A Theory of Conflict Resolution in Planning

Qiang Yang [*]

Department of Computer Science
University of Waterloo

Abstract

Conflict resolution in planning is the process of constraining a plan to remove harmful interactions that threaten its correctness. It has been a major contributing factor to the complexity of classical planning systems. Traditional planning methods have dealt with the problem of conflict resolution in a local and incremental manner, by considering and resolving conflicts individually. This paper presents a theory of conflict resolution that supports a global consideration of conflicts. The theory enables one to formally represent, reason about and resolve conflicts using an extended framework of constraint satisfaction. The computational advantage of the theory stems from its ability to remove inconsistencies early in a search process, to detect deadends with low computational overhead, to remove redundancies in a search space, and to guide the search by providing an intelligent order in which to resolve conflicts. The paper also presents empirical results showing the utilities of the theory, by investigating the characteristics of problem domains where the theory is expected to work well, and the types of planning systems for which the theory can offer a marked computational advantage.

Address: Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1
Tel: (519) 888-4716. E-mail: qyang@logos.waterloo.edu

[*]The author is supported in part by an operating grant OGP0089686 from the Natural Sciences and Engineering Research Council of Canada, and ITRC: Information Technology Research Centre.

1 Introduction

A central problem faced by classical AI planning is the composition of sets of operators, or plans, to achieve certain specified goals, given the capabilities, the environment and the initial situation of agents. A major obstacle in the composition process is that different parts of a plan may interact in harmful ways. The harmful interactions, which are normally called *conflicts*, can often be removed by further imposing various kinds of constraints onto the plan. The constraint-posting process has been known as *conflict resolution*. The purpose of this paper is to develop a computational theory of conflict resolution in planning, using extended techniques of constraint satisfaction.

1.1 Background

Conflict resolution in planning is a complex computational process. A single plan may contain many different kinds of conflicts, and each conflict in turn may be resolved by several alternative sets of constraints. To make the matter even more complex, these constraints can also interact among themselves in different ways. For example, some constraints cannot be simultaneously imposed onto a plan to avoid creating inconsistency. Others may be redundant in the presence of more powerful ones. Conflict resolution is often a major contributing factor to the complexity of planning.

As an example, suppose that one wants to paint a ceiling as well as a ladder (An example used in [11]). A proposed plan may consist of two parts, one for painting the ceiling and the other for painting the ladder, such that no ordering constraint is imposed between the two parts. If the robot hand can only hold one brush at a time, then a resource conflict occurs between the part of the plan that paints the ceiling, and the part of the plan that paints the ladder, because of their competition for the robot hand. Similarly, if the wet paint from painting the ladder precludes one from climbing up, then another conflict occurs because performing the former negates a precondition of the latter, which requires that the ladder to be dry.

In the above example, a conflict is caused by one operator which can potentially delete a precondition of another operator. The resource conflict can be resolved by painting the ceiling either before or after painting the ladder. And the wet-paint-on-ladder conflict is resolved by painting the ceiling first. The successful resolution of both conflicts involves the recognition of the fact that one cannot paint the ceiling both before and after painting the ladder, and that the ordering constraint of painting the ceiling first, not only resolves the wet-paint-on-ladder conflict, but also resolves the resource conflict. Thus, a consistent solution to resolving both conflicts is to paint the ceiling first.

Because of its importance in planning, methods for conflict resolution were explored early on. Sussman's system Hacker[14] recognized and fixed "bugs," which were certain classes of conflicts. The bugs were fixed using a bag of hacks, which were different types of ordering constraints that could be imposed upon the operators of a plan. Sacerdoti's NOAH[11] used a partial order to represent the structure of a plan, and implemented a more elaborate set of ordering constraints that could resolve different classes of conflicts. A problem of NOAH is that given several alternative choices in the constraints, it commits to one of them, and does not have the ability to backtrack should an inconsistent situation occur later. Tate's NONLIN[15] fixed this problem, and introduced a complete set of alternative ordering constraints that are capable of resolving conflicts in a completely instantiated plan. Recognizing the need to represent resources using variables in a plan, Stefik's MOLGEN[13] and Wilkins' SIPE[17] both could further impose constraints on variable bindings to resolve conflicts. Chapman's TWEAK[2] introduced an additional type of constraint on variable bindings that forces two variables to instantiate to different objects. He also provided a formal language for expressing plans that we will use as a basis for our computational theory.

1.2 Motivation

A major theme of the previous approaches to conflict reasoning is their incremental nature: conflicts are reasoned about one at a time, and in the presence of more than one conflict, no systematic theory exists that can guide the resolution process. Such incremental, local analysis has a number of drawbacks. First, in the presence of a large number of conflicts, the order in which the conflicts are resolved may have dramatic effects on efficiency. One order can lead to a reduction in the size of the search space more than others, but an arbitrarily chosen order may happen to be the worst one. For example, recall that in order to resolve the two conflicts in the painting problem, the resource conflict can be resolved by two alternative constraints, while the wet-paint-on-ladder conflict can be resolved by only one. Since each alternative choice corresponds to a branching point in the search tree, if the resource conflict is chosen first, then four states will be generated in the worst case. But if the wet-paint-on-ladder conflict is chosen first, then only two states will be generated. The difference in savings could be much larger if more conflicts were involved.

A conflict ordering heuristic can be further strengthened by recognizing "redundant" constraints, which exist because some conflict resolution constraints may be stronger than others. For example, for a set of conflicts $\mathcal{C}$, there may exist a subset $\mathcal{S}$ of $\mathcal{C}$ so that once conflicts in $\mathcal{S}$ are resolved, all other conflicts in $\mathcal{C}$ are also resolved. This *subsumption* information can further enable a planner to find a good order for resolving conflicts, since by resolving the subset first, one can avoid resolving all others. In the painting example, if a decision is made to resolve the wet-paint-on-ladder conflict by painting the ceiling first, then the resource conflict is also resolved automatically. A related issue is to use the subsumption information to guide the choices of alternative constraints for resolving each conflict.

Second, a plan in which conflicts cannot be resolved together corresponds to a dead end in a planner's search space. The ability to detect such dead ends early is vital to a planner's efficiency. In many situations, unresolvable conflicts can be detected with low cost when considered together, but may not be obvious when only a single conflict is considered at a time. Thus, a planning system based on the incremental method for conflict resolution may incur expensive computation due to the expansion of plans that eventually leads to dead ends. For example, if painting the ceiling also makes it impossible to further paint the ladder, due to the dripping paint from the ceiling, then a third conflict exists in our painting example. The only way to resolve this new conflict is to paint the ladder before the ceiling. When all three conflicts are considered together, it is apparent that there is no solution; painting the ceiling first will render the ladder unpaintable, while painting the ladder first makes it impossible to use the ladder for painting the ceiling. However, an incremental system may discover this situation after many plan-expansions.

Finally, considering conflicts in a global manner may lead to the discovery that some constraints can never participate in any final solutions. For example, by comparing the constraints for both the resource conflict and the wet-paint-on-ladder conflict, it is easy to discover that one of the ordering constraints for the resource conflict, namely that of applying paint to the ladder first, is not consistent with the only ordering constraint for the other conflict. Therefore, the constraint of applying paint to ladder first will not be part of any solution for resolving both conflicts. Noticing inconsistencies early can enable a planner to produce a smaller search tree.

A natural extension to the previous work, then, is to reason about conflicts in a global manner. This requires a formal representation of the individual conflicts and their resolution methods, an analysis of the relations among different conflicts, as well as the design of reasoning techniques that can facilitate global conflict resolution. We briefly summarize these contributions in the following section.

1.3 An Overview of the Paper

This paper presents a formalization of conflicts and their inter-relations, using an extended framework for solving constraint satisfaction problems (or CSPs). This formalization makes it possible to apply many existing techniques from the CSP area to aid efficient conflict resolution in planning. An additional feature of conflict resolution which makes use of a subsumption relation, extends the existing methods for solving CSPs. The paper also explores the utilities of applying the CSP formalization to conflict resolution, by pointing out where the proposed technique is expected to be most effective. To justify the claims about improved efficiency, experiments have been conducted to show that using the CSP method for global conflict resolution can lead to dramatic improvement in planning efficiency.

To understand conflict resolution from a global viewpoint, one has to be precise about the language for expressing plans, operators and precondition establishments. In Sections 2 and 3, we review TWEAK's plan language, and express a formal representation of conflicts using this language. Then in Section 4, we present our formalization of conflicts and conflict resolution using a CSP framework. Sections 5 to 8 explore the utilities of applying the theoretical results to planning along several dimensions, and present test results. Section 9 concludes the paper.

2 Conflicts and Conflict Resolutions

2.1 Plan Language

A formal account of conflicts and their resolutions requires a precise characterization of the language for expressing plans. We adopt the TWEAK language designed by Chapman[2], because it has been one of the most formal and influential to date. Later in the paper we point out possible extensions to other plan languages.

A plan Π consists of a set of operators, *operators*(Π), and a set of precedence constraints on the operators. Each operator α is defined in terms of a set of preconditions, *Preconditions*(α), and a set of effects *Effects*(α). For simplicity, the initial state I can be represented by a special operator *Init*, where $E_{Init} = I$ and $P_{Init} = \emptyset$. The goal G can likewise be viewed as a special operator *Goal*, with $E_{Goal} = \emptyset$ and $P_{Goal} = G$. These two operators will be an element of every plan Π, such that *Init* precedes every other operator, and *Goal* is preceded by every other operator.

Each plan is also associated with three kinds of constraints:

1. A set of precedence constraints on operator ordering, *Ordering*(Π), that enforces a *partial order* on the operators. Each ordering constraint between operators α and β is denoted $\alpha \prec \beta$.

2. A set of codesignation constraints *Co*(Π) on variable bindings. These constraints enforce an *equivalence relation* on the variables $x_i, i = 1, 2, \ldots$ and constants $C_i, i = 1, 2, \ldots$. Each codesignation constraint between variables (or constants) x_i and x_j is denoted $x_i \approx x_j$.

3. A set of non-codesignation constraints, *Nonco*(Π), that forces two variables to instantiate to different constants. Each noncodesignation constraint between x_i and x_j is denoted $x_i \not\approx x_j$. The noncodesignation constraints enforce a symmetric relation on variables and constants, such that

   (a) if $x_1 \not\approx x_2$ then $x_2 \not\approx x_1$,

   (b) if $x_1 \approx x_2, x_3 \approx x_4$ and $x_2 \not\approx x_4$, then $x_1 \not\approx x_3$.

