Pakistan Journal of Information and Technology 2(1): 13-20, 2003

Non-Preemptive Scheduling of Real-Time Tasks under Precedence and Resource Constraints

A. Mahmood
Department of Computer Science, University of Bahrain, Bahrain

Abstract: Optimal scheduling of real-time tasks in multiprocessor systems is known to be computationally intractable for large task sets. Therefore, heuristic approaches seem appropriate to these classes of problems. In this paper, we propose a tabu search algorithm for nonpreemptive scheduling of periodic real-time tasks in multiprocessor systems under precedence and resource constraints. The real-time tasks are characterized by their periods, deadlines, worst-case computation times and precedence and resources constraints. We present various strategies to obtain the initial solution, neighborhood solutions and the best move. The performance of the proposed algorithm is demonstrated through a simulation study. The solution quality and execution efficiency of the proposed tabu search has also been compared with a hybrid genetic algorithm.

Key Words: Real-Time Systems, Task Scheduling, Combinatorial Optimization, Multiprocessor Systems, Metaheuristics, Heuristics, Tabu Search

Introduction

Real-time systems are characterized by computational activities with timing constraints (*deadlines*). The correctness of real-time tasks depends as much on tasks completing by their deadlines as on the logical correctness of their results. Multiprocessors have emerged as a powerful computing means for real time applications, such as avionic control and nuclear control, because of their capability for high performance and reliability.

Even though real-time systems are expected to greatly benefits from multiprocessor technology, employing multiprocessor systems for real-time application has shown to be difficult. A major obstacle is that scheduling algorithms for multiprocessor real-time systems are significantly more complex than for uniprocessor systems (Leung and Whitehead, 1982). In multiprocessor systems, the scheduling algorithm must not only specify an ordering of tasks, but must also determine the specific processors to be used for executing a set of tasks.

The problem of scheduling real-time tasks has been studied by many researchers using different formulations. These formulations represent different types of real-time systems and different requirements of the systems. Except for the most trivial cases, the underlying problem of optimal task scheduling on multiprocessors systems has been known to be NP-hard in the strong sense (Garey and Johnson, 1979). Therefore, many researchers have adopted for heuristic approaches, which find a satisfactory suboptimal task scheduling in a reasonable amount of time.

Task scheduling in multiprocessor systems can be done either statistically or dynamically. In static algorithms, the assignment of tasks to processors and the time at which the tasks start execution are determined a priori. Static algorithms (Ramamritham, 1993) are often used to schedule periodic tasks with hard deadlines. The main advantage of static scheduling is that if a solution is found, then one can be sure that all deadlines will be guaranteed. However, this approach is not applicable to aperiodic tasks whose characteristics are not known a priori. Scheduling such tasks in a multiprocessor real-time system requires dynamic scheduling algorithms. In dynamic scheduling (Xu and Parns, 1990), when new tasks arrive, the scheduler determines the feasibility of scheduling these new tasks without jeopardizing the guarantees that have been provided for the previously scheduled tasks. Thus, for predictable execution, scheduling analysis must be done before a task's execution is begun. A feasible schedule is guaranteed if the timing and resource constraints of all the tasks can be satisfied, i.e. if the schedulability analysis is successful. The tasks are dispatched according to this feasible schedule (Manimaram and Ram Murthy, 1998).

Dynamic scheduling algorithms can be either distributed or centralized. In a distributed dynamic scheduling scheme, tasks arrive independently at each processor. When a task arrives at a processor, the local scheduler at the processor determines whether it can satisfy the deadline constraint of the incoming task. The task is accepted if it can be satisfied, otherwise, the local scheduler tries to find another processor which can accept the task. In a centralized scheme, all the tasks arrive at a central processor, called the scheduler, from where they are distributed to other processors in the systems for execution.

For multiprocessor systems with resource-constrained tasks, a heuristic search algorithm, called myopic scheduling algorithm, was proposed in (Ramamritham *et al.*, 1990). It was shown by the authors that an integrated heuristic which is a function of deadline and earliest start time of a task performs better than simple heuristics, such as EDF, least laxity first (i.e. a task with least value of deadline minus computation time executes first), and minimum processing time first. The myopic algorithm was later improved by Manimaran and Murthy (Ramamritham *et al.*, 1990) by introducing the concept of feasibility check window and parallelizing a task whenever deadline of a task cannot be met. It was shown through an intensive simulation study that the improved myopic algorithm outperforms the original algorithm. In (Manimaram and Ram Murthy, 1998), Manimaran and Murthy also tailored myopic algorithm for fault-tolerant dynamic scheduling for multiprocessor hard real-time tasks.

