Scientia Iranica, Vol. 9, No. 3, pp 268–275

SCIENTIA
IRANICA

Research Note

An Efficient Exact Procedure for Project Scheduling Under Multiple Resource Constraints

M.R. Zamani[1]

A new procedure is presented for finding optimal solutions to resource constrained project scheduling problems. It traverses the decision tree by investigating the most promising partial schedule at each iteration, avoiding two main drawbacks associated with other silmilar procedures. These drawbacks are huge memory requirements and a time consuming task of selecting the most promising partial schedule among many candidates in each iteration. Computational experience indicates that the procedure finds optimal schedules for 100-activity projects, regardless of their strength of resource constraints.

INTRODUCTION

The Resource Constrained Project Scheduling (RCPS) problem may be stated as follows: A finite set of activities are given, each activity requiring a fixed integer duration and a fixed amount of one or more different resource types. Activities are subject to a set of precedence relations and may not be interrupted once started. There are specified fixed limits on the availability of each resource type and the objective is to minimize the project duration.

Because of the generality of RCPS problem, a great deal of research has been undertaken and many procedures have been proposed for the problem. A large number of these procedures generate feasible, rather than optimal solutions. A survey, along with an extensive reference list, on these heuristic procedures is presented in [1,2]. Among exact procedures, which generate optimal solutions, there are some efficient approaches presented by Davis and Heidorn [3], Talbot [4], Stinson, Davis and Khumavala [5], Bell and Park [6], Demeulemeester and Herroelen [7], Icmeli and Rom [8], Demeulemeester and Herroelen [9], Brucker et al. [10], Reyck and Herroelen [11] and Zamani [12].

Despite their efficiency, none of these approaches has been tested on tightly resource-constrained problems with more than 100 activities.

1. *Department of Industrial Engineering and Center for Systems Planning, Isfahan University of Technology, Isfahan, I.R. Iran.*

A NEW IMPLICIT ENUMERATION METHOD

The implicit enumeration method described in this paper, is based on an optimal learning search technique developed by Zamani and Shue [1] and Zamani [12]. The positive aspect of this technique is that it utilizes initial heuristic estimates and lower-bounds and continuously improves the estimates as the search process continues. It updates the heuristic estimates of states by comparing them with those of their children in the search process. Updating heuristic estimates, together with backtracking, help the method pause the expansion of a branch to consider other promising branches. This enables the technique to perform very effectively.

The method presented in this paper, systematically searches a tree of partial schedules, each partial schedule representing a node of the tree. Associated with each partial schedule is an earliest possible time at which new scheduling decisions are possible. In each iteration, all subsets of activities, including the "null" set, which can be started without violating resource and precedence constraints, are identified as the children of the current node in the tree. Each child, itself a new partial schedule, resolves the conflict regarding concurrent demand for resources by the activities that can start at the decision time. All leaf nodes of this tree are complete schedules and optimal solutions are located among these nodes. Despite using a backtracking schema, the method selects the most promising partial schedule at each iteration and,

therefore, reaches only one of the leaf nodes which is optimal solution.

Implicit enumeration methods for scheduling differ in the manner in which two important issues are addressed, first, the way in which the tree of partial schedules is constructed and second the means used to prune inferior branches from the tree. These two issues will be raised later.

Mechanisms of Improving Heuristic Estimates and Generating Optimal Solutions

For the purpose of improving heuristic estimates during the search process, as well as generating optimal solutions, an algorithm called Learning and Backtracking A* (LBA*) is used [1,12,13]. LBA* finds an optimal solution to a general state-space search problem as a sequence of decisions (operators) that transforms the initial state to the goal state. All states which are directly accessible from a state are called its neighbors. LBA* starts from the initial state, as the current state, and in each iteration either changes the current state with one of its neighbors or backtracks. It repeats this process until the current state becomes the goal state.

