

Structure and Complexity in Planning with Unary Operators

Carmel Domshlak and Ronen I. Brafman

Department of Computer Science
Ben-Gurion University of the Negev
P.O. Box 653, 84105 Beer-Sheva, Israel
dcarmel, brafman@cs.bgu.ac.il

Abstract

Unary operator domains – i.e., domains in which operators have a single effect – arise naturally in many control problems. In its most general form, the problem of STRIPS planning in unary operator domains is known to be as hard as the general STRIPS planning problem – both are PSPACE-complete. However, unary operator domains induce a natural structure, called the domain’s *causal graph*. This graph relates between the preconditions and effect of each domain operator. Causal graphs were exploited by Williams and Nayak in order to analyze plan generation for one of the controllers in NASA’s Deep-Space One spacecraft. There, they utilized the fact that when this graph is a tree, a serialization ordering over any subgoal can be obtained quickly. In this paper we conduct a comprehensive study of the relationship between the structure of a domain’s causal graph and the complexity of planning in this domain. On the positive side, we show that a non-trivial polynomial time plan generation algorithm exists for domains whose causal graph induces a polytree with a constant bound on its node indegree. On the negative side, we show that even plan existence is hard when the graph is a singly connected DAG. More generally, we show that the number of paths in the causal graph is closely related to the complexity of planning in the associated domain. Finally we relate our results to the question of complexity of planning with serializable subgoals.

Introduction

One of the first well formulated problems addressed by AI researchers was the planning problem. Simply stated, it involves the generation of a sequence of system transformations, taken out of a given set of system transformations (called *actions* or *plan operators*), whose combined effect is to move the system from some given initial state into one of a set of desired goal states. The planning problem is known to be intractable in general (Chapman 1987), and tractable algorithms exist for very restrictive classes of problems only. This discouraging fact has not deterred planning researchers. Indeed, many researchers believe that real-world problems have some properties, or *structure*, that could be exploited, either implicitly or explicitly. In this paper we attempt to

understand the relationship between structure and complexity in planning problems in which each action changes the value of a single variable.

To study the relation between the structure and the complexity in a class of problems we must identify a set of parameters that characterize it. In the case of planning, a number of structural properties have been studied in the past (which we review in more detail in the next section). These structural properties have been mostly local, i.e., they involve restriction on the structure of operators e.g., the type and number of preconditions or effects that operators have. For example, Bylander (Bylander 1994) showed that STRIPS planning in domains where each operator is restricted to have positive preconditions and one postcondition only is tractable. Bäckström and Klein (Bäckström & Klein 1991b) 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, they 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.

In this paper we concentrate on more global properties of unary operator domains; properties that capture some of the interactions between different planning operators. The tool we use to study these properties is the domains’ *causal graph*. A causal graph is a directed graph whose nodes stand for the domain propositions. An edge (p, q) appears in the causal graph if and only if some operator that changes the value of q has a prevail condition involving p . Such a problem structure was introduced by Knoblock (Knoblock 1994) in the context of automatic generating abstractions for planning. Subsequently, in (Jonsson & Bäckström 1995), Jonsson and Bäckström introduced the 3S class of planning problems with unary operators, which was characterized by the acyclicity of the causal graph, and some restrictions on the operator set. It was shown that determining plan existence for this class of problems is polynomial, while plan generation is provably intractable.

Complexity results for unary operators would be of theoretical interest alone if one could not supply interesting problems in which unary operators are used. One interesting application in which this problem arises is the determination of dominance relationship between different outcomes in a CP-network (Boutilier *et al.* 1999). This problem is reducible to STRIPS planning with unary operators.

Another example, of greater interest to the planning community, is a planning-based reactive control system that commands the NASA Deep Space One autonomous spacecraft (Pell *et al.* 1997; Williams & Nayak 1996; 1997). This system was recently hailed by Weld in his recent survey of AI planning (Weld 1999) as one of the most exciting recent developments in the area of planning. Naturally, the complete system (Pell *et al.* 1997) is very complex, however, its configuration planning and execution subsystem are of particular interest to us. In the context of controlling Deep-Space One, Williams and Nayak (Williams & Nayak 1996; 1997) present a reactive planner, Burton, that generates a single control action for the main engine subsystem of the spacecraft, and compensates for anomalies at every step. Given a high-level goal (for example, thrust in one of the engines), Burton continually tries to transition the system toward a state that satisfies the desired goal. What is particularly relevant for us is that Burton's task can be described as a STRIPS planning problem in which each operator affects only a *single* variable (hardware component) – Williams and Nayak (Williams & Nayak 1997) argue that in physical hardware it is usually the case that each state variable is commanded separately. However, Burton is based on two additional important restrictions: First, the planner is explicitly supplied with a serialization order for any satisfiable set of goals. Second, all operators must be reversible.

One of the reasons cited for designing Burton as a reactive planner that generates a single action at a time was the potential intractability of generating whole plans. Indeed, Williams and Nayak were pessimistic about the prospects of generating whole plans quickly even for Burton, i.e., for problem instances with serializable sub-goals and single-effect operators. As our results show, this pessimism was not fully justified.

