Journal of Artificial Intelligence Research 60 (2017) 595–638 Submitted 04/17; published 11/17

Time and Space Bounds for Planning

Christer Bäckström CHRISTER.BACKSTROM@LIU.SE
Peter Jonsson PETER.JONSSON@LIU.SE
Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden

Abstract

There is an extensive literature on the complexity of planning, but explicit bounds on time and space complexity are very rare. On the other hand, problems like the constraint satisfaction problem (CSP) have been thoroughly analysed in this respect. We provide a number of upper- and lower-bound results (the latter based on various complexity-theoretic assumptions such as the Exponential Time Hypothesis) for both satisficing and optimal planning. We show that many classes of planning instances exhibit a dichotomy: either they can be solved in polynomial time or they cannot be solved in subexponential time and thus require $O(2^{cn})$ time for some $c > 0$. In many cases, we can even prove closely matching upper and lower bounds; for every $\varepsilon > 0$, the problem can be solved in time $O(2^{(1+\varepsilon)n})$ but not in time $O(2^{(1-\varepsilon)n})$. Our results also indicate, analogously to CSPs, the existence of sharp phase transitions. We finally study and discuss the trade-off between time and space. In particular, we show that depth-first search may sometimes be a viable option for planning under severe space constraints.

1. Introduction

This section starts with a discussion on complexity analysis in planning, in order to situate this article in the literature. This is followed by a brief presentation of our main results and the section ends with an outline of the article.

1.1 Background

There is nowadays a quite extensive literature on the computational complexity of propositional planning. The two most well studied planning languages are the SAS^+ language, which uses multi-valued state variables, and propositional STRIPS with negative goals (PSN), which can be viewed as the special case of SAS^+ with binary variables. The most commonly studied planning problems are plan satisfiability (PSAT), which asks if there is a plan at all, length-optimal planning (LOP), which asks for a plan of minimum length, and cost-optimal planning (COP), which assigns costs to actions and asks for a plan of minimum cost. All three problems are **PSPACE**-complete, both for PSN and for SAS^+. Bylander (1994) considered subclasses of PSN by restricting the number and polarity of defined variables in the preconditions and effects of actions, and analysed the complexity of both PSAT and LOP for a large number of such subclasses. Bäckström and Nebel (1995) made an analogous classification of PSAT and LOP for (almost) all subclasses of SAS^+ defined by the so called PUBS restrictions (Bäckström & Klein, 1991). Another common way to define subclasses is to restrict the structure of the causal graph of planning problems (cf. Domshlak & Dinitz, 2001; Brafman & Domshlak, 2003; Giménez & Jonsson, 2009; Bäckström & Jonsson, 2013). In the case of COP, almost all analyses are based on such subclasses (cf. Katz & Domshlak, 2008, 2010a; Katz & Keyder, 2012; Aghighi, Jonsson, & Ståhlberg, 2015). This line of research has mainly resulted in very coarse classifications of problems as **PSPACE**-complete, **NP**-complete or tractable. There has, thus, been a recent interest in applying parameterised complexity theory, which allows for more fine-grained results, where the complexity depends both on the problem itself and the choice of parameters. Bäckström, Jonsson, Ordyniak, and Szeider (2015) analysed LOP using plan length as parameter for all subclasses of SAS$^+$ using the PUBS restrictions, finding that the problem is $\mathbf{W}[2]$-complete in the general case, but drops to $\mathbf{W}[1]$-complete or even fixed-parameter tractable for some interesting subclasses. Kronegger, Pfandler, and Pichler (2013) perfomed a multi-parameter analysis of LOP for PSN using various combinations of parameters, including plan length. Aghighi and Bäckström (2015) analysed COP with plan cost as parameter for different types of action costs and considering all subclasses of SAS$^+$ using the PUBS restrictions. They found that, in the general case, COP is $\mathbf{W}[2]$-complete, i.e. no harder than LOP, if using positive integer action costs, but it is para-**NP**-hard if allowing zero costs or arbitrary positive rational costs. This was followed up by a multi-parameter analysis using plan cost in combination with several other parameters (Aghighi & Bäckström, 2016).

All these studies have in common that they classify problems into complexity classes, either standard complexity classes or parameterised ones, rather than providing explicit upper or lower time bounds. An upper bound for a problem can usually be derived by analysing the runnning time of the fastest existing algorithm, but it is difficult to know if this bound is optimal since there may be faster algorithms that are not yet discovered. Proving good lower bounds has turned out to be much harder, and there is a huge difference between the best known upper and lower bounds for hard problems, like the SAT problem. It is, thus, common to instead prove hardness for **NP** or some other complexity class, and the whole theory of **NP**-completeness was invented for this very reason, and is based on the assumption that $\mathbf{P}\neq\mathbf{NP}$. An important step forward in proving lower time bounds was achieved by conditioning also such proofs with an assumption. The most common one is the Exponential Time Hypothesis (ETH) by Impagliazzo and Paturi (2001), which conjectures that the 3-SAT problem cannot be solved in subexponential time $2^{o(n)}$, where n is the number of variables. This has proven a very useful hypothesis since there is a large number of **NP**-complete problems that are related in the sense that either all of them can be solved in subexponential time or none of them can. Hence, proving that a problem cannot be solved in subexponential time under the assumption that the ETH holds is a very strong indication of hardness. While all **NP**-complete problems are equivalent under the theory of **NP**-completeness, it is known that they differ widely in hardness in practice. The ETH has enabled to separate the **NP**-complete problems with respect to concrete time bounds, which is more fine grained and better related to practice than the usual classifications into complexity classes. The ETH is nowadays a standard assumption in complexity theory (Lokshtanov, Marx, & Saurabh, 2011).

