SRI International

PARALLELISM IN PLANNING AND PROBLEM SOLVING: REASONING ABOUT RESOURCES

Technical Note 258

SRI Project 8871

January 5, 1982

By: David E. Wilkins, Computer Scientist
Artificial Intelligence Center
Computer Science and Technology Division

The research reported here is supported by the Air Force Office of Scientific Research, Contract F49620-79-C-0188

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

ABSTRACT

The implications of allowing parallel actions in a plan or problem solution are discussed. The planning system should take advantage of helpful interactions between parallel branches, must detect harmful interactions, and, if possible, remedy them. This paper describes what is involved in this and presents some new techniques that are implemented in an actual planning system and are useful in seeking solutions to these problems. The most important of these techniques, reasoning about resources, is emphasized and explained.

1. Introduction

This paper discusses problems faced by an automatic planning system concerned with detecting and responding to interactions in parallel branches of a plan. (This could also be thought of as a problem solving system producing problem solutions.) There are three aspects to this situation: recognizing interactions between branches; correcting harmful interactions that keep the plan from accomplishing its overall goal; taking advantage of helpful effects on parallel branches so as not to produce inefficient plans.

We shall consider a domain-independent planning system that provides some formalism for representing the domain in which the planning will be done (as well as the goals to be achieved by the planner). The system also allows for the representation of operators. These are the system's representation of actions that may be performed in the domain or, in the case of hierarchical planners, abstractions of actions that can be performed in the domain. A plan consists of partially ordered goals and actions produced by applying operators to the initial goal. Actions in the plan explicitly list their effects, i.e., they specifically state which relationships in the world model change their truth value after the action is performed. (This is part of what Waldinger [6] has called the STRIPS assumption.) NOAH [3] and NONLIN [5] are examples of such planners.

In this discussion, parallelism is considered beneficial. (Two segments of a plan are in parallel if the partial ordering of the plan does not specify that one segment must be done before the other.) In much automatic programming research the goal is to produce a sequential program, so it is often easier to completely order the plan (program) rather than reason about parallel interactions. However, in multieffector environments parallelism is preferable. For example, if there are two robot arms, plans that use them in parallel to accomplish the goal are often preferred to a sequential plan that could get by with only one arm (even though it might take twice as long to accomplish). Our approach, therefore, is to keep as much parallelism as possible and to reason about the interactions that result from it.

2. Parallel Interactions

The canonical simple problem for thinking about parallel interactions in this context is the three-blocks problem. Blocks A, B, and C are on a table or on one another (only one block may be on another at any one time). The goal is to achieve (ON A B) in conjunction with (ON B C), thus making a three-block tower. (Initially the two goals are represented as being in parallel.) To move the blocks there is a PUTON operator (expressed in the formalism of whatever planning system we wish to talk about) that puts OBJECT1 on OBJECT2. Its definition specifies the goals of making both OBJECT1 and OBJECT2 clear before performing a primitive move action. (The table is assumed always to be clear and a block is clear only when no block is on top of it.) This problem will be used below to provide examples of interactions.

If two branches of a plan are in parallel, an interaction is defined to occur when a goal that is trying to be achieved in one branch (at any level in the hierarchy) is made either true or false by an action in the other branch. Since the actions in a plan explicitly list their effects, it is always possible to recognize such interactions. (In a hierarchical planner, however, they may not appear until lower levels of the hierarchy of both branches have been planned.) By requiring that a goal be involved in the interaction, we attempt to eliminate interactions that we do not care about. For this to succeed, the domain must be encoded so that all important relationships are represented as goals at some level. This will be discussed later.

The planner can possibly take advantage of a situation in which a goal in one branch is made true in another branch (a helpful interaction). Suppose we solve the three-blocks problem, starting with A and C on the table and B on A. In solving the (ON B C) parallel branch, the planner will plan to move B onto C, thus making A clear and C not clear. Now, while an attempt is made to move A onto B in the (ON A B) branch, the goal of making A clear becomes part of the plan. Since A is not clear in the initial state, the planner may decide to make it true by moving B from A to the table (after which it will move A onto B). In this case it would be better to recognize the helpful effect of making A clear, which happens in the parallel branch. Then the planner could decide to do (ON B C) first, in which case both A and B become clear and the (ON A B) goal is

easily accomplished later.

