Asian Journal of Information Technology 5(6): 640-646, 2006

A Heuristic Approach for Rescheduling of Meetings with A-Algorithm

[1]M. Sugumaran, [2]K.S. Easwarakumar and [2]P. Narayanasamy
[1]Department of Computer Science and Engineering and Information Technology,
Pondicherry Engineering College, Pondicherry-605 014, India
[2]Department of Computer Science and Engineering,
Anna University, Chennai-600 025, India

Abstract: Meeting scheduling is generally considered as one of the most common activities that takes place in organizations. The basic problem in meeting scheduling is to find a common free time for all participants of a particular meeting. It is an important activity in many organizations as most of the office workers spend much of their times in meeting scheduling. It is a time-consuming, iterative and tedious task. There exists several solutions in the literature for centralized calendar management and meeting scheduling, but they are of limited success. In this paper, we propose a new approach based on A-algorithm using heuristic function for meeting scheduling, which is performs effectively even in the case of larger problem instances than the existing algorithm.

Key words: Meeting scheduling, office automation, A-algorithm, artificial intelligence

INTRODUCTION

The research conducted at AT & T[1], reveals that nearly 95 percent of the time an executive devotes in interpersonal communications: face-to-face meeting, document processing and phoning. Office workers, other than executives, also spend over four fifth of their time on the above activities. It is realized that meetings are the most time-consuming activity particularly for the executives and managers. To work efficiently, office workers need to manage the time and the entire schedule of meetings instead of considering individual meetings. Because of the inherent tedious, interactive, iterative and time-consuming features, meeting schedule is to be considered for automation. The benefits of automated meeting schedulers are not only save time, effort on the part of human, but also gives more efficient schedules. Information management systems, such as data communications, electronic document communication systems (e-mail), teleconferencing and sensor-based systems like those used for energy conservation continue to grow. To process the voluminous data, automation is essential. As a factor of truth, the lay off software engineers is due to the office automation. So, automation is essential for the problems like meeting scheduling.

The scheduler given in Sugihara *et al.*[2] generates all solutions and then chooses the optimal one. Since the timetable rearrangement problem in Sugihara *et al.*[2] is NP-hard[3] and has been shown that the feasibility problem is NP-complete[3]. So, we use the concept of A-algorithm[4-6] to choose the best probable node among the available nodes from the search tree for the expansion and thus pruning the rest to get the solution quicker.

PROBLEM SPECIFICATION

Let n be a positive integer. A timetable of n meetings is an 8-tuple $T(n) = (P, M_n, <, t, p, w, \tau, \rho)$, where $P = \{1, 2, \ldots, m\}$ is a set of m persons; $M_n = \{m_1, m_2, \ldots, m_n\}$ is a set of n meetings; $<$ is a partial order on M_n; $t(m_i)$ is the time duration of meeting m_i; $p(m_i)$ is a set of groups of persons such that exactly one person in each group is required to attend the meeting m_i; $w(m_i)$ is a set of time instances at which meeting m_i can start; $\tau(m_i)$ is the starting time at which meeting m_i is scheduled; and $\rho(m_i)$ is the set of attendants of meeting m_i chosen from $p(m_i)$. Further, the schedule also satisfies the following four conditions:
Let m_i and m_j be any two meetings.

- C1: No person attends more than one meeting simultaneously.
- C2: If $m_i<m_j$, then m_j starts after m_i ends, in case some persons attend both meetings m_i and m_j.
- C3: For each group $g \in p(m_i)$, exactly one person can attend the meeting m_i
- C4: m_i can start at one of the time instances in $w(m_i)$.
- C5: The time interval of a meeting is contiguous and cannot be split across days.

Corresponding Author: M. Sugumaran, Department of Computer Science and Engineering and Information Technology, Pondicherry Engineering College, Pondicherry-605 014, India

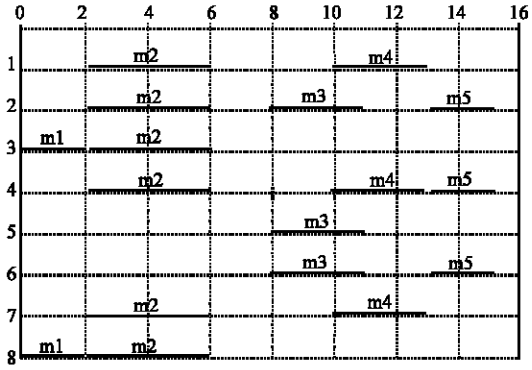

Fig. 1: Graphical representation of T(5)

C6: The venue of the meeting m_i is one of the agents' site or near to them.