To represent a state-space formulation for any search problem, three issues should be specified: (a) The form of the state description and, in particular, the description of the initial state, (b) The set of operators and their effects on state description and (c) The properties of goal state description. In optimization problems it is not merely sufficient to find any path to a goal; it is also necessary to find some path optimizing some criterion (such as minimizing the total costs associated with operators applied). Associated with each state is a set of operators and each operator, which has a cost, transforms this state to one of its neighboring states. Each state has a heuristic estimate, lower bound, as well as a set of operators. The initial heuristic value of every state is an evaluation of the distance of that state from the goal state. In LBA* these initial heuristic values are assumed not to overestimate the actual distances and are improved in the search process by being compared with those of the neighboring states.

In LBA*, the optimal solution can be found in a single trial. From the initial state, as the current state, the search process starts by comparing its heuristic estimate with the "compound values" of all its neighboring states. For each neighboring state, the "compound value" represents the sum of its estimate to the goal state and the edge cost from the current state to the neighboring state. The neighboring state with the minimum compound value is chosen for the next stage of expansion. If smaller, the heuristic estimate of the current state is replaced with the minimum compound value, to reflect a more accurate estimate and a backtrack occurs. The process proceeds until a final solution is reached.

The reason for this replacement is that since the compound value represents a lower-bound on the actual distance to the goal through each of the neighbors, then the actual distance from the given state must be, at least, as large as the smallest of these compound values.

The LBA* can be implemented as follows:

Step 0: Consider a heuristic function to generate a non-overestimating initial heuristic estimate $h(x)$ for the distance of every state x to the goal state;

Step 1: Put the initial state on the backtrack list called OPEN;

Step 2: Let x be the top-most state on the OPEN list. If x is the goal state, stop;

Step 3: If x can be pruned by some pruning rule, replace its heuristic estimate, $h(x)$, with a very large value, remove x from the OPEN list and go to Step 2;

Step 4: Let $k(x,y)$ represent the cost of transforming state x to state y (the positive edge cost from state x to state y). Evaluate the compound value of $k(x,y)+h(y)$ for every child state y of x and find the state with the minimum value (ties can be broken randomly or based on any particular priority). Call this state x';

Step 5: If $h(x) >= k(x,x') + h(x')$, then add x' to the OPEN list as the top-most state and go to Step 2;

Step 6: Replace $h(x)$ with $k(x,x') + h(x')$;

Step 7: If x is not the root state, remove x from the OPEN list;

Step 8: Go to Step 2.

A full description of the algorithm, as well as a proof for the theorem that the application of the algorithm will lead to finding optimal solutions, can be found in [1,12].

The algorithm keeps all the states generated and their associated heuristic estimates in a tree structure, so that it can have access to them by moving forward and backward on the tree. This gives the algorithm the advantages of avoiding a great deal of duplicate searching, as well as being practical in the sense of memory requirements. The reason why the application of the algorithm to the RCPS problem, despite storing all the partial schedules generated, does not need a huge memory, has been described in the section of computational results.

Setting up a State-Space Formulation for the Problem

In terms of the above algorithm, in the RCPS problem, the initial state is an empty schedule where none of the activities has started yet and the goal state is a complete schedule where all activities have been completed. The process of finding the optimal solution for the RCPS problem can be considered as making a sequence of decisions (determining operators) that transform the null partial schedule (the initial state) into a complete schedule with minimum duration (the goal).

A state (partial schedule) is a schedule in which some of the activities have already started. Operators associated with each state are all the possible decisions about scheduling other activities with respect to both precedence and resource constraints. In the above algorithm, the cost of any operator, $k(x, x0)$, can depend on both states transformed to and from. However in the RCPS problem, this cost depends only on the state transformed from and is defined as the minimum time remaining for at least one of the incomplete activities of the state to be completed. Therefore, associated with each operator is a cost that determines the minimum time left for an incomplete activity of the state transformed from to be accomplished.