The planner must decide whether or not to add more ordering constraints to the plan to take advantage of such effects on a parallel branch. Ordering the parallel branches sequentially is the best solution to this problem because (ON B C) must be done first in any case, but in other problems an ordering suggested to take advantage of helpful effects may be the wrong thing to do from the standpoint of eventually achieving the overall goal. In general, the planner cannot make such an ordering decision without error unless it completely investigates all the consequences of such a decision. Since this is not always practical or desirable, planning systems use heuristics to make such decisions.

If an interaction is detected that makes a goal false in a parallel branch, there is a problematic (i.e., possibly harmful) interaction which may mean that the plan is not a valid solution. For example, suppose the planner does not recognize the helpful interaction in our problem and proceeds to plan to put B on the table and A on B in the (ON A B) branch. The plan is no longer a valid solution (if it is assumed that one of the two parallel branches will be executed before the other). The planner must recognize this by detecting the problematic interaction. Namely, the goal of having B clear in the (ON B C) branch is made false in the (ON A B) branch when A is put onto B. The planner must then decide how to rectify this situation.

As with helpful interactions, there is no easy way to solve harmful interactions. Here too a correct solution may require that all future consequences of an ordering decision be explored. Stratagems other than ordering may be necessary to solve the problem. For example, a new operator may perhaps need to be applied at a higher level. Consider the problem of switching the values of the two registers in a two-register machine. Applying the register-to-register move operator creates a harmful interaction that no ordering can solve, since a value is destroyed. The solution to this interaction involves applying a register-to-memory move operator at a high level in order to store one of the values temporarily. Correcting many types of harmful interactions efficiently seems very difficult in a domain independent planner – domain specific heuristics may be required.

3. Summary of SIPE System

The problem of parallel interactions has been studied at SRI International in the context of a domain-independent planning system. This section briefly summarizes the system and the following section explains how parallel interactions are handled. Ann Robinson and I have designed and implemented a system, SIPE (System for Interactive Planning and Execution monitoring) [7], that supports both interactive and automatic planning. At present, its automatic search is not very knowledgeable. SIPE is designed to allow interaction with users who are able to watch (graphically) and, when desired, guide and/or control the planning process, thus enabling the solution of much more difficult problems than would be possible with the automatic search. The system can keep many alternative plans in readiness, each with the appropriate context–thus making breadth-first or best-first searches easy to implement.

The system provides for representation of domain objects and their invariant properties in a type hierarchy with inheritance. Relationships among these objects that may change over time are represented as predicates in first-order logic, with universal quantification allowed. Predicates are used to describe the preconditions and effects of operators.

The system permits the posting of many types of constraints on the values of variables in a plan (e.g., specifying that the eventual instantiation of the variable must have a certain value for a certain attribute, that it must be the same as the instantiation of another variable, etc.). This allows partial descriptions of objects to be built up so that the planner can accumulate information before making decisions. Constraints help encode many things that would be hard to represent in the predicate calculus language used for preconditions and effects. Constraints are also used to represent information about parallel interactions (see the next section).

The planning is hierarchical–each goal and action can be expanded into a more detailed multistep plan until the primitive level is reached. Plans are represented as procedural nets (similar to those in NOAH [3]), their partial ordering being encoded by successor links in the net. SPLIT and JOIN nodes in the net allow for parallelism in plans.

Operators represent actions in the domain or abstractions of actions (that will eventually be

expanded at lower levels in the hierarchy to actual actions). Operators can introduce new variables, impose constraints upon new or old variables, and represent their instructions for expanding a node in a formalism similar to procedural nets–namely, nodes with slots that are partially ordered by the links between them (see [3]). The slots of a node will represent (among other properties), the name of an action to be performed for an action node, the predicates to be achieved for a goal node, the arguments to be used, and the effects of performing the given action. SIPE allows arguments of an action or goal node to be specified as "resources". As we shall see later, this is very useful for reasoning about parallel interactions.

The effects of actions must be appropriately defined in the operators so the system can use the STRIPS assumption to determine the state of the world in the middle of the plan. Predicates in the effects may contain universal quantifiers. SIPE also permits specification of deductive operators; tightly controlled deductions are performed automatically by the system. This makes the encoding of operators much simpler, since, in general, only the primary effect must be encoded in an operator and most side effects can then be deduced from the primary effect by using the deductive operators.

