Structure and Complexity in Planning with Unary Operators

Carmel Domshlak and **Ronen I. Brafman**[1]

Abstract. In this paper we study the complexity of STRIPS planning when operators have a single effect. In particular, we show how the structure of the domain's causal graph influences the complexity of planning. Causal graphs relate between preconditions and effects of domain operators. They were introduced by Williams and Nayak, who studied unary operator domains because of their direct applicability to the control of NASA's Deep-Space One spacecraft. Williams and Nayak's reactive planner can be trivially extended into a polynomial time plan generator in the context of tree-structured causal graphs. In this paper, we treat more complex causal graph structures, such as undirected polytrees, singly-connected networks, and general DAGs. We show that a polynomial time plan generation algorithm exists for graphs that induce an undirected polytree. More generally, we show that a certain relation exists between the number of paths in the causal graph and the complexity of planning in the associated domain.

1 INTRODUCTION

Generating plans in the context of the STRIPS representation language is known to be a difficult (P-SPACE complete) problem [5]. Thus, various authors have explored the existence of more constrained problem classes for which planning is easier. For example, Bylander showed that STRIPS planning in domains where each operator is restricted to have positive preconditions and one postcondition only is tractable. Bäckström and Klein [1] considered other types of local restrictions, but using a more refined model in which two types of preconditions are considered: *prevail* conditions, which are variable values that are required prior to the execution of the operator and are not affected by the operator, and *preconditions*, which *are* affected by the operator. For example, [1] have shown that when operators have a single effect, no two operators have the same effect, and each variable can be affected only in one context (of prevail conditions) then the planning problem can be solved in polynomial time. However, these restrictions are very strict, and it is difficult to find reasonable domains satisfying them.

More recently, Williams and Nayak [3] studied planning problems where all operators affect a single variable, in the context of their work on controlling NASA's Deep-Space One spacecraft. In this context, they defined the notion of a *causal graph* which relates the causal structure of the domain, i.e., how different variables play a role in our ability to affect other variables. A causal graph is a directed graph whose nodes are the domain propositions. An edge (p, q) appears in the causal graph if some operator that changes the value of q has a prevail condition involving p. When the causal graph is a tree, it is easy to determine a serializability ordering over any set of sub-goals, and consequently, obtain a plan in polynomial time.

An important byproduct of Williams and Nayak's work is its demonstration that unary operator domain are of practical interest. Interestingly, unary operator domains show up naturally in another application – answering dominance queries in CP-networks [4].

Our work continues Williams and Nayak's study of unary operator domains, concentrating on the relationship between the domain's causal graph and the complexity of plan generation and plan existence. In particular we prove the following results:

- When the undirected graph induced by the causal graph is singly connected, plan existence and plan generation can be performed in $O(e)$ time (where e is the number of edges in the causal graph).
- When the causal graph is singly connected, plan generation is in NP.
- When the causal graph has more than three paths between two variables, plan generation is NP-hard.
- In general, the complexity of plan generation can be bound by a function of the number of paths within the causal graph.

The rest of this paper is devoted to a more formal presentation of these results and their proofs.

2 COMPLEXITY RESULTS

We now show how, by bounding the structural complexity of the causal graph, we can bound the complexity of plan generation. Recall that we use a propositional language to describe the state of the world, and that our operators are described by a set of prevail conditions – i.e., a set of literals that must hold in a world for the operator to be applicable, a single precondition, and a single post-condition (or effect). The precondition and the post-condition are represented by single literals, one the negation of the other.

2.1 Undirected Polytrees

A polytree is a singly connected graph, i.e., a graph in which there is a single path between two nodes. Here, we consider the case of a causal graph in which there is a single path between every pair of nodes in the induced *undirected* graph. For this class of problems we will present a polynomial time planning algorithm. We will rely on [6]'s formulation of the POP algorithm, and we will assume that the reader is familiar with that algorithm.