The genetic algorithm community has also proposed some genetic algorithms for task scheduling problem. Kim and Hong (Kim and Hong, 1993) proposed a genetic algorithm for static scheduling of tasks with the objective of minimizing total communication costs while achieving a load balance. Hou *et al.,* (1994) presented a genetic algorithm for static scheduling of non-identical tasks with known execution time and precedence relationships to identical processors. Their goal was to minimize the finishing time of the schedule. Kidwell and Cook (1994) used a genetic dynamic scheduling algorithm to assign non-identical tasks to identical processors. Mahmood proposed a hybrid generic algorithm in (Mahmood, 2000). The algorithm not only allocates real-time tasks to processors but also schedules them for each processor. Tabu search metaheuristic has also been applied to task allocation and scheduling problems. Porto and Ribeiro (1995) applied a tabu search metaheuristic to the solution of task scheduling problem on a heterogeneous multiprocessor environment under precedence constraints. Each task is assigned to a particular processor through an allocation function. A neighborhood solution is obtained by taking a single task from the task list of a processor and transferring it to a different processor. The whole neighborhood is obtained by going through every task and then building a new solution by placing the task into every position of the task list of every other processor in the system. A *recency-based* tabu mechanism was used to make certain moves tabu for a number of iterations. In (Hubscher and Glover, 1994), a tabu search algorithm was proposed to minimizing the makespan on *n* tasks on *m* equivalent processors. The authors used a candidate list strategy that generates only a small subset of moves. A dynamic tabu list is employed for handling tabu restrictions. They also used an influential diversification to improve the quality of the solution. Falco *et al.,* (1994) proposed a parallel tabu search algorithm and compared its performance with a parallel genetic algorithm. They showed through a simulation study that the parallel tabu search outperforms the parallel genetic algorithm. In (Mahmood, 2002), Mahmood tailored tabu search algorithm for scheduling real-time task in a distributed computing environment with the objective of minimizing task jitters.

In this paper, we present a tabu search algorithm to schedule periodic hard real-time tasks with resource and precedence constraints in multiprocessor systems. The proposed algorithm incorporates traditional scheduling heuristics to generate a feasible schedule for executing a set of hard real-time tasks on a multiprocessor system.

Model Description: We consider a set of *n* periodic hard real-time tasks $T = \{t_1, t_2, \cdots, t_n\}$ to be executed on a set of *m* (*m*>1) identical processors $P = \{P_1, P_2, \cdots, P_m\}$, which are connected through a shared medium. Each task t_i is characterized by a worst-case computation time c_i, a period p_i and a relative deadline d_i. The worst-case computation time of a task can be obtained by static code analysis and the average computation time under possible worst cases. The actual computation time of a task may be less than its worst case computation time due to dependable loops and conditional statements and due to architectural features of the systems such as cache hits and dynamic branch prediction (Manimaram and Ram Murthy, 1998). However, there might be cases in which the actual computation time of a task may be more than its worst-case computation time. There are techniques, such as task-pair scheme (Streich, 1995), to handle such situations.

The tasks may have precedence constraints i.e. if t_i and t_j are two tasks and $t_i > t_j$ (read as t_i precedes t_j) then t_i must be completed before t_j is started. Accordingly, with each task set, we can associate a graph *G*, whose nodes correspond to the set of tasks. There is a directed arc from the node representing task t_i to node representing task t_j if and only if $t_i > t_j$ and there is no task t_k such that $t_i > t_k > t_j$. We also assume that tasks are non-preemptive, i.e., when a task starts execution on a processor, it finishes to its completion.

We assume that relative arrival time of task t_i is equal to its period p_i. When a task arrives in the system, it may start execution any time if the required resources are available and all of its precedence tasks have already finished execution (we do not consider the cases where a certain amount of time should elapse between two consecutive executions of a task). If $st(t_i)$ and $ft(t_i)$ are relative scheduled start time and finish time of task t_i respectively, then t_i meets its deadline if $st(t_i) \le d_i - c_i$ and $ft(t_i) \le d_i$.

