Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Tractable Cost-Optimal Planning over Restricted Polytree Causal Graphs

Meysam Aghighi, Peter Jonsson and **Simon Ståhlberg**
Department of Computer and Information Science
Linköping University
Linköping, Sweden
{meysam.aghighi, peter.jonsson, simon.stahlberg} at liu.se

Abstract

Causal graphs are widely used to analyze the complexity of planning problems. Many tractable classes have been identified with their aid and state-of-the-art heuristics have been derived by exploiting such classes. In particular, Katz and Keyder have studied causal graphs that are hourglasses (which is a generalization of forks and inverted-forks) and shown that the corresponding cost-optimal planning problem is tractable under certain restrictions. We continue this work by studying polytrees (which is a generalization of hourglasses) under similar restrictions. We prove tractability of cost-optimal planning by providing an algorithm based on a novel notion of variable isomorphism. Our algorithm also sheds light on the k-consistency procedure for identifying unsolvable planning instances. We speculate that this may, at least partially, explain why merge-and-shrink heuristics have been successful for recognizing unsolvable instances.

1 Introduction

It is well known that (first-order) planning is undecidable and propositional planning is PSPACE-complete (Bylander 1994). Despite this, propositional planners have been highly successful and are nowadays able to find solutions for relatively large instances. A common approach for finding solutions quickly is by using heuristic search (Katz and Domshlak 2010). One popular approach for constructing heuristic functions is to: 1) find a tractable (i.e. polynomial-time solvable) class of planning instances, 2) abstract the original instance to an instance in the tractable class and solve it, and 3) use the cost of the solution for the abstract instance as a heuristic value (cf., Helmert (2004); Helmert, Haslum and Hoffman (2007); Katz and Domshlak (2010)). An ongoing quest in the planning community is to map as many and as general tractable classes as possible, since having knowledge of these might prove invaluable for constructing efficient planners (Katz and Domshlak 2008a).

Before we delve into complexity results, we define the problems under consideration. Let Θ be an arbitrary set of SAS$^+$ instances (V, A, I, G, c) where $c : A \rightarrow \mathbb{N}_0$ is an action cost function. We have the following computational problems.

- PG(Θ): (Plan Generation) Generate a plan for Π or indicate that no plan exists.
- COPG(Θ): (Cost-optimal Plan Generation) Generate a plan with minimal total cost or indicate that no plan exists.

We let COPE(Θ, k) denote the existence version of COPG(Θ), i.e. determine the weight of the plan with minimal total cost or indicate that no plan exists. Note that generating a plan cannot be computationally easier than determining the existence of one. Furthermore, PG cannot be easier than COPG.

A common approach to the problem of identifying tractable instances is to impose restrictions on the *causal graph* (cf., Brafman and Domshlak (2003), Katz and Domshlak (2007), or Gimenéz and Jonsson (2012)). In this paper, we consider instances whose causal graph is a *polytree*, i.e. acyclic graphs whose underlying undirected graphs are trees. PG is known to be NP-hard for instances whose causal graph is a directed path and having domain size ≥ 5 (Giménez and Jonsson 2009). This implies that such instances must be avoided when searching for tractable fragments. A natural parameter when considering causal graphs is the *diameter*, i.e. the length of the longest simple path in the undirected graph underlying the causal graph. Restrictions on the diameter have been used in previous work. For instance, Katz & Domshlak (2008b; 2010) have studied *forks* and *inverted-forks* while Katz & Keyder (2012) have studied *hourglasses*. In all these cases, the causal graph is a polytree with diameter ≤ 2. With this in mind, the restrictions we use in this paper are the following: the causal graph is a polytree, the domain size (of the variables) is bounded by a constant k, and the diameter of the causal graph is bounded by a constant d.

Table 1 shows the current known results for instances using these restrictions. We note that some of the previous work have used a slightly different set of restrictions and is more general, but the results are still applicable to the set of restrictions we consider. These results leave an open case: what happens if both the domain size and the diameter are bounded by arbitrary constants? We show that COPE is in P which immediately implies that COPG is in P, too. This follows from the fact that the length of our optimal plans are bounded by a polynomial in the instance size so we can use prefix search for generating a cost-optimal plans given that

d \ k	2	fix. const	free
2	COPG is in P	COPG is in P*	PG is NP-h.**
fix. const	**COPG is in P**	**COPG is in P**	PG is NP-h.
free	PG is NP-h.***	PG is NP-h.	PG is NP-h.

Table 1: The known results about instances whose causal graphs are polytrees, domain size is at most k and the diameter of the causal graph is at most d. The shaded cells are results that can be derived from the neighbouring cells, and the results in bold are the new results presented in this paper. *: (Katz and Keyder 2012). **: (Domshlak and Dinitz 2001). ***: (Giménez and Jonsson 2008).

we have a polynomial-time algorithm for COPE.

We devise an algorithm built on the notion of *isomorphic variables*. Intuitively, two variables are isomorphic if the stucture of their domain transition graphs are isomorphic with regard to initial value, goal value, actions, prevailconditions, and so on. In other words, the two variables will behave exactly the same in every optimal solution, which allows us to combine them into a single variable. By combining isomorphic variables, we can gradually reduce the size of the initial instance, and this allows us to devise an algorithm that runs in polynomial time. We suspect that the notion of variable isomorphism may have other uses in planning besides what is demonstrated in this paper.