Our algorithm proceeds in two stages: First, we perform a forward check step. Following this step, which takes time linear in the size of the input, we can answer the question whether or not a plan exists. If the answer if positive, we run a particular instantiation of the POP

[1] Dept. of Computer Science, Ben-Gurion University of the Negev, P. O. Box 653, Beer-Sheva 84105, Israel, e-mail: {dcarmel, brafman}@cs.bgu.ac.il

algorithm which generates the plan without backtracking in linear time.

The forward checking procedures, described in Figure 1, works as follows: we perform a topological sort of the causal graph and start processing each node (=variable) from top to bottom. At each point, a set of operators is associated with each variable, i.e., the set of operators that can change the value of that variable. Initially, this would be the set of all such operators. Now, for each variable v, we check whether its value in the initial state, v^0, differs from its value in the goal state, v^*. If this is the case and there is no operator transforming v^0 to v^* we return *failure*. If $v^0 = v^*$ then we first check whether there are two operators associated with v that have both its values as effects. If this is the case, then, intuitively, this implies that we can change v's current value and still regain the value needed in the goal state. Otherwise, we mark v *locked* and we extract all operators in which the negation of v's current value appears as a prevail condition – it is clear that we will never be able to apply these operators in a valid plan.

It is apparent that the procedure Forward-Check's running time is linear in the number of operators.[2]

Procedure Forward-Check (Π, Λ, $\mathcal{G}$)

1. Topologically sort all variables $\mathcal{V}$ based on the the causal graph.
2. For each variable $v \in \mathcal{V}$, call Recursive-Locking(Π, Λ, v, $\mathcal{G}$), respecting the above ordering.
3. If all calls to Recursive-Locking return success, then return success. Otherwise return failure.

Procedure Recursive-Locking(Π, , Λ, v_i, $\mathcal{G}$)

1. If $v_i^0 \neq v_i^*$ then
   (a) If no operator A_i^+ in Λ has v_i^* as post-condition, return failure.
   (b) Otherwise return success.
2. If $v_i^0 = v_i^*$ then
   (a) If there are two operators A_i^- and A_i^+ in Λ, that have $\neg v_i^*$ and v_i^* as their post-conditions respectively, return success.
   (b) Otherwise, mark the variable v_i *locked* and remove from Λ all operators that have $\neg v_i^*$ as a precondition or prevail condition. (Note that this requires considering operators affecting the children of v_i only.)

Figure 1. Forward checking procedure

Lemma 1 *Forward-Check returns* success *if and only if a plan exists.*

Clearly, if Forward-Check fails, then no plan exists. To prove the opposite direction we proceed as follows: We define a partial order planning algorithm POP-UPC (partial-order planner for undirected polytree causal graph) and show that it will succeed without backtracking if Forward-Check succeeds. POP-UPC is described in detail in Figure 2. In its description, we assume that a minimal number of operators exists, i.e., if we remove a single operator from the domain, Forward-Check would no longer return success.

Intuitively, POP-UPC works as follows: it maintains a goal agenda sorted based on the causal graph structure: parent variables appear after their descendents. At each point, the next agenda item is selected;

[2] This assumes some appropriate indexing is used. This indexing should be performed once for each planning domain.

Algorithm: POP-UPC ($\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$, `agenda`, Λ)

