The Influence of k-Dependence on the Complexity of Planning

Omer Giménez[a], Anders Jonsson[b]

[a]*Dept. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Jordi Girona 1-3, 08034 Barcelona, Spain*
[b]*Dept. of Information and Communication Technologies, Universitat Pompeu Fabra, Roc Boronat 138, 08018 Barcelona, Spain*

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 of interest because k is a constant for all but a few of the current benchmark domains in planning, and is known to have implications for tractability. In this paper, we present an algorithm for solving planning problems in $\mathbf{P}(k)$, the class of k-dependent planning problems with binary variables and polytree causal graphs. We prove that our algorithm runs in polynomial time when k is a fixed constant. If, in addition, the causal graph has bounded depth, we show that plan generation is linear in the size of the input. Although these contributions are theoretical due to the limited scope of the class $\mathbf{P}(k)$, suitable reductions from more complex planning problems to $\mathbf{P}(k)$ could potentially give rise to fast domain-independent heuristics.

Keywords: planning, tractable planning, computational complexity

1. 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,

Email addresses: omer.gimenez@upc.edu (Omer Giménez), anders.jonsson@upf.edu (Anders Jonsson)

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.

In this paper we devise a polynomial-time algorithm for solving planning problems in $\mathbf{P}(k)$, the class of k-dependent planning problems with binary variables and polytree causal graphs, for any fixed value of k. This extends the results of a previous version of the paper (Giménez and Jonsson, 2009a), which only proved polynomial-time complexity for $\mathbf{P}(2)$ and $\mathbf{P}(3)$. Being polynomial-time solvable is of special interest since it is 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 (Amir and Engelhardt, 2003), in which planning problems are divided into subproblems whose independent solutions are combined to produce a global solution.

1.1. 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 (Chen et al., 2008; Jonsson, 2009). Also, the Fast Downward planner (Helmert, 2006b) is based on the domain-independent causal graph heuristic, which approximates the cost of solving a 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, 2009b). For any infinite family $\mathcal{C}$ of directed graphs whose underlying undirected graph is connected, no polynomial-time algorithm exists for solving the class of

planning problems with multi-valued variables and causal graphs in $\mathcal{C}$ unless standard complexity-theoretic assumptions fail (Chen and Giménez, 2008).

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 in polynomial time (Williams and Nayak, 1997; Helmert, 2006b). 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).

This paper focuses on two other restrictions, namely polytree causal graphs and k-dependence. A *polytree* is a directed graph whose underlying undirected graph (i.e., with every directed edge replaced by an undirected one) is acyclic. 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.

Katz and Domshlak (2008a) 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. The authors then proceeded to define the subclass $\mathbf{P}(k)$ of $\mathbf{P}$, in which planning problems have 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 fixed 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.

1.2. Our Contribution

In this work, we present several novel tractability results for the class $\mathbf{P}(k)$ of planning problems. First and foremost, we show that planning problems in $\mathbf{P}(k)$ can be solved in polynomial time, for any fixed value of k. This result is more general than in the previous version of our paper (Giménez and Jonsson, 2009a), in which polynomial-time complexity of the class $\mathbf{P}(k)$

was proven for $k = 2$ and $k = 3$, but where the proof for $k > 3$ required bounded depth and/or bounded indegree. We also show that if the causal graph has bounded depth, the complexity of solving any instance of $\mathbf{P}(k)$ is linear in the size of the input.

The key insight behind both algorithms is that although a variable v may have an unbounded number of predecessors in the causal graph, only a constant number 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. We can then solve a reduced problem of constant size that only considers the relevant predecessors.

Although our algorithm shows that $\mathbf{P}(k)$ is a polynomial-time solvable problem for every k, the constants in the polynomial expression of complexity are too large to be useful in practice. Even in the case $k = 2$, the upper bound on the running time of the algorithm involves exponents with approximately 10^{14} digits! It appears, however, that these large numbers are a side-effect of our algorithm being valid for all values of k: to keep the algorithm and the proof as simple as possible, we generalize in ways that do not exploit the structure of $\mathbf{P}(k)$ for small k. For this reason, we also perform a case-by-case analysis of planning problems in $\mathbf{P}(2)$ and obtain much lower bounds on the size of the relevant predecessors and actions, leading to an $O(n^5)$ algorithm with no large constants involved. On the other hand, applying the same case-by-case analysis to planning problems in $\mathbf{P}(3)$ becomes very tedious and results in a much larger bound on the complexity.

From a theoretical perspective, our result regarding $\mathbf{P}(k)$ contributes a novel element to the small set of known tractable classes of planning problems. Furthermore, $\mathbf{P}(k)$ imposes three restrictions on planning problems: binary variables, polytree causal graphs, and k-dependent actions. It is interesting to note that it is the combination of these three restrictions that makes it possible to solve planning problems in $\mathbf{P}(k)$ in polynomial time. If variables have domains larger than 4, Giménez and Jonsson (2009b) showed that plan existence is NP-hard, even if causal graphs are directed paths (implying they are polytree) and actions are 1-dependent. If the causal graph is acyclic but not polytree, Brafman and Domshlak (2003) showed that planning is NP-hard, even for binary variables and causal graphs with bounded degree, implying bounded dependence. Finally, if we drop the bound on dependence, Giménez and Jonsson (2008) showed that plan existence is NP-

IPC2		**IPC5**	
Blocks	0/4	Openstacks	95/98
Miconic-STRIPS	1/3	Pathways	1/3
FreeCell	1/4	Pipesworld	0/8
Logistics	1/2	Rovers	2/4
Schedule-STRIPS	1/9	Storage	1/5
IPC3		TPP	1/4
Depots	1/6	Trucks	6/9
DriverLog	1/3	**IPC6**	
ZenoTravel	1/2	Cyber security	8/32
Rovers	2/3	Elevator	1/3
Satellite	3/4	Openstacks	9/10
IPC4		PARC printer	1/8
Airport	25/37	Peg solitaire	0/4
Pipesworld-No tankage	0/4	Scanalyzer	0/5
Pipesworld-Tankage	0/5	Sokoban	0/5
Promela optical	46/335	Transport	1/3
Promela philosophers	31/154	Woodworking	2/8
PSR	15/16		

Table 1: Largest k/p for IPC STRIPS planning domains, where k is the dependence of actions and p is the total number of pre-conditions.

complete, even for binary variables and polytree causal graphs.

From a practical perspective, our result offers the possibility to devise heuristics based on reductions from more complex planning problems to $\mathbf{P}(k)$. Although the efficiency of our algorithm for solving planning problems in $\mathbf{P}(k)$ decreases as k increases, we show below that the value of k rarely exceeds 2 for current benchmark domains in planning. However, the true significance of our result in this respect remains to be seen since no such heuristic has yet been implemented.

1.3. k-Dependence in Practice

Table 1 shows the largest k of actions in STRIPS planning domains from the International Planning Competition (IPC). The values were computed using the translator of Helmert (2006a), used in the successful Fast Downward and LAMA planners. The translation removes static pre-conditions and only

considers grounded actions. The translated problems have a multi-valued variable representation, in which actions are defined in terms of prevail-conditions and effects. Prevail-conditions correspond exactly to the dependence of actions, while effects correspond to variable value changes. In addition to k, the table shows the largest p of actions, defined as the sum of prevail-conditions and effects, which roughly corresponds to the total number of pre-conditions. Some of the problems in the Pipesworld-No tankage and Pipesworld-Tankage domains from IPC4 did not translate, so the table shows a partial result for those entries.

We see that for most STRIPS planning domains, k is a small constant, typically less than or equal to 2. The exception is for domains that have been translated to STRIPS from ADL. Although the values of k were obtained from a multi-valued variable representation, the values should remain constant for binary variables. In the multi-valued variable representation, variable domains are sets of fluents, so action effects change the value of one fluent from true to false, and the value of another from false to true. In neither case is the fluent unaffected by the action.

1.4. 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 et al., 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. To this end, they made use of the tractability result regarding $\mathbf{P}(1)$ of Katz and Domshlak (2008a) to define and analyze the *inverted-fork* CGSP decomposition.

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 $\mathbf{P}(1)$. However, at present these ideas are conjectural at best since we have not implemented a corresponding heuristic based on $\mathbf{P}(k)$.

1.5. Organization of the Paper

Some of the proofs in the paper are very technical and may appear daunting. For this reason, we have tried to organize the paper in a way that makes it more accessible to the reader. In particular, Section 3 contains the main results regarding the class $\mathbf{P}(k)$, appearing as Theorems 3.18, 3.22, and 3.24, respectively. However, the proofs of several key propositions are left for Sections 4 and 5 and Appendix A, which could thus be skipped by someone not interested in the proof details. The remaining sections of the paper are Section 2, which provides the notation used in later sections, and Section 6, which concludes the paper with a summary and discussion.

2. Notation

Let V be a set of propositional variables. A *literal* l is a truth assignment to some variable $v \in V$, i.e., v or $\neg v$. The *complement* $\bar{l}$ of l is the negation of l, i.e., $\bar{l} = \neg l$. A *partial state* p is a set of literals such that for each $l \in p$, it holds that $\bar{l} \notin p$. The complement $\overline{p}$ of p is defined as $\overline{p} = \{l : \bar{l} \in p\}$. For a set of literals L, let $V(L) \subseteq V$ be the subset of variables that appear as literals in L. A *state* s is a partial state such that $V(s) = V$. Given a subset $W \subseteq V$ of variables, $L(W) = \{v, \neg v : v \in W\}$ is the set of all literals on W. A set of literals L is *complete* if, for each $l \in L$, it holds that $\bar{l} \in L$, i.e., $L = L(V(L))$.

A planning problem is a tuple $P = \langle V, \mathsf{init}, \mathsf{goal}, A \rangle$, where V is a set of propositional variables, init is an initial state, goal is a partial goal state, and A is a set of actions. An action $a = \langle \mathsf{pre}(a); \mathsf{post}(a) \rangle \in A$ consists of a partial

state $\mathsf{pre}(a)$ called the *pre-condition* and a partial state $\mathsf{post}(a)$ called the *post-condition*. Action a is applicable in any state s such that $\mathsf{pre}(a) \subseteq s$, and applying a in state s results in a new state $s \oplus a = (s - \mathsf{post}(a)) \cup \mathsf{post}(a)$.

A plan $\pi = \langle a_1, \ldots, a_k \rangle$ is a sequence of actions from A. We say that π is *valid* if it holds that for each $1 \leq i \leq k$, action a_i is applicable in state s_i, defined as $s_1 = \mathsf{init}$ and $s_i = s_{i-1} \oplus a_{i-1}$ for $2 \leq i \leq k+1$. If, in addition, $\mathsf{goal} \subseteq s_{k+1}$, we say that π *solves* the planning problem P.

The causal graph (V, E) of a planning problem P is a directed graph with the variables in V 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 $u \in V(\mathsf{pre}(a)) \cup V(\mathsf{post}(a))$ and $v \in V(\mathsf{post}(a))$. The causal graph is a *polytree* if the underlying undirected graph has no cycles. Since any action with two or more literals in the post-condition induces a cycle in the causal graph, a polytree causal graph implies that all actions are unary, i.e., $|\mathsf{post}(a)| = 1$ for each $a \in A$.

3. The Class $\mathbf{P}(k)$

In this paper we study the class $\mathbf{P}(k)$ of planning problems, where $1 \leq k < |V|$ is a bound on the dependence of actions. A planning problem P is in $\mathbf{P}(k)$ if and only if it has propositional variables, polytree causal graph, and k-dependent actions. Using the notation from the previous section, an action a is k-dependent if and only if $|V(\mathsf{pre}(a)) - V(\mathsf{post}(a))| \leq k$.

Let P be a planning problem in $\mathbf{P}(k)$. In this section we show that the problem of generating a plan solving P is equivalent to generating, for each variable $v \in V$, a plan that changes the value of v the maximum possible number of times. Although a similar 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 what we call k-dependent MAX-CHANGE problems, previously also known as "inverted-fork computational problems" (Giménez and Jonsson, 2009b).

In Section 4, we prove that the complexity of solving MAX-CHANGE problems with k-dependent actions is polynomial for fixed k. Using our previous reduction, this implies that plan generation for $\mathbf{P}(k)$ has polynomial complexity. We also study the special case for which the causal graph has bounded depth, i.e., the length of a path between any pair of nodes in the graph is bounded by a constant. In this case, the complexity of plan generation for $\mathbf{P}(k)$ is linear in the number of actions of the problem.

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

In this section we prove that plan generation for $\mathbf{P}(k)$ can be reduced to solving a series of MAX-CHANGE problems. We first define the restriction $P(v)$ of a planning problem P to a variable $v \in V$ and its ancestors in the causal graph. We then define the maximum number of times $N(v)$ that the value of v can change while solving $P(v)$. We show that solving P is equivalent to computing the exact value of $N(v)$ for each variable $v \in V$ and generating a plan solving $P(v)$ that changes the value of v exactly $N(v)$ times.

To determine $N(v)$ and generate a corresponding plan for $P(v)$, it is sufficient to know, for each immediate predecessor u of v in the causal graph, the value of $N(u)$ and a corresponding plan for $P(u)$. All other ancestors of v can thus be ignored. This is the basis of our definition of MAX-CHANGE problems and gives rise to a bottom-up procedure for plan generation for $\mathbf{P}(k)$.

Without loss of generality, we assume throughout that $\mathsf{init} = \bigcup_{v \in V} \{v\}$. Otherwise, for each $v \in V$ such that $\neg v \in \mathsf{init}$, we can just invert the truth value of v, and change accordingly all actions which affect or depend on v. For each $v \in V$, let $U = \{u_1, \ldots, u_m\}$ be the set of predecessors of v in the causal graph, i.e., $(u_i, v) \in E$ for each $u_i \in U$. We recursively define the set $\mathsf{anc}(v)$ of *ancestors* of v as $\bigcup_{u_i \in U} (\{u_i\} \cup \mathsf{anc}(u_i))$.

