JPL Publication 89-12

Strategies for Automatic Planning

A Collection of Ideas

Carol Collins
Julia George
Elaine Zamani

May 1, 1989

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL Publication 89-12

Strategies for Automatic Planning

A Collection of Ideas

Carol Collins
Julia George
Elaine Zamani

May 1, 1989

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

ABSTRACT

The main goal of the Jet Propulsion Laboratory (JPL) is to obtain science return from interplanetary probes. The uplink process is concerned with communicating commands to a spacecraft in order to achieve science objectives. There are two main parts to the development of the command file which is sent to a spacecraft. First, the activity planning process integrates the science requests for utilization of spacecraft time into a feasible sequence. Then the command generation process converts the sequence into a set of commands. We are concerned with the first of these, activity planning.

The development of a feasible sequence plan is an expensive and labor intensive process requiring many months of effort. In order to save time and manpower in the uplink process, automation of parts of this process is desired. There is an ongoing effort to develop automatic planning systems. This has met with some success, but has also been informative about the nature of this effort. It is now clear that innovative techniques and state-of-the-art technology will be required in order to produce a system which can provide automatic sequence planning.

As part of this effort to develop automatic planning systems, we have conducted a survey of the literature, looking for known techniques which may be applicable to our work. This report contains descriptions of and references for these methods, together with ideas for applying the techniques to automatic planning.

CONTENTS

Strategies for Automatic Planning
A Collection of Ideas*

I. INTRODUCTION

We want to provide an automated method for resolving conflicting resource usages by activities in a schedule. The main goals for automating the uplink system are speed, comprehensiveness, and reliability [MW]. The future capabilities that this report addresses are directed toward two of these goals: a fast spacecraft activity planning system which is comprehensive enough to include automated conflict resolutions.

Our objective is to provide an automated method for resolving resource contention in a schedule. There are two ways that such a capability may be used:

1. As a scheduler builds a timeline, many resource usage conflicts must be resolved. The automated capability can be invoked to provide suggestions for changes which would resolve the conflicts.

2. When the activity requests are first gathered, the automated conflict resolution capability may be invoked to provide an initial timeline which has many conflicts resolved.

These are closely related since the methods to resolve a single conflict can possibly be extended to build an initial low-conflict timeline for a scheduler; one method is to display every activity at its most desired starting time and resolve the conflicts.

An automated conflict resolution system will require the development and implementation of advanced programming strategies. In this report we present a collection of ideas which will lead to the development of complete strategies for automatic planning. First we discuss sequence planning.

*This research was supported by the Director's Discretionary Fund at the Jet Propulsion Laboratory.

II. SEQUENCE PLANNING

Planning a sequence of activities for a spacecraft is becoming a more complicated process as spacecraft become more complex [B,RN]. The planning process must take into account the capabilities of the spacecraft and the trajectory and Deep Space Network (DSN) information as well as the activity requests from the scientists and from the spacecraft engineers.

Each activity request contains the following information about an activity: the most desired starting time, intervals of acceptable starting times, the duration requirements, and the spacecraft resources that it uses. The resource usage descriptions specify the amount of power required, the data handling required, and the instruments required. Also included is an explanation of the manner in which the instrument is to be used; for example, the direction in which the camera should be pointing.

The complications arise from several sources [ZGCZ]. Resources such as power and data handling should not be oversubscribed. Interactivity dependencies must be met, and forbidden states of the spacecraft must not be entered. The resource requirements which stem from the description of a spacecraft and are needed to support an activity must be included; for example, if the scan platform needs to slew, the power for doing so is figured into the usage display at an appropriate time.

A sequence is considered complete when it is ready for command generation, the process of writing the actual spacecraft commands which achieve the activities in the determined sequence. In order to be considered complete, a sequence must have resolved resource conflicts; that is, the requests to the payload (instruments), power, attitude control, and data subsystems must be feasible. Also, rules involving resources from several systems must be obeyed.

III. STRATEGY IDEAS

In order to develop strategies, we have done research into areas which may apply to automatic planning. This section contains a list of tools and ideas which we have encountered. Some of the ideas are based on automatic planning systems in

other domains, such as factory scheduling. Some of the ideas are the result of our cogitations. Some ideas are well-formed and some are still fuzzy.

We plan to use these ideas and tools to develop complete strategies for automatic planning. These strategies will probably be combinations and expansions of the ideas listed here. The first six are tools and topics used in computer science, the next three are based on scheduling systems, and the last three are more general ideas.