In addition, constraints on variable bindings can be propagated through the literals that refer to the variables. For example, let l_1 be $P(x_1, x_2, \ldots, x_n)$ and l_2 be $Q(y_1, y_2, \ldots, y_n)$. Then $l_1 \approx l_2$ if and only if $P = Q$ and $x_i \approx y_i, i = 1, 2, \ldots, n$.

A partially ordered and instantiated plan can have more than one instance that satisfies the constraints. Each instance of a plan is a totally ordered set of operators, with all variables bound to constants. Each instance is called a *completion* of the plan. A constraint R *necessarily holds* in the plan if it holds in every completion of the plan[1]. A plan is

[1]We have chosen not to use the modal operators $\Box$ and $\Diamond$ that Chapman has defined in [2], because the meanings of necessity and possibility are already clear from the definitions of partial ordering and equivalence relations.

necessarily correct, if and only if every precondition p of every operator α is true, just before α, in every completion of the plan. A plan may be incorrect because of "conflicts" among its operators, which we formalize below.

First, we define *precondition establishment* (adopted from [19]): an operator E (which stands for *E*stablisher) is said to *establish* a precondition p_U for operator U (which stands for *User*), or $Est(E, U, p_U)$, if and only if

1. $E \prec U$,
2. $\exists e_E \in \mathit{Effects}(E)$ such that $(e_E \approx p_U)$, and
3. $\forall E'$. if $(E \prec E')$, $(E' \prec U)$ and $\forall e_{E'} \in \mathit{Effects}(E')$, then $\neg(e_{E'} \approx p_U)$ and $\neg(e_{E'} \approx \neg p_U)$.

The last condition states that no other operators necessarily between E and U also necessarily assert or deny p_U. This ensures that operator E is the last operator that asserts the precondition p_U. For example, in a plan for going between two rooms, the operator "opening the door" can be thought of as establishing a precondition of the operator "going between the two rooms," as long as no other operators between these two either opens or closes the door. The concept of precondition establishment has been used under different names. For example, it is called "protection intervals" in NONLIN, and causal relations in the systematic planner by McAllester and Rosenblitt[10].

In a partially ordered and instantiated plan, an establishment relation may possibly be undone by some other operators in the same plan. Such situations are called *conflicts* in a plan. For example, in the above example of door-opening establishment relation, if a "closing door" operator is placed between the establisher and the user, then a conflict occurs.

Formally, let E and U be operators in a plan such that $Est(E, U, p_U)$. Suppose that there is another operator C in the plan such that

1. $\neg(C \prec E)$ *and* $\neg(U \prec C)$. That is, C can possibly be between E and U, and
2. $\exists e_C \in \mathit{Effects}(C)$ such that $\neg(e_C \not\approx \neg p_U)$. That is, possibly C denies p_U.

Then C is called a *clobberer* of the establishment relation $Est(E, U, p_U)$.

A clobberer in a plan can be "defeated" by imposing ordering or codesignation constraints on the plan, or by inserting a *white knight* W between the clobberer and the operator U. Intuitively, a white knight is an operator which re-establishes the clobbered precondition p_U, whenever p_U is "threatened" by the clobberer C with its effect e_C. Formally, W is a white knight for $Est(E, U, p_U)$, e_C and C, if and only if

1. C is a clobberer of $Est(E, U, p_U)$,
2. $(C \prec W)$ *and* $(W \prec U)$, and
3. $\exists e_W \in \mathit{Effects}(W)$ such that either $e_W \approx p_U$, or $(e_W \approx \neg e_C)$.

In the door-opening example, if someone reopens the door whenever it is closed, then the operator that reopens the door is a white knight.

The tuple $\langle E, U, C, p_U, e_C \rangle$ is called a *conflict* in a plan Π, if the following conditions hold:

1. C is a clobberer of *Est*(E, U, p_U), and
2. there is no W in Π, such that W is a white knight for *Est*(E, U, p_U), e_C and C.

2.2 Conflict Resolution Methods

Conflicts threaten the correctness of plans. Based on the TWEAK plan language, Chapman [2] formulated a necessary and sufficient goal achievement criterion, known as the necessary modal truth criterion, or MTC, which includes "promotion," "demotion," "establishments," "separation," and "introducing white knights" as methods to make a goal true. This set of methods is sufficient for resolving a set of conflicts, in the sense that any of them can be chosen to resolve a conflict, as long as it is consistent with the existing constraints in the plan. The methods are also necessary because, as shown by Chapman, no other methods are needed.

Let $\langle E, U, C, p_U, e_C \rangle$ be a conflict in Π. Then any of the following constraints are sufficient for resolving it:

(1) Promotion of clobberer: $U \prec C$,
(2) Demotion of clobberer: $C \prec E$,
(3) Separation: $p_U \not\approx \neg e_C$,
(4) Introducing white knights: for some W, where W is either an existing operator in the plan, or a new operator, and for some $e_W \in$ *Effects*(W), $C \prec W \prec U$ and either (a) $e_W \approx p_U$, or (b) $(e_W \approx \neg e_C)$.

Suppose that a plan Π is possibly correct. That is, there is a completion of Π in which every precondition p of every operator U holds just before U. Then from the fact that the above conflict resolution methods are both necessary and sufficient, there is a set of resolution methods, one chosen for each conflict in Π as defined above, that forces the plan Π to be also necessarily correct. This guarantee suggests the following procedure to "fix" a faulty plan: first, find out all conflicts in the plan. Then for each conflict, generate a set of conflict resolution methods. Finally, choose one method from each set and impose the selected constraints onto the plan.

Given an establishment relation *Est*(E, U, p_U), the set of all conflicts can be found in time $O(n^2)$, where n is the total number of operators in the plan. This is because in the worst case $O(n)$ operators have to be examined for clobberers, and for each clobberer, $O(n)$ operators have to be checked to see if they are white knights. If no new operators are to be inserted in Π, then for each conflict, it takes $O(n)$ time to completely generate all four types of resolution methods above, since in the worst case, $O(n)$ operators have to be tested for white knights.

Operator	Preconditions	Effects
getbrush	HandEmpty,Dry($b)	Have(b), ¬ HandEmpty
paintceiling	Have(b),Dry(Ladder),Have(Paint)	¬ Dry(Ceiling),¬ Dry(b), Painted(Ceiling)
paintladder	Have(b),Have(Paint)	¬ Dry(Ladder),¬ Dry(b), Painted(Ladder)
returnbrush	Have(b)	¬ Have(b), HandEmpty
Init		HandEmpty, Dry(Ladder),Dry(lb), Dry(cb),Have(Paint)
Goal	Painted(Ceiling),Painted(Ladder)	

Table 1: Operator definitions for the painting example.

Let Conf = $\langle E, U, C, p_U, e_C \rangle$ be a conflict in Π, and M(Conf) the set of resolution methods for resolving Conf, then the set of alternative methods can be represented as a disjunctive set:

$$M(\text{Conf}) = \{\{U \prec C\}, \{C \prec E\}, \{p_U \not\approx \neg e_C\}\} \bigcup \text{WKs(Conf)}$$

where WKs(Conf) is the set of white knight constraints for Conf. In the actual implementation of this generation process, however, the total number of conflict resolution methods can be reduced by taking into account the structure of the plan. For example, if the three operators E, U, and C are ordered in a linear sequence in a plan, such that the clobberer C is located necessarily between E and U, then only the separation and white knight methods are applicable without violating the existing ordering constraints.

Consider the painting example introduced in Section 1. The plan for painting both the ceiling and the ladder consists of two unordered linear sequences of operators, as follows:

$$Init \prec \textbf{getbrush}(\$cb) \prec \textbf{paintceiling}(Ceiling) \prec \textbf{returnbrush}(\$cb) \prec Goal$$

$$Init \prec \textbf{getbrush}(\$lb) \prec \textbf{paintladder}(Ladder) \prec \textbf{returnbrush}(\$lb) \prec Goal$$

where $cb is a variable which stands for any ceiling brush, and $lb is a variable that likewise refers to any ladder brush. The preconditions and effects of each operator in the plan are shown in Table 1. The conflicts in this plan are listed in Table 2.

The conflict resolution methods are:

$\text{M}(\text{Conf}_a) = \{\{\textbf{getbrush}(\$cb) \prec \textbf{getbrush}(\$lb)\}, \{\textbf{returnbrush}(\$lb) \prec \textbf{getbrush}(\$cb)\}\}$.

$\text{M}(\text{Conf}_b) = \{\{\textbf{getbrush}(\$lb) \prec \textbf{getbrush}(\$cb)\}, \{\textbf{returnbrush}(\$cb) \prec \textbf{getbrush}(\$lb)\}\}$.

$\text{M}(\text{Conf}_c) = \{\{\textbf{getbrush}(\$cb) \prec \textbf{paintladder}\}, \{\$cb \not\approx \$lb\}\}$.

$\text{M}(\text{Conf}_d) = \{\{\textbf{getbrush}(\$lb) \prec \textbf{paintceiling}\}, \{\$cb \not\approx \$lb\}\}$.

$\text{M}(\text{Conf}_e) = \{\{\textbf{paintceiling} \prec \textbf{paintladder}\}\}$.

Conflict	Producer	User	Clobberer	precondition	clobbering effect
Conf$_a$	Init	**getbrush**($\$cb$)	**getbrush**($\$lb$)	HandEmpty	$\neg$HandEmpty
Conf$_b$	Init	**getbrush**($\$lb$)	**getbrush**($\$cb$)	HandEmpty	$\neg$HandEmpty
Conf$_c$	Init	**getbrush**($\$cb$)	**paintladder**	Dry($\$cb$)	$\neg$Dry($\$lb$)
Conf$_d$	init	**getbrush**($\$lb$)	**paintceiling**	Dry($\$lb$)	$\neg$Dry($\$cb$)
Conf$_e$	Init	**paintceiling**	**paintladder**	Dry(Ladder)	$\neg$ Dry(Ladder)

Table 2: Conflicts in the painting example.

