The Influence of k-Dependence on the Complexity of Planning

Omer Giménez
Dept. de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Jordi Girona, 1-3
08034 Barcelona, Spain
omer.gimenez@upc.edu

Anders Jonsson
Dept. of Information and Communication Technologies
Universitat Pompeu Fabra
Roc Boronat, 138
08018 Barcelona, Spain
anders.jonsson@upf.edu

Abstract

A planning problem is k-dependent if each action has at most k pre-conditions on variables unaffected by the action. This concept is well-founded since k is a constant for all but a few of the standard planning domains, and is known to have implications for tractability. In this paper, we present several new complexity results for $\mathbf{P}(k)$, the class of k-dependent planning problems with binary variables and polytree causal graphs. The problem of plan generation for $\mathbf{P}(k)$ is equivalent to determining how many times each variable can change. Using this fact, we present a polytime plan generation algorithm for $\mathbf{P}(2)$ and $\mathbf{P}(3)$. For constant $k > 3$, we introduce and use the notion of a cover to find conditions under which plan generation for $\mathbf{P}(k)$ is polynomial.

Introduction

It is generally acknowledged that most interesting planning domains exhibit some sort of structure, and that identifying this structure is often key to solving them efficiently. This is at least clear from a theoretical point of view, since it is well known that planning is intractable in general (Chapman 1987), and PSPACE-complete when restricted to propositional variables (Bylander 1994). An ongoing research project in the planning community is to classify planning problems according to their computational complexity.

From a practical point of view, the previous research is of special interest when some class of planning problems is shown to be polynomial-time solvable, a well-established theoretical notion that is often identified with practical tractability. Now and then, the significance of this type of result goes further than identifying a new class of planning problems that is easy to solve. Tractable classes of planning have been exploited in the past to define domain-independent heuristics: typically, one estimates the cost of solving a (complex) planning problem by relaxing it until it can be solved efficiently. Also, tractable classes that are defined in terms of structural restrictions play an important role in the context of factored planning.

Causal graphs and k-dependence

The causal graph of a planning problem is a directed graph whose edges describe variable dependencies (Knoblock 1994). The shape of the causal graph captures part of the underlying structure of the problem. For example, causal graphs have been used in the context of hierarchical decomposition. Also, the Fast Downward planner (Helmert 2006) is based on the domain-independent causal graph heuristic, which approximates the cost of solving a multi-valued planning problem by exploiting its causal graph structure.

However, a simple causal graph does not guarantee that the corresponding planning problem is easy to solve. For instance, solving planning problems with directed-path singly connected causal graphs is NP-hard (Brafman and Domshlak 2003), even for binary variables and causal graphs with bounded degree. When the causal graph is a directed path, the problem is NP-hard for variables with domains of size at least 5 (Giménez and Jonsson 2009). The situation is even worse for multi-valued variables, since Chen and Giménez (2008) showed the following. Let $\mathcal{C}$ be *any* infinite family of directed graphs whose underlying undirected graph is connected. Then, unless standard complexity-theoretic assumptions fail, no polynomial-time algorithm exists for solving the class of planning problems with causal graphs in $\mathcal{C}$.

Based on these results, a question that naturally arises is which additional assumptions we need to impose on planning problems to solve them in polynomial time. Clearly, the focus here is to find reasonable assumptions that are present in actual planning problems. We mention just a few. If the domain transition graphs are strongly connected, we can solve problems with acyclic causal graphs (Williams and Nayak 1997; Helmert 2006). For bounded local depth (i.e., the number of times that the value of a variable has to change on a plan solving the problem), plan generation is polynomial for planning problems with causal graphs of bounded tree-width (Brafman and Domshlak 2006).

With this in mind, Katz and Domshlak (2008b) introduced the notion of k-dependent actions. The dependence of an action is the number of pre-conditions on variables unaffected by the action; an action is k-dependent if its dependence is at most k. Generally, the dependence is bounded: Table 1 shows the largest k of actions in STRIPS planning domains from the IPC. For reference, the table also shows the largest p of actions, defined as the total number of pre-conditions. The values include static pre-conditions, which is why the results for Pipesworld and Rovers differ, since the actions in the IPC5 domains are already grounded. For all but a few

IPC2		**IPC5**	
Blocks	0/3	Openstacks	95/97
Miconic-STRIPS	2/3	Pathways	1/2
FreeCell	6/10	Pipesworld	0/4
Logistics	2/3	Rovers	2/4
Schedule-STRIPS	5/9	Storage	3/5
IPC3		TPP	1/3
Depots	2/5	Trucks	6/8
DriverLog	2/3	**IPC6**	
ZenoTravel	2/4	Cyber security	8/31
Rovers	5/6	Elevator	3/5
Satellite	6/6	Openstacks	9/11
IPC4		PARC printer	3/7
Airport	4/29	Peg solitaire	1/5
Pipesworld-No tankage	5/8	Scanalyzer	1/5
Pipesworld-Tankage	6/11	Sokoban	3/6
Promela optical	46/47	Transport	1/4
Promela philosophers	31/32	Woodworking	6/9
PSR	52/53		

Table 1: Largest k/p for IPC STRIPS planning domains.

domains, k appears to be a small constant.

Interestingly, the actual dependence of actions cannot be inferred from the causal graph. Consider, for instance, the causal graph in Figure 1. Since variable v has high indegree, there exist actions affecting v that depend on the values of its many predecessors. However, what is lost is the *structure* of these actions: there may be multiple 1-dependent actions, one for each predecessor; or maybe there is a single action depending on all predecessors; or a combination of several types of actions. In particular, problems with actions whose dependence is small may be easier to solve than is apparent from the shape of the causal graph alone.

Polytree causal graphs

A *polytree* is a directed graph whose underlying undirected graph (i.e., with every directed edge replaced by an undirected one) is acyclic, such as the graph in Figure 1. Brafman and Domshlak (2003) showed that the class $\mathbf{P}$ of planning problems with polytree causal graphs and binary variables can be solved in polynomial time if variables have bounded indegree. On the other hand, if the indegree is unbounded, Giménez and Jonsson (2008) showed that plan existence for this class is NP-complete by reduction from 3-SAT. Figure 1 shows an example causal graph of planning problems that 3-SAT formulas reduce to. Interestingly, the reduction produces graphs that contain a single variable v whose indegree depends on the size of the formula; moreover, all actions affecting v are 3-dependent[1], except one, which has an unbounded number of pre-conditions.