A task might need some resources such as data structures, variables and communication buffers for its execution. Every task may have two types of accesses to a resource: exclusive and shared. In the case of *exclusive access*, no other task can use the resource; a *shared access* allows a resource to be shared with other tasks (the other tasks should also be willing to share the resource). A resource conflict exits if two tasks access the same resource and one of the accesses is exclusive. The earliest time when a resource R_k is available for shared (or exclusive) usage, denoted by EAT_k^e, can be estimated from the resource requirements of already schedule tasks. The earliest start time of a task t_i, denoted by $EST(t_i)$, is defined as:

$$EST\ (t_i) = Max\ (Min_{\ j \in P}(avail\ \ time\ (j)),\ Max_{\ k \in R_i} EAT_{\ k}^{u},\ Max_{\ t_j < t_i} ft\ (t_j))$$

Where *avail time(j)* denotes the earliest time at which processor P_j becomes available for executing a task, and the second term denotes the maximum among the earliest available times of resources requested by task t_i (set of resources requested by task t_i is denoted by R_i), in which $u = s$ for shared mode and $u = e$

for exclusive mode. The last term denotes the latest finish time of a task that should be executed before t_i.

The objective of task scheduling is to determine an execution order of tasks in T and the processor on which each task should be executed such that the all the tasks meet their deadlines subject to the precedence and resource constraints.

Tabu Search: Tabu search is an iterative procedure for solving discrete combinatorial optimization problems. It was first suggested by Glover (Glover, 1990) and since then, it has been increasingly used. It has been successfully applied to obtain optimal and suboptimal solutions to such problems as scheduling, time tabling, travelling salesman and layout optimization.

Tabu search starts from some initial feasible solution and attempts to determine a better solution in a manner of a "greatest descent neighborhood" search algorithm. The basic idea of the method is to explore the search space of all feasible solutions by a sequence of moves. A move is an atomic change, which transforms the current solution into one of its neighboring solutions. Associated with each move is a move value, which represents the change in the objective function value as a result of the move. Move values generally provide a fundamental basis for evaluating the quality of a move. At every iteration of the tabu search algorithm, an admissible best move is applied to the current solution to obtain a new current best solution to be used in the next iteration, even if the move is a non-improving one, i.e. it does not lead to a solution better than the current best one. However, to escape from local optima and to prevent cycling, a subset of moves is classified as tabu (or forbidden) for certain number of iterations. The classification depends on the history of the search, particularly as manifested in the recency or frequency that certain moves of solution components, called attributes, have participated in generating past solutions. A simple implementation, for example, might classify a move as tabu if the reverse move has been made recently. These restrictions are based on the maintenance of a short-term memory function (the tabu list), which determines how long a tabu restriction will be enforced, or alternatively, which moves are admissible at each iteration. The tabu tenure (i.e. the duration for which a move will be kept tabu) is an important feature of tabu search, because it determines how restrictive the neighborhood search is. The tabu restrictions are not inviolable under all circumstances and a tabu move can be overridden under some conditions. A condition that allows such an override is called an aspiration criterion. For example, an aspiration criterion might allow overriding a tabu move if the move leads to a solution which is the best obtained so far and hence allowing backtracking to previous solutions. The tabu search method for minimizing the objective *c(.)* is summarized in Fig. 1.

Task Scheduling with Tabu Search: The basic tabu search algorithm is briefly discussed in the previous section. In this section, we specialize the basic tabu search into a specific algorithm for the task scheduling problem. This implies in turning the abstract concepts of tabu search, such as, initial solution, solution space, neighborhood, move generation, among others, into more concrete and implementable definitions.

The objective of a task scheduling algorithm is to determine an execution order for a set of tasks and the processor on which each task should execute such that each task meets its deadline.

Initial Solution: The initial solution s_0 is obtained using a variant of an integrated heuristic, which is a function of deadline and earliest start time of a task (Xu and Parns, 1990). At each iteration, an executable task $t_i \in T$ with the earliest deadline and start time is selected and scheduled on a least loaded processor taking into account the precedence constraints. We notice that the initial solution may not be feasible because some tasks may not meet their deadlines.

Neighborhood and Move Generation: A neighborhood solution $\bar{s} \in N(s)$ is obtained by taking a single task $t_i \in T$ from the task list of a processor $P_i \in P$ and transferring it to the task list of another processor $P_j \in P$. The whole neighborhood is obtained by going thorough every task and then building a new solution by placing this task into every position of the task list of every processor in the system. The cardinality of the neighborhood is $O(n^2)$. However, the objective of a good search algorithm should be to generate a small candidate list of moves that are likely to be superior. Below, we explain how the candidate moves are generated in the proposed algorithm.

