is based on the latest starting time. This rule is used for the best solution found in each iteration. This rule is named as a "*dynamic*" heuristic rule. The more accurately *Lj* is found, then the more appropriate η function value can be estimated. The job tardiness and *Lj* adjusting rule is defined as follow:

$$L_j = \begin{cases} L_j\ , \text{ if } E_j < S_j \le L_j \\ S_j\ , \text{ if } L_j < S_j \end{cases} \quad (12)$$

3.2 Delay solution generation rule

The delay solution generation rule allows some jobs to be pushed later, which is designed to escape the local optimal solution. One job *j* is selected and has to be delayed, this job will not in $J_k(t)$ for a certain delay length, which is a uniform distribution of [*0, Lj – t*] as shown in Eq. (13). This assumption is that one job can be processed later to let the other jobs be processed first under the resource constraint. If there is exists one job that requires much of the resources scheduled at time *t*, then it will prohibit some other jobs from being processed for some time. But these jobs will result in a *makespan* that is greater than the optimal solution.

Figure 5 is an example for 10 jobs with 3 resource types and precedence relations. Based on proposed method without delay strategy, it will choose 2 jobs from $J_k(1)$={1,2,3} to be run when 2 processors exist at *t*=1. Assume that job 1 and 2 are assigned to be processed. And then the $J_k(2)$ ={3,4,5} at time 2. In this case, job 2 needs resources 2 of R_3 and job 5 needs resources 3 of R_3. The total available amount of resource R_3 is 4. It is not sufficient for job 5 processed concurrently when *t*=2, neither to job 3 or 4. And it is never an optimal solution if job 2 is scheduled at *t*=1. If job 2 or 3 can be pushed later to be processed, then the other jobs (job 4 and 5) can be assigned earlier. And the successor jobs of job 4 and 5 can be started as soon as possible. The job 7 can be processed early, which is a critical path job. The comparison is showed in Figure 6 and 7.

To delay the eligible job on purpose is called the "delay" rule. It is a Consideration of this rule enables the finding of feasible solution space. The delay time is defined as following:

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

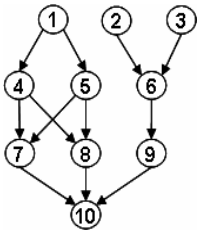
where q is a random number uniformly distributed in [0,1], and $0 \leqq q_1 \leqq 1$ is an predetermined parameter that determines the probability of changing the influence on the decisions of the ants during the run of the algorithm, and the option that an elitist ant forgets the found solution. The q_1 value increases close to 1 after certain iteration using $q_1 = q_1 + ln(1+\text{iteration\#} / \text{iteration}_{max})$

The DDACS combing the dynamic rule and delay solution generation rule is to explore the search space of undiscovered solution.

4: Experimental simulations

The simulations involve different sets of scheduling problems with different jobs from 10 to 60 jobs. All simulation is assumed that there are 3 or 4 different type resources and using 10 ants. The following simulation set is iteration_{max}=10000, τ_0=0.5, *c*=10, δ=*0.1,* ρ=*0.1,* α=*1*, β=*1,* q_0 =*0.9* and q_1 =*0.95*, if no other values are mentioned.

Figure 5 is the 10 jobs case using 2 processors. Figure 6 is the result of no delay rule with the state diagram and Gantt chart. Figure 7 is scheduling result using delay rule. If the delay rule is unused, the optimal schedule can not be found at all.



JOB#	Process time	Resource require $R_1=5, R_2=6, R_3=4$
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

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

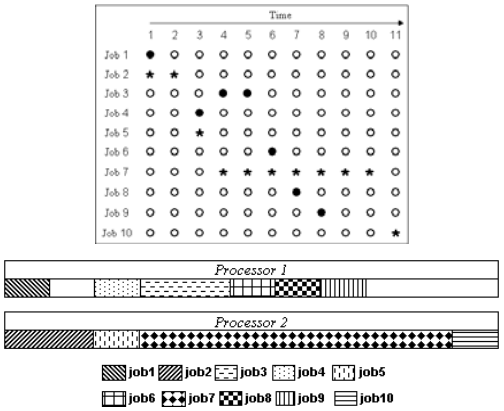


Figure 6. The solution matrix and Gantt chart with no delay rule for 2 processors.

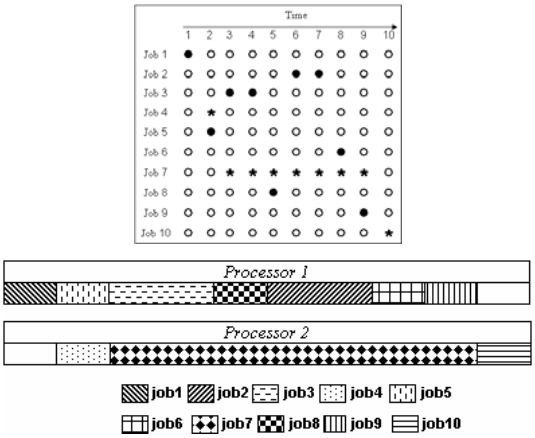


Figure 7. The solution matrix and Gantt chart with delay rule for 2 processors.

