An Algebraic Approach to Conflict Resolution in Planning

Qiang Yang *
Department of Computer Science
University of Waterloo
Waterloo, Ont. Canada, N2L 3G1
qyang@watdragon.waterloo.edu

Abstract

This paper presents an algebra for conflict resolution in nonlinear planning. A set of conflicts in a plan is considered as a constraint network. Each node in the network represents a conflict, and is associated with a set of alternative ways for resolving it. Thus, resolving conflicts in a plan corresponds to selecting a set of consistent resolution methods so that, after they are applied to the plan, every conflict can be removed. The paper discusses the representional issues related to the conflict resolution, presents an algebra for resolving conflicts, and illustrates that some modified algorithms for preprocessing networks of constraints can greatly enhance the efficiency of conflict resolution.

Introduction

Many planners can be considered as search in a space of possible plans [Chapman, 1985; Sacerdoti, 1977; Stefik, 1981; Tate, 1977; Wilkins, 1988]. A major contributing factor to the branching factor in this space is the number of alternative ways for resolving a set of conflicts in a plan. Since search efficiency is greatly affected by the branching factor in the search space, reducing the number of ways for resolving a set of conflicts is an important way for improving planning efficiency.

Unfortunately, most existing planning systems [Chapman, 1985; Sacerdoti, 1977; Stefik, 1981; Tate, 1977; Wilkins, 1988] spend little or no effort in analyzing conflicts in an intermediate plan, in order to reduce the number of ways for resolving them. Usually there is more than one conflict to be introduced to a given plan as a result of some planning activity. These planners will simply generate a set of resolution methods for each conflict, and either commit to one of them in a depth-first manner, or generate the set of all possible successor states, in a breadth-first way. However, as we will show later in the paper, some of the conflict resolution methods can be proven to be either not applicable to the current plan, or related to other methods in such a way that they are redundant. In most of the existing planning systems, a great deal of computational overhead can be spent on these "useless" branches. It would be desirable to reduce the number of alternative ways in many circumstances through an analysis of conflicts in a plan.

Without any analysis at all on the inter-relations among the conflicts can be considered as one extreme on a spectrum of search control, while doing a complete analysis in order to minimize the number of possible alternatives can be considered as the other extreme on that spectrum. Certainly, it may not be advantageous to do a complete analysis over doing no analysis in improving planning efficiency, since there may exist some middle points on the spectrum that are better than either extreme. However, without knowing both ends of the spectrum well, it is hard to make an intelligent decision on how much analysis is needed. It is the purpose of this paper to investigate various ways for minimizing the number of alternative methods for resolving a given set of conflicts.

In particular, this paper presents a *conflict algebra* for analyzing methods for resolving a set of conflicts in nonlinear plans. Modified versions of the preprocessing algorithms for network-based constraint satisfaction problems (CSP) can be used for efficient application of the algebra. The modified algorithms take into account a wider range of possible constraints, so that it is possible to prune from the constraint network not only values that are inconsistent, but also those that can be proven "redundant." Interestingly, these algorithms can be applicable to large classes of problem domains rather than just the planning domain, and in this respect, part of the results in this paper should also be of interest to researchers in CSP area.

Below we discuss how conflicts in a plan are represented. Then we consider the details of the conflict algebra, and show how to apply it to planning. This is followed by a discussion of the algorithms for preprocessing a constraint network.

*This work was supported in part by an interim research grant from the Faculty of Mathematics at the University of Waterloo.

Conflicts and Conflict Resolutions

Preliminaries

A plan consists of a set of operators, a set of precedence constraints on the operators, and a set of codesignation constraints on the binding of variables of the operators. Each operator o is defined in terms of a set of preconditions, P_o, and a set of effects E_o. Two special operators, start and finish, exist in any plan, representing the initial and goal situations, respectively. The operator, start (finish), has a set of empty preconditions (effects), and has as its effects (preconditions) the set of conditions true in the initial (goal) situation.

Let P be a plan. We adopt the notational convention of [Chapman, 1985] for precedence and codesignation constraints. Thus, $a \prec b$ denotes that the operator a precedes operator b in P, and $p \approx q$ or $p \not\approx q$ denotes that p and q are constrained to codesignate or noncodesignate in P, respectively. If for two variables x and y, $x \approx y$, then x and y are constrained to be bound to the same constant. Moreover, we also assume both the definitions and graphical notation of *necessarily* ($\Box$) and *possibly* ($\Diamond$) in [Chapman, 1985].