De Haan, Kanj, and Szeider (2015) recently used the ETH to analyse lower time bounds for the constraint satisfaction problem (CSP). They showed that the CSP problem cannot be solved in subexponential time $2^{o(n)}$, where n is the number of variables, even for a large number of common restrictions, unless the ETH is false. They also demonstrated that there is a phase-transition phenomenon in the sense that CSP can be solved in time $2^{o(n)}$ if the number of constraint tuples is subexponential in n, but unless the ETH is false it cannot be solved in time $2^{o(n)}$ if there is at least a linear number of tuples. CSP is a very important **NP**-complete problem. Apart from its widespread use in AI and elsewhere, it is also an archetypical **NP**-complete problem in the sense that a large number of other **NP**-complete problems can easily be modelled as CSP problems.

Planning is a harder problem than CSP since PSAT, LOP and COP are all **PSPACE**-complete in the general case for SAS$^+$ and PSN, but is likewise a good and natural modelling language for problems in **PSPACE**, and in **NP**. Contrary to the CSP problem, lower bounds for planning have not yet been extensively analysed in a corresponding way, although a few results in this direction have recently appeared in the literature. Aghighi, Bäckström, Jonsson, and Ståhlberg (2016b, Thm. 14) proved that PSAT(PSN) can be solved in time $4^{||\mathbb{P}||^c}$ for all $c \geq 1$ but not for any $c < 1$, unless the ETH is false, where $||\mathbb{P}||$ is the instance size. In this case, the value c is essentially the concept of power indices (Stearns & Hunt, 1990; Stearns, 1994), and these upper and lower bounds are obviously tight for complexity functions of this form. An even tighter lower bound was given as a function of the number of variables, v, instead of the instance size, stating that PSAT(PSN) cannot be solved in time 2^{v^c} for any $c < 1$, unless the ETH is false (Aghighi et al., 2016b, Lemma 13). Also approximation of LOP(PSN) has been studied using the ETH, stating results of the type that approximation within a certain factor has a lower time bound of the form $2^{||\mathbb{P}||^c}$ (Aghighi, Bäckström, Jonsson, & Ståhlberg, 2016a). Furthermore, Aghighi et al. (2016b, Section 5.2) showed that the usual classification of hard problems into **NP**-complete and **PSPACE**-complete may not be very useful with regard to actual time complexity; it is possible to construct two subclassses of PSN such that one of these is **NP**-complete and the other one is **PSPACE**-complete, but both have the property that they can be solved in time $4^{||\mathbb{P}||^c}$ for all $c \geq 1$ but not for any $c < 1$, unless the ETH is false. That is, they belong to distinct complexity classes, but do not differ vastly in actual time complexity. This supports a prediction in the Turing Award lecture by Stearns (1994):

> Although **PSPACE**-completeness is stronger evidence of hardness than **NP**-completeness, there is no reason to believe that **PSPACE**-complete problems are harder in the sense that they require more time.

An obvious conclusion is that it is desirable to study actual time bounds, whenever possible, instead of merely classifying problems into complexity classes.

In this article, we will focus on bounds that are functions of the number of variables (or the number of actions). This gives tighter bounds than using the instance size, which is always larger. For instance, if $||\mathbb{P}|| = v^2$, then an upper bound of 2^v is much tighter than $2^{||\mathbb{P}||} = 2^{v^2}$. We will also focus on tighter bounds in the sense that we will consider exponential functions of the form 2^{cn}, rather than 2^{n^c}. That is, we study functions of the same form as in the definition of the ETH. This allows for much finer distinctions since it is possible that a problem can be solved in time $O(2^{cn})$ for all $c > 0$, yet not in time $O(2^{n^c})$ for any $c < 1$.

1.2 Main Results

The following are the main results in this article, where v is the number of variables of an instance, d is the variable domain size, a is the number of actions and $||\mathbb{P}||$ is the size of the instance.

(1) Bylander (1994) considered subclasses of PSN defined by restricting the number and polarity of variables in the action preconditions and effects, and categorized both Psat and LOP for all these subclasses as either tractable or **NP**-hard.[1] Bäckström and Nebel (1995) made an analogous classification of Psat and LOP for subclasses of SAS$^+$ based on the PUBS restrictions. In Section 4, we strengthen all these **NP**-hardness results by proving that none of these **NP**-hard problems can be solved in time $2^{o(v)}$ unless the ETH is false (or in one case, unless a conjecture about the Set Cover problem is false). These proofs are greatly simplified by first proving a general result (Lemma 5) which implies that if there is a reduction from 3-SAT to a planning problem X such that v is bounded by a linear combination in the number of variables and clauses of the 3-SAT instance, then X cannot be solved in time $2^{o(v)}$ unless the ETH is false.

(2) The lower-bound results in (1) above hold even when a is linear in v. In Section 5 we demonstrate three PSN classes where COP can be solved in time $2^{o(v)}$ if a is subexponential in v. This can be understood as a kind of phase transition between a linear and a sub-linear number of actions, analogous to the phase transition for CSPs demonstrated by De Haan et al. (2015).