Many of the above features of the SIPE planning system are extensions of previous domain-independent planning systems. Among such extensions are the following: development of a perspicuous formalism for encoding descriptions of actions; the use of constraints to partially describe objects; the creation of mechanisms that permit concurrent exploration of alternative plans; the incorporation of heuristics for reasoning about resources; use of quantifiers in the effects of actions and mechanisms that make it possible to perform simple deductions (thereby simplifying operator descriptions).

4. Handling Parallel Interactions in SIPE

The SIPE system has produced correct parallel plans for problems in four different domains (the blocks world, cooking, aircraft operations, and a simple robotics assembly task). This section describes new features and heuristics in the system that aid in handling parallel interactions. These

fall into four areas: (1) reasoning about resources, which is the major contribution of SIPE; (2) using constraints to generate correct parallel plans; (3) explicitly representing the purpose of each action and goal to help solve harmful interactions correctly; (4) taking advantage of helpful interactions. Other planners have had some of the features (e.g., NONLIN [5] has features similar to the last two areas mentioned), but SIPE develops these ideas with a different emphasis. Our own emphasis has been to represent the information that must be reasoned about in a way that is natural to humans so that the planner can be easily controlled interactively–thus allowing more difficult problems to be solved.

4.1 Resources

In the following discussion, actual examples will be presented to show how resources help with parallel interactions. Figure 1 shows a PUTON operator written in SIPE. The above example of achieving (ON A B) and (ON B C) as a conjunction shows how resource reasoning is helpful. Figure 2 depicts a plan that might be produced by NOAH or NONLIN (or SIPE without making use of resource reasoning) for this problem. Figure 3 shows a plan from SIPE using resources in the operators.

The formalism for representing operators in SIPE includes a means of specifying that some of the variables associated with an action or goal actually serve as resources for that action or goal. Resources are to be employed during a particular action and then released, just as a saw is used during a cutting action. Reasoning about resources is a common phenomenon. It is a useful way of representing many domains, a natural way for humans to think of problems, and, consequently, an important aid to interaction with the system.

SIPE has specialized knowledge for handling resources; merely specifying that something is a resource causes the system to check on resource availability and on resource conflicts. It is often difficult or awkward to keep track of resources in current planning systems (e.g., [3] [5]). Resource availability in the latter would have to be axiomatized and checked in the preconditions of operators, while resource conflicts would have to be caught by the normal problematic interaction

```
OPERATOR: PUTON
ARGUMENTS: BLOCK1, OBJECT1 IS NOT BLOCK1;
PURPOSE: (ON BLOCK1 OBJECT1);
PLOT:
    PARALLEL
        BRANCH 1:
            GOAL
            GOALS: (CLEARTOP OBJECT1);
            ARGUMENTS: OBJECT1;
        BRANCH 2:
            GOAL
            GOALS: (CLEARTOP BLOCK1);
            ARGUMENTS: BLOCK1;
    END PARALLEL

    PROCESS
    ACTION: PUTON.PRIMITIVE;
    ARGUMENTS: OBJECT1;
    RESOURCES: BLOCK1;
    EFFECTS: (ON BLOCK1 OBJECT1);
END PLOT
END OPERATOR
```

Figure 1
a PUTON operator in SIPE

detector, which is less efficient (as we shall see below).

One advantage of resources, therefore, is that they help in the axiomatization and representation of domains. Declaration of a resource associated with an action connotes that one precondition of the action is the availability of that resource. Mechanisms in the planning system, as they allocate and deallocate resources, automatically ensure that these implicit preconditions will be satisfied. The user of the planning system does not have to axiomatize as a precondition the availability of resources in the domain operators.

Another important advantage of resources is that they help in early detection of problematic interactions on parallel branches. The system does not allow one branch to use an object which is a resource in a parallel branch. In NOAH and NONLIN, both original GOAL nodes are expanded with the PUTON operator or its equivalent. This produces a plan similar to the one shown in Figure 2. The central problem is to be aware that B must be put on C before A is put on B