Our candidate selection strategy evaluates only part of the neighborhood before making a move. This candidate selection strategy randomly orders the tasks, and from this random ordering, selects the first task and evaluates its neighborhood. It continues with the next task until a specified number of tasks have been processed. The ratio of the number of tasks to process to the total number of tasks to be scheduled is referred to as window size. As the window size is reduced, the search moves to new solutions faster than standard tabu search, especially when large scheduling problems are being solved. However, the quality of the move is unlikely to be as high. Therefore, there is a time/quality tradeoff, which is clearly dependent on the number of tasks.

A candidate move is characterized by a vector $(t_i, P_j, P_k, s_1, s_2)$. This indicates the transfer of task t_i from position s_1 of the task list of processor P_j to position s_2 of the task list of processor P_k. We allow a move that transfers task t_i at a different position in the task list of the same processor, in which case $P_j = P_k$. The number of admissible moves is further reduced by placing task t_i after the last predecessor task and before the first successor of t_i in the task list of P_k because other positions in the task list of processor P_k are unlikely to be appropriate, since the task list could not be processed in its natural order. That is, if $t_a < t_i < t_b$ (t_a and t_b are in the task list of processor P_k), then all the candidate moves are obtained by placing t_i at all the positions between t_a and t_b.

```
Initialize the short-term memory function
Generate the starting solution s0
s, s* = s0    {s is curent solution and s* is best solution found so far}
While (moves without improvement < maxmoves) do
   bestmovevalue = ∞
   for (all candidate moves) do
     if ( candidate move is admissible i.e. it does not belong to a tabu list) then
        obtain the neighbor solution s̄ by applying a candidate move to the current solution s
        movevalue = c(s̄) − c(s)  {c(s) is the objective function to me optimized}
        if (movevalue<bestmovevalue) then
           bestmovevalue=movevalue
           s' = s̄
        endif
     endif
   endfor
   update the short term memory function
   if (c(s') < c(s*)) then s* = s' endif
   s = s'
endwhile
```

Fig.1: A General Framework of Tabu Search

The best candidate move is one that results in the maximum number of tasks that meet their deadlines. In case of a tie, a second move evaluation criterion is used which defines a move as a best move if it gives the least start time for task t_i. If a tie still exits, it can be broken arbitrarily.

Tabu List: A chief mechanism for exploiting memory in tabu search is to classify a subset of moves in the neighborhood as forbidden (tabu). This classification depends on the history of search, particularly manifested in the recency or frequency that certain move or solution components have participated in generating past solutions.

A candidate move is tabu if it fulfills one of the following conditions. First, the move $(t_i, P_j, P_j, s_1, s_1)$ is automatically excluded from the consideration. Such a move is called a null move in tabu search terminology and is forbidden by default. Second, a move $(t_i, P_j, P_k, s_1, s_2)$ is tabu if its reverse move $(t_i, P_k, P_j, s_2, s_1)$ has been executed recently. Unlike, standard tabu search in which the value of tabu tenure is fixed, we determine the tabu tenure of a move through a feedback (reactive) mechanism during the search. The tabu tenure of a move is equal to one at the beginning (the inverse move is prohibited only at the next iteration), it increases only when there is evidence that diversification is needed, and it decreases when this evidence disappears. In detail, the evidence that diversification is needed is signaled by the repetition of previously visited configurations. All configurations found during the last I iterations of the search are stored in memory (use of a queue is a possible implementation strategy). After a move is executed, the algorithm checks whether the current configuration has already been found and it reacts accordingly (tabu tenure of the move increases if a configuration is repeated, it decreases if no repetition occurred during a sufficiently long period).

Extended Tabu Lists: In some situations, it may be desirable to increase the number of available moves that receive a tabu classification. This may be achieved either by increasing the tabu tenure or by changing the tabu restrictions. Several other applications of tabu search (Hansan *et al.*, 1992, Laguna *et al.*, 1990) have shown that frequently it may be interesting to turn tabu-active certain attributes that not only avoid the reversal move towards the original solution, but also avoid a great variety of other moves towards other solutions which resemble the original one. Using this approach, we define another restrictive tabu attribute to prohibit task t_i from leaving processor P_k for a fixed number of iterations after the move $(t_i, P_j, P_k, s_1, s_2)$.