1. **Termination:** If `agenda` is empty, return $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$
2. **Goal selection:** Let $\langle \vartheta_i, A_{need} \rangle$ be a *rightmost* pair on the `agenda` (by definition, $A_{need} \in \mathcal{A}$ and ϑ_i is one of the preconditions of A_{need}).
3. **Operator selection:**
   (a) If $v_i^0 = v_i^*$ and $\vartheta_i = v_i^*$:
      i. If there are no items on the agenda requiring $\neg v_i^*$ and $A_i^- \notin \mathcal{A}'$, or if $A_{need} = A_i^-$ then $A_{add} = A_i^0$.
      ii. Otherwise, $A_{add} = A_i^+$.
   (b) If $v_i^0 = v_i^*$ and $\vartheta_i \neq v_i^*$ then $A_{add} = A_i^-$.
   (c) If $v_i^0 \neq v_i^*$ and $\vartheta_i \neq v_i^*$ then $A_{add} = A_i^0$.
   (d) If $v_i^0 \neq v_i^*$ and $\vartheta_i = v_i^*$ then $A_{add} = A_i^+$.
4. **Plan updating:** Let $\mathcal{L}' = \mathcal{L} \cup \{A_{add} \stackrel{\vartheta_i}{\rightarrow} A_{need}\}$, and let $\mathcal{O}' = \mathcal{O} \cup \{A_{add} < A_{need}\}$. If A_{add} is newly instantiated, then $\mathcal{A}' = \mathcal{A} \cup \{A_{add}\}$ and $\mathcal{O}' = \mathcal{O} \cup \{A_i^0 < A_{add} < A_i^*\}$ (otherwise let $\mathcal{A}' = \mathcal{A}$ and $\mathcal{O}' = \mathcal{O}$).
5. **Update goal set:** Let `agenda`$'$ = `agenda` - $\{\langle \vartheta_i, A_{need} \rangle\}$. If A_{add} is newly instantiated, then for each of its precondition Q, add $\langle Q, A_{add} \rangle$ to `agenda`$'$.
6. **Threat prevention:** If $A_{add} = A_i^+$, then, for each $A \in \mathcal{A}'$, s.t. $\neg v_i^*$ belongs to the preconditions of A add $\{A < A_{add}\}$ to $\mathcal{O}'$.
7. **Recursive invocation:** POP-UPC ($\langle \mathcal{A}', \mathcal{O}', \mathcal{L}' \rangle$, `agenda`$'$, Λ), where `agenda` is topologically ordered (based on the causal graph with respect to the precondition part of each pair).

Figure 2. POP-UPC algorithm

if it requires achieving some value for v that differs from its initial value, we add an operator to the plan with the desired effect. Otherwise, we need the same value v_* for v as that which appears in the initial state. If no operator was added which has the opposite of v_0 as a prevail condition, we will use the initial state (or in POP terminology, the operator A^0) to achieve this value (i.e., we simply do not change this value throughout the plan). If we added an operator which negates v_0, we must re-establish it, and we add an operator with that effect. No threats arise in POP-UPC, and the ordering constraints are consistent.

Lemma 2 *If Forward-Check was successful then POP-UPC will return a valid plan.*

Proof The Lemma will follow from the following claims:

1. For every agenda item, there exists an operator that has it as an effect.
2. There are no threats in the output of POP-UPC.
3. The ordering constraints in $\mathcal{O}$ are consistent.

(1) The first claim follows from the success of the Forward-Check procedure. It implies that for every variable v if v's initial value differs from its final value, there is an operator for achieving that value. For any other variable, we can always use the initial state as the source of its value. If v's initial and final value are the same and there are no two operators that can change v's value in both directions, then because of the locking mechanism, we will not allow any operator that relies on the value of v that differs from its initial value. Hence, the need for an appropriate precondition will not arise.

(2) Suppose that some operator A_t threatens $A_p \stackrel{\vartheta_i}{\rightarrow} A_c$, i.e.,

- $\mathcal{O} \cup \{A_p < A_t < A_c\}$ is consistent, and
- A_t has $\neg\vartheta_i$ as an effect.

For a given variable v_i, only three operators can have an effect pertaining to v_i: A_0, A_i^+, and A_i^-. POP-UPC forces these operators to be ordered as follows: $A_0 < A_i^- < A_i^+$, so A_c can only be an operator with ϑ_i as a prevail condition. There are two cases to consider: $A_p = A_0, A_t = A_i^-$ and $A_p = A_i^-, A_t = A_i^+$.
Suppose that $A_p = A_0, A_t = A_i^-$. In that case $\vartheta_i = v_i^*$, but the only A_c for which A_0 supplies v_i^* is A_i^-.
Suppose that $A_p = A_i^-, A_t = A_i^+$. In that case, $\vartheta_i \neq v_i^*$, and step 6 guarantees that $A_c < A_i^+$. Hence, again, no threat occurs.