(3) Straightforward upper bounds give that Psat, LOP and COP can all be solved in time $d^{2v} \cdot \mathrm{poly}(||\mathbb{P}||)$ for SAS$^+$, and time $4^v \cdot \mathrm{poly}(||\mathbb{P}||)$ for PSN. We show the much tighter upper bound that COP for SAS$^+$ can be solved in time $d^{(1+\varepsilon)v} \cdot \mathrm{poly}(||\mathbb{P}||)$ for all $\varepsilon > 0$, if at most a polynomial number of states in the state-transition graph have an outdegree that is not polynomially bounded in d and v (Theorem 7). Most naturally arising classes of planning problems can be assumed to satisfy this criterion, and it immediately applies to well-known classes like SAS$^+$-U, SAS$^+$-P and even the whole class of instances where a is polynomially bounded in v. We similarily improve the previous lower bounds in (1) by proving that Psat for PSN restricted to unary actions (i.e. at most one defined variable in the effect of an action) cannot be solved in time $2^{(1-\varepsilon)v} \cdot \mathrm{poly}(||\mathbb{P}||)$ for any $\varepsilon > 0$ unless the strong variant of the ETH (the Seth) is false (Theorem 37). This immediately applies also to problems LOP and COP, and to many other severly restricted classes like SAS$^+$-UB. Together, these two results give very tight upper and lower bounds for a very large number of subclasses of PSN (i.e. binary SAS$^+$), stating that Psat, LOP and COP for these can be solved in time $2^{(1+\varepsilon)v} \cdot \mathrm{poly}(||\mathbb{P}||)$ for all $\varepsilon > 0$, but not in time $2^{(1-\varepsilon)v} \cdot \mathrm{poly}(||\mathbb{P}||)$ for any $\varepsilon > 0$ unless the Seth is false. An implication of the new tighter upper bound is that in most cases planning has approximately the same hardness as the SAT problem, with respect to time complexity.

1. Most of the **NP**-hard problems were further classified as either **NP**-complete or **PSPACE**-complete, which is of minor relevance in this article.

(4) We demonstate an alternative way to achieve lower bounds for planning that is not based on the ETH, but on lower bounds for other problems than SAT. Theorem 40 states that LOP for PSN where actions have at most one positive precondition cannot be solved in time $2^{\frac{cv}{2}} \cdot \text{poly}(v)$ unless the GRAPH COLOURABILITY problem can be solved in time $2^{cn} \cdot \text{poly}(n)$. This implies that if we can plan faster than time $2^{\frac{v}{2}} \cdot \text{poly}(v)$, then we also have a better algorithm for GRAPH COLOURABILITY than is currently known. The bound $2^{\frac{v}{2}} \cdot \text{poly}(v)$ is tighter than the ETH-based bound $2^{o(v)}$ in (1), but not as tight as the SETH-based bound $2^{(1-\varepsilon)v} \cdot \text{poly}(||\mathbb{P}||)$ for all $\varepsilon > 0$ in (3). In contrast to the previous types of lower bounds, this result gives a functional connection between the lower bounds of two problems.

(5) All currently known upper time bounds for any **NP**-hard problem are exponential, and the best bounds usually also require exponential space. There is, however, a trade-off between time and space; we can always improve the space bound at the cost of using more time. This is an important issue since planners often run out of memory in practice. We prove that COP for PSN with only positive action effects can be solved in time $F(v) \cdot \text{poly}(||\mathbb{P}||)$ and polynomial space (Theorem 45), where $F(v)$ is the vth Fubini number (i.e. the number of ordered partitions of the variables) which is approximately $2^{\Theta(v \log v)}$. We additionally prove a better bound for LOP when also the preconditions are positive, showing that this problem can be solved in time $v! \cdot \text{poly}(||\mathbb{P}||)$ and polynomial space (Theorem 50). These results are not proven by devising new algorithms, but by using ordinary depth-first search (DFS) and derive improved upper bounds for these two restricted cases of planning. While DFS is usually considered inefficient and avoided, it is at advantage here since it is one of the few search algorithms that can solve these problems in polynomial space.

1.3 Overview of the Article

The remainder of the article is structured as follows. In Section 2, we first define the planning languages and planning problems we consider, and then define the exponential-time hypothesis. In Section 3 we first prove some straightforward upper time bounds for planning in SAS$^+$ and PSN. Then we prove the tighter upper bound mentioned in (3) above. In Section 4, we prove the results mentioned in (1), that none of the previously reported **NP**-hard planning problems can be solved in subexponential time, unless the ETH is false. As a spin-off result, we also settle an open question and prove **NP**-hardness for those PUBS restrictions for SAS$^+$ that have remained unclassified in the literature. In Section 5, we show the phase-transition results mentioned in (2), demonstrating three cases where planning in subexponential time is possible if the number of actions is subexponential. In Section 6, we first prove the tighter lower bound for PSAT based on the SETH, mentioned in (4). Then we prove the lower-bound result for LOP that is conditional on the lower bound for GRAPH COLOURABILITY, as mentioned in (4). In Section 7 we discuss the trade-off between time and space. We first show that even COP for SAS$^+$ can be solved in polynomial space, but at the cost of using considerably more time than standard methods do, and then prove the two special cases in (5). The paper ends with a discussion in Section 8.

This article is based on a previous conference paper (Bäckström & Jonsson, 2016), but is considerably revised and extended.

2. Preliminaries

In this section, we first define the SAS$^+$ and PSN planning languages, together with the commonly used restrictions that we will consider for definining subclasses. We also define the different planning problems we will consider. After that, we define the boolean satisfiability problem and the exponential-time hypothesis. However, we start with a few remarks on notation. For a string, set or sequence X of objects, we write $|X|$ to denote the cardinality (i.e. the number of objects) of X and we write $||X||$ to denote the size (i.e. the number of bits) of the representation of X. We use log to denote logarithms of base 2, $\log_b$ to denote logarithms of base b and ln to denote natural logarithms, i.e. base e.

2.1 Planning

We will first define the general case, the SAS$^+$ language. Then we define the PSN language for the special case of binary variables and the further specialisation to monotone planning. Finally, we define the various planning problems we will study in this article.

2.1.1 SAS$^+$

In the general case, we will use the SAS$^+$ planning language (Bäckström & Klein, 1991; Bäckström & Nebel, 1995), which uses state variables with arbitrary finite domain.