Aspiration Criterion: Tabu conditions based on the activation of some moves attributes may be too restrictive and result in forbidding a whole set of unvisited moves which might be attractive. The aspiration criterion allows overriding a tabu move if the move leads to a solution which is the best obtained so far. This provides a kind of backtracking to previous solutions. The complete algorithm is given in Fig.2.

Computational Results: We present, in this section, the computation results obtained with the application of the proposed task scheduling algorithm to a variety of task graphs. We evaluated the performance of the algorithm with the variation of many of its parameters. The performance of the algorithm is measured by its success ratio, which is defined as the ratio of the number of task sets found schedulable by the algorithm to the number of task sets considered for scheduling.

Generate the starting solution s_0 as describes in section 4

Evaluate $c(s_0)$ {$c(s)$ is the function that determines the number of task that meat their deadlines in schedule s}

$s, s^* = s_0$ {s is current solution and s^* is best solution found so far}

Let *window_size* be the number of tasks whose neighborhood is to be evaluated
Let *nmoves* be the maximum allowed number of moves without improvement in the solution
Let *tabu* be a matrix to keep track of tabu status of every move
Let *extened_tabu* is a matrix to keep track of extended tabu restriction
Let *history* is a matrix that keeps track of moves made in last *I=a_constant* iterations and the last tabu tenure of a move

Initialize *tabu* list by setting $tabu(t_i, p_j, p_k, s_1, s_2) = 0$ for all

$t_i \in T$, $P_i, P_j \in P$ and $s_1, s_2 = 1..n$

Initialize *history* by setting $history(t_i, P_j, P_k, s_1, s_2).moves = 0$ and

$history(t_i, P_j, P_k, s_1, s_2).last\_tabu\_tenure = 0$ for all $t_i \in T$, $P_i, P_j \in P$ and $s_1, s_2 = 1..n$

for all $t_i \in T$ do **for** all $P_i \in P$ **do** $extended\_tabu(t_i, P_i) = 0$ **endfor**

itr=0
while (*itr* < *nmoves*) **do**
  $bestmovevalue = -\infty$
  itr=itr+1;
  Randomly order the tasks
  for *i*=1 to *window_size* **do**
  for (all candidate moves for *i*th task in the ordered list) **do**
    Obtain the neighbor solution $\bar{s}$ by applying a candidate move to the current solution *s*
    $movevalue = c(\bar{s})$
    if (*movevalue>bestmovevalue* AND candidate move is not tabu
      OR $movevalue > c(s^*)$) **then**
        bestmovevalue=movevalue
        $s' = \bar{s}$
    endif
  endfor endfor
  $s = s'$
  if $(c(s') > c(s))$ then *itr*=0 **endif**
  if $(c(s') < c(s^*))$ **then** $s^* = s'$ **endif**
  {Updates the tabu list, say the best move was $(t_i, P_j, P_k, s_1, s_2)$ }
  Decrease the tabu tenure by one of all tabu moves in *tabu* and *extended -tabu* lists
  if $history(t_i, P_j, P_k, s_1, s_2).moves > 0$ **then**
    $tabu(t_i, P_j, P_k, s_1, s_2) = history(t_i, P_j, P_k, s_1, s_2).last\_tabu\_tenure + 1$
  else
    $tabu(t_i, P_j, P_k, s_1, s_2) = \max(history(t_i, P_j, P_k, s_1, s_2).last\_tabu\_tenure - 1, 1)$
  endif
  $extended\_tabu(t_i, P_i) = const\_value$
  update *history*
endwhile

Fig. 2: Tabu Search Algorithm for Task Scheduling

We have taken 20 sets of task graphs of different sizes ranging from 50 to 400 tasks. The number of processors was varied from 10 to 200 depending on the number of tasks in the task graph. In no case, the number of processors was more than half of the number of tasks in the graph. The worst case computation time, c_i, of a task t_i was chosen randomly between MIN_C and MAX_C, where MIN_C and MAX_C were set to 30 and 60 respectively.

The deadline of a task t_i with no precedence constraint is uniformly chosen between $2*c_i$ and $R*c_i$, where $R \geq 2$. The dead line of a task with precedence constraint was randomly chosen from the range $\max(\max_{t_j>t_i}(d_j)*1.3, \max_{t_j>t_i}(d_j)*R)$, where $R \geq 1.5$. The resource requirements of a task are generated based on the input parameters UseP and ShareP.