Below we formally define conflicts. To do this, we first define *precondition establishment*:operator a is said to establish a condition p for operator b, or $\text{Est}(a, b, p)$, if and only if (i) p is a precondition for b, (ii) $\Box(a \prec b)$, (iii) $\exists u \in E_a$ such that $\Box(u \approx p)$, and (iv) $\forall a'$ such that $\Box(a \prec a' \prec b)$ and $\forall u' \in E_{a'}$, $\neg\Box(u' \approx p)$. That is, no other operators necessarily between a and b necessarily assert p.

In [Hertzberg and Horz, 1989], a is called a *producer* of p for b, while b is called a *user* of p. If for every operator b in a plan and for every precondition p of b, there exists an operator a such that $\text{Est}(a, b, p)$, then P is said to be *well-formed* [Hertzberg and Horz, 1989]. Note that a well-formed plan is not necessarily a correct plan, since some operators may exist that can possibly deny certain preconditions.

In this paper, we make the same assumptions as [Hertzberg and Horz, 1989]. That is,

(1) All plans are well-formed.
(2) The *locality assumption* holds. That is, every operator must specify all the domain conditions it may change, and every change is independent of the situation before the operator is applied.
(3) The STRIPS assumption holds. That is, conditions change only if mentioned in the effects some operators in a plan.

Although the conflict resolution methods to be introduced below become inadequate without these assumptions, the preprocessing algorithms later in the paper are independent of them.

Conflicts in a Plan

Let P and U be operators in a plan such that $\text{Est}(P, U, p)$. Suppose there is another operator N in the plan such that (i) N can possibly be between P and U, and (ii) $\exists \neg q \in E_N$ such that $\Diamond(q \approx p)$. Then N is called a *clobberer* of p for U, and tuple $\langle P, U, N, p, q\rangle$ is called a *conflict* in the plan. To distinguish from other forms of conflicts in a plan, such as consumable resources, we call the conflicts defined above as *deleted-condition conflicts*. In this paper, we only consider deleted-condition conflicts.

Hertzberg et. al. [Hertzberg and Horz, 1989] have shown that all deleted-condition conflicts in a well-formed plan are compositions of four types of conflicts listed below:

(1) Linear Conflict (LN(P,U,N,p,q)) if N is between P and U.
(2) Left Fork (LF(P,U,N,p,q)) if U is after P and N, and P and N are unordered in the plan.
(3) Right Fork (RF(P,U,N,p,q)) if P precedes both U and N, and the latter two are unodered in the plan.
(4) Parallel Conflict (PR(P,U,N,p,q)) if P precedes U, and N is unordered with both operators.

A plan is *correct*, if its set of operators is partially ordered by the precedence constraints, it is well-formed, and it is free of conflicts.

Conflict Resolution Methods

To resolve a conflict, a planner imposes various kinds of constraints. Chapman [Chapman, 1985] formulated a necessary and sufficient goal achievement criterion, known as the necessary modal truth criterion, or MTC. He also provided a set of sufficient procedural interpretations of the MTC, which includes "promotion," "demotion," "establishments," "separation," and "introducing white knights." These methods can be considered as various alternative constraints one can impose on a plan for resolving a given conflict. A simplified version of the above methods is listed below: Let $\langle P, U, N, p, q\rangle$ be a conflict. Then the following constraints are sufficient for resolving it:

(1) promotion of clobberer: $U \prec N$,
(2) demotion of clobberer: $N \prec P$,
(3) separation: $p \not\approx q$,
(4) demotion and establishment : for some W, where W is either an existing operator in the plan, or an inserted operator, and for some $r \in E_W$, $N \prec W \prec U$ and $r \approx p$.

Now consider how to resolve each type of conflict in a well-formed plan. We use $+$ for logical disjunction, and $\cdot$ for conjunction. Let r_{de} represent the establishment and demotion methods. That is,

$$r_{de} = (N \prec W \prec U) \cdot (r \approx p),$$

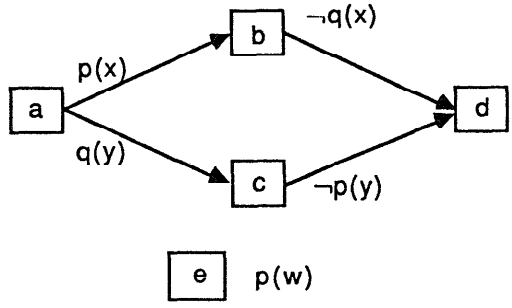

Figure 1: An example plan with two conflicts. The literals $p(x), q(y)$, etc., are the effects of the operator immediately before them.

where $r \in E_W$ and W is an operator. Let R_{de} be the disjunction of all r_{de}. Then

$$\begin{aligned} LN(P,U,N,p,q) &= (p \not\approx q) + R_{de}, \\ LF(P,U,N,p,q) &= (N \prec P) + (p \not\approx q) + R_{de}, \\ RF(P,U,N,p,q) &= (U \prec N) + (p \not\approx q) + R_{de}, \\ PR(P,U,N,p,q) &= LF(P,U,N,p,q) \\ &\quad + RF(P,U,N,p,q). \end{aligned}$$

where '=' means "can be resolved by."