Since each state (partial schedule) is a set of some incomplete activities, it can be considered as a new project with a heuristic estimate as a lower-bound on its completion time. What makes the minimum completion time of a project difficult to find is the simultaneous existence of two sets of precedence and resource constraints. Many different methods have been proposed for finding lowerbounds on these partial schedules [14-16]. Here, the modified version of precedence- and resource-based lowerbounds are used as follows. By ignoring resource constraints, the RCPS problem becomes a simple problem whose CPM time duration is a lower-bound to the optimal solution of the original problem. This lower-bound can be improved by considering the fact that none of the activities in the "unscheduled" set can be started unless at least one of the activities in the "in progress" set is completed.

It is clear that the more accurate non-overestimating heuristic estimates of states are the more efficiently the method performs. Therefore, for every state, a second non-overestimating heuristic estimate is calculated and the maximum estimate between the two is selected. The calculation of this heuristic estimate is based on ignoring precedence-constraints. When precedence-constraints are ignored, there are some resource requirements needed for incomplete activities. These resource requirements are multiplied by their duration and the sum result is divided by the amount of that resource available per day, yielding a minimum number of days for which the project continues. In the case of multiple-resource constraints, this idea is generalized by finding the maximum value among different types of resources. As stated, the maximum estimate is selected from the two non-overestimating estimates, one calculated by ignoring resource constraints and the other calculated by ignoring precedence constraints. The third heuristic estimate calculated is based on repeatedly solving subproblems.

Having discussed the mechanism of calculating heuristic estimates, the way in which the tree of partial schedules is constructed is now described. The space of states reachable from the initial state can be considered as a tree containing nodes corresponding to the states. The nodes of this tree are linked to each other by arcs that correspond to operators (decisions). The root of this tree, the initial state, is an empty schedule and the intermediate nodes are partial schedules. In this tree, a leaf (terminal node) is a complete feasible schedule in which all activities have already started.

As the tree is expanded from some given intermediate node (partial schedule), a new set of partial schedules is created. Each member of this new set has, in common with its parent, all scheduling decisions made previously. The only difference between any partial schedule and its parent is that it includes one new decision about the scheduling of one or more activities that have not yet been scheduled. This decision is made when at least one of the incomplete activities of a partial schedule is completed and, consequently, some resources are released.

The process of constructing the tree starts with generating a null partial schedule as the current node. After calculating heuristic estimates, from among the children of the current node, a child with the minimum heuristic estimate is selected. Ties are broken in favor of children which have managed to start more activities and the remaining ties are broken randomly. The branching process takes place from the selected child, which becomes the current node and its children, if not yet having done so, are generated. Among these children, based on their heuristic estimate, the best one is selected again. This branching process continues until all activities are scheduled or an updating in the heuristic estimate of a current node occurs. In the case of such updating, if the current node is not the root (initial state), a backtracking from the current node occurs and its parent will become the current node; then the process continues afterwards. When the heuristic estimate of the initial state is updated, no backtracking occurs and the process proceeds.

If it can be established that further branching from a node cannot lead to an optimal solution, then the node can be pruned away. Two means are

used to prune inferior branches from the tree. The first pruning rule used is called left-shift rule. In the tree of partial schedules, if a node includes any activity already scheduled which can be left-shifted to an earlier start time without violating either resource or precedence constraints, this node can be pruned away. This dominance role was originally established by Schrage [17] and, since then, has been used by many other researchers in this area [6,7,18-20].

The second pruning rule used works based on the comparison of some nodes as follows. Every current node at the time of expansion is checked to see if a node with the same set of scheduled activities and the following three conditions has previously been expanded: (a) Its completed activities include those of the current node; (b) The starting times of its uncompleted activities do not exceed those of the current node and (c) Its time of next decision is less than, or equal to, that of the current node. The current node is pruned away, if such a previously expanded node exists. This pruning rule is similar to that used by Stinson et al. [5], the only difference is that they considered the previously expanded nodes whose scheduled activities included those of the current node, rather than being the same activities. Considering the nodes with the same activities weakens the pruning power to some extent, however, this can be handled with just a fast hash procedure rather than a time consuming key-search one.