Our work continues the study of planning with unary operators. This apparently easier problem is in fact as hard as the general STRIPS planning problem (Bylander 1994). However, we can obtain finer distinctions and some positive results if we pay closer attention to the causal structure of the domain. For example, it is easy to show that 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. In this paper we analyze 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 causal graph forms a polytree (the induced undirected graph is acyclic), and its node indegree is bounded by a constant, then plan existence and plan generation are polynomial.
- When the causal graph is singly connected (there is at

most one directed path between any pair of nodes), then plan existence is NP-complete.

- When the causal graph is a general DAG, plan generation is provably intractable, i.e., the problem is in EXPTIME.
- In general, the complexity of plan generation can be bounded by a function of the number of paths within the causal graph.

Finally, we relate our results to an old open question: how difficult is it to generate plans for problems with *serializable subgoals* (Korf 1987)? This question was stated by Bylander in (Bylander 1992), and different hypothesis were raised by different researchers. Here, we present a clear, though somewhat disappointing answer.

The rest of this paper is organized as follows: First, we describe some related work on structure and complexity in planning, discuss our motivation for exploiting the causal graph structure, and provide a short review of the POP algorithm. Second, we present our results on the relation between the form of the causal graph and the complexity of the planning problem. Third, we discuss the sub-goal serializability issue and the impact of our results on it. The full proofs were omitted because of space limitations, but they are available in the technical report (Domshlak & Brafman 2001). This technical report is available online and contains all the missing details.

Background

Related Work on Complexity of Planning

The idea of analyzing and exploiting structural properties is not new to classical planning, and in the last few years a number of important results have emerged. Generating plans in the context of the STRIPS representation language was shown by Bylander (Bylander 1994) to be PSPACE-complete. Despite this fact, the existence of many successful planning systems, especially in recent years, demonstrates that planning is possible and practical for a wide list of domains. Bylander argues that the large gap between the theoretical hardness of planning and its practical success stems from the use of domain-dependent problem analysis and algorithms. Consequently, various authors have explored the existence of some constrained problem classes for which planning is easier. We believe that many “natural” planning problems are likely to have special structure and special properties that can be utilized, implicitly or explicitly, to generate plans quickly. This work is part of our effort to characterize structural properties of planning problems that impact the complexity of plan generation.

Bylander (Bylander 1994) presents a number of complexity results for propositional STRIPS planning, analyzing different planning problems based on the type of formulas used, the number and type of operator preconditions and postconditions, etc. For example, he shows that STRIPS planning in domains where each operator is restricted to have positive preconditions and one postcondition only is tractable. Generally, extremely severe restrictions on operators are required to guarantee tractability, or even membership in NP. Note that Bylander focuses on *local* properties of

operators, i.e., properties of single operators, and he leaves global properties of planning problems open for future work.

Bäckström and Klein (Bäckström & Klein 1991a; 1991b) consider other types of local restrictions, but using a more refined model (the SAS formalism) 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 *pre-conditions*, which are affected by the operator. Hence, prevail conditions, such as having a visa, are needed in order to apply an operator, such as Enter-USA, but their values do not change after the operator is applied. For example, (Bäckström & Klein 1991a) have shown that when operators have a single effect (*unary*), no two operators have the same effect (*post-uniqueness*), and each variable can be affected only in a unique context of prevail conditions (*single-valuedness*) then the planning problem can be solved in polynomial time. However, as argued in (Erol, Nau, & Subrahmanian 1995), the properties of SAS required for tractability or NP membership are so strict that it is hard to find domains satisfying them. The most severe restriction for practical applications is *single-valuedness*. For example, we could not model a problem in which one operator, affecting a particular engine, can be applied only when a certain valve is open, and another operator, affecting the same engine, can be applied only when the same valve is closed. For a thorough analysis of the complexity of SAS planning, see (Bäckström & Nebel 1995).

In (Jonsson & Bäckström 1995), Jonsson and Bäckström present the 3S class of planning problems. This class is most closely related to the problems examined in this paper, since it defines a special subclass of problems with unary operators and an acyclic causal graph. This subclass is defined by posing some, relatively severe, restrictions on the problem's operator set: Each proposition p in 3S problem instance is required to be either (i) *static*, i.e., unchangeable; (ii) *symmetrically reversible*, i.e., for each operator o affecting p , there exist an operator o' affecting p with the same prevail conditions and the opposite effect; or (iii) *splitting*. For the formal definition of the splitting property we refer to (Jonsson & Bäckström 1995). Informally, if a proposition p is splitting then the problem instance can be split into three, well-defined subproblems that can be solved independently. For this class of planning problems it was shown that plan existence can be determined in polynomial time, while plan generation is provably intractable.

In (Jonsson & Bäckström 1998), Jonsson and Bäckström analyze another class of structural properties by examining *domain transition graphs* which describe possible transitions between different values of the same multi-valued (non-propositional) variable. The domain-transition graph of a state variable v is a directed labeled graph $G_v = (V, E)$, where V is associated with the v 's set of possible values, and $(x, o, y) \in E$ if and only if the operator o can be applied at some state in which $v = x$, and its application results in a state in which $v = y$ holds. Domain transition graphs are less local in nature because they combine the influence of many operators. However, they do not capture the relationship between different variables.