A. Simulated Annealing

Human brain activity seems to use a process similar to simulated annealing. Since humans have much more success with planning than do computers, it is worthwhile to consider using brain-like actions in our scheduling strategy.

To anneal a metal means to heat it and then cool it in a controlled manner to induce particular characteristics. At high temperatures, molecules move more rapidly and traverse larger distances. As the temperature drops they settle down.

The brain works in a similar manner. Scans of brain activity show that a lot of randomness is involved in the firing of neurons. As a human thinks about something, the brain seems not only to collect ideas which are strongly related, but also to randomly shift to "far out" ideas. In this way, the person is able to conceive the entire idea without settling into a "local minimum" for the objective.

A computer science technique which mimics this behavior has found application in several areas, including robotics. Simulated annealing is a method of solving combinatorial optimization problems using techniques derived from statistical mechanics. This method is used for finding approximate minima for objective functions in problems with discrete but very large sets of possible solutions. To solve a problem using simulated annealing, the estimated solution configuration is allowed to change in ways that cause large random changes in the objective function initially and smaller random changes later on. [PFTV], [KGV]

Combinatorial optimization refers to a set of problems that require finding minimum or maximum values of a function of very many independent variables.

Typically, solutions are problem-specific, and are either a divide-and-conquer method or an iterative method. Standard iterative methods sometimes result in local optima which are not global.

Because simulated annealing is not a greedy method, local optima are more easily avoided. Another advantage to simulated annealing is that decisions proceed in a logical order. For example, in the traveling salesman example the authors in [KGV] inserted a fictitious river in their map and added to the objective function a penalty for crossing the river. Early in the solution, while the temperature was still fairly high, the decisions about when to cross the river were made. Later the particular paths on each side of the river were finalized.

To use simulated annealing, the problem needs the following:

1. A description of all possible solution configurations;
2. A generator of random changes in a given configuration;
3. An objective function to minimize; and
4. An analog of temperature -- a value which decreases over time, restricting the allowed changes as it decreases.

We shall now describe one strategy for automatic planning which uses simulated annealing. The description addresses the four requirements listed above:

1. The problem is to find a schedule which minimizes resource conflicts. Another part of the goal may be to maximize the scheduling of high priority activities. The possible solution configurations are the possible schedules. A schedule can be described as a list of numbers, where each number is the starting time of an activity, and the order of the list corresponds to some ordering of the activities. Each number must lie within an acceptable time interval for that schedule. The numbers may be further constrained to lie within acceptable time intervals for their respective activities. A dummy value for a time may be used for indicating an unscheduled activity.

2. A configuration may be changed by moving one or more activities; that is, increasing or decreasing one or more of the start-time numbers. Random selection may be made of the number of start times to change, the distance each is changed, and the direction that each is changed.

3. The objective function should encompass both the amount of resource conflict and the goodness of the schedule for each of the high priority items. If activities are allowed to be unscheduled, we want to minimize the sum of the priorities of the unscheduled activities.

4. The control parameter, called the temperature, can be used in three ways: 1) The number of start times to change should lie between 1 and (constant$_1$ x temperature). 2) The distance each activity can change is restricted to lie between 1 and (constant$_2$ x temperature). 3) An altered schedule replaces the current answer schedule if its objective value is smaller. On occasion, replacement occurs if the objective value is larger, but not by more than the temperature. The probability function used to determine when to take the larger value is

$$P = exp\ (\ -\ (amount\ of\ change)\ /\ (temperature))$$

That is, the larger value is taken with probability P.

There is a natural extension of simulated annealing to parallel processing. Each node performs the annealing independently. Periodically, the results are pooled: each node broadcasts its best solution so far to node 0. Node 0 finds the best of these solutions. If it is good enough the program terminates. Otherwise, node 0 broadcasts this solution to all nodes and the simulated annealing is continued.

B. Parallelism

The main advantage of parallel computation over sequential computation is speed. Known approaches to automatic planning are exponential; the amount of time required increases exponentially as the number of activities to schedule is increased. Parallel computation can provide, at best, a linear speedup. Even so, a linear speedup can mean the difference between an acceptable system and an unacceptable one.

We shall discuss one method of using parallel computers in the section on neural networks. In this section we turn our attention to computers, such as the hypercube, which have more sophisticated processors connected together.

1. Topical Parallelism. One consideration other than speed is to use parallel computers to provide a cognitive breakdown of the processing. A planner which reasons on a model of the spacecraft may work well with each of the subsystems on a separate node. Then, when an activity is moved, the subsystems can check for resource conflicts in parallel. Categories of items which can live on separate nodes are: power, attitude control, data, payload, sequence, and flight rules.