As an example, consider the plan shown in Figure 1. The set of precedence constraints in this plan is:

$$P = \{a \prec b, a \prec c, b \prec d, c \prec d, a \prec d\}.$$

Suppose that $\text{Est}(a, b, p(x))$, and $\text{Est}(a, c, q(y))$, then there are two conflicts in this plan, C_1 and C_2, where $C_1 = RF(a, b, c, p(x), p(y)) = r_{11} + r_{12} + R$, $r_{11} = (b \prec c)$, $r_{12} = (x \not\approx y)$, and $R = r_{13} \cdot r_{14} \cdot r_{15}$, with $r_{13} = (c \prec e)$, $r_{14} = (e \prec b)$, and $r_{15} = (w \approx x)$. $C_2 = RF(a, c, b, q(y), q(x)) = r_{21} + r_{22}$, where $r_{21} = (c \prec b)$, and $r_{22} = (x \not\approx y)$.

Relations Among Conflicts

Above we have shown that all conflicts in a well-formed plan can be resolved using a set of resolution methods, represented in a concise form. If all the conflicts in a plan are found, then each conflict will be automatically associated with a set of alternative resolution methods, and the set of all conflicts can be represented in a conjunction of disjunctive normal form. The purpose of representing the conflict resolution methods is to find one or all constraints that can resolve the conflicts in a plan. Each consistent set of constraints that can resolve all the conflicts is called a solution. Below we consider the relationship between different conflict resolution methods.

Let R_1 and R_2 be two precedence constraints. R_1 *subsumes* R_2, or $S(R_1, R_2)$, if and only if $\{R_1\} \bigcup B(P) \supset R_2$, where $B(P)$ is the set of precedence (co-designation) constraints in P. Likewise for codesignation and non-codesignation constraints.

Intuitively, R_1 subsumes R_2 if imposing R_1 will guarantee that R_2 is also imposed. Thus, R_2 is considered to be weaker than R_1. For example, let $r_1 = (b \prec c)$, and $r_2 = (a \prec d)$. If $\{(a \prec b), (c \prec d)\} \in B(P)$, then $S(r_1, r_2)$. As another example, let $r_1 = (x \approx y)$ and $r_2 = (y \not\approx z)$. If $(x \not\approx z) \in C(P)$ then $S(r_1, r_2)$.

Imposing a set of constraints on a plan may result in an incorrect plan. With the assumptions in this paper, such incorrect plans are signaled by inconsistent constraints of the form $a \prec a$ or $x \not\approx x$, for some operator a and variable x. Below, we use "Fail" for such situations.

Two constraints R_1 and R_2 are inconsistent, or $I(R_1, R_2)$, if and only if

$$\{R_1, R_2\} \bigcup C(P) \bigcup B(P) \supset Fail.$$

Intuitively, R_1 and R_2 are inconsistent if imposing R_1 and R_2 together will result in a contradiction in a plan. For example, if $r_1 = (a \prec b)$, and $r_2 = (b \prec a)$, then $I(r_1, r_2)$. Also consider an example of inconsistent codesignation constraints. If $r_1 = (x \approx y)$ and $r_2 = (u \approx v)$, and $\{(x \approx u), (y \not\approx v)\} \in C(P)$, then $I(r_1, r_2)$.

Having the above definitions, we now can prove the following theorems:

Theorem 1 *If* $S(R_1, R), S(R_2, R')$, *and* $I(R, R')$, *then* $I(R_1, R_2)$.

Intuitively, this theorem says that if the weaker constraints implied by two constraints are inconsistent, then they are inconsistent themselves.

Theorem 2 *If* $S(R_1, R_2), S(R_2, R_3)$, *then* $S(R_1, R_3)$.

This says the subsumption relation is transitive. We also require that S is reflexive. However, the inconsistency relation is not transitive.