Let us compare our algorithm with some other algorithms for instances with polytree causal graphs. Brafman and Domshlak (2003) gave a polynomial-time algorithm for instances with binary variables and where the causal graph is a polytree such that the indegree of every vertex is bounded by a constant. Giménez and Jonsson (2012) later generalized this result by giving an algorithm that, instead of restricting the indegree, restricts the number of prevailconditions. Note that a bound on the indegree implies a bound on the number of prevailconditions but not the other way round. Our algorithm is a bit different: it allows not only binary domains, but any domain size bounded by a constant and it allows the actions to have an unbounded number of prevailconditions. However, this flexibility comes at the expense of the diameter which must be bounded by a constant.

The paper has the following structure. We formally define the planning framework in Section 2 together with some graph-theoretic preliminaries. In Section 3, we introduce the concept of isomorphic variables. We continue in Section 4 by describing and analysing certain operations on planning instances. The actual algorithm is presented in Section 5, we prove its correctness in Section 6. Finally, in Section 7, we discuss the results, and ways of using the isomorphism concept by investigating its connections with the identification of unsolvable instances. Some proofs are omitted due to space constraints.

2 Preliminaries

2.1 Planning Framework

We use the SAS$^+$ formalism (Bäckström and Nebel 1995). A SAS$^+$ planning instance is a tuple $\Pi = (V, A, I, G)$ where $V = \{v_1, \ldots, v_n\}$ is the set of *variables* each one associated with a *domain* D_v. We add a special value $\mathbf{u}$ to this domain (where $\mathbf{u}$ stands for *undefined*) resulting in a new domain D_v^+. $\mathcal{S}_v = D_{v_1} \times \ldots \times D_{v_n}$ is the set of *total states*, and $\mathcal{S}_v^+ = D_{v_1}^+ \times \ldots \times D_{v_n}^+$ is the set of *partial states* over V. The value of a variable v in a state $s \in \mathcal{S}_v^+$ is denoted as $s[v]$. $I \in \mathcal{S}_v$ is the *initial state* and $G \in \mathcal{S}_v^+$ is the *goal state*. A is the set of *actions* where every action $a \in A$ has a *precondition* pre$(a) \in \mathcal{S}_v^+$ and an *effect* eff$(a) \in \mathcal{S}_v^+$. An action has a *prevailcondition* on a variable v if pre$(a)[v] \neq \mathbf{u}$ and eff$(a)[v] = \mathbf{u}$.

For two states s_1, s_2, we write $s_1 \sqsubseteq s_2$ if and only if for all $v \in V$, either $s_1[v] = \mathbf{u}$ or $s_1[v] = s_2[v]$. An action a is *applicable* in a state $s \in \mathcal{S}_v$ if and only if pre$(a) \sqsubseteq s$. The *result of a in s*, if a is applicable in s, is the state $t \in \mathcal{S}_v$ such that for all $v \in V, t[v] = s[v]$ if eff$(a)[v] = \mathbf{u}$, otherwise $t[v] =$ eff$(a)[v]$. For a variable $v \in V$, we define two sets of actions $A^v = \{a \in A : \text{pre}(a)[v] \neq \mathbf{u}\}$ and $A_v = \{a \in A : \text{eff}(a)[v] \neq \mathbf{u}\}$.

Given two states $s_I, s_G \in \mathcal{S}_v$, a sequence of actions $\omega = \langle a_1, \ldots, a_m \rangle$ is called a *plan* from s_I to s_G if and only if there exists a set of states $\{s_1, \ldots, s_{m-1}\}, s_i \in \mathcal{S}_v$, such that s_1 is the result of a_1 in s_I, s_i is the result of a_i in s_{i-1} for all $2 \leq i \leq m-1$, and s_G is the result of a_m in s_{m-1}. We also say that ω is a solution for the SAS$^+$ instance $\Pi = (V, A, I, G)$ if and only if ω is a plan from I to some state s_G such that $G \sqsubseteq s_G$.

We use the notation $a = \{v = d, w = d''\} \overset{c}{\Rightarrow} \{v = d'\}$ to denote an action a with cost c. This action in particular changes the value of v from d to d' and has the prevailcondition $w = d''$.

Let $\omega = \langle a_1, \ldots, a_n \rangle$ be a sequence of actions, then $\mathcal{P}(\omega, v) = \langle \text{pre}(a_1)[v], \ldots, \text{pre}(a_n)[v] \rangle$ denote the prevail- or preconditions ω has on the variable v. Also let $\mathcal{S}(\omega, v) = \langle I[v], \text{eff}(a_1)[v], \ldots, \text{eff}(a_n)[v] \rangle$ denote the values v visits. If either the prevail-, precondition or effect is undefined for v, then the result is simply $\mathbf{u}$.

2.2 Graphs

The *causal graph* CG_Π of a SAS$^+$ instance $\Pi = (V, A, I, G)$ is the digraph (V, E) where an arc (v_1, v_2), $v_1 \neq v_2$ belongs to E if and only if there exists an action $a \in A$ such that eff$(a)[v_2] \neq \mathbf{u}$ and either pre$(a)[v_1] \neq \mathbf{u}$ or eff$(a)[v_1] \neq \mathbf{u}$. The *domain transition graph* DTG_v for a variable $v \in V$ is the labeled digraph (D_v, A) where for every distinct $d_1, d_2 \in D_v$, $(d_1, d_2) \in A$ if and only if there exists an action $a \in A$ such that pre$(a)[v] \in \{d_1, \mathbf{u}\}$ and eff$(a)[v] = d_2$. Furthermore, we label the arcs with the set of sets of prevailconditions (one such set per action).