Example 3.1. We define an example planning problem $P = \langle V, \mathsf{init}, \mathsf{goal}, A \rangle \in \mathbf{P}(3)$ as follows. The set of variables is $V = \{v_1, \ldots, v_{13}\}$, the initial state is $\mathsf{init} = \{v_1, \ldots, v_{13}\}$, and the goal state is $\mathsf{goal} = \{\neg v_9, \neg v_{10}, v_{11}, v_{12}, \neg v_{13}\}$. The actions in A are as follows:

Action	Pre-condition	Effect
a_1^1	$\emptyset$	$v_1 \rightarrow \neg v_1$
a_1^2	$\emptyset$	$\neg v_1 \rightarrow v_1$
a_2^1	$\emptyset$	$v_2 \rightarrow \neg v_2$
a_3^1	$\neg v_1$	$v_3 \rightarrow \neg v_3$
a_4^1	$\neg v_2$	$v_4 \rightarrow \neg v_4$
a_4^2	v_2	$\neg v_4 \rightarrow v_4$
a_6^1	$\emptyset$	$v_6 \rightarrow \neg v_6$
a_6^2	$\emptyset$	$\neg v_6 \rightarrow v_6$
a_7^1	v_3	$v_7 \rightarrow \neg v_7$
a_7^2	$\neg v_3$	$\neg v_7 \rightarrow v_7$

Action	Pre-condition	Effect
a_8^1	v_4	$v_8 \rightarrow \neg v_8$
a_9^1	$v_4, \neg v_5, v_6$	$v_9 \rightarrow \neg v_9$
a_9^2	$\neg v_4, \neg v_6, v_7$	$v_9 \rightarrow \neg v_9$
a_9^3	$v_5, v_6, \neg v_7$	$\neg v_9 \rightarrow v_9$
a_{10}^1	v_8	$v_{10} \rightarrow \neg v_{10}$
a_{10}^2	$\neg v_8$	$\neg v_{10} \rightarrow v_{10}$
a_{11}^1	v_9	$\neg v_{11} \rightarrow v_{11}$
a_{12}^1	$\neg v_9$	$v_{12} \rightarrow \neg v_{12}$
a_{12}^1	v_9	$\neg v_{12} \rightarrow v_{12}$
a_{13}^1	$\neg v_{12}$	$v_{13} \rightarrow \neg v_{13}$

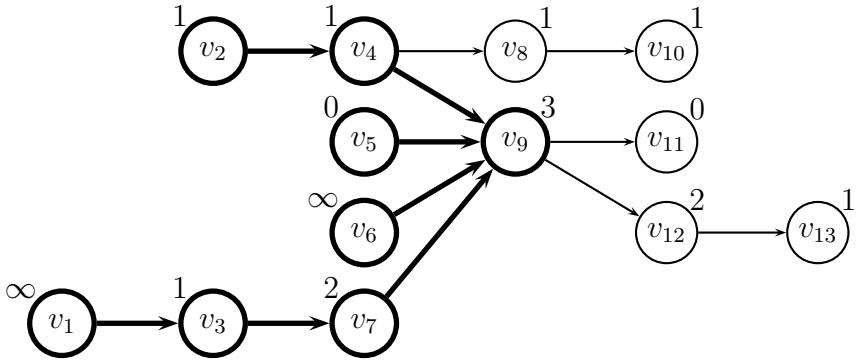

Figure 1: Polytree causal graph of the example planning problem P. In bold, the causal graph of $P(v_9)$. Each node v_i is labelled with the value $N(v_i)$ (see text for explanation).

Note that each action is 3-dependent, implying $P \in \mathbf{P}(3)$.

We now introduce a series of definitions which we later use to formalize the reduction from $\mathbf{P}(k)$ to k-dependent Max-Change problems.

Definition 3.2. Let $P = \langle V, \mathsf{init}, \mathsf{goal}, A \rangle$ be a planning problem in $\mathbf{P}(k)$. Define the planning problem $P(v)$ as the restriction of P to variables in $V' = \{v\} \cup \mathsf{anc}(v)$, i.e., $P(v) = \langle V', \mathsf{init}', \mathsf{goal}', A' \rangle$ where $\mathsf{init}' = \mathsf{init} \cap L(V')$, $\mathsf{goal}' = \mathsf{goal} \cap L(V')$, and $A' \subseteq A$ only contains actions a such that $V(\mathsf{post}(a)) \subseteq V'$.

Figure 1 shows the causal graph of the example planning problem P, as well as (in bold) the causal graph of the planning problem $P(v_9)$, which is a subgraph of the former. The figure also shows the values $N(v_i)$ for each variable $v_i \in V$, defined as follows:

Definition 3.3. For a variable $v \in V$, we define $N(v) \in \mathbb{N} \cup \{\infty, \bot\}$ as the maximum number of times that the value of v can change on a plan solving $P(v)$, with $\bot$ denoting that $P(v)$ is unsolvable.

Note that if $N(v) \notin \{\infty, \bot\}$ and $v \in V(\mathsf{goal})$, the parity of $N(v)$ must agree with the goal. Namely, if $v \in \mathsf{goal}$, $N(v)$ must be even, and if $\neg v \in \mathsf{goal}$, $N(v)$ 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.

As previously mentioned, 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 a polytree it is possible to compute these values bottom-up. In addition, we identify three special cases regarding the value of $N(u)$:

- If $N(u) = \bot$, it is impossible to satisfy the goal state on $\{u\} \cup \mathsf{anc}(u)$. Since u is a predecessor of v, it follows that $\{u\} \cup \mathsf{anc}(u) \subseteq \mathsf{anc}(v)$, making it impossible to satisfy the goal state on $V' = \{v\} \cup \mathsf{anc}(v)$. Thus $P(v)$ is unsolvable, implying $N(v) = \bot$.
- If $N(u) = 0$, the value of u can never change, so we do not need to consider u while computing $N(v)$, although actions with pre-condition $\neg u$ are inadmissible and should be removed.
- If $N(u) = \infty$, u does not restrict the number of times that v can change, since any pre-condition on u can always be satisfied by changing the value of u once more.

As a consequence of the above observations, unless $P(v)$ is unsolvable, the relevant predecessors for determining $N(v)$ are those whose values can change a positive finite number of times. Any other predecessors of v can be safely ignored. We formalize this idea in the following definition.

Definition 3.4. For each $v \in V$, let $\mathsf{relpred}(v) = \{u \in V : (u, v) \in E \wedge 0 < N(u) < \infty\}$ be the set of relevant predecessors of v.

Next, we define the set of relevant actions for changing the value of v. Here, we distinguish between those actions that set v to true (superscript 1) and those that set v to false (superscript 0). We first remove any actions with pre-condition $\neg u$ on some predecessor u of v with $N(u) = 0$, since such actions are always inadmissible. We then project the pre-conditions of the remaining actions onto the set $\mathsf{relpred}(v)$, since those are the variables that are relevant for determining $N(v)$.

Definition 3.5. For each $v \in V$, let $A_v^1 = \{\mathsf{pre}(a) \cap L(\mathsf{relpred}(v)) : a \in A \wedge \mathsf{post}(a) = \{v\} \wedge \nexists u \text{ s.t. } N(u) = 0 \wedge \neg u \in \mathsf{pre}(a)\}$ be the set of relevant actions that set v to true. Likewise, let A_v^0 be the set of relevant actions that set v to false.

Note that we abuse notation slightly by claiming that A_v^0 and A_v^1 are sets of actions, even though they are in fact sets of partial states. The reason is that there is a one-to-one correspondence between actions and partial states,

since we can set v to true in any state s for which there exists $p \in A_v^1$ such that $p \subseteq s$, by applying the associated action. For $A_v = \langle A_v^0, A_v^1 \rangle$, we also abuse notation by writing $|A_v|$ and $a \in A_v$ to mean $|A_v^0| + |A_v^1|$ and $a \in A_v^0 \cup A_v^1$, respectively. The sets A_v^0 and A_v^1 may each contain the empty partial state $\emptyset$; in particular, unless A_v^0 or A_v^1 is empty, this is always the case when $\mathsf{relpred}(v) = \emptyset$ (i.e. when v has no relevant predecessors).

Also note that the relevant predecessors u of v and the relevant actions for v are defined in terms of $N(u)$. In other words, the sets $\mathsf{relpred}(v)$, A_v^0, and A_v^1 can be computed bottom-up in the same fashion as $N(v)$, and are in fact needed to compute $N(v)$ and a corresponding plan for $P(v)$.

We now turn to the problem of determining $N(v)$. As we have already seen, if $N(u) = \bot$ for any predecessor u of v, then $N(v) = \bot$ must hold. In what follows, we assume that this is not the case, i.e., $N(u) \neq \bot$ for each predecessor u of v. We identify four easy cases for determining $N(v)$.

Lemma 3.6. *Let P be a planning problem in $\mathbf{P}(k)$, and $v \in V$ a variable such that $N(u) \neq \bot$ for each predecessor u of v. Then*

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

This classification is not exhaustive, i.e., there may be instances of v that do not fall into any of the four cases.

Proof. The first case states that if the goal value is $\neg v$ and there is no relevant action for setting v to false, the problem $P(v)$ is unsolvable, so $N(v) = \bot$. The second case states that there is no relevant action for setting v to false, but the goal value is not $\neg v$, or the goal value is v and there is no relevant action that resets v to true. In either case, we cannot change the value of v. The third case states that there exist relevant actions for setting v to false, but there is no relevant action for resetting v to true, so we can change the value of v at most once. 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 3.6 always holds, a necessary condition for the bottom-up procedure that we describe here.

Example 3.7. In the planning problem P from Example 3.1, variable v_1 has no predecessors in the causal graph, so $\mathsf{relpred}(v_1) = \emptyset$. Furthermore, $A^0_{v_1} = A^1_{v_1} = \{\emptyset\}$, since there is an action (a^1_1) with pre-condition $\emptyset$ and post-condition $\neg v_1$, and another action (a^2_1) with pre-condition $\emptyset$ and post-condition v_1. The pre-condition of both actions can be satisfied simultaneously, implying $N(v_1) = \infty$ due to Lemma 3.6. The same argument can be used to obtain $N(v_6) = \infty$.

For variables v_5 and v_{11}, there are no actions for setting the variable value to false, implying $A^0_{v_5} = A^0_{v_{11}} = \emptyset$. In other words, $N(v_5) = N(v_{11}) = 0$ due to Lemma 3.6. For variables v_2, v_3, v_8, and v_{13}, there are actions for setting the variable value to false, but no actions that reset the variable value to true. As a consequence, $|A^0_{v_2}| > 0$, $|A^0_{v_3}| > 0$, $|A^0_{v_8}| > 0$, and $|A^0_{v_{13}}| > 0$, but $A^1_{v_2} = A^1_{v_3} = A^1_{v_8} = A^1_{v_{13}} = \emptyset$. Thus $N(v_2) = N(v_3) = N(v_8) = N(v_{13}) = 1$ due to Lemma 3.6. For the remaining variables (v_4, v_7, v_9, v_{10}, v_{12}), we cannot apply an easy case to obtain the value of $N(v_i)$. Note that, strictly speaking, the above analysis would also require us to verify that $N(u) = \perp$ does not hold for some predecessor u of variables v_8, v_{11}, and v_{13}.

We next introduce the notion of a cover, which we use to provide a global bound on $N(v)$ for variables that do not belong to any of the four easy cases.

Definition 3.8. Let $\langle A^0, A^1\rangle$ be sets of partial states. A *cover* $C \subseteq L(V)$ for $\langle A^0, A^1\rangle$ is a set of literals such that for each $(p, q) \in A^0 \times A^1$, $|p \cap \overline{q} \cap C| \geq 1$.

Lemma 3.9. *Let C be a cover for $\langle A^0_v, A^1_v\rangle$. Then $N(v) \leq 1 + \sum_{u \in V(C)} N(u)$.*

Proof. Since C is a cover, we know that for each pair $(p, q) \in A^0_v \times A^1_v$ there exists at least one literal $l \in p$ such that $\neg l \in q$. As a consequence, whenever we change the value of v we cannot change its value again without first changing the value of at least one variable in $V(C)$. The maximum number of times we can do this is $\sum_{u \in V(C)} N(u)$. Possibly, we can change the value of v an additional time in the initial state. □

Lemma 3.10. *For each $v \in V$ such that $N(v) \notin \{\infty, \perp\}$, it holds that $N(v) \leq 1 + |\mathsf{anc}(v)|$.*

Proof. By induction on v, with respect to a topological order. If v has no ancestors, one of the easy cases of Lemma 3.6 always applies, so $N(v) \notin \{\infty, \perp\}$ implies $N(v) \leq 1 = 1 + 0 = 1 + |\mathsf{anc}(v)|$. Otherwise, $N(v) \notin \{\infty, \perp\}$ and Lemma 3.6 imply that there does not exist $(p, q) \in A^0_v \times A^1_v$ such that

$p \cap \overline{q} = \emptyset$. Then by definition $L(\mathsf{relpred}(v))$ is a cover for $\langle A_v^0, A_v^1 \rangle$. Since $N(u) \notin \{\infty, \bot\}$ for each $u \in \mathsf{relpred}(v)$, from Lemma 3.9 we obtain

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

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. □

As a consequence of Lemma 3.10, unless we can change the value of v an arbitrary number of times, $N(v)$ is bounded by the number of ancestors of v, which in turn is bounded by the number $|V|$ of variables of the problem.

We now proceed to define a particular class of problems that we call Max-Change problems. Just like planning problems, a Max-Change problem has a set of variables and a set of actions. However, unlike planning problems, a Max-Change problem does not have a goal state. Instead, the aim is to produce a sequence of actions that changes the value of a target variable v the maximum possible number of times. The input is the maximum number of times that the value of each predecessor u of v can change.