Let $V = \{v_1, \ldots, v_n\}$ be a finite set of *variables*, over some finite domain D. Without losing generality, we will assume that D is of the form $D = \{0, \ldots, k\}$ for some positive integer $k \geq 1$.[2] The variables and the domain together induce a *state space* $S = D^n$, where each member $s \in S$ is called a *(total) state* and can be viewed as a total function that specifies a value in D for each $v \in V$. A *partial state* may leave the value undefined for some (or all) variables, and is thus a partial function. Note that a total state is also a partial state. The set of variables with a defined value in a partial state s is denoted $\mathrm{vars}(s)$, and $s[v]$ denotes the value of v in s for each $v \in \mathrm{vars}(s)$. Let s and t be partial states. Then s is *satisfied by* t, denoted $s \sqsubseteq t$, if $s[v] = t[v]$ for all $v \in \mathrm{vars}(s)$. The *update operator* $\ltimes$ is defined such that if s is a total state and t is a partial state, then $s \ltimes t = r$, where r is a total state satisfying that $r[v] = t[v]$ for all $v \in \mathrm{vars}(t)$ and $r[v] = s[v]$ for all $v \in V \setminus \mathrm{vars}(t)$.

A *planning instance* $\mathbb{P} = \langle V, D, A, s_I, s_G \rangle$ has a set V of variables with domain D, a set A of *actions*, a total *initial state* s_I and a partial *goal state* s_G. Each action $a \in A$ has a *precondition* $\mathrm{pre}(a)$ and an *effect* $\mathrm{eff}(a)$, which are both partial states. Let $a \in A$ and $s \in S$. Then a is *valid in* s if $\mathrm{pre}(a) \sqsubseteq s$ and the *result of* a *in* s is the state $s \ltimes \mathrm{eff}(a)$. For all states $s, t \in S$ and actions $a \in A$, we say that a is *from* s *to* t if $\mathrm{pre}(a) \sqsubseteq s$ and $t = s \ltimes \mathrm{eff}(a)$. Let $s_0, s_\ell \in S$ and let $\omega = a_1, \ldots, a_\ell$ be a sequence of actions. Then ω is a *plan from* s_0 *to* s_ℓ if either (1) $\omega = \langle\rangle$ and $\ell = 0$ or (2) there are states $s_1, \ldots, s_{\ell-1} \in S$ such that for all i ($1 \leq i \leq \ell$), a_i is valid in s_{i-1} and s_i is the result of a_i in s_{i-1}. The sequence $s_0, \ldots, s_\ell$ is referred to as the *state sequence of* ω. Furthermore, ω is a *plan (i.e. a solution) for* $\mathbb{P}$ if it is a plan from s_I to some state s such that $s_G \sqsubseteq s$. The *length* of an action sequence $\omega = a_1, \ldots, a_\ell$ is $|\omega| = \ell$.

2. It is common, and often practical, to assume a separate domain for each variable. However, for the purpose of complexity analysis it is both simpler and sufficient to use one single domain, since it is usually the size of the largest domain that matters.

We will sometimes consider *action costs* by defining a *cost function* $c : A \to \mathbb{D}$, where $\mathbb{D}$ is some numeric domain. This function is implicitly extended to action sets such that $c(A') = \sum_{a \in A'} c(a)$ for all $A' \subseteq A$ and to action sequences such that if $\omega = a_1, \ldots, a_\ell$ is a sequence of actions over A, then $c(\omega) = \sum_{i=1}^{\ell} c(a_i)$.

Let $\mathbb{P}$ be a SAS$^+$ instance. A plan ω for $\mathbb{P}$ is a *length-optimal plan* (or a *shortest* plan) for $\mathbb{P}$ if there is no other plan ω' for $\mathbb{P}$ such that $|\omega'| < |\omega|$. Let c be a cost function for $\mathbb{P}$. A plan ω for $\mathbb{P}$ is a *cost-optimal plan* (or a *cheapest* plan) for $\mathbb{P}$ and c if there is no other plan ω' for $\mathbb{P}$ such that $c(\omega') < c(\omega)$. Furthermore, ω is a *shortest cost-optimal plan* for $\mathbb{P}$ and c if there is no other cost-optimal plan ω' for $\mathbb{P}$ and c such that $|\omega'| < |\omega|$ holds. Note that it is important to distinguish between a shortest cost-optimal plan and any cost-optimal plan, since there can be two plans with the same cost that differ vastly in length. If there are zero-cost loops in the state-transition graph, then the cost-optimal plans may be arbitrarily long.

The *state-transition graph* for a SAS$^+$ instance $\mathbb{P} = \langle V, D, A, s_I, s_G \rangle$ is the directed graph $G = \langle S, E \rangle$, where S is the state space and E contains an edge $\langle s, t \rangle$ for all states $s, t \in S$ such that there is some action $a \in A$ from s to t. The edges may be labelled with additional information, like the cost of a cheapest action from s to t.

We will often use the notation $a : P \Rightarrow E$ to define an action a with precondition P and effect E, where P and E are written as sets of elements of the form $v = d$. For instance, $a : \{v_1 = 0, v_2 = 1\} \Rightarrow \{v_1 = 1\}$ defines an action a that has as precondition that $v_1 = 0$ and $v_2 = 1$ and has the effect of setting v_1 to 1.

It is common to consider subclasses defined by combinations of the following four restrictions on instances (Bäckström & Klein, 1991; Bäckström & Nebel, 1995).

- **P (post-unique):** For all $v \in V$ and all $x \in D$, there is at most one $a \in A$ such that $\mathrm{eff}(a)[v] = x$.
- **U (unary):** For each $a \in A$, $|\mathrm{vars}(\mathrm{eff}(a))| = 1$.
- **B (binary):** $|D| = 2$.
- **S (single-valued):** For all $a, b \in A$ and $v \in V$, if $v \in \mathrm{vars}(\mathrm{pre}(a)) \cap \mathrm{vars}(\mathrm{pre}(b))$ and $v \notin \mathrm{vars}(\mathrm{eff}(a)) \cup \mathrm{vars}(\mathrm{eff}(b))$, then $\mathrm{pre}(a)[v] = \mathrm{pre}(b)[v]$.