Algebraic Rules

The disjunction and conjunction operations can be considered as algebraic operations, with special semantics. For example, the meaning of a conjunction $R_1 \cdot R_2$ in a disjunctive normal form representation of a conflict is that it is sufficient to impose both R_1 and R_2 for removing some conflicts. The meaning of disjunction $R_1 + R_2$ is that either R_1 or R_2 is sufficient for resolving some conflicts. Given the precise meaning of these operations, we can prove that "$\cdot$" and "+" satisfy the rules of boolean algebra, as well as the following set of rules:

Rule 1 *If* $S(R_1, R_2)$ *then* $R_1 \cdot R_2 = R_1$.

Rule 2 *If $S(R_1, R_2)$ and u is any constraint, then $R_1 \cdot u + R_2 = R_2$.*

Rule 3 *If $S(R_1, R_2)$ and $S(R_1, R_3)$, then $S(R_1, R_2 \cdot R_3)$, and $S(R_1, R_2 + R_3)$.*

Rule 4 *If $R = R_1 \cdot R_2$, then $S(R, R_1)$ and $S(R, R_2)$.*

Consider the plan shown in Figure 1. Analysis of the conflicts establishes the following relationships: $I(r_{11}, r_{21})$ $S(r_{12}, r_{22})$, $S(r_{22}, r_{12})$, and $S(R, r_{21})$.

Expanding $C_1 \cdot C_2$ we get

$$
\begin{array}{lll}
 & r_{11} \cdot r_{21} + r_{11} \cdot r_{22} + r_{12} \cdot r_{21} & \\
 & + r_{12} \cdot r_{22} + R \cdot r_{21} + R \cdot r_{22} & \\
= & r_{11} \cdot r_{22} + r_{12} \cdot r_{21} + r_{12} \cdot r_{22} & \\
 & + R \cdot r_{21} + R \cdot r_{22} & (I(r_{11}, r_{21})) \\
= & r_{11} \cdot r_{22} + r_{12} \cdot r_{21} + r_{12} & \\
 & + R \cdot r_{21} + R \cdot r_{22} & (Rule1) \\
= & r_{12} + R \cdot r_{21} + R \cdot r_{22} & (Rule2) \\
= & r_{12} + R \cdot r_{21} + R & (Rule1) \\
= & r_{12} + R & (Rule2)
\end{array}
$$

Thus, using algebraic rules we are able to reduce the number of backtracking points from 6 to 2.

Pruning using CSP Techniques

A set of conflicts can be considered as a constraint network, where each conflict is a node, and the consistency relations between the conflicts are arcs in the network. Each node has a set of values to choose from, each value being an alternate conflict resolution method. Thus, a conflict resolution problem can be considered as a constraint satisfaction problem (CSP). In particular, the goal of a conflict resolution problem in planning is either to find out the set of all consistent values, or to find out just one value.

As in CSP, we would like to enforce arc and path consistency of the network. The degree of constraints in such a network of conflicts can be the size of the network. To see this, consider the following example. Each node N_i in the network contains a primitive precedence constraint $a_i \prec a_{i+1}$, for $i = 0, \ldots, n-1$. Also, node N_n contains a value $a_n \prec a_0$. Thus, if no two a_i are identical, then no proper subset of the set of these values is inconsistent, while the set of values when considered together is. This particular network is constrained by n-ary constraints. It follows that ensuring arc and path consistencies is not sufficient for global consistency in general.

Important difference exists between this particular CSP and a traditional one, because of the existence of subsumption relations between the different values of a node. Using this relation, redundant backtrack points can be quickly discovered, and removed. The set of rules that enable this ability is what we call "pruning rules." Below, we consider path consistency and redundancy removal separately.

Path Consistency

The first type of pruning is the same as a traditional definition for arc and path-consistency in CSP: Let C_1 and C_2 be two nodes in a constraint network. If for some $R \in C_1$ such that $\forall R' \in C_2$ $I(R, R')$ holds, then R can be pruned from C_1.

If all the values of C_1 are pruned,then the network has no solution. This type of pruning is called "inconsistency pruning."

New relations concerning inconsistency can also be obtained when considering groups of nodes greater than two. The most commonly known such algorithms for establishing new relations are the path-consistency algorithms. We present a modified path consistency algorithm based on algorithm PC-2 in [Mackworth and Freuder, 1985]. As in PC-2, this algorithm returns an updated set of inconsistency relations, possibly implemented in a matrix form. The difference here is that upon termination, a list Q of arcs that are modified during the execution of the entire algorithm is returned. This list is used for further removal of redundant values or nodes in the network, and we will discuss this in detail in the next subsection.