Our procedure for solving planning problems in $\mathbf{P}(k)$ constructs and solves Max-Change problems whenever we cannot apply one of the four easy cases in Lemma 3.6 for computing $N(v)$. For this reason, our definition of Max-Change problems requires the actions to be such that we are never in an easy case. Also, the maximum number of value changes of the predecessors of v is always a positive finite number, since the reduction only considers the relevant predecessors of v. Although more restrictive than necessary, this definition nevertheless makes some of our subsequent proofs easier.

Definition 3.11. A Max-Change 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 the maximum number of times that each variable $u \in U$ can change;
- $A = \langle A^0, A^1 \rangle$, where A^1 is a non-empty set of unary actions setting v to true, and A^0 is a non-empty set of unary actions setting v to false, i.e., non-empty sets of partial states on $L(U)$. 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 are true to begin with and that each variable $u \in U$ can change freely, but no more than $n(u)$ times; π is a sequence of actions in A attaining this maximum.

Note that both the choice of notation and the restrictions in the third and four items reflect our goal of associating a Max-Change problem instance to each variable v whose corresponding value $N(v)$ cannot be computed using one the four easy cases in Lemma 3.6. In this case, it always holds that $N(v) = n(v)$ or $N(v) = n(v) - 1$. In what follows, we show that $n(v) < \infty$ for any instance of Max-Change, ensuring that the output $\langle n(v), \pi \rangle$ is well defined.

Lemma 3.12. *Let $\langle v, U, n, \langle A^0, A^1 \rangle \rangle$ be a* Max-Change *problem, and let C be a cover for $\langle A^0, A^1 \rangle$. Then $n(v) \leq 1 + \sum_{u \in V(C)} n(u)$.*

Proof. The proof of Lemma 3.9 applies verbatim by substituting n for N. □

Since $L(U)$ is a cover for $\langle A^0, A^1 \rangle$, Lemma 3.12 implies that $n(v) \leq 1 + \sum_{u \in U} n(u) < \infty$. On the contrary, Lemma 3.9 does not imply $N(v) < \infty$, since it is possible that $N(u) = \infty$ for some predecessor u of v.

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. Without this condition, the size $n(v)$ of the output could be exponentially larger than the size of the input.

Definition 3.13. Let $\boldsymbol{\mathcal{I}}(k)$ be the class of Max-Change problems with k-dependent actions, i.e., $|p| \leq k$ for each $p \in A$.

Example 3.14. We define the following example MAX-CHANGE problem $\Pi = \langle v; U; n; A = \langle A^0, A^1 \rangle \rangle \in \boldsymbol{\mathcal{I}}(3)$:

$$\begin{aligned} U &= \{x, y, z, s, t, w\}, \\ n(u) &= \begin{cases} 1, & \text{if } u \in \{x, z, t\}, \\ 2, & \text{if } u \in \{y, s, w\}, \end{cases} \\ A^0 &= \{p_1 = \{x, y, z\}, p_2 = \{\neg x, \neg y, w\}\}, \\ A^1 &= \{q_1 = \{x, \neg y, t\}, q_2 = \{\neg x, y, s\}, q_3 = \{x, \neg z, w\}\}. \end{aligned}$$

Observe that Π satisfies the items of Definition 3.11.

We proceed to prove that plan generation for $\mathbf{P}(k)$ is polynomial-time reducible to solving MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(k)$.

Proposition 3.15. *Any planning problem $P \in \mathbf{P}(k)$ is polynomial-time reducible to a series of* MAX-CHANGE *problems in $\boldsymbol{\mathcal{I}}(k)$. There is an algorithm implementing the reduction with complexity $O(T|V|+|A|)$, where T is an upper bound on the values $N(v)$ for variables $v \in V$ with $N(v) < \infty$.*

In what follows, we explain the intuition behind the proof and describe an algorithm that performs the reduction in polynomial time. Although the idea is relatively intuitive, achieving the desired complexity $O(T|V| + |A|)$ requires a fairly sophisticated algorithm. For this reason, we have moved the proof of Proposition 3.15 to Appendix A.

The idea behind the reduction is to independently generate plans that change the value of each variable $v \in V$ a maximum number of times. More precisely, for each variable $v \in V$ in topological order, we compute $N(v)$ and a sequence of actions in $\langle A^0_v, A^1_v \rangle$ attaining this maximum. We first consider the easy cases in Lemma 3.6. If $N(v) = \bot$, the planning problem $P(v)$ has no solution, so neither does P. In this case we can just stop and report that P is unsolvable. If $N(v) = 0$, the empty plan attains the maximum. If $N(v) = 1$, any action $p \in A^0_v$ attains the maximum. If $N(v) = \infty$, we keep any pair of actions $p, q \in A^0_v \times A^1_v$ such that $p \cap \overline{q} = \emptyset$.

If we are not in an easy case, we ask for a solution $\langle n(v), \pi_v \rangle$ to the MAX-CHANGE problem $\langle v, \mathsf{relpred}(v), N, \langle A^0_v, A^1_v \rangle \rangle \in \boldsymbol{\mathcal{I}}(k)$. Note that we already know $N(u)$ for each $u \in \mathsf{relpred}(v)$ since u comes before v in topological order. The value of $N(v)$ is either $n(v)$ or $n(v) - 1$, depending on the goal value of v. If $N(v) = n(v) - 1$, we have to remove the last action from the sequence π_v to obtain a sequence changing v exactly $N(v)$ times.

Note that each resulting action sequence π_v only includes actions for changing the value of v. To obtain an actual plan for the subproblem $P(v)$, we have to *merge* the sequence for v with the plans for $P(u)$, for any predecessor u 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. We remark that it is very easy to merge the plans for $P(u)$ to obtain a plan for $P(v)$ in time $O(T|V|)$, but that it is considerably harder to obtain all such plans for all variables $v \in V$ in time $O(T|V| + |A|)$, as required by the proposition. The actual procedure is described in the appendix.

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 $P(v)$. Since the merged plan solves $P(v)$ for each $v \in V$, it also solves P.

Example 3.16. We illustrate how to construct the MAX-CHANGE problem associated with variable v_9 in the planning problem P from Example 3.1. The set of predecessors of v_9 is $\{v_4, v_5, v_6, v_7\}$, with values $N(v_4) = 1$, $N(v_5) = 0$, $N(v_6) = \infty$, and $N(v_7) = 2$, respectively. According to Definition 3.4, the set of relevant predecessors of v_9 is $\mathsf{relpred}(v_9) = \{v_4, v_7\}$. Furthermore, the action a_9^1 is inadmissible since it requires a pre-condition $(\neg v_5)$ that cannot be satisfied. Projecting the pre-conditions of the remaining actions onto $L(\mathsf{relpred}(v_9))$ we obtain $A_{v_9}^0 = \{\{\neg v_4, v_7\}\}$ and $A_{v_9}^1 = \{\{\neg v_7\}\}$. The corresponding MAX-CHANGE problem is given by $\langle v_9, \{v_4, v_7\}, N, \langle A_{v_9}^0, A_{v_9}^1 \rangle\rangle$.

We remark that the converse of Proposition 3.15 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 directed-path causal graph constructions proposed by Giménez and Jonsson (2008).

3.2. Complexity of Plan Generation for $\mathbf{P}(k)$

In this section we study the complexity of plan generation for the class $\mathbf{P}(k)$. We first prove that for fixed k, the complexity of plan generation for

$\mathbf{P}(k)$ is polynomial in the size of the input. Our approach is to study the complexity of solving MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(k)$, and apply Proposition 3.15 to obtain a result for $\mathbf{P}(k)$. We then study two special cases: the subclass of $\mathbf{P}(k)$ of planning problems whose causal graph has bounded depth, and the case $k = 2$.

The complexity results in this section are based on the following proposition regarding MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(k)$.

Proposition 3.17. *There is an algorithm solving* MAX-CHANGE *problems* $\Pi = \langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(k)$ *in time* $O(|A| + T^{f(k)})$*, where* T *is an upper bound on the values* $n(u)$ *of the predecessors* $u \in U$*.*

Since the proof of Proposition 3.17 is rather complex, we defer it to Section 4. The function $f(k)$ is an upper bound on the number of predecessors in U that we need to take into account while solving MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(k)$. It is characterized in Definition 4.17 and grows too quickly to be of practical use, yet is sufficient to prove polynomial-time complexity.

Using the result for $\boldsymbol{\mathcal{I}}(k)$, it is now straightforward to prove the following theorem regarding the complexity of plan generation for $\mathbf{P}(k)$.

Theorem 3.18. *For any fixed value of* k*, planning problems* $P \in \mathbf{P}(k)$ *can be solved in polynomial time with complexity* $O(|A|^{f(k)+1})$*.*

Proof. Proposition 3.15 states that the complexity of reducing P to a series of MAX-CHANGE problems is $O(T|V|+|A|)$. We need to solve at most $|V|$ such MAX-CHANGE problems, one for each variable $v \in V$. Due to Proposition 3.17, the complexity of doing so is $O(|V||A| + |V|T^{f(k)})$, where T is an upper bound on the values $n(u)$ for all such MAX-CHANGE problems. Recall that when we construct MAX-CHANGE problems, we only include predecessors u such that $0 < N(u) < \infty$. Furthermore, we set $n(u) \equiv N(u)$. Since $N(u) < \infty$, Lemma 3.10 implies that $n(u) = N(u) \leq 1 + \mathsf{anc}(u) \leq |V|$ for each such variable, so we obtain $T \leq |V|$.

We assume without loss of generality that the causal graph of the planning problem is connected. If it is not, we can simply break the planning problem into two or several parts that can be solved independently to produce a solution to the original problem. If the causal graph is connected, there has to be at least one action per $k + 1$ variables, since each action a can have at most k pre-conditions on variables not in $V(\mathsf{post}(a))$. As a consequence, $|V| \leq (k+1)|A|$, implying $|V| = O(|A|)$ and $T = O(|A|)$ since $T \leq |V|$ and k

is assumed to be fixed. Substituting $|A|$ for T and $|V|$, we obtain the desired bound $O(|A|^2 + |A| + |A|^2 + |A|^{f(k)+1}) = O(|A|^{f(k)+1})$. □

We also study the special case for which the depth of the causal graph is bounded. We define the depth $D(v)$ of a variable v in the causal graph to be the maximum length of a path from another variable to v. The depth of the causal graph is the maximum depth of any of its variables. We can now define the subclass of planning problems with bounded depth in $\mathbf{P}(k)$.

Definition 3.19. Let $\mathbf{P}(k, d)$ be the subclass of planning problems in $\mathbf{P}(k)$ whose causal graph has depth at most d.

In earlier work (Giménez and Jonsson, 2009a), we proved the following lemma regarding the existence of a constant-size cover for MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(k)$.

Lemma 3.20. *For each* MAX-CHANGE *problem* $\langle v, U, n, \langle A^0, A^1\rangle\rangle \in \boldsymbol{\mathcal{I}}(k)$, *the size of a minimum cover for* $\langle A^0, A^1\rangle$ *is at most* $4^k k^k$.

We can use this lemma to obtain the following result for planning problems in $\mathbf{P}(k, d)$.

Proposition 3.21. *For fixed values of* k *and* d, *there exists a constant* $T(k, d)$ *such that for each planning problem* $P \in \mathbf{P}(k)$ *and each variable* $v \in V$ *such that* $D(v) = d$ *and* $N(v) < \infty$, $N(v) \leq T(k, d)$.

Proof. Let C be a minimum cover for $\langle A^0_v, A^1_v\rangle$. Then we have $N(v) \leq 1 + \sum_{u \in V(C)} N(u)$ by Lemma 3.9 and $|V(C)| \leq |C| < 4^k k^k$ by Lemma 3.20. The fact that v has depth $D(v) = d$ in the causal graph implies that the depth of its predecessors u is $D(u) \leq d - 1$. We can now use induction on the variables to assume that $N(u) \leq T(k, d-1)$ for each predecessor u of v, implying $N(v) \leq 1 + 4^k k^k T(k, d-1)$.

If v has no predecessor in the causal graph, $N(v) < \infty$ implies $N(v) \leq 1$ since computing $N(v)$ can always be done using one of the easy cases in Lemma 3.6. It now follows by induction that $T(k, d) \equiv 1 + 4^k k^k T(k, d-1) = O((4^k k^k)^d)$, which is constant for fixed k and d. □

Combining these results, it is possible to show that plan generation for $\mathbf{P}(k, d)$ has linear complexity in the number of actions.

Theorem 3.22. *For fixed values of* k *and* d, *the complexity of plan generation for* $P \in \mathbf{P}(k, d)$ *is* $O(|A|)$.

Proof. By Proposition 3.17 there exists an algorithm solving MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(k)$ with complexity $O(|A| + T^{f(k)})$. However, Proposition 3.21 states that for planning problems in $\mathbf{P}(k, d)$, T is bounded by a constant $T(k, d)$. In this case, solving MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(k)$ has complexity $O(|A|)$ since the second term is a constant.

In the reduction from $\mathbf{P}(k, d)$ to $\boldsymbol{\mathcal{I}}(k)$, the MAX-CHANGE problem associated with a variable v only includes actions that change the value of v. The complexity of solving all MAX-CHANGE problems is thus $O(\sum_{v \in V} |A_v|) = O(|A|)$, where A is the set of actions of the planning problem P and A_v is the subset of actions that change the value of v.

Finally, Proposition 3.15 states that the complexity of the reduction is $O(T|V|+|A|)$, which equals $O(|V|+|A|)$ since T is bounded by a constant for $\mathbf{P}(k, d)$. We can use the same argument as in the proof of Theorem 3.18 to obtain $|V| \leq (k+1)|A|$, implying an overall complexity of $O(|V|+|A|+|A|) = O(|A|)$. □

Finally, we study the complexity of the class $\mathbf{P}(2)$, i.e., in which each action a has at most 2 pre-conditions on variables not affected by a. Our motivation is to show that planning problems in $\mathbf{P}(2)$ can be solved much more efficiently than our general proof for $\mathbf{P}(k)$ suggests. In Section 5 we prove the following proposition regarding MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(2)$.