Combinations of these restrictions are written by juxtaposing the corresponding letters, eg. SAS$^+$-PUB is the class of all SAS$^+$ instances that are post-unique, unary and binary.

2.1.2 PSN and Monotone PSN

Most of our results, in particular the lower-bound results, only make use of binary variables. In these cases, it is often clearer and more convenient to use Propositional Strips with Negative goals (PSN) (Bylander, 1994), which can be viewed as a different way to define SAS$^+$ restricted to binary variables. Let V be a set of binary *variables*, i.e. propositional atoms. For any set $V' \subseteq V$, the set of *literals* over V' is $L(V) = \{v, \overline{v} \mid v \in V\}$. A *total state* s over V is a subset $s \subseteq V$, where a variable v is true in s if and only if $v \in s$. The *space of total states* over V is $S = 2^V$. A *partial state* t over V is a consistent subset $t \subseteq L(V)$, i.e. it does not contain both v and $\overline{v}$ for any $v \in V$. A variable v is true in t if $v \in t$, false if $\overline{v} \in t$ and undefined if $t \cap \{v, \overline{v}\} = \varnothing$. We also define $t^+ = \{v \in V \mid v \in t\}$ and $t^- = \{v \in V \mid \overline{v} \in t\}$. Let t be a partial state and s a total state. Then t is *satisfied* in s, denoted $t \sqsubseteq s$ if both $t^+ \subseteq s$ and $t^- \cap s = \varnothing$. The $\ltimes$ *operator* is defined as $s \ltimes t = (s \setminus t^-) \cup t^+$. Finally, $\mathrm{vars}(t) = \{v \mid v \in t \text{ or } \overline{v} \in t\}$.

A PSN *instance* is a tuple $\mathbb{P} = \langle V, A, s_I, s_G \rangle$ where V is a set of variables, A is a set of *actions*, the *initial state* s_I is a total state over V and the *goal* s_G is a partial state over V. Note that we omit the variable domain, since it is always binary for PSN. Each action a in A has a precondition $\mathrm{pre}(a)$ and an effect $\mathrm{eff}(a)$, which are both partial states. For all total states s, t over V and all $a \in A$, a is *from s to t* if both (1) $\mathrm{pre}(a) \sqsubseteq s$ and (2) $t = s \ltimes \mathrm{eff}(a)$. A sequence $\omega = a_1, \ldots, a_\ell$ of actions in A is a *plan* from a state s_0 to a state s_ℓ if either (1) $\omega = \langle\rangle$ and $s_0 = s_\ell$ or (2) there are total states $s_1, \ldots, s_{\ell-1}$ such that a_i is from s_{i-1} to s_i for all i $(1 \leq i \leq \ell)$. The sequence $s_0, \ldots, s_\ell$ is the *state sequence* of ω. A solution for $\mathbb{P}$ is a plan from s_I to some total state s such that $s_G \sqsubseteq s$. A solution for $\mathbb{P}$ is called a *plan* for $\mathbb{P}$. Analogously to the SAS$^+$ case, we will often use the notation $a : P \Rightarrow E$, to define an action, where P and E are partial PSN states. For instance, $a : \{\overline{v_1}, v_2\} \Rightarrow \{v_1\}$ defines an action a with precondition that v_1 is false and v_2 true and with the effect of setting v_1 to true.

Following Bylander (1994), we will consider PSN subclasses based on the number and polarity of the literals in the preconditions and effects of actions. For instance, PSN_1^{2+} denotes the class of PSN instances where the actions have at most two positive literals in the precondition and one literal in the effect. We use $*$ to denote an unrestricted number of literals, i.e. PSN_1^* allows any number of literals in the preconditions but only one literal in the effects. Note that $\mathrm{PSN}_*^* = \mathrm{PSN}$. This type of restrictions is not entirely independent from the ones for SAS$^+$. We have already noted that PSN is actually the class SAS$^+$-B, so PSN_*^{*+} is a subclass of SAS$^+$-BS and PSN_1^* is the class SAS$^+$-UB.

Let $\sigma = s_0, \ldots, s_\ell$ be a sequence of states. Then σ is *monotone* if $s_{i-1} \subseteq s_i$ for all i $(1 \leq i \leq \ell)$ and σ is *strictly monotone* if $s_{i-1} \subset s_i$ for all i $(1 \leq i \leq \ell)$. We will refer to a PSN instance as *monotone* if no action has any negative effects, i.e. PSN_{*+}^* is the class of all monotone instances. It obviously holds for all monotone instances that $s \ltimes t = s \cup t$, so the following properties are immediate.

Proposition 1. *Let $\mathbb{P} = \langle V, A, s_I, s_G \rangle$ be a monotone* PSN *instance and let ω be an action sequence over A. Then:*

1. *If ω is a plan for $\mathbb{P}$, then ω has a monotone state sequence.*
2. *If ω is a length-optimal or shortest cost-optimal plan for $\mathbb{P}$, then ω has a strictly monotone state sequence.*
3. *If ω is a length-optimal or shortest cost-optimal plan for $\mathbb{P}$, then no action in A occurs more than once in ω.*

Monotone PSN is interesting in many ways, despite being quite restrictive. Firstly, it is **NP**-complete to find length-optimal plans for monotone instances, even if the actions have only only one positive precondition and one positive effect (Bylander, 1994, Corollary 4.3). Secondly, it is the basis for many successful heuristics, like the *delete-relaxation heuristic h^+*.

2.1.3 Planning Problems

For every class C of SAS$^+$ instances, we define the following three decision problems, where $\mathbb{Q}_0$ denotes the non-negative rational numbers.