(3) The ordering constraints are consistent if no two operators A_i and A_j are such that $\mathcal{O}$ implies $\{\{A_i < A_j\}, \{A_i < A_j\}\}$. In what follows, A_i will be used to denote an arbitrary operator affecting variable v_i.
First note that each ordering constraint added in Step 4 or Step 6 is between operators affecting a variable and its child (with respect to the causal graph). In particular, if $A_i < A_j$ was added in Step 4 then v_i is a parent of v_j, whereas if $A_i < A_j$ was added in Step 6, v_j is a parent of v_i. In particular, this means that if $A_i < A_j$ is implied by $\mathcal{O}$ then there is a path between v_i and v_j in the undirected graph induced by the causal graph.
Assume, to the contrary that $\mathcal{O}$ implies $A_i < A_j$ and $A_j < A_i$. From the argument above, we know that there is a path between v_i and v_j in the undirected graph induced by the causal graph. By our structural assumption, we know that there is a unique path between v_i and v_j. Thus, the situation is as follows: We have a chain of operators $A_i = A_{i_0} < A_{i_1} < \cdots < A_{i_m} = A_j$ implying $A_i < A_j$, and a chain $A_i = A'_{i_0} > A'_{i_1} > \cdots > A'_{i_{m-1}} > A'_{i_m} = A_j$ implying $A_i > A_j$. Without loss of generality, the internal A'_{i_l} and A_{i_l} are different (otherwise, we can reduce the chain and deduce $A_i < A_{i_l}$ and $A_i > A_{i_l}$).
We know that $A'_{i_1} < A_{i_0} < A_{i_1}$:
(1) If v_{i_0} is a parent of v_{i_1} then $A'_{i_1} < A_{i_0}$ can only stem from Step 6 because $\neg v^*_{i_0}$ is a precondition of A'_{i_1} and $A_{i_0} = A^+_{i_0}$. $A^+_{i_0} < A_{i_1}$ can only stem from Step 4 because $v^*_{i_0}$ is a precondition of A_{i_1}. Hence, A_{i_1} and A'_{i_1} must be different operators. Given the algorithm we must have $A'_{i_1} = A^-_{i_1}$ and $A_{i_1} = A^+_{i_1}$. (Otherwise, we have both $A^+_{i_0} < A^+_{i_1}$ and $A^+_{i_1} < A^+_{i_0}$ which implies conflicting preconditions for $A^+_{i_1}$.)
(2) If v_{i_1} is a parent of v_{i_0} then $A_{i_0} < A_{i_1}$ can only stem from Step 6, $A_{i_1} = A^+_{i_1}$, and $\neg v^*_{i_1}$ is a precondition of A_{i_0}. In that case A'_{i_1} must be $A^-_{i_1}$ and, again A_{i_1} and A'_{i_1} are different.
Continuing with the next variable, v_{i_2} we know that $A'_{i_2} < A^-_{i_1}$ and $A^+_{i_1} < A_{i_2}$. We claim that $A'_{i_2} \neq A_{i_2}$. More specifically, we claim that $A'_{i_2} = A^-_{i_2}$ and $A_{i_2} = A^+_{i_2}$.
First, suppose that v_{i_1} is the parent of v_{i_2}. In that case, $A'_{i_2} < A^-_{i_1}$ is impossible. Hence, v_{i_2} must be the parent of v_{i_1}. From $A^+_{i_1} < A_{i_2}$ we can deduce that $A_{i_2} = A^+_{i_2}$. As in the previous case, the fact that $A'_{i_2} \neq A^+_{i_2}$ follows easily, and hence $A'_{i_2} = A^-_{i_2}$.
Having established that $A'_{i_2} = A^-_{i_2}$ and that $A_{i_2} = A^+_{i_2}$, it is apparent that an inductive argument will allow us to show that for all $n > 0$ we have that $A'_{i_n} = A^-_{i_n}$ and $A_{i_n} = A^+_{i_n}$. This contradicts our assumption that $A_{i_m} = A'_{i_m}$.
■