Katz and Domshlak (2008a) introduced the subclass $\mathbf{P}(k)$ of $\mathbf{P}$ of planning problems with k-dependent actions, and proposed to study the complexity of solving planning problems in $\mathbf{P}(k)$. Since the proof of NP-completeness for $\mathbf{P}$ requires an action that is not k-dependent for any k, it does not apply to $\mathbf{P}(k)$. Indeed, Katz and Domshlak (2008a) described a polynomial-time algorithm for solving planning problems in $\mathbf{P}(1)$ optimally, thus showing that the indegree of variables in the causal graph does not have to be bounded for a problem to be tractable.

[1] Although the given reduction uses 7-dependent actions, only 3 of the pre-conditions on predecessors of v are strictly necessary.

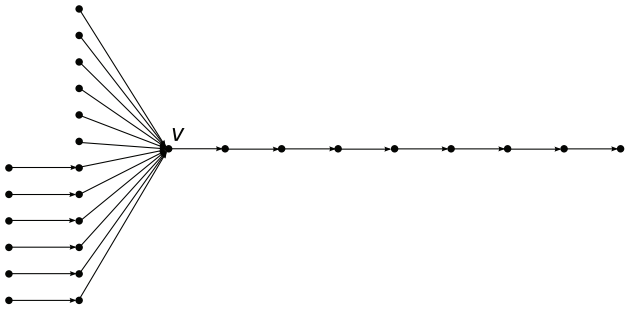

Figure 1: Polytree causal graph in the 3-SAT reduction of Giménez and Jonsson (2008).

Our contribution

In this work, we present several new tractability results for the class $\mathbf{P}(k)$ of planning problems. First, we provide a polynomial-time plan generation algorithm for the class $\mathbf{P}(2)$, and claim that the same result applies to $\mathbf{P}(3)$ (details are omitted due to lack of space). In contrast with the work of Katz and Domshlak (2008a), our algorithm does not produce optimal solutions. This way, we are able to prove tractability for planning problems with k larger than 1. This relates to a question raised by Katz and Domshlak (2008a), namely whether one can aim for more by relaxing the optimality requirement.

For $k > 3$, we also present an plan generation algorithm that can solve planning problems in a subclass of $\mathbf{P}(k)$ in polynomial time. Let the depth of a variable v be the longest path in the causal graph between a source node and v. For instance, the variable v in Figure 1 has depth 2. Our algorithm is polynomial-time for planning problems in $\mathbf{P}(k)$ such that variables with large indegree have bounded depth.

Note that the only variable with unbounded indegree in the polytree reduction of Giménez and Jonsson (2008) has depth 2. Thus, our algorithm would be able to solve these planning problems, were it not for the presence of a single action with unbounded dependence. Consequently, it is not possible to rewrite the action with unbounded dependence in terms of k-dependent actions, for any k. Hence, if one is to show that $\mathbf{P}(k)$ plan existence is NP-complete for some fixed k, a more involved reduction is necessary.

The key insight behind both algorithms is that although a variable v may have an unbounded number of predecessors in the causal graph, only some of these predecessors are relevant for changing the value of v. If there are actions for alternating the value of v whose pre-conditions can be satisfied simultaneously, the problem is easy to solve with respect to v. Otherwise, the fact that actions are k-dependent severely restricts the possible configurations to consider.

Causal-graph structural patterns

Most interesting planning domains do not fall inside the few classes of planning that are known to be polynomial-time tractable. One of the techniques used by general purpose planners to deal with these domains is *heuristic search*. To develop domain-independent heuristics, one typically performs abstraction in the planning problem at hand so that the abstracted problem falls inside a tractable class, and then estimates the cost of solving the real instance by computing the optimal cost of solving the simplified instance. Clearly, one seeks simplifications that are both informative and efficient to solve.

Domain-independent *pattern database* (PDB) heuristics (Edelkamp 2001; Haslum et al. 2007; Helmert, Haslum, and Hoffmann 2007) try to achieve these simplifications by projecting sets of states of the planning instance onto subproblems of small dimensionality, to be solved by exhaustive search. Katz and Domshlak (2008b) introduced a generalization of the PDB abstractions that they call *causal graph structural patterns* (CGSP). They proposed to let the causal graph of the planning instance guide the projection onto smaller subproblems, in order to leverage the knowledge of tractable planning with respect to the causal graph. In principle, such projections could overcome the limitation of PDB heuristics with respect to the size on the subproblems, since exhaustive search would no longer be required to solve them. Hence, tractable classes of planning can be used to extend the palette of patterns the problem can be decomposed into.

Cast into this context, our work is not a direct extension of the work of Katz and Domshlak (2008a), since our algorithms do not produce optimal plans for solving the corresponding planning problems. Thus, any resulting heuristic would not be admissible. However, increasing the value of k allows actions to be more expressive, since they can have pre-conditions on multiple variables. Thus, a reduction onto $\mathbf{P}(k)$ for $k > 1$ might be more informative than a corresponding reduction onto $\mathbf{P}(1)$. Our work thus offers the possibility to trade off the admissibility of the resulting heuristic with the informativeness of the same heuristic.

Notation

Let V be a set of binary variables with domain $\{0, 1\}$. A *value set* $C \subseteq V \times \{0, 1\}$ is a subset of variable-value pairs. Let $W(C) = \{v \in V : (v, 0) \in C \vee (v, 1) \in C\}$ be the set of variables appearing in C. Let $\overline{C} = \{(v, x) \in V \times \{0, 1\} : (v, 1-x) \in C\}$ be the *complement* of C. A *partial state* p is a value set such that for each $v \in V$, $(v, 0) \notin p$ or $(v, 1) \notin p$. A *state* s is a partial state for which $|s| = |V|$.

A planning problem is a tuple $P = \langle V, \mathsf{init}, \mathsf{goal}, A\rangle$, where V is the set of variables, init is an initial state, goal is a partial goal state, and A is a set of actions. An action $a = \langle pre(a); post(a)\rangle \in A$ consists of a partial state $pre(a)$ called the *pre-condition* and a partial state $post(a)$ called the *post-condition*. Action a is applicable in any state s such that $pre(a) \subseteq s$, and applying a in state s results in a new state $s' = (s - \overline{post(a)}) \cup post(a)$.

The causal graph (V, E) of a planning problem P is a directed graph with the variables of P as nodes. There is an edge $(u, v) \in E$ if and only if $u \neq v$ and there exists an action a such that $u \in W(pre(a)) \cup W(post(a))$ and $v \in W(post(a))$. A planning problem P belongs to the class $\mathbf{P}(k)$ if and only if its causal graph is a polytree and for each action $a \in A$, $|W(pre(a)) - W(post(a))| \leq k$. The fact that the causal graph is a polytree implies that actions are unary, i.e., $|post(a)| = 1$ for each $a \in A$.