For example, on four nodes, the host (node 0) will be responsible for the timeline display and the message passing. Then node 1 can hold the sequence, node 2 the data-handling model, and node 3 the rest of the spacecraft models.

2. Algorithmic Parallelism. In order to achieve a speedup in processing time, a parallel processor needs to have "balanced loads" so that each processor has about the same amount of work as all other processors. If n processors are equally busy on a task, then the job may be completed in about 1/n of the time that it would take a single processor.

Better load balancing may result from parallelizing the individual planning algorithms instead of assigning subsystems to nodes.

3. A Combination. In order to achieve the speedup of algorithmic parallelism and the cognitive breakdown provided by topical parallelism, it may be desirable to design a system which includes both. For example, if 16 nodes are used, then each subsystem can reside on a 4-node subcube. The algorithmic parallelism would occur within each of these subcubes.

The code for each subsystem can be developed on a smaller cube. Then these programs can be integrated for running a planner on a larger hypercube.

C. Neural Networks

The concept of a neural network is based on models of how neurons in the brain work. The reason that neural networks have received a lot of attention in the computer science community lately is the hope that by modeling the chemical interactions of neurons in the brain we can achieve some of the qualities of human thought in our

computing systems. The three main qualities expected are speed, fault tolerance, and learning. [L]

We are interested in possible applications of neural networks to sequencing for two of these reasons: speed and learning. One application area of neural networks is in solving NP-complete problems such as the traveling salesman problem. Automatic planning is an NP-hard problem, so it is possible that neural networks can provide a form of parallelism which will achieve good performance in sequencing as well. Two other application areas which involve computer learning are pattern recognition and cluster analysis.

Here are some details about how a neural net is configured. A neural network is a computing system which consists of a number of interconnected simple processing elements. Each element receives input from nodes to which it is connected. Based upon that input, the element decides what output to sent to its neighbors. Often the output is either a 0 or a 1. The connections, also called edges, are assigned numbers called weights. When one node produces an output, it is multiplied by the weight of the edge to produce the input to the neighboring node.

To design a net, three decisions must be made. First an algorithm must be chosen for the nodes. This algorithm will apply a function to the inputs to determine the output. Second, the connections must be determined: what nodes are connected and what edge weights are assigned. Third, the training or learning rules must be selected.

The traditional method of teaching a neural network is called *back propagation.* The net is given sets of stimuli together with expected results, and the edge weights are adjusted to accommodate the training set. A breakthrough would be the design of a neural network which can be trained on the sequences that have been generated by humans in the past, resulting in a capability of automatically generating initial timelines which are close to the desired final product.

Another idea for applying neural networks to automatic planning is to combine networks with the object-oriented approach in PLANNER, the JPL Space Flight Operations Center (SFOC) planning software system. Each object is represented by a

neuron. Activities fire when involved in a resource conflict, stimulating a chain of computations which result in the resolution of that conflict.

D. Operations Research

There are three areas of operations research [HL] which may have applications to automatic scheduling. They are linear and nonlinear programming, project planning, and decision analysis.

1. Linear and Nonlinear Programming. Linear and nonlinear programming are subject areas concerned with allocating limited resources among competing activities in an optimal way. An objective function is expressed in terms of decision variables. The objective function is optimized subject to resource constraints. If the expressions involved are linear, it is a linear programming problem; otherwise, it is considered a nonlinear programming problem. Common methods for solving nonlinear programming problems are applicable only when the functions involved are differentiable.

For automatic scheduling, the decision variables are the starting times for the activities. The objective function is the function of these starting times which gives us an idea of how good a schedule is. If only some of the activities can be scheduled, the function may include a portion which takes into account the priorities of the unscheduled activities. If all activities are scheduled and conflicts are allowed to occur, then the objective function should be a measure of the total amount of conflict in the schedule. It may also be desirable to figure in the priorities of the activities involved in a conflict.

Let us consider the objective function based solely on the amount of conflict in the schedule. As activity start times change slightly, the total area of conflict depicted on a PLANNER screen also changes only slightly. Indeed, this area is a continuous function of the start times. This function can be approximated by a differentiable function if we allow small amounts of conflict between competing activities which are scheduled near each other but do not in fact overlap. Such conflict additions can be justified by multiplying the effects of minor clock synchronization errors by the probabilities that such errors are present. Standard techniques of differential geometry and real analysis may be used in writing such functions.