The following cases are from the PSLIB which is the problems of RCPSP. In this library there are 30 to 120 jobs case, each has at least 480 instances. The PSLIB about project scheduling problem which is no processor constraint involved which is a special case of proposed problem. It is a simple case for RMPSP which assumed the processor is unlimited or these problems are no processor. The processors number is an input parameter or assumed to contain enough processors.

The following simulation cases as shown in Figure 8 are for 30 jobs /case. Simulation results show that DDACS work more efficient to find more number of optimal solutions, the suggested ant algorithm with appropriate rule design increases the probability of finding the optimal solutions.

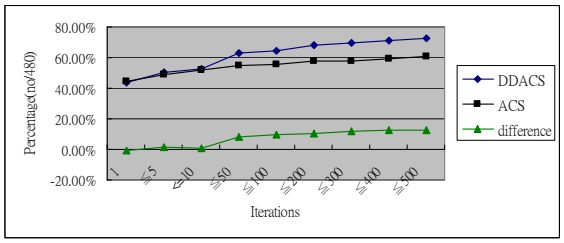


Figure 8. Probability of finding optimal solutions comparison with ACS and DDACS.

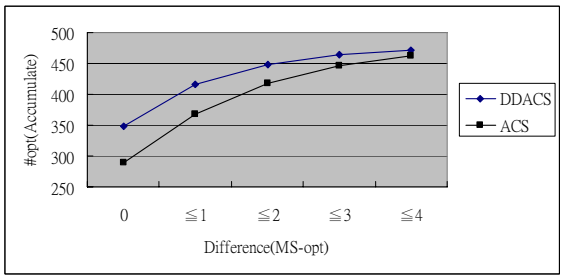


Figure 9. The number of the solution found near optimal solution for 480 different instances after 500 iterations.

Figure 9 shows the difference between the *makespan* with the optimal solution for 30 jobs with 480 instances (after 500 iterations). The total number of near optimal solution with delay solution generation rule is superior to not employing the rule. For example, the DDACS get about 93.3% (=448/480) cases near the optimal solution, and the difference of *makespan* with optimal solution is no more than 2.

Figure 10 shows *makespan* of the simulation results for a 30 jobs case and a 60 jobs case. These two cases show that more processors than needed make no improvement to the schedule with precedence and resource constraints. The minimum processors needed are not considered in this work. It can easy be found which numbers of processor are sufficient, or to anticipate the processor requirement from the simulation results. The 60 jobs case needs about 5 processors, the 30 jobs case needs 4 processors.

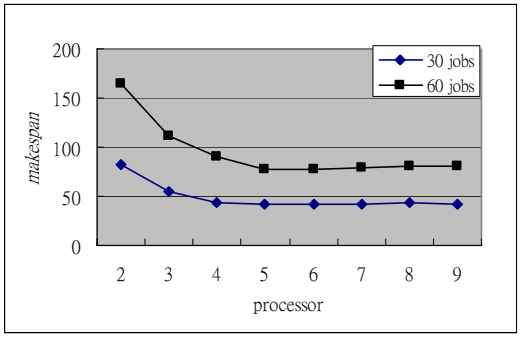


Figure 10. The *makespan* for one 30jobs case after 100 iterations with different processors.

5: Conclusions and discussion

This work introduces an ACO approach for precedence and resource constrained multiprocessor scheduling problems. Additional features added in the proposed algorithm are the changing of the latest starting time of each job for heuristic influence, and the delay solution generation rule to escape the local minimum. The resource-constrained project scheduling problem is a special kind scheduling problem of RMPSP. The proposed method can be applied to solve RCPSP directly without modification. This approach also provides a method to predict the minimum number of processors required in multiprocessor system. The α, β parameters determination in ACO algorithm is quite time consuming. The suggested scheme uses the critical path characteristic to reduce the inconvenience in finding these two parameters.

Compared to other meta-heuristics such GA, SA and tabu search, relatively less attempts have been made to solve multi-processor scheduling problem using the ant colony algorithms.

Future works should examine the setup time between jobs of a certain machine, or if there is communication cost between two jobs which are processing in different processors. In a dynamic situation, there may be some emergency jobs arriving at a certain time or changing the resources available and requirement.

