A Heuristic Approach to Incremental and Reactive Scheduling

Jidé B. Odubiyi and David R. Zoch

Ford Aerospace Corporation
Space Systems Engineering Operation
7375 Executive Place
Seabrook, MD 20706

Abstract

This paper describes a heuristic approach to incremental and reactive scheduling. Incremental scheduling is the process of modifying an existing schedule if the initial schedule does not meet its stated initial goals. Modifications made to a schedule during incremental scheduling typically consist of adding one or more activities by re-scheduling existing activities. Reactive scheduling is performed when changes need to be made to an existing schedule due to uncertain or dynamic environments such as changes in available resources or the occurrence of targets of opportunity. Only minor changes are made during both incremental and reactive scheduling because a goal of re-scheduling procedures is to minimally impact the schedule.

A scheduling system generates a schedule in three phases. An initial batch scheduling phase, an incremental scheduling phase and a reactive scheduling phase. During the first phase, no rescheduling is attempted. All user requests are submitted to the scheduler and an initial schedule is created. During the second phase, non-computationally complex strategies must be used since the number of possible schedules that can be generated increases exponentially with the number of requests. Since simple strategies must be used for initial schedule creation, any schedule can potentially be greatly improved through the use of an incremental scheduling phase.

Reactive scheduling occurs in near real-time in response to the occurrence of targets of opportunity. Consequently, a reactive scheduler must be able to generate schedules within acceptable time limits. Manual reactive scheduling is an inefficient strategy, and automated exhaustive search techniques are infeasible because of time limits.

This paper describes the heuristic search techniques employed by the Request Oriented Scheduling Engine (ROSE), a prototype generic scheduler (3). Specifically, we describe heuristics that efficiently approximate the cost of reaching a goal from a given state and effective mechanisms for controlling search.

Introduction

Scheduling the Tracking and Data Relay Satellite System's (TDRSS) communications' events and user preferences present the NASA-GSFC's Network Control Center's personnel with a very complex scheduling problem. The schedulers must deal with limited TDRSS resources, such as antennas, ground equipment and communications bandwidth. In addition to these resource constraints, the scheduling requirements also have user constraints, such as TDRS visibility of user spacecraft, as well as temporal and dynamic (request placement with respect to other scheduled requests) constraints.

A sample request is shown in Figure 1 where a user of the Upper Atmospheric Research Satellite (UARS) requests the NCC to schedule a house-keeping activity for UARS 19 times, once every 80 minutes, and each request must start within a 40 minute time window. In addition, each request must use a single access antenna from TDRS-East for a period of 15 minutes and it should be scheduled when UARS is in view of TDRS-East.

The scheduling of these requests is premised by the fact that any instances of this generic request should be scheduled only if alternate request instances in a generic request which performs UARS house-keeping using TDRS-West, have not been scheduled. The NCC personnel receive and process several hundreds of requests with more complex requirements from several users on a weekly basis. During the space station era users will generate thousands of such requests.

The Request Oriented Scheduling Engine (ROSE) is a generic scheduling software prototype which has successfully demonstrated the scheduling of user requests in the scheduling of scientific instrument operations for the Space Station distributed scheduling environment, and the scheduling of user requests in the NCC environment. The rest of this paper provides a brief description of the ROSE scheduler, incremental and reactive scheduling processes and the implementation of a hybrid search algorithm to speed automated rescheduling activities.

THE PROBLEM

With thousands of requests to schedule, the initial batch scheduling approach does not usually meet the user's scheduling goals. Also the initial schedule is sub-optimal due to the necessity to use simple heuristics. ROSE provides tools to allow the user to do re-scheduling by deleting or moving scheduled requests, adding unscheduled requests, or relaxing requests' constraints manually.

When re-scheduling involves a large number of requests, in order to find a location for an unscheduled request, extensive search of the attributes (i.e., constraints, resources requirements, etc.) of scheduled requests must be performed. This step is required in order to identify appropriate heuristics to improve the search. Also, if the schedule is to be generated in near real-time, the search algorithm must be efficient and fast enough for the resulting schedule to be of any use. Therefore, an automated incremental and reactive scheduling capability is needed in ROSE.