3 Relations Among Conflict Resolution Methods

To find one or all of the consistent methods for resolving a set of conflicts in a plan Π, one has to take into account the various kinds of relationships among different constraints. In this section, we define and analyse two kinds of such relations, the subsumption relation and the inconsistency relation.

3.1 Inconsistency Relation

Resolving conflicts involves imposing constraints onto the structure of a plan. Some constraints cannot be imposed together, because they are *inconsistent* with each other. For example, imposing two ordering constraints $\alpha_1 \prec \alpha_2$ and $\alpha_2 \prec \alpha_1$ onto the same plan results in a cycle in operator ordering, which is disallowed in a partial order. Likewise, constraints $x_1 \approx x_2$ and $x_2 \not\approx x_3$ are inconsistent if the variables are already constrained in the plan such that $x_1 \approx x_3$.

We now formally define an inconsistency relation on constraints. For convenience, if $\mathcal{O}$ is a set of ordering constraints, then $TR(\mathcal{O})$ represents the transitive relation corresponding to $\mathcal{O}$. Also, if $\mathcal{CO}$ is a set of codesignation constraints, then $ER(\mathcal{CO})$ is the equivalence relation corresponding to $\mathcal{CO}$.

Definition 3.1 *Let R_1 and R_2 be two sets of conjunctive ordering constraints. R_1 is inconsistent with R_2 in plan Π, or $I_\Pi(R_1, R_2)$, if and only if* $\exists \alpha, \beta \in \text{operators}(\Pi)$ *such that*

$$\begin{array}{ll} (\alpha \prec \beta) & \in \text{TR}(R_1 \bigcup \text{Ordering}(\Pi)) \textit{ and} \\ (\beta \prec \alpha) & \in \text{TR}(R_2 \bigcup \text{Ordering}(\Pi)). \end{array}$$

We can similarly define an inconsistency relation among two variable binding constraints. Let R_1 and R_2 be two sets of conjunctive codesignation and noncodesigation constraints. Let $Co(R)$ be the set of all codesignation constraints in R and $Nonco(R)$ be the set of all noncodesignation constraints in R.

Definition 3.2 *R_1 is inconsistent with R_2 in plan Π, or $I_\Pi(R_1, R_2)$, if and only if there are variables x and y such that*

$$\begin{array}{ll} (x \approx y) & \in \mathrm{ER}(\mathrm{Co}(R_1) \bigcup \mathrm{Co}(R_2) \bigcup \mathrm{Co}(\Pi)) \textit{ and} \\ (x \not\approx y) & \in (\mathrm{Nonco}(R_1) \bigcup \mathrm{Nonco}(R_2) \bigcup \mathrm{Nonco}(\Pi)). \end{array}$$

Inconsistency relations can be extended to sets of conjunctive constraints containing both ordering, codesingation, and noncodesingation ones, in terms of the inconsistency relation of their respective parts. For example, let $R_1 = \{\alpha_1 \prec \alpha_2, x \not\approx y\}$ and $R_2 = \{\alpha_3 \prec \alpha_4, x \approx y\}$. Then $I(R_1, R_2)$ holds. Two constraints R_1 and R_2 are *consistent*, if they are not inconsistent with each other.

3.2 Subsumption Relation

Imposing one set of constraints R_1 may also make another set R_2 of constraints redundant. For example, let $R_1 = \{\alpha_2 \prec \alpha_3\}$, and $R_2 = \{\alpha_1 \prec \alpha_3\}$. If $(\alpha_1 \prec \alpha_2) \in \mathit{Ordering}(\Pi)$, then clearly imposing R_1 makes it unnecessary to further impose R_2.

In general, 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. Formally,

Definition 3.3 *Let R_1 and R_2 be two sets of conjunctive ordering constraints. R_1 subsumes R_2 in plan Π, or $S_\Pi(R_1, R_2)$, if and only if*

$$R_2 \subseteq \mathrm{TR}(R_1 \bigcup \mathrm{Ordering}(\Pi)).$$

Let R_1 and R_2 be two sets of codesignation and noncodesignation constraints.

Definition 3.4 *R_1 subsumes R_2 in plan Π, or $S_\Pi(R_1, R_2)$, if and only if*

(1) $\mathrm{Co}(R_2) \subseteq \mathrm{ER}(\mathrm{Co}(R_1) \bigcup \mathrm{Co}(\Pi))$ *and*
(2) $\forall r = (x \not\approx y) \in \mathrm{Nonco}(R_2), \exists r_1 = (x \approx x'), r_2 = (y \approx y')$ *and* $r_3 = (x' \not\approx y')$ *where* $r_1, r_2 \in \mathrm{ER}(\mathrm{Co}(R_1) \bigcup \mathrm{Co}(\Pi))$, *and* $r_3 \in (\mathrm{Nonco}(R_1) \bigcup \mathrm{Nonco}(\Pi))$.

The last condition says that every noncodesignation constraint of R_2 can be inferred from the noncodesignation constraints of R_1 and Π. As an example, let $R_1 = \{(x \approx y)\}$ and $R_2 = \{(y \not\approx z)\}$. If $(x \not\approx z) \in \mathit{Nonco}(\Pi)$ then $S_\Pi(R_1, R_2)$.

Similar to inconsistency relations, subsumption relations can also be extended to include sets of both ordering and codesignation constraints. For example, let $R_1 = \{\alpha_2 \prec \alpha_3, x_1 \approx x_2\}$ and $R_2 = \{\alpha_1 \prec \alpha_3, x_1 \approx x_3\}$. If constraints $\alpha_1 \prec \alpha_2$ and $x_2 \approx x_3$ already hold in plan Π, then $S_\Pi(R_1, R_2)$ holds.

Given a plan Π, one can establish the subsumption and inconsistency relations between any pair of constraints R_1 and R_2, by computing the transitive and equivalence closures of the ordering and codesignation constraints, respectively, in R_1, R_2 and Π. Both computations take time $O(n^3)$, for n operators in Π.

For convenience, the subscript Π of both I_Π and S_Π relations are droped in situations where it is clear about the plan under consideration.

Because the subsumption relation S is defined via subset relations, it can be easily verified that S is transitive. That is,

Lemma 3.5 *If $S(R_1, R_2)$ and $S(R_2, R_3)$, then $S(R_1, R_3)$.*

In addition, it is also easy to see that the following property holds:

Lemma 3.6 *If $S(R_1, R_2)$, $S(R_3, R_4)$ and $I(R_2, R_4)$, then $I(R_1, R_3)$.*

This lemma states that if two constraints are inconsistent, then stonger versions of the two constraints are also inconsistent. By letting R_3 and R_4 both be R', it holds as a corallary that if $S(R_1, R_2)$, and $I(R_2, R')$, then $I(R_1, R')$.

Subsumption and inconsistency relations have so far been defined between pairs of constraints. Extensions to higher order relations can be naturally made in a similar way. For example, two sets of precedence constraints R_1, R_2 subsume R_3 in Π, if and only if, $R_3 \subseteq TR(\{R_1, R_2\} \bigcup Ordering(\Pi))$.

3.3 Minimal Solutions

Above we have defined a set of constraint R to be consistent with respect to a plan Π, as the condition that imposing R onto Π will not create any cycle in the operator ordering of Π, and will not produce any contradictory codesignation and noncodesignation constraints. If $\mathcal{C}$ is the set of all conflicts in plan Π, and if a consistent set of constraints Sol resolves all conflicts in Π, then Sol is called a *solution* to $\mathcal{C}$. Clearly, if two sets of constraints R_1 and R_2 are inconsistent, then they cannot both be part of a solution.

It is possible to find a certain amount of redundancy in a solution. For example, if $\alpha_1 \prec \alpha_2$ and $x \not\approx y$ both resolve the same conflict Conf, then the set of conjunctive constraints $\{(\alpha_1 \prec \alpha_2), (x \not\approx y)\}$ also resolves Conf. However, the latter is unnecessarily strong, because either conjunct is able to resolve the conflict without the other. Thus, it is possible to reduce a solution Sol to another solution that is in some sense minimal. A *minimal solution Sol'* for $\mathcal{C}$, is a solution for $\mathcal{C}$ such that no proper subset of Sol' also resolves all conflicts in $\mathcal{C}$.

As the previous example illustrates, a solution may have several alternative sets of minimal solutions. If Sol' is a minimal solution, and if $Sol' \subset Sol$, then the constraints in set difference $Sol - Sol'$ are considered redundant, with respect to Sol'. As a conseqence, if R_1 and R_2 are two disjoint subsets of a solution Sol, and if $S(R_1, R_2)$ is true, then removing R_2 from Sol doesn't affect the minimal solution corresponding to Sol. This observation is the basis of a constraint propagation rule presented in the next section.

As we have pointed out in Section 1, if all conflicts in a plan are known, then using a *global analysis* for conflict resolution is more advantageous than considering the conflicts one at a time. A global analysis of conflicts should then take into account both the inconsistency relation and subsumption relation. To do this, an existing problem-solving paradigm, known as constraint satisfaction, provides an ideal framework for conducting such a global analysis. In the next section, we present a formalization of conflict resolution using CSP.

4 Conflict Resolution as Constraint Satisfaction

Constraint satisfaction problems (CSPs) provide a simple but powerful framework for solving a large variety of AI problems. The technique has been successfully applied to machine vision, belief maintenance, scheduling, as well as many design tasks. An overview of the techniques can be found in [6].

4.1 CSP Representations

A CSP can be formulated abstractly as consisting of a set of variables, each variable is associated with a domain of values that can be assigned to the variable. In addition, a set of constraints exists that defines the permissible subsets of assignments to variables. The goal is to find one (or all) assignment of values to the variables such that no constraints are violated. Note that one should not confuse the variables used in the parameters of a planning operator with the variables in a CSP.

As an example, consider the map coloring problem, where the variables are regions that are to be colored. A domain for a variable is the set of alternative colors that a region can be painted with. A constraint exists between every pair of adjacent variables, which states that the pair cannot be assigned the same color. A solution to the problem is a set of colors, one for each region, that satisfies the constraints.

Conflict resolution in planning can be mapped into a CSP in the following manner. For a given plan Π, each conflict Conf_i in Π corresponds to a variable. The domain of Conf_i is the set of alternative conflict resolution methods that are capable of resolving the conflict. The constraints among the variables are defined via the inconsistency relations among different sets of conflict resolution methods. A solution to the CSP corresponds to selecting a set of consistent resolution methods that resolves all conflicts in Π.