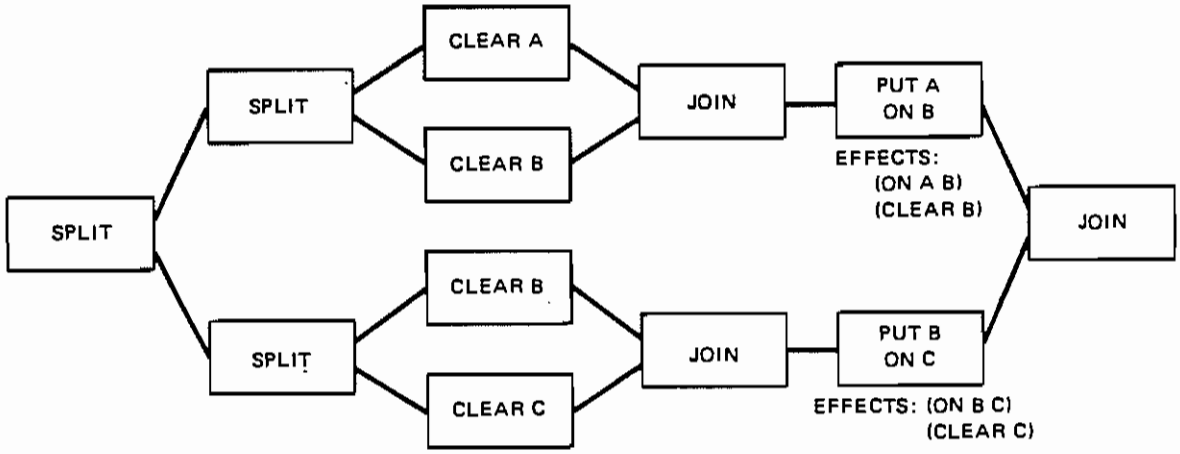

Figure 2
a plan without resources

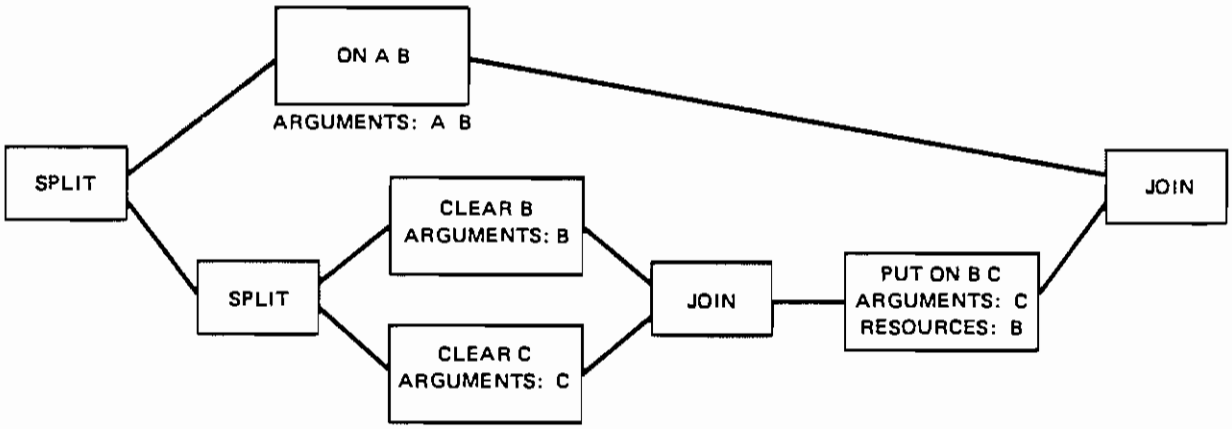

Figure 3
a plan with resources

(otherwise B will not be clear when it is to be moved onto C). NOAH and NONLIN both build up a table of multiple effects (TOME) that tabulates every predicate instance asserted or denied in the parallel expansions of the two GOAL nodes. Using this table, the programs detect that B is made clear in the expansion of (ON B C), but is made not clear in the (ON A B) expansion. Both programs then solve this problem by doing (ON B C) first.

SIPE uses its resource heuristic to detect this problem and propose the solution without having to generate a TOME. (SIPE does do a TOME-like analysis to detect interactions that do not fit into the resource reasoning paradigm.) When some object is listed in an action as a resource, the system then prevents that particular object from being mentioned as either a resource or an argument in any action or goal that is in parallel. In the example above, the block being moved is listed as a resource in the primitive PUTON operator because it is being physically moved. Therefore, nothing in a parallel branch should try to move it–or even be dependent on its current location. This restriction is enforced by not allowing a predicate on a parallel branch to mention the resource. This is a strong restriction (though useful in practice) and can be avoided by using shared resources (discussed below). Thus, as soon as the expansion of (ON B C) with the PUTON operator is accomplished and the plan in Figure 3 produced, SIPE recognizes that the plan is invalid since B is a resource in the expansion of (ON B C) and an argument in (ON A B). This can de detected without expanding the (ON A B) goal at all and without generating a TOME.