Plan Satisfiability $(\textsc{Psat}(C))$
Instance: An instance $\mathbb{P} = \langle V, D, A, s_I, s_G \rangle$ in C.
Question: Does $\mathbb{P}$ have a plan?

Length-optimal Planning $(\text{LOP}(C))$
Instance: An instance $\mathbb{P} = \langle V, D, A, s_I, s_G \rangle$ in C and a non-negative integer k.
Question: Does $\mathbb{P}$ have a plan ω of length $|\omega| \leq k$?

Cost-optimal Planning $(\text{COP}(C))$
Instance: An instance $\mathbb{P} = \langle V, D, A, s_I, s_G \rangle$ in C, a *cost function* $c : A \to \mathbb{Q}_0$ and a value $k \in \mathbb{Q}_0$.
Question: Does $\mathbb{P}$ have a plan ω of cost $c(\omega) \leq k$?

Note that these problems are related such that a lower bound for $\textsc{Psat}(C)$ is also a lower bound for $\text{LOP}(C)$ and a lower bound for $\text{LOP}(C)$ is also a lower bound for $\text{COP}(C)$. Analogously, an upper bound for $\text{COP}(C)$ is also an upper bound for $\text{LOP}(C)$ and an upper bound for $\text{LOP}(C)$ is also an upper bound for $\textsc{Psat}(C)$. Hence, it is often not necessary to provide separate results for all three problems. Also note that LOP can be viewed as the special case of COP where all actions have cost 1.

One may also consider the corrsponding generation problems. For instance, length-optimal plan generation either returns a plan of length k at most, or rejects if there is no such plan. It may alternatively be defined without a length parameter and always return a shortest possible plan, whenever there is a plan. We will state all complexity results using decision problems. This strengthens the lower-bound results, since it is possible that it is easier to decide if there is a plan than to actually generate a plan. Furthermore, all our upper-bound results are constructive, i.e. the proofs are based on describing some algorithm for actually generating a plan with specified properties.

We will finally briefly discuss the issue of numeric representations for costs. Since a shortest plan can be of length $d^{|V|}-1$ in the worst case, we cannot assume that a fixed-size representation of numbers is sufficient even for solving LOP. Using floating-point numbers does not resolve this issue, since we may lose precision due to rounding. While such a loss of precision may, perhaps, be acceptable in some practical cases, it does not work for complexity analysis; we no longer solve the COP problem, but some kind of ill-defined approximation of it. We will only consider the domain $\mathbb{Q}_0$ of non-negative rational costs in this paper. Firstly, negative action costs are rare in the literature, since such costs can cause anomalies of the kind that we can find a plan with arbitrary negative cost, if there are cycles with negative cost in the state-transition graph. Secondly, the restriction to the rationals, rather than the reals, is motived by the fact that the rational numbers have finite representations, while the reals do not, i.e. the reals cannot be properly used in practice. Furthermore, the rationals are closed under addition and multiplication so they are safe to use for cost-optimal planning. They are also safe in the sense that they cannot blow up the time and space requirement more than a polynomial factor in the instance size. Let $d_1, \ldots, d_n$ be all different denominators occuring in the numbers in $\mathbb{P}$. Then $||d_1|| + \ldots + ||d_n|| < ||\mathbb{P}||$. Each number may have to be multiplied with the factor $d_1 \cdot \ldots \cdot d_n$, or less, in summations and comparisons. Since $||d_1 \cdot \ldots \cdot d_n|| \leq ||d_1|| + \ldots + ||d_n|| \leq ||\mathbb{P}||$ and $||k|| \leq ||\mathbb{P}||$, it follows that we can add the action costs in a plan and compare with k in polynomial time in $||\mathbb{P}||$. Even though we will often measure the complexity as a function of the number of variables, or actions, we will almost always also have a factor poly($||\mathbb{P}||$) in the cases we consider COP, which makes it safe to use rational costs. Note, however, that the choice of numeric domain can be crucial when using other types of complexity analysis, such as parameterised complexity. For instance, Aghighi and Bäckström (2015, 2016) demonstrate differences in complexity for domains such as the positive integers, the non-negative integers, the postive rationals, the rationals greater than or equal to one and the non-negative rationals, by considering different parameterisations.

2.2 Satisfiability and the Exponential Time Hypothesis

A boolean formula $\mathbb{F}$ over a set $X = \{x_1, \ldots, x_n\}$ of variables is on *conjunctive normal form (CNF)* if it is a conjunction of disjunctions of literals, i.e. $\mathbb{F}$ is of the form $\mathbb{F} = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ where $c_j = l^j_1 \vee l^j_2 \vee \cdots \vee l^j_{m_j}$ and $l^j_1, l^j_2, \ldots, l^j_{m_j}$ are literals over X. The disjunctions are often referred to as *clauses*. An assignment of truth values to the variables in X *satisfies* $\mathbb{F}$ if $\mathbb{F}$ evaluates to true for this assignment. The formula $\mathbb{F}$ is further of k-CNF form if no clause has more than k literals. We will use several different variants of the Boolean Satisfiability problem (SAT) during the course of this article. In its basic form it is defined as follows.

> Boolean Satisfiability (SAT)
> *Instance:* A boolean formula $\mathbb{F}$ on conjunctive normal form.
> *Question:* Does $\mathbb{F}$ have a satisfying assignment?

We require, without loss of generality, that repeated clauses are not allowed and that no empty clauses appear. Note that the definition of SAT instances then implies that there are no unused variables, i.e. every variable appears in at least one clause. The problem k-SAT, where $k \geq 1$, is the SAT problem restricted to k-CNF formulae. The SAT problem is **NP**-complete and so is the k-SAT problem when $k \geq 3$ (Garey & Johnson, 1979, problem LO1). The UNSAT problem is the complementary problem to SAT, i.e. it takes a CNF formula $\mathbb{F}$ and asks if $\mathbb{F}$ does not have a satisfying assignment, and k-UNSAT is analogously the complement of k-SAT. The UNSAT problem is co**NP**-complete and so is the k-UNSAT problem for all $k \geq 3$. We will use n for the number of variables and m for the number of clauses in CNF formulae.