An advantage of the mapping from conflict resolution problems to CSPs is that many existing strategies for solving general CSPs can be directly applied to facilitate a global analysis of conflicts. In addition, the existence of subsumption relations among the variables provides new opportunities for simplifying a CSP further than permitted by traditional CSP techniques.

The methods for solving a CSP can be roughly divided into two categories: constraint propagation and heuristically guided backtracking algorithms.

4.2 Propagating Constraints among Conflicts

When two or more variables are considered together, certain implicit constraints among them can be inferred from the explicitly given ones. Consider, for example, a plan containing two conflicts, Conf_A and Conf_B, where the resolution methods have been found out to be

$$M(\mathrm{Conf}_A) = \{\{x \approx y, b \prec c\}\}, \textit{ and } M(\mathrm{Conf}_B) = \{\{x \not\approx y\}, \{c \prec b\}\}.$$

Then from the inconsistency relation $I(x \approx y, x \not\approx y)$ and $I(b \prec c, c \prec b)$, it is clear that the constraint set $\{x \approx y, b \prec c\}$, for Conf_A, cannot be used as part of a solution for solving both conflicts. Therefore, it can be removed from the set of resolution methods for Conf_1 without affecting any solutions. Furthermore, if it is also the only method for resolving Conf_1, then the plan Π corresponds to a dead end; it cannot be resolved by simply imposing constraints.

The above example is an instance of a general procedure known as arc-consistency in CSP. Given a CSP, an arc-consistency algorithm checks every pair X, Y of variables to search for a situation where there is a value V_X for X that is inconsistent with every value of Y. Then V_X can be removed from the domain of X without losing solutions to the CSP. The algorithm AC-3[9] which is based on this idea, ensures that no more values can be further removed as described above. In this case, the CSP is called arc-consistent. If a CSP has n variables, and each has a domain size no more than v, then the time complexity is $O(n^2v^2)$. As a special case, if any variable ends up with an empty domain, then the entire CSP has no solution.

Arc-consistency can be used as a pre-processing routine before a backtracking algorithm is used, or, as we'll explain later, it can also be used during backtracking. Arc-consistency computation considers pairs of conflicts, and is thus more powerful than considering individual conflicts alone. As demonstrated by the above example, an advantage of arc-consistency processing is that dead ends can be found early in many cases. In terms of search, the pruning of inconsistent choices corresponds to a reduction of the branching factor of a planner's search tree. It can also reduce a "thrashing" effect notorious for backtracking problem solving. A thrashing effect occurs when search in different parts of the search space may fail because of exactly the same reason. For example, if Conf_i and Conf_j are not arc-consistent, i.e., if a resolution method R_{ik} for conflict Conf_i is inconsistent with every method for resolving a conflict Conf_j, and if Conf_i is resolved first, then choosing R_{ik} for Conf_i will always result in a failure, which is repeated for every selection of resolution methods for every conflict the planner chooses between Conf_i and Conf_j. However, Arc-consistency can detect and avoid such situation with quadratic time complexity.

4.3 Redundancy Removal via Subsumption Relation

Recall that a resolution method R_1 subsumes R_2, if R_1 can resolve any conflict that R_2 can. Therefore, with respect to a conflict Conf_2 that R_2 can resolve, R_1 is stronger than necessary. Subsumption relations make certain constraints in the solutions to a CSP redundant. Below, we consider two cases in which redundancy can be detected.

Consider a plan Π containing, among others, two conflicts, Conf_1 and Conf_2. Suppose that every set of constraints for Conf_1 subsumes some constraint for Conf_2. Because any solution Sol for resolving all conflicts must also resolve Conf_1, every choice of a resolution method from $M(\text{Conf}_1)$ must also resolve Conf_2. This fact holds even when a constraint chosen from $M(\text{Conf}_2)$ is removed from Sol. Therefore, if Conf_2 is removed from the CSP, the set of minimal solutions to the CSP will not be affected. The argument is summarized in the theorem below.

Theorem 4.1 *Let Π be a plan with a conflict set $\mathcal{C}$. Let* Conf_1 *and* Conf_2 *be two conflicts in $\mathcal{C}$, and let* $M(\mathrm{Conf}_1)$ *and* $M(\mathrm{Conf}_2)$ *be their corresponding sets of conflict resolution constraints. Suppose that*

$$\forall R_1 \in M(\mathrm{Conf}_1), \exists R_2 \in M(\mathrm{Conf}_2) \textit{ such that } S(R_1, R_2).$$

Then Conf_2 *can be pruned from the CSP without affecting the set of minimal solutions to $\mathcal{C}$.*

In this case, Conf_2 is redundant. Formal proofs for this theorem and the next can be found in Appendix A.

Pruning of redundant variables from a CSP reduces the size of the CSP and therefore can lead to improved efficiency in constraint reasoning. Removal of redundancy in the above form only utilizes the subsumption information. When both inconsistency and subsumption relations are considered together, it is also possible to remove individual redundant values from a CSP.

Consider again a plan Π containing two conflicts, Conf_1 and Conf_2. Suppose that there is some constraint set R_2 in $M(\mathrm{Conf}_2)$, such that for every method R_1 in $M(\mathrm{Conf}_1)$, either

1. $I(R_1, R_2)$, i.e. R_1 is inconsistent with R_2; or
2. $\exists R_3 \in \mathrm{Conf}_2$ such that $R_2 \neq R_3$ and $S(R_1, R_3)$. That is, R_1 subsumes some other constraints in $M(\mathrm{Conf}_2)$.

A solution *Sol* for the set of all conflicts in Π must also resolve Conf_1. Since R_1 satisfies the above condition, if R_1 subsumes $R_3 \in M(\mathrm{Conf}_2)$ such that $R_3 \neq R_2$, then it is equivalent to selecting R_3 resolving Conf_2, instead of selecting R_2. On the other hand, if R_1 is inconsistent with R_2, then the solution cannot include both R_1 and R_2 anyway. As a result, no matter what method is chosen for Conf_1, R_2 will not be chosen for a minimal solution. This means that R_2 can be removed from $M(\mathrm{Conf}_2)$ without affecting the set of minimal solutions for resolving all conflicts in Π. This conclusion is summarized in the following theorem.

Theorem 4.2 *Let Π be a plan with a conflict set $\mathcal{C}$. Let* Conf_1 *and* Conf_2 *be two conflicts in $\mathcal{C}$, and let* $M(\mathrm{Conf}_1)$ *and* $M(\mathrm{Conf}_2)$ *be their corresponding sets of conflict resolution constraints. Suppose that* $\exists R_2 \in M(\mathrm{Conf}_2)$, *such that* $\forall R_1 \in M(\mathrm{Conf}_1)$, *either*

1. $I(R_1, R_2)$, *or*

2. $\exists R_3 \in M(\mathrm{Conf}_2)$ *such that* $R_2 \neq R_3$ *and* $S(R_1, R_3)$.

Then R_2 can be pruned from $M(\mathrm{Conf}_2)$ *without affecting the set of minimal solutions to $\mathcal{C}$.*

Removal of redundant variables or values in a CSP is called *redundancy removal.* It can be used to augment a traditional arc-consistency algorithm in the following manner: at each time a pair of variables X, Y are examined in an arc-consistency algorithm, a check is also made to first verify whether Y is redundant using Theorem 4.1. Then a second test using Theorem 4.2 can be made to test whether each value of Y is redundant due to a combined consideration of both inconsistency and subsumption relations. For a given set of constraints, the computations of both inconsistency and subsumption relations have the same time complexity. These relations are computed only once when the CSP is first initialized. Furthermore, the augmented arc-consistency algorithm takes these two relations as inputs, and considers pairs of conflicts for both of them. Therefore, the additional consideration of subsumption reltations in the augmented algorithm increases the complexity of the original algorithm only by a constant factor.

Redundancy-removal can also be extended in a similar manner to augment path-consistency algorithms, which examine and infer inconsistencies within groups of three variables[8]. Such an extension is straightforward, and a detailed description can be found in [18]. We will now illustrate the application of constraint propagation algorithms, and then turn our attention to a consideration of possible applications of heuristically guided backtracking algorithms to global conflict resolution.

4.4 The Painting Example

Consider again the painting problem described in Section 1. The conflict resolution methods for this problem have been formulated in Section 2. The conflicts are listed in Table 2. The conflict resolution process are listed below.

(1) The only choice for Conf_e is inconsistent with the second set of constraints for Conf_a. Therefore, the latter is removed from $\mathrm{M}(\mathrm{Conf}_a)$ by arc-consistency.

(2) After the last step, the only alternative left for $\mathrm{M}(\mathrm{Conf}_a)$ is $\{\mathbf{getbrush}(\$cb) \prec \mathbf{getbrush}(\$lb)\}$, which is inconsistent with the first choice for Conf_b. Thus, due to arc-consistency $\mathrm{M}(\mathrm{Conf}_b)$ is reduced to $\{\{\mathbf{returnbrush}(\$cb) \prec \mathbf{getbrush}(\$lb)\}\}$.

(3) The only remaining constraint for Conf_b now subsumes constraints for Conf_a, Conf_c and Conf_e. Thus, from Theorem 4.1, all three conflicts become redundant in the CSP, and can be removed.

(4) The remaining constraint $\{\mathbf{returnbrush}(\$cb) \prec \mathbf{getbrush}(\$lb)\}$ for Conf_b is inconsistent with the first constraint for Conf_d. Thus, the first constraint can therefore be removed using arc-consistency.

(5) Finally, the CSP contains only two conflicts, Conf_b and Conf_d. The remaining constraints left in $M(\mathrm{Conf}_b)$ and $M(\mathrm{Conf}_d)$ are combined as a global solution to the CSP:

$$\mathbf{returnbrush}(\$cb) \prec \mathbf{getbrush}(\$lb), \$cb \not\approx \$lb.$$

This solution is also a minimal solution for the CSP. The resulting plan is formed by ordering all ceiling-painting operations to be before all ladder painting operations, and making sure that the ceiling-painting brush is different from the ladder-painting brush.