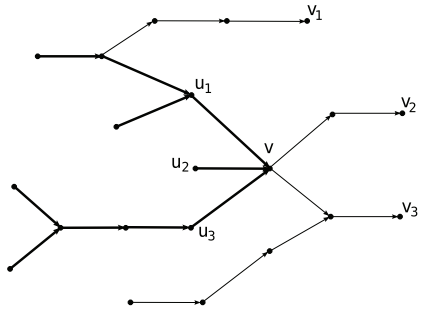

Figure 2: Polytree causal graph of a planning problem P. In bold, the causal graph of $P(v)$.

Plan Generation for $\mathbf{P}(k)$

Let P be a planning problem in $\mathbf{P}(k)$. In this section we show that determining plan existence for P can be reduced to the problem of determining, for each $v \in V$, the maximum number of times that the value of v can change. Moreover, the problem of plan generation for P can be reduced to generating, for each $v \in V$, plans that change v a maximum number of times. Although the same argument was used by Brafman and Domshlak (2003), we repeat large parts of it here, since our subsequent proofs require a specific form of reduction: that of P to k-dependent inverted fork problems.

Without loss of generality, we assume throughout that $\mathsf{init} = V \times \{0\}$. Otherwise, for each $v \in V$ such that $(v, 1) \in \mathsf{init}$, we can just relabel the domain of v. For each $v \in V$, let $u_1, \ldots, u_m$ be the predecessors of v in the causal graph, i.e., $(u_i, v) \in E$ for each $i \in \{1, \ldots, m\}$. We recursively define the set $\mathbf{anc}(v)$ of *ancestors* of v as $\bigcup_i (\{u_i\} \cup \mathbf{anc}(u_i))$.

Definition 1. *Let $P = (V, \mathsf{init}, \mathsf{goal}, A)$ be a planning problem in $\mathbf{P}(k)$. Define the planning problem $P(v)$ as the restriction of P to variables in $V' = \{v\} \cup \mathbf{anc}(v)$, i.e., $P(v) = (V', \mathsf{init}', \mathsf{goal}', A')$ where init' and goal' are the intersections of init and goal with $V' \times \{0, 1\}$, and $A' \subseteq A$ contains actions a such that $\mathbf{pre}(a), \mathbf{post}(a) \subseteq V' \times \{0, 1\}$.*

Figure 2 shows the causal graph of a planning problem $P \in \mathbf{P}(k)$, as well as the causal graph of the planning problem $P(v)$, which is a subgraph of the former.

Definition 2. *We define $N(v) \in \mathbb{N} \cup \{\infty, \perp\}$ as the maximum number of times that v can change on a plan solving $P(v)$, with $\perp$ denoting that $P(v)$ is unsolvable.*

Note that unless $N(v) = \infty$, $N(v)$ must agree with the goal. Namely, if $(v, 0) \in \mathsf{goal}$, $N(v)$ must be even, and if $(v, 1) \in \mathsf{goal}$, $N(v)$ must be odd. While computing the

maximum number of changes of v, we can safely ignore the goal: if the parity of the computed value does not agree with the goal, subtracting 1 from it results in the correct value.

Our procedure for computing $N(v)$ requires $N(u)$ to be known for each predecessor u of v in the causal graph. Since the causal graph is acyclic it is possible to compute these values bottom-up.

For a predecessor u of v such that $N(u) = 0$, the value of u can never change, so we do not need to consider u while computing $N(v)$. For a predecessor u of v such that $N(u) = \infty$, u does not restrict the number of times that v can change, since any pre-condition on u can always be satisfied by changing u an extra time. Thus, the set of relevant predecessors are those that can change a positive finite number of times.

Definition 3. *For each $v \in V$, let $\mathbf{pred}(v) = \{u \in V : (u, v) \in E \wedge 0 < N(u) < \infty\}$ be the set of relevant predecessors of v.*

Note that if $N(u) = \bot$ for some predecessor u of v, the planning problem $P(v)$ is unsolvable, implying $N(v) = \bot$.

Next, we define the set of relevant actions that change the value of v. For a predecessor u of v such that $N(u) = 0$, any action with pre-condition $(u, 1)$ is irrelevant, in the sense that it could never appear in a valid plan. Hence, we exclude such actions while computing $N(v)$. Furthermore, we project the pre-conditions of relevant actions onto the set $\mathbf{pred}(v)$ of relevant predecessors of v.

Definition 4. *For each $(v, x) \in V \times \{0, 1\}$, let $A_v^x = \{pre(a) \cap \mathbf{pred}(v) \times \{0, 1\} : a \in A \wedge post(a) = \{(v, x)\} \wedge \nexists (u, 1) \in pre(a) \text{ s.t. } N(u) = 0\}$ be the set of relevant actions that change v to x.*

Note that we abuse notation slightly by claiming that A_v^x is a set of actions even though it is in fact a set of partial states. The reason is that there is a one-to-one correspondence between actions and partial states in A_v^x, since we can change the value of v to x in any state s for which there exists $p \in A_v^x$ such that $p \subseteq s$, by applying the associated action. Note that A_v^x may contain the empty partial state $\emptyset$; in particular, this happens if v has no predecessors in the causal graph.

There are four easy cases for computing $N(v)$:

Lemma 5. *Let P be a planning problem in $\mathbf{P}(k)$, and $v \in V$ a variable. Then*

$$N(v) = \begin{cases} \bot, & |A_v^1| = 0 \wedge (v, 1) \in \mathsf{goal}, \\ 0, & (|A_v^1| = 0 \wedge (v, 1) \notin \mathsf{goal}) \vee \\ & (|A_v^0| = 0 \wedge (v, 0) \in \mathsf{goal}), \\ 1, & |A_v^1| > 0 \wedge |A_v^0| = 0 \wedge (v, 0) \notin \mathsf{goal}, \\ \infty, & \exists (p, q) \in A_v^0 \times A_v^1 \text{ s.t. } p \cap \overline{q} = \emptyset. \end{cases}$$

This classification is not exhaustive, since there are instances of v that do not fall within these four cases.