In a schedule T(n), the parameters<, t, p and w represent the input requirements of meetings, whereas τ and ρ represent a schedule of meetings that satisfies all the requirements. The assignment of rooms to meetings can easily be incorporated into the schedule by considering rooms as pseudo-persons. For example, selection of a room for m_i from rooms $R_1, R_2, \ldots, R_k$ is represented by considering a group of corresponding pseudo-persons in $p(m_i)$[2].

Example 1: Let us consider the example given in Sugihara *et al.*[2] for timetable T(5) of 5 meetings, shown in Fig. 1.

P= {1, 2, 3, 4, 5, 6, 7, 8}; $M_5 = \{m_1, m_2, m_3, m_4, m_5\}$; $m_1 < m_5$ and $m_2 < m_4$.

$t(m_1)$= 2, $t(m_2) = 4$, $t(m_3) = 3$, $t(m_4) = 3$ and $t(m_5) = 2$.

$p(m_1)$= {{2, 3}, {7, 8}}, $p(m_2)$ = {{1}, {2}, {3}, {4}, {7}, {8}}, $p(m_3)$ = {{2}, {5}, {6, 7, 8}},

$p(m_4)$= {{1, 2}, {3, 4}, {6, 7, 8}} and $p(m_5)$ = {{1, 2}, {3, 4}, {6, 7}}.

$w(m_1)$= {0, 1, 2, 3, 4}, $w(m_2)$ = {2, 3}, $w(m_3)$ = {2, 3, 8, 9}, $w(m_4)$ = {k|2≤k≤12} and $w(m_5)$ = {k | 0≤k≤16}.

$\tau(m_1)$= 0, $\tau(m_2) = 2$, $\tau(m_3) = 8$, $\tau(m_4) = 10$ and $\tau(m_5) = 13$.

$\rho(m_1)$= {3, 8}, $\rho(m_2)$ = {1, 2, 3, 4, 7, 8}, $\rho(m_3)$ = {2, 5, 6}, $\rho(m_4)$ = {1, 4, 7} and

$\rho(m_5)$= {2, 4, 6}.

The timetable rearrangement problem is done iteratively. That is, from T(i), we can generate T(i+1) by giving appropriate input parameters. For instance, the input parameters required for the new timetable T(n+1) by rearrangement is defined as

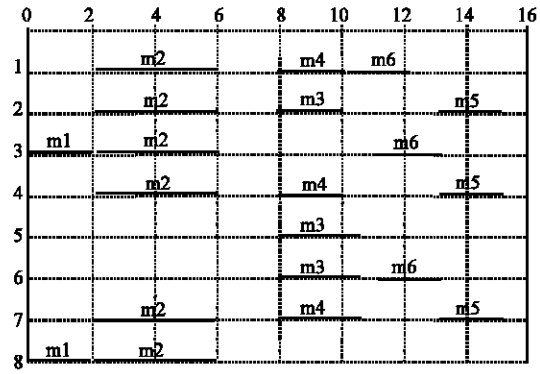

Fig. 2: Graphical representation of T(6)

I1: a timetable $T(n) = (P, M_n, <, t, p, w, \tau, \rho)$,
I2: a partial order < on $M_{n+1} = M_n \cup \{m_{n+1}\}$,
I3: $t(m_n+1)$, $p(m_n+1)$ and $w(m_{n+1})$ and
I4: a set F ($\subseteq M_n$) of meetings whose start times are fixed.

Example 2: Consider an instance to schedule a new meeting m_6 on Example1 as per the following input parameters.

I1: T(5) as in Example 1 (Fig. 1).
I2: $m_1 < m_5$, $m_2 < m_4$, and $m_3 < m_6$.
I3: $t(m_6) = 3$, $p(m_6)$ = {{1}, {3, 4}, {6}} and $w(m_6)$ = {8, 9, 10, 11, 12}.
I4: F = $\{m_5\}$.

Now, Fig. 2 shows an optimum timetable T(6) = $(P, M_6, <, t, p, w, \tau', \rho')$, where $\tau'(m_4) = 8$, $\rho'(m_5)$ = {2, 4, 7}, $\tau'(m_6) = 11$ and $\rho'(m_6)$ = {1, 3, 6}.