A NUMERICAL EXAMPLE

A sample problem from [7,12] is used to explain the application, as well as the efficiency of the method. Figure 1 illustrates the problem. The two dummy activities "a" and "i" define the beginning and end of the project, respectively. Figure 2 depicts the tree of partial schedules (states) generated by the method. At the root of this tree, there stands state 1, resulting from the only possible decision at time zero. As stated, activity "a" is a dummy activity, which shows the beginning of the project and, based on the project network, no activity can start before this dummy activity is completed. The duration of this activity is zero; therefore, at time zero, when it is completed, there are four eligible activities to start, namely, activities b, c, d and e. Because of the constraint on the resources, it is not possible to start all four activities simultaneously and, even in the case of such a possibility, all other possible decisions should be considered.

In determining possible decisions at time zero, when activity "a" is completed, all feasible combinations of eligible activities that can enter the corresponding partial schedule are considered. As stated, even in the case where a group of activities can start together, the scheduling of either of them alone or a subgroup of them is not ignored. For instance, although in state 2, the three activities b, c and d are managed to start together, state 4 schedules only activities b and c or, in state 8, just activity b starts. Notice that the left-shift

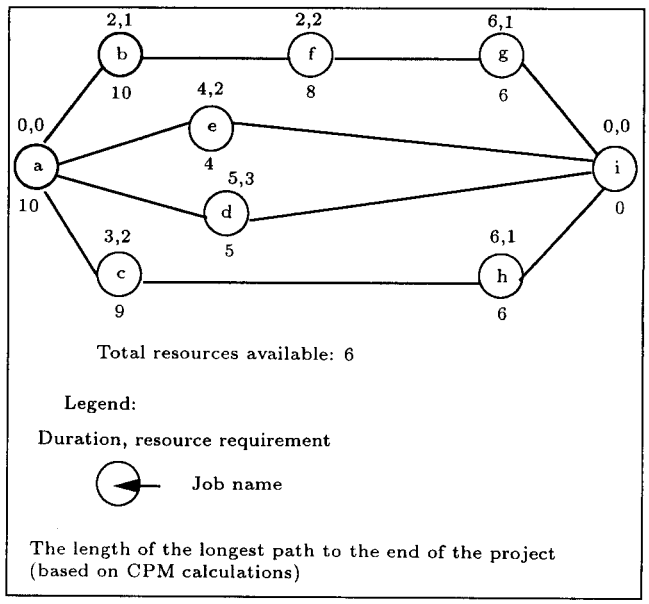

Figure 1. A sample problem.

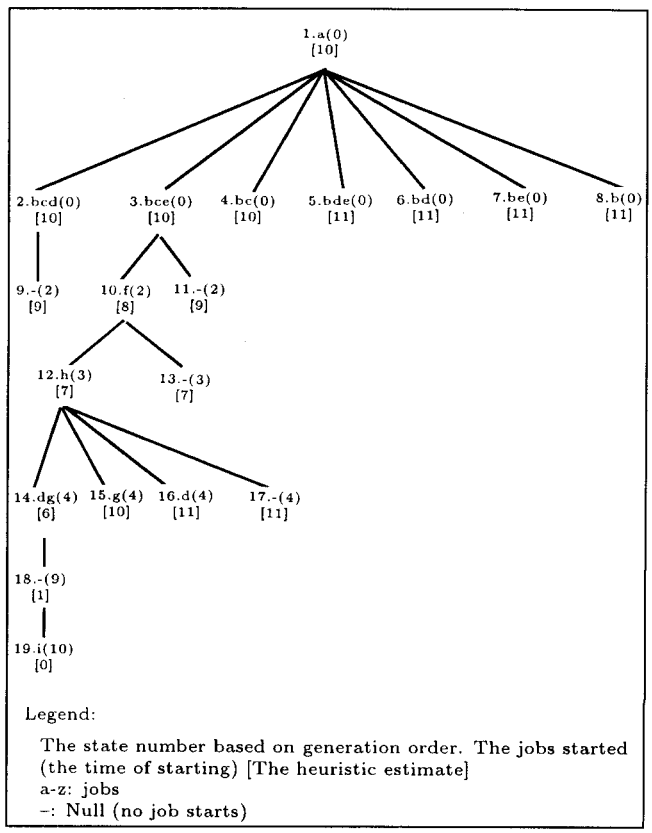

Figure 2. The tree of partial schedules (states) in which each partial schedule is not represented in detail.