ROSE - A Generic Scheduling Software System

The ROSE software prototype has been developed to provide NASA customers in the Space Station distributed scheduling environment with an automated mechanism for communicating their scheduling requirements to NASA-Goddard Space Flight Center (NASA-GSFC) and receiving their scheduled requests. In ROSE, the feasibility of communicating user requests from remote locations (where appropriate) to a scheduler is being explored. The scheduling requirements are communicated to a NASA scheduler in a Flexible Envelope Request Notation (FERN). This notation enables a user to specify his/her requests with preferential constraints. ROSE/FERN is described in more detail in [3].

ROSE is a generic scheduler currently running on the Symbolics computer workstation with the Genera 7.0 operating system and the Common LISP language on the Symbolics computer workstation at the NASA-GSFC in code 520. Figure 2 depicts the ROSE user interface. The interface consists of several windows. The user executes many of the ROSE commands by activating the menu items in the Commands window. The NCC scheduling network window shows three users (GRO, STS and UARS) in this example with the NCC as the scheduler. Generic requests from the users to the schedulers are monitored and presented in a scrollable window, titled "Real-Time Message Monitoring".

Figure 2 also shows a day's schedule in the window titled "Timeline of Scheduled Requests". Scheduled requests are displayed as unshaded rectangular boxes along a timeline. The names of the user or campaign are displayed to the left of the corresponding scheduled requests. More information about each scheduled requests can be displayed, and the parameters of the request can be modified through the interface. Below the requests in Figure 2 in shaded rectangles is a sample of TDRS's visibility constraint. The first row of shaded rectangles displays the time windows when UARS is in view of the TDRS-West antenna, while the

next row depicts when the same spacecraft is in view of TDRS-East antenna. The bottom right corner window displays a list of unscheduled requests. The window is scrollable, and the user has the option to scroll the window for a list of other unscheduled requests. The user can also mouse each request to obtain a detailed information about the request. ROSE has many features that enable the user to display information about schedules, requests and resource usage.