Proof. The first case states that if the goal value of v is 1 and there is no relevant action for changing the value of v to 1, the problem $P(v)$ is unsolvable, so $N(v) = \bot$. The second case states that there is no relevant action for changing the value of v to 1, but the goal value of v is not 1, or the goal value of v is 0 and there is no relevant action for resetting the value of v to 0. The third case states that there exist relevant actions for changing the value of v to 1, but there is no relevant action for resetting the value to 0. The fact that there exists $(p, q) \in A_v^0 \times A_v^1$ such that $p \cap \overline{q} = \emptyset$ implies that we can satisfy p and q simultaneously and repeatedly apply the associated actions to change the value of v an arbitrary number of times. □

Note that if v has no predecessors in the causal graph, one of the easy cases in Lemma 5 always holds.

Next, we introduce the notion of a cover. As we shall see, covers are useful since they bound the number of value changes for each variable v that does not fall into an easy case for computing $N(v)$. In particular, we use covers to provide a global bound on the number of value changes for such variables.

Definition 6. *Let $\langle K^0, K^1 \rangle$ be two sets of partial states on $V \times \{0, 1\}$. A* cover *$C \subseteq V \times \{0, 1\}$ for $\langle K^0, K^1 \rangle$ is a value set such that for each $(p, q) \in K^0 \times K^1$, $|p \cap \overline{q} \cap C| \geq 1$. A* minimum cover *is a minimum such value set.*

Lemma 7. *Let C be a cover for $\langle A_v^0, A_v^1 \rangle$. Then $N(v) \leq 1 + \sum_{u \in W(C)} N(u)$.*

Proof. Since $|p \cap \overline{q} \cap C| \geq 1$ for each $(p, q) \in A_v^0 \times A_v^1$, whenever we change the value of v we cannot change its value again without first changing the value of at least one variable in $W(C)$. The maximum number of times we can do this is $\sum_{u \in W(C)} N(u)$. Possibly, we can change the value of v an additional time in the initial state. □

Lemma 8. *For each $v \in V$ such that $N(v) < \infty$, it holds that $N(v) \leq 1 + |\mathbf{anc}(v)|$.*

Proof. By induction on v. Since $N(v) < \infty$, there exists no pair $(p, q) \in A_v^0 \times A_v^1$ such that $p \cap \overline{q} = \emptyset$. Then by definition $\mathbf{pred}(v) \times \{0, 1\}$ is a cover for $\langle A_v^0, A_v^1 \rangle$. From Lemma 7 we obtain

$$\begin{aligned} N(v) &\leq 1 + \sum_{u \in \mathbf{pred}(v)} N(u) \leq \\ &\leq 1 + \sum_{u \in \mathbf{pred}(v)} (1 + |\mathbf{anc}(u)|) = \\ &= 1 + |\mathbf{pred}(v)| + \sum_{u \in \mathbf{pred}(v)} |\mathbf{anc}(u)| \leq \\ &\leq 1 + |\mathbf{anc}(v)|, \end{aligned}$$

where we have applied the inductive argument on the second line. The last inequality follows from the fact that the causal graph is a polytree, implying that the predecessors of v can have no common ancestors. □

We now show that solving $P \in \mathbf{P}(k)$ is equivalent to solving several instances of *inverted fork* problems, i.e., planning problems whose causal graphs are inverted forks. Such problems were introduced by Katz and Domshlak (2008b), but with a different definition. In our definition, an inverted fork problem consists of determining the maximum number

of times that the root variable v can change and generating an action sequence that produces these changes, given bounds on the number of changes of each predecessor u.

Definition 9. *An* inverted fork *computational problem is a tuple $\langle v, U, n, A\rangle$ where*

- *v is the root variable;*
- *U is the set of predecessors of v;*
- *$n : U \mapsto \mathbb{N} - \{0\}$ (i.e., $0 < n(u) < \infty$) is a bound on the number of times that each variable $u \in U$ can change;*
- *$A = \langle A^0, A^1\rangle$, where A^x is a non-empty set of unary actions changing v to x, i.e., a non-empty set of partial states on $U \times \{0, 1\}$. Moreover, we require that for each $(p, q) \in A^0 \times A^1$, it holds that $|p \cap \overline{q}| \geq 1$.*

The output of $\langle v, U, n, A\rangle$ is a pair $\langle n(v), \pi\rangle$, where $n(v)$ is the maximum number of times that v can change using actions of A, assuming that all variables start with value 0 and that each variable $u \in U$ can change freely, but no more than $n(u)$ times; π is a sequence of actions attaining this maximum.

Lemma 10. *Let $\langle v, U, n, \langle A^0, A^1\rangle\rangle$ be an inverted fork problem, and let C be a cover for $\langle A^0, A^1\rangle$. Then $n(v) \leq 1 + \sum_{u\in W(C)} n(u)$.*

Proof. Follows by the same argument as in the proof of Lemma 7. □

Let $\|A\|$ denote the sum of the sizes of actions in A^0 and A^1. We take the *size* of $\langle v, U, n, A\rangle$ to be $|U| + \|A\| + \sum_{u\in U} n(u)$, i.e., as if the numbers $n(u)$ were expressed in unary notation. Otherwise, the solution could be exponential in the size of the input. Note that the output satisfies $0 < n(v) < \infty$; $n(v) > 0$ since A^1 is non-empty, and $n(v) \leq 1 + \sum_{u\in U} n(u) < \infty$ by Lemma 10 and the fact that $U \times \{0, 1\}$ is a cover for $\langle A^0, A^1\rangle$.

Definition 11. *Let $\boldsymbol{\mathcal{I}}(k)$ be the class of inverted fork problems with k-dependent actions, i.e., $|p| \leq k$ for each $p \in A^0 \cup A^1$.*

We show that plan generation for $\mathbf{P}(k)$ can be reduced to solving inverted fork problems in $\boldsymbol{\mathcal{I}}(k)$. The idea is to independently generate plans that change each variable $v \in V$ a maximum number of times by solving the corresponding planning problem $P(v)$. To do so, it is only necessary to consider the immediate predecessors $\{u_1, \ldots, u_m\}$ of v. For example, to solve the planning problem $P(v)$ in Figure 2 we only have to consider the predecessors u_1, u_2, and u_3 of v.

Proposition 12. *Plan generation for $\mathbf{P}(k)$ is polynomial-time reducible to solving $\boldsymbol{\mathcal{I}}(k)$.*

Proof (Sketch). Let $P = \langle V, \mathsf{init}, \mathsf{goal}, A\rangle$ be an instance of $\mathbf{P}(k)$. We describe an algorithm that generates a plan solving P. Recall that $P(v)$ is the restriction of P to variables $\{v\} \cup \mathbf{anc}(v)$. For each $v \in V$ in topological order, compute $N(v)$, the maximum number of times that v can change in $P(v)$, and if $N(v) < \infty$, a plan solving $P(v)$ attaining this maximum. If $N(v) = \infty$, instead produce a pair of actions that can change the value of v as many times as necessary.