Several tests were made in order to obtain the best tabu configuration patterns (i.e. tabu parameters and implementation strategies), which would provide the best performance for the proposed scheduling algorithm. We got the best results (on average) with different values of *nmoves* depending on the problem size, and window size *n*/20, where *n* is the number of tasks in the task graph. The value of *I* was set to 15. The candidate set, tabu tenure and aspiration criterion strategies are those described in the previous section.

Our simulation results show an improvement of 10% to 75% in the success ratio in the initial solution. The average percentage success ratio for all the simulation runs with the best setting is 90.4%. The effect of different parameters on the performance of the proposed algorithm was also studied and is summarized here.

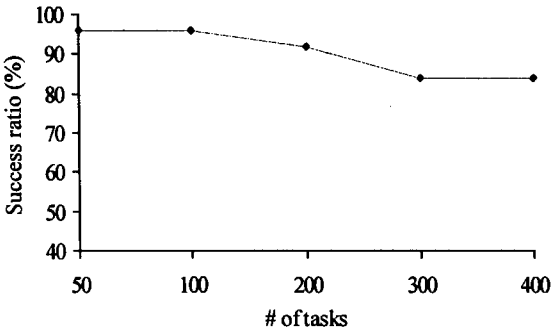

Fig.3: Effect of Graph Size on Success Ratio

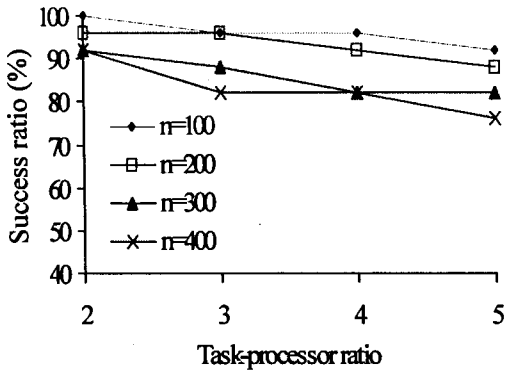

Fig. 4: Effect of Task-Processor Ratio on Success Ratio

a. Number of Task: In the first set of experiments, we studied the effect of number of tasks in a task graph on the performance of the proposed algorithm. Fig. 3 shows the percentage success ratio for different graph sizes. Each point in the performance curve is the average of twenty-five simulation runs with the best parameter settings. We observed that the tabu search algorithm is quite robust. The quality of solutions obtained for different graph sizes does not seem to be much affected for different graph sizes. The main reason for this behavior seems to be the efficiency of the candidate selection strategy, which for every configuration, discards most of the admissible moves and keeps only those leading to good solutions.

b. Effect of Task-Processor Ratio: In the second set of experiments, we studied the effect of task-processor ratio on the performance of the algorithm. Fig. 4 shows the results obtained with the best parameter settings. The task-processor ratio ($\lceil n/m \rceil$) is along the x-axis and the percentage success ratio along the y-axis. It is evident from the figure that the algorithm performs quite well with different values of task-processor ratio. However, it decreases slowly as the task-processor ratio increases, i.e. when a relatively large number of tasks are scheduled on a small number of processor.

c. Effect of Window Size: In the third set of experiments, we investigated the effect of window size (i.e. the number of tasks whose neighborhood is explored at each iteration) on the performance of the algorithm. The result (average of 25 runs) is shown in Fig. 5. As can be seen that the best results are obtained when window size was set to *n*/20. Increasing the window size does not improve the success ratio significantly. Rather, it slows down the convergence to the (sub)optimal result. That is, if the maximum number of iterations is kept fixed, then success ratio decreases as the windows size decreases.

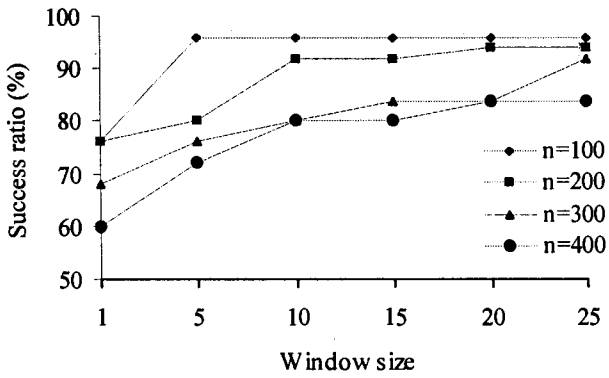

Fig. 5: Effect of Window Size on Success Ratio for Different Values of *n*

d. Effect of Allowed Non-Improving Moves: Another important parameter of tabu search is the number of non-improving moves that are allowed before terminating the algorithm. If this parameter is kept too low, we may end up with fast but poor quality

solutions. If it is too large, we may get better quality solution at the expense of execution time of the search algorithm. In addition, allowing much larger non-improving moves may not lead to a better solution. To verify this claim, we investigated the impact of maximum non-improving moves, which are made before terminating the algorithm. The results are shown in Fig. 6, which support our claim.