4.5 Heuristically Guided Backtracking Algorithms and Their Extensions

Arc-consistency algorithms may not discover all implicit constraints in a CSP, as it considers only pairs of variables. If three or more sets of constraints are inconsistent, then a problem solver may have to backtrack due to inconsistency among groups of three or more constraints. For example, suppose that there are N conflicts, a resolution method for the i^{th} one is $a_i \prec a_{i+1}$, for $1 \leq i < N$, and $a_N \prec a_1$ for Conf_N. Then although any pair of values for two variables may not contradict with each other, the set of all N constraints will result in a cycle in the plan. Therefore, although arc-consistency can prune many inconsistent values, in general a global constraint management algorithm, such as a backtracking algorithm, has to be used.

A backtracking algorithm instantiates the variables one at a time in a depth-first manner. It backtracks when the constraints accumulated so far signal inconsistency. With both inconsistency and subsumption relations in a CSP, a backtracking algorithm can be guided by the order of variables to be solved, and the order of value assignments to the variables.

Variable ordering corresponds directly to ordering the conflicts to be resolved in a plan. Under this mapping, one useful heuristic is to resolve a conflict with the smallest number of resolution methods first[6]. For example, if $M(\mathrm{Conf}_1)$ has a size of two, and $M(\mathrm{Conf}_2)$ has a size of ten, then this heuristic resolves Conf_1 before Conf_2. A problem with this heuristic is that there may be many conflicts with the same number of resolution methods. Given subsumption relations among the conflicts, a tie-breacking heuristic can be further used to augment the above heuristic by preferring to resolve a conflict which resolution methods subsume a large number of others.

Given an ordering of conflicts, a value ordering heuristic could choose a resolution method that leaves choices for future variable assignments as open as possible[6], and similar to variable ordering, tie-breacking can be further achieved by preferring a value which subsumes the most number of values belonging to the remaining variables.

After each variable assignment, a backtracking algorithm can also propagate constraints through the unassigned variables. A straightforward but powerful method, known as Forward-Checking[6], performs partial arc-consistency by removing from each future variable domain those values that are inconsistent with the current assignment. An extension using subsumption relation can further simplify the remaining CSP, by removing from the network any variable with a domain value subsumed by the current assignment. Specifically, let Conf be the current variable, and *Rem* be the set of remaining conflicts yet to be resolved. If u is chosen to be the instantiation for Conf, then the forward-checking algorithm performs a look-ahead step:

```
for each $\mathrm{Conf}_j \in Rem$ do
        if there is a value $v \in M(\mathrm{Conf}_j)$ such that $(u, v)$ is inconsistent,
        then delete $v$ from $M(\mathrm{Conf}_j)$;
        endif;
        if there is a value $v \in M(\mathrm{Conf}_j)$ such that $u$ subsumes $v$,
        then delete $\text{Conf}_j$ from Rem;
        endif;
endfor;
```

So far we have been concerned with the construction of reasoning tools used for resolving conflicts. We next consider how to integrate these tools with actual planning systems.

5 Planning with Global Conflict Resolution

Classical planning systems often plan in an incremental manner, by repeatedly inserting new operators and resolving conflicts. During each iteration of a planning routine, a few existing or new operators are chosen to establish a precondition or subgoal. Then one or more conflicts are detected and resolved. This process repeats until no more conflicts exist in a plan, and when there are enough operators to establish all preconditions and goals. Some examples are Chapman's TWEAK and the systematic nonlinear planner by McAllester and Rosenblitt[10].

In a simple domain, the number of conflicts introduced and considered in each planning cycle by an incremental method may be very small in number. In blocks world domains, for example, our experience has been that the number of conflicts introduced in each cycle by TWEAK is often about two on the average. Because of the small number of conflicts, it may not appear very effective to apply the CSP conflict resolution method to incremental planning, since many of the supposed advantages of the global analysis may indeed be too small to be noticed: the order in which to resolve the conflicts may not matter much, and dead ends may not occur often enough to justify a global constraint propagation. Furthermore, the utility of our conflict resolution method based on CSP may even become a serious question; although it only takes cubic time to detect conflicts and build a CSP representation, the accumulated amount of effort over the entire search space could surpass its benefit in the long run. Therefore, a complete theory of conflict resolution should also cast a boundary indicating in what kind of domains, and with what type of planning methods, the global analysis is expected to work well.

To address the utility question, we have performed empirical tests of the algorithms. We expected from these tests that the CSP method for conflict resolution will be the most effective when a large number of conflicts could be detected in a plan. In addition, the benefit of doing the global analysis increases with the number of conflicts, relative to an incremental method.

In terms of application domains, a large number of conflicts may occur if the operators are tightly inter-related, and sensitive to operator ordering and variable binding constraints. An example of such domains is where there are *non-serializable subgoals*[5], such that solutions to the individual subgoals must be properly interleaved in order to yield a correct solution. With planning techniques, our expectation leads to the following predictions. First, for planning systems that adopt a problem-decomposition strategy, a global conflict analysis can be benefitial. Examples of problem-decompositon based problem solvers are Lansky's GEMPLAN[7], which has been applied to building-construction domains, Yang, Nau and Hendler's restricted interaction planner[?], which has been tested on metal-cutting problems in automated manufacturing, and Simmons' GORDIUS system [12] which has been applied to geologic interpretation. Problem-decomposition is the process of breaking apart a large and complex problem into several smaller, more or less self-contained parts. A solution can then be found for each individual part concurrently, by constraining problem-solving activities to be forcused on only that part. When the sub-solutions are combined, however, the interactions among the different parts are likely to occur and need to be resolved. It is in this combination phase where the CSP-based conflict resolution method can show a marked difference. Using problem-decomposition, one can generate plans for each individual subproblem from scratch. But one can also rely on problem-dependent problem solvers for providing sub-solutions of decomposed parts, and use the CSP method as a problem-independent routine for combining the solution plans. For example, in a manufacturing domain there is usually a number of specialists who can provide several alternative plans and constraints for sub-problems within their expertise. But when a complex part is to be produced, a domain-independent module can be used profitably for sequencing and resource control[4].

Related to problem-decomposition systems, a second type of planner for which the CSP method may be useful is one that employs a task network hierarchy. A task network planning system starts with a set of subgoals, and reduces each one according to a library of pre-defined networks of sub-plans. Each sub-plan may also contain more detailed subgoals that can be further reduced. The system terminates when every remaining operator in the plan can be successfully executed. Examples of such systems are SIPE, NONLIN, and DEVISER. When these systems are applied to complex domains, each task reduction may introduce many new steps that interact, which may in turn cause a large number of conflicts to occur.

Finally, the CSP method is expected to be useful in *plan revision*, where the input is a used plan that is possibly incorrect, and the output is a modified version of the plan which fits a new situation. For example, in the PRIAR system of Kambhampati and Hendler[3], a previously generated plan is first retrieved. The system then identifies those preconditions of the operators that are no longer established or conflicted in a new situation, and proposes plan-modification operations for reachieving them. The inserted new operators may render the plan possibly incorrect by creating new conflicts, and the number of such conflicts may increase with the number of faults in the original plan. In this case, our CSP method for conflict resolution can fix the remaining conflicts and arrive at a conclusion about the validity of the fix fast.

In the sections that follow, we present empirical, average-case results confirming the hypothesis that a global analysis based on CSP methods is more advantageous than an incremental method in domains where there is a large number of conflicts. We also test the prediction that with a problem-decomposition strategy, the CSP-based method offers dramatic improvement in efficiency. We start by providing a detailed picture of the implementation.

6 Implementation

This section describes the implementation of two planning systems, TWEAK and WATPLAN. Both systems are coded in Allegro Common Lisp on a SUN4/Sparc Station. Care has been taken so that both planners share exactly the same unification and consistency-checking routines.

6.1 TWEAK

TWEAK[2]is implemented as a cycle of two activities: establishing a precondition of an operator, and then resolving all conflicts for that establishment relation. More precisely, it can be specified as the following procedure:

1. Select a plan state from the search frontier. Apply a correctness checking routine (the Modal Truth Criterion) to the plan to verify its necessary correctness. If the plan is correct, then exit with success.

2. Find a precondition p of an operator A such that p is not necessarily true. Find all establishers from the operators in the plan as well as by instantiating new operator schemata in a plan library. For each establisher E, construct a new establishment relation $Est(E, A, p)$ in a copy of the plan.

3. For each successor plan, detect and resolve all conflicts with the new establshment relation. Each alternative set of constraints that resolves the conflicts gives rise to a new successor state.

4. Extend the search frontier of TWEAK by including all resultant successors from the last step. Go to step 1.

This implementation of TWEAK is sound, in that every solution it finds is necessarily correct. With a breadth-first search control strategy, it is also complete in that it will always find a solution in one exists.

An option can also be chosen in TWEAK for performing either depth-first search or breadth-first search. Under the condition that only ordering and variable binding constraints can be imposed, both depth-first and breadth-first strategies guarantee that a correct completion of a plan can be found, if one such completion exists.

6.2 WATPLAN

The theory of conflict resolution has been implemented in a planner we call WATPLAN[3]. Its input is assumed to be a possibly incorrect plan, and it outputs a necessarily correct instance of the plan if one exists. WATPLAN consists of four modules, a conflict detection

[2]Implemented in collaboration with Steve Woods.

[3] *Wat*erloo *Plan*ner

module, a preprocessing module, a variable ordering module, and a backtracking module. Each module is described briefly below.

6.2.1 Conflict Detection

The first module of WATPLAN detects all conflicts with the establishment relations in the plan. It starts by trying to find the an establishment relation for each operator precondition and subgoal. Then it looks for the set of all conflicts with the establishment relations in the entire plan. For each conflict, it proposes a set of resolution methods as outlined in Section 2. The conflicts, together with the conflict-resolution methods, form a CSP that is the basis of the subsequent modules.

6.2.2 Preprocessing

If executed, this module will perform two tasks: using a partial arc-consistency algorithm to check for dead ends and for removing inconsistencies, and using a redundancy-removal algorithm for eliminating subsumed nodes or values in the CSP.

The partial arc-consistency algorithm checks for every pair X and Y of variables in the CSP whether a value of X is inconsistent with every value of Y. If so, then the value is removed from the domain of X. The difference between this algorithm and AC-3 is that it only does one pass over the network, thus, it doesn't re-check the consistency of X with other variables due to an update in X's domain. Although the partial arc-consistency enforcement does not ensure the CSP network to be completely arc-consistent, it does allow significant elimination of inconsistencies. This implementation decision is for the purpose of minimizing the complexity of preprocessing algorithms.