Besides differentiability of functions, another hurdle in fitting scheduling to an efficient nonlinear programming paradigm is that the number of variables is the number of activities to be scheduled. It may be possible to reduce the number of variables by reducing the amount of timeline considered at any one time. Then, after applying the techniques of nonlinear programming to each part, it will be necessary to patch the pieces together to form a complete timeline. The patching will have to include resolution of conflicts at the borders and conflicts introduced in the patching process.

2. Project Planning. In planning for large projects, first a network of activities is built which shows precedence. Each activity is represented by a node. For each node the time required to perform that activity is noted. The earliest start time and the latest start time are calculated. Earliest start time means the earliest time when an event can start if all preceding activities are started as early as possible; latest start time means the last time at which an event can occur without delaying completion of the project. The critical path is the path through the network using the nodes with 0 slack, where slack is the difference between earliest and latest start times.

The techniques for project planning may be useful in dealing with time-interdependent steps. However, it does not allow for absolute time dependence of activities. Applied to sequence planning, the concepts of earliest and latest start times may be changed to encompass freedom from resource conflicts.

3. Decision Analysis. Decision analysis provides a methodology for decision making in the face of uncertainty. Decision making with experimentation invokes statistical methods based upon gathered data to make decisions which are optimal in some sense.

It may be possible to gather information about previous mission schedules and use that information when creating an initial schedule. Information can be gathered from previous schedules which correlates the actual scheduling of an activity request with characteristics of the request, such as priority, duration, and resource utilization. The statistics can then be applied to creating a new schedule from a new set of activity requests by looking at the characteristics of each request.

This strategy for scheduling would probably work best in combination with other strategies. The goal is to automatically create a fairly good initial schedule which would then be available to a human scheduler for fine tuning.

E. NP-Complete Problem Solution Techniques

The problem of finding an initial conflict-free schedule given a set of activities is NP-hard, so it is worth considering the techniques of solving similar NP-complete problems. Our problem is closest to two NP-complete problems in the literature [GJ]: scheduling with release times and resource-constrained scheduling. Sequence planning is also related to the NP-complete problem of bin packing.

1. Scheduling with Release Times. In this type of problem, we are given a set of tasks, and for each of the tasks we are given a duration, a release time, and a deadline. The goal is to determine if there is a schedule for the tasks that satisfies the release time constraints and meets all the deadlines. This problem is NP-complete even if the release times and the deadlines can each take on only 2 values. It can be solved in polynomial time if all task durations are 1, or preemptions are allowed, or all release times are 0. This polynomial time remains possible even when precedence constraints are included.

2. Resource-Constrained Scheduling. In this type of problem, we are given a set of tasks each of duration 1, a number of processors, a number of resources, resource bounds for each resource, resource requirements for each task and resource, and an overall deadline. The goal is to determine if there is a schedule for the tasks on the processors that meets the deadline and obeys the resource constraints at all times. This problem is NP-complete even with one resource and only 3 processors. For 2 processors and any number of resources, it can be solved in polynomial time by matching. If the tasks are partially ordered and the partial order is a forest, the problem is NP-complete for one resource and 2 processors.

3. Bin Packing. The traditional bin-packing problem is stated as follows: Given a finite set of packages and a size for each package, partition the set into disjoint subsets such that the sum of the sizes of the package in each subset is no more than a fixed amount, and the number of subsets is minimized. In multidimensional bin packing, the package sizes have several dimensions.

To apply bin packing to sequencing, we let each activity represent a package and each resource represent a dimension. The measurement of a package in a given dimension is the amount of the resource that activity requires. Time is a special dimension which must be treated slightly differently. However, standard bin-packing algorithms may apply to give us automatic planning.

Two methods for implementing bin packing on a parallel processor are:

1. First-fit decreasing (FFD) on a queue. Sort the packages by size, then put the entire list on node 0. Pass packages along the queue route, passing a dummy if no package is there. Each node starts with one bin. Node 0 takes the first package and tries to fit it in. If it fits, node 0 passes a dummy and takes the next package. When a package does not fit in any bin, then each node gets a new bin.

2. Sort the packages and give node i packages 4n + i. Perform a FFD packing on each node. Then take the least full bin from each node and reconsider those packages. Send these packages to other nodes to see what fits into other packed bins, or else send the leftovers to node 0 and pack them with an FFD algorithm there.