2.2 Polytrees

In this section we provide an upper bound on the complexity of plan generation when the causal graph is a polytree. In particular, we show that this problem is in NP. However, the question of the exact placement of this problem in the computational complexity hierarchy is left open.

First we make the following observation, upon which we base our proof. The central claim will follow using an induction on the number of variables.

Consider an arbitrary planning problem instance Π with a variable set $\mathcal{V}$, and an operator set Λ. Denote by *Must*(v) the maximal number of times that a variable v *must* change its value in the course of execution of a valid plan for this problem. For the type of problems we deal with, for all variables in $\mathcal{V}$ *Must*(v) satisfies:

$$\textit{Must}(v) \leq 1 + \sum_{Sons(v)} \textit{Must}(u) \tag{1}$$

where *Sons*(v) denote the immediate successors of v in the corresponding causal graph. That is, a variable must change its value at most once for each requested change of its successors (in order to satisfy necessary prevail conditions), and then at most once in order to accept the value requested by the goal state.

Let *MinPlanSize*(Π) denote the size of the minimal plan for the problem Π. Using the *Must* property of the state variables, the following upper bound for *MinPlanSize*(Π) is straightforward:

$$\textit{MinPlanSize}(\Pi) \leq \sum_{v \in \mathcal{V}} \textit{Must}(v) \tag{2}$$

This bound holds for all unary operator domains whose causal graph is acyclic. We will use this bound to prove the following lemma.

Lemma 3 *Plan generation for propositional planning problems in domains whose underlying causal graph is a polytree is in* NP.

Proof In order to prove this claim it is sufficient to show that for any solvable problem instance $\Pi = \langle \mathcal{V}, \Lambda, Init, Goal \rangle$ for domains whose causal graph is a polytree, the length of the minimal (optimal) solution will be polynomial in the size of input. Since the verification of the solution takes time linear in its length, the bound follows. We will show that the length of the minimal solution is less than or equal to n^2, where n is the number of variables in $\mathcal{V}$. Our proof does not rely on the particular initial or goal state, and so we will ignore them, from now on.
The proof is by induction on n, and is based on the previously achieved upper bound on *MinPlanSize*. For $n = 1$, Equation 1 implies that

$$\sum_{v \in \mathcal{V}} \textit{Must}(v) \leq \textit{Must}(v_1) \leq 1 = 1^2.$$

Now, suppose that when $|\mathcal{V}| = n - 1$ then

$$\sum_{v \in \mathcal{V}} \textit{Must}(v) \leq (n-1)^2.$$

Let Π' be some problem instance for which $|\mathcal{V}'| = n$. Suppose that the variables in $\mathcal{V}' = \{v_1, \ldots v_n\}$ are topologically ordered based on the domain's causal graph. Clearly, v_n is a leaf node (i.e., *Sons* $(v_n) = \emptyset$). We will denote by Π the problem instance obtained by removing v_n from the domain, and the corresponding variable set by $\mathcal{V}$. According to Equation 1, for each immediate predecessor v of v_n in the causal graph,

$$\textit{newMust}(v) \leq \textit{Must}(v) + \textit{newMust}(v_n) \leq \textit{Must}(v) + 1$$

where *newMust*(v) denotes *Must*(v) with respect to Π'. Generally, for each variable $v \in \mathcal{V}$,

$$\textit{newMust}(v) \leq \begin{cases} \textit{Must}(v)+1, & if\ there\ is\ a\ path\ from\ v\ \text{to}\ v_n \\ \textit{Must}(v), & otherwise \end{cases} \tag{3}$$