Jonsson and Bäckström identify sets of structural restrictions on domain-transition graphs which make planning instances tractable. Roughly, the properties are the following: (1) The problem domain is *interference-safe*, i.e., each operator is either unary or irreplaceable w.r.t. (with respect to) every variable it affects. An operator o is *irreplaceable* w.r.t. a variable v if the removal of all edges from G_v that stem from o disconnects some connected component of G_v . (2) For every variable v , the graph G_v , restricted to the set of values that appear in the prevail conditions of some operators, is acyclic. (3) Any sequence of operators annotating a path from x to y in the domain-transition graph of v , is stronger than the shortest such sequence connecting x and y . Here, a sequence o_1, \dots, o_k is stronger than o'_1, \dots, o'_l if there is a subsequence o_{i_1}, \dots, o_{i_l} of o_1, \dots, o_k such that for every $1 \leq j \leq l$, the prevail conditions of o'_j are a subset of the prevail conditions of o_{i_j} . Jonsson and Bäckström present a map of the computational complexity of problems with different restrictions, displaying the frontier between the tractable and intractable cases.

Finally, the work of Domshlak and Dinitz on multi-entity off-line coordination (Domshlak & Dinitz 2001) can be seen as investigating connection between the structure of the causal graph and the complexity of the corresponding problems in case of multi-valued domains.

Causal Graphs

Causal graphs were used in (Williams & Nayak 1997) as a tool for describing the structure of planning domains with unary operators. They represent a dependence relation between the state variables in the domain. A causal graph \mathcal{G} is a directed graph whose nodes correspond to domain variables. An edge (p, q) appears in the causal graph if and only if some operator that changes the value of q has a prevail condition involving some value of p . Hence the immediate predecessors of q in \mathcal{G} are all those variables that affect our ability to change the value of q . The causal graph is an intuitive model which is easily constructed given any planning problem. The structural properties of the causal graph investigated in this paper can be verified in low polynomial time in the size of the graph.

Causal graphs have an important potential role in the design of autonomous industrial systems, as argued and demonstrated in (Williams & Nayak 1997): Unary operators are natural when the manipulated objects are hardware components, since the basic control actions in such systems change the state of a single hardware component. The applicability of these control actions in any state depends on the state of the affected component as well as on the state of the related hardware components. This naturally gives rise to a planning domain with unary operators. Moreover, since the state variables correspond to hardware components, in the induced causal graph we typically see that the prevail dependencies between variables are usually implicitly entailed by the inter-composition of the hardware components. Thus, the causal graph of such domains resembles the structure of and the relationships between the system's hardware components. This resemblance has important practical ramifications for system design given the relationship between

causal graph structure and the complexity of plan generation: It enables the system designer to consider the effect of his hardware design on the system's ability to autonomously generate control sequences.

A case in point is the planning problem studied by Williams and Nayak (Williams & Nayak 1997), which had a number of important features: all operators were unary and reversible, and the causal graph was acyclic. Williams and Nayak argued that acyclic connectivity frequently occurs in designed systems. However, the requirement that all operators should be reversible seems to us restrictive, and it has important impact on the complexity of the problem. In the case of the Burton planner, there were good reasons to make this assumption. Burton's reactive nature precludes extensive deliberation on the consequences of its operators. Thus it leaves open the possibility that operators may degrade the system's capabilities, leading it to dead-ends. In that case, the restriction to reversible operators was required in order to achieve a more reliable system. As we show later, in certain cases, complete plans can be generated efficiently even when the operators are not reversible.

Williams and Nayak's work has another interesting aspect, as noted by Weld (Weld 1999). For a long time, researchers have known that planning problems with serializable subgoals are likely to be easier to solve. Williams and Nayak recognized that their spacecraft configuration task was serializable (many real-world problems are not), and, more importantly, they developed a fast algorithm for computing the correct order based on the fact that the underlying causal graph is acyclic. However, their algorithm makes heavy use of the fact that all operators are reversible. Informally, reversibility implies that we can solve our subgoals one by one as long as they are consistent with some topological order of the causal graph without taking into account any global considerations: any side-effect can always be undone. Without the assumption of operator reversibility, it is relatively easy to show that Williams and Nayak's algorithm works only if the causal graph forms a directed chain. Even when the causal graph is a tree, although the problem is easy, one must take care in the choice of which subgoal to achieve next when operators are not reversible. As we show later, when the structure of the causal graph is more complicated than a directed tree then either the problem is hard or, if not, a more sophisticated algorithm is required.

Finally, we note that the existence of reversible operators might make the problem seem easier than it actually is. In this paper we present an example of a propositional planning problem with unary operators, acyclic causal graph, and totally reversible operators, the minimal solution of which is exponentially long in the size of the problem's description.

A Short Review of POP, Causal Links and Threats

Our algorithm for problems with polytree causal graphs is a specialized, deterministic instance of the POP algorithm (Weld 1994). Therefore, here we provide a short review of the POP formalism and notation.

We represent a plan as a tuple: $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$, where \mathcal{A} is a set of unary operators, \mathcal{O} is a set of ordering constraints over \mathcal{A} , and \mathcal{L} is a set of causal links. For ex-