rule does not permit states like a state starting only activities c and d to be shown in Figure 2. The reason this state is not shown is that, if only activities c and d were started, then, by the time of the next decision, at time 3 when activity c would be completed, resource availability would permit activity b, which takes only 2 days and needs only one unit of resources to be completed. Figure 3 shows how activity "b" in this situation can be left-shifted.

The construction of the tree begins by generating the initial state, state 1, which is put in the OPEN list. In this state only the dummy activity "a" starts and all other activities are in the "unscheduled" set. The heuristic estimate with this state is 10, which is a lower-bound on the duration of the entire project and has been obtained by ignoring resource constraints. After the completion of activity "a", again at time zero, all possible decisions are considered and those which cannot be pruned by the left-shift rule, i.e., seven states, are determined. The costs of transforming from state 1 to all these states; $k(1,2), k(1,3), \ldots k(1,8)$, which are equal to the duration of activity "a", are zero and need not be added to the heuristic estimate of their corresponding state for the selection of the best state. Therefore, to select the best state to transform to/from state 1, it is enough to consider just $h(2), h(3), \ldots h(8)$ rather than $k(1,2)+h(2), k(1,3)+h(3), \ldots k(1,8)+h(8)$ as stated in Step 4 of the algorithm.

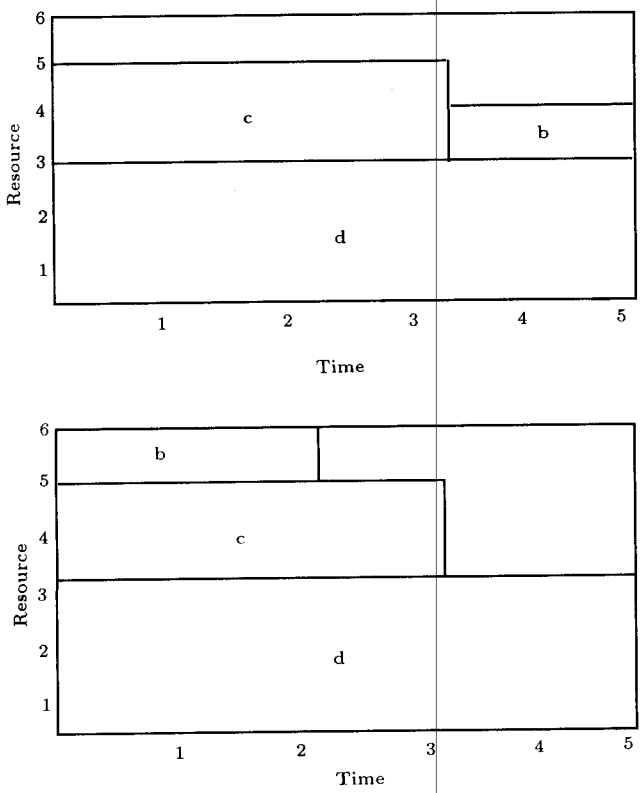

Figure 3. Left-shifting activity b in the situation where activities c and d start at time 0.