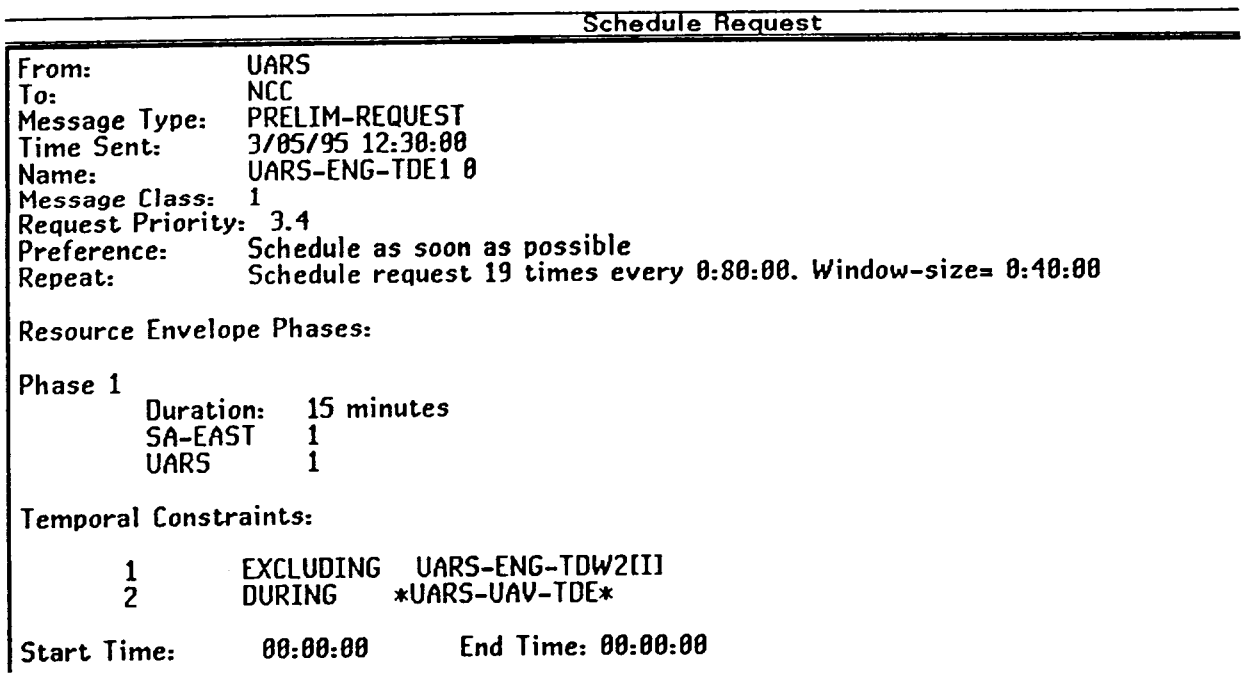

FIGURE 1. A Sample User's Generic Request

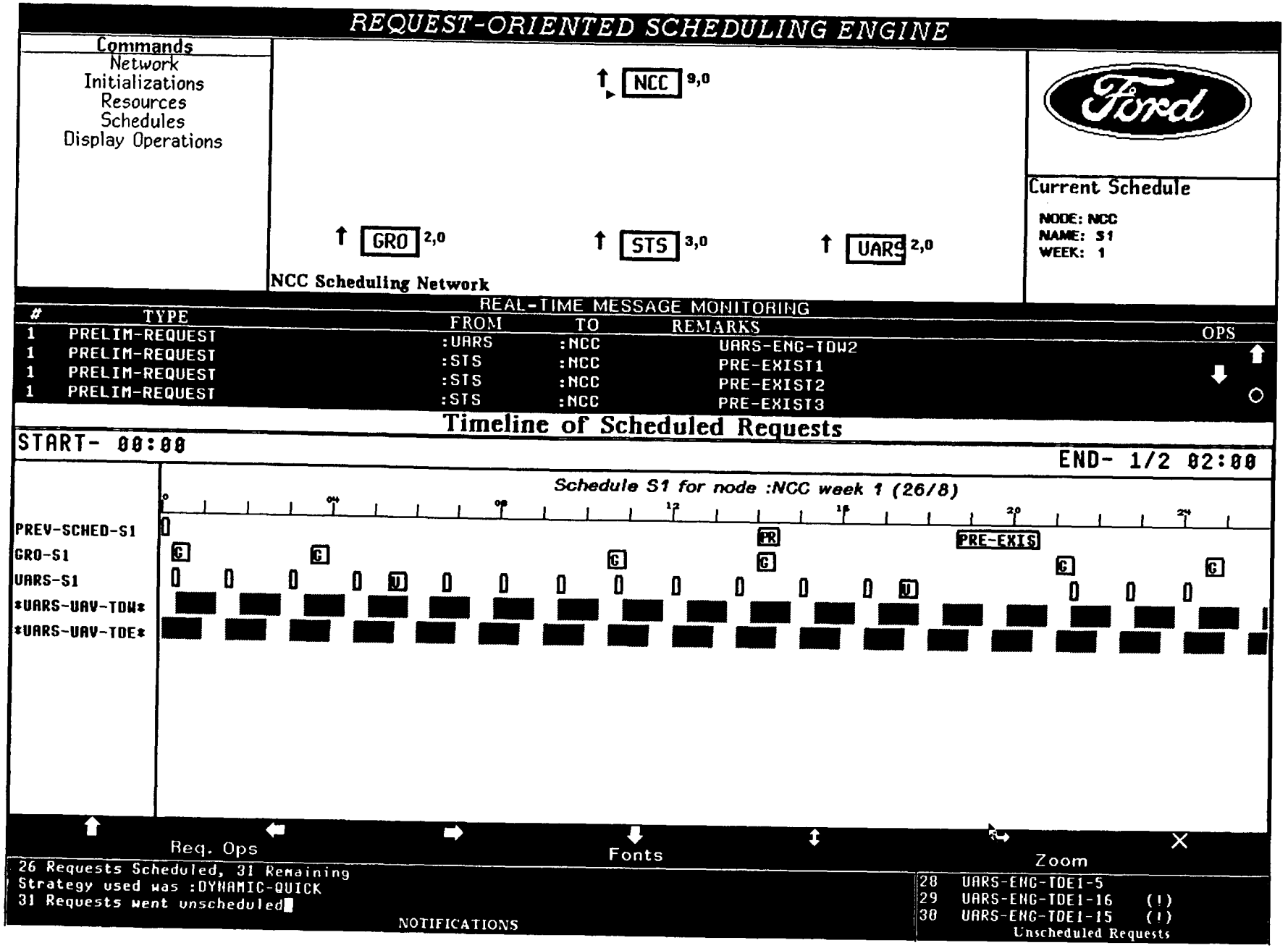

FIGURE 2. ROSE User Interface

Re-Scheduling Strategies to Meet Scheduling Goals

ROSE is a generic scheduler, and it has been developed so that the user can generate schedules with different scheduling goals. When the initial schedule does not meet its goal, the scheduling software, (i.e. ROSE) may take one or more of the following conflict resolution strategies.

- Relax the requirements of unscheduled requests
- Overbook certain resources
- Relax the requirements of scheduled requests or de-allocate certain resources
- Acquire additional resources from another scheduler in a distributed scheduling architecture
- Implement an Incremental Scheduling strategy
- Implement a Reactive Scheduling strategy which incorporates one or more of the courses of action above

In this paper, we only describe the incremental and reactive scheduling strategies for re-scheduling.

Scheduling Goals

A user's scheduling goals can take several forms, for instance:

- Create a schedule within the time limit of T hours.