Not being able to refer to a resource in another branch is sometimes too strong a restriction. SIPE also permits the specification of shared resources, whereby the same object can be a resource or an argument in a parallel branch if certain conditions for the sharing are met. (Currently shared resources are sharable under all conditions as the sharing conditions have not yet been implemented.)

Resources help in solving harmful interactions, as well as in detecting them. In these interactions no goal is made false on a parallel branch; there is simply a resource conflict. However, if the resource availability requirements were axiomatized with predicates, an availability goal would be made false on a parallel branch. Thus, resource conflicts are considered to be problematic interactions. SIPE uses a heuristic for solving resource-argument conflicts. Such an interaction occurs when a resource in one parallel branch is used as an argument in another parallel branch (distinct from a resource-resource conflict, in which the same object is used as a resource in two parallel branches). This is the type of conflict that occurs in the plan in Figure 3, since B is a resource in the primitive PUTON action and an argument in (ON A B).

SIPE's heuristic for solving a resource-argument conflict is to put the branch using the object as a resource before the parallel branch using the same object as an argument. In this way SIPE decides that (ON B C) must come before (ON A B) in Figure 3. This is done without generating a TOME, without expanding both nodes, and without analyzing the interaction. The assumption is that an object used as a resource will have its state or location changed by such use; consequently, the associated action must be done first to ensure that it will be "in place" when later actions occur that use it as an argument. This heuristic is not guaranteed to be correct, but it has proven useful in the four domains encoded in SIPE. (The user can easily prevent the employment of this heuristic by interacting with the system.)

To take full advantage of resources, the system posts constraints. This capability is discussed briefly in the next section.

4.2 Constraints

Unbound variables in a plan can accumulate various constraints in SIPE, thus building up a partial description of the object. (Constraints were first used in planning by Stefik in his domain-specific planner MOLGEN [4].) This is useful for taking full advantage of resources to avoid harmful interactions. When variables that are not fully instantiated are listed as resources, the system posts constraints on the variables which point to other variables that are potential resource conflicts. When allocating resources, the system then attempts to instantiate variables so that no resource conflicts will occur. For example, if a robot arm is used as a resource in the block-moving operators, the system will try to use different robot arms (if they are available) on parallel branches, thus avoiding resource conflicts. If only one arm is available, it will be assigned to both parallel branches in the hope that the plan can later be ordered to resolve the conflict. In this way many harmful interactions are averted by intelligent assignment of resources.

4.3 Solving Harmful Interactions: Purposes and Side Effects

The difficulty entailed in eliminating harmful interactions has already been discussed. However,

if the system knows why each part of the plan is present, it can use this information to come up with reasonable solutions to some harmful interactions. Suppose a particular predicate is made false at some node on one parallel branch and true at another node on another parallel branch. Depending on the rationale for including these nodes in the plan, it may be the case that the predicate is not relevant to the plan (an extraneous side effect), or must be kept permanently true (the purpose of the plan), or must be kept only temporarily true (a precondition for later achievement of a purpose).

Solutions to a harmful interaction may depend on which of these cases hold. Let us call the three cases side effect, purpose, and precondition, respectively, and analyze the consequent possibilities. If the predicate in conflict on one branch is a precondition, one possible solution is to further order the plan, first doing the part of the plan from the precondition on through its corresponding purpose. Once this purpose has been accomplished, there will be no problem in negating the precondition later. This solution applies no matter which of the three cases applies to the predicate in the other conflicting branch.

In case both conflicting predicates are side effects, it is immaterial to us if the truth value of the predicate changes and thus no real conflict exists. In the case of a side effect that conflicts with a purpose, one solution is to order the plan so that the side effect occurs before the purpose; thus, once the purpose has been accomplished it will remain true. When both conflicting predicates are purposes, there is no possible ordering that will achieve both purposes at the end of the plan. The planner must use a different operator at a higher level or plan to reachieve one of the purposes later. However, none of the above suggestions for dealing with interactions can be guaranteed to produce the best solution.