Consider a digraph $G = (V, A)$. G is a *polytree* if it is acyclic and the underlying undirected graph is a tree. Planning instances whose causal graph is a polytree only have unary actions (a is unary if there is only one $v \in V$ such that eff$(a)[v] \neq \mathbf{u}$), otherwise a cycle would have been introduced in the causal graph. The diameter of G is the longest shortest path in the underlying undirected graph. A vertex $v \in V$ is a leaf if the degree of v is exactly 1. Furthermore, we say that v is an *ingoing leaf* if $(v, w) \in A$, and we say that v is an *outgoing leaf* if $(w, v) \in A$, for some vertex $w \in V$. A vertex $v \in V$ is *critical* if it is connected to at least one leaf, and at most one non-leaf. The non-leaf connected to a critical vertex is the *parent*.

We let ζ denote the set of SAS$^+$ instances that have polytree causal graph with bounded diameter d and bounded domain size k.

3 Isomorphism

Identifying *isomorphic variables* is the keystone of our algorithm; in fact, we will only need to take leaves into consideration. The basic idea is to look for isomorphic leaves that are connected to the same critical vertex and remove them in a systematic way. The algorithm will remove variables such that the set of critical vertices changes. The next lemma guarantees that we will not run out of critical vertices.

Lemma 1. *Every tree consisting of at least three vertices has a critical vertex.*

Proof. Let $G = (V, E)$ be a tree, and since $|V| \geq 3$ there exists a vertex $v \in V$ that is not a leaf. Remove all of the leaves of G along with their connected edges, resulting in $G' = (V', E')$. Since v is not a leaf of G then $v \in V'$, i.e., $|V'| \geq 1$. Furthermore, per definition, every leaf of G' is a critical vertex of G and there exists at least one leaf. $\square$

In graph-theoretic terms, two graphs (V, A), (V', A') are said to be *isomorphic* if there exists a bijective function $\sigma : V \to V'$ such that $(v, w) \in A$ if and only if $(\sigma(v), \sigma(w)) \in A'$. However, we obviously need to take more structure into consideration when considering isomorphisms between variables in planning instances. This implies that we need a slightly more involved definition.

Definition 1. Let $\Pi \in \textsc{Cope}(\zeta)$. Two leaves $l_1, l_2 \in \mathit{CG}_\Pi$ that are connected to the critical vertex x are isomorphic if and only if there exists a bijective function $\sigma : D_{l_1} \to D_{l_2}$ such that:

1. DTG_{l_1} is isomorphic to DTG_{l_2} by σ,
2. $\sigma(I[l_1]) = I[l_2]$ and $\sigma(G[l_1]) = G[l_2]$ (where $\sigma(\mathbf{u}) = \mathbf{u}$),
3. $A^{l_1} \setminus A_{l_1} = A^{l_2} \setminus A_{l_2}$, and for every $a \in A^{l_1} \setminus A_{l_1}$, $\sigma(\text{pre}(a)[l_1]) = \text{pre}(a)[l_2]$,
4. For every $(d, d') \in \mathit{DTG}_{l_1}$ with label L_1, $(\sigma(d), \sigma(d')) \in \mathit{DTG}_{l_2}$ with label L_2, $L_1 = L_2$, and
5. For every $d, d' \in D_{l_1}$, let $\omega_{l_1}, \omega'_{l_1}$ be sequences of actions no longer than $|D_{l_1}|$ that take l_1 from d to d', and $\omega_{l_2}, \omega'_{l_2}$ be sequences of actions that take l_2 from $\sigma(d)$ to $\sigma(d')$ such that $\mathcal{P}(\omega_{l_1}, x) = \mathcal{P}(\omega_{l_2}, x)$, $\sigma(\mathcal{S}(\omega_{l_1}, l_1)) = \mathcal{S}(\omega_{l_2}, l_2)$, $\mathcal{P}(\omega'_{l_1}, x) = \mathcal{P}(\omega'_{l_2}, x)$, $\sigma(\mathcal{S}(\omega'_{l_1}, l_1)) = \mathcal{S}(\omega'_{l_2}, l_2)$. Then $R(c(\omega_{l_1}), c(\omega'_{l_1})) = R(c(\omega_{l_2}), c(\omega'_{l_2}))$, where: $R(x, y) = -1$ if $x < y$; $R(x, y) = 0$ if $x = y$; and $R(x, y) = 1$ otherwise.

We note that the definition implicitly requires l_1, l_2 to be either both in- or outgoing from p. Condition 3 requires every action in x to have the same prevailconditions on both l_1, l_2 (i.e., both are ingoing leaves), and condition 4 requires the actions in the leaves to have the same prevailconditions on x (i.e., both are outgoing leaves). Condition 5 requires the costs of the actions for l_1 and l_2 to be "compatible" with each other, i.e., the best actions to use to get to a particular value are the same no matter how x behaves.

We will now see that there are explicit bounds on the number of non-isomorphic leaves. Define $\textsc{In}(x)$ and $\textsc{Out}(x)$ to be the set of ingoing and outgoing leaves of a critical vertex x, respectively. We give a proof for out-leaves; the proof for in-leaves can be done in a similar way.