- Schedule all requests above priority N
- Reserve X% of resource R during time T1.....T2
- Schedule as many requests as possible

With the scheduling goal(s) identified, the ROSE software generates a plan as to which actions to take and in what order, and attempts to generate a schedule that meets the user's goal(s). For example, if the user's goal is to schedule as many requests as possible, the plan may include a step to relax the resource requirements of all requests. Figure 3 depicts a process flow chart used in ROSE for incremental scheduling. ROSE applies each strategy in the plan to the initial schedule until either the user's goals are met or the plan is exhausted.

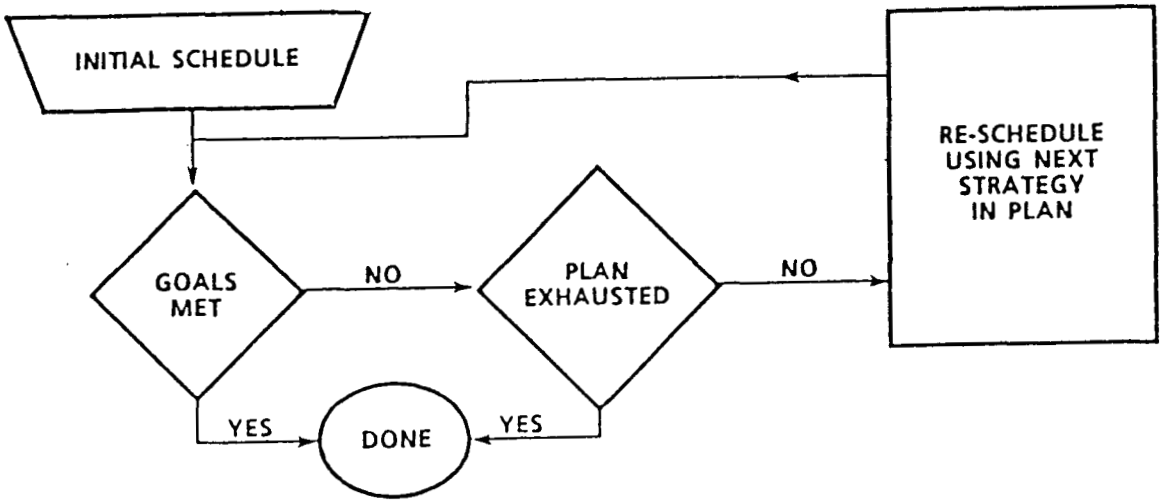

FIGURE 3. A Process Flow Chart for an Incremental Scheduling Strategy

Reactive Scheduling

When an unexpected occurrence of some events triggers the need for replanning, ROSE provides the capabilities to apply the opportunistic scheduling procedure diagrammed in Figure 4. Re-scheduling is performed by adding, moving or deleting requests until the effects of the impacts are eliminated.