Note that in T(6), the start time of the meeting m_4 is changed from 10 to 8 and an attendant of meeting m_5 is changed from 6 to 7.

THE NEW MEETING SCHEDULER

In this model, six operations such as CP (Change of Person), XP (eXchange of Person), SL (Shift Left), SR (Shift Right), CSL (Continuous Shift Left and CSR (Continuous Shift Right) are used to rearrange the meeting scheduling. Based on these operations heuristic value is determined. This heuristic value will be the deciding factor for selecting next node in the search tree expansion. We apply here the method of A*-algorithm with this heuristic value for the generation of search tree. This process will speed up the searching in an optimal way.

Generally, in A-algorithm, the function f is defined for its value f(n) at any node n is the actual cost of an optimal

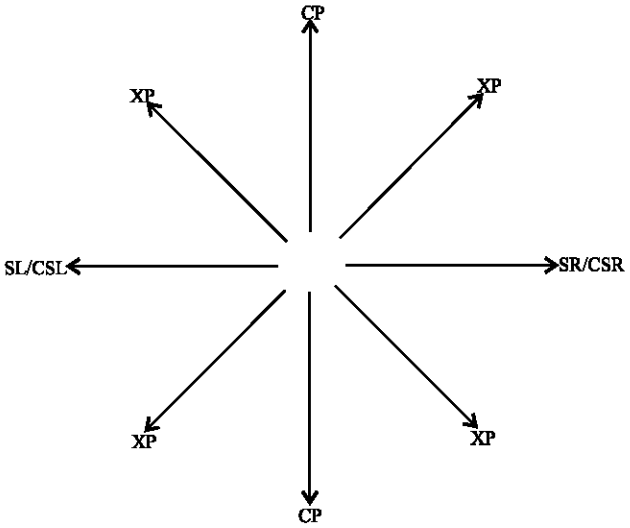

Fig. 3: Operations applied in the search tree

path from the root node to the node n plus the cost of an optimal path from node n to a goal node, that is, f(n) = g(n) + h(n). The estimate of f can be given by

$$f(n) = g(n) + h(n),$$

where g is an estimate of g and h is an estimate of h. An obvious choice for g(n) is the cost of the path in the search tree from the root to n given by summing the arc costs encountered while tracing the pointers from n to the root, whereas the estimate h(n) of h(n) rely on heuristic information from the problem domain. The function h(n) is called as heuristic function[4,5,7].

In this approach, the heuristic function h(n) is applied to the nodes of the search tree that counts how many slots are not free and that will be determined by the six operations stated earlier such a way that in a particular duration (x, x + t(m_{n+1})), where x ∈ w(m_{n+1}), among the persons in p(m_{n+1}) for the current meeting m_{n+1}. The function h(n) is used primarily to choose the best probable node among the available live nodes and thus guide the search process in the optimal way.

Let m_{n+1} be the current meeting to be scheduled and p(m_{n+1}) = {g_1, g_2, …, g_r} be the set of groups of persons to be considered for scheduling of m_{n+1}. Let ρ(m_{n+1}) be the set of attendees for m_{n+1}, a set of candidates to be chosen from p(m_{n+1}) one person per group. Further, let (x, x + t(m_{n+1})) be the current interval to be considered for the meeting m_{n+1}. These assumptions are used in the following six operations, shown in Fig. 3.

SL: Shifts meetings horizontally to the left. For each person j ∈ ρ(m_{n+1}) and each meeting m_i ∈ (x, x + t(m_{n+1})), if there is a time interval (s, s + t(m_i)) to the left and disjoint with (x, x + t(m_{n+1})) such that s ∈ w(m_i) and no person in ρ(m_i) attends any meeting during (s, s + t(m_i)), then change the start time of m_i to s.

SR: Shifts meetings horizontally to the right. For each person j ∈ ρ(m_{n+1}) and each meeting m_i ∈ (x, x + t(m_{n+1})), if there is a time interval (s, s + t(m_i)) to the right and disjoint with (x, x + t(m_{n+1})) such that s ∈ w(m_i) and no person in ρ(m_i) attends any meeting during (s, s + t(m_i)), then change the start time of m_i to s.