If one of the easy cases in Lemma 5 apply, we can compute $N(v)$ in polynomial time. If $N(v) = 0$, the empty sequence attains the maximum. If $N(v) = 1$, any action $q \in A^1_v$ attains the maximum. If $N(v) = \infty$, we keep the two actions $(p, q) \in A^0_v \times A^1_v$ such that $p \cap \overline{q} = \emptyset$. Otherwise, we ask for a solution to $\langle v, \mathbf{pred}(v), N, \langle A^0_v, A^1_v\rangle\rangle \in \boldsymbol{\mathcal{I}}(k)$; note that we already know $N(u)$ for each $u \in \mathbf{pred}(v)$ since u comes before v in topological order. In this case, $N(v)$ is either $n(v)$ or $n(v) - 1$, depending on the goal value of v.

Note that each resulting action sequence only includes actions for changing the value of v. To obtain a plan for $P(v)$, we have to *merge* the sequence for v with the plans for $P(u)$, where u is a predecessor of v with $N(u) > 0$. To do this, whenever the sequence for v requires a value of u different from its current value, insert the part of the plan for $P(u)$ that changes the value of u once. Merging is possible precisely because the causal graph is a polytree, implying that the planning problems $P(u)$ have no variables in common.

Finally, to construct a plan solving P we have to merge the plans solving $P(v_1), \ldots, P(v_n)$, where v_i are the sink variables in the causal graph of P. This is possible since, if v is a common ancestor of v_i and v_j, the plans solving $P(v_i)$ and $P(v_j)$ were generated using the same partial plan for solving $P(v)$. P is unsolvable if and only if $P(v_i)$ is unsolvable for some such v_i. Otherwise, since the merged plan solves $P(v)$ for each $v \in V$, it also solves P.

Our algorithm for solving P is polynomial if we can solve the instances of $\boldsymbol{\mathcal{I}}(k)$ in polynomial time. Whenever we ask for a solution to an instance of $\boldsymbol{\mathcal{I}}(k)$, the numbers $n(u)$ are bounded by $1 + |\mathbf{anc}(u)|$ due to Lemma 8. This ensures that the reduction is polynomial-size preserving. □

We remark that the converse is also true, i.e., we can solve instances of $\mathcal{I}(k)$ by solving planning instances of $\mathbf{P}(k)$. The idea is to construct planning problems $P_t \in \mathbf{P}(k)$ where variables $u \in U$ can only change $n(u)$ times, and variable v must change at least t times; then, we look for the smallest t such that P_t has a solution. These restrictions can be imposed, for instance, by using the direct-path causal graph constructions proposed by (Giménez and Jonsson 2008).

$\mathbf{P}(2)$ and $\mathbf{P}(3)$

In this section we show that inverted fork problems in $\boldsymbol{\mathcal{I}}(2)$ and $\boldsymbol{\mathcal{I}}(3)$ can be solved in polynomial time. By Proposition 12, this implies that plan generation is polynomial for $\mathbf{P}(2)$ and $\mathbf{P}(3)$. To show that $\boldsymbol{\mathcal{I}}(2)$ and $\boldsymbol{\mathcal{I}}(3)$ can be solved in polynomial time, we prove that the problems of each type can be reduced to equivalent problems with only a constant number of variables and actions. Here, equivalent means that the problems have identical solutions.

Lemma 13. *Each inverted fork problem $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(2)$ can be reduced in polynomial time to an equivalent problem $\langle v, U', n, A'\rangle$ such that $|U'| \leq 4$ and $|A'| \leq 6$.*

Proof. Recall that, by definition of inverted fork problems, $|p \cap \overline{q}| \geq 1$ for each $(p, q) \in A^0 \times A^1$. We use this property to classify the inverted fork problems in $\boldsymbol{\mathcal{I}}(k)$ into several cases. For some of them, the desired property already holds.

For each remaining case, we explain how to perform the reduction. We begin by analyzing the partial states in A^0.

I There exists $(u, x) \in U \times \{0, 1\}$ such that $(u, x) \in p$ for each $p \in A^0$.
  (a) For each $q \in A^1$, $(u, 1-x) \in q$.
  (b) There exists $q \in A^1$ such that $(u, x) \notin \overline{q}$, implying $|A^0| \leq 2$ since $|q| \leq 2$ and $|p \cap \overline{q}| \geq 1$ for each $p \in A^0$.

II There exist $\{(u_1, x_1), (u_2, x_2)\}, \{(u_3, x_3), (u_4, x_4)\} \in A^0$, where (u_1, x_1)–(u_4, x_4) are distinct. Each $q \in A^1$ has to contain either $(u_1, 1-x_1)$ or $(u_2, 1-x_2)$, and either $(u_3, 1-x_3)$ or $(u_4, 1-x_4)$, implying $|A^1| \leq 4$.
  (a) A^0 contains no other partial states, so $|A^0| = 2$.
  (b) $\{(u_i, x_i), (u_5, x_5)\} \in A^0$ for $i \in \{1, 2, 3, 4\}$ and (u_5, x_5) distinct from (u_1, x_1)–(u_4, x_4). Then each $q \in A^1$ has to contain $(u_i, 1-x_i)$, so $|A^1| \leq 2$.
  (c) $\{(u_1, x_1), (u_3, x_3)\} \in A^0$. Then A^1 cannot contain $\{(u_2, 1-x_2), (u_4, 1-x_4)\}$, so $|A^1| \leq 3$. The same argument holds for any $\{(u_i, x_i), (u_j, x_j)\}$, $i \in \{1, 2\}$ and $j \in \{3, 4\}$.

III Remaining cases.

Note that in Case II(c), we cannot add another partial state to A^0 without excluding at least one more partial state from A^1. By symmetry, the same cases hold for A^1.

If we are in case III for both A^0 and A^1, we show that the only remaining possibility is $|A^0| = |A^1| = 3$. Let $\{(u_1, x_1), (u_2, x_2)\}, \{(u_1, x_1), (u_3, x_3)\}$ be two partial states in A^0. Since Case I does not hold, there exists $p \in A^0$ such that $(u_1, x_1) \notin p$. Since Case II does not hold, p has to share a variable-value pair with each other partial state in A^0. The only possibility is $p = \{(u_2, x_2), (u_3, x_3)\}$. A similar argument for A^1 suffices to show that A^1 contains $\{(u_1, 1-x_1), (u_2, 1-x_2)\}$,$\{(u_1, 1-x_1), (u_3, 1-x_3)\}$, and $\{(u_2, 1-x_2), (u_3, 1-x_3)\}$.