Proposition 3.23. *The complexity of solving a* MAX-CHANGE *problem* $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(2)$ *is* $O(T^4)$*, where* T *is an upper bound on the values* $n(u)$ *of the predecessors* $u \in U$*.*

We can now prove the following theorem regarding the complexity of plan generation for $\mathbf{P}(2)$.

Theorem 3.24. *The complexity of solving planning problems* $P \in \mathbf{P}(2)$ *is* $O(|V||A| + |V|T^4)$*.*

Proof. Analogous to the proof of Theorem 3.18, with the only difference that we have kept the notation T and $|V|$ instead of replacing it with $|A|$. □

As a result of Theorem 3.24, we can efficiently solve planning problems in $\mathbf{P}(2)$ with hundreds of variables, and often even larger since T is in practice lower than $|V|$. Current benchmark domains in planning rarely exceed this number of variables, although there might of course be many planning problems of interest that do.

4. $\mathcal{I}(k)$ is Polynomial-Time Solvable

The aim of this section is to prove Proposition 3.17, i.e., there is an algorithm that, given any MAX-CHANGE problem $\Pi = \langle v, U, n, A\rangle \in \mathcal{I}(k)$, computes a solution to Π in time $O(|A| + T^{f(k)})$, where $f(k)$ is a constant depending on k, and T is a bound on the number of times $n(u)$ that any variable $u \in U$ is allowed to change. Recall that $n(u)$ is given in unary notation in the description of the problem instance Π, so that the resulting algorithm indeed runs in polynomial time in terms of $|\Pi|$.

Previously, the best known way to solve this problem was a dynamic programming algorithm due to Brafman and Domshlak (2003) with complexity $O(|A|T^{|U|})$. Note that this is not a polynomial-time algorithm, since there is an exponential dependence on $|U|$, the number of variables of Π. Our work needs this dynamic programming algorithm, and since our definition of MAX-CHANGE problems is different from that of Brafman and Domshlak (2003), we describe the algorithm in Theorem 4.1 using our notation.

Our algorithm is based on the following key idea: every MAX-CHANGE problem Π in $\mathcal{I}(k)$ is *equivalent* to some MAX-CHANGE problem Π' with at most $f(k)$ variables, where $f(k)$ is a constant that only depends on k (and not on the size of Π). By *equivalent*, we mean that one can obtain solutions for Π by solving Π' (the precise meaning of *equivalent*, which is akin to the notion of *reduction* between computational problems, is introduced in Definition 4.13). This theoretical result is sufficient for our purposes: since Π' has a constant number of variables, we can use the dynamic programming algorithm of Brafman and Domshlak (2003) to produce a solution for Π' and, consequently, for Π, in polynomial time $O(T^{f(k)})$. (We have removed the factor $|A|$ from Theorem 4.1 since, if the number of variables is bounded, so is the number of actions that can be defined on them.)

Clearly, what remains to be shown is why every problem Π is equivalent to some problem Π' with at most $f(k)$ variables, and to describe an efficient polynomial-time procedure for finding Π' from Π. For convenience, in what follows we give a very rough description of the underlying reason why this is true. The readers who find this description too informal are invited to skip it and go directly to Subsection 4.1, where we describe the dynamic programming algorithm of Brafman and Domshlak (2003), or to Subsection 4.2, where we state and prove the precise theoretical results about MAX-CHANGE problems that our algorithm is based on.

Consider the two diagrams in Figure 2. Here, each diagram illustrates

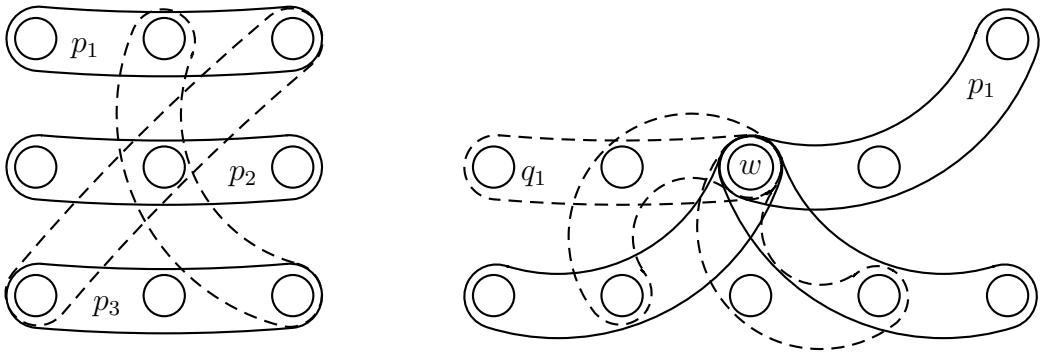

Figure 2: Two example MAX-CHANGE problems. Left: a MAX-CHANGE problem such that all actions in A^1 (dashed lines) are restricted to the 9 variables of actions p_1, p_2, p_3. Right: a MAX-CHANGE problem which can be simplified to just two actions (p_1, q_1) and a single variable w.

a MAX-CHANGE problem in $\boldsymbol{\mathcal{I}}(3)$, such that nodes correspond to variables, and curved boxes surrounding nodes correspond to actions that depend on these variables (for the moment, we disregard literals and consider variables only). Solid lines correspond to actions $p \in A^0$, and dashed lines correspond to actions $q \in A^1$. Note that in these diagrams each box of one type has, at least, one variable in common with each box of the other type. This is implied by the fourth item of Definition 3.11, namely, that for each $(p, q) \in A^0 \times A^1$ of a MAX-CHANGE problem, it holds that $|p \cap \overline{q}| \geq 1$.

Consider the diagram on the left, where actions in A^0 are spread on a set of 9 variables. Because of the way MAX-CHANGE problems are defined, every action in A^0 is a *constraint* for the actions in A^1, and vice versa: the more actions of one type, the harder to add new actions of the other type while satisfying all the constraints. For instance, consider actions p_1, p_2, and p_3: each dashed box (that is, actions in A^1) must intersect these three actions at some variable. Since the actions p_1, p_2, and p_3 do not share variables, 3-dependent actions in A^1 must be defined on this set of 9 variables. Clearly, this is good news for us: the MAX-CHANGE problem already has few actions of one type, possibly obviating the need to reduce it to a smaller problem.

Consider now the diagram on the right, and assume that $w \in p$ and $\neg w \in q$ for each pair of actions $(p, q) \in A^0 \times A^1$. Thus all intersection constraints are satisfied by the variable w, and actions are free to use their $k-1$ remaining pre-conditions in any way. Hence, neither A^0 nor A^1 might have bounded size, as was the case in the previous example. However, consider the actions

p_1 and q_1, whose pre-conditions do not share any variable other than w. We claim that we can solve the MAX-CHANGE problem using these two actions only. First, we satisfy the pre-conditions of p_1 and q_1 on variables other than w (this is always possible due to the fact that $n(u) > 0$ for each variable $u \in U$ of a MAX-CHANGE problem). Then we repeatedly apply these two actions, changing the value of w between successive applications. The maximum number of times we can do this is $n(w)+1$ (the pre-condition of p_1 on w is satisfied in the initial state). We claim that we can simply ignore the remaining actions, since their use does not improve the solution $(n(v), \pi)$. The reason is that $w \in p$ and $\neg w \in q$ for each $(p, q) \in A^0 \times A^1$: whatever choice of actions other than p_1 and q_1, we would still have to change the value of w between succesive applications of actions in A^0 and A^1. Hence, the original MAX-CHANGE problem is, in some sense, *equivalent* to a MAX-CHANGE problem with just two actions, p_1 and q_1, and one variable, w.

In our terminology, introduced below, we say that the set $G = L(\{w\}) = \{w, \neg w\}$ is a *cut* (a set of literals such that every action of the problem has at least one pre-condition among them, Definition 4.2), and that the set $R = \{p_1, q_1\}$ is a set of *representatives* for G (a set of actions, pairwise compatible outside G, that makes other actions irrelevant with respect to G, Definition 4.10). We later show that if a MAX-CHANGE problem Π has some cut G admitting a set of representatives R, then Π is equivalent to a smaller problem Π' defined on the variables of G and the actions in R (Proposition 4.14).

The previous diagrams represent simplified situations, and many details have been left out in this informal discussion. To name just a few: cuts (like G) may contain several variables instead of a single one; actions do not depend on variables, but on literals; and, finally, not all cuts admit sets of representatives. We handle these issues in Theorem 4.16, which roughly states that, although not all cuts admit sets of representatives, for any MAX-CHANGE problem Π on k-dependent actions there is always some cut of size not larger than $f(k)$ that admits one. The proof of this existence result is quite technical, and involves induction on two variables (Lemma 4.19). The precise definition of the constants $f(k)$ is given in Definition 4.17 by means of recurrence formulas, which have their origin in the previous inductive proof.

Finally, at the end of this section we show that the proof of Theorem 4.16 can be made constructive. That is, we do not just claim the existence of some small cut admitting a set of representatives, but we actually show how

to obtain one. This extension (Lemma 4.20) is a necessary ingredient to obtain the desired running time $O(|A| + T^{f(k)})$ of our algorithm in Theorem 4.21. (Without this extension, we could still obtain a polynomial-time algorithm for $\boldsymbol{\mathcal{I}}(k)$ and $\mathbf{P}(k)$, but the complexity would be worse.) Also note that Theorem 3.22 regarding planning problems in $\mathbf{P}(k, d)$ depends on the algorithm of Theorem 4.21.

4.1. Dynamic Programming Algorithm

We start by introducing the dynamic programming algorithm for solving MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(k)$, assuming k is fixed. As mentioned, this was previously the best known way for solving this kind of problem, and our new algorithm is based on it.

Theorem 4.1. *Let $\Pi = \langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(k)$ be a* MAX-CHANGE *problem. There is a dynamic programming algorithm that computes a solution to Π in time $O(|A|T^{|U|})$, where T is a bound on the values $n(u)$ of Π for every $u \in U$.*

Proof. Let $U = \{u_1, \ldots, u_m\}$. We compute an optimal solution $\langle n(v), \pi\rangle$ by dynamic programming on the number of changes of the m variables $u_1, \ldots, u_m$. More precisely, for each $x \in \{0, 1\}$ and each number $0 \leq \nu_i \leq n(u_i)$ for each variable $u_i \in U$, we define $Q(x, \nu_1, \ldots, \nu_m)$ as the maximum number of times that v can change, given that, for each $1 \leq i \leq n$, u_i has already changed ν_i times, and that v is currently true (if $x = 1$) or false (if $x = 0$). Clearly, the solution $n(v)$ is given by $Q(1, 0, \ldots, 0)$ since initially, v is true and none of the variables in U have changed.

We can compute $Q(x, \nu_1, \ldots, \nu_m)$ recursively by taking the maximum over the following values:

- $Q(x, \nu_1, \ldots, \nu_{i-1}, \nu_i + 1, \nu_{i+1}, \ldots, \nu_m)$ for each $i \in \{1, \ldots, m\}$ such that $\nu_i < n(u_i)$.
- $1 + Q(1 - x, \nu_1, \ldots, \nu_m)$ if there is some $a \in A^x$ such that for each $u_i \in \mathsf{pre}(a)$ it holds that ν_i is even, and for each $\neg u_i \in \mathsf{pre}(a)$ it holds that ν_i is odd.

The first type of value corresponds to changing the value of variable u_i, while the second type of value corresponds to the application of action a in the current state.

Recall that for MAX-CHANGE problems, there exists no pair of actions $(p, q) \in A^0 \times A^1$ such that $p \cap \overline{q} = \emptyset$. This ensures that $Q(x, \nu_1, \ldots, \nu_m)$ is

well-defined since we cannot repeatedly apply actions in A^0 and A^1 to obtain an arbitrarily large value. Moreover, the base case of the recursion is given by $Q(x, n(u_1), \ldots, n(u_m)) = 0$ for at least one of $x = 0$ and $x = 1$, since we cannot change the value of any $u_i \in U$ in this case, and since at most one of A^0 and A^1 can contain an action which is applicable in this state.

To extract a plan π maximizing the total number of changes, we just have to keep track of the way that each $Q(x, \nu_1, \ldots, \nu_m)$ was obtained. It is clear that the resulting table holds $2\prod_{i=1}^{m}(1 + n(u_i))$ values, and dynamic programming can be done in time $2(|A| + m)\prod_{i=1}^{m}(1 + n(u_i))$. Let T denote an upper bound on the values $n(u_i)$. As before, we assume that each variable appears in the pre-condition of at least one action, implying $m \leq k|A|$. Thus $m = O(|A|)$ and the worst-case complexity is $O(|A|T^m)$, as claimed. □

4.2. Existence of a Set of Representatives

As stated in the beginning of this section, its purpose is to prove a theoretical result about MAX-CHANGE problems, namely Theorem 4.16. Along the way, we introduce several definitions and intermediate results. To exemplify these definitions, we make use of the MAX-CHANGE problem Π defined in Example 3.14.

We start by defining the notion of a *cut* for a MAX-CHANGE problem. A cut is a set of literals G such that each action has a pre-condition on at least one of them. In addition, we require G to be *complete*. Recall that a set of literals G is complete when $l \in G$ implies $\neg l \in G$ for any literal l. Equivalently, G is complete if $G = L(V(G))$.

Definition 4.2. Let $\langle v, U, n, A\rangle$ be a MAX-CHANGE problem in $\boldsymbol{\mathcal{I}}(k)$. A set of literals G is a *cut* if G is complete and for each $p \in A^0$ and $q \in A^1$ it holds that $|p \cap G| \geq 1$ and $|q \cap G| \geq 1$. For convenience, we define the size $|G|$ of G to be the number of variables $|V(G)|$, although G does in fact contain twice as many literals.