Lemma 2. *Every critical vertex x has at most $f_O(m) = m(m+1)2^{(m+1)^3}\left(((m+1)^3+1)^m!\right)$ non-isomorphic outgoing leaves, where $m = \max_{v \in \{x\} \cup \textsc{Out}(x)} |D_v|$.*

Proof. Let $l \in \textsc{Out}(x)$. Every $a \in A_l$ is of the form $\{x = \alpha, l = \beta \Rightarrow l = \gamma\}$, so $|A_l| \leq (m+1)^3$. Hence, there are at most $2^{(m+1)^3}$ different *DTG*s that l can have. l also has m different ways for an initial value and $m+1$ ways for a goal value (including no goal value). Furthermore, there are at most $(|A_l|+1)^m$ different sequences of actions of length no longer than m and they can be sorted in at most $(|A_l|+1)^m!$ different ways, which brings us to the final value $m(m+1)2^{(m+1)^3}\left(((m+1)^3+1)^m!\right)$. $\square$

Lemma 3. *Every critical vertex x has at most $f_I(m) = f_O(m)(m+1)^q$ non-isomorphic ingoing leaves, where $q = |A_x|$ and $m = \max_{v \in \{x\} \cup \textsc{In}(x)} |D_v|$.*

4 Defoliation of Polytrees

After having identified isomorphic leaves in the causal graph, our algorithm will remove them. This will be accomplished with two methods where we *combine* isomorphic leaves and then take the *product* of the remaining leaves and their correpsonding critical vertex. The idea behind this is that the combination step removes sufficiently many leaves and, thanks to that, the new product vertices will have domain sizes that grow in a controlled way. This method also guarantees that the number of vertices will decrease monotonically.

Definition 2. Let $\Pi = (V, A, I, G, c)$ be an instance of $\textsc{Cope}(\zeta)$, and $v_1, v_2 \in V, v_1 \neq v_2$ be two isomorphic variables by a function σ, both connected to a critical variable $x \in V$. The resulting instance of *combining two variables* is the instance $\Pi' = (V', A', I', G', c')$ where:

- $V' = (V \setminus \{v_1, v_2\}) \cup \{v'\}$,
- $D_{v'} = D_{v_1}$,
- $A' = (A \setminus (A_{v_1} \cup A_{v_2} \cup A_x)) \cup \bar{A} \cup \hat{A}$,
- $\bar{A} = \{\lambda(a_1, a_2) : a_1 \in A_{v_1}, a_2 \in A_{v_2}, \sigma(\text{pre}(a_1)[v_1]) = \text{pre}(a_2)[v_2], \sigma(\text{eff}(a_1)[v_1]) = \text{eff}(a_2)[v_2], \text{pre}(a_1)[x] = \text{pre}(a_2)[x]\}$,
- $\lambda(a_1, a_2) : \text{pre}(a_1) \xRightarrow{c(a_1)+c(a_2)} \text{eff}(a_1)$ (replace v_1 with v')
- $\hat{A} = \{a' : a \in A_x, \forall v \in V \setminus \{v_1, v_2\}, \text{pre}(a')[v] = \text{pre}(a)[v], \text{eff}(a') = \text{eff}(a), \text{pre}(a')[v'] = \text{pre}(a)[v_1] \text{ and } c(a') = c(a)\}$

- $I'[v] = \begin{cases} I[v_1] & \text{if } v = v' \\ I[v] & \text{otherwise} \end{cases}$, and
- $G'[v] = \begin{cases} G[v_1] & \text{if } v = v' \\ G[v] & \text{otherwise} \end{cases}$.

Intuitively, we replace v_2 by removing every occurrence of it and for every action a affecting v_1 we adjust $c(a)$ to be the sum of the original cost and the cost of corresponding action for v_2. From now on, we extend the bijective function σ to A_{v_1} such that for every $a_1 \in A_{v_1}$, $\sigma(a_1) = a_2 \in A_{v_2}$ where $\sigma(\text{pre}(a_1)[v_1]) = \text{pre}(a_2)[v_2], \sigma(\text{eff}(a_1)[v_1]) = \text{eff}(a_2)[v_2]$ and $\text{pre}(a_1)[x] = \text{pre}(a_2)[x]$.

Theorem 1. *The cost of an optimal solution does not change by combining isomorphic variables.*

Proof. (*Sketch.*) Let $\Pi = (V, A, I, G, c)$ be an instance, let $v_1, v_2 \in V$ be two isomorphic variables by a function σ, and let v', Π' be the resulting variable and instance by combining v_1 and v_2. We show that there is a correspondence between solutions of Π and Π'.

Let ω be an optimal solution for Π. Let x be the parent of v_1 and v_2. We note that ω might use "non-isomorphic" sequences of actions for v_1 and v_2 to reach their respective goal value, but by Definition 1 it is easy to show that we can replace either sequence with an isomorphic sequence of actions which does not increase the total cost of the solution. We construct ω' from ω in the following way: first, remove every action in A_{v_2} from ω; second, replace every $a \in \omega \cap A_{v_1}$ with $\lambda(a, \sigma(a))$; finally, replace every $a \in \omega \cap A_x$ with the corresponding action a' from $\hat{A}$ of Definition 2. Since v_1 and v_2 are isomorphic it is straightforward to show that ω' is a solution for Π' and using condition 5 of Definition 1, they both have the same cost.

Note that the process is reversible. Let ω' be a solution for Π', then we can construct a solution ω for Π from ω' in the following way: replace every $a \in \omega' \cap \bar{A}$ with a_1 and a_2 where $a = \lambda(a_1, a_2)$; and replace every $a' \in \omega' \cap \hat{A}$ with the corresponding $a \in A_x$ (according to definition of $\hat{A}$ in Definition 1). The rest of the actions remain unchanged. Therefore, the cost of an optimal solution for Π and Π' is the same. □

By combining leaves (with a common critical vertex) we can achieve the bounds given by Lemmata 2 and 3. After doing so we will construct a new variable whose domain is the *product* of all the leaves and the critical vertex. The product of two variables is a variable whose domain consists of possible states the two variables can be in, and an action for one of the original variables are then instantiated to several actions, one for each applicable "state" in the domain of the new variable. The definition of product is very similar to synchronized product or the merge step from merge-and-shrink. The result from merge of M&S ignores the dependencies between variables that have not been merged, while Definition 3 maintains these dependencies. In other words, taking the product of two variables by Definition 3 changes the representation of the problem, but not the problem itself.