ample, if $\mathcal{A} = \{A_1, A_2, A_3\}$ then \mathcal{O} might be the set $\{A_1 < A_3, A_2 < A_3\}$. These constraints specify a plan in which A_3 is necessarily the last operator, but do not commit to a particular order on A_1 and A_2 . Naturally, the set of ordering constraints must be consistent, i.e., there must exist some total order satisfying them. A causal link has the form $A_p \xrightarrow{\vartheta_i} A_c$, where A_p and A_c are operators and ϑ_i is a possible value for some propositional variable v_i . It denotes the fact that A_p produces (i.e., has the postcondition) $v_i = \vartheta_i$ which is consumed by A_c (i.e., used to satisfy a pre- or prevail-condition of A_c). Causal links help us detect whether one operator A_t interferes with enabling execution of some other operator A_c . In that case, A_t is said to constitute a *threat* to one of A_c 's causal links. Formally, suppose that $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$ is a plan, and $A_p \xrightarrow{\vartheta_i} A_c$ is a causal link in \mathcal{L} . Let A_t be a different operator in \mathcal{A} . We say that A_t *threatens* $A_p \xrightarrow{\vartheta_i} A_c$ when the following two criteria are satisfied: (i) $\mathcal{O} \cup \{A_p < A_t < A_c\}$ is consistent, and (ii) A_t has $\neg\vartheta_i$ as an effect. When a partial order plan P contains threats, it is possible that the goal will not be achieved by some (or all) of the total order plans consistent with P 's ordering constraints. To prevent this, the plan generator must check for threats and remove them by adding one of two possible ordering constraints: $A_t < A_p$ or $A_c < A_t$.

A tutorial introduction to POP algorithms can be found in (Weld 1994). POP is a regressive framework for partial order planning that starts with the null plan and continuously updates it by inserting new actions and removing threats. This process continues until the precondition and the prevail conditions of every operator in the plan are supported by some causal link and no threats exist. The first argument to POP is a plan and the second argument is an agenda of goals that need to be supported by causal links. Each item on the agenda is represented by a pair $\langle \vartheta_i, A \rangle$ where ϑ_i is either pre- or prevail condition of a plan action A . The last argument to POP is the whole collection of the operators defined by the planning instance. The initial call to POP contains the null plan, the specially initialized agenda, and the operator set Λ of the given problem.

In this paper we introduce a specialized, *deterministic* POP algorithm that starts the planning process using a variant of the null plan which encodes the planning problem. In particular, if the planning instance has v_1^*, \dots, v_n^* as the goal then the corresponding null plan has exactly $2n$ dummy unary operators, $\mathcal{A} = \{A_1^0, \dots, A_n^0, A_1^*, \dots, A_n^*\}$, n ordering constraints, $\mathcal{O} = \{\{A_1^0 < A_1^*\}, \dots, \{A_n^0 < A_n^*\}\}$, and no causal links, $\mathcal{L} = \{\}$. A_i^0 is the **start_i** operator - it has neither pre- nor prevail conditions, and its effect specifies the value of the variable v_i in the initial state, which is denoted by v_i^0 . A_i^* is the **end_i** operator - it has no effect, no prevail conditions, but its precondition is set to the value of v_i in the goal state, which in turn is denoted by v_i^* .¹ Our description of the null plan is modified from that

¹Actually, the goal state may not specify the values of all the variables, thus the number of the end operators can be less than n . However, for clarity of presentation, we leave this definition of the null plan.

of (Weld 1994) to suit the restriction to unary operators better. Likewise, the initial call to our POP algorithm contains the agenda $\{\langle v_1^*, A_1^* \rangle, \dots, \langle v_n^*, A_n^* \rangle\}$.

Polytree Causal Graphs

Starting at this section, we show how, by bounding the structural complexity of the causal graph, we can bound the complexity of plan generation. We use a propositional language (binary variables) to describe the state of the world. Each operator is described by its prevail conditions, single precondition, and single effect (or post-condition). The prevail condition is a set of literals that must hold in a world for the operator to be applicable. They are not affected by the operator. The precondition and the effect are two literals, one the negation of the other. Finally, we represent a planning instance Π by a four-tuple $\langle \mathcal{V}, \Lambda, Init, Goal \rangle$ in which \mathcal{V} is a set of propositional variables, Λ is a collection of operators (or actions) over \mathcal{V} , $Init$ is an initial state, and $Goal$ is a goal state.

A causal graph forms a *polytree* if there is a single path between every pair of nodes in the induced *undirected* graph. For this class of problems we present a planning algorithm which is polynomial if the indegree of all nodes in the causal graph is bounded by a constant.

Given a propositional planning instance with a polytree causal graph, we:

1. Bound the number of times that a variable may be required to change its value on a valid, irreducible plan.
2. Using this upper bound, provide a polynomial time procedure that determines the *maximal achievable* number of *possibly required* value changes for a given variable.
3. Provide a preprocessing algorithm that determines whether or not a plan for a given problem instance of our class exist. This algorithm is based on a top-down execution of the previously defined procedure on the variables of the given problem instance.
4. If the answer of the plan existence check is positive we run a particular *deterministic* instance of the POP algorithm, called POP-PCG, that generates the required plan, without backtracking, in linear time.

First we make the following observation upon which we base our algorithm. Given a planning instance Π , denote by $\text{MaxReq}(v)$ the maximal number of times that a variable $v \in \mathcal{V}$ changes its value in the course of execution of a valid, irreducible plan for Π . Informally it means that if a plan for Π contains more than $\text{MaxReq}(v)$ changes of v then it contains some redundant steps that can be removed from the plan. Observe that, for any planning problem with unary operators, a variable must change its value at most once for each required change of its immediate successors in the causal graph (in order to satisfy the necessary prevail conditions), and then at most once in order to obtain the value requested by the goal state. Thus for all variables in \mathcal{V} , $\text{MaxReq}(v)$ satisfies:

$$\text{MaxReq}(v) \leq 1 + \sum_{\text{succ}(v)} \text{MaxReq}(u) \quad (1)$$

where $\text{succ}(v)$ denote the immediate successors of v in the corresponding causal graph. The following lemma shows that if the causal graph forms a singly connected DAG then we can bound $\text{MaxReq}(v)$ by n .