Example 4.3. For the example MAX-CHANGE problem Π, the sets $\{x, \neg x\}$ and $\{y, \neg y, z, \neg z\}$ are cuts of size 1 and 2, respectively. The sets $\{y, \neg y\}$, $\{z\neg z, w, \neg w\}$, and $\{x, y, w\}$ are not cuts (the first does not intersect q_3; the second intersects neither q_1 nor q_2; the third is not complete). The smallest cut containing neither x nor y has size 4.

Lemma 4.4. *Every* MAX-CHANGE *problem* $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(k)$ *has some cut* G *of size* $|G| \leq 2k - 1$.

Proof. Consider any $(p, q) \in A^0 \times A^1$, and let $G = L(V(p \cup \overline{q}))$. Clearly $|G| \leq 2k - 1$, since $|p \cap \overline{q}| \geq 1$ by definition of MAX-CHANGE problems, and $|p| \leq k$ and $|q| \leq k$ by definition of $\boldsymbol{\mathcal{I}}(k)$. Moreover, for each $p' \in A^0$ it holds that $|p' \cap \overline{q}| \geq 1$, implying $|p' \cap G| \geq 1$, and 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. □

Example 4.5. Actions p_1 and q_3 of the example MAX-CHANGE problem Π define a cut $G = L(V(\{x, \neg x, y, z, \neg w\})) = L(\{x, y, z, w\})$, of size 4.

A complete set of literals G (not necessarily a cut) induces a partition of the set of actions $A = \langle A^0, A^1 \rangle$ according to the way in which they intersect G: two actions belong to the same part of the partition if and only if they have the same pre-conditions on G and they belong to the same set among A^0 and A^1.

Definition 4.6. Let $A = \langle A^0, A^1 \rangle$ be a set of actions and G a complete set of literals. The *partition* $A_G = \langle A^0_G, A^1_G \rangle$ *of* A *induced by* G is $A^x_G = \{B^x_1, \ldots, B^x_{m_x}\}$ for $x \in \{0, 1\}$, where m_x denotes the size of partition A^x_G, and A^x_G is such that $p, q \in B^x_i$ iff $p, q \in A^x$ and $p \cap G = q \cap G$.

We also define $G(B^x_i) \subseteq G$ to be the unique projection $p \cap G$ onto G of partial states $p \in B^x_i$ and, for $1 \leq j \leq k$, we define $A^x_G(j) = \{B^x_i \in A^x_G : |G(B^x_i)| = j\}$, that is, the subset of A^x_G such that the projection onto G has cardinality j.

Example 4.7. For the example MAX-CHANGE problem, the set $G = \{x, \neg x\}$ induces a partition $A_G = \langle \{B^0_1, B^0_2\}, \{B^1_1, B^1_2\} \rangle$ with $B^0_1 = \{p_1\}$, $B^0_2 = \{p_2\}$, $B^1_1 = \{q_1, q_3\}$ and $B^1_2 = \{q_2\}$. Actions q_1 and q_3 belong to the same part since $q_1 \cap G = q_3 \cap G = G(B^1_1) = \{x\}$. Note that $p_1 \cap G = \{x\}$ as well, but since $p_1 \in A^0$ and $q_1, q_3 \in A^1$, p_1 belongs to a different part.

The set $G = \{x, \neg x, y, \neg y\}$ induces a partition with 5 parts, one for each action, since $\{x, \neg y\} = q_1 \cap G \neq q_3 \cap G = \{x\}$.

The following is an upper bound on the number of parts $m_0 + m_1$ of a partition A_G.

Definition 4.8. We write $S(t, k)$ for twice the number of ways to choose subsets of k or less elements from a set of $2t$ elements. That is,

$$S(t, k) := 2 \sum_{i=0}^{k} \binom{2t}{i}.$$

Proposition 4.9. *Let $A = \langle A^0, A^1 \rangle$ be a set of k-dependent actions and G a complete set of literals with $|G| = t$. Let m_0 and m_1 be the number of parts of the partition $A_G = \langle A^0_G, A^1_G \rangle$ induced by G. Then $m_0 + m_1 \leq S(t, k)$.*

Proof. Clearly, if B^0_i and B^0_j are different parts of A^0_G, then they must differ on G, that is, $G(B^0_i) \neq G(B^0_j)$. Since G has $2t$ literals and actions are k-dependent, there are at most $\sum_{i=0}^{k} \binom{2t}{i}$ possible subsets of G to choose from, so this number is an upper bound of the number of parts m_0 of A^0_G. Since the same is true for A^1_G, we obtain $m_0 + m_1 \leq S(t, k)$. □

We now define the notion of a set of representatives R for a pair A, G. In short, it is a subset of actions $R \subseteq A$ that are pairwise compatible outside G, and such that each part B^x_i of the partition A_G has some *representative* $r \in R^x$.

Definition 4.10. Let $A = \langle A^0, A^1 \rangle$ be a set of actions, and let G be a complete set of literals. We say that $R = \langle R^0, R^1 \rangle$ is a set of *representatives* for A, G if the following holds:

- For each $x \in \{0, 1\}$, $R^x \subseteq A^x$.
- For each $x \in \{0, 1\}$ and $r, r' \in R^x$, $r \cap G \neq r' \cap G$.
- For each $r, r' \in R$, $(r - G) \cap (\overline{r' - G}) = \emptyset$.
- For each $x \in \{0, 1\}$ and $B^x_i \in A^x_G$, there exists $r \in R^x$ such that $r \cap G \subseteq G(B^x_i)$.

Example 4.11. The cut $G = \{x, \neg x\}$ induces a partition A_G (see Example 4.7) that admits no set of representatives. Indeed, the only possible representatives for parts $B^0_2 = \{p_2\}$ and $B^1_2 = \{q_2\}$ are p_2 and q_2 themselves, but these two actions do not satisfy the third item of the definition, since $p_2 - G = \{\neg y, w\}$ and $\overline{q_2 - G} = \{\neg y, \neg s\}$, and hence $(p_2 - G) \cap (\overline{q_2 - G}) = \{\neg y\} \neq \emptyset$.

On the other hand, the partition A_G induced by the cut $G = \{x, \neg x, y, \neg y\}$ does admit a set of representatives $R = \langle R_0 = \{p_1, p_2\}, R_1 = \{q_2, q_3\} \rangle$. Note that the action q_3 is the representative of both $B^1_1 = \{q_1\}$ and $B^1_3 = \{q_3\}$, since $q_3 \cap G = \{x\}$ is a subset of both $G(B^1_1) = \{x, \neg y\}$ and $G(B^1_3) = \{x\}$.

The following is a straightforward consequence of the previous definition: there cannot be more representatives than parts in the partition.

Proposition 4.12. *Let R be a set of representatives for a pair A, G where $A = \langle A^0, A^1 \rangle$ contains k-dependent actions and G is a complete set of literals. Then $|R| \leq S(|G|, k)$.*

Proof. A direct consequence of Proposition 4.9 since, by the second item of Definition 4.10, for every $x \in \{0, 1\}$ and every part B_i^x in the partition A_G^x there is at most one representative $r \in R^x$ with $r \cap G = G(B_i^x)$. □

This notion of a set of representatives is important for the following reason. Whenever a MAX-CHANGE problem $\Pi = \langle v, U, n, A \rangle \in \boldsymbol{\mathcal{I}}(k)$ has a complete set of literals G such that the pair A, G admits a set of representatives R, then we can rewrite Π into an *equivalent* MAX-CHANGE problem Π' using only the variables of G and the actions of R. Before proving this, we formalize what we mean by equivalent.

Definition 4.13. Let $\Pi = \langle v, U, n, A \rangle$ and $\Pi' = \langle v', U', n', A' \rangle$ be two MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(k)$. We say that Π *reduces to* Π' if there exists a function $\phi : A' \to A$ satisfying the following property: if $\langle N, a_1 \cdots a_N \rangle$ is a solution to Π', then $\langle N, \phi(a_1) \cdots \phi(a_N) \rangle$ is a solution to Π.

Note that if Π reduces to Π', we can produce a solution to Π by solving Π' and translating the solution back to Π by means of ϕ. Typically, the problem Π' we reduce Π to is simpler than Π.

Proposition 4.14. *Let $\Pi = \langle v, U, n, A = \langle A^0, A^1 \rangle \rangle$ be a* MAX-CHANGE *problem in $\boldsymbol{\mathcal{I}}(k)$, let G be a complete set of literals, and let $R = \langle R^0, R^1 \rangle$ be a set of representatives for A, G. Then Π can be reduced to $\Pi' = \langle v, V(G), n, R' \rangle$, where $R' = \langle R'^0, R'^1 \rangle$ and $R'^x = \{p \cap G : p \in R^x\}$ for $x \in \{0, 1\}$ is the set of representatives in R^x projected onto G.*

Example 4.15. Consider again the example MAX-CHANGE problem Π. Since the set $G = \{x, \neg x, y, \neg y\}$ admits a set of representatives (see Example 4.11), the example problem Π reduces to

$$\Pi' = \big\langle v;\ U' = \{x, y\};\ n; R' = \langle R'^0 = \big\{p'_1 = \{x, y\}, p'_2 = \{\neg x, \neg y\}\big\}, \\ R'^1 = \big\{q'_2 = \{\neg x, y\}, q'_3 = \{x\}\big\}\rangle\big\rangle.$$

Proof of Proposition 4.14. First, we must show that Π' is a MAX-CHANGE problem in the sense of Definition 3.11. That is, we must show that for each $(r, r') \in R'^0 \times R'^1$, it holds that $|r \cap \overline{r'}| \geq 1$. However, $r = p \cap G$ and $r' = q \cap G$

for some $p \in R^0 \subseteq A^0$ and $q \in R^1 \subseteq A^1$; $|(p - G) \cap \overline{(q - G)}| = 0$ since R is a set of representatives for the pair A, G; and $|p \cap \overline{q}| \geq 1$ since Π is a MAX-CHANGE problem. Since p, q do not intersect outside of G they must intersect inside, so that $|r \cap \overline{r'}| \geq 1$, as required.

To prove that Π reduces to Π', we must define a function $\phi : R' \to A$ such that any solution $\langle N, r'_1 \cdots r'_N \rangle$ to Π' can be translated to a solution $\langle N, \phi(r'_1) \cdots \phi(r'_N) \rangle$ to Π. Recall that $\langle N, \phi(r'_1) \cdots \phi(r'_N) \rangle$ is a solution to Π if and only if

- $\phi(r'_1) \cdots \phi(r'_N)$ is a valid plan for Π, and
- N is maximal.

To achieve this double goal we show that *every* valid plan in Π' translates to a valid plan in Π, but also the reciprocal, i.e., every valid plan in Π translates to a valid plan in Π', by means of functions $\phi : R' \to A$ and $\pi : A \to R'$. A direct consequence of this two-way translation is that both planning problems Π and Π' must admit the same maximal plan length N, since maximal valid plans from one problem translate to maximal valid plans of the other. (In fact, this two-way translation implies that, in our case, not only Π reduces to Π', but Π' also reduces to Π. Note that this is not true in general, that is, if Π reduces to some problem Π'', it may be the case that Π'' does not reduce to Π.)

The function $\phi : R' \to A$ is defined as follows. For $x \in \{0, 1\}$ and $r' \in R'^x$, we set $\phi(r') = r$ where $r \in R^x \subseteq A^x$ is such that $r' = r \cap G$. There is only one such r due to the second item of Definition 4.10.

Let $r'_1 \cdots r'_k$ be a valid plan for Π'. Consider the actions $\phi(r'_1), \ldots, \phi(r'_k) \in R \subseteq A$. For every $i \in \{1, \ldots, k\}$, it holds that $r'_i = \phi(r'_i) \cap G$; also, since R is a set of representatives, the partial states $\{\phi(r'_1), \ldots, \phi(r'_k)\}$ are pairwise compatible outside G (third item of Definition 4.10). In particular, every $u \in U - V(G)$ does not need to change more than once to satisfy the pre-conditions of the plan $\phi(r'_1) \cdots \phi(r'_k)$ in Π, and this is always possible since $0 < n(u)$, by definition of MAX-CHANGE problems. Hence the plan $\phi(r'_1) \cdots \phi(r'_k)$ is valid in Π.

To show the reciprocal, we define the function $\pi : A \to R'$ as follows. For $x \in \{0, 1\}$ and $p \in A^x$, we set $\pi(p) = r \cap G$ where $r \in R^x$ is such that $r \cap G \subseteq p \cap G$. Note that there is always such an r by the fourth item of Definition 4.10, and that $r \cap G$ is an element belonging to R', by definition

of R'. Also note that there may be several possible choices of r, giving rise to different choices for π (any one will do).

Let $p_1 \cdots p_k$ be a valid plan of Π. Note that Π' is defined only on the set of variables $W = V(G)$, and that for every $i \in \{1, \ldots, k\}$ it holds that $\pi(p_i) \subseteq p_i \cap G$. In particular, variables $u \in W$ need to change their value at most as many times as before. Hence $\pi(p_1) \cdots \pi(p_k)$ is valid in Π'.

□

Having proven Proposition 4.14, it remains to show that there exist small cuts admitting sets of representatives. Indeed, we prove the following theorem.

Theorem 4.16. *Fix $k > 0$. There is a constant $f(k)$ such that any* MAX-CHANGE *problem $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(k)$ has a cut G of size $|G| \leq f(k)$ such that the pair A, G admits a set of representatives R.*

We first provide an *existential* proof of this Theorem. In Lemma 4.20, we show how this existential proof can be made constructive, obtaining an algorithm that finds the cut G in time $O(|A|)$. We have decided to break the proof in two parts instead of providing a single algorithmic description because the existential result of Theorem 4.16 is interesting by itself, and its proof is already quite technical.

