Systematic Nonlinear Planning

David McAllester[*]
MIT Artificial Intelligence Laboratory
dam@ai.mit.edu

David Rosenblitt
WEB Development Corporation
415 McFarlan Road, Suite 100
Kennett Square, PA 19348

This paper appeared in AAAI-91.
A postscript electronic source for this paper can be found in ftp.ai.mit.edu:/pub/dam/aaai91c.ps.
A bibtex reference can be found in ftp.ai.mit.edu:/pub/dam/dam.bib.

Abstract

This paper presents a simple, sound, complete, and systematic algorithm for domain independent STRIPS planning. Simplicity is achieved by starting with a ground procedure and then applying a general, and independently verifiable, lifting transformation. Previous planners have been designed directly as lifted procedures. Our ground procedure is a ground version of Tate's NONLIN procedure. In Tate's procedure one is not required to determine whether a prerequisite of a step in an unfinished plan is guaranteed to hold in all linearizations. This allows Tate's procedure to avoid the use of Chapman's modal truth criterion. Systematicity is the property that the same plan, or partial plan, is never examined more than once. Systematicity is achieved through a simple modification of Tate's procedure.

Introduction

STRIPS planning was introduced in (Fikes and Nilsson 1971) as a model of the kind of planning problems that people appear to solve in common sense reasoning. STRIPS planning corresponds to a certain formal graph search problem. John Canny has observed that the formal STRIPS planning problem is PSPACE complete (Canny 1985). This means, essentially, that any sound and complete planner must search. It is well known that certain NP complete and PSPACE complete problems can be solved efficiently for the vast majority of problems that arise in practice. Although it is still controversial whether STRIPS planning can be made practical for large problems, it now seems clear that certain methods of search optimization can dramatically improve the performance of planning algorithms.

Planning procedures have used three basic techniques to optimize the required search process. First, even the earliest planning systems were "lifted". This means that they used expressions involving variables, e.g., **PUTON(A** x**)**, to represent a large number of different possible ground instances, e.g., **PUTON(A B)**. The earliest planners were based on resolution theorem proving and inherited their lifted nature from the lifting lemma of resolution (Fikes and Nilsson 1971) (Robinson 1965). In addition to being lifted, most modern planners are "nonlinear" — they maintain a partial order on plan steps rather than a total order. This partial order is gradually refined during the planning process (Sacerdoti 1975), (Sacerdoti 1977), (Tate 1977), (Chapman 1987). Finally, some planners use "abstraction spaces" in which planning is first done at a high level of abstraction and then low level details are filled in once a high level plan has been found (Sacerdoti 1974), (Korf 1987), (Yang and Tenenberg 1990).

Nonlinear planners are sometimes called "least commitment planners". In general, the informal principle of least commitment states that one should should make low commitment choices before making high commitment choices. Lifting is a good example of the general principle of least commitment. In searching for a plan we might select **PUTON(A B)** as the first step of the plan. This is a high commitment choice. A lower commitment choice is to state that the first step of the plan is an expression of the form **PUTON(A** x**)**. Nonlinearity is another example of the principle of least commitment. Rather than select an expression of the form **PUTON(A** x**)** as the first step in the plan, we can state that **PUTON(A** x**)** is to appear *somewhere* in the plan without committing ourselves as to where. This is a very low commitment choice.

This paper presents a simple, sound, complete, and systematic planning algorithm. As with many previous planners, our algorithm uses lifting and nonlinearity to optimize the required search.[1] Our algorithm has two novel features which justify the present paper — simplicity and systematicity. The planner is sim-

[*]This research was supported in part by National Science Foundation Grant IRI-8819624 and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124 and N00014-89-j-3202

[1]Abstraction can also be easily incorporated into our algorithm, as described in a later section.

ple for two reasons. First, the procedure is constructed by combining two independent components — a simple ground procedure and a general lifting technique. Previous planning algorithms have been designed directly as lifted procedures. Our ground procedure is a simplified ground version of Tate's NONLIN procedure. We observe that the ground version of Tate's procedure is sound and complete. This is true in spite of the absence of any subroutine for determining whether a prerequisite of a step in an incomplete plan is guaranteed in all linearization. Chapman has shown that any such subroutine must evaluate a certain complex modal truth criterion (Chapman 1987). The algorithm presented here does not involve any such subroutine, nor does it make use of any modal truth criterion.