Reactive scheduling is used to modify a schedule already in use. Therefore, conflict resolution strategies which are valid in incremental scheduling may not be applicable for reactive scheduling. For example, if a week's schedule already in use requires reactive scheduling at mid-week (i.e. Wednesday), then any requests prior to Wednesday cannot be moved. In other words, anything in the past cannot be moved, and no requests can be scheduled prior to Wednesday. In incremental scheduling, the scheduling system focusses its attention on re-scheduling existing requests in order to accommodate additional requests. In reactive scheduling, however, the scheduler must consider alternative strategies, such as relaxing the requirements of requests in order to minimally impact the schedule. Still, the goal in reactive scheduling is to minimally impact the rest of the schedule.

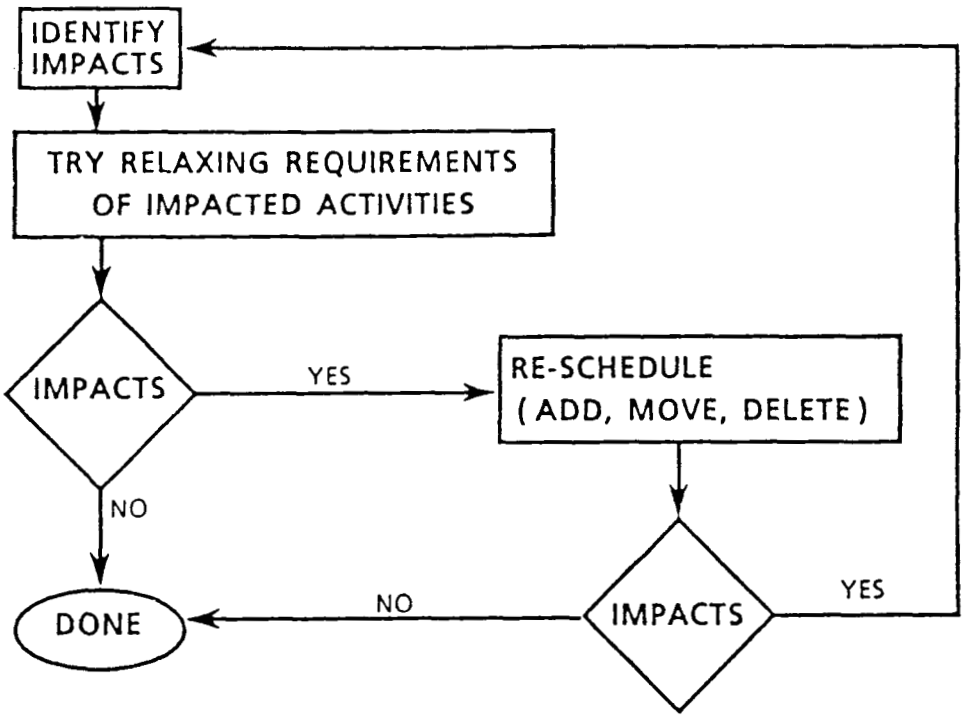

FIGURE 4. A Process Flow Chart for a Reactive Scheduling Strategy

Heuristics for Efficient Re-Scheduling

In the field of Artificial Intelligence, several researchers have focussed on developing efficient search techniques for complex mathematically intractable problems. Simon (1962) proposed the "Hierarchical" approach during search by planning at different levels of abstraction. In 1975, Sacerdoti proposed the "Least Commitment" approach which suggests delaying any decision making as much as possible until most of the facts are known and thereby reducing the amount of backtracking. In 1979, Hayes-Roth proposed the "Opportunistic Reasoning" approach by focussing search in highly constrained areas or areas of highest certainty. Dependency directed backtracking is another popular approach employed in searching to reduce the search space of states. Mark S. Fox (7) research efforts on constraint-directed reasoning provides several approaches to reducing the amount of search required in planning. This paper applies a hybrid approach by combining the generate and test problem solving method and the A* algorithm to search the problem space for a solution to a re-scheduling problem.