After performing partial arc-consistency, a redundancy-elimination procedure checks every pair of variables X and Y to see if every value of X subsumes some value of Y. If so, then according to Theorem 4.1, Y can be removed from the CSP while keeping the solution set intact. This procedure also records the total number of times a value of a variable X subsumes some values of other variables. The recorded measure will be used in the next module as a variable ordering heuristic.

6.2.3 Variable Ordering

The third module of WATPLAN sorts the variables of the CSP in ascending order of the cardinality of their domains. For variables of the same domain size, an option can be selected to order them in decreasing number of subsumption recordings given by the previous module. The purpose of this ordering process is for the backtracking algorithm to search a small search tree, and discover redundant nodes as soon as possible.

6.2.4 Backtracking

The last module performs depth-first search using the forward-checking algorithm augmented by subsumption pruning. The algorithm is listed in Section 4. An option has also been implemented for the algorithm to find just one solution, or the set of all solutions. Finally, the solution constraints are imposed onto the plan to produce a correct instance.

7 Fixing Incorrect Plans

The first experiment compares the average performance of WATPLAN and TWEAK, on a group of artificially and randomly generated plans that are possibly incorrect. Each test problem contains a user-specified number of randomly generated conflicts among a fixed number of linear sequences of operators that are unordered with each other. Each operator can have preconditions and effects. To avoid cases that favor WATPLAN over TWEAK, trivial plans in which some preconditions don't have any establisher, or plans that contain obvious unresolvable conflicts, have been rejected. Each conflict is created by randomly choosing a clobbering operator that conflicts with a randomly selected establishment relation.

The test problems are designed to simulate an important subclass of planning problems in general, which can be described as follows:

> Given an incorrect plan, impose only ordering and codesgination constraints to make it necessarily correct.

This problem is characteristic of the kinds of problems concerned by plan-reuse systems such as PRIAR, the conflict-resolution components of task-network based planners such as SIPE, and the sub-solution combination phase of any problem-decomposition system.

The first group of data compares the average performance[4] of WATPLAN with TWEAK, with an increasing number of conflicts and increasing size of the plans. Tests were done for plans containing two, four and six unordered linear sequences, where each sequence contains 10 operators. Each operator has two preconditions, and more than two effects. For each plan size, 150 randomly generated plans are generated as inputs to WATPLAN and TWEAK with either depth-first or breadth-first search strategy. For each specific number of conflicts, ten random plans are generated and the test results are averaged.

Test results for plans with 2×10 operators are partitioned into two classes, based on whether a plan contains resolvable conflicts or not. Figure 1 shows the average results of the first class, where conflicts in all plans can be successfully resolved. Figure 2 shows the tests of plans with unresolvable conflicts, with the same plan size as in Figure 1. It is clear from Figure 1 that the average-case complexity of WATPLAN is much lower than TWEAK, and that as the number of conflicts increases, the difference between the two also increases. For plans with unresolvable conflicts, WATPLAN displays a more stable pattern in CPU time, as compared to TWEAK. This can be attributed to the application of partial arc-consistency for global dead end detection in WATPLAN. On the other hand, TWEAK often cannot realize the dead end situation until late in the search process.

[4]In most tests, CPU seconds are used as units of measurements, as opposed to the total number of states explored, since the costs of preprocessing algorithms in WATPLAN do not show up in the number of states expanded.

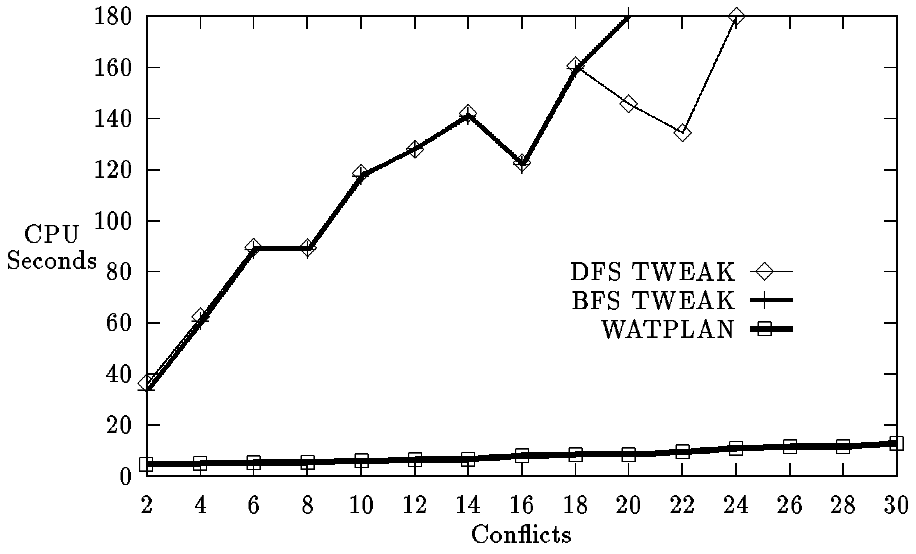

Figure 1: Comparison over successful plans.

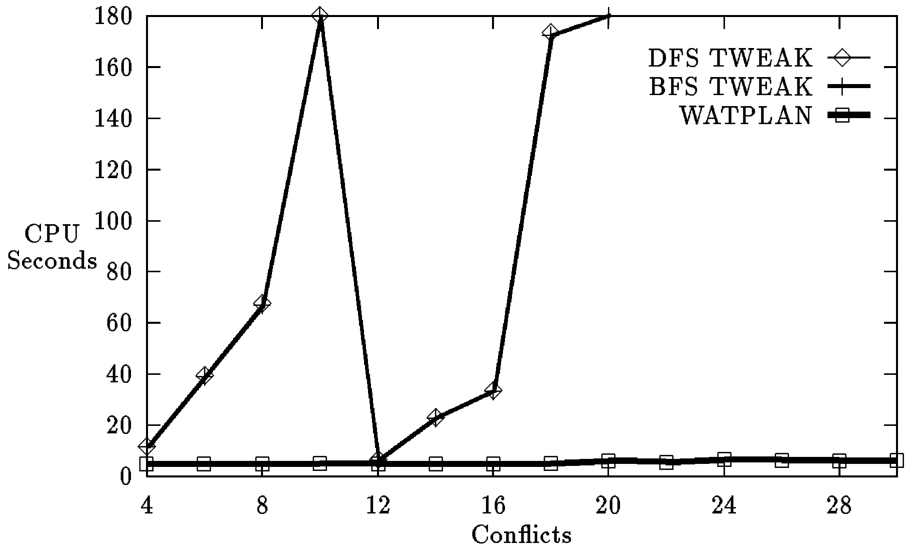

Figure 2: Comparison over failed plans.

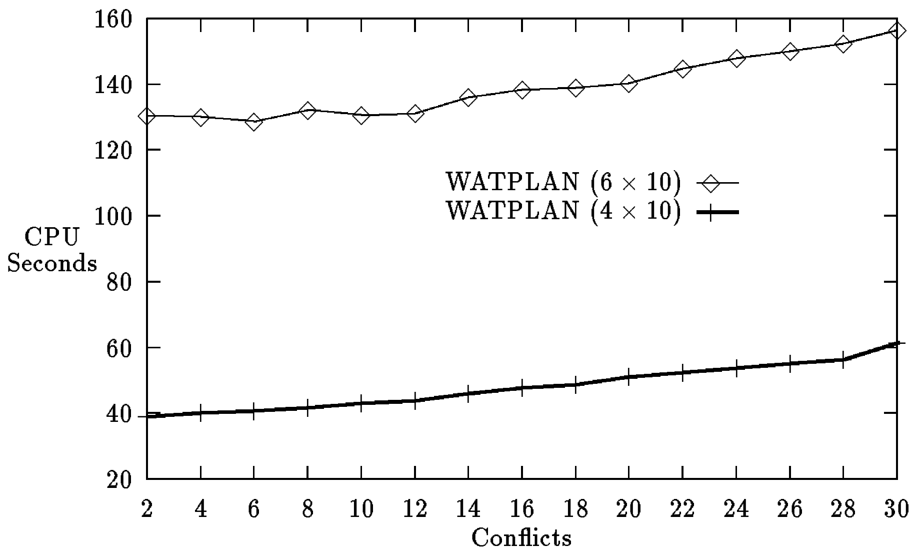

Figure 3: Comparison over plans with large sizes.

With plans containing 4×10 and 6×10 operators, TWEAK cannot finish by a limit of 180 CPU seconds. Thus, only WATPLAN is used for testing these plans. This class of tests is aimed at finding the effects of large plan size on the complexity of WATPLAN. Figure 3 shows the results, where each datum is an average of 10 tests, with no distinction made between successful and unsuccessful plan revision. It is easy to observe the constant amount of increase in average time complexity of WATPLAN when the plan sizes increase. This constant factor is due to the fact that the initial set-up costs — the costs of conflict detection operations — increase with plan sizes in cubic manner. But once set up, the costs for conflict resolution is a function of only the total number of conflicts in the plan.

Our second group of experiments tests the utilities of using the subsumption relation in both preprocessing and backtracking. We expect that subsumption is most useful when the conflicts are tightly coupled, in situtions where a small portion of a plan contains a large number of conflicts. In such cases, it is more likely for some constraints to subsume a large number of others, making it more efficient to impose these constraints first. For example, in an extreme case, there can be one clobberer in a linear branch that creates all conflicts with operators on the other branch. Figure 4 shows comparisons, in CPU seconds, of WATPLAN using and not using subsumption relations in such an extreme case (where the number of preconditions for each operator is three, the plan size is 2×10, and all the conflicts are caused by a single operator). Figure 5 further demonstrates the number of states expanded for each case of WATPLAN. The number of states expanded by WATPLAN using subsumption stays almost constant with increasing number of conflicts, because redundancy removal eliminates almost any need for search in tightly coupled plans. Therefore, our expectation about the utility of subsumption relation holds true.