Now,

$$\sum_{v\in\mathcal{V}'} \textit{newMust}(v) \;\leq\; n + \sum_{v\in\mathcal{V}} \textit{Must}(v) \;\leq\; n + (n-1)^2 \;<\; n^2$$

and thus, according to the upper bound on *MinPlanSize*,

$$\textit{MinPlanSize}(\Pi') \leq \sum_{v\in\mathcal{V}'} \textit{newMust}(v) \leq n^2$$

■

Lemma 3 shows that any propositional, "polytree-structured" planning problem, is in NP. Moreover, the size of the minimal solution is bounded by low polynomial in $|\mathcal{V}|$, which does not depend on the size of the whole input, $|\mathcal{V}| + |\Lambda|$. Following subsection will highlight the significance of structural properties in the unary operator planning problems.

2.3 General DAGs

The polytree structure of the causal graph turns out to be crucial for guaranteeing reasonable solution times. As we now show, there are solvable propositional planning problems with an arbitrary acyclic (DAG) causal graph that have minimal solutions of exponential size. Analysis of this class of problems points to the reason for such inherent intractability. This allows us to characterize an important parameter of the causal graph affecting planning complexity and to extend the class of problems which are in NP. However, we also show that most of these restricted problems are NP-complete.

Lemma 4 *Plan generation for STRIPS planning problems with a unary operator domain whose causal graph is acyclic is inherently intractable.*

Proof We prove this claim simply by showing the supporting example. However, we will perform more detailed analysis, postponing the example to the end of the proof. Our analysis is based on the fact that the upper bound for *MinPlanSize*, presented in Equation 2 can be exponential in the size of input. First, we prove this claim, then we show by example that this upper bound can be achieved.
The escalation of the complexity, when the number of variables in $\mathcal{V}$ grows, can be shown by bounding *newMust*(v) using a different method than that of Equation 3. For the problems considered in Lemma 4,

$$\textit{newMust}(v) \leq \textit{Must}(v) + \rho_{v\rightsquigarrow v_n}$$

where $\rho_{v_i\rightsquigarrow v_j}$ denotes the total number of different, not necessary disjoint, paths from v_i to v_j. This means that for a given propositional planning problem with acyclic causal graph, with variables numbered according to a topological sort induced by the causal graph,

$$\textit{Must}(v_i) \leq \sum_{j=i+1}^{n} \rho_{v_i\rightsquigarrow v_j} \tag{4}$$

Thus, the upper bound for *MinPlanSize*, presented in Equation 2 can be exponential in the size of the problem description.

Now we show an example, for which such an exponential upper bound can be achieved. This particular example was used in different context by Bäckström and Nebel in [2]. Consider a propositional planning problem with $|\mathcal{V}| = n$, and *Parents*$(v_i) = \{v_1, \ldots v_{i-1}\}$ for $1 \leq i \leq n$. The operator set Λ consist of $2n$ operators $\{A_1, A_1', \ldots A_n, A_n'\}$ where

$$\begin{aligned} &\textit{pre}(A_i) = \textit{post}(A_i') = 0 \\ &\textit{post}(A_i') = \textit{pre}(A_i) = 1 \\ &\textit{prv}(A_i)[j] = \textit{prv}(A_i')[j] = \begin{cases} 0 & \text{if } j < i-1 \\ 1 & \text{if } j = i-1 \end{cases} \end{aligned}$$

Easy to see that the causal graph of this problem forms a DAG, and an instance of this planning problem with the initial state $\langle 0, \ldots, 0\rangle$ and the goal state $\langle 0, \ldots, 0, 1\rangle$ have a unique minimal solution of length $2^n - 1$ corresponding to a Hamilton path in the state space.
■

The analysis of the proof of Lemma 4 was performed in order to point out the reason for this potential exponential escalation of the solution's size. An immediate conclusion of Lemma 4 is that there is a significant class domains with an acyclic causal graph for which planning is in NP.