As shown in Figure 2, among the seven candidates, the three of them have the same heuristic estimate, $h(2) = h(3) = h(4) = 10$. In this case, the priority is given to the states which have managed to start more activities and, then, the remaining ties are broken randomly. Among these three states, there are two states which have both managed to start three activities, namely states 2 and 3. The tie is broken randomly and state 2 is selected. The time associated with this state is zero, when activities b, c, and d start and its heuristic estimate is 10. Since the cost of transforming from state 1 to state 2, $k(1,2) = 0$, which is the difference between the times at which they started their activities added to the heuristic estimate of state 2, $h(2) = 10$, is not greater than the heuristic estimate of state 1 ($h(1) >= k(1,2) + h(2)$), no updating in the heuristic estimate of state 1 occurs and state 2 is put on the OPEN list as the current state.

To determine the candidate states to transform to/from state 2, the time is first determined when the next set of scheduling decisions, with regard to this state, can potentially be made. In state 2 the activities b, c and d were scheduled to start at time zero, among them activity b is completed sooner than the other two (at time 2). Therefore, the time associated with each child of state 2, when some other decisions can be made, is 2. With respect to precedence constraints, when activity b is completed, activity f can start, but it needs 2 units of resources whereas just one unit of resource is available and the remaining 5 units are still seized by activities c and d. Hence, regarding state 2, just one possible decision exists at time 2: Starting no activity (-). The result of this decision is represented as state 9. The heuristic estimate associated with state 9, $h(9)$, is 9 and, since $k(2,9) + h(9) > h(2)$, based on steps 6 and 7 of the algorithm, the value of $h(2)$, 10, is replaced with $k(2,9) + h(9)$, 11, and a backtracking occurs which causes state 2 to be removed from the OPEN list.

Now, again, state 1 is at the top of the OPEN list (but this time all its children have been generated and there is no need to regenerate them). This time, among the candidate states to transform to, there are 2 rather than 3 states, namely, states 3 and 4 both with heuristic estimate 10. Again, the tie is broken in favor of the state which has managed to start more activities and state 3 is selected as the best state to transform to/from state 1. As depicted in Figure 2, from this point on, the expansion of states continues and no other backtracking is encountered until the optimal solution is obtained. To obtain the optimal solution, 7 states have been expanded, namely, states 1, 2, 3, 10, 12, 14 and 18, and just one backtracking has occurred.

As shown in Figure 2, the method does not

have the drawback of best-first search schemes which suffer from the redundancy of calculation and the need for high memory for keeping partial schedules as independent data. On the other hand, it does not have the disadvantages with backtracking schemes, in which, once a node is selected to be expanded, all its children must be exhausted before any other node is selected. This was shown in this example when a branch originating from state 2, before being exhausted, was put aside and a branch originating from state 3 was considered instead. The branch originating from state 2 would have had the chance of reconsideration if all states 3, 4, 5, 6, 7 and 8 had held updated heuristic estimates greater than, or equal to, those of states 2 and 11. Demulemeester and Herroelen [7], who used a backtracking scheme, had to exhaust this branch before considering the branch originating from state 3.

COMPUTATIONAL RESULTS

There are two main generators for generating RCPS problems [18,21]. Because of its simplicity and generality, [18] was selected and the procedure was tested on 68 randomly generated problems. A feature considered in [18] that appears very important is the strength of resource constraints.

This feature is measured as the ratio between the total resource requirements (the sum of the resource requirements of each activity by its processing time) and the total available resources (the sum of resource availability by the length of the longest path in the original graph). If this ratio varied between 0.5 and 0.99 the problem was considered loosely constrained. Problems with ratios between 1.0 and 1.5 were considered to be tightly constrained. This ratio was originally mentioned by Davis and Patterson [22] who generated problems with ratios between 0.53 and 1.5.