F. Persistence

Ordinarily, a linked data structure changes over time as elements are inserted and deleted. Such a structure is said to be ephemeral since making a change destroys the old information and replaces it with the new. A *persistent* or *partially persistent* data structure [ST],[C] is one which allows access to all previous versions of the structure as well as to the current version. Updates (insertions and deletions) are allowed only in the current version, but all past versions can be queried. A data structure is said to be *fully persistent* if every version can be both accessed and modified. [DSST]

Persistence is useful in maintaining a data structure which changes over time with two characteristics:

1. Changes do not need to be presented in time order; and
2. All versions of the structure are accessible.

The act of scheduling an activity causes changes in a plan. As a schedule is being prepared, the changes to the plan are not presented in time order. Also, information about the schedule at particular times must be accessible. For these reasons, a persistent data structure may be useful in a planning system.

1. Partial Persistence. The brute force way to make a search tree persistent is to recopy the entire structure for each update. This gives us a relatively good search time, but requires a vast amount of storage space.

At the other extreme, we may achieve persistence by maintaining only one node for each key. Each time we want to change a pointer, we add another field to our "fat" nodes. The new field contains the new pointer and a time stamp indicating when the change occurred.

The best method of persistence in a search tree is called limited node copying. It gives fast access and update times and requires a small amount of space. In the limited node-copying method, we allow each node to contain a small fixed number k of additional pointers, say 1 or 2. We copy a node only if its pointer fields are filled and a new pointer needs to be added.

2. Full Persistence. There are two methods for making a linked structure fully persistent: node splitting and displaced storage of changes. These methods are presented in [DSST] in sections 3 and 5, respectively.

Node splitting is a variant of limited node copying. Each node contains up to a fixed number of records, each with a version stamp. When a node has a new version of pointers to store and no records left to store in, it splits into two nodes, placing half of the records in one node and half in the other. Inverse pointers must be maintained so that parent nodes will point to the correct result of the split. Node splitting is efficient to use when the in-degree of each node is bounded.

In the node splitting method, it may be the case that one node will have a large number of change records while other nodes have very few. In order to use the record space wisely, the displaced storage method allows change records to be maintained by ancestors to the changed node. By using this method with appropriate algorithms

for copying nodes and for maintaining the change record displacement paths, it is possible to guarantee an O(1) worst-case space bound per update step and an O(1) worst-case time bound per access step.

G. Unit Boundaries

Two of the sequence planners developed in the JPL Sequence Automation Research Group (SARG) are Plan-It and Plan-It II [DHW], [EG]. Both provide a user interface for manual sequence generation using a computer, as well as some automated conflict resolution capabilities. This section contains descriptions of the automatic planning strategies developed for Plan-It and Plan-It II. These strategies have already been implemented in sequence-planning systems and may be further developed in combination with other strategy ideas in this report.

Plan-It provides capabilities for automated conflict resolution aids to the human scheduler. Given an activity sequence, each resource creates a list of change times (unit boundaries) which includes the starting times and ending times of the activities as well as the times at which the resource amounts available change. An activity list is associated with each unit, or period between adjacent change times. For each unit the resource calculates the amount available, the amount used by activities during that interval, and the amount of conflict. Each resource then calculates statistics by summing values for each unit, including the sum of the amounts used, the sum of the amounts of conflict, the sum of the squares of the amounts used, and the sum of the square of the amounts of conflict. The square sums (deviations) are calculated to indicate how evenly distributed the usage and conflicts are. The goodness of a sequence is determined by a function of the statistics for the resources.

If the scheduler decides to move a particular event (activity) in order to resolve a conflict, Plan-It can offer suggestions for better placements for that event. The method used involves studying the resource line to find a better place. The "best so far" starting time for the event and corresponding goodness are initialized to the present values. The objective time windows to look in are determined by taking into account any user-imposed "move it here" intervals, the windows specified in the event request, and dependency separation requirements from other activities. Within the objective time windows, a list is made of the times of unit boundaries for the resources that the

activity uses. Plan-It then considers each time on the list as a possible starting time for the activity and chooses the time which optimizes the goodness of the timeline.

Plan-It II uses the same concept of units and unit boundaries. However, one problem with using the goodness function is that it can produce timelines in which there are areas where one resource has a great deal of conflict and other resources have minimal conflict. To avoid this, Plan-It II uses a different scheme for determining suggestions for new times.

Each resource reports the times at which it has amounts available, and the activity which has been selected to move finds times at which the resources can best fill its requirements. The reports made by the resources consist of a list of the unit boundary times and the free amount of the resource within each unit, where the free amount is calculated by subtracting the current usage and the usage of the selected activity from the amount of the resource available during that unit. These lists are simplified by such means as removing all but the first unit boundary time of a set of contiguous times in which the free amounts are equal.