Definition 3. Let $\Pi = (V, A, I, G, c)$ be an instance of COPE(ζ) and $v_1, v_2 \in V$. The *product* of v_1 and v_2 is the instance $\Pi' = (V', A', I', G', c')$:

- $V' = (V \setminus \{v_1, v_2\}) \cup \{v_3\}$
- $D_{v_3} = \{\{d_1, d_2\} : d_1 \in D_{v_1}, d_2 \in D_{v_2}\} \cup \{\{g\}\}$ (assume $D_{v_1} \cap D_{v_2} = \emptyset$)
- $A' = A_1 \cup A_2 \cup A_3 \cup A_g$ where:
  - **Actions with an effect on v_1 or v_2:** $A_1 = \bigcup_{a \in A} \{\{v_3 = d\} \cup \text{pre}(a) \setminus \{v_1 = \text{pre}(a)[v_1], v_2 = \text{pre}(a)[v_2]\} \xRightarrow{c(a)} \{v_3 = d'\} : d \in D_{v_3}, d' \in D_{v_3}, \text{pre}(a)[v_1] \in d \cup \{\mathbf{u}\}, \text{pre}(a)[v_2] \in d \cup \{\mathbf{u}\}, \{\text{eff}(a)[v_1], \text{eff}(a)[v_2]\} \cap d = \emptyset, \{\text{eff}(a)[v_1], \text{eff}(a)[v_2]\} \cap d' \neq \emptyset, |d \cap d'| = 1\}$
  - **Actions with a prevailcondition on v_1 or v_2:** $A_2 = \bigcup_{a \in A} \{(\text{pre}(a) \cup \{v_3 = d\}) \setminus \{v_1 = \text{pre}(a)[v_1], v_2 = \text{pre}(a)[v_2]\} \xRightarrow{c(a)} \text{eff}(a) : d \in D_{v_3}, \text{eff}(a)[v_1] = \text{eff}(a)[v_2] = \mathbf{u}, \text{pre}(a)[v_1] \neq \text{pre}(a)[v_2], \{\text{pre}(a)[v_1], \text{pre}(a)[v_2]\} \subseteq d \cup \{\mathbf{u}\}\}$
  - **Actions with no effect nor prevailcondition on v_1 or v_2:** $A_3 = \{a : \text{pre}(a)[v_1] = \text{pre}(a)[v_2] = \text{eff}(a)[v_1] = \text{eff}(a)[v_2] = \mathbf{u}\}$, where the actions keep their cost
  - **New goal actions:** $A_g = \{\{v_3 = d\} \overset{0}{\Rightarrow} \{v_3 = \{g\}\} : d \in D_{v_3}, G[v_1] \in d \cup \{\mathbf{u}\}, G[v_2] \in d \cup \{\mathbf{u}\}\}$
- $I' = (I \setminus \{v_1 = I[v_1], v_2 = I[v_2]\}) \cup \{v_3 = \{I[v_1], I[v_2]\}\}$
- $G' = (G \setminus \{v_1 = G[v_1], v_2 = G[v_2]\}) \cup \{v_3 = \{g\}\}$

Basically, the domain of the new variable is the set product of the domains of the first two variables, and a special value g for the goal. It can be seen that for every action at most $\max\{|D_{v_1}|, |D_{v_2}|\}^2$ actions are added plus $|A_g|$. Once again, we omit the technical proof of the following result.

Theorem 2. *The cost of an optimal solution does not change by taking the product of two variables.*

We finally make the following simple observation.

Lemma 4. *The product of s variables having domain size of at most k results in a variable with domain size of at most $(k + 1)^s$.*

5 The Algorithm

We now present our algorithm, which can be found in Figure 1. The steps in the algorithm is roughly the following: 1) select a critical vertex x, 2) identify the isomorphic leaves of x, 3) combine the isomorphic leaves, and 4) take the product of the remaining leaves and the critical vertex. The resulting instance has strictly fewer variables so this process may be repeated in a recursive fashion. The algorithm terminates when there is two or fewer variables left, at which point the algorithm takes the product of the variables and uses Dijkstra's algorithm to compute the cost of an optimal solution.

Recall that, for a critical vertex x, the number of ingoing leaves for x is bounded by a function in $|A_x|$. We need a constant bound on the number of leaves so this poses a problem. In other words, we must find a way to construct a set $\mathcal{R}_x \subseteq A_x$ that contains every action used in an optimal solution and $|\mathcal{R}_x|$ is bounded by a constant. Then, we can safely replace A_x with $\mathcal{R}_x$. By knowing how many times a variables needs to change at most, we can construct the set $\mathcal{R}_x$.