Christofides et al. [18] generated problems with duration in the range of 1-9 and resource requirements in the range of 0-6. They also considered six units of resource availability and in their graphs of precedence constraints, the ratio between arcs and vertices varied between 1 and 3. With this network structure and resource requirements, they tried problems with 25 activities and a maximum of three resource types. Half of the problems (20) were tightly constrained and the others were loosely constrained. Despite considering such a small number of activities, they stated that 25 percent of the tightly constrained problems (5) could not be solved within the limit of 60 CPU seconds using the UNIVAC 1100 mainframe. In this paper, however, with the same network structure and resource requirements, problems were tried with the number of activities varying between 80 and 100 and number of resource types being between one and six. Half of these problems were loosely constrained and the others were tightly constrained. All 68 problems, regardless of their strength of resources, were solved within the limit of 60 CPU seconds using IBM PC DX486. On average, each problem took 8.02 seconds to be solved and the upper limit for computation times was 57.64 seconds. The lower limit was just 0.1 second and the tightly constrained problems took, on average, only thirty percent less time to be solved compared to loosely constrained ones.

These computational results show the capability of the method for solving a wide range of medium-sized RCPS problems and any method capable of solving medium-sized combinatorial problems, undoubtedly also performs well on small-sized ones. However, it was decided to apply the method to a complicated small-sized problem, too. This problem was selected from [5] and has 43 activities with three different resource types. Bell and Park [6] have also solved this problem and reported their computation times on different machines. They reported that this problem required 122 minutes on Macintosh plus, 30 minutes on Macintosh II and 10.7 minutes on a Sun workstation. Stinson's algorithm solved this problem in 6.05 minutes on a mainframe IBM 370/155. As was expected, the method described in this paper solved this small-sized problem in a very short time, in effect, as small as 1.65 seconds on a personal computer DX486.

There are some superficial similarities between this algorithm and Truncated Branch and Bound Algorithms which Use Virtual Upper Bounds (TBBAVUB). This might strengthen the hypothesis that the algorithm works with the same efficiency as worked with previously. To test this hypothesis, the computer program was changed so that it worked based on TBBAVUB (the object oriented property of C^{++} made these changes easy). The result was that the program based on TBBAVUB solved less than 10 percent of the test problems within the time limit, whereas this program was able to solve 100 percent of the test problems. Besides, on average, the presented program worked more than forty times faster. The causes for these performance distinctions are as follows.

For each successive restart, TBBAVUB had to search a larger area of the tree and, therefore, compared to its preceding restart, taking more time to find a better lower-bound for the root. The situation was the reverse for the algorithm described by the author, because it kept and used all partial schedules generated previously, as well as their updated heuristic estimates. In other words, this algorithm continuously improved the heuristic estimates of partial schedules and used updated estimates rather than repeating a lot of duplicated searching, as TBBAVUB was forced

to do. This can be verified by looking at steps 0 through 8 of LBA*. These steps show that heuristic estimates ($h(x)$'s) never lose their updated values, even when the algorithm changes the heuristic estimate of the root and, in the terminology of TBBAVUB, restarts. In order to keep all partial schedules and their associated heuristic estimates, generated, the algorithm uses a tree structure.

In regards to using the tree structure for keeping partial schedules, the algorithm is similar to best-first schemes [6]. However, as is known, best-first schemes are impractical because of their huge space requirements. None of the best-first schemes has been able to handle tightly resource-constrained problems with more than 60 activities, whereas, as shown in this paper, such problems with more than 100 activities have been handled by the algorithm described by the authors and their optimal solutions have been obtained. The reason why this algorithm does not require huge space requirements lies in the way in which partial schedules can be represented. Because the algorithm is based on a depth-first scheme and, hence, can easily move forward and backward on the tree structure to access partial schedules and to update their heuristic estimates, partial schedules need not be stored in detail. Each node of this tree includes only a decision about starting one or more activities (rather than representing all specifications of a partial schedule). By moving forward and backward on this tree, the algorithm puts these decisions together or disintegrates them, respectively, to find out the actual specifications of the partial schedule associated with each node of the tree. In other words, by traversing a path from the root of the tree to any node of it and looking at the activities which have been started on the path, the associated partial schedule of that node is specified. Therefore, by using a tree structure and a depth-first scheme, which accepts some extra calculations for building partial schedules, the algorithm does not suffer from the need of a huge memory for keeping partial schedules as independent data. This gives the algorithm the advantage of being practical in the sense of memory requirements, as well as an ability to avoid a great deal of duplicated searching.