In addition to being simple, the procedure presented here performs systematic search, i.e., the same plan (or partial plan) is never examined more than once. Systematicity is achieved through a simple modification of the ground version of Tate's procedure.

STRIPS Planning

First we formally define the STRIPS planning problem. In the initial formulation we only consider the ground case — we do not allow variables to appear in propositions or in the specification of operators.

Definition: A *STRIPS operator* consists of an operator name plus a *prerequisite list*, an *add list* and a *delete list*. The elements of the prerequisite, add, and delete lists are all proposition expressions.

For example, in blocks world planning the operator with name `MOVE(A, B, C)`, which should be read as "move block A from block B to block C", has prerequisites `CLEAR(A)`, `ON(A, B)`, and `CLEAR(C)`, delete list `ON(A, B)` and `CLEAR(C)`, and add list `CLEAR(B)` and `ON(A, C)`. A "state of the world" is modeled by a set of proposition expressions. We now give the standard definition for the result of performing a STRIPS operation in a given state of the world and the definition of a STRIPS planning problem.

Definition: If o is a STRIPS operator, and Σ is a set of proposition expressions, then, if the prerequisites list of o is a subset of Σ, then the result of performing operation o in the state Σ is Σ minus the delete list of o plus the add list of o. If the prerequisite list of o is not a subset of Σ, then then result of performing o in the state Σ is the empty set.

Definition: A *STRIPS planning problem* is a triple $<\mathcal{O}, \Sigma, \Omega>$ where $\mathcal{O}$ is a set of STRIPS operators, Σ is a set of initial propositions, and Ω is a set of goal propositions.

Definition: A *solution* to a STRIPS planning problem $<\mathcal{O}, \Sigma, \Omega>$ is a sequence α of operations, each of which must be a member of $\mathcal{O}$, such that the result of consecutively applying the operations in α starting with the initial state Σ results in a set that contains the goal set Γ.

As mentioned above, determining whether or not an arbitrary STRIPS planning problem has a solution is a PSPACE complete problem. The following section presents a simplified ground version of Tate's NONLIN planning procedure.

Nonlinear Planning

A plan is a sequence of operations. Intuitively, two plans are considered to be equivalent if one can be derived from the other by reordering non-interacting steps. For example, consider a robot that must perform a bunch of tasks in room `A` and a bunch of tasks in room `B`. Each task is formally associated with a certain goal proposition. We want to achieve propositions $P_1, \cdots, P_n, Q_1, \cdots, Q_m$. We are given operators $A_1, \cdots, A_n, B_1, \cdots, B_m$ where A_i achieves P_i but must be done in room `A` and B_i achieves Q_i but must be done in room `B`. More formally, each A_i has the single prerequisite `IN(A)`, adds the proposition P_i, and does not delete any propositions. Each B_i is defined similarly with the prerequisite `IN(B)`. We also have an operator `GO(A)` which has no prerequisites, adds the proposition `IN(A)`, and deletes the proposition `IN(B)`. We also have an analogous operator `GO(B)`. The goal set $\{P_1, \cdots, P_n, Q_1, \cdots, Q_m\}$ can be achieved (without any initial assumptions) by the plan `GO(A)`$; A_1; \cdots; A_n;$`GO(B)`$; B_1; \cdots; B_m$. Clearly, the order of the A_i steps and the order of the B_i steps does not matter as long as all the A_i steps are done in room `A` and all the B_i steps are done in room `B`. This plan should be considered to be equivalent to the plan `GO(A)`$; A_n; \cdots; A_1;$`GO(B)`$; B_m; \cdots; B_1$ which performs the A_i steps and B_i steps in the opposite order. Every (linear) plan that is a solution to a given planning problem can be abstracted to a "nonlinear plan" where the nonlinear plan contains only a partial order on the plan steps. Two linear plans are considered to be equivalent if they are different representations of the same nonlinear plan.