CP: Changes person vertically within a group. This operation makes free slots for the persons ρ(m_{n+1}) in (x, x + t(m_{n+1})) within groups. For each meeting m_i ∈ (x, x + t(m_{n+1})), if no person in ρ(m_i) except some person j∈ ρ(m_i) attends m_i and if there is a person k in {a | a, j∈ g_r and g_r ∈ p(m_i), k ≠ j} who does not attend any meeting during (τ(m_i), τ(m_i) + t(m_i)), then change the attendant j of m_i to k.

XP: Exchanges persons within a group among different meetings. This makes free slots for the current meeting by making changes diagonally. For any two persons j, k ∈ g_p in p(m_i) and j, k ∈ g_q in p(m_n) such that j ∈ ρ(m_i) and k∈ρ(m_n), then swap j and k between ρ(m_i) and ρ(m_n) if j is free in (x, x + t(m_n)) and k is free in (x, x + t(m_i)).

CSL: Shifts the meetings in (x, x + t(m_{n+1})) as well as the meetings to the left of this interval further to the left in order so that each person j ∈ ρ(m_{n+1}) is free in (x, x + t(m_{n+1})).

CSR: Shifts the meetings in (x, x + t(m_{n+1})) as well as the meetings to the right of this interval further to the right in order so that each person j ∈ ρ(m_{n+1}) is free in (x, x + t(m_{n+1})).

A-algorithm: The A-algorithm for the meeting scheduler uses one heap and one set, called open-heap and node-set, respectively. The heap is used to keep the generated nodes with their heuristic values and node-ids. The heap always has the most probable node or minimum heuristic value node at the root so that this node always gets generated before all other nodes. Whenever a node is generated, it is inserted into the heap with its heuristic value and node-id. The heap is always maintained such that it satisfies the heap properties. Once a solution is found for the current meeting, the open-heap and node-set are initialized for scheduling the next meeting.

The node-set is used to keep the nodes of the search tree as and when get generated. It is very useful to reduce the time complexity of the algorithm by having

constant time of checking, subtraction and inclusion. The node-set is verified whenever a node is generated. Here we do not require to check separately whether a particular node is already expanded or not, since the heap having the nodes which are not expanded and each node configuration is unique.

The algorithm starts with the heap having the schedule T(n) as the root node. The iteration of the algorithm begins by removing the root node from the heap and designated as the current node. If the current node is not the solution node, the operations CP, XP, SL, SR, CSL and CSR are applied to this current node to generate a set of new nodes. These new nodes are verified for duplication with the node-set. If the nodes are not already available in the node-set, heuristic function is applied and then inserted into the heap, attached as successors to the current node and then inserted into the node-set. The next iteration of expansion starts by removing the root from the heap and treated as the current node. If the heap is empty, it declares no solution for the request and the process is terminated. If the current node is a solution node, the current meeting is scheduled permanently and this configuration is treated as the root node for the next meeting request, otherwise the process continues.

Algorithm 1

Schedule($w(m_{n+1})$, $p(m_{n+1})$)
// $w(m_{n+1})$ – a set of starting time instances for m_{n+1}
// $p(m_{n+1})$ – a set of group of persons considered for
// m_{n+1}

Initialize the open-heap and the node-set
Create a search tree, G, starting with the root node s of the schedule T(n)
Insert s into the open-heap and node-set
while(answer is not found)
begin
if(open-heap is empty)
print the error message and exit
n←delete the root of the open-heap
if(n is a goal node)
schedule the meeting m_{n+1} and exit
generate a set M of successors which are not in the node-set by applying the operations:
CP, XP, SL, SR, CSL and CSR on n; insert into the node-set
for each node in M, apply the heuristic function and establish a pointer to n
insert the nodes of M into the open-heap according to their heuristic values
end
end Schedule

Example: Let us consider Figure 4, it shows the search tree for the generation of schedule T(6) from T(5). The full generation of the tree is not required, but it is shown here that how the A-algorithm performs better than computing all solutions and select the optimal one. The nodes are given unique numbers to identify the order in which the nodes are getting generated and these numbers are given at the left side of the nodes. The heuristic values of the nodes are given at the right side. The nodes with heuristic value greater than or equal to zero are used to form the actual search tree.