However, when conflicts are only loosely coupled, using subsumption relation is not very

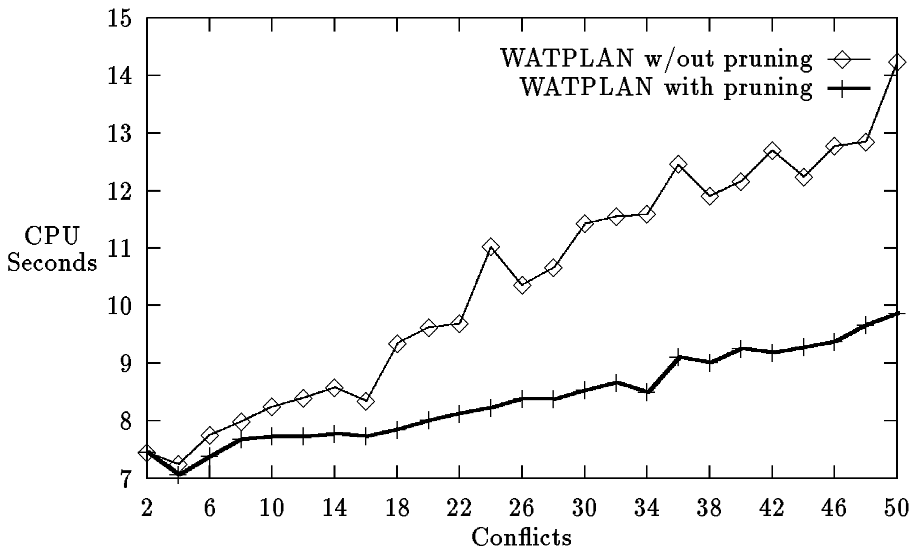

Figure 4: Utility of redundancy pruning (1).

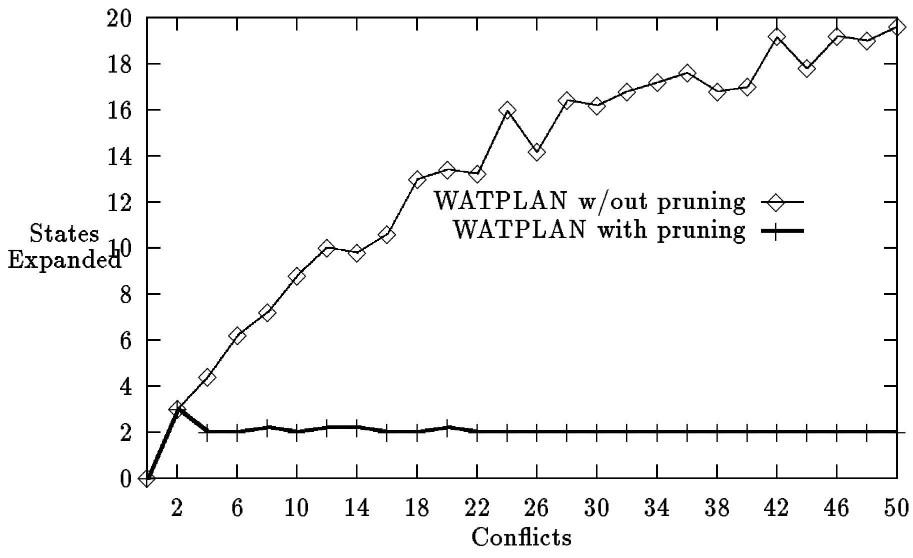

Figure 5: Utility of redundancy pruning (2).

different from not using it. For example, in plans containing six branches and ten operators per branch, there is no observable difference between the two instances of WATPLAN.

To sum up, the results of our experiments can be stated in the following three conclusions:

1. A global conflict processing algorithm is more efficient than an incremental planning algorithm,
2. With the increasing number of conflicts, the relative computational advantage of WATPLAN over TWEAK grows.
3. Subsumption is more useful when conflicts are more tightly coupled in a small portion of a plan.

8 Problem-Decomposition with WATPLAN

As predicted in Section 5, a global analysis of conflicts is expected to be particularly useful for problem solvers that are based on a problem-decomposition strategy. A situation in which problem-decomposition can be profitably applied is where there is enough domain-dependent knowledge for generating solutions to each individual sub-problem, but the conflicts among the sub-solutions need to be resolved when a global solution is formed. WATPLAN is extended to interface with a set of specialists to facilitate this way of problem solving. In particular, it is assumed that an ordered set of alternative solutions has been generated by each specialist within his/her domain. WATPLAN then conducts a systematic selection of the sub-solutions, and applies its conflict-detection, preprocessing, variable-ordering and backtracking algorithms to combine the solutions. If the resultant plan can be made correct, then one such correct plan is returned. Otherwise, it returns to the previous step to select the next set of plans for combination. The process repeats until either there is a succcessful combination, or there is no more new combination to be considered.

To test the efficiency of this strategy, experiments have been done in the blocks world domain, where a planning problem is defined by the initial and final configurations of stacks of blocks on a table. The restrictions are that only one block can be moved at a time, and that a block cannot simultaneously support more than one block. The operators in this domain are **move**(x, y, z), for moving a block x from y to a block z, and **newtower**(x, y) for moving a block x from the top of y to Table.

One way to decompose the blocks world domain is to consider the movement of each block as the task of a specialist. Suppose that a specialist knows exactly how a block x can be moved, in the following manner: From any initial situation On(x, y) to a goal situation On(x, z), every block x is moved in precisely one of the following ways:

1. If $y = z =$Table, then return $\{($**donothing**$)\}$ as the only solution for moving block x. **donothing** denotes an empty sub-plan, in which all goal conditions are established by the initial situation.

2. If $y =$Table but $z \neq$ Table, then return $\{($**move**$(x,$Table$,z))\}$.

3. If $y \neq$ Table and $z =$Table, then return $\{(\textbf{newtower}(x, y))\}$.

4. If $y = z \neq$ Table, then return a set of two alternative solutions:
$\{(\textbf{donothing}), (\textbf{newtower}(x, y) \prec \textbf{move}(x, \text{Table}, z))\}$.

5. Otherwise, return $\{(\textbf{move}(x, y, z)), (\textbf{newtower}(x, y) \prec \textbf{move}(x, \text{Table}, z))\}$.

The above domain-dependent enumeration of the movement of a block completely characterizes its possible movement for any given initial and final situations. Therefore, given a blocks world problem, if there is a plan for all blocks, then a sub-plan exists for each individual block that can be combined to result in a correct one. In other words, WATPLAN is complete for this domain. The difficulty lies in the selection of sub-plans which can be combined to result in a final solution. When more than one block exists a choice made may not only affect the successful movement of one block, but may also make the other blocks' movements either easier or harder, or even impossible in some situations. We illustrate the selection process through the following example.
The initial situation is

On(C,A),On(A,B),On(B,Table),Clear(C),Clear(Table)

and the goal is

On(A,B),On(B,C),On(C,Table).

The initial sub-plans, which are provided by the specialists for the blocks, are listed below.

For	`block A`:	$\{(\textbf{donothing}), (\textbf{newtower}(\text{A}, \text{B}) \prec \textbf{move}(\text{A}, \text{Table}, \text{B}))\}$
For	`block B`:	$\{(\textbf{move}(\text{B}, \text{Table}, \text{C})\}$
For	`block C`:	$\{(\textbf{newtower}(\text{C}, \text{A}))\}$.

The first choice for sub-plan combination includes the sub-plan **donothing** for block A. However, when the three sub-plans are combined, no operator can be found in the plan that establishes the precondition Clear(B) of **move**(B,Table,C). But the second choice for combination, listed below, can be successfully merged.

For	`block A` :	$(\textbf{newtower}(\text{A}, \text{B}) \prec \textbf{move}(\text{A}, \text{Table}, \text{B}))$
For	`block B` :	$\textbf{move}(\text{B}, \text{Table}, \text{C})$
For	`block C` :	$\textbf{newtower}(\text{C}, \text{A})$

In particular, when the three sub-plans are combined, the newly found establishment relations for precondition Clear(A) of **move**(A,Table,B) and precondition Clear(B) of **move**(B,Table,C) require the imposition of ordering constraints

$$\textbf{newtower}(\text{C}, \text{A}) \prec \textbf{move}(\text{A}, \text{Table}, \text{B}),\ \ \textbf{newtower}(\text{A}, \text{B}) \prec \textbf{move}(\text{B}, \text{Table}, \text{C}).$$

Furthermore, the following conflicts are detected:

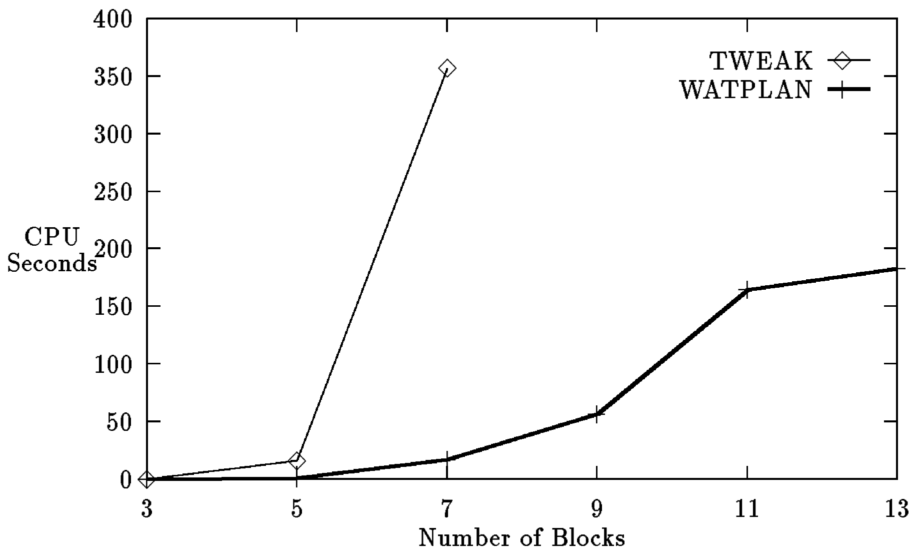

Figure 6: Comparison in blocks world domain.

1. **move**(A,Table,B) is a clobberer for the establishment relation *Est*(*Init*,**move**(B,Table,C),Clear(B)), and

2. **move**(B,Table,C) is a clobberer for *Est*(*Init*,**newtower**(C,A),Clear(C)).