The selected activity then examines the reports from the resources to find better places in the timeline. First the reports are combined by forming a master list of all the unit boundary times on the individual lists and finding the average of the values in each of the new intervals. The resulting list is simplified, and the unit boundary times of the entries with the highest numbers are the times sent to the event for final selection. This selection depends upon several factors, including the dependency constraints on the activity and its delay-separation requirements.

H. Profiles

The Resource ALlocation and Planning Helper (RALPH) system was developed to assist in planning the use of the resources of JPL's Deep Space Network [JW]. It contains an allocation subsystem which creates a distribution of resources to users, supplying feasible support to each user. The input to the allocation subsystem includes event requests and information about resources.

The allocator is a two-pass process; that is, it passes through the list of event requests twice. On the first pass it builds a profile of assignment likelihoods for each

resource. For example, if an activity of duration 8 can use a particular resource in any one of 4 non-overlapping time periods, each of length 9, then the probability it is scheduled at any one of the included times is 8/36. In the first pass, inflexible, event-independent constraints are taken into account, but flexible constraints and event priorities are not considered.

In the second pass, the requests are processed in order of priority. Resource assignments are made using the resource likelihood profiles developed in the first pass. As resources are allocated, the likelihood profiles are adjusted to reflect the times that events have been scheduled. During this pass, inflexible constraints are honored and flexible constraints are used to provide biases against constraint violations.

When an activity is chosen to be scheduled, the following process is used. The profile of each unscheduled activity is weighted by the relative priority of the activity with the chosen activity. The usage profile of each scheduled activity is weighted heavily. The timeline is searched for a place to schedule the chosen activity which minimizes the total weight of the profiles of the other activities over the scheduled time period.

The result of the automatic scheduling in the RALPH system is a plan which has a relatively low amount of resource conflict. This plan then provides a starting point for a conflict resolution meeting.

Our goals for automatic planning are also to provide a schedule with a low amount of resource contention. The final decisions for conflict resolution will still lie with the human scheduler and with the scientists in conflict resolution meetings. The processes used in RALPH's allocation subsystem allow for the creation of final products in which conflicts exist, but are minimized.

I. One Pass Resolve

The One Pass Resolve is a method for providing automatic planning to supply a human scheduler with an initial timeline in which many conflicts are resolved. This strategy is based on the OPIS [OST] approach for factory planning.

Recall that each activity description includes a most desired starting time and intervals of acceptable starting times. The One Pass Resolve begins by scheduling each activity at its earliest acceptable starting time. Then we pass through the list of activities, considering each in time order, to resolve as many conflicts as possible. Resolutions are created only by moving activities to later times. Since conflicts are resolved in time order, parts of a timeline before a conflict are never changed in resolving that conflict. After the resolution pass, one or two clean-up passes may be performed in which it is determined whether or not each activity can be moved to its desired time without increasing the conflicts.

The OPIS approach for factory planning [OST] uses a conflict monitor and a resolution formulator. The sequence is fed into a conflict monitor, which produces a time-ordered list of time intervals with conflicts. These are fed one at a time in order into a resolution formulator, which selects a resolution action from a fixed set. The selected action (with appropriate parameter values) is applied to the sequence, the conflict monitor updates the conflict list, and then the next conflict is fed to the resolution formulator.

Further development of this strategy requires devising specific algorithms to be used by the conflict monitor and the resolution formulator. Because of the differences between factory scheduling and spacecraft scheduling, only some of the resolution actions available in OPIS may be applicable to our planner; others more natural to our problem domain will need to be developed.

This strategy would be straightforward to implement and would probably give a human scheduler a good head-start on creating a sequence. Since the conflict resolutions take place in time order, early parts of a sequence will be available for human schedulers fairly quickly, with later time periods becoming available subsequently.

J. Narrow Windows

In this strategy, we use narrow activity windows and a coarse granularity of time units. All times are rounded up to the nearest unit.

Each activity has a duration given as an integer number of units of time. Recall that each activity has a preferred starting time and intervals of acceptable starting times. Assign each activity a narrow starting-time interval which lies within an acceptable starting-time interval and is centered around the preferred starting time. Then the *narrow activity window* for that activity is the interval beginning at the earliest time in the narrow starting-time interval and ending at the time at which the activity would end if it started at the last time in the narrow starting-time interval.

For each time slot on the timeline, make a list of the activities which include that time slot in their narrow activity windows. For each possible subset of these activities, determine the resource usage and conflict totals. Using this information, it should then be possible to construct an initial schedule in a reasonable amount of time. Further development of this strategy requires a determination of how the initial schedule is constructed once the narrow activity windows data is calculated.