Implementation of a Heuristic for Efficient Re-Scheduling

We have implemented a hybrid algorithm similar to the A* (Best-first search) algorithm to provide effective search during the re-scheduling process.

Figure 5 shows a directed graph of the search space for re-scheduling in the ROSE scheduling software system. The problem space is developed from the steps involved in re-scheduling in ROSE as described earlier for Figure 2.

Our algorithm searches a directed graph in which each node represents a state in the problem space. It is used to find a minimal-cost overall path or any other path as quickly as possible. In ROSE, the initial state is the initial batch schedule and a request to be scheduled; a goal state is reached when the unscheduled request is scheduled, and no existing request violates any of its resource requirements or temporal or dynamic constraints. The intermediate states consists of the possible states between the initial state and a goal state.

To accomplish the objective of going from the initial state to the goal state in Figure 5, we employ the generate and test problem solving strategy to generate the rules to guide possible moves. These rules are described in the steps below:

Step 1. Start at level 0 and select an unscheduled request.

Step 2. Generate a set of start times and assign ratings to how good the possible locations where the request can be scheduled are. Good locations are those where the minimum number of constraints are violated and the minimum number and amount of resources are required.

Step 3. Schedule this request in a location where it is constrained the least, either by a resource or a dynamic constraint. Break ties by selecting the location with the earliest time along the timeline. A location with a missing resource is preferred over another location with a violated dynamic constraint. A temporal constraint must not be violated. This step will usually invalidate the current schedule.

Step 4. Create a window around all the requests overlapped by the current request, and identify any such requests as possible candidates to be moved. The local goal is to move one or more requests and re-schedule them elsewhere to make the schedule within this window valid.