A linear plan is obtained by resolving both conflicts:

newtower(C,A) $\prec$ **newtower**(A,B) $\prec$ **move**(B,Table,C) $\prec$ **move**(A,Table,B).

Tests have been conducted with randomly generated blocks world problems, which are simply randomly generated initial situations for a given number of blocks. For each random problem, a domain-specific routine is first applied to generate the set of alternative movements for each block. Then WATPLAN is applied for selecting sub-solutions and resolving conflicts. To compare with an incremental planner, an additional run is made for each test problem using TWEAK to combine sub-plans and resolve conflicts. The results are shown in Figure 6, where each datum is the average of 10 randomly generated problems for a given number of blocks. This test again demonstrates that with WATPLAN the computational cost of the combination phase is much lower than the incremantal planner, TWEAK.

9 Conclusion

We have described a theory of conflicts and conflict resolution methods in planning. Each conflict is modeled as a variable in a CSP, and the set of conflict resolution methods is modeled as the domain of a variable. Two types of relations are described. The inconsistency relation corresponds directly to its counterpart in CSPs, and the subsumption relation provides new insights into the removal of redundancy values and variables. The formalization supports a number of efficient reasoning tasks, including arc-consistency enforcement, redundancy-removal, dead end detection, and the ordering of conflicts in which to conduct their resolution.

Our empirical results have also revealed that for problems where a large number of conflicts is expected to occur, the theory will work well. In addition, for planning systems that rely on problem-decomposition, that are based on task networks and that perform plan-revisions for reuse, a global reasoning of conflicts based on our theory promises improved efficiency.

Our theory of conflict resolution can be considered as a framework for making inferences between time points in a temporal constraint network. In this respect, it is closely related to Allen and Koomen's work on temporal constraint propagation in planning [1]. In that work, a time interval algebra is used to express the relationship between actions, facts, and goals. When a new temporal relation is added into a plan, constraint propagation is automatically conducted, resulting in temporal relations that are more specific. This is similar to removing inconsistent or redundant relations contained in the variables in our constraint network during conflict resolution. However, as a proposal for a general plan representation language, Allen and Koomen did not focus on any specific control strategy for resolving conflicts in a plan, nor did they consider codesignation and non-codesignation constraints among the variables in a plan. One problem that faced Allen and Koomen's planning system was how to control constraint propagation when a new relation was inserted into a plan, so that only "interesting" inferences were made. Our theory on conflict resolution provides a guidline for controlling the propagation of constraints: propagations should be done only when they are useful in establishing inconsistency or subsumption relations among the conflict resolution constraints.

One advantage of our theory is its extensibility; with a more elaborate planning language, the underlying theory for global conflict resolution need not change. For example, one can extend the TWEAK language to include the time point algebra of Vilain and Kautz[16], by assoicating the occurrence of each action with a time point. One can also extend the TWEAK language to include Allen's interval representation of actions. With Vilain and Kautz's time point logic, the relationships between two time points include "precedes," "follows," "same," and "not-same." With the new language, one can also augment the set of conflict resolution methods by providing an additional set of constraints. For example, suppose that whenever two operators occur simultaneously, one of their combined effects will clobber an establishment relation. Then one way to resolve the conflict is to impose a "not-same" constraint onto the time points of the two operators. This augmentation only enlarges the domain of individual variables that represent conflicts in a CSP, and thus the same computational framework can be directly applied to resolve conflicts in the extended language.

Acknowledgement

The author wishes to thank Josh Tenenberg, Peter van Beek and Jean Patrick Tsang for many useful comments on an earlier version of the paper.

References

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

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

[3] Subbarao Kambhampati. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach.* PhD thesis, University of Maryland, College Park, Maryland, Oct. 1989.

[4] Subbarao Kambhampati, Mark Cutkosky, Marty Tenenbaum, and Soo Hong Lee. Combining specialized reasoners and general purpose planners: A case study. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 199–205, Anaheim, CA, 1991.

[5] Richard Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1985.

[6] Vipin Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, Spring, 1992:32–44, 1992.

[7] Amy L. Lansky. Localized event-based reasoning for multiagent domains. *Computational Intelligence*, 4(4), 1988.

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

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

[10] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 634–639, San Mateo, CA, 1991. Morgan Kaufmann Publishers, Inc.

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

[12] R. G. Simmons. The roles of associational and causal reasoning in problem solving. *Artificial Intelligence*, 53(2-3):159–208, 1992.

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

[14] G.A. Sussman. *A Computational Model of Skill Acquisition.* M.I.T. AI Lab Memo no. AI-TR-297, 1973.

[15] Austin Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 888–893, San Mateo, CA, 1977. Morgan Kaufmann Publishers, Inc.

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

[17] David Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[18] Qiang Yang. Reasoning about conflicts in least-commitment planning. Technical Report cs-90-23, Department of Computer Science, University of Waterloo, Waterloo, Ont. Canada, N2L 3G1, 1990.

[19] Qiang Yang and Josh Tenenberg. Abtweak: Abstracting a nonlinear, least commitment planner. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 204–209, Boston, MA, August 1990.

A Proofs of the Theorems

Theorem 4.1

Let Π be a plan with a conflict set $\mathcal{C}$. Let Conf_1 and Conf_2 be two conflicts in $\mathcal{C}$, and let $M(\text{Conf}_1)$ and $M(\text{Conf}_2)$ be their corresponding sets of conflict resolution constraints. Suppose that

$$\forall R_1 \in M(\text{Conf}_1), \exists R_2 \in M(\text{Conf}_2) \text{ such that } S(R_1, R_2).$$

Then Conf_2 can be pruned from $\mathcal{C}$ without affecting the its set of minimal solutions.

Proof: Let $\mathcal{C}'$ be $\mathcal{C} - \{\text{Conf}_2\}$. We would like to show that every minimal solution to $\mathcal{C}'$ is a minimal solution to $\mathcal{C}$, and vice versa.

Let $Sol_{\mathcal{C}'}$ be a minimal solution to $\mathcal{C}'$. This implies that the constraints in $Sol_{\mathcal{C}'}$ resolve every conflict in $\mathcal{C}'$. Since Conf_1 is a member of $\mathcal{C}'$ and $\mathcal{C}'$ is solved by $Sol_{\mathcal{C}'}$, some constraint $R_1 \in M(\text{Conf}_1)$ must be subsumed by $Sol_{\mathcal{C}'}$. From the assumption that every constraint in Conf_1 subsumes some constraint in Conf_2, there must exist a constraint $R_2 \in M(\text{Conf}_2)$

such that R_1 subsumes R_2. Because the subsumption relation is transitive, $Sol_{\mathcal{C}'}$ subsumes R_2 also. Therefore, $Sol_{\mathcal{C}'}$ is a solution for $\mathcal{C} = \mathcal{C}' \cup \{\text{Conf}_2\}$. Furthermore, $Sol_{\mathcal{C}'}$ must also be a *minimal* solution to $\mathcal{C}$, since otherwise, a proper subset of $Sol_{\mathcal{C}'}$ could solve $\mathcal{C}$ as well as $\mathcal{C}'$, violating the assumption that $Sol_{\mathcal{C}'}$ is a minimal solution to $\mathcal{C}'$. Thus, every minimal solution to $\mathcal{C}'$ must also be a minimal solution to $\mathcal{C}$.

On the other hand, a minimal solution $Sol_{\mathcal{C}}$ to $\mathcal{C}$ is clearly a solution to $\mathcal{C}'$, since $\mathcal{C}'$ is a subset of $\mathcal{C}$. Suppose that it is *not* a minimal solution to $\mathcal{C}'$. Then a proper subset of $Sol_{\mathcal{C}}$ is a solution to $\mathcal{C}'$. This implies that, using the result from the above paragraph, the subset is also a solution to $\mathcal{C}$, violating the assumption that $Sol_{\mathcal{C}}$ is already a minimal solution to $\mathcal{C}$. Therefore, $Sol_{\mathcal{C}}$ must also be a minimal solution to $\mathcal{C}'$. □

Theorem 4.2

Let Π be a plan with a conflict set $\mathcal{C}$. Let Conf_1 and Conf_2 be two conflicts in $\mathcal{C}$, and let $M(\text{Conf}_1)$ and $M(\text{Conf}_2)$ be their corresponding sets of conflict resolution constraints. Suppose that $\exists R_2 \in M(\text{Conf}_2)$, such that $\forall R_1 \in M(\text{Conf}_1)$, either

1. $I(R_1, R_2)$, or
2. $\exists R_3 \in M(\text{Conf}_2)$ such that $R_2 \neq R_3$ and $S(R_1, R_3)$.

Then R_2 can be pruned from $M(\text{Conf}_2)$ without affecting the set of minimal solutions to $\mathcal{C}$.

Proof: Let R_2 be the constraint in $M(\text{Conf}_2)$ that satisfies the condition of Theorem 4.2, and let $M_2{}'$ be $M(\text{Conf}_2) - \{R_2\}$. We would like to show that every minimal solution to the CSP corresponding to $\mathcal{C}$ is a minimal solution to the modified CSP, obtained by removing R_2 from $M(\text{Conf}_2)$, and vice versa.

Let $Sol_{\mathcal{C}}$ be a minimal solution to $\mathcal{C}$. Since $Sol_{\mathcal{C}}$ resolves Conf_1, it must subsume a constraint R_1 in $M(\text{Conf}_1)$. There are two possibilities regarding R_1 and R_2:

1. R_1 is inconsistent with R_2. Then the solution $Sol_{\mathcal{C}}$ cannot include R_2 as a member. Therefore $Sol_{\mathcal{C}}$ must subsume a member of $M_2{}'$ in order to solve Conf_2.

2. R_1 is consistent with R_2. From the condition of Theorem 4.2, R_1 must also subsume some constraint R_3 in $M(\text{Conf}_2)$, where $R_3 \neq R_2$ and $R_3 \in M_2{}'$.

Thus, every minimal solution to $\mathcal{C}$ subsumes a constraint in $M_2{}'$. If there exists a solution to the CSP corresponding to $\mathcal{C}$, then the same solution is also a solution for the modified CSP $\mathcal{C}'$, obtained by removing R_2 from $M(\text{Conf}_2)$. On the other hand, every minimal solution to the modified CSP resolves a conflicts in $\mathcal{C}$, and is clearly a solution to the original CSP as well. □