The function PC is listed below. In PC, Related-Paths(i, k, j) is a function which returns a set of length-2 paths that might have their consistency affected by a change in the consistency of (i, k, j). Likewise, REVISE$((i, k, j))$ returns true if $I(i, j)$ is modified due to path inconsistency. Both functions are defined in [Mackworth, 1981].

```
Function PC(Q')
begin
  Q := emptyset;
  while Q' is not empty, do
        begin
          select and delete a path (i,k,j) from Q';
          if REVISE((i,k,j)) then
            begin
              Q' := Q' ⋃ Related-Paths(i,k,j);
              Q := Q ⋃ {(i,j)}
            end; (if)
        end; (while)
        return Q
end
```

Redundancy Removal

More pruning can be achieved using subsumption relations among constraints. Note that the redundancy as defined below is caused by the subsumption and the inconsistency relations, instead of the latter alone. Thus, our results here are fundamentally different from that in [Dechter and Dechter, 1977].

Theorem 3 *Suppose $\exists R_2 \in C_2$, such that $\forall R_1 \in C_1$, $S(R_1, R_2)$. Then C_2 can be pruned from the network, without affecting the set of solutions.*

Note that this theorem is different from inconsistent pruning. It says that if some value of C_2 is subsumed by all values of C_1, then C_2 is subsumed by C_1, in the sense that any solution for C_1 must also be a solution for C_2. This type of pruning can be called "subsumption pruning."

Subsumption relations also allow for the removal of individual values.

Theorem 4 *If* $\exists R_2 \in C_2$, *such that* $\forall R_1 \in C_1$, *either*

1. $\exists R_2' \in C_2$, *such that* $R_2 \neq R_2'$ *and* $S(R_1, R_2')$, *or*
2. $I(R_1, R_2)$,

then R_2 *can be pruned from* C_2, *without affecting the solution of the network.*

Removal of nodes or values in some node is called *redundancy removal.* This is different from inconsistency removal. If a node becomes empty after applying the inconsistency pruning rule, then the whole network is inconsistent. In planning, this means that the current plan corresponds to a dead end in the search space. On the other hand, if a node is made empty by applying the subsumption theorems, then it simply means that the removed node is redundant, and has no direct relation with the consistency of the whole network.

Algorithm RR (Redundancy Removal), listed below, is an implementation of the above two redundancy-pruning theorems. In the function RR, the list Q, which is returned at the end, contains length-2 paths which have changes in their domains because of the redundancy pruning. Q will be used for checking more possible inconsistency pruning.

```
Function REVISE-RR((i,j))
begin
   DELETE := false
   for each R ∈ C_i do
       if for all R' ∈ C_j such that
         either I(R', R) or S(R', R'') for some R'' ∈ C_i
           such that R ≠ R'', then
         begin
           delete R from C_i;
           DELETE := true
         end;
       if C_i becomes empty, then
           delete it from the network;
       return DELETE
end

Function RR(Q')
begin
   Q := emptyset;
   while Q' is not empty, do
         begin
           select and delete any arc (i, j) from Q';
           INC := {(k, i) | k ≠ i, k ≠ j}
           if REVISE-RR((i,j)) then
             begin
               Q' := Q' ⋃ INC;
               Q := Q ⋃ Related-Paths((i, j, i))
             end;(if)
         end;(while)
         return Q
end
```

Combining Path Consistency and Redundancy Pruning

When both inconsistency and redundancy pruning are done, the outcome of modifying one relation can possibly affect the status of the other. For example, removing a redundant value from a node can produce further inconsistency, and thus, the two types of pruning have to be used interchangeably. The basic idea is to interleave the two algorithms PC and RR, until no changes can possibly be made. Algorithm PP (Pre-Processing), listed below, achieves this purpose. It can be shown that the algorithm PP has a worst case complexity of $O(k^5 n^3)$, where k is the number of values in a node, and n is the number of nodes in the network.

```
Procedure PP
begin
   Q := {(i, j, k) | ¬(i = j = k)};
   PC(Q);
   if any node is deleted, then return(fail);
   Q := {(i, j) | (i, j) ∈ arcs(G), i ≠ j};
   Q :=RR(Q);
   while Q is not empty, do
         begin
           Q := PC(Q);
           if any node is deleted, then return(fail);
           Q := RR(Q);
         end;
end
```

Example

Now consider again the example given in Figure 1. We start with the set of relations among the constraints: $I(r_{11}, r_{21})$, $S(r_{12}, r_{22})$, and $S(r_{22}, r_{12})$, and $S(R, r_{21})$. Now apply the pruning rules. By Theorem 4, r_{11} can be pruned from C_1. Thus, after updating, $C_1 = r_{12} + R$. By Theorem 3, the node C_2 can be pruned. Thus, what is left is $C_1 = r_{12} + R$. The number of backtracking points for planning is again reduced form six to two. Therefore, using preprocessing techniques, one can simplify the constraint network while avoiding an exponential number of algebraic expansions.