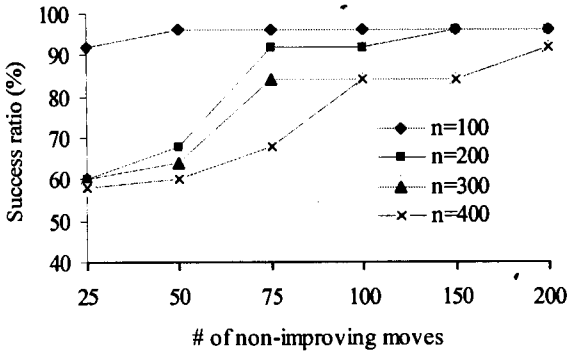

Fig. 6: Effect of Allowed Non-Improving Moves on the Success Ratio

e. Comparison of Proposed Algorithm with a Hybrid Genetic Algorithm: Genetic Algorithms (GAs) are general, domain independent search and optimization techniques. Mahmood (2000) proposed a hybrid genetic algorithm for scheduling of real-time task in multiprocessor systems. We conducted a series of simulation studies to compare the relative performance of the proposed tabu search algorithm and the hybrid genetic algorithm proposed by Mahmood (2000).

In the first set of experiments, we studied the effect of number of tasks in a task graph on the performance of the tabu search and GA. Fig. 7 shows the percentage success ratio for different graph sizes for both algorithms. Each point in the performance curve is the average of twenty-five simulation runs with the best parameter settings for both algorithms. Each algorithm was allowed to run for same amount of time (from 100 seconds to 2500 seconds depending on the problem size) in order to make a fair comparison of the solution quality. We observed that tabu search algorithm produced solution of better or of the same quality in most of the cases as compared to that of GA.

However, on average, tabu search took less time to converge to the best solution as compared to the time taken by GA. In fact, the experiments show that the time required by the tabu search to reach to the final solution found by the GA is, on overage, about 65% of the total execution time. So that if we are not interested in the further improvement of the solution quality offered by the GA, we can obtain the same final solution by tabu search in about 35% less time. The main reason for this behavior seems to be the efficiency of the candidate selection strategy, which for every configuration, discards most of the admissible moves and keeps only those leading to good solutions. Therefore, when both tabu search and GA were run for the same amount of time, the tabu search reaches to the best solution in less time (on average) and the remaining time is used to improve the solution, if possible. This explains why tabu search was able produce better solutions when both tabu search and GA were run for the same amount of time.

Another very important point for the evaluation of the reliability of a heuristic technique is the robustness, i.e. its ability to provide quite similar results for different executions of the same task. With respect to this point, Table 1 shows the standards deviation for both techniques for different task graph sizes. From the first two columns, we may note that the standard deviation (on average) for the tabu search is always lower than that for the GA. The last column of the table contains the values of the ratio between the two standard deviations (i.e. Stan_dev of GA/Stan_dev of tabu search). As can be seen all the ratios are between 0.43 and 0.63 indicating that the tabu search technique is more robust that the GA.

Table 1: The Standard Deviations for the Tabu Search and GA

# of tasks	Std-dev for tabu search (*SDTS*)	Std-dev for GA (*SDGA*)	*SDTS/SDGA*
50	3.20	1.50	0.47
100	5.30	8.74	0.61
200	5.92	9.43	0.63
300	7.32	17.45	0.42
400	9.27	19.36	0.48