If there are n time slots and about 5 activities in each, then there are about 32n subsets for which the resource usage and conflicts must be calculated. These calculations can easily be done in parallel.

Another extension of this strategy is a combination with the ideas in the profiles strategy. That is for each activity on the list, include a numerical rating of how much the activity needs the slot, as in the profiles strategy. Use this additional information in determining the initial schedule.

K. Foveation

Suppose that two activities are in conflict for resources at their currently scheduled times, and that one of these activities has been chosen by the scheduler to be moved.

If the human scheduler looks for another place to schedule that activity, a display may be used which shows profiles of resource usage for the current schedule. The information which comes to the cognition from the eyes is not evenly distributed across the entire time span of the sequence. Instead, there is a limited field of vision which encompasses a portion of the timeline. Central to this field of vision is the focal point.

The eye provides a great deal of information about the areas close to the focal point and sparser information about areas closer to the periphery. [VA]

Based on this information, the human may select a candidate position for the activity. Then the eyes move so that the candidate position becomes the focal point. This process may iterate several times as the scheduler narrows in on a position which is finally selected for the move.

The foveation strategy for automatic planning may be used to provide automatic relocation of an activity which is in conflict for resource usage. In deciding on a new location, using complete knowledge about all of the resource usages for the entire timeline at once would be too slow. Instead, we use information samples which are gathered in a time interval, called a field of vision, about the activity. The samples are most dense in the immediate area of the conflict and sparser toward the endpoints of the interval, called the periphery. Using information samples instead of complete information may provide a large speedup in time, since accurate resource usage profiles will not have to be built.

Based on the information samples, a relatively low-use area is selected to be the new focal point. Another set of information samples is collected about this focal point. If the new information indicates a high probability that the activity can be moved here, then the detailed calculations can be made. Otherwise a new focal point is selected and the process is repeated.

There are two direct extensions to this strategy. One is to combine it with simulated annealing to more directly mimic the way the mind works. A human scheduler will attempt to avoid reaching dead ends by turning to other random focal points at times. This suggests a usefulness in combining foveation and simulated annealing.

The other extension is to use a multiprocessor as a team of schedulers in the sense that each processor is an eye. The multiple eyes can be used to provide an n-fold increase in possible new locations studied if each processor uses a slightly different scheme for determining a new focal point. Alternatively, if several activities are in conflict, each processor can search for a new place for a different one of the

activities. The results can be selected or combined to find the best resolution given the quantitative schedule evaluation criterion.

L. Subsumption Architecture

Rodney Brooks at the Massachusetts Institute of Technology (MIT) has developed a layered control system for a mobile robot which allows independent isolated agents to cooperate to achieve a specific goal. Here is a description in his own words [B,RA]:

> "Layers of control system are built to let the robot operate at increasing levels of competence. Layers are made up of asynchronous modules that communicate over low-bandwidth channels. Each module is an instance of a fairly simple computational machine. Higher-level layers can subsume the roles of lower levels by suppressing their outputs. However, lower levels continue to function as higher levels are added. The result is a robust and flexible robot control system."

The way the layers are built for controlling the robot indicates a method for breaking down a problem which may have applications outside of robotics. For the mobile robot, the lowest level of capability is to avoid objects. The next level is to wander about. There is a switch between the capability of wandering about and the ability to avoid objects, so that as the robot wanders it will still avoid objects. However, when there are no objects nearby, the robot will wander. The capabilities increase through additional levels of capability to a top level which causes the robot to reason about the behavior of objects in the environment and interact with the environment to influence it in a productive manner. [BC]

This type of system allows the robot to handle multiple goals, even when some are conflicting, without forgetting to do simple things such as maintaining its balance. It also allows a straightforward method of integrating input from multiple sources. The system is robust enough to cope with a loss of part of its functions with a graceful degradation of functionality. Another advantage to this type of system is the extensibility of the design -- new capabilities can be added on top of existing ones without destroying the old ones. As the need arises for further capabilities, the existing design does not have to be revised, only extended.

To follow a similar strategy for developing an automatic planning system, we need to specify levels of capability that we would expect our system to encompass. The second step is to design an architecture that would allow each level to achieve its purpose in cooperation with the other levels. For automatic planning, the modules may be in a tree structure rather than a linear level structure. The specification of capabilities and the design of the module structure are matters open for further investigation.

IV. CONCLUSION

In this report, we have presented a collection of ideas for developing strategies for automatic planning. The next step is to combine and evolve these ideas into several complete automatic planning strategies.