REFERENCES

1. Zamani, M.R. and Shue, L.Y. "Developing an optimal learning search method for networks", *Scientica Iranica*, **2**(3), pp 197-206 (1995).
2. Tsubakitani, S. and Deckro, R.F. "Heuristic for multi-project with limited resources in the housing industry", *European Journal of Operational Research*, **49**(1), pp 80-91 (1990).
3. Davis, E.W. and Hiedorn, G.E. "An algorithm for optimal project scheduling under multiple resource constraint", *Management Science*, **17**(12), pp 803-816 (1971).
4. Talbot, F.B. "An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems", *Management Science*, **24**(11), pp 1163-1174 (1978).
5. Stinson, J.P., Davis E.W. and Khumavala, B.M. "Multiple resource-constrained scheduling using branch and bound", *AIIE Transactions*, **10**(3), pp 252-259 (1978).
6. Bell, C.E. and Park, K. "Solving resource-constrained project scheduling problems by A* search", *Naval Research Logistics*, **37**(1), pp 280-318 (1990).
7. Demeulemeester, E. and Herroelen, W. "A branch-and-bound procedure for multiple resource-constrained project scheduling problem", *Management Science*, **38**(12), pp 1803-1818 (1992).
8. Icmeli, O. and Rom, W.O. "Solving the resource constrained project scheduling problem with optimization subroutine library", *Journal of Computers and Operations Research*, **23**(8), pp 801-817 (1996).
9. Demeulemeester, E. and Herroelen, W. "A new benchmark results for the resource-constrained project scheduling problem", *Management Science*, **43**(11), pp 1485-1492 (1997).
10. Brucker, P., Knust, S., Scoo, A. and Thiele, O. "A branch and bound algorithm for the resource-constrained project scheduling problem", *European Journal of Operational Research*, **107**(1), pp 272-288 (1998).
11. Reyck, B.D. and Herroelen, W. "A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations", *European Journal of Operational Research*, **111**(1), pp 152-174 (1998).
12. Zamani, M.R. "A high performance exact method for resource constrained project scheduling problems", *Journal of Computers and Operations Research*, **28**(14), pp 1387-1401 (2001).
13. Korf, R.E. "Real-time heuristic search", *Artificial Intelligence*, **42**(2), pp 189-211 (1990).
14. Ozdamar, L. and Ulusoy, G. "A survey on the resource-constrained project scheduling problem", *IIE Transactions*, **27**(5), pp 574-586 (1995).
15. Davis, E.W. "Network resource allocation", *Industrial Engineering*, **21**(4), pp 59-69 (1974).
16. Deckro, R.F., Winkofsky, E.P. and Herbert, J.E. "A decomposition approach to multi-project scheduling", *European Journal of Operational Research*, **51**(1), pp 110-118 (1991).
17. Schrage, L. "Solving resource-constrained network problems by implicit enumeration non-perceptive case", *Operations Research*, **18**(2), pp 263-278 (1969).

18. Christofides, N., Alvarez-Valdes, R. and Taramit, J.M. "Project scheduling with resource constraints, a branch and bound approach", *European Journal of Operational Research*, **29**(3), pp 262-273 (1987).
19. Nillson, N.J., *Principles of Artificial Intelligence*, California, Tioga, Palo Alto (1980).
20. Patterson, J.H. "A comparison of exact approaches for solving the multi constrained resource project scheduling", *Management Science*, pp 854-867 (1984).
21. Kolisch, R., Sprecher, A. and Drexl, A. "Characterization and generation of a general class of resource-constrained project scheduling problems", *Management Science*, **41**(10), pp 1693-1702 (1995).
22. Davis, E.W. and Patterson, J. "A comparison of heuristic and optimal solutions in resource-constrained project scheduling", *Management Science*, **21**(8), pp 944-955 (1975).