To define the nonlinear plan associated with a given linear plan we must first overcome a minor technical difficulty. In a linear plan we can name the individual plan steps by referring to the first step, the second step, and so on. In a nonlinear plan, however, there may not be any well defined second step. We can not name a step by giving the operator used at that step because several steps may involve the same operator. To provide names for the steps in a nonlinear plan we assume that each step is associated with a distinct symbol called the

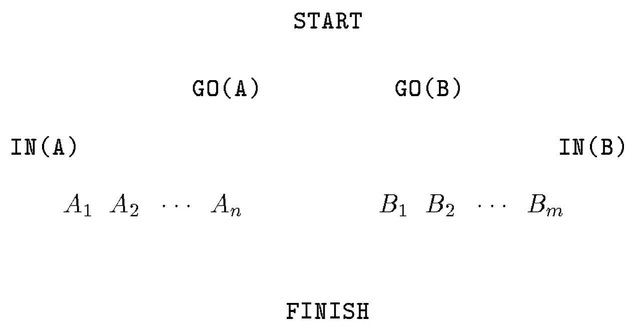

Figure 1: The causal links in the linear plan `GO(A)`; $A_1; \cdots; A_n$;`GO(B)`; $B_1; \cdots; B_m$
.

name of that step.

> **Definition:** A *symbol table* is a mapping from a finite set of step names to operators. Every symbol table is required to contain two distinguished step names called `START` and `FINISH`. `START` is mapped to an operator that has no prerequisites and no delete list but which adds a set of "initial propositions". `FINISH` is mapped to an operator that has a set of prerequisites called "goal formulas" but has an empty add list and delete list.

Note that a symbol table does not impose ordering constraints on the step names. Also note that step names are different from operator names. Two step names, say `STEP-37` and `STEP-52`, may both map to the operator named `MOVE(A, B, C)`. Also note that `STEP-37` does not necessarily precede `STEP-52` — the step names have no significance other than as place holders for steps in a plan.

Consider a (linear) solution to a STRIPS planning problem. Without loss of generality we can assume that every prerequisite of every step in the solution plan is true when that step is executed. If w is a step name, and P is a prerequisite of w, then there must be some "source" for P at w — although the source may be the virtual step `START`. If every prerequisite is satisfied for every operation in the plan, then every prerequisite P of every step has a unique source — the last preceding step that adds P. This includes the prerequisites of the virtual step `FINISH`, i.e., the goal formulas. The notion of source motivates the definition of a causal link.[2]

> **Definition:** A *causal link* is a triple $<s, P, w>$ where P is a proposition symbol, w is a step name that has P as a prerequisite, and s is a step name that has P in its add list. Causal links are written as $s \xrightarrow{P} w$

[2]Tate uses the term "range" rather than causal link.

Causal links indicate the dependencies among steps in a plan. A causal link should be viewed as a constraint. The causal link $s \xrightarrow{P} w$ requires that step s precede step w and that *no step between s and w either adds or deletes P*. Given any linear plan, one can extract the causal links of that plan. For example, consider the plan `GO(A)`; $A_1; \cdots; A_n$;`GO(B)`; $B_1; \cdots; B_m$ discussed earlier. This plan has the set of causal links shown in figure 1. Unfortunately, the causal links do not contain all of the relevant ordering constraints in this plan. The partial order information implicit in the casual links does not require each A_i to be done before `GO(B)`. This example shows the need for ordering information other than causal links.

> **Definition:** A step name v is called a *threat* to a causal link $s \xrightarrow{P} w$ if v is a step name, other than s and w, that either adds or deletes P.

> **Definition:** A *safety condition* is an ordering $s < w$ or $w > s$ where s and w are step names.

In the plan shown in figure 1, the step `GO(A)` is a threat to each casual link of the form $\texttt{GO(B)} \xrightarrow{\texttt{IN(B)}} B_i$ (since `GO(A)` deletes the proposition `IN(B)`), leading to the safety condition `GO(A)` < `GO(B)`. Similarly, the step `GO(B)` is a threat to each causal link of the form $\texttt{GO(A)} \xrightarrow{\texttt{IN(A)}} A_i$, leading to safety conditions of the form $A_i <$ `GO(B)`. These safety conditions, along with the orderings implicit in the causal links, allow the steps to be executed in any order in which `GO(A)` precedes every A_i, every A_i precedes `GO(B)`, and `GO(B)` precedes every B_i.

> **Definition:** A *nonlinear plan* consists of a symbol table, a set of casual links, and a set of safety conditions.