We briefly describe the ideas underlying the existential proof of Theorem 4.16. Let $k > 0$ be some constant. Start with a cut G of constant size (say, the one from Lemma 4.4). This cut partitions the actions into a constant number of parts. For every such part B of the partition, look for a set of variables $W_B \subseteq U - V(G)$ of minimal size such that every action in B contains at least one literal in $L(W_B)$. If $|W_B|$ is large (exceeding a bound that depends on k), we show that G already admits a representative for actions in B. On the contrary, if W_B is small, we form a new cut $G' = G \cup L(W_B)$, which is still of constant size. Observe that, by definition of W_B, each action in B has strictly more literals in G' than in G. We repeat this process for each part B of the original partition of G, adding a new set of literals to the cut G' if W_B is small, or finding a representative for B if W_B is large. In case no representative is found for some B, we repeat the whole process for the new cut G', which by definition intersects all actions at strictly more literals than G. As long as no set of representatives is found, we keep growing the cut until it contains all literals of all actions, while still having constant size.

The maximum size the cut can grow to is precisely the value $f(k)$, which we now proceed to formally define.

Definition 4.17. Fix $k > 0$. We define $f(k) := G_k(2k-1, k-1)$, where the function $G_k(t, j)$ is given by

$$\begin{aligned}
G_k(t, 0) &:= t, \\
G_k(t, j) &:= G_k(t + \sum_{\ell=1}^{S(t,k)} b_k(t, j, \ell), j-1), \\
b_k(t, j, 1) &:= j(S(t, k) - 1), \\
b_k(t, j, i) &:= j(S(t, k) - i) + jS(G_k(t + \sum_{\ell=1}^{i-1} b_k(t, j, \ell), j-1), k),
\end{aligned}$$

and $S(t, k)$ is as in Definition 4.8.

Note that these numbers are well-defined, since they can be computed in the order

$$G_k(t, j) \succ b_k(t, j, S(t, j)) \succ \cdots \succ b_k(t, j, 1) \succ G_k(t', j-1) \succ \cdots$$

The proof of Theorem 4.16 is by induction. We proceed to state and prove the inductive statement that the theorem requires.

Definition 4.18. A cut G is a j-cut if, for each action $p \in A$, $|p \cap G| \geq j$, i.e., each action has at least j pre-conditions in G.

Lemma 4.19. *Let A be a set of k-dependent actions, and let G be a j-cut. Then there is some cut $G' \supseteq G$ of size $|G'| \leq G_k(|G|, k-j)$ such that the pair A, G' admits a set of representatives R.*

Proof. The base case is given by $j = k$, i.e., G is a k-cut. Then all actions $p \in A$ satisfy $p \subseteq G$, so they are trivially pairwise compatible outside G. Hence $R = A$ is a set of representatives for the pair A, G. Thus the lemma holds for $G' = G$, since $G \supseteq G$, $|G| \leq G_k(|G|, k-j) = G_k(|G|, 0) = |G|$ by Definition 4.17, and A, G admits the set of representatives $R = A$.

The recursive case is given by $j < k$. By induction, we assume that the lemma holds for any j'-cut such that $j < j'$. Let $A_G = \langle A_G^0, A_G^1 \rangle$ be the partition of A induced by G, where $A_G^x = \{B_1^x, \ldots, B_{m_x}^x\}$ for $x \in \{0, 1\}$.

From Proposition 4.9 we know that $m = m_0 + m_1 \leq S(|G|, k)$. For each part $B = B_i^x$, define $B' = \{p - G : p \in B\}$. Let $H(B)$ be a minimal cut of B', and let $h(B) = |H(B)|$ be its size. If B' has no cut (this only happens when B' contains an empty partial state), set $h(B) = \infty$. Finally, let $B_1, \ldots, B_m$ be a reordering of the sets $B_1^0, \ldots, B_{m_0}^0, B_1^1, \ldots, B_{m_1}^1$ in increasing order with respect to $h(B_i)$, i.e., such that $h(B_1) \leq h(B_2) \leq \cdots \leq h(B_m)$.

Consider the set B_1. If $h(B_1) > b_k(|G|, k-j, 1)$, we prove by induction on the sets B_i that we can construct a set of representatives R for the pair A, G. The base case is given by $i = 1$, in which case we are free to select any representative for B_1. For $1 < i \leq m$, assume R' is a partial choice of representatives for the sets $B_1, \ldots, B_{i-1}$. Since each action in R' has at most $k - j$ pre-conditions outside G, the actions in R' are defined on at most $(k-j)(i-1)$ variables outside G. For B_i we have $h(B_i) \geq h(B_1) > b_k(|G|, k-j, 1) = (k-j)(S(|G|, k) - 1) \geq (k-j)(m-1) \geq (k-j)(i-1)$, i.e., B_i' has no minimal cut of size less than or equal to $(k-j)(i-1)$. Thus there has to exist an action $p \in B_i$ that does not intersect any action in R' outside G, and we can always choose this action to extend R' for B_i.

Next, consider the case where, for some $1 < i \leq m$, it holds that $h(B_{i'}) \leq b_k(|G|, k-j, i')$ for each $i' < i$, but $h(B_i) > b_k(|G|, k-j, i)$. Let $B = \langle B^0, B^1 \rangle$ be the set of actions of $B_1, \ldots, B_{i-1}$, and let $H = G \cup H(B_1) \cup \ldots \cup H(B_{i-1})$ (the existence of these cuts is implied by $h(B_{i'}) \leq b_k(|G|, k-j, i') < \infty$). Clearly, H is a $(j+1)$-cut for B, since G is a j-cut for A (and hence, for B) and $H(B_1), \ldots, H(B_{i-1})$ are cuts of $B_1', \ldots, B_{i-1}'$ not intersecting G. Also, note that

$$|H| \leq |G| + \sum_{\ell=1}^{i-1} b_k(|G|, k-j, \ell).$$

By inductive application of the lemma on the set B of actions and the $(j+1)$-cut H, there exists a set $H' \supseteq H$ with $|H'| \leq G_k(|H|, k-(j+1))$ such that the pair B, H' admits a set of representatives R'.

We now claim that the pair A, H' admits a set of representants R. The argument is similar to the case for B_1. Start with the set of representatives R' for B, H'. The cut H' partitions B into at most $S(|H'|, k)$ parts, so R' is

defined on at most $(k-(j+1))S(|H'|,k)$ variables outside H'. For $i \leq i' \leq m$,

$$\begin{aligned} h(B_{i'}) &\geq h(B_i) > b_k(|G|, k-j, i) = (k-j)(S(|G|,k) - i) + \\ &+ (k-j)S(G_k(|G| + \sum_{\ell=1}^{i-1} b_k(|G|, k-j, \ell), k-j-1), k) \geq \\ &\geq (k-j)(m-i) + (k-j)S(G_k(|H|, k-(j+1)), k) \geq \\ &\geq (k-j)(i'-i) + (k-j)S(|H'|,k) > \\ &> (k-j)(i'-i) + (k-(j+1))S(|H'|,k). \end{aligned}$$

Thus $B'_{i'}$ has no minimal cut of size less than or equal to $(k-j)(i'-i) + (k-(j+1))S(|H'|,k)$, i.e., there is always some action $p \in B_{i'}$ that does not intersect any action in R' outside H', even after extending R' with actions from the sets $B_i, \ldots, B_{i'-1}$. Hence it is always possible to extend R' with actions from the sets $B_i, \ldots, B_m$ to obtain a set of representatives for A, H'.

Finally, the only remaining possibility is that $h(B_i) \leq b_k(|G|, k-j, i)$ for each $1 \leq i \leq m$. In this case, $H = G \cup H(B_1) \cup \ldots \cup H(B_m)$ is a $(j+1)$-cut of A. By induction there exists a cut $H' \supseteq H \supseteq G$ of size $|H'| \leq G_k(|H|, k-(j+1)) \leq G_k(|G| + \sum_{\ell=1}^{S(|G|,k)} b_k(|G|, k-j, \ell), (k-j)-1) = G_k(|G|, k-j)$ such that the pair A, H' admits a set of representatives R. □

The proof of Theorem 4.16 now follows immediately from Lemma 4.19.

Proof of Theorem 4.16. Lemma 4.4 states that every MAX-CHANGE problem $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(k)$ has a cut G of size at most $2k-1$. Since a cut is always a 1-cut, applying Lemma 4.19 to G implies that there exists a cut G' of size at most $f(k) = G_k(2k-1, k-1)$ such that A, G' admits a set of representatives R. □

4.3. Constructive Proof, Algorithm and Complexity

As a consequence of Theorem 4.16, for fixed $k > 0$ there is a straightforward algorithm solving instances $\Pi = \langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(k)$: try all cuts of size at most $f(k)$ and all combinations of representatives for the resulting partition. By Theorem 4.16, this is guaranteed to find a cut that admits a set of representatives. Then reduce Π to Π' and apply the dynamic programming algorithm to Π'. The complexity of the resulting algorithm is $O(|A|^{S(f(k),k)} + T^{f(k)})$, where the term $|A|^{S(f(k),k)}$ arises from considering all possible sets of representatives R for a cut of size $f(k)$, and the term $T^{f(k)}$

is due to the dynamic programming algorithm. This algorithm runs in polynomial time, but is not sufficient to prove Theorem 3.22.

Our algorithm, which has complexity $O(|A| + T^{f(k)})$, is an improvement of the previous idea: instead of trying all possible pairs G', R, carefully (and efficiently) repeat the process of the proof of Lemma 4.19 to obtain a pair G', R.

Lemma 4.20. *There is an algorithm that finds the sets G' and R of Lemma 4.19 in time $O(g(k)|A|) = O(|A|)$, for some function $g(k)$.*

Proof. The proof of this lemma is an extension of the proof of Lemma 4.19. For this reason, we assume that the reader is familiar with the notation introduced in that proof. Recall that there we had a reordering $B_1, \ldots, B_m$ of the sets B_i^x for $x \in \{0, 1\}$ and $i \in \{1, \ldots, m_x\}$ of the partition A_G, according to the size of the minimal cuts for B_i^x. However, computing minimal cuts is computationally expensive. We describe a process obtaining an alternative reordering $\tilde{B}_1, \ldots, \tilde{B}_m$, such that for each $\tilde{B}_i$ we either obtain a cut $H(\tilde{B}_i)$ of size at most $b_k(|G|, k-j, i)$, or find a set of representatives for the remaining sets $\tilde{B}_i, \ldots, \tilde{B}_m$.

First, fix any ordering $\overline{B}_1, \ldots, \overline{B}_m$ of the sets B_i^x. Then attempt to construct a set of representatives R for the pair A, G by selecting one representative $a_1, \ldots, a_m$ from each of the sets $\overline{B}_1, \ldots, \overline{B}_m$, in this order, and without backtracking. That is, we only require that, when selecting a new representative a_i for $\overline{B}_i$, the action a_i must be pairwise compatible outside G with the representatives $a_1, \ldots, a_{i-1}$ already selected. Clearly, this process takes linear time $O(|A|)$, since for each action $a_i \in \overline{B}_i$ we have to check whether it intersects at most m other actions, each of which has at most k pre-conditions, and m and k are both constant.

As a result of this process, there are two possible outcomes: either we succeed in finding a set of representatives R for the pair A, G, or there is some set $\overline{B}_i$ such that no representative a_i can be chosen from $\overline{B}_i$ without intersecting the other representatives $a_1, \ldots, a_{i-1}$. In the first case, the algorithm succeeds in finding a set of representatives for A, G. In the second case, it holds that $H = L(V(a_1) \cup \ldots \cup V(a_{i-1})) - G$ is a cut of $\overline{B}_i$ of size smaller than $(i-1)(k-j) \leq b_k(|G|, k-j, 1)$. We define $\tilde{B}_1 := \overline{B}_i$ and $H(\tilde{B}_1) = H$.

Now assume that we have already defined a reordering $\tilde{B}_1, \ldots, \tilde{B}_{i-1}$ in such a way that, for each $i' < i$, the set $\tilde{B}'_{i'}$ has some cut $H(\tilde{B}_{i'})$ of size not larger than $b_k(|G|, k-j, i')$. Define $\tilde{B} = \tilde{B}_1 \cup \cdots \cup \tilde{B}_{i-1}$, and consider the $(j+1)$-cut $\tilde{C} = G \cup H(\tilde{B}_1) \cup \cdots \cup H(\tilde{B}_{i-1})$. We recursively apply our algorithm

to obtain a set of representatives $\tilde{R}$ for the pair $\tilde{B}, \tilde{C}$. Then we extend this set of representatives with actions from the sets $\overline{B}_1, \ldots, \overline{B}_m$ not appearing in the reordering $\tilde{B}_1, \ldots, \tilde{B}_{i-1}$, using the same backtrack-free procedure as before. We either succeed in obtaining a set of representatives for the pair $A, \tilde{C}$, or we find some set $\overline{B}_t$ for which the set of representatives $\tilde{R}$ cannot be extended, i.e., a set $\overline{B}_t$ having some cut H not larger than $b_k(|G|, k-j, i)$. In this case, we define $\tilde{B}_i := \overline{B}_t$ and $H(\tilde{B}_i) = H$.

To summarize, our process either finds some cut $\tilde{C}$ and a set of representatives $\tilde{R}$ for the pair $A, \tilde{C}$, or it finds a reordering $\tilde{B}_1, \ldots, \tilde{B}_m$ of the partition A_G such that the set $\tilde{B}_i$ has a cut $H(\tilde{B}_i)$ of size bounded by $b_k(|G|, k-j, i)$. In the latter case, we obtain a cut G' and a set of representatives R for the pair A, G' by recursively applying our algorithm to the $(j+1)$-cut $\tilde{C} = G \cup H(\tilde{B}_1) \cup \cdots \cup H(\tilde{B}_m)$.