A straightforward upper time bound for SAT is that it can be solved in time $O\big(2^n \cdot \text{poly}(||\mathbb{F}||)\big)$, since we can enumerate all 2^n assignments and test them. We can also give a more precise characterisation of the lower bound of k-SAT, than merely saying that it is an **NP**-complete problem, by using the *Exponential Time Hypothesis* (ETH) by Impagliazzo and Paturi (2001). The ETH and its strong variant, the Seth, are defined as follows.

Definition 2. *For all constant integers $k \geq 3$, let s_k be the infimum of all real numbers δ such that k-SAT can be solved in time $O(2^{\delta n})$. The* ETH *says that $s_k > 0$ for all $k \geq 3$. The* Seth *additionally says that* $\lim_{k \to \infty} s_k = 1$.

We will briefly discuss Seth at the end of this section. It is also common to use the following, slightly weaker, definition of the ETH.

Definition 3. *The* ETH *says that* 3-SAT *cannot be solved in time* $2^{o(n)}$.

The differences between these two definitions of the ETH are very small and subtle[3] (cf. Flum & Grohe, 2006, Section 16). We will mainly use the second variant, since it often makes both theorems and proofs simpler, but the first variant is still needed for the definition of the Seth. The ETH is a quite strong assumption that allows for defining a theory similar to the one of **NP**-completeness. There is a large number of **NP**-complete problems that form a completeness class in the sense that either all of them can be solved in subexponential time, or none of them can (Impagliazzo, Paturi, & Zane, 2001). Since the ETH refers to actual deterministic time bounds it is possible to swap the answers, i.e. if we could solve k-SAT in time $O\big(f(n)\big)$ for some function f, then we could also solve k-UNSAT in time $O\big(f(n)\big)$, and vice versa. Hence, both ETH and Seth can equivalently be defined in terms of the k-UNSAT problem.

A general CNF formula with n variables can have up to 3^n unique non-tautological clauses, while a k-CNF formula can have at most $\Theta(n^k)$ clauses. However, the hardness of the SAT and k-SAT problems does not depend on having a large number of clauses, as the following lemma demonstrates.

Lemma 4. *(De Haan et al., 2015, Lemma 1) k-SAT ($k \geq 3$) is solvable in time $2^{o(n)}$ if and only if k-SAT with a linear number of clauses and in which the number of occurences of each variable is at most 3 is solvable in time $2^{o(n)}$.*

This lemma is an adaptation of the *sparsification* lemma by Impagliazzo et al. (2001) and it is essential for many of our results. However, in order to simplify the proofs, we will use it indirectly by applying the following lemma. In our applications of it, problem X will be either Psat(C) or LOP(C) for some class C of planning instances and the measure x will be either the number of variables, v, or the number of actions, a.

Lemma 5 (Linear combination lemma)**.** *Let X be some decision problem and let x be some measure of instances of X. Let $\lambda, \mu > 0$ be constants. Let ρ be a polynomial reduction from* 3-SAT *to X with the property that if $\mathbb{F}$ is a 3-CNF formula with n variables and m clauses, then $\rho(\mathbb{F})$ has measure $x \leq \lambda n + \mu m$ for large n and m. Then it holds that X cannot be solved in time $2^{o(x)}$ unless the* ETH *is false.*

Proof. Assume constants λ and μ and a reduction ρ as stated in the lemma. Let S_d denote the set of 3-CNF formulae where $m \leq dn$ and d is chosen such that satisfiability for S_d cannot be solved in time $2^{o(n)}$ if the ETH is true. Such a d must exist according to Lemma 4. Let $\mathbb{F}$ be an arbitrary 3-CNF formula in S_d with n variables. Then $\mathbb{F}$ has $m \leq dn$ clauses, so instance $\rho(\mathbb{F})$ has measure $x \leq \lambda n + \mu m \leq \lambda n + \mu dn = (\lambda + \mu d)n$. Let $\kappa = \lambda + \mu d$, i.e. $x \leq \kappa n$. Since $||\mathbb{F}||$ is polynomial in n for all formulae $\mathbb{F}$ in S_d, we can compute the reduction ρ in polynomial time in n.

Suppose problem X can be solved in time $2^{o(x)}$. Then X can be solved in time 2^{cx} for all $c > 0$ and large x. It follows that we can solve satisfiability for S_d in time $\text{poly}(n) + 2^{c\kappa n} \leq$

3. If there were an algorithm solving 3-SAT in time $2^{o(n)}$, then we would have $s_3 = 0$, so both definitions agree on this case. However, Definition 3 does not rule out the possibility that there is a sequence of algorithms $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \ldots$ with running times $O(2^{\delta_i n})$ for a corresponding sequence of values $\delta_1, \delta_2, \delta_3, \ldots$ that tends to 0, although it would be practically impossible to choose the correct algorithm to run from the sequence.

$2^{(c\kappa+\varepsilon)n}$, for all $c, \varepsilon > 0$ and large n. It thus holds for all $c' > 0$ that satisfiability for S_d can be solved in time $2^{c'n}$ for large n by chosing $c > 0$ and $\varepsilon > 0$ such that $c\kappa + \varepsilon \leq c'$. However, it then follows from the definition of S_d that the ETH must be false. □

This lemma also holds if $x \leq \lambda n + \mu m + \nu$, for some constant ν, since we can always choose slightly larger values λ' and μ' such that $\lambda n + \mu m + \nu \leq \lambda' n + \mu' m$ for large n and m. The lemma obviously also holds if ρ is a reduction from 3-UNSAT to X. Note that this lemma does not depend on the restricted number of variable occurences in Lemma 4.