> **Definition:** A nonlinear plan is called *complete* if the following conditions hold.
>
> - Every step name appearing in the causal links and safety conditions has an entry in the symbol table.
> - If w is a step name in the symbol table, and w has prerequisite P, then the plan contains some causal link of the form $s \xrightarrow{P} w$.
> - If the plan contains a causal link $s \xrightarrow{P} w$, and the symbol table contains a step name v that is a threat to the causal link $s \xrightarrow{P} w$, then the plan contains either the safety condition $v < s$ or the safety condition $v > w$.

It is possible to show that any (linear) solution to a STRIPS planning problem corresponds to a nonlinear plan that is the least (smallest number of causal links and safety conditions) complete nonlinear plan corresponding to the given linear plan. This "nonlinear abstraction" of a given linear plan is unique up to the arbitrary choice of step names. The nonlinear abstraction of linear plans determines an equivalence relation on linear plans — two linear plans are considered to be equivalent if they have the same nonlinear abstraction.

Definition: A *topological sort* of a nonlinear plan is a linear sequence of all the step names in the symbol table such that the following conditions hold.

- The first step in the sequence is `START`.
- The last step in the sequence is `FINISH`.
- For each causal link $s \xrightarrow{P} w$ in the plan, the step s precedes the step w.
- For each safety constraint $u < v$ (or $v > u$) in the plan, the step u precedes the step v.

Definition: A topological sort of a nonlinear plan is a *solution* if executing the sequence of operations of the steps between the `START` and `FINISH` steps, starting in the state given by the add list of the `START` step, results in a state that contain all the preconditions of the `FINISH` step.

Lemma: Any topological sort of a complete nonlinear plan is a solution.

It is possible to construct a planning procedure which systematically searches the space of nonlinear plans. The search is systematic in the technical sense that it never visits the same plan, or even equivalent plans, twice — every branch in the search tree divides the remaining possibilities into *disjoint* sets of potential solutions such that all equivalent plans are down the same branch of the search tree. Before defining the procedure, however, one additional definition is needed.

Definition: A nonlinear plan (not necessarily complete) is called *order inconsistent* if it has no topological sort.

A transitive closure algorithm can be used to determine if a given nonlinear plan is order inconsistent. A nonlinear plan is order inconsistent if and only if the causal links and safety conditions of the plan define a cycle in the plan steps.

Our search procedure is a bounded depth first procedure that can be used with iterative deepening (Korf 1985). The procedure takes an (incomplete) nonlinear plan and a cost bound and searches for a completion of the given plan such that total cost of the steps in the completion is not greater than the given bound. Initially the procedure is called on a (partial) nonlinear plan that contains only the `START` and `FINISH` steps corresponding to a given STRIPS planning problem. We also assume a set of given allowed operations.

The Procedure `FIND-COMPLETION(`β c`)`

1. If the nonlinear plan β is order inconsistent, or the total cost of the steps in β is greater than c, then fail.
2. If the nonlinear plan β is complete then return β.
3. If there is a causal link $s \xrightarrow{P} w$ in β and a threat v to this link in the symbol table such that β does not contain either $v < s$ or $v > w$, then nondeterministically return one of the following.
   (a) `FIND-COMPLETION(`$\beta + (v < s), c$`)`
   (b) `FIND-COMPLETION(`$\beta + (v > w), c$`)`
4. There must now exist some step w in the symbol table and some prerequisite P of w such that there is no causal link of the form $s \xrightarrow{P} w$. In this case nondeterministically do one of the following.
   (a) Let s be (nondeterministically) some step name in the symbol table that adds P and return the plan

   `FIND-COMPLETION(`$\beta + s \xrightarrow{P} w, c$`)`.

   (b) Select (nondeterministically) an operator o_i from the allowed set of operations such that o_i adds P. Create a new entry in the symbol table that maps a new step name s to the operator o_i. Then return the plan

   `FIND-COMPLETION(`$\beta + s \xrightarrow{P} w, c$`)`.