The A-algorithm starts with the root node 1, the configuration T(5). The operations CP, XP, SL, SR, CSL and CSR are applied when a node is expanded. Consider the Example 2, with four input parameters (I1-I4). From I2, m_6 can be scheduled only after m_3. For m_3, the starting instance is $w(m_3) = \{2, 3, 8, 9\}$. Since m_3 is scheduled at the time instance 8, the possible starting instances for m_6 are $\{8, 9, 10, 11, 12\} - \{8, 9, 10\} = \{11, 12\}$. That is, the time instances 11 and 12 alone can be considered for the meeting m_6.

When heuristic values are applied to the time instances 11 and 12, both are having heuristic value 2. So, the time instance 11 is considered first for scheduling of meeting m_6 in (11, 11+3). In (11, 14), the meetings already scheduled are $\{m_4, m_5\}$, but we are not concerned merely with the meetings in (11, 14) alone. That is, we are applying the operations to those persons who have been scheduled in (11, 14) and also their meetings so as to get $\tilde{n}(m_6)$. To schedule m_6, the persons in $p(m_6) = \{\{1\}, \{3, 4\}, \{6\}\}$, we have to find one person who is free from each group of $p(m_6)$. The person 1, is not free in (11, 14) and there is no substitute for him. So, the meeting m_4 is to be moved out of (11, 14). m_4 can not be moved to the right as it has 12 the only starting time available and fall in (11, 14). So, m_4 has to be moved to the left of (11, 14). Since, we are trying m_6 to schedule in (11, 14), m_4 can be moved to (8, 11). The attendees of m_4, $\rho(m_4) = \{1, 4, 7\}$, are free. So m_4 is moved to (8, 11). Now, the person 6 is required to be free in (11, 14). But the person 6 is scheduled for m_5. m_5 can not be moved as it is fixed (I4). The only possibility is to have a substitute for the person 6 in m_5. So, for m_5, the person 6 is changed to 7. Now m_6 is scheduled with $\rho(m_6) = \{1, 3, 6\}$. The process of arriving schedule T(6) is shown in the Fig. 4.

When the node 1 is expanded, four nodes, 2, 3, 4 and 5, are generated as successors. Nodes 2 and 4 have the same value, but node 4 has been selected as the current node. Exploration of this node gives the solution node (node 6) marked with ✡. The rest of the exploration of nodes is not required as this heuristic leads to optimal

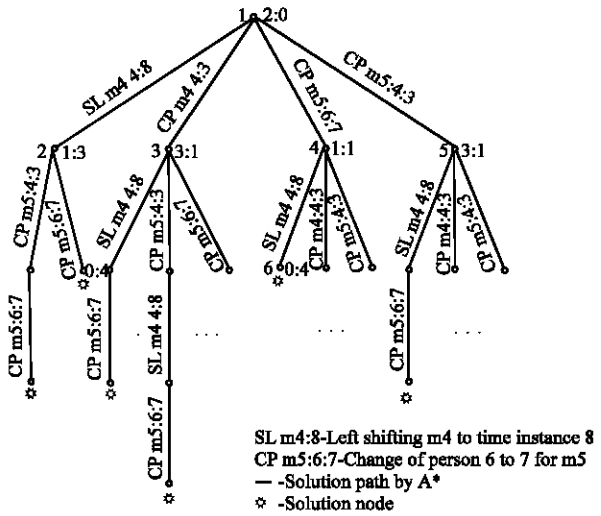

Fig. 4: Search tree

solution. The correctness of the algorithm is obtained in the following theorems.

Theorem 1: If the operations CP, XP, SL, SR, CSL and CSR are applied to the Algorithm1, it correctly finds an optimal solution if any exists.

Proof: Let us consider Fig. 3. The operations CP, XP, SL, SR, CSL and CSR are used to enumerate all possible states for the current meeting, m_{n+1}. As the algorithm starts with the root node and the operations are applied in each level of the search tree, it generates a finite number of nodes on each level. The heap always has the minimum value or the most probable node at the root for expansion and removed each time of expansion.

Case (i): If slots are free for $\rho(m_{n+1})$ in $(x, x + t(m_{n+1}))$, Algorithm 1 finds the solution and that is the optimal one.

Case (ii): If slots are not free for $\rho(m_{n+1})$ in $(x, x + t(m_{n+1}))$, then the four operations CP, XP, SL, SR CSL and CSR are applied to enumerate all possible states for the current meeting, $m_{n+1,}$ from the current node.