REFERENCES

1. Cardeira, C. & Mammeri, Z. Neural network versus max-flow algorithms for multi-processor real-time scheduling, Real-Time Systems, Proceedings of the Eighth Euromicro Workshop ,Page(s):175 – 180,1996
2. Hou, E.S.H.; Ansari, N.; Hong Ren; A genetic algorithm for multiprocessor schedulingParallel and Distributed Systems, IEEE Transactions on Volume 5, Issue 2, Feb. 1994 Page(s):113 – 120
3. Correa, R.C.; Ferreira, A.; Rebreyend, P., Integrating list heuristics into genetic algorithms for multiprocessor scheduling, Parallel and Distributed Processing, 1996. Eighth IEEE Symposium on 23-26 Oct. 1996 Page(s):462 – 469
4. Correa, R.C.; Ferreira, A.; Rebreyend, P., Scheduling multiprocessor tasks with genetic algorithms IEEE Transactions on Parallel and Distributed Systems, Volume 10, Issue 8, Aug. 1999 Page(s):825 – 837
5. Zomaya, A.Y.; Ward, C.; Macey, B., Genetic scheduling for parallel processor systems: comparative studies and performance issues IEEE Transactions on Parallel and Distributed Systems, Volume 10, Issue 8, Aug. 1999 Page(s):795 – 812
6. Jaewon Oh and Chisu Wu, Genetic-algorithm-based real-time task scheduling with multiple goals, Journal of Systems and Software, Volume 71, Issue 3, May 2004, Pages 245-258
7. Min-You Wu; Wei Shu; Jun Gu, Local search for DAG scheduling and task assignment, Parallel Processing, 1997., Proceedings of the 1997 International Conference on 11-15 Aug. 1997 Page(s):174 – 180
8. Yu-Kwong Kwok; Ahmad, I.; Jun Gu, FAST: a low-complexity algorithm for efficient scheduling of DAGs on parallel processors, Parallel Processing, 1996., Proceedings of the 1996 International Conference on Volume 2, 12-16 Aug. 1996 Page(s):150 - 157 vol.2
9. Maniezzo V. and Carbonaro A. (1999). Ant Colony Optimization: an Overview. Proceedings of MIC'99,III Metaheuristics International Conference, Brazil
10. M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," IEEE Trans. Evol. Comput., vol. 1, pp. 53-66, Apr. 1997.
11. M. den Besten, T. Sttzle, M. Dorigo, Ant colony opt imization for the total weighted tardiness problem, Parallel Problem Solving from Nature: 6th international conference, September 2000. Springer Verlag.
12. Yuvraj Gajpal, Chandrasekharan Rajendran and Hans Ziegler, An ant colony algorithm for scheduling in flowshops with sequence-dependent setup times of jobs, European Journal of Operational Research, Volume 155, Issue 2, 1 June 2004, Pages 426-438
13. Chandrasekharan Rajendran and Hans Ziegler, Ant-colony algorithms for permutation flowshop scheduling to minimize *makespan*/total flowtime of jobs, European Journal of Operational Research, Volume 155, Issue 2, 1 June 2004, Pages 426-438
14. Merkle, D. Middendorf, M. Schmeck, H., Ant colony optimization for resource-constrained project scheduling IEEE Transactions on Evolutionary Computation, Aug. 2002, Volume: 6, Issue: 4 On page(s): 333- 346
15. Hopfield, J.J.,& Tank, D.W.(1985). Neural computation of decision in optimization problems. Biological Cybernetics , 52, 141-152.
16. Huang, Y.M., & Chen, R.M.(1999). Scheduling multiprocessor job with resource and timing constraints using neural network. IEEE Trans. on System, Man and Cybernetics, part B, 29(4): 490-502.
17. Chen R.M.,& Huang Y.M. (2001). Competitive Neural Network to Solve Scheduling Problem. Neurocomputing, 37(1-4):177-196.
18. Thomas Stützle , Holger H. Hoos, MAX-MIN Ant system, Future Generation Computer Systems, v.16 n.9, p.889-914, June 2000
19. M. Dorigo, V. Maniezzo and A. Colorni. The ant system: Optimization by a colony of cooperating agents. IEEE Transaction on System, Man and Cybernetics 1996; 26(1): 1-13.
20. M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1997; 1(1): 53-66.
21. Bauer, A.; Bullnheimer, B.; Hartl, R.F.; Strauss, C.; An ant colony optimization approach for the single machine total tardiness problem Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on Volume 2, 6-9 July 1999 Page(s):
22. Steen Iredi, Daniel Merkle, and Martin Middendorf, Bi-Criterion Optimization with Multi Colony Ant Algorithms
23. D. Merkle and M. Middendorf., A new approach to solve permutation scheduling problems with ant colony optimization. In Proceedings of the EvoWorkshops 2001, number 2037 in Lecture Notes in Computer Science, pages 484–494. Springer Verlag, 2001.
24. Park, J.G., Park, J.M., Kim, D.S., Lee, C.H., Suh, S.W., & Han, M.S. (1994). Dynamic neural network with heuristic. In: IEEE Int. Conf. on Neural Networks, vol. 7, pp. 4650-4654.