Definition 1 *A causal graph is called* δ-path-restricted *if the number of different paths between every two nodes is bounded by* δ.

Lemma 5 *Plan generation for (STRIPS unary operator) planning problems with an underlying causal graph that is* δ*-path-restricted is in NP.*

Proof Based on the observations above

$$\textit{MinPlanSize}(\Pi) \leq \delta n^2$$

Again, we have found a class of planning problems that is in NP. But is it NP-hard? The following Lemma shows that in most cases, this is indeed the case.

Lemma 6 *Plan generation for propositional, 4-path-restricted planning problems is* NP*-complete.*

Proof The proof is by polynomial reduction from 3-SAT to the corresponding propositional, 4-path-restricted plan generation problems. 3-SAT is the problem of finding a satisfying assignment to a propositional formula in conjunctive normal form in which each conjunct (clause) has at most three literals.
Let $\mathcal{F} = C_1 \wedge \ldots \wedge C_n$ be a propositional formula belonging to 3-SAT, and let and $X_1, \ldots, X_m$ be the variables used in $\mathcal{F}$. An equivalent propositional, 4-path-restricted planning problem can be constructed as follows:

- $\mathcal{V} = \{A, B, X_1, \ldots, X_m, C_1, \ldots, C_n\}$
- *Parents*(A) = *Parents*$(B) = \{\emptyset\}$
- *Parents*$(X_1) = \ldots =$ *Parents*$(X_m) = \{A, B\}$
- *Parents*$(C_i) = \{A, B, X_{i_1}, X_{i_2}, X_{i_3}\}$, where X_{i_1}, X_{i_2}, and X_{i_3} are the variables that participate in the ith clause of $\mathcal{F}$.
- $Init$ - consist of false assignments to all variables in $\mathcal{V}$ ($\bar{v}$ for each $V \in \mathcal{V}$).
- $Goal$ - consist of true assignments to all variables in $\mathcal{V}$ (v for each $V \in \mathcal{V}$).

Let every operator $A \in \Lambda$ be presented as a three-tuple $\langle \{pre\}, \{post\}, \{prv\} \rangle$ of pre-, post-, and prevail conditions respectively. Then, the corresponding operator set Λ is specified as follows:

$$
\begin{array}{lll}
\Lambda_A & = \{ & \langle \{\bar{a}\}, \{a\}, \{\}\rangle \quad \} \\
\Lambda_B & = \{ & \langle \{\bar{b}\}, \{b\}, \{\}\rangle \quad \} \\
\vdots & & \\
\Lambda_{X_i} & = \{ & \langle \{\bar{x_i}\}, \{x_i\}, \{\bar{a}, \bar{b}\}\rangle, \\
 & & \langle \{\bar{x_i}\}, \{x_i\}, \{a, b\}\rangle \quad \} \\
\vdots & & \\
\Lambda_{C_i} & = \{ & \langle \{\bar{c_i}\}, \{c_i\}, \{\bar{a}, b, \alpha_1^i\}\rangle, \\
 & & \ldots \\
 & & \langle \{\bar{c_i}\}, \{c_i\}, \{\bar{a}, b, \alpha_{k_i}^i\}\rangle, \\
 & & \langle \{\bar{c_i}\}, \{c_i\}, \{a, \bar{b}, \alpha_1^i\}\rangle, \\
 & & \ldots \\
 & & \langle \{\bar{c_i}\}, \{c_i\}, \{a, \bar{b}, \alpha_{k_i}^i\}\rangle \quad \} \\
\vdots & &
\end{array}
$$

where $\alpha_1^i, \ldots \alpha_{k_i}^i$ are all possible truth assignments on the variables $X_{i_1}, X_{i_2}, X_{i_3}$, that satisfy the ith clause of $\mathcal{F}$. Easy to see, that the described planning problem have all propositional variables, single-effect operators and an acyclic causal graph (see figure 3).