Lemma 1 *For any solvable problem instance Π with a singly connected causal graph over n variables, for any variable v , we have that:*

$$\text{MaxReq}(v) \leq n$$

The proof is obtained by a relatively straightforward induction. The main point of establishing such bounds is that using bounds on MaxReq , we can bound the number of value changes, and thus, the size of any plan without redundant steps.

Recall that $\text{MaxReq}(v)$ stands for an upper bound on the number of value changes of v that may be required by a valid, irreducible plan. However, the maximal achievable number of value changes of v , denoted by $\text{MaxPoss}(v)$ can be greater or less than $\text{MaxReq}(v)$. For example, if v has no predecessors in the causal graph, and there are two operators affecting v differently, then $\text{MaxPoss}(v) = \infty$.

We denote the upper bound on the *feasible* number of value changes of v that may be required in a valid, irreducible plan for Π by $\text{FMaxReq}(v)$. Informally, no more than $\text{MaxPoss}(v)$ value changes of v *can* be required and no more than $\text{MaxReq}(v)$ value changes of v *should* be required, thus

$$\text{FMaxReq}(v) = \min(\text{MaxPoss}(v), \text{MaxReq}(v)) \quad (2)$$

Determining $\text{FMaxReq}(v)$ for all variables requires explicit examination of a given problem instance. Recall that here we restrict the causal graph of Π to form a polytree. Denote by v^0 and v^* the initial and the goal values of v in Π , and by $\Lambda_v \subseteq \Lambda$ the set of all operators affecting v . First we examine the root variables of the causal graph, then we analyze the rest of the variables.

Denote by $\text{pred}(v)$ the immediate predecessors of v in the causal graph. If $\text{pred}(v) = \emptyset$, then there are at most two operators A_v^-, A_v^+ in Λ_v : A_v^+ has v^* as its postcondition, while A_v^- has the reverse effect. Since these operators have no prevail condition, if both A_v^- and A_v^+ are presented in Λ , then they can be applied one after another an infinite number of times. Therefore, from Eq. 2, $\text{FMaxReq}(v) = n$. If $\Lambda_v \neq \{A_v^-, A_v^+\}$ then we have two cases: If the initial and the goal values of v are the same, then we cannot change the value of v and reconstruct it later, and thus $\text{FMaxReq}(v) = 0$. Alternatively, if the initial and the goal values of v are different then if $\Lambda_v = \{A_v^+\}$ then we can achieve the goal value of v but only once and thus $\text{FMaxReq}(v) = 1$. Otherwise, the goal value of v is unachievable, thus the given problem instance is unsolvable.

Now consider a variable v which is presented by an internal node in the causal graph: $\text{pred}(v) = \{w_1, \dots, w_k\} \neq \emptyset$. Observe that the number of possible value changes of v depends on and only on:

1. The initial and the goal values of v , i.e., v^0 and v^* .
2. The set of operators affecting v , i.e., Λ_v .

3. The maximally possible (but still reasonable) number of times that predecessors of v can change their values, i.e., $\text{FMaxReq}(w_1), \dots, \text{FMaxReq}(w_k)$.
4. The actual scheduling of the value changes of the predecessors of v .

The last point is important – it means that in order to determine $\text{FMaxReq}(v)$ we should find a particular scheduling of the value changes of $\text{pred}(v)$ that allows such a maximal number of value changes for v . The corresponding interleaving sequence of v 's values, starting and finishing by v^0 and v^* respectively, with $\text{FMaxReq}(v)$ value changes will be called *maximal* and will be denoted by $\sigma(v)$ ($|\sigma(v)| = \text{FMaxReq}(v) + 1$).

From Lemma 1, for $1 \leq i \leq k$, $\text{FMaxReq}(w_i) \leq n$, thus the number of different orderings of value changes of $\text{pred}(v)$ can be greater than n^{nk} . Clearly, we cannot check all these orderings in a naive manner. Following, we provide an algorithm that determines $\sigma(v)$ in polynomial time.

For clarity of presentation we want to distinguish between the different elements of a maximal sequence $\sigma(v)$. Since all variables are binary, for clarity of presentation, we denote the initial value of v , v^0 , by b_v and the opposite value by w_v (black/white). Similarly, b_i and w_i will stand for the corresponding values of the variable v_i . If so, we can think about all the operators in Λ as described in this language. Likewise, we sequentially number the appearances of each value of v on $\sigma(v)$. For example, b_v^i stands for the i th appearance of the value b_v along $\sigma(v)$.

Given the maximal sequences $\sigma(w_1), \dots, \sigma(w_k)$ and the operator set Λ_v we construct a directed labeled graph $G(v)$ which is defined as follows:

1. $G(v)$ consist of η nodes, where

$$\eta = \begin{cases} n, & ((n = 2j) \text{ and } (v^0 = v^*)) \text{ or} \\ & ((n = 2j + 1) \text{ and } (v^0 \neq v^*)), j \in \mathbb{N} \\ n - 1, & \text{otherwise} \end{cases}$$

2. $G(v)$ forms a *2-colored multichain*, i.e., (i) the nodes of the graph are colored by black and white, starting by black; (ii) there are no two subsequent nodes with the same color; (iii) for $1 \leq i \leq \eta - 1$, edges from the node i are only to the node $i + 1$.