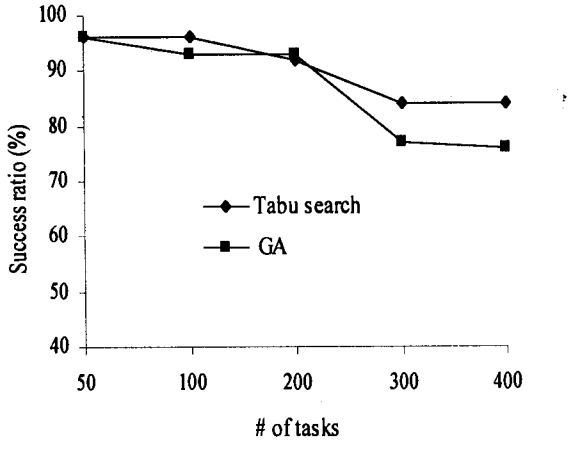

Fig. 7: Success Ratio of Tabu Search and GA for Different Graph Sizes

Conclusion

Meeting deadlines and achieving high resource utilization are two main goals of task scheduling in real-time systems. Both preemptive and

nonpreemptive algorithms are available in the literature to satisfy such requirements. In this paper, we presented a new tabu search algorithm to schedule nonpreemptive hard real-time tasks with resource and precedence constraints. We have presented various strategies to obtain the initial solution, neighborhood solutions, and the best moves. We have demonstrated through a simulation study that the solutions obtained by the proposed algorithm are of high quality. We have shown that the robustness of tabu search by applying it to problems of various sizes. Also, a comparison shows the superiority, in terms of solution quality, execution efficiency, and robustness, of the proposed algorithm over a hybrid genetic algorithm.

References

Falco, I. D., *et al.*, 1994. Parallel Tabu Search Verses Parallel Evolution Strategies. Proc. 1st Int. Conf. on Massively Parallel Computing Sys. 564-569.

Garey, M. R. and D. S. Johnson, 1979. Computer and Interactivity: A guide to the Theory of NP-Completeness. New York: W. H. Freeman and Co.

Glover, F., 1990. Tabu Search: a Tutorial. Interfaces. 20: 74-94.

Hansen, P., *et al.*, 1992. Location and Sizing of Off-Shore Platforms for Oil Exploration. European J. of Operations Research. 58: 202-214.

Hubscher, R. and F. Glover, 1994. Applying Tabu Search with Influential Diversification to Multiprocessor Scheduling. Computers and Operations Research. 21: 877-884.

Hou, E. S., *et al.*, 1994. Genetic Algorithm for Multiprocessing Scheduling. IEEE Trans. Parallel and Distributed Sys. 5: 113-120.

Kidwell, M.D. and D. J. Cook, 1994. Genetic Algorithm for Dynamic Task Scheduling. IEEE 13th Annual International Phoenix Conference on Computers and Communications. 61-67.

Kim, Y. C. and Y. S. Hong, 1993. Task Allocation Using a Genetic Algorithm in Multicomputer Sys.", Proc. 10th IEEE Region Conference on Computer Communication, Control and Power Eng. 258-261.

Leung, J. Y. T. and J. Whitehead, 1982. On the Complexity of Fixed Priority Scheduling of Periodic Real-Time Tasks. Performance Evaluation. 2: 237-250.

Laguna, M., *et al.*, 1990. Scheduling Jobs with Linear Delay Penalties and Sequence Dependent Setup Costs and Times Using Tabu Search. App. Intelligence. 34-46.

Mahmood, A., 2000. A Hybrid Genetic Algorithm for Task Scheduling in Multiprocessor Real-Time Systems. Studies in Informatics and Control. 9: 207-218.

Manimaram, G. and C. S. Ram Murthy, 1998. An Efficient Dynamic Scheduling Algorithm for Multiprocessor Real-Time Sys. IEEE Trans. Parallel and Distributed Sys. 9: 312-319.

Mahmood, A., 2002. Scheduling Real-Time Tasks in Distributed Environment with Tabu Search Algorithm. Pak. J. of Infor. and Tech. 1: 153-159.

Porto, S. C. S. and C. C. Ribeiro, 1995. A Tabu Search Approach to Task Scheduling on Heterogeneous Processors Under Precedence Constraints. Int. J. of High Speed Computing. 17: 21-40.

Ramamritham, K., 1993. Allocation and Scheduling of Precedence Related Periodic Tasks. IEEE Trans. Parallel and Distributed Sys. 4: 382-397.

Ramamritham, K., *et al.*, 1990. Efficient Scheduling Algorithms for Real-Time Multiprocessor Sys. IEEE Trans. Parallel and Distributed Sys. 1: 184-194.

Streich, H., 1995. Taskpair-Scheduling an Approach for Dynamic Real-Time Systems. Int. J. Mini and Microcomputers. 7: 77-83.

Xu, J. and L. Parns, 1990. Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations. IEEE Trans. Software Eng. 16: 360-369.