Theorem 3. *Let Π be an instance of* COPE(ζ). *The value of each variable in Π changes at most μ times in every optimal solution, where*

$$\mu = \underbrace{t^{t^{\cdot^{\cdot^{\cdot^{t}}}}}}_{d\ times} \qquad t = (k(k+1))^2$$

Proof. Let *Change*$_\omega(v)$ be the number of times that the value of variable v changes in an optimal solution ω. We claim that *Change*$_\omega(v) \leq pk(k+1)^p + k$, where $p = \max\{\textit{Change}_\omega(v') : (v, v') \in \textit{CG}_\Pi\}$, and $p = 0$ when there are no such variables v'. Each one of the v' variables changes at most p times, so each one of them can only have p prevailconditions on v. Therefore, there are at most $(k+1)^p$ different sequences of prevailconditions on v. Variable v takes at most p different values in each sequence of prevailconditions, and there are at most k changes between every two consecutive values (corresponding to a path in *DTG*$_v$). This brings the number of changes up to $pk(k+1)^p$. Finally, there are at most k actions needed to change the value of v to $G[v]$ so *Change*$_\omega(v) \leq pk(k+1)^p + k$. By repeated use of this inequality and the fact that the diameter for *CG*$_\Pi$ is bounded by d, a straightforward induction proof shows that *Change*$_\omega(v) \leq \mu$. □

Let RELEVANTACTIONS(Π, x) where $\Pi = (V, A, I, G, c)$ and $x \in V$ be a function that computes the set $\mathcal{R}_x$ (with the properties we mentioned earlier) for the critical vertex x. Let p be the parent of x (we assume there exists one, otherwise it is trivial). The function does the following:

1. Let $\mathcal{R}_x = \emptyset$.
2. For every possible sequence of prevailconditions $P = \langle d_1, \ldots, d_n \rangle$, $d_i \in D_p^+$, $n \leq \mu$ compute the cheapest sequence $\omega_x = \langle a_1, \ldots, a_n \rangle$ such that $\mathcal{P}(\omega_x, p) = P$. The cost associated with ω_x is the sum of $c(\omega_x)$ and the cost of the cheapest possible action sequence for the leaves with regard to the prevailconditions of ω_x. The cheapest ω_x can be computed by simply trying all $|A_x|^\mu$ combinations. We then set $\mathcal{R}_x$ to $\mathcal{R}_x \cup \{a_1, \ldots, a_n\}$. The cost of the cheapest possible action sequence for the leaves can be computed in the following way:
   (a) For every $l \in$ IN(x), ω_x might have prevailconditions on l. To compute the cost of the cheapest path to the next prevailcondition, or goal value, we simply try every path in *DTG*$_l$ no longer than $|D_l|(\mu + 1)$.
   (b) For every $l \in$ OUT(x), ω_x does not have any prevailconditions on l, but the other way around may happen. To compute the cost of the cheapest sequence of actions, we simply try every sequence for l no longer than $|D_l|$ such that its prevailconditions can be satisfied by ω_x.
3. Return $\mathcal{R}_x$.

```
function POLYTREE($\Pi = (V, A, I, G, c)$)
    if $|V| \leq 2$ then
        return PRODUCT($\Pi$, $V$)
    end if
    $x$ = CRITICALVERTEX($\Pi$)
    $(V_o, \Pi_1)$ = COMBINEISOVARS($\Pi$, OUT($x$))
    $\mathcal{R}_x$ = RELEVANTACTIONS($\Pi_1$, $x$)
    Replace $A$ with $(A \setminus A_x) \cup \mathcal{R}_x$
    $(V_i, \Pi_2)$ = COMBINEISOVARS($\Pi_1$, IN($x$))
    $\Pi_3$ = PRODUCT($\Pi_2$, $\{x\} \cup V_o \cup V_i$)
    return POLYTREE($\Pi_3$)
end function
```

Figure 1: The algorithm devised in this paper, which outputs a new instance with a single variable of constant size.

If there exists a solution for the original Π, then p has a sequence of prevailconditions on x that is not longer than μ and we have computed the cheapest sequence of actions for it. An optimal solution (if there is one) is preserved by replacing A_x with $\mathcal{R}_x$, which we will prove later.

The other functions occurring in the algorithm are:

- CRITICALVERTEX(Π) returns a variable that is a critical vertex in *CG*$_\Pi$.
- COMBINEISOVARS takes the instance and a set of variables who are either ingoing leaves or outgoing leaves in the causal graph, and checks if they are pairwise isomorphic, and combine them if they are.
- PRODUCT takes an instance and a set of variables and returns a new instance in which the variables in the second argument are replaced by their product.

6 Correctness and Time Complexity

We begin by proving that algorithm POLYTREE is correct.

Lemma 5. *The cost of an optimal solution does not change by replacing A with $(A \setminus A_x) \cup \mathcal{R}_x$.*

Proof. Let $\Pi = (V, A, I, G, c) \in$ COPE(ζ), $\Pi' = (V, (A \setminus A_x) \cup \mathcal{R}_x, I, G, c)$ and p the parent of x. Let Π have a solution ω and let $\alpha = \omega \cap A_x = \langle \alpha_1, \ldots, \alpha_n \rangle$, $a = \{a_i\}_{i=1}^n$ and $b = \{b_i\}_{i=1}^n$ such that $\text{eff}(\alpha_i)[x] = a_i$ and $\text{pre}(\alpha_i)[p] = b_i$. $\mathcal{R}_x$ contains enough actions for a locally optimal solution over $\{x\} \cup$ IN(x) $\cup$ OUT(x) for any (a, b). So, there exists a plan ω with actions in $(A \setminus A_x) \cup \mathcal{R}_x$ over $\{x\} \cup$ IN(x) $\cup$ OUT(x) with the same cost as the part of ω over $\{x\} \cup$ IN(x) $\cup$ OUT(x). Let $\omega' = \{a \in \omega : \nexists v \in \{x\} \cup \text{IN}(x) \cup \text{OUT}(x), \text{eff}(a)[v] \neq \mathbf{u}\}$. Now, since both ω' and ω require x, p to take the values in a, b for their execution, they can be combined together into a solution for Π' with the same cost as ω. □

Theorem 4. *For every $\Pi \in$* COPE(ζ)*, the cost of the optimal solution for* POLYTREE(Π) *and Π are the same.*

Proof. According to Lemma 1, CRITICALVERTEX(Π) returns a critical vertex x, which is connected to at least one leaf and the PRODUCT function reduces the number of variables by at least one. This means that the recursion stops after at most $|V|$ steps. Using Theorems 1, 2 and Lemma 5, the cost of the optimal solution for Π and POLYTREE(Π) are the same. □

We now turn our attention to the time complexity of algorithm POLYTREE. The main result is proven in Theorem 5 which builds on a Lemmata 6–9. Given an instance $\Pi \in$ COPE(ζ), let $||\Pi||$ denote its representational size.