Finally, we show that the algorithm takes linear time $O(F(|G|, j, k)|A|)$, for some function F. Clearly, the process described to obtain each set $\tilde{B}_i$ takes linear time $O(|A|)$, the time required to extend the set of representatives in a backtrack-free manner. Since there are $m \leq S(|G|, k)$ such sets, the total time of this step is $O(S(|G|, k)|A|)$. Now consider the recursive calls. There are at most m of them, each taking as input a $(j+1)$-cut of size at most $b_k(|G|, k-j, i)$ and a set of actions $\tilde{B} \subseteq A$. By induction, each takes time $O(F(b_k(|G|, k-j, i), j+1, k)|\tilde{B}|)$. Note that the sum of these constants for each $i \in \{1, \ldots, m\}$ is some value depending only on $|G|$, k, and j. In particular, it does not depend on the size $|A|$ of the set of actions. Hence, the algorithm takes time $O(F(|G|, j, k)|A|)$ for some function F, as desired. □

Theorem 4.21. *There is an algorithm solving* Max-Change *problems* $\Pi = \langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(k)$ *in time* $O(|A| + T^{f(k)})$.

Proof. By Lemma 4.20, we can find the cut G' and the set of representatives R of Lemma 4.19 in time $O(|A|)$. Note that G' has at most $f(k)$ variables and R has at most $S(f(k), k)$ actions. Then we reduce the problem Π to a Max-Change problem $\Pi' = \langle v, V(G), n, R'\rangle$ according to Proposition 4.14 (easily done in time $k|R|$). The resulting Max-Change problem has at most $f(k)$ variables and $S(f(k), k)$ actions, that is, constants that do not depend on the size $|\Pi|$ of the input Max-Change problem. However, we note that the values $n(u)$ in Π' are not bounded. Let T be the maximum such value $n(u)$ for all $u \in V(G)$. By using the dynamic programming algorithm of Theorem 4.1, we can solve Π' in time proportional to $S(f(k), k)T^{f(k)}$, that is, $O(T^{f(k)})$. We then transform this solution $\langle N, r'_1 \cdots r'_N\rangle$ into a solution

to Π by applying the transformation function ϕ obtained when reduced Π to Π'. Note that, since the number of actions $|R'|$ in Π' is bounded by the constant $S(f(k), k)$, we can simply implement ϕ as a constant-time look-up table, and apply it to the $N \in O(T^{f(k)})$ actions of the plan. In summary, we obtain an algorithm that runs in time $O(|A| + T^{f(k)})$. □

5. $\boldsymbol{\mathcal{I}(2)}$

Although in the previous section we showed that MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(k)$ can be solved in polynomial time, the constant bounds on the size of the cuts used in the proof are very large, even for relatively small values of k (in Appendix B we provide lower and upper bounds on the value of $f(2)$). In this section, we prove that MAX-CHANGE problems in $\boldsymbol{\mathcal{I}}(2)$ can be reduced to equivalent problems on at most 4 variables and 6 actions. To do this, we first prove a lemma that restricts the size of a cut and a set of representatives for each MAX-CHANGE problem in $\boldsymbol{\mathcal{I}}(2)$.

Lemma 5.1. *Each* MAX-CHANGE *problem* $\langle v, U, n, A\rangle \in \boldsymbol{\mathcal{I}}(2)$ *has a cut* G *and a set of representatives* R *for* A, G *that satisfy one of the following four conditions:*

1. $|G| \leq 2$, $|A_G| \leq 6$, $|A^0_G(1)| + |A^1_G(1)| = 2$, *and* $|R| \leq 3$.
2. $|G| \leq 3$, $|A_G| \leq 8$, $|A^0_G(1)| + |A^1_G(1)| = 1$, *and* $|R| \leq 4$.
3. $|G| \leq 3$, $|A_G| \leq 7$, $|A^0_G(1)| + |A^1_G(1)| = 0$, *and* $|R| \leq 7$.
4. $|G| \leq 4$, $|A_G| \leq 6$, $|A^0_G(1)| + |A^1_G(1)| = 0$, *and* $|R| \leq 6$.

Proof. Recall that, by definition of MAX-CHANGE problems, $|p \cap \overline{q}| \geq 1$ for each $(p, q) \in A^0 \times A^1$. We use this property to classify the actions in A^0 by cases. By symmetry, the same cases hold for A^1.

I There exists a literal l_1 such that $l_1 \in p$ for each $p \in A^0$.
   (a) For each $q \in A^1$, $\bar{l}_1 \in q$.
   (b) There exists $q \in A^1$ such that $\bar{l}_1 \notin q$, implying $|A^0| \leq 2$ since each $p \in A^0$ has to intersect l_1 and $\overline{q}$.

II There exist p_1 and p_2 in A^0 such that $p_1 \cap p_2 = \emptyset$. Each $q \in A^1$ has to intersect $\overline{p}_1$ and $\overline{p}_2$, implying $|A^1| \leq 4$.
   (a) There exists $p_3 \in A^0$ such that $|p_1 \cap p_3| = 1$ and $p_2 \cap p_3 = \emptyset$. Then each $q \in A^1$ has to intersect $\overline{p}_1 \cap \overline{p}_3$. If p_3 intersects p_2 but not p_1, we can just swap the labels of p_1 and p_2.

(b) There are zero or more actions $p_3 \in A^0$ such that $|p_1 \cap p_3| = 1$ and $|p_2 \cap p_3| = 1$. Each such action excludes a potential action from A^1, since each $q \in A^1$ has to intersect $\overline{p}_3$.

III Remaining cases.

In Case I(a), $G = L(\{l_1\})$ is a cut and partitions A into $\langle\{A^0\}, \{A^1\}\rangle$, since $p \cap G = \overline{q} \cap G = \{l_1\}$ for each $p \in A^0$ and $q \in A^1$. If there exists $p, q \in A^0 \times A^1$ such that $p \cap \overline{q} = \{l_1\}$, $R = \langle\{p\}, \{q\}\rangle$ is a set of representatives for A, G. Recall that $A^0_G(1)$ is the subset of $A^0_G = \{A^0\}$ such that the projection onto G has cardinality 1. Thus $A^0_G(1) = A^0_G$ and $|A^0_G(1)| = 1$. Likewise, $|A^1_G(1)| = 1$. Then condition 1 of the lemma holds since $|G| = 1$, $|A_G| = 2$, $|A^0_G(1)| + |A^1_G(1)| = 2$, and $|R| = 2$.

If no such p, q exists, the only possibility is that A^0 and A^1 each contains a single action, so $|A| = 2$. Call the two partial states p' and q'. p' and $\overline{q}'$ have to intersect outside G, implying $V(p') = V(q')$. Thus $|U| = 2$. Trivially, $G' = L(U)$ is a cut with $|G'| = |U|$ that partitions A into one subset per action in A, each of which is contained in G'. Moreover, $R' = A$ is a set of representatives for A, G'. Thus condition 4 of the lemma holds since $|G'| = 2$, $|A_{G'}| = |A| = 2$, $|A^0_{G'}(1)| + |A^1_{G'}(1)| = 0$, and $|R'| = |A| = 2$.

In Case I(b), we first consider the case that $A^0 = \{p\}$ contains a single action $p = \{l_1, l_2\}$. Then $G = L(\{l_1, l_2\})$ is a cut and partitions A^1 into at most 5 subsets:

- $B^1_1 = \{q' \in A^1 : q' \cap G = \{\bar{l}_1\}\}$.
- $B^1_2 = \{q' \in A^1 : q' \cap G = \{\bar{l}_2\}\}$.
- $\{\{\bar{l}_1, \bar{l}_2\}\}$, $\{\{l_1, \bar{l}_2\}\}$, and $\{\{\bar{l}_1, l_2\}\}$.

Since we are not in case I(a), B^1_1 and B^1_2 are non-empty, so $|A^1_G(1)| = |\{B^1_1, B^1_2\}| = 2$, while $|A^0_G(1)| = 0$.

If there exists $q_1, q_2 \in B^1_1 \times B^1_2$ such that $q_1 \cap q_2 = \emptyset$, $R = \langle\{p\}, \{q_1, q_2\}\rangle$ is a set of representatives for A, G, since either $\{\bar{l}_1\}$ or $\{\bar{l}_2\}$ is a subset of each other action of A^1. Then condition 1 of the lemma holds since $|G| = 2$, $|A_G| \leq 6$, $|A^0_G(1)| + |A^1_G(1)| = 2$, and $|R| = 3$. If no such q_1, q_2 exists, the only possibility is $|B^1_1| = |B^1_2| = 1$, implying $|U| = 3$ and $|A| \leq 6$. Then condition 4 of the lemma holds for $G' = L(U)$ and $R' = A$, since $|G'| = 3$, $|A_{G'}| = |A| \leq 6$, $|A^0_{G'}(1)| + |A^1_{G'}(1)| = 0$, and $|R'| = |A| \leq 6$.

We next prove Case I(b) for $A^0 = \{p_1, p_2\}$. Let $q = \{l_2, l_3\}$, so that $p_1 = \{l_1, \bar{l}_2\}$ and $p_2 = \{l_1, \bar{l}_3\}$. In this case, $G = L(\{l_1, l_2, l_3\})$ is a cut and partitions A into at most 8 subsets:

- $\{p_1\}$, $\{p_2\}$, and $\{q\}$.
- $\{q_1\}$, $\{q_2\}$, $\{q_3\}$, and $\{q_4\}$, where $q_1 = \{\bar{l}_1, l_2\}$, $q_2 = \{\bar{l}_1, \bar{l}_2\}$, $q_3 = \{\bar{l}_1, l_3\}$, and $q_4 = \{\bar{l}_1, \bar{l}_3\}$.
- $B_1^1 = \{q' \in A^1 : q' \cap G = \{\bar{l}_1\}\}$.

If B_1^1 is non-empty, $R = \langle \{p_1, p_2\}, \{q, r\} \rangle$ is a set of representatives for A, G for any $r \in B_1^1$, since $r \cap G = \{\bar{l}_1\}$ is a subset of each q_i, $i \in \{1, 2, 3, 4\}$. Then condition 2 of the lemma holds since $|G| = 3$, $|A_G| \leq 8$, $|A_G^0(1)| + |A_G^1(1)| = |\{B_1^1\}| = 1$, and $|R| = 4$. If B_1^1 is empty, $|U| = 3$ and $|A| \leq 7$, so condition 3 of the lemma holds for $G' = L(U)$ and $R' = A$, since $|G'| = 3$, $|A_{G'}| = |A| \leq 7$, $|A_{G'}^0(1)| + |A_{G'}^1(1)| = 0$, and $|R'| = |A| \leq 7$.

Case II(a) corresponds to Case I(b) for A^1, by letting l_1 be the lone literal in $\overline{p}_1 \cap \overline{p}_3$, since each $q \in A^1$ has to contain this literal and since $\bar{l}_1 \notin p_2$. Case II(b) implies that $|U| = 4$ and $|A| \leq 6$, since each additional action in A^0 excludes a potential action from A^1. Thus condition 4 of the lemma holds for $G' = L(U)$ and $R' = A$, since $|G'| = 4$, $|A_{G'}| = |A| \leq 6$, $|A_{G'}^0(1)| + |A_{G'}^1(1)| = 0$, and $|R'| = |A| \leq 6$.

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 $\{l_1, l_2\}, \{l_1, l_3\}$ be two partial states in A^0. Since Case I does not hold, there exists $p \in A^0$ such that $l_1 \notin p$. Since Case II does not hold, p has to share a literal with each other action in A^0. The only possibility is $p = \{l_2, l_3\}$. A similar argument for A^1 suffices to show that A^1 contains $\{\bar{l}_1, \bar{l}_2\}$, $\{\bar{l}_1, \bar{l}_3\}$, and $\{\bar{l}_2, \bar{l}_3\}$. Thus $|U| = 3$ and $|A| = 6$, so condition 4 of the lemma holds for $G' = L(U)$ and $R' = A$, since $|G'| = 3$, $|A_{G'}| = |A| = 6$, $|A_{G'}^0(1)| + |A_{G'}^1(1)| = 0$, and $|R'| = |A| = 6$. □

Theorem 5.2. *Each* Max-Change *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. Theorem 5.2 follows immediately from Proposition 4.14 for conditions 1, 2, and 4 of Lemma 5.1, as well as condition 3 in case $|A| < 7$ since $|R| \leq |A|$. It remains to prove the theorem for Case I(b) with $A^0 = \{p_1, p_2\}$

and $A^1 = \{q, q_1, q_2, q_3, q_4\}$, i.e., B_1^1 is empty and $|A| = 7$. We show that in this case, an equivalent inverted fork problem is given by $\langle v, U, n, A' \rangle$, with $A' = \langle \{p_1, p_2\}, \{q, q_2, q_4\} \rangle$.

Assume that π is a sequence for changing the value of v a maximum number of times in the original MAX-CHANGE problem. We consider four subsequences of π of length 2: $\langle p_1, q_1 \rangle$, $\langle p_1, q_3 \rangle$, $\langle p_2, q_1 \rangle$, and $\langle p_2, q_3 \rangle$. Either subsequence requires changing l_1 to $\bar{l}_1$. Since $\bar{l}_2$ holds after p_1, q_2 is immediately applicable after changing l_1 to $\bar{l}_1$. Thus we can replace $\langle p_1, q_1 \rangle$ and $\langle p_1, q_3 \rangle$ by $\langle p_1, q_2 \rangle$, pushing any change of $\bar{l}_2$ to after q_2. The same is true for p_2 and q_4, so we can replace instances of $\langle p_2, q_1 \rangle$ and $\langle p_2, q_3 \rangle$ with $\langle p_2, q_4 \rangle$.