Note that the property $|U| \leq 4$ and $|A| \leq 6$ already holds in Cases II(a), II(c), and III. By symmetry, Case II(b) for A^0 is equivalent to Case I(b) for A^1 using $(u, x) = (u_i, 1-x_i)$. It remains to show that Cases I(a) and I(b) can be reduced to equivalent problems with the desired property.

In Case I(a), let $p = \{(u, x), (w, y)\}$ be a partial state in A^0, and assume $|A^1| > 1$. At least one partial state in A^1 has to be different from $\{(u, 1-x), (w, 1-y)\}$. Call it q. Since we can simultaneously satisfy variable-value pairs of p and q not involving u, we can ignore those variable-value pairs while computing $n(v)$. Thus, an equivalent problem is given by $A^0 = \{\{(u, x)\}\}$ and $A^1 = \{\{(u, 1-x)\}\}$. The case $|A^1| = 1$, $|A^0| > 1$ follows by symmetry.

In Case I(b), let p_1 and p_2 be the (at most) two partial states in A^0. At most, A^1 contains q, $\overline{p}_1$, $\overline{p}_2$, and an unbounded number of partial states that contain $(u, 1-x)$ and a variable-value pair outside q. Since the second variable-value pair does not interfere with either p_1 or p_2, it can be satisfied and ignored, so replacing all such partial states in A^1 with a single fourth partial state $\{(u, 1-x)\}$ results in an equivalent problem.

In Cases I(a) and I(b), any solution to a reduced problem can be projected back onto the original problem, inserting the pre-conditions that were removed if necessary. □

Theorem 14. *Each inverted fork problem $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(2)$ can be solved in polynomial time.*

Proof. Due to Lemma 13 we only need to consider 4 predecessors u_1–u_4 of v. We can now determine $n(v)$ using dynamic programming. For each $x \in \{0, 1\}$, each predecessor u_i of v, $i \in \{1, 2, 3, 4\}$, and each number of changes $0 \leq \nu(u_i) \leq n(u_i)$, let $Q(x, \nu(u_1), \ldots, \nu(u_4))$ be the maximum number of times that v can change given that its value is currently x and that u_i has changed $\nu(u_i)$ times.

The base case is $Q(x, n(u_1), \ldots, n(u_4)) = 0$ for $x = 0$ or $x = 1$, since v cannot change further for at least one value of x when u_1–u_4 have changed a maximum number of times. To solve any other subproblem, choose a partial state $p \in A^{1-x}$ and increase $\nu(u_i)$ by one for each u_i such that $(u_i, \nu(u_i) \bmod 2)$ disagrees with p. Recursively compute the maximum number of changes for v in the resulting subproblem. The value for $Q(x, \nu(u_1), \ldots, \nu(u_4))$ is obtained by taking the maximum over all such p and adding 1. By definition, $n(v) = Q(0, 0, \ldots, 0)$.

To extract the corresponding plan, store the partial state p that maximizes each value $Q(x, \nu(u_1), \ldots, \nu(u_4))$. The resulting table holds $2 \prod_{i=1}^{4} (1 + n(u_i))$ values, and dynamic programming can be done in time $6 \prod_{i=1}^{4} (1 + n(u_i))$ since there are at most 6 actions in A. If c is an upper bound on the values $n(u_i)$, the worst-case complexity is $O(c^4)$, which is polynomial in the size of the input. □

Remark. The dynamic programming algorithm described here uses the same idea underlying the algorithm of Brafman and Domshlak (2003) for solving planning problems with polytree causal graphs of bounded indegree (although in that work, the algorithm is expressed in terms of graphs).

Lemma 15. *Each inverted fork problem $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(3)$ can be reduced in polynomial time to an equivalent problem $\langle v, U', n, A'\rangle$ such that $|U'| \leq 10$ and $|A'| \leq 30$.*

Just like the proof of Lemma 13, the proof of Lemma 15 is based on a case-by-case analysis. However, the number of cases is much larger and we omit the details here. The upper bound on the number of actions follows from a situation equivalent to Case II for $\boldsymbol{\mathcal{I}}(2)$. Namely, in this situation there are three partial states in A^0 that do not intersect among themselves. This allows up to 27 actions of the other type. An additional complication is that if A^0 has many actions, not all of them intersect actions in A^1 in the same point, like in Case I for $\boldsymbol{\mathcal{I}}(2)$.

Theorem 16. *Each inverted fork problem $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(3)$ can be solved in polynomial time.*

Proof. Due to Lemma 15, we can use the same dynamic programming algorithm as in the proof of Theorem 14 to compute $n(v)$. The worst-case complexity is $O(c^{10})$ where c is an upper bound on $n(u)$ for each predecessor u of v. □

Corollary 17. *For planning problems in $\mathbf{P}(2)$ and $\mathbf{P}(3)$, plan generation can be done in polynomial time.*

Note that for $k = 4$, the case-by-case analysis of Lemmas 13 and 15 becomes so large that it is unclear to us whether the same result applies to inverted fork problems in $\boldsymbol{\mathcal{I}}(4)$.

$\mathbf{P}(k)$

In this section, we consider inverted fork problems in $\boldsymbol{\mathcal{I}}(k)$. Although we have been unable to extend the analysis of the previous section to $\boldsymbol{\mathcal{I}}(k)$ for $k > 3$, we can still show interesting properties on this type of problems. We prove that *any* inverted fork problem in $\boldsymbol{\mathcal{I}}(k)$ has a cover whose size is bounded by $4^k k^k$. That is, inverted fork problems with bounded dependence have bounded-size covers.

We highlight two consequences of this fact. First, we can improve the bound on $n(v)$ of Lemma 10, since it is no longer necessary to consider all variables in U, but only those belonging to the cover. A second consequence is that we can solve problems in $\boldsymbol{\mathcal{I}}(k)$ in polynomial time if the numbers $n(u)$ are bounded.