Lemma 6. *During the execution of* POLYTREE*(Π), every $v \in V_\Pi$ is the output of* CRITICALVERTEX *at most once.*

Proof. Assume that $x \in V$ is the output of a CRITICALVERTEX call. After PRODUCT(Π, $\{x\} \cup V_l$), x is a leaf. Hence, it cannot be a critical vertex anymore. □

The fact that $\mathcal{R}_x$ contains at most μ actions for every pair (a, b) of sequences of length at most μ over D_x and D_p^+, results in $|\mathcal{R}_x| < \mu^2 |D_x|^\mu (|D_p| + 1)^\mu$. After replacing A_x with $\mathcal{R}_x$, we can use Lemma 3 and the bound on $|\mathcal{R}_x|$ to derive a new bound on the number of non-isomorphic ingoing leaves. Note that for instances in COPE(ζ), μ is a constant that only depends on k and d. Now let $f(k) = (k + 1)^{f_O(k)+f_I(k)+1}$ (where f_I and f_O are the functions defined in Lemmata 2 and 3, respectively) and let $f^n = f \circ f^{n-1}$ denote the n-th iteration of f.

Lemma 7. *If y is the new variable created by* PRODUCT*(Π, $\{x\} \cup V_o \cup V_i$) (where V_o is the outgoing leaves connected to x and V_i are the ingoing), then $|D_y| \leq f(m)$ where $m = \max\{|D_v| : v \in \{x, p\} \cup V_o \cup V_i\}$ and p is the parent of x.*

Proof. This is straightforward according to Lemmata 2, 3, 4 and the fact that f_I and f_O are strictly increasing. □

Lemma 8. $|D_{\text{POLYTREE}(\Pi)}| \leq f^d(k)$ *for* $\Pi \in$ COPE(ζ).

Proof. Assume to the contrary that v is the variable in POLYTREE(Π) and $|D_v| > f^d(k)$. Going one step back and using Lemma 7, either one of the multiplied vertices or the parent of v must have had a domain size larger than $f^{d-1}(k)$. According to Lemma 6, v has not been a critical vertex before so it has a domain size of at most k. Therefore, there exists a vertex among the leaves or the parent of v, call it v_1, such that $|D_{v_1}| > f^{d-1}(k)$. Again, going one step back, there exists a vertex among the leaves or the parent of v_1, call it v_2, such that $|D_{v_2}| > f^{d-2}(k)$. Continuing this, we reach a vertex, v_d, such that $|D_{v_d}| > k$. Since the domain size of v_d is greater than k, it cannot be a variable in the original instance Π, so we must have another vertex, call it v_{d+1}, that was either a leaf for v_d or its parent if we go back again. This means that there is a path of length $d + 1$ from v to v_{d+1} in the polytree, but the causal graph for Π is supposed to be a tree with a diameter of d and we have a contradiction. □

Lemma 9. RELEVANTACTIONS *runs in polynomial time.*

Proof. As a result of Lemma 8, the domain size of all variables is bounded by $f^d(k)$ throughout the algorithm. It is not hard to see that the algorithm runs in polynomial time by combining this fact with the fact that μ is a constant. □

Theorem 5. COPE(ζ) *is in P.*

Proof. Let $\Pi = (V, A, I, G, c)$ be an instance of COPE(ζ). The functions appearing in the algorithm run in polynomial time. In particular, one may note COMBINEISOVARS runs in polynomial time since Lemma 8 implies that the variable domains are of constant size which implies that the isomorphism check can be performed in polynomial time. Hence, according to Lemma 6, POLYTREE(Π) runs in polynomial time, too. Finally, applying Dijkstra's algorithm to the resulting DTG does not change the overall complexity due to Lemma 8 and the fact that the action weights are bounded by a polynomial (that only depends on k and d) in $||\Pi||$. □

In order to use prefix search to construct an optimal solution for the original instance Π we need an upper bound on the plan length that is less than some polynomial $p(||\Pi||)$. After running the algorithm on an instance Π, we get a new instance Π'. Let ω be an optimal solution for Π'. We know that the cost of an optimal solution for Π is $c(\omega)$, but we do not have an upper bound on the length of it. Note that every action in Π' represents at most $|V|$ actions in Π, where V is the variable set of Π, because the algorithm combines at most $|V|$ variables. Furthermore, $|\omega|$ is bounded by $f^d(k)$. Hence, a loose polynomial upper bound on the length of an optimal solution for Π is $f^d(k) \cdot |V|$.