Observe that such a construction of $G(v)$ promises that the color of the last node will be consistent with v^* .

3. The nodes of $G(v)$ are denoted precisely by the elements of the maximal sequence $\sigma(v)$, i.e., b_v^i stands for the i th black node in $G(v)$.
4. Suppose that there are m operators in Λ_v that change the value of v from b_v to w_v . In this case, for each i , there are m edges from b_v^i to w_v^i , and $|\Lambda_v| - m$ edges from w_v^i to b_v^{i+1} . All edges are labeled by the prevail conditions of the corresponding operators, i.e., a k -tuple of the values of w_1, \dots, w_k . This tuple is denoted by $l(e)$ (label of the edge e) and its component, corresponding to a predecessor w_i , is denoted by $l(e)_{w_i}$.

This formal definition of $G(v)$ is relatively complicated, thus we provide a demonstrating example: Suppose that we

are given a problem instance over 5 variables, and we consider a variable v with $\text{pred}(v) = \{u, w\}$, $v^0 = b_v$, and $v^* = w_v$. Let every operator in Λ be presented as a three-tuple $\langle \{\text{pre}\}, \{\text{post}\}, \{\text{prv}\} \rangle$ of pre-, post-, and prevail conditions of the operator respectively. Suppose that:

$$\sigma(u) = b_u^1 \cdot w_u^1 \quad \sigma(w) = b_w^1 \cdot w_w^1 \cdot b_w^2 \cdot w_w^2$$

$$\Lambda_v = \begin{cases} o_v^1 = \{\{b_v\}, \{w_v\}, \{b_u, w_w\}\} \\ o_v^2 = \{\{w_v\}, \{b_v\}, \{b_u, b_w\}\} \\ o_v^3 = \{\{w_v\}, \{b_v\}, \{w_u, w_w\}\} \end{cases}$$

In this case, the graph $G(v)$ is presented by Figure 1.

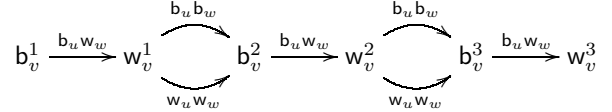


Figure 1: Example of the graph $G(v)$.

The constructed graph $G(v)$ captures information about all *potentially possible* executions of the operators in Λ_v that can provide us $\text{MaxReq}(v)$ or less value changes of v . Each path, started at the source node of $G(v)$, uniquely corresponds to such an execution. Although the number of these alternative executions may be exponential in n , this graphical representation is compact: the number of edges in $G(v)$ is $O(n \cdot |\Lambda_v|)$. Note that the information about the number of times that each operator in Λ_v can be executed is not captured by $G(v)$. The following two steps add this information indirectly and exploit it to find a maximal sequence $\sigma(v)$.

First, we expand $G(v)$ with respect to the maximal sequences $\sigma(w_1), \dots, \sigma(w_k)$ as follows: Each edge $e \in G(v)$ is replaced by a set of edges such that their labels correspond to all possible assignments of the elements of $\sigma(w_1), \dots, \sigma(w_k)$ to $l(e)$. Likewise, we add a dummy source node s_v , with an edge from s_v to the original source node of $G(v)$ labeled by a tuple of the first elements of $\sigma(w_1), \dots, \sigma(w_k)$ (= initial values of w_1, \dots, w_k). Similarly, we add a dummy target node t_v , with an edge from the original target node of $G(v)$ to t_v labeled by a tuple of the last elements of $\sigma(w_1), \dots, \sigma(w_k)$ (= goal values of w_1, \dots, w_k). We denote this extended graph by $G'(v)$, and Figure 2 illustrates $G'(v)$ for the example above.

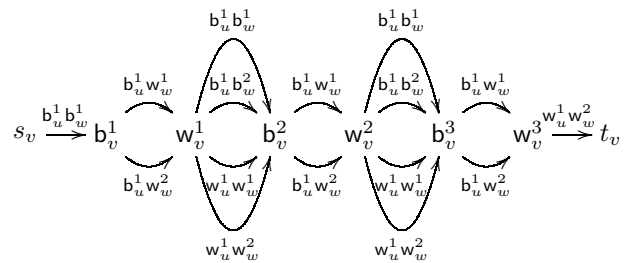


Figure 2: Example of the graph $G'(v)$.

The extended graph $G'(v)$ can be viewed as a projection of the maximal sequences $\sigma(w_i)$, $1 \leq i \leq k$, on $G(v)$. Each

edge in $G(v)$ may be replaced by $O(n^k)$ edges in $G'(v)$, and thus the number of edges in $G'(v)$ is $O(n^{k+1} \cdot |\Lambda_v|)$.