One can check that every completion of the given plan with cost c or less is equivalent (up to renaming of steps) to a possible value of the above procedure. Furthermore, one can show that no two distinct execution paths can produce equivalent complete plans. To see this note that every plan step can be uniquely named by starting with the `FINISH` step and moving backward over causal links noting the prerequisite at each link. This implies that each step can be uniquely identified in a way that is independent of step names. So no two equivalent plans can be generated by different choices for s in steps 4a and 4b. It also implies that the order constraints $v < s$ and $v > w$ used in step 3 can be defined independent of the step names. This implies that no completion satisfying $v < s$ can be equivalent (under step renaming) to a completion satisfying $v > w$. So the above procedure defines a systematic search of the space of complete nonlinear plans.

Comments on the Procedure

Although the procedure given in the previous section is essentially a simplification of Tate's NONLIN procedure, there is one technical difference worth noting. Tate's procedure uses a different notion of threat under which v is considered to be a threat to $s \xrightarrow{P} w$ only if v deletes P. The stronger notion of threat used here (in which v is a threat to $s \xrightarrow{P} w$ if v either adds or deletes P) is needed for systematicity. Under Tate's notion of

s_1
s_2
w_1
w_2
FINISH

Figure 2: A safe but incomplete nonlinear plan

threat it is possible that two distinct complete nonlinear plans share a common topological sort. In this case linear plans do not have unique nonlinear abstractions. However, it seems likely that Tate's weaker notion of threat works just as well, if not better, in practice.

The procedure given in the previous section can modified to handle planning in a series of abstraction spaces. Suppose that each proposition expression is given a number expressing "abstractness". We want to ensure that abstract prerequisites are satisfied before we attempt to satisfy concrete prerequisites. Steps 3 and 4 of the procedure select either a prerequisite that is not involved in a causal link, or a threat to an existing causal link. The selection of prerequisite or threatened causal link can be made arbitrarily. In particular, this selection can be done in a way that maximizes the abstractness of the prerequisite involved. If this is done, the procedure will only consider concrete prerequisites after all the more abstract prerequisites have been fully handled. Intuitively, one should consider "difficult" prerequisites before considering "easy" prerequisites.

The lemma in the previous section states that every complete nonlinear plan is safe in the sense that every topological sort of the plan is a solution. However, the converse of this lemma does not hold — there exist safe partial orders on plan steps which do not correspond to any complete nonlinear plan. For example, consider the partial order on plan steps shown in figure 2.[3] In this plan **FINISH** has prerequisites P, Q and R. The step w_1 adds P and Q while the step w_2 adds P and R. Step w_1 has prerequisite $W1$, which is added by s_1, and w_2 has prerequisite $W2$, which is added by s_2. Unfortunately, both s_1 and s_2 delete P. The partial order on plan steps shown in figure 2 is safe.[4] However, any complete plan (under the definition given here) must specify which of w_1 or w_2 is the casual source of the prerequisite P of the **FINISH** step. This will enforce an explicit ordering of w_1 and w_2.

It is possible to systematically search all partial orders on step names and find the most abstract (least committed) partial order that is safe. Unfortunately, there appears to be no way of doing this efficiently. Evaluating Chapman's modal truth criterion at each node of the search space is very expensive. Furthermore, treating the modal truth criterion as a nondeterministic procedure, as is done in TWEAK, destroys the systematicity of the search — the choices in the modal truth criterion (the disjunctions and existential quantifiers) do not correspond to dividing the space of partial orders into disjoint sets (the same partial order can be reached through different branches in the modal truth criterion). Even if one did systematically search the space of partial orders on step names, different orderings of step names can correspond to the same ordering of actual operations. This implies that, unlike the procedure given here, the search would still not be a systematic search of operation sequences.

[3]This example is due to Subbarao Kambhampati.

[4]This example demonstrates the necessity of the white night condition in checking the safety of ground nonlinear plans.

Lifting

Lifting was invented by J. A. Robinson in conjunction with the development of resolution theorem proving (Robinson 1965). Lifting is now a standard technique in the construction of new theorem proving and term rewriting algorithms. Any application of lifting is associated with a lifting lemma which states that for every possible computation involving ground expressions there is a lifted computation involving variables such that the ground computation is a substitution instance of the lifted computation.

The procedure given above is designed for the ground case — the case where the propositions in the prerequisite list, add list, and delete lists of the operators do not contain variables. This procedure can be lifted to handle operator specification schemas and plan schemas involving variables. The lifting transformation used here is quite general — it applies to a large class of nondeterministic programs.