In addition to the research for this publication, we have written a few computer programs to further our understanding of the strategy tools and to help determine what ideas will be straightforward to implement as parts of complete strategies. These programs include a simulated annealing program which solves a simplified planning problem and a first fit bin-packing algorithm on the hypercube.

Next we plan to develop complete strategies for automatic planning and determine their feasibility. To make this determination, we shall write computer programs implementing the strategies. We plan to report on the completed strategies and on the results of the implementation feasibility studies.

Acknowledgments

The authors thank the following people for discussions and talks about their work: Curt Eggemeyer and Jennifer Cruz, Plan-It and Plan-It II; Peng Si Ow, OPIS; David Werntz, RALPH; Charles Anderson, foveation; Rodney Brooks, subsumption architecture; and Barbara Zimmerman, the hypercube. The authors also acknowledge the contributions of D. Friesen of the Computer Science Department at Texas A&M University and T. Starbird of the Mission Profile and Sequencing Section of JPL.

Other members of the Sequence Automation Research Group include W. Dias, M. Hollander, W. Lombard, D. Mittman, S. Peters, M. Rokey, J. Sisino, and our leader Sven Grenander.

REFERENCES

[B,RA] Rodney A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation,* Vol. RA-2, No. 1, March 1986.

[BC] Rodney A. Brooks and Jonathan H. Connell, "Asynchronous Distributed Control System for a Mobile Robot," *SPIE Proceedings,* Vol. 727, *Mobile Robots,* 1986.

[B,RN] Robert N. Brooks, Jr., "The Evolution of the Voyager Mission Sequence Software and Trends for Future Mission Sequence Software Systems," AIAA 26th Aerospace Sciences Meeting, Reno, Nevada, January 11-14, 1988.

[C] C. Collins, **Persistent Search Trees and Maxima Finding**, Masters Thesis, Department of Computer Science, Texas A&M University, College Station, Texas, August 1987.

[DHW] W. Dias, J. Hendricks, J. Wong, "Plan-It: Scheduling Assistant for Solar System Exploration," *Telematics and Informatics,* Vol. 4, No. 4, pp. 275-287, 1987.

[DSST] J. Driscoll, N. Sarnak, D. Sleator, and R. Tarjan, "Making Data Structures Persistent," *Proc. 18th annual ACM Symposium on Theory of Computing,* pp. 109-121, 1986.

[EG] W. C. Eggemeyer and S. U. Grenander, "Plan-It Applications and Knowledge Gained," Workshop on Operations Planning and Scheduling Systems for the Space Station Era, University of Colorado-Boulder, Boulder, Colorado, August 1987.

[GJ] Michael R. Garey and David S. Johnson, **Computers and Intractability: A Guide to the Theory of NP-Completeness**, W. H. Freeman and Company, New York, New York, 1979.

[HL] Hillier and Lieberman, **Introduction to Operations Research, Fourth Edition**, Holden-Day, Inc., Oakland, California, 1986.

[JW] Craig D. Johnson and David G. Werntz, "A New Methodology for Resource Allocation and Planning" presented at the Space Conference in Houston, Texas, November 17, 1987.

[KGV] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing," *Science,* Vol. 220, No. 4598, May 13, 1983.

[L] Richard P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine,* vol. 4, No. 2, April 1987.

[MW] William I. McLaughlin and Donna M. Wolff, "Automating the Uplink Process for Planetary Missions," AIAA '89, Reno, Nevada, January 9-12, 1989.

[OST] Peng Si Ow, Stephen F. Smith, and Alfred Thiriey, "Reactive Plan Revision," *The Proceedings of the National Conference on Artificial Intelligence,* Minneapolis, Minnesota, August 1988.

[PFTV] William Press, Brian Flannery, Saul Teukolsky, and William Vetterling, **Numerical Recipes: The Art of Scientific Computing**, section 10.9, Cambridge University Press, Cambridge, New York, 1986.

[ST] N. Sarnak and R. Tarjan, "Planar point location using Persistent Search Trees," *Comm. ACM 29,* pp. 669-679, 1986.

[VA] David C. Van Essen and Charles H. Anderson, "Information Processing Strategies and Pathways in the Primate Retina and Visual Cortex," in **Introduction to Neural and Electronic Networks**, edited by S. F. Zornetzer, J. L. Davis, and C. Lau, Academic Press, Orlando, Florida, to appear.

[ZGCZ] E. B. Zamani, J. R. George, C. E. Collins, and B. A. Zimmerman, *Spacecraft Activity Planning in a Multiprocessor Environment,* to appear.