It is easy to see that not all paths in $G'(v)$ starting at s_v are relevant. For example, in $G'(v)$ above, an operator instance prevailed by $b_u^1 b_w^2$, can not be performed after an operator instance prevailed by $b_u^1 w_w^2$. The following step provides a reduction of the problem of finding a longest feasible path from s_v to a v^* -colored node in $G'(v)$ to a known problem of finding a longest path in a directed acyclic graph. Let the new graph $G'_e(v)$ have the edges of $G'(v)$ as nodes, and let its edges be defined by all *allowed* pairs of immediately subsequent edges in $G'(v)$: (e, e') is allowed if, for $1 \leq i \leq k$, either $l(e)_{w_i} = l(e')_{w_i}$ or $l(e')_{w_i}$ appears after $l(e)_{w_i}$ on $\sigma(w_i)$. Such a construction is a variant of a so called “edge graph” known in graph theory; the addition in our case is the exclusion of non-allowed edges from it. Clearly, $G'_e(v)$ can be constructed in time polynomial in size of $G'(v)$, and the number of edges in $G'_e(v)$ is $O(n^{2k+2} \cdot |\Lambda_v|^2)$.

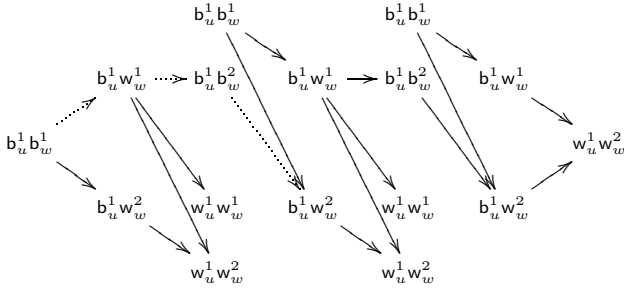


Figure 3: Example of the graph $G'_e(v)$.

Figure 3 presents $G'_e(v)$ for our example. The dotted arrows present the longest path from the dummy source node to a node that corresponds to a value change from $-v^*$ to v^* (from b_v to w_v). Such a longest path in $G'_e(v)$ describes a maximal sequence of value changes $\sigma(v)$, and its length is actually $\text{FMaxReq}(v) + 1$. In our example, $\sigma(v) = b_v^1 \cdot w_v^1 \cdot b_v^2 \cdot w_v^2$, and $\text{FMaxReq}(v) = 3$. Note that if $v^0 = v^*$ then the empty path will be also acceptable since, in general, v does not have to change its value. In this case $\text{FMaxReq}(v) = 0$ and $\sigma(v)$ will consist of only one element which corresponds to the initial (= goal) value of v .

Observe that a longest path in $G'_e(v)$ describes not only $\sigma(v)$ but also the actual sequence of invocations of the operators from Λ_v that provides $\sigma(v)$. We denote by $\{A(b_v^j)\}$ and $\{A(w_v^j)\}$ the sequences of operator instances that have as effects the corresponding elements from the sequences $\{b_v^j\}$ and $\{w_v^j\}$ ($\{b_v^j\} \cup \{w_v^j\} = \sigma(v)$) of v 's values, respectively. In what follows, we address these sequences of operator instances as one sequence of operator instances $\Gamma_v = \{A(\nu_v^i)\}_{i=2}^{\text{FMaxReq}(v)}$, where $\text{post}(A(\nu_v^i)) = \nu_v^i$, and

$$\nu_v^i = \begin{cases} b_v^{\frac{i+1}{2}}, & i = 2k + 1 \\ w_v^{\frac{i}{2}}, & i = 2k \end{cases} \quad k \in \mathbb{N}$$

Procedure FORWARD-CHECK in Figure 4 summarizes the presented approach. Note that finding a set of longest paths from a node to all other nodes in a directed acyclic graph

can be done in time linear in the size of the graph. Therefore, the time complexity of a call to the DETERMINE-MAX-SEQUENCE procedure with a variable v is bounded by the size of the constructed graph $G'_e(v)$ and thus is $O(n^{2k+2} \cdot |\Lambda_v|^2)$. FORWARD-CHECK calls DETERMINE-MAX-SEQUENCE n times. Therefore, if the maximal node indegree is bounded by a constant κ , then the overall complexity of the algorithm is $O(|\mathcal{V}|^{2\kappa+3} \cdot |\Lambda|^2)$, i.e., polynomial in the size of the problem description.

Theorem 1 *A given problem instance with a polytree causal graph is solvable if and only if, for each $v \in \mathcal{V}$, FORWARD-CHECK succeeds to construct the maximal sequence $\sigma(v)$.*

FORWARD-CHECK fails if and only if at least one of the calls to the DETERMINE-MAX-SEQUENCE procedure fails. In turn, a call to DETERMINE-MAX-SEQUENCE on a variable v fails if and only if the initial and the goal values of v are different but there is no way to change the value of v even once. Thus, if FORWARD-CHECK fails, then no plan exists.

To prove the opposite direction we proceed as follows: We define the POP-PCG algorithm (POP for polytree causal graphs) and show that it will succeed without backtracking if FORWARD-CHECK succeeds. POP-PCG is described in detail in Figure 6, and it works as follows: First, let us expand each sequence of operator instances Γ_i by $A(\nu_i^1)$ ($A(b_i^1)$) which will stand for the dummy operator A_i^0 . (Recall that up until now, only operators of the form $A(\nu_i^j)$ for $j > 1$ were defined.) The algorithm 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; if it requires achieving some value for v_i we add the corresponding operator to the plan with the desired effect (step 3a). Actually, if we would be ready to accept plans with possible redundant steps, we can omit the next step 3b from the algorithm by assuming that the goal value of each variable v is the last element of the maximal sequence $\sigma(v)$. However, if we would like our plan to be irreducible, then a careful decision about the really required number of value changes of each variable is required. This decision is captured in step 3b by analysis of the value changes of a variable v_i that were found necessary in the previous iterations of the algorithm in order to satisfy the predecessors of v_i in the causal graph. Note that the agenda is sorted with respect to some reverse topological ordering of the causal graph, thus if an operator affecting v_i was selected from the agenda then no operator affecting some predecessor of v_i in the causal graph will appear on the agenda until the end of the algorithm. No threats arise in POP-PCG, and the ordering constraints are consistent.