To lift the procedure of the previous section first note that the procedure, as written, can be used with operator schemas. An operator schema is an operator in which variables appear in the operator name, prerequisite list, add list, and delete list. For example, in a blocks world with n blocks there are n^3 operators of the form **MOVE(A, B, C)**. These n^3 different operators are just different substitution instances of one operator schema **MOVE**(x, y, z) which has variables x, y, and z. To use the procedure of the previous section with operator schemas rather than ground operators step 4b is changed to read as follows.

4b Let o_i be some ground instance of one of the given operator schemas. If P is not a member of the add list of o_i then fail. Otherwise, create a new entry in the symbol table that maps a new step name s to the operator o_i. ...

The above version of step 4b is still a ground procedure. Note that the nondeterministic branching factor of this step is infinite — if arbitrary ground terms may be introduced by the substitution operation then there are infinitely many ground instances of a single operator schema. This is clearly unacceptable for practical planning. To lift the ground procedure we replace step 4b as stated above with the following.

4b Let o_i be a copy, with fresh variables, of one of the given operator schemas. If P is not a member of the add list of o_i, fail. Otherwise, create a new entry in the symbol table that maps a new step name s to the schema copy o_i. ...

Note that only the first part of the step has changed. The branching factor of first part of the lifted step 4b is equal to the number of different operator schemas given — in the case of the blocks world there is only one schema so the step is deterministic. The remainder of the lifted procedure reads identically to the procedure given in the previous section. However, in the lifted version of the procedure expressions may contain variables. An equality test between two expressions that contain variables is treated as a nondeterministic operation. The equality test may return true, in which case an "equality constraint" is generated. The equality test may also return false in which case a disequality constraint is generated.[5] For example, the second part of step 4b reads "If P is not a member of the add list of o_i then fail". To determine if P is a member of the add list of o_i we can write a simple recursive membership test that performs an equality test between the proposition P and each member of the add list of o_i. Each equality test can either return true or false. The computation only proceeds if some equality test returns true and therefore generates an equality constraint between P and a particular member of the add list of o_i. Equality constraints invoke unification. If an equality constraint is generated between expressions that can not be unified, then that branch of the nondeterministic computation fails. Thus, the above version of step 4b only succeeds if P unifies with some element of the add list of o_i.

In the general lifting transformation each statement of the form "let x be a ground instance of schema y" is replaced by "let x be a copy of y with fresh variables". Each equality test is then treated as a nondeterministic operation that either returns true and generates an equality constraint or returns false and generates a disequality constraint. If the set of equality and disequality constraints ever becomes unsatisfiable, that branch of the nondeterministic computation fails. Given any such set of equality and disequality constraints on expressions involving variables it is possible to quickly determine if the constraints are satisfiable. More specifically, one first applies a unification algorithm to the equality constraints to see if the these constraints can be satisfied simultaneously. If the unification succeeds, then one applies the substitution resulting from that unification to all expressions. If there is a disequality constraint between two expressions that become the same expression when the substitution is applied, then the constraints are unsatisfiable. Otherwise, the constraints are satisfiable. In practice, the unification operation would be performed incrementally as equality constraints are generated.

[5] Disequality constraints of the form $s \neq w$ are an essential part of sound and complete lifted planning procedures such as TWEAK.

Conclusion

Previous lifted nonlinear planning algorithms have been quite complex and have failed to generate systematic searches. By treating lifting as a separate optimization that can be performed after a ground algorithm is designed, we have found a simple, sound, complete, and systematic lifted nonlinear procedure.

References

John Canny. Unpublished Observation, 1985.

David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:198–208, 1971.

Richard E. Korf. Iterative-deepening a*: An optimal admissible tree search. In *Proceedings of the 9th IJCAI*, pages 1034–1036, August 1985.

Richard E. Korf. Planning as search, a quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.

J. A. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12(1), January 1965.

Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

Earl D. Sacerdoti. The nonlinear nature of plans. In *IJCAI75*, pages 206–214, 1975.

Earl D. Sacerdoti. *A Structure for Plans and Behavior*. American Elsevier, New York, NY, 1977.

Austin Tate. Generating project networks. In *IJCAI77*, pages 888–893, 1977.

Qiang Yang and Josh D. Tenenberg. Abtweak: Abstracting a nonlinear least commitment planner. In *IJCAI90*, pages 204–209, 1990.