Applying the Algebraic and CSP Techniques to Planning

A typical planning session can be considered as iterations of several steps:

(1) Select a condition to be achieved and some operators for achieving it. Insert these operators into the plan, possibly with certain precedence

and codesignation constraints.
(2) Activate a conflict detection routine, and compute the set of all conflicts introduced by the inserted operators and constraints.
(3) Impose a set of conflict resolution constraints for resolving one or more conflicts. Save the rest of the alternative conflict resolution methods as backtrack points.

After step 2 is done, the conflict algebra can be applied for simplifying the conflict resolution methods. The resultant methods can be represented in a disjunctive normal form. One or more disjuncts can then be chosen to be imposed on the plan, and the rest saved as backtrack points.

Conclusion

This paper proposes to analyze conflicts in a plan in order to reduce the number of backtracking points in a planner's search space. In particular, a set of algebraic rules, together with a set of preprocessing algorithms are presented for simplifying a set of conflict resolution methods. Via subsumption relationship between constraint to be imposed on a plan, the algorithms are able to not only remove inconsistent choices, but also those that are redundant. Furthermore, these algorithms are applicable to a class of constraint satisfaction problems in which subsumption relations are involved.

With the theoretical foundation of this paper, we intend to further explore the amount of analysis of conflicts necessary for a planner's best performance. Such exploration will have to involve a considerable amount of experimentation. We will also look for other kinds of relations in a constraint network, similar to subsumption ones, in order to allow more powerful preprocessing algorithms to be designed.

Acknowledgments. Thanks to Peter Van Beek for many useful comments.

References

[Allen and Koomen, 1983] J. Allen and J Koomen. Planning using a temporal world model. In *Proceedings of the 8th IJCAI*, pages 741–747, 1983.

[Allen, 1984] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.

[Chapman, 1985] David Chapman. Planning for conjunctive goals. AI Technical Report 802, Massachusetts Institute of Technology, 1985.

[Charniak and McDermott, 1985] Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley Publishing Company, 1985.

[Dean and Boddy, 1988] T. Dean and M. Boddy. Reasoning about partially ordered events. *Artificial Intelligence*, 36:375–399, 1988.

[Dechter and Dechter, 1977] A. Dechter and R. Dechter. Removing redundancies in constraint networks. In *Proceedings of the 6th AAAI*, pages 105–109, 1977.

[Dechter and Pearl, 1987] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34, 1987.

[Hertzberg and Horz, 1989] Hertzberg and Horz. Towards a theory of conflict detection and resolution in nonlinear plans. In *Proceedings of the 11th IJCAI*, pages 937–942, Detroit, Michigan, 1989.

[Mackworth and Freuder, 1985] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 125:65–74, 1985.

[Mackworth, 1981] A.K. Mackworth. Consistency in networks of relations. In Webber and Nilsson, editors, *Readings in Artificial Intelligence*, pages 69–78. Morgan Kaufmann Publishers Inc., 1981.

[Sacerdoti, 1977] Earl Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier, 1977.

[Stefik, 1981] Mark Stefik. Planning with constraints. *Artificial Intelligence*, 16(2):111–140, 1981.

[Tate, 1977] Austin Tate. Generating project networks. In *Proceedings of the 5th IJCAI*, pages 888–893, 1977.

[Van Beek and Cohen, 1989] P. Van Beek and R. Cohen. Approximation algorithms for temporal reasoning. Technical Report CS-89-12, Department of Computer Science, University of Waterloo, 1989.

[Vilain and Kautz, 1986] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the 5th AAAI*, pages 337–382, 1986.

[Wilkins, 1988] David Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, CA, 1988.

[Yang and Tenenberg, 1990] Qiang Yang and Josh Tenenberg. Abtweak: Abstracting a nonlinear, least commitment planner. Department of Computer Science, University of Waterloo, Technical Report No. cs-90-09, 1990.

[Yang, 1990] Qiang Yang. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, 6, 1990.