Lemma 2 *If FORWARD-CHECK was successful then POP-PCG will return a valid plan.*

Proof sketch: Informally, the lemma will follow from the following claims: (i) for every agenda item, there exists an operator that has it as an effect; (ii) there are no threats in the output of POP-PCG; (iii) the ordering constraints in \mathcal{O} are consistent; (iv) the agenda will be empty after a polynomial number of steps. Because of the space limitations, we refer

Procedure FORWARD-CHECK (II)

1. Topologically sort all variables \mathcal{V} based on the the causal graph.
2. For each variable $v \in \mathcal{V}$, call DETERMINE-MAX-SEQUENCE(II, v), respecting the above ordering.
3. If one of the calls to DETERMINE-MAX-SEQUENCE return failure, then return failure. Otherwise return success.

Procedure DETERMINE-MAX-SEQUENCE (II, v)

1. If $\text{pred}(v) = \emptyset$ then
 - (a) If $v^0 \neq v^*$ and $A_v^+ \notin \Lambda_v$, return failure.
 - (b) Otherwise, determine $\sigma(v)$ according to the rules for the root variable and return success.
2. Otherwise, if $\text{pred}(v) = \{w_1, \dots, w_k\}$ then
 - (a) Construct $G(v)$ (based on v^0, v^* , and Λ_v).
 - (b) Construct $G'(v)$ (from $G(v)$, based on $\sigma(w_1), \dots, \sigma(w_k)$).
 - (c) Construct $G'_e(v)$ (from $G'(v)$, based on $\sigma(w_1), \dots, \sigma(w_k)$).
 - (d) Determine the longest path in $G'_e(v)$ to a node corresponding to a v^* -ended value change, and derive $\sigma(v)$ and the corresponding sequence of operators from it.
 - (e) If $v^0 \neq v^*$ and $\text{FMaxReq}(v) = 0$, return failure. Otherwise, return success.

Figure 4: FORWARD-CHECK algorithm

the reader to (Domshlak & Brafman 2001) for the proof of the lemma. ■

Singly Connected and General DAGs

In this section we analyze planning complexity in face of more complicated causal graphs. First, we show that when the causal graph is singly connected even plan existence is NP-complete. Second, we show that for general causal graphs the situation is even worse. Finally, we characterize an important parameter of the causal graph affecting planning complexity, which allows us to extend the class of problems which are in NP.

Theorem 2 *Plan existence for STRIPS planning problems with unary operators and singly connected causal graph is NP-complete.*

Proof sketch: The membership in NP is shown as follows: Let $\text{MinPlanSize}(\Pi)$ denote the size of a minimal plan for a problem instance Π . Using the MaxReq property of the state variables, the following upper bound for $\text{MinPlanSize}(\Pi)$ is straightforward from the Lemma 1:

$$\text{MinPlanSize}(\Pi) \leq \sum_{v \in \mathcal{V}} \text{MaxReq}(v) \leq n^2 \quad (3)$$

Thus, if we guess a minimal solution for a given solvable problem, we can verify it in low polynomial time.

The proof of the hardness is by polynomial reduction from 3-SAT to the corresponding propositional plan generation problem with a singly connected causal graph. ■

The singly connected 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 provable 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, all these restricted problems are still NP-complete.

Theorem 3 *Plan generation for general STRIPS planning problems with unary operators and acyclic causal graph is provably intractable, i.e. it is in EXPTIME.*

This theorem follows from Theorem 4.10 in (Jonsson & Bäckström 1995), that shows that plan generation for 3S problem class is provably intractable. The point is that the upper bound for MinPlanSize , presented in Eq. 3, in this case can be exponential in the size of the input. First, we show by example that this upper bound can be achieved, then we present some analysis of reasons for this intractability.

The following problem example, for which such an exponential upper bound can be achieved, was used in the proof of Theorem 4.10 in (Jonsson & Bäckström 1995), and was originally presented in a different context in (Bäckström & Nebel 1995). Consider a propositional planning problem with $|\mathcal{V}| = n$, and $\text{pred}(v_i) = \{v_1, \dots, v_{i-1}\}$ for $1 \leq i \leq n$. The operator set Λ consist of $2n$ operators $\{A_1, A'_1, \dots, A_n, A'_n\}$ where

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

It is easy to see that the causal graph of this problem forms a DAG (see Figure 5), and an instance of this planning problem with the initial state $\langle 0, \dots, 0 \rangle$ and the goal state $\langle 0, \dots, 0, 1 \rangle$ has a unique minimal solution of length $2^n - 1$ corresponding to a Hamilton path in the state space.

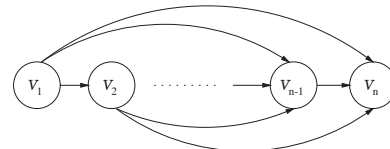


Figure 5: Causal graph for the proof of Theorem 3

Now we show that this escalation in complexity can be “parametrized” by the form of the causal graph. Rela-