Definition 18. *Let $\boldsymbol{\mathcal{I}}(i,j)$ be the class of inverted fork problems such that $|p| \leq i$ for each $p \in A^0$ and $|q| \leq j$ for each $q \in A^1$.*

Definition 19. *A* cut $G \subseteq U \times \{0,1\}$ *for inverted fork problem $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(i,j)$ is a value set such that for each $p \in A^0$, $|p \cap G| \geq 1$, and for each $q \in A^1$, $|\overline{q} \cap G| \geq 1$. A* minimum cut *is a minimum such value set.*

The notion of a cut is weaker than that of a cover. A cut G is a subset of $U \times \{0,1\}$ that every action in $p \in A^0$ and $q \in A^1$ intersects *with*, that is, $|p \cap G| \geq 1$ and $|\overline{q} \cap G| \geq 1$. In contrast, a cover C is a subset of $U \times \{0,1\}$ that every pair of actions $(p,q) \in A^0 \times A^1$ intersects *within*, that is, $|(p \cap \overline{q}) \cap C| \geq 1$. Cuts are easy to find since, by definition of inverted fork problems, each pair of actions $(p,q) \in A^0 \times A^1$ intersects.

Lemma 20. *A minimum cut G for $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(i,j)$ has size $|G| \leq i + j - 1$.*

Proof. Consider any $(p,q) \in A^0 \times A^1$, and let $G = p \cup \overline{q}$. From the definitions of $\langle v, U, n, A\rangle$ and $\boldsymbol{\mathcal{I}}(i,j)$ it follows that $|G| \leq i + j - 1$. For each $p' \in A^0$ it holds that $|p' \cap \overline{q}| \geq 1$, implying $|p' \cap G| \geq 1$. For each $q' \in A^1$ it holds that $|p \cap \overline{q}'| \geq 1$, implying $|\overline{q}' \cap G| \geq 1$. Thus G is a cut. □

To prove that minimum covers also have bounded size, we show how to extend a cut G into a cover. If G is a cut that is not a cover, then there are pairs of actions $(p,q) \in A^0 \times A^1$ that intersect outside G. In particular, if we remove the values of G from $U \times \{0,1\}$, these pairs of actions still intersect. Moreover, they have smaller dependence. The following lemma concretizes this argument.

Lemma 21. *Let G be a cut for $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(i,j)$ and let S, $|S| < i$, be a subset of G. Let $K^0 = \{p - G : p \in A^0 \wedge p \cap G = S\}$ and $K^1 = \{q - \overline{G} : q \in A^1 \wedge \overline{q} \cap S = \emptyset\}$. If K^0 and K^1 are non-empty, $\langle v, U, n, \langle K^0, K^1\rangle\rangle \in \boldsymbol{\mathcal{I}}(i - |S|, j - 1)$.*

Proof. Since each $p \in A^0$ has size at most i and $p \cap G = S$, $p - G$ has size at most $i - |S|$. Since each $q \in A^1$ has size at most j and $|\overline{q} \cap G| \geq 1$, $q - \overline{G}$ has size at most $j - 1$. Finally, $p \cap G = S$ and $\overline{q} \cap S = \emptyset$ imply $p \cap G \cap \overline{q} = S \cap \overline{q} = \emptyset$. Since $|p \cap \overline{q}| \geq 1$ by definition of $\langle v, U, n, A\rangle$, $p \cap G \cap \overline{q} = \emptyset$ implies that p and $\overline{q}$ intersect outside G, so $|(p - G) \cap (\overline{q} - G)| \geq 1$, implying that $\langle v, U, n, \langle K^0, K^1\rangle\rangle$ is an inverted fork problem. □

The key argument of the previous lemma is that the property of inverted fork problems is preserved, namely that each pair of actions $(p,q) \in K^0 \times K^1$ of the new inverted fork problem intersect. We can now use an inductive argument to prove a bound on the size of a minimum cover.

Lemma 22. *Let $\langle v, U, n, \langle A^0, A^1\rangle\rangle \in \boldsymbol{\mathcal{I}}(i,j)$ be an inverted fork problem, and let $M^{i,j}$ be the size of a minimum cover for $\langle A^0, A^1\rangle$. For each $1 \leq i, j \leq k$, $M^{i,j} < 2^i(i + j - 1)^i$.*

Proof. By induction on i and j. If $i = 1$, each partial state $p \in A^0$ contains a single variable-value pair. Since $|p \cap \overline{q}| \geq 1$ for each partial state $q \in A^1$, A^0 cannot contain more than j partial states (else a partial state in A^1 could not intersect all of them). The union of the partial states in A^0 forms a cover of size at most j. Hence $M^{1,j} \leq j < 2^1(1+j-1)^1 = 2j$. By symmetry, $M^{i,1} \leq i < 2^i(i + 1 - 1)^i = (2i)^i$.

For $i > 1$ and $j > 1$, consider a minimum cut G for $\langle v, U, n, \langle A^0, A^1\rangle\rangle$. In the worst case, a minimum cover C for $\langle A^0, A^1\rangle$ contains all variable-value pairs in G. We bound the number of variable-value pairs of C outside G in the following way. Consider all subsets S of G of size less than i, and construct the inverted fork problem $\langle v, U, n, \langle K^0, K^1\rangle\rangle \in \boldsymbol{\mathcal{I}}(i - |S|, j - 1)$ described in Lemma 21. In the worst case, C contains all variable-value pairs in a minimum cover for $\langle K^0, K^1\rangle$. Summing over all such subsets we obtain the following inequality for $M^{i,j}$:

$$
\begin{aligned}
M^{i,j} &\leq |G| + \sum_{S \subseteq G, |S| < i} M^{i-|S|, j-1} = \\
&= |G| + \sum_{m=1}^{i-1} \binom{|G|}{i-m} M^{m,j-1} < \\
&< |G| + \sum_{m=1}^{i-1} \binom{i+j-1}{i-m} 2^m (m + j - 2)^m \leq \\
&\leq |G| + \sum_{m=1}^{i-1} (i + j - 1)^{i-m} 2^m (i + j - 1)^m < \\
&< \left(1 + \sum_{m=1}^{i-1} 2^m\right) (i + j - 1)^i < 2^i (i + j - 1)^i.
\end{aligned}
$$

The inequality uses Lemma 20, the inductive argument, and the fact that $|G| \leq (i + j - 1) < (i + j - 1)^i$ for $i > 1$. □

Lemma 23. *For each $\langle v, U, n, \langle A^0, A^1\rangle\rangle \in \boldsymbol{\mathcal{I}}(k)$, the size of a minimum cover for $\langle A^0, A^1\rangle$ is strictly bounded by $4^k k^k$.*

Proof. By definition, $\langle v, U, n, \langle A^0, A^1\rangle\rangle$ belongs to $\boldsymbol{\mathcal{I}}(k,k)$. Lemma 22 implies that a minimum cover for $\langle A^0, A^1\rangle$ has size $M^{k,k} < 2^k(k + k - 1)^k = (4k - 2)^k < 4^k k^k$. □

Remark. The bound in Lemma 22 (and hence in Lemma 23) is not tight, since it assumes that the covers for the inverted fork problems corresponding to subsets S of G are disjoint. This is not true in general since the sets K^1 of such inverted fork problems share many elements. However, the bound is at least exponential in k since we have observed minimum covers with size $O(2^k)$.