7 Discussion

7.1 Exploiting Isomorphisms

Consider the class of instances Θ having fork causal graphs and domain size bounded by k. Definition 1 tells us when two leaves in V_Π are isomorphic. Since the domain size is bounded by k, then the number of non-isomorphic leaves is bounded by a constant c. This bound can be exploited by variable projection in the following way. It is well known (Helmert 2004) that if the projection of Π onto $V' \subseteq V_\Pi$ is unsolvable, then the Π is unsolvable. Suppose V' contains two isomorphic variables l_1, l_2. Clearly, removing l_1 or l_2 from V' would not make the projection solvable. Thus, it is pointless of to choose V' such that $|V'| > c + 1$ if one wants to check unsolvability. With this in mind, we can use the *consistency checking* algorithm suggested by Bäckström et. al (2013) for solving PE(Θ): 1) enumerate all variable sets of size c, 2) for every such variable set project the instance onto it and solve it, 3) if it is unsolvable then report that the original instance is unsolvable, and 4) if all variable sets lead to solvable instances, then report that the instance is solvable. We see that this procedure is a sound and complete method for PE(Θ) and it trivially runs in polynomial time. In fact, this is the first example of a non-trivial class of planning instances where consistency checking yields a polynomial-time algorithm that is simultaneously sound and complete.

This simple idea can, of course, be generalized. Up to now, we have only discussed isomorphic leaves but it is possible to generalize the definition to isomorphic subforks and more general substructures. While isomorphic variables would most likely be rare in practice, the observation that projecting onto a set containing isomorphic variables is pointless might hint that the inclusion of "similar" variables should be avoided. Consequently, it might be interesting to measure how similar two variables are. We speculate that choosing a set whose variables are dissimilar are the most useful for proving that an instance is unsolvable. We also speculate that this may, at least partially, explain why methods such as merge-and-shrink (Helmert, Haslum, and Hoffmann 2007) have been more successful than other methods for identifying unsolvable instances (Hoffmann, Kissmann, and Torralba 2014). The way merge-and-shrink choses variables to merge may somehow correlate to their dissimilarity.

7.2 The Algorithm

Many polynomial-time algorithms for planning under polytree causal graphs share a common problem: the degree of the polynomials are very high, cf., Giménez and Jonsson (2012) and Katz and Keyder (2012). The algorithm presented in this paper is, unfortunately, plagued with the very same problem. We believe that our algorithm can be vastly improved in at least two different ways: (1) the upper bound on the number of times a variable has to change is very pessimistic and (2) the isomorphism checking procedure is based on blind exhaustive enumeration. However, such improvements do not change the idea of enumerating many different paths in the DTGs that our and the previously mentioned algorithms are based upon. A completely new idea may be needed in order to obtain significantly lower complexity figures.

We observe that the high degree of the polynomial stems from one single step, RELEVANTACTIONS, which is needed because we allow causal graphs with unbounded indegree. If the indegree is bounded by a constant, then after merging isomorphic outgoing leaves, the number of actions affecting the critical vertex is bounded by a constant. Hence, we can skip RELEVANTACTIONS and the time complexity of the algorithm is instead $O(||\Pi||^4)$, which is dramatically better. This might hint that imposing a restriction on the indegree might be necessary to, for example, construct efficient heuristics based on polytree causal graphs. Furthermore, Katz and Domshlak (2009) implemented and evaluated an admissible heuristic based on forks and inverted-forks, which the class studied in this paper generalizes. To overcome the problem that their heuristic was computationally quite expensive, they precompiled values and stored them in a database. The very same idea may be useful for a heuristic based on our class, or a subset of it.

Acknowledgements

We would like to thank the three anonymous reviewers for their very helpful comments. Meysam Aghighi and Simon Ståhlberg are partially supported by the *National Graduate School in Computer Science* (CUGS), Sweden.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11(4):625–655.

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast detection of unsolvable planning instances using local consistency. In *6th International Symposium on Combinatorial Search (SoCS)*, 29–37.

Brafman, R. I., and Domshlak, C. 2003. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research (JAIR)* 315–349.

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

Domshlak, C., and Dinitz, Y. 2001. Multi-agent off-line coordination: Structure and complexity. In *6th European Conference on Planning (ECP)*, 34–43.

Giménez, O., and Jonsson, A. 2008. The complexity of planning problems with simple causal graphs. *Journal of Artificial Intelligence Research (JAIR)* 319–351.

Giménez, O., and Jonsson, A. 2009. Planning over chain causal graphs for variables with domains of size 5 is NP-hard. *Journal of Artificial Intelligence Research (JAIR)* 675–706.

Giménez, O., and Jonsson, A. 2012. The influence of k-dependence on the complexity of planning. *Artificial Intelligence* 177-179:25–45.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *17th International Conference on Automated Planning & Scheduling (ICAPS)*, 176–183.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *14th International Conference on Automated Planning & Scheduling (ICAPS)*, 161–170.

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. "Distance"? Who cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In *ECAI 2014 - 21st European Conference on Artificial Intelligence*, 441–446.

Katz, M., and Domshlak, C. 2007. Structural patterns of tractable sequentially-optimal planning. In *17th International Conference on Automated Planning & Scheduling (ICAPS)*, 200–207.

Katz, M., and Domshlak, C. 2008a. New islands of tractability of cost-optimal planning. *Journal of Artificial Intelligence Research* 32(1):203–288.

Katz, M., and Domshlak, C. 2008b. Structural patterns heuristics via fork decomposition. In *18th International Conference on Automated Planning & Scheduling (ICAPS)*, 182–189.

Katz, M., and Domshlak, C. 2009. Structural-pattern databases. In *19th International Conference on Automated Planning & Scheduling (ICAPS)*, 186–193.

Katz, M., and Domshlak, C. 2010. Implicit abstraction heuristics. *Journal of Artificial Intelligence Research* 39:51–126.

Katz, M., and Keyder, E. 2012. Structural patterns beyond forks: Extending the complexity boundaries of classical planning. In *26th AAAI Conference on Artificial Intelligence*, 1779–1785.