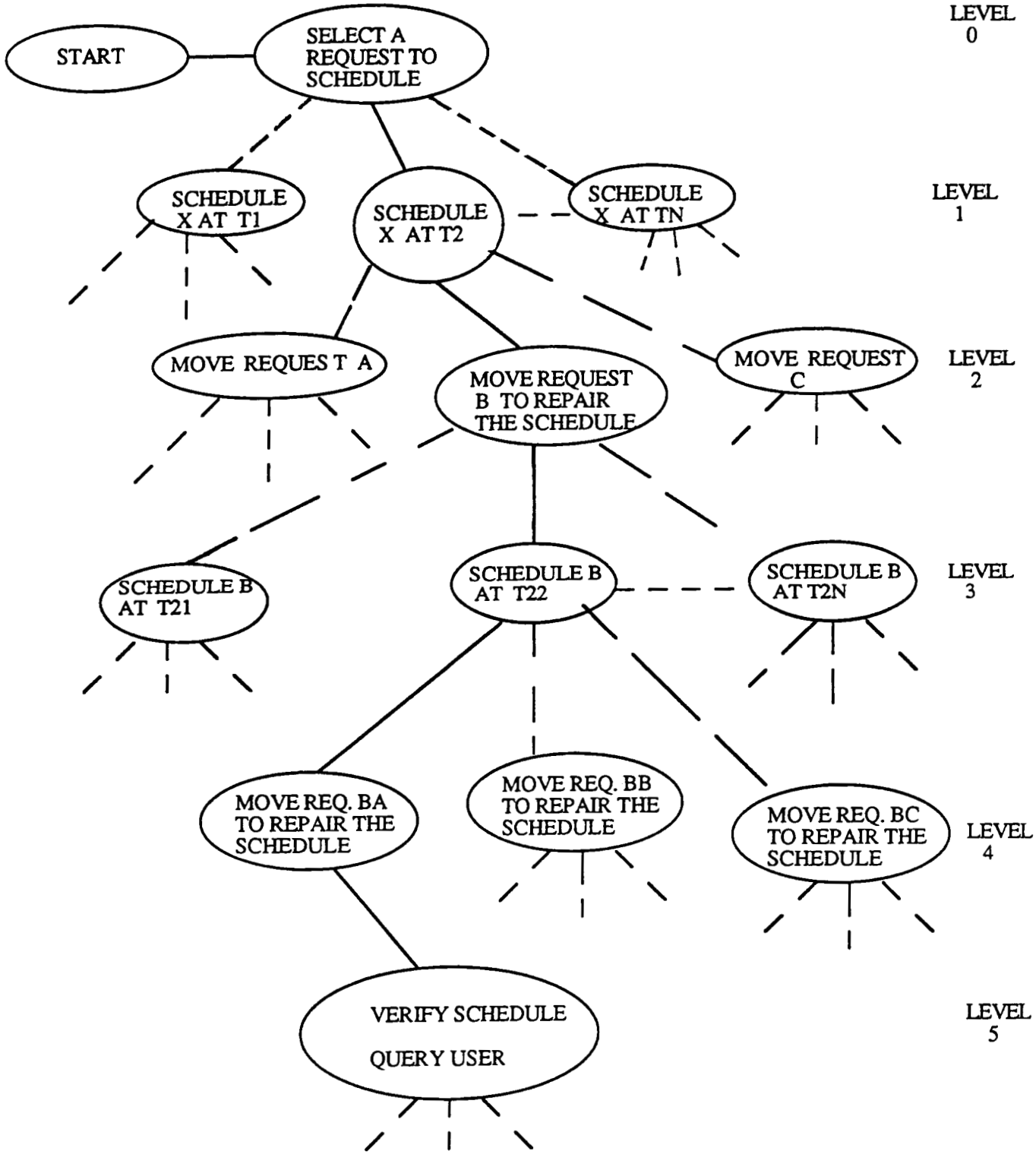

FIGURE 5. A SEARCH SPACE FOR RE-SCHEDULING IN ROSE

Step 5. Move an overlapped request within the window. Go to Step 2. Avoid generating any loops by not moving a request more than once.

Step 6. Determine the goodness of a schedule by tallying and evaluating the amount and number of overused resources and the number of dynamic constraints violated by the unscheduled requests

Step 7. Terminate the search at a pre-set time corresponding to the time it takes to reach a certain number of branching and/or depth factor, or until a solution is reached.

Step 8. Display the "best" solution and ask the user if he/she wishes to continue.

Step 9. If the user wishes to continue, attempt to schedule the remaining unscheduled requests.

After establishing the rules that guide acceptable procedures for rescheduling, we are ready to apply our algorithm. To apply this algorithm, we develop an evaluation function, f' which estimates the relative merit or cost of continuing a search from a given state after applying a rule. The evaluation function is a cost function which must be designed to estimate the remaining length of a path between a node n and the goal node. It is used to set up the order as to which nodes to consider during a search such that the goal is reached with the minimum number of steps.

Application of the A* Algorithm

The problem space consists of nodes (shown in Figure 5), and these nodes fall into two categories: OPEN and CLOSED. OPEN is a list of nodes containing the nodes to which the heuristic evaluation function have been applied, but for which their successor nodes have not been generated. The nodes in the list are sorted in a priority sequence such that the highest priority is assigned to the node for which the value returned by the heuristic evaluation function is most promising. The CLOSED list contains the nodes with non-promising values for the evaluation function.

Function f' has two components, a g component and an h' component.

f'(successor node) = g(successor node) + cost to new node
or f'(successor node) = g(successor node) + h'(successor node)

where

g(successor node) = g(best node) + h′(successor node)

and

a best node is defined as a node on OPEN list of nodes with the lowest f′.

The g component is defined as the measure of the cost of getting from the initial state to the current state. It is the sum of the costs of applying the evaluation function along the best path leading to the current node. Function h′ returns an estimate of the additional cost of getting to the goal node from the current node. Since h′ represents cost, low values for h′ lead to good nodes. Implementing the functions described above enables the re-scheduling functions to search and reach the goal by manipulating the list of nodes in the OPEN and CLOSED lists.

Since the only action taken at each step is to re-schedule an existing request, the cost of going from one node to its successor node (h′) is a constant. If different actions were taken at different nodes (for instance, relaxation and deletions), the h′ function will not be constant.