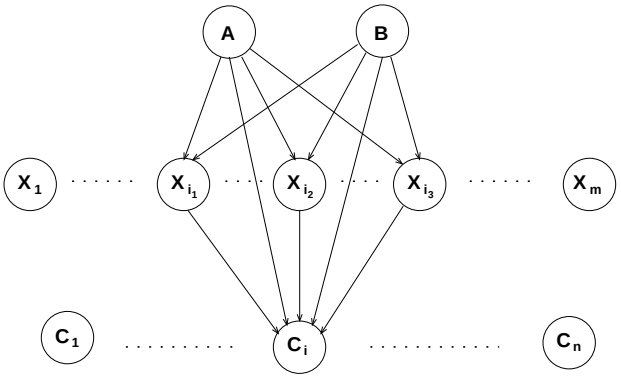

Figure 3. Causal graph of 3SAT satisfaction planning problem

First, we note that the resulting graph has at most 4 directed paths between any pair of nodes. Clearly, there are no paths between nodes A and B; exactly one path between A and B and any of the X_i's; at most one path between a particular X_i and a particular C_j; and exactly 4 paths between A or B and any of the C_i (one direct link, and three passing through the 3 variables consitituting C_i).
Now we describe the dynamics of the problem. As long as $A = \bar{a}$, and $B = \bar{b}$, each X_i can change its value from $\bar{x_i}$ to x_i, and no C_j can change its value. After either the value of A is changed to a, or the value of B is changed to b, no X_i can change its value (the assignment on the formula's variables is locked), but some variables from $C_1, \ldots, C_n$ can change their values from $\bar{c_i}$ to c_i. Finally, after the remaining variable from $\{A, B\}$ changes to its positive value, (i.e. the assignment on $\{A, B\}$ become a, b), each X_i, that still has the value $\bar{x_i}$ can be changed to x_i, but no C_j can change its value (the truth values of the formula's clauses depends only on previously locked values of the formula's variables).
Clearly, $Goal$ is reachable (Π is solvable) if and only if a satisfying assignment for $\mathcal{F}$ can be found. Likewise, the maximal number of paths between pairs of vertices in the causal graph is achieved between each locking variable (A and B), and each clause variable ($C_1, \ldots, C_n$), and is equal to 4. Thus, plan generation for propositional, 4-path-restricted planning problems is NP-hard, and from Lemma 5, we know that it is NP-complete.
■

3 SUMMARY

We have shown that the structure of the causal graph for unary operator STRIPS domains is an important factor in determining the computational complexity of plan generation. In particular, we have shown that a polynomial time algorithm exists for graphs in which there is at most one undirected path between nodes, and that in polytrees the maximal plan length is a low order polynomial. More generally, we have shown a relation between the number of path between variables in the causal graph and the computational complexity of the corresponding planning problem.

ACKNOWLEDGEMENTS

We would like to thank Samir Genaim for his assistance in one of the proofs, and the anonymous referee for useful remarks.

REFERENCES

[1] Christer Bäckström and Inger Klein, 'Planning in polynomial time: The SAS-PUBS class', *Computational Intelligence*, **7**(3), 181–197, (Aug 1991).
[2] Christer Bäckström and Bernhard Nebel, 'Complexity results for SAS^+ planning', *Computational Intelligence*, **11**(4), 625–655, (1995).
[3] B.C.Williams and P.P.Nayak, 'A reactive planner for model-based execution', in *Proceedings of the Fifteen International Joint Conference on Artificial Intelligence (IJCAI–97)*, (August 1997).
[4] C. Boutilier, R. Brafman, H. Hoos, and D. Poole, 'Reasoning with Conditional Ceteris Paribus Preference Statements', in *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–99)*, (1999).
[5] Tom Bylander, 'The computational complexity of propositional STRIPS planning', *Artificial Intelligence*, **69**(1-2), 165–204, (1994).
[6] Daniel S. Weld, 'An introduction to least commitment planning', *AI Magazine*, (Summer 1994).