The algorithm starts with the root node and the operations are applied at each level of the search tree. As the operations correctly capture all possible states from the root and the nodes are stored without duplicates into the heap, it generates a finite number of nodes on each level. Since the heap is used to get the most probable node for expansion and removed, it leads an optimal solution in a fewer levels of expansion, otherwise ends up with the empty heap.

Theorem 2: The heuristic function used in the Algorithm 1 gives the optimal solution.

Proof: Let L(n) be the minimum cost incurred on applying the operations to arrive n from the root. When this is proved, the optimality of the algorithm follows since when z(solution node) is chosen at line 8, L(z) will give the minimum cost operations required to reach z from the root.

Basis Step (i = 1): The first time we arrive at line 8, the root is chosen. Since L(root) is zero, L(root) is the minimum cost operations from root to root.

Inductive step: Assume that for all $k < i$, the k^{th} time we arrive at line 8, L(n) is the minimum cost operations from root to n.

Suppose that we are at line 8 for the i^{th} time and we remove n from the top of the heap with minimum value L(n).

First we show that if there is a sequence of operations applied from the root to a node w whose cost is less than L(n), then w is not in the heap (that is, w was previously removed from the heap and expanded at line 8).

Suppose that, by contradiction, w is in the heap. Let P be a minimum path from the root to w, x be the node nearest to the root on P that is in the heap and u be the predecessor of x on P. Then u is not in the heap, so u was chosen at line 8 during a previous iteration in the while loop. By the inductive assumption, L(u) is the minimum cost operations from the root to u.

Now $L(x) \leq L(u) + h(u, x)$, where h(u, x) is the heuristic value from u to x
$\leq$ cost of P
$< L(n)$.

But this inequality shows that n is not the node in the heap with minimum L(n). This contradiction completes the proof that if there is a path from the root to a node w whose cost is less than L(n), then w is not in the heap.

COMPLEXITY OF HEURISTIC SEARCH

The running time of heuristic search algorithms depends on the quality of the heuristic function and is proportional to the number of nodes expanded. The effect of admissible heuristic function is to reduce the effective branching factor of a heuristic search relative to the complete search from the search tree[1]. It is observed that the most probable nodes give shorter solution than other nodes. So, the nodes which are not most probable need

not be considered for expansion. In case, most probable nodes are not available then the version of IDA (Iterative-Deepening-A)[8] could be used to perform a series of depth-first searches, pruning a path and backtracking when the cost f(n) of a node n on the path exceeds a cut-off threshold for that iteration. IDA also guarantees an optimal solution if the heuristic function is admissible. IDA requires memory that is linear in the maximum search depth.

RESULTS AND DISCUSSION

The main goal of this work is to compare the performance of our approach with Sugihara approach[2] when rescheduling of meeting takes place. For any given meeting, $m_{n+1,}$ if there is no solution for any of the starting time instances in $w(m_{n+1})$ from the root configuration, then rescheduling or rearrangement of previously scheduled meetings are take place in both Sugihara[2] as well as in our approach.

Consider Table 1, for scheduling the meetings m_6. For the first five meeting requests, the meetings are fixed without using the rescheduling process in both approaches, whereas for the sixth meeting, the rescheduling process take place in both approaches. The A-approach generated 3 nodes to fix m_6 from T(5), whereas Sugihara's approach (Sugihara *et al*., 1989) generated 42 nodes.

It is observed that the A-algorithm with relevant set of operations inside the interval generates least number of nodes than A*-approach with common-set of operations in the 3rd, 4th columns of Table 1 and Sugihara approach. Another observation is that the number of nodes generated and the cost may vary based on the selection of node from the set of nodes having same heuristic value. If we look at the Fig. 5, nodes 1 and 6 are having same heuristic value. Node 1 has only 3 children whereas node 6 has 5 children. Node 6 is chosen for expansion because it has least cost. Even though node 7 is the solution node as well as the first child, the scheduler has to completely explore the node 6 and choose the optimal one.