Another Approach to Speed Search During Re-scheduling

The AO* or the AND/OR graph can be used to represent search strategies by decomposing a problem into subproblems. This allows for the generation of alternative solutions to the problem. The initial problem corresponds to the root node of the graph. At an AND node, all the successor nodes must be solved to obtain a solution for that AND node. However, at an OR node, only one of the children nodes must be solved. It is not necessary to generate a solution for more than one node.

Applying this problem solving strategy to searching the search space in Figure 5, it means that in locations where more than one scheduled request must be moved, all the scheduled requests that need to be moved must be moved in parallel until the schedule in a local region becomes valid. This action requires more knowledge of multiple requests. With more knowledge of each requests moved, the amount of search required is reduced, and solution can be obtained at a smaller cost than with the A* algorithm.

Future Work in Automated Re-Scheduling for ROSE

In the future, we plan to explore the application of assumption-based or justification-based truth maintenance system concepts to

evaluate their effectiveness in helping to repair invalid schedules generated during re-scheduling. Also, due to the extensive amount of search required, and since we want to limit back-tracking while the scheduler is in search of a goal, some machine learning paradigms such as, learning from experience can speed the re-scheduling time. Also the effectiveness of the application of neural network algorithms in re-scheduling will be explored.

The effects automating other conflict resolution strategies, such as overbooking certain resources, acquiring additional resources from other schedulers in a distributed scheduling architecture will be employed.

CONCLUSION

Quoting Raj Reddy's [6], fourth and fifth rules of Artificial Intelligence, "Search compensates for lack of knowledge" and "Knowledge eliminates the need for search", these statements apply in many problem solving efforts, specifically when solving planning problems. The amount of search required in the heuristic described above can be reduced significantly with more knowledge of the constraints. With a better knowledge of the constraints, the AO* search heuristic can provide a faster solution and a shorter path search than the technique described in this paper. According to Mark Fox (7), scheduling is not yet a science, it is still an art. As a result efficient problem solving techniques must be explored to improve search and reduce re-scheduling time. This paper presents our attempt at improving the time required for automatic re-scheduling in the Space Station and the TDRSS Network Control Center environment. We employed a hybrid problem solving technique to reduce automated re-scheduling time. Given the knowledge of the problem space, the hybrid problem solving approach described here is efficient for re-scheduling.

Bibliography

1. Dean, Thomas, Planning Paradigms, Brown University, Department of Computer Science, Providence, R.I. 02912.

2. Rich, Elaine, Artificial Intelligence, The University of Texas at Austin, McGraw-Hill Book Company, N.Y., 1983.

3. Zoch, David and Hall, Gardiner, 1988, Integrated Resource Scheduling In A Distributed Environment, Ford Aerospace Corporation, 7375 Executive Place, Seabrook, MD 20706.

4. Tanimoto, Steven L. The Elements of Artificial Intelligence, Department of Computer Science, FR-35, University of Washington, Seattle Washington, Computer Science Press, 1987.

5. Dougherty, E. R., and Giardina, C. R., Mathematical Methods for Artificial Intelligence and Autonomous Systems, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1988.

6. Reddy, Raj, (1988), Foundations and Grand Challenges of Artificial Intelligence, AI Magazine, Winter 1988, pp. 9-21.

7. Fox, S. M., (1987), Constraint Directed Search: A Case Study of Job-Shop Scheduling, Morgan Kaufmann Publishers, Inc. Los Altos, California.

8. Minasi, Mark, (1989), A Final Look at Simple Search, AI Expert, February 1989, pp. 15-20.

9. GSFC Code 522, Software and Automated Systems Branch, User's Guide for the Flexible Envelope Request Notation (FERN), Schedule Integration Requirements Study, SEAS Task 29-1000, January 1989

10. GSFC Code 522, Software and Automated Systems Branch, User's Guide for the Request Oriented Scheduling Engine (ROSE), Integrated Resource Scheduling (IRS), SEAS Task 29-1000, January 1989

11. GSFC Code 522, Software and Automated Systems Branch, Schedule Integration Requirements Study, Final Report, SEAS Task 29-1000 January 1989.

Acknowledgements

We would like to thank Nancy Goodman, Larry Hull, Mike Tong and other staff members of NASA-GSFC Code 522 for their support in the development of ROSE.