Theorem 24. *If c and k are constants and $n(u) < c$ for each $u \in U$, the inverted fork problem $\langle v, U, n, \langle A^0, A^1 \rangle\rangle \in \boldsymbol{\mathcal{I}}(k)$ can be solved in polynomial time. Moreover, the output $n(v)$ is bounded by $4^k k^k c$.*

Proof. Let C be a minimum cover for $\langle A^0, A^1 \rangle$. Then $n(v) \leq 1 + \sum_{u \in W(C)} n(u)$ by Lemma 10 and $|W(C)| \leq |C| < 4^k k^k$ by Lemma 23. Since $n(u) < c$, this implies $n(v) \leq 4^k k^k c$, which is constant (albeit large) for constant k. We compute the output $\langle n(v), \pi \rangle$ via iterative deepening, trying all action sequences of increasing constant length, with worst-case complexity $O(|A|^{n(v)})$. □

Remark. Clearly, a constant bound which is exponential in k is too large to be useful in practice. However, we believe that the bound could be significantly improved.

Using this result, we can define a subclass of planning problems in $\mathbf{P}(k)$ that can be solved in polynomial time.

Definition 25. *If $G = (V, E)$ is an acyclic graph, we define the depth $D(v)$ of a node $v \in V$ as the longest path in the graph between a source node and v.*

Definition 26. *Let $\mathbf{P}(k, d, b)$ be the subclass of $\mathbf{P}(k)$ such that $D(v) < d$ for each variable $v \in V$ with more than b predecessors, where $D(v)$ is the depth of v in the causal graph of P.*

Theorem 27. *For constant k, d, and b, plan generation for $\mathbf{P}(k, d, b)$ can be done in polynomial time.*

Proof. We show by induction on v that $N(v)$ and the corresponding plan for $P(v)$ can be computed in polynomial time. Moreover, $N(v)$ is either ∞ or bounded by a constant for each v such that $D(v) < d$. If v has no predecessor in the causal graph, its depth is 0 and we can compute $N(v)$ and the corresponding plan in polynomial time using one of the easy cases in Lemma 5.

If $D(v) < d$, then by definition of depth $D(u) < d$ for each predecessor u of v. By induction, $N(u)$ is either ∞ or bounded by a constant. Because of Proposition 12, computing $N(v)$ and the corresponding plan for $P(v)$ is equivalent to solving an inverted fork problem in $\boldsymbol{\mathcal{I}}(k)$ with v as root variable. Note that any predecessor u of v such that $N(u) = \infty$ is excluded from the inverted fork problem by definition of $\mathbf{pred}(v)$. We can now apply Theorem 24 to solve the inverted fork problem in polynomial time.

Finally, for each variable $v \in V$ with no more than b predecessors, we can apply the dynamic programming algorithm described in the proof of Theorem 14 to solve the corresponding inverted fork problem. Instead of 4 predecessors there now are b predecessors, so the complexity of dynamic programming is exponential in b. □

Conclusion

We have presented two novel complexity results for the class $\mathbf{P}(k)$ of planning problems with polytree causal graphs, binary variables, and k-dependent actions. First, we showed that plan generation for the classes $\mathbf{P}(2)$ and $\mathbf{P}(3)$ can be done in polynomial time. In addition, we showed that for any constant k, plan generation for $\mathbf{P}(k)$ is polynomial if variables with unbounded indegree have bounded depth. Both algorithms are based on a reduction from planning problems in $\mathbf{P}(k)$ to inverted fork problems in $\boldsymbol{\mathcal{I}}(k)$.

As long as an inverted fork problem can be reduced to an equivalent problem with a constant number of variables and actions, we can apply dynamic programming to solve it. The existence of a constant-size minimum cover for $\boldsymbol{\mathcal{I}}(k)$ hints at the possibility of doing this for any fixed k. However, nothing prevents the pre-conditions of actions from intersecting outside a cover, so a more careful analysis is necessary to extend the result for $\boldsymbol{\mathcal{I}}(2)$ and $\boldsymbol{\mathcal{I}}(3)$ to $\boldsymbol{\mathcal{I}}(k)$ for fixed $k > 3$.

As a consequence of our second result, the maximum number of changes $n(u)$ of the predecessors of v play an important role for the complexity of solving inverted fork problems. In addition, imposing a bound c on the number of such changes is similar to the notion of bounded local depth introduced by Brafman and Domshlak (2006). However, our algorithm does not require *all* variables to have bounded local depth, as long as the other variables either a) can change an arbitrary number of times, or b) have bounded indegree.

References

Brafman, R., and Domshlak, C. 2003. Structure and Complexity in Planning with Unary Operators. *Journal of Artificial Intelligence Research* 18:315–349.

Brafman, R., and Domshlak, C. 2006. Factored Planning: How, When, and When Not. In *AAAI*, 809–814.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69:165–204.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Chen, H., and Giménez, O. 2008. Causal Graphs and Structurally Restricted Planning. In *ICAPS*, 36–43.

Edelkamp, S. 2001. Planning with pattern databases. In *ECP*, 13–24.

Giménez, O., and Jonsson, A. 2008. The Complexity of Planning Problems with Simple Causal Graphs. *Journal of Artificial Intelligence Research* 31:319–351.

Giménez, O., and Jonsson, A. 2009. Planning over Chain Causal Graphs for Variables with Domains of Size 5 Is NP-Hard. *Journal of Artificial Intelligence Research* 34:675–706.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, 1007–1012.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 176–183.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Katz, M., and Domshlak, C. 2008a. New Islands of Tractability of Cost-Optimal Planning. *Journal of Artificial Intelligence Research* 32:203–288.

Katz, M., and Domshlak, C. 2008b. Structural Patterns Heuristics via Fork Decompositions. In *ICAPS*, 182–189.

Knoblock, C. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2):243–302.

Williams, B., and Nayak, P. 1997. A reactive planner for a model-based executive. In *IJCAI*, 1178–1185.