By symmetry, the same reasoning holds for the subsequences $\langle q_1, p_1 \rangle$, $\langle q_3, p_1 \rangle$, $\langle q_1, p_2 \rangle$, and $\langle q_3, p_2 \rangle$. Since we only solve MAX-CHANGE problems when we are not in an easy case, $N(v) > 1$, so any instance of q_1 or q_3 will always be preceded or succeeded by p_1 or p_2. We can thus construct a plan for changing the value of v a maximum number of times that does not include q_1 nor q_3. □

Proposition 3.23 now follows immediately from Theorem 5.2 and the dynamic programming algorithm in Theorem 4.1.

6. Conclusion

In this paper we have showed that the class $\mathbf{P}(k)$ of planning problems with binary variables, polytree causal graphs, and k-dependent actions is polynomial-time solvable, for any fixed value of k. We have also showed that $\mathbf{P}(k, d)$, the class of planning problems in $\mathbf{P}(k)$ whose causal graphs also have depth bounded by d, is linear-time solvable for fixed k and d. This latter result corresponds to the notion of fixed-parameter tractability with respect to the parameters k and d. The same does not hold for $\mathbf{P}(k)$ since the exponent in the polynomial term depends on the parameter k.

Our results are mainly of theoretical interest, since the constants involved in the analysis grow very quickly, making the corresponding algorithms impractical even for small values of k. However, the upper bounds we have imposed on these constants are likely not very tight at all. Our analysis of the case $\mathbf{P}(2)$ shows that there exist tighter bounds, improving the efficiency of the corresponding algorithms. We attempted to perform a similar analysis for $k = 3$, but the number of cases is already so large that such an analysis is almost impossible unless another proof technique is used.

As we previously showed, bounding the depth of the causal graph by a constant implies that there exist constant-size covers for each variable of the problem. In turn, this means that the maximum number of times that the value of each variable can change is also a constant (except the special case of variables whose values can change indefinitely). This is not a new idea: imposing this type of bound on the number of variable value changes is similar to the notion of bounded local depth (Brafman and Domshlak, 2006).

Finally, an important open question remains: what is the complexity of solving planning problems in $\mathbf{P}(k)$ *optimally* for fixed $k > 1$? Developing useful heuristics based on $\mathbf{P}(k)$ would likely have a much higher chance of succeeding if optimal solutions could be generated in polynomial time. However, we have failed to devise such algorithms, nor have we been able to prove that optimal planning for $\mathbf{P}(k)$ is NP-hard. Thus, for the moment, determining the complexity of optimal planning for $\mathbf{P}(k)$ has to be left for future work.

Appendix A. Proof of Proposition 3.15

Recall that the goal is to reduce a planning problem $P \in \mathbf{P}(k)$ to a series of Max-Change problems in $\boldsymbol{\mathcal{I}}(k)$. The discussion following Proposition 3.15 explains how to do this in polynomial time. We now describe a variation of the same idea that yields a running time $O(T|V| + |A|)$, where T is an upper bound on the values $N(v)$ for variables $v \in V$ with $N(v) < \infty$.

We assume that the reader is familiar with the notation of the discussion following Proposition 3.15. First note that every action in A corresponds to at most k edges in the causal graph of P, so that a topological sort of V can be done in linear time $O(|A|)$. Also, note that the size of the Max-Change problem corresponding to v is $O(T+|A_v|)$ and that the size of the output is at most $O(T)$, so that the whole process for each $v \in V$ takes time $O(T|V|+|A|)$ (disregarding the time required to solve the Max-Change problems).

It remains to show that the plans π_v of size at most T resulting from the Max-Change problems can be merged efficiently. We cannot explicitly construct the plans solving the subproblems $P(v)$ for each v, since the size of such a plan can be up to $T|\mathsf{anc}(v)| = O(T|V|)$. Hence, just to generate all the intermediate plans solving $P(v)$ for each $v \in V$ would take time $O(T|V|^2)$.

We describe a process that generates the actions of a plan π solving P *in sequential order* from the plans π_v, without constructing the intermediate plans solving $P(v)$. The process uses the following variables, whose values

change: the causal graph $G = (V, E)$ of P; a state $s := \mathsf{init}$ to store the current state of variables in V during the execution of the plan π we are creating; for each $v \in V$, a list l_v containing the actions of the plan π_v obtained from solving the corresponding MAX-CHANGE problem, the first element of l_v being the first action of π_v. In case no plan π_v was ever generated because $N(v) = 0$, $N(v) = 1$ or $N(v) = \infty$, the list l_v contains, respectively, no action, a single action $p \in A^0_v$, or $T + 1$ alternations of any pair of actions p, q such that $p \cap \overline{q} = \emptyset$. Let a_v denote the first action of a non-empty list l_v. We say that the *state of v in s satisfies a partial state p* if $\{v, \neg v\} \cap p \subseteq \{v, \neg v\} \cap s$, that is, if v does not appear in $V(p)$, or if it does, its value coincides with the value of v in s.

The process consists in the iterative application, in any order, of the following two rules for variables $v \in V$ appearing in the graph G:

Rule 1. If v has no successor in G, and the state of v in s satisfies goal, then remove v from G.

Rule 2. If v is such that s satisfies the pre-condition $\mathsf{pre}(a_v)$ (that is, $\mathsf{pre}(a_v) \subseteq s$), and for each of the $m \geq 0$ successors $v_1, \ldots, v_m$ of v in G either l_{v_i} is empty, or the value of v in s does not satisfy the pre-condition $\mathsf{pre}(a_{v_i})$, then we apply action a_v. More precisely, we add a_v to the plan π, we apply a_v to s, and we remove a_v from l_v.

We make several observations regarding the previous process. The actions changing v that appear in π form an initial segment of π_v, and they appear in the same order; whenever we add an action a_v to π, it is applicable in the current state s; whenever we remove a variable from G, the value of v satisfies the goal state; if a list l_v runs out of actions, then the value of v also satisfies the goal state (this is simply due to a counting argument: l_v has an even number of actions if $v \in \mathsf{goal}$, and an odd number if $\overline{v} \in \mathsf{goal}$); whenever the value of a predecessor u of v changes, then either a_v requires u to change, or l_v has no actions. This last property is important, because it means that the predecessors of v do not change their value more times than necessary to execute the plan π_v. In particular, if l_v runs out of actions, then no successor of v will require v to further change its value.

It remains to show that this process always ends with G empty. Assume not, i.e., assume that the process ends with a non-empty $G' = (V', E')$. Based on this graph, we explain how to construct a non-empty subgraph $G'' = (V'', E'')$ of G' with the property that $|E''| \geq |V''|$. This contradicts the fact that G is a polytree.

When the process finishes, we cannot apply Rule 2 to any $v \in V'$. There are only three possible reasons: v has some successor $v_i \in V'$ such that the state of v in s satisfies $\mathsf{pre}(a_{v_i})$; v has some predecessor $v' \in V'$ such that the state of v' in s does not satisfy $\mathsf{pre}(a_v)$; l_v is empty. If the last case holds, we do not add v to V''. If the first case holds, we add v to V'', and a directed edge (v, v_i) to E''; note that, in this case, l_{v_i} is not empty. If the second case holds, we add v to V'' and a directed edge (v', v) to E''; note that since the state of v' in s does not satisfy $pre(a_v)$, the list $l_{v'}$ cannot be empty by the observation above. The fact that lists l_{v_i} or $l_{v'}$ are not empty implies that vertices v_i or v' will eventually be added to V'', ensuring that the set E'' is indeed a subset of $V'' \times V''$.

Finally, observe that the same edge (u, v) is never added twice to E'': indeed, when considering v, an edge (u, v) means that the value of u in s does not satisfy the pre-condition of the action a_v; when considering u, an edge (u, v) means that the value of u in s satisfies the pre-condition of a_v. Since whenever we add a vertex v to V'' we also add an edge E'', and no edge is added twice, it follows that $|E''| \geq |V''|$, as desired.

We now justify that the successive application of the previous two rules until G becomes empty can be done in time proportional to the size of the resulting plan π, that is, $O(N|V|)$. To this end, we describe how to find a vertex v to apply a rule to, and how to apply the rule to v, in amortized constant time. For each variable v, maintain a counter of the number of predecessors v' or successors v_i of v that prevent to apply Rule 2 to v. Note that whenever we apply a rule to v, we only need to update the counter of v and the counters of its neighbors. This can be done in time proportional to the number of neighbors of v. That is, we can do this process in amortized constant time, because the average number of neighbors for a vertex v in a polytree is less than 2. Finally, observe that we never need to look for a new vertex to apply the second rule to, since we can maintain a list of vertices whose counters are 0 (to apply Rule 2 to), and a list of vertices with no successors that are in the goal state (to apply Rule 1 to). This shows that the merging can be done in time $O(N|V|)$.

Appendix B. Lower and Upper Bounds on $f(2)$

By Definition 4.8, the value of $S(3,2)$ is given by

$$S(3,2) = 2\sum_{i=0}^{2}\binom{2\cdot 3}{i} = 2\left(\binom{6}{0}+\binom{6}{1}+\binom{6}{2}\right) = 2(1+6+15) = 44.$$

By Definition 4.17, the value of $f(2)$ is given by

$$f(2) = G_2(3,1) = G_2(3+\sum_{\ell=1}^{S(3,2)} b_2(3,1,\ell),0) = 3+\sum_{\ell=1}^{44} b_2(3,1,\ell).$$

It remains to estimate the value of each $b_2(3,1,i)$, $1 \le i \le 44$. By definition, $b_2(3,1,1) = 1(44-1) = 43$. For $i > 1$, the formula for $b_2(3,1,i)$ is given by

$$b_2(3,1,i) = 1(44-i) + 1\cdot S(G_2(3+\sum_{\ell=1}^{i-1} b_2(3,1,\ell),0),2) =$$
$$= (44-i) + S(3+\sum_{\ell=1}^{i-1} b_2(3,1,\ell),2).$$

We obtain bounds on $b_2(3,1,i)$ in the following way. For any $t > 0$, $S(t,2) = 1+2t+2t(2t-1)/2 = 2t^2+t+1$, so that $t^2 < S(t,2) \le 4t^2$ for any $t > 0$. Likewise, for any $i > 1$, $3+\sum_{l=1}^{i-1} b_2(3,1,i-1) > b_2(3,1,i-i)$. For $1 < i \le 44$ we thus have

$$b_2(3,1,i) = (44-i) + S(3+\sum_{\ell=1}^{i-1} b_2(3,1,\ell),2) \ge S(\sum_{\ell=1}^{i-1} b_2(3,1,\ell),2) >$$
$$> S(b_2(3,1,i-1),2) > b_2^2(3,1,i-1).$$

With $b_2(3,1,1) = 43$, we obtain $b_2(3,1,i) \ge 43^{2^{i-1}}$. On the other hand, the inequality $b_2(3,1,i) > b_2^2(3,1,i-1)$ implies that $b_2(3,1,i)$ is a strictly increasing function of i, so that $3+\sum_{l=1}^{i-1} b_2(3,1,l) < ib_2(3,1,i-1)$. Thus, for $i > 1$ it holds that

$$b_2(3,1,i) = (44-i) + S(3+\sum_{l=1}^{i-1} b_2(3,1,l),2) \le 42 + 4(ib_2(3,1,i-1))^2 \le$$
$$\le 5(ib_2(3,1,i-1))^2.$$

By iteration, we obtain $b_2(3,1,i) \leq 5^i (i!)^2 43^{2^{i-1}}$. A lower and upper bound for $f(2)$ are thus given by

$$f(2) = 3 + \sum_{l=1}^{4} 4b_2(3,1,l) > b_2(3,1,44) \geq 43^{2^{43}},$$

$$f(2) = 3 + \sum_{l=1}^{4} 4b_2(3,1,l) < 3 + 44b_2(3,1,44) \leq 3 + 44 \cdot 5^{44} 44!^2 43^{2^{43}}.$$

References

Amir, E., Engelhardt, B., 2003. Factored Planning. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence. pp. 929–935.

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

Brafman, R., Domshlak, C., 2006. Factored Planning: How, When, and When Not. In: Proceedings of the 21st National Conference on Artificial Intelligence. pp. 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., Giménez, O., 2008. Causal Graphs and Structurally Restricted Planning. In: Proceedings of the 18th International Conference on Automated Planning and Scheduling. pp. 36–43.

Chen, Y., Huang, R., Zhang, W., 2008. Fast Planning by Search in Domain Transition Graphs. In: Proceedings of the 23rd National Conference on Artificial Intelligence. pp. 886–891.

Edelkamp, S., 2001. Planning with pattern databases. In: Proceedings of the 6th European Conference on Planning. pp. 13–24.

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

Giménez, O., Jonsson, A., 2009a. The Influence of k-Dependence on the Complexity of Planning. In: Proceedings of the 19th International Conference on Automated Planning and Scheduling. pp. 138–145.

Giménez, O., Jonsson, A., 2009b. 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., Koenig, S., 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In: Proceedings of the 22nd National Conference on Artificial Intelligence. pp. 1007–1012.

Helmert, M., 2006a. Solving Planning Tasks in Theory and Practice. Ph.D Thesis, Albert-Ludwigs-Universität, Freiburg, Germany.

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

Helmert, M., Haslum, P., Hoffmann, J., 2007. Flexible abstraction heuristics for optimal sequential planning. In: Proceedings of the 17th International Conference on Automated Planning and Scheduling. pp. 176–183.

Jonsson, A., 2009. The Role of Macros in Tractable Planning. Journal of Artificial Intelligence Research 36, 471–511.

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

Katz, M., Domshlak, C., 2008b. Structural Patterns Heuristics via Fork Decompositions. In: Proceedings of the 18th International Conference on Automated Planning and Scheduling. pp. 182–189.

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

Williams, B., Nayak, P., 1997. A reactive planner for a model-based executive. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence. pp. 1178–1185.