In order to restrict the generation of some nodes which are not the most promising nodes, relevant set of operations could be used. In Fig. 5, the root node has 6 children. Among the six children, nodes 1 and 6 are

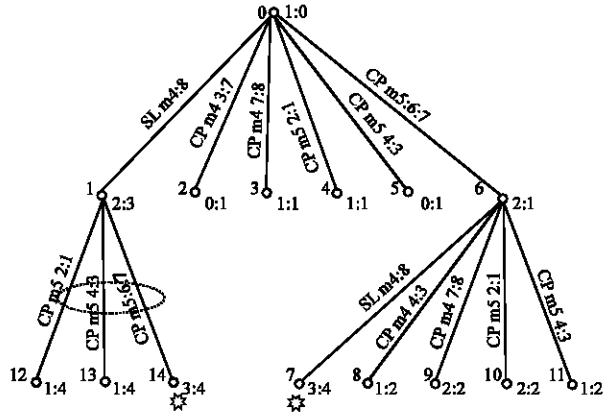

Fig. 5: A* with commen-set of operations

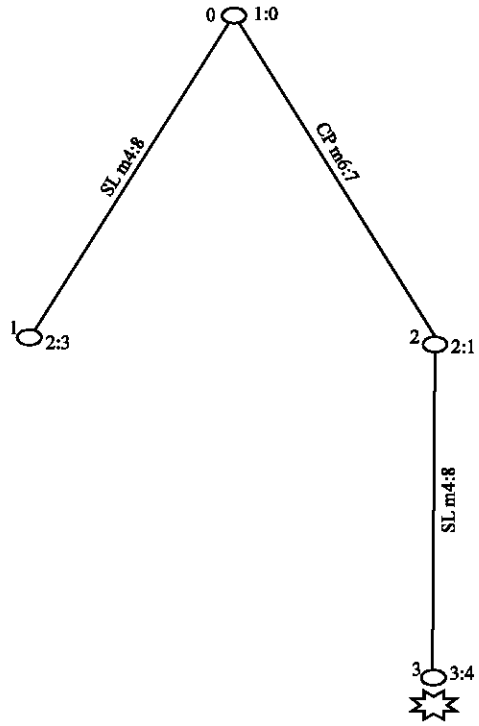

Fig. 6: A* with relevant-set operations

obtained by relevant-set operations. Nodes 1 and 6 have same heuristic value but node 6 has least cost, so node 6 is chosen for expansion. The relevant operation SL m_4 8 is applied to the node 6, node 7 is obtained, a solution node. If we look at the node 6 of Fig. 5, the children 8 to

Table 1: Comparison of A* algorithm, based on number of nodes generated, with Sugihara approach for rescheduling

		A* with		
mi	Sugihara approach	Common-set of operations & relevant operations outside of interval	Common-set of operations within the interval	Relevant-set of operations within the interval
m6	42	11	11	3

11 are not obtained by relevant operations. Hence, further expansion of node 6 is stopped. The process of generating a solution by relevant-set operations is shown in Fig. 6. The variations in the number of nodes generated and the cost of generation could be avoided when the relevant-set operations are applied instead of common-set operations in the search tree expansion.

CONCLUSION

In this study, we have discussed the scheduling and rescheduling of meetings for office automation and implemented with A-algorithm to speedup the solution instead of computing all solutions. Our approach performs better in practice than the earlier approach[2] by pruning the branches of the search tree using heuristic. A chooses only one branch or node in each level of the search tree for expansion. By reduction of the search space, over head operations involved in the processing also reduced. The proposed approach with A-algorithm gives the solution with optimum or near optimum rearrangement without checking all solutions. Even for the larger problem instances, that is, the number of persons, number of meetings and number of starting instances increase this approach performs better.

REFERENCES

1. Teger, S.L., 1983. Factors impacting the evolution of office automation. Proceedings of the IEEE, 71: 503-511.
2. Sugihara, K., T. Kikuno and N. Yoshida, 1989. A Meeting Scheduler for Office Automation. IEEE Transaction on Software Engin., 15: 1141-1146.
3. Garey, M.R. and D.S. Johnson, 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco, CA: Freeman.
4. Nilsson, N.J., 1990. Principles of Artificial Intelligence, Narosa Publishing House.
5. Rich, E. and K. Knight, 1995. Artificial Intelligence. Tata McGraw-Hill.
7. Russell, S.J. and P. Norvig, 2004. Artificial Intelligence-A Modern Approach. 2nd Edn., Pearson Education Series in Artificial intelligence.
6. Sugumaran, M., K.S. Easwarakumar and P. Narayanasamy, 2003. A New approach for Meeting Scheduling using A-Algorithm. Proceedings of the IEEE TENCON International Conference on Convergent Technologies for Asia-Pacific Region, 1: 419-423.
8. Korf, R.K., 2000. Recent Progress in the Design and Analysis of Admissible Heuristic Functions. American Association of Artificial Intelligence.