This has been a brief summary of SIPE's algorithm for dealing with problematic interactions. Systems like NOAH and NONLIN do similar things. However, SIPE provides methods for more precise and efficient detection. It should be emphasized that many interactions that would be problematical in the other systems are dealt with in SIPE by the resource-reasoning mechanisms and therefore do not need to be analyzed. When interactions are being analyzed, SIPE requires

that one of the conflicting predicates be a goal (not just a side effect) at some level in the hierarchy. In this way, interactions between side effects that pose no problems are not even detected. This requires that all important predicates be recognized as goals at some level, which is easily done in SIPE's hierarchical planning scheme. The system also distinguishes between main and side effects at each node in the plan. This makes it easy to tell which predicates are of interest to us at any level of the plan without looking up the hierarchy (since higher-level goals will become main effects at lower-level actions).

SIPE also provides for exact expression of the purpose of any goal in its operators. NOAH used a heuristic, according to which the last node in an expansion was the purpose of the expansion. This is not always accurate and in SIPE a node can specify any later node in the expansion as its purpose. This enables better analysis of problematic conflicts.

4.4 Changing Goals to Phantoms Through Linearization

SIPE recognizes helpful interactions and will try to further order the plan to take advantage of them, although the user can control this interactively if he wishes. If a goal that must be made true on one parallel branch is actually made true on another parallel branch, the system will order the plan so that the other branch occurs first (if this causes no other conflicts). The goal can then be changed to a phantom and need not be achieved.

NOAH was not able to take advantage of such helpful effects. NONLIN did have an ability to order the plan in this way. This is an important ability in many real-world domains, since helpful side effects occur frequently. For example, if parallel actions in a robot world both require the same tool, only one branch need plan to get the tool out of the tool box; the other branch should be able to recognize that the tool is already out on the table.

5. Related Work

Much planning-like research does not fit into the context we have defined here–because it is

specialized to one domain, because it does not make the STRIPS assumption, or because it does not reason about parallel actions. For example, much automatic programming research does not deal with parallel actions or does not make the STRIPS assumption. The STRIPS planner [1] itself does not deal with parallel actions.

The most relevant systems, NOAH and NONLIN, have already been compared with SIPE throughout this paper. Both NOAH and NONLIN find interactions by generating a TOME (table of multiple effects). The TOME finds all interactions, even harmless ones (i.e., those in which both predicates are side effects), work which SIPE avoids. SIPE also provides for explicit designation of purpose for preconditions that NOAH does not provide. NONLIN provides a similar capability with its "goal structure". The most significant improvement in SIPE is the use of resource reasoning (and the ability to post constraints), which averts many harmful interactions and enables many others to be recognized quickly and solved. Neither NOAH nor NONLIN provides a similar capability.

8. Conclusion

We have defined the problem of parallel interactions in a context that is not unique to SIPE. The difficulty of solving harmful interactions was discussed and a case-by-case analysis of different situations presented. An actual planning system was described that incorporates several new mechanisms able to assist in dealing with the parallel interaction problem. The most significant of these mechanisms is the ability to reason about resources. Combined with the system's ability to post constraints, resource reasoning helps the system avoid many harmful interactions, helps it recognize sooner those interactions that do occur, and helps the system solve some of these interactions more quickly.

REFERENCES

1. Fikes, R., Hart, P., and Nilsson, N., "Learning and Executing Generalized Robot Plans," *Readings in Artificial Intelligence*, Nils Nilsson and Bonnie Webber, ed., Tioga Publishing, Palo Alto, California (1981), pp. 231–249.

2. Robinson, A.E., and Wilkins D.E., "Representing Knowledge in an Interactive Planner," *Proceedings of the First Annual Conference of the AAAI*, Stanford, California (August 1980), pp. 148–150.

3. Sacerdoti, E., *A Structure for Plans and Behavior*, Elsevier, North-Holland, New York, 1977.

4. Stefik, M., "Planning With Constraints," *Report STAN-CS-80-784*, Computer Science Department, Stanford University, Ph.D. Dissertation (1980).

5. Tate, A., "Generating Project Networks," *Proceedings IJCAI-77*, Cambridge, Massachusetts (August 1977), pp. 888–893.

6. Waldinger, R., "Achieving Several Goals Simultaneously," *Readings in Artificial Intelligence*, Nils Nilsson and Bonnie Webber, ed., Tioga Publishing, Palo Alto, California (1981), pp. 250–271.

7. Wilkins, D.E., and Robinson, A.E., "An Interactive Planning System," *Artificial Intelligence Center Technical Note 245*, SRI International, Menlo Park, California (March 1980).