Even if the ETH is true, there is some constant c such that SAT can be solved in time $O(2^{\delta n})$ for all $\delta \geq c$, but the ETH does not say anything about the value of c. Tighter bounds can be achieved by using the Seth instead of the ETH. If the Seth holds, we know that SAT cannot be solved in time $O(2^{\delta n})$ for any $\delta < 1$, so we get tight upper and lower bounds since we also know that SAT can be solved in time $2^n \cdot \text{poly}(||\mathbb{F}||)$. By assuming the Seth we get a better precision in the results, but at the expense of using a stronger assumption than the ETH.

3. Upper Bounds

We first briefly discuss how the size of an instance is influenced by its various parameters, i.e. number of variables, domain size and number of actions. Then we derive straightforward upper time bounds for planning in SAS$^+$, PSN and monotone PSN. After that we show that much better upper bounds can be achieved under restrictions that most natural classes of planning instances can be assumed to satisfy. We end the section by discussing different algorithms for deriving upper-bound results.

3.1 Instance Size

We will henceforth use the notation that v is the number of variables in a planning instance, d is the domain size of the variables and a is the number of actions. Before discussing upper bounds for planning, we first need to briefly discuss instance sizes and action sets. If we have v variables, with domain size d, then there are $(d+1)^v$ different partial states. Hence, there are $(d+1)^v$ different possible preconditions and $(d+1)^v - 1$ different possible effects (assuming every action must have a defined effect on at least one variable). That is, the maximum number of distinct actions is approximately $(d+1)^{2v}$. Although it is often reasonable to assume that the number of actions, a, is polynomially bounded in the number of variables, we cannot generally make this assumption. Most of the complexity results in this article are based on some superpolynomial function like d^{cv} or $2^{o(v)}$. There will usually also be some polynomial factor of the form $\text{poly}(||\mathbb{P}||)$. Although the instance size $||\mathbb{P}||$ is polynomially bounded in v and a, the factor $\text{poly}(||\mathbb{P}||)$ cannot generally be assumed dominated by an exponential function in v, since a can be exponential in v. In most cases it is, thus, necessary to state this polynomial factor explicitly. For instance, it is often necessary to write $O\big(2^v \cdot \text{poly}(||\mathbb{P}||)\big)$ instead of $O(2^v)$. On the other hand, we may drop the O notation and write only $2^v \cdot \text{poly}(||\mathbb{P}||)$, since we can always choose an appropriate polynomial. Note, that this does not generally hold when we use explicit polynomials, as in expressions of the form $O\big(2^v \cdot p(||\mathbb{P}||)\big)$ for some specific polynomial p.

3.2 General Upper Bounds

We will first derive straightforward upper bounds for planning. Let $\mathbb{P} = \langle V, D, A, s_I, s_G \rangle$ be a SAS$^+$ instance and let $G = \langle S, E \rangle$ be the state-transition graph of $\mathbb{P}$. Then the state space S has $|S| = d^v$ states. In the worst case, each state can have an edge to every state, including itself, so we get an upper bound on the number of edges of $|E| \leq d^v \cdot d^v = d^{2v}$. A straightforward way to solve COP is to first construct G and then apply Dijkstra's algorithm. Aghighi et al. (2016b, Section 2.2) showed that G can be constructed in time $O\big(|S| \cdot \text{poly}(||\mathbb{P}||)\big)$. Note that this figure holds also without their assumption that $d = 2$ and also if we additionally choose a cheapest action for each edge. Fredman and Tarjan's (1987) implementation of Dijkstra's algorithm solves the single source shortest path problem for G in time $O(|E| + |S| \log |S|)$ and space $O(|E| + |S| \log |S|)$. Their implementation assumes a fixed-size representation of the costs, which is not sufficient for our purposes (recall that we must use exact arithmetics). However, they use no special tricks for representing the costs, so we can use an arbitrary-size representation instead, and use the factor poly($||\mathbb{P}||$) to cover for the overhead. In total, this takes time $O\big((|E| + |S| \log |S|) \cdot \text{poly}(||\mathbb{P}||)\big)$ for both constructing G and finding a length- or cost-optimal plan. Using the worst-case upper bound $|E| = d^{2v}$, it follows that we can solve COP(SAS$^+$), and even generate a cost-optimal plan, in time $(d^{2v} + d^v \log d^v) \cdot \text{poly}(||\mathbb{P}||) = d^{2v} \cdot \text{poly}(||\mathbb{P}||)$. Note that this time bound automatically applies also to LOP(SAS$^+$) and Psat(SAS$^+$), since these problems are special cases of COP(SAS$^+$). Heuristic search algorithms may be preferrable in many practical cases, but they give no advantage in the worst case, when they also have to process on the order of $|E|$ edges and store up to $|S|$ states explicitly. We prefer to use Dijkstra's algorithm here since it is more well studied and better time bounds are known for it. This choice will be further discussed at the end of this section. In the special case of PSN we have $d = 2$, so the state space is of size 2^v and can have up to $2^{2v} = 4^v$ edges, i.e. we can solve COP(PSN) in time $4^v \cdot \text{poly}(||\mathbb{P}||)$. We consider all of these results straightforward, and we have previously stated the results for the PSN case (Aghighi et al., 2016b).

We then proceed to monotone PSN, i.e. the class PSN$^*_{*+}$. Then there can only be an edge from a state t to a state s if $t \subseteq s$, since $s \ltimes \text{eff}(a) = s \cup \text{eff}(a)$ in this case. For each i $(0 \leq i \leq v)$, there are $\binom{v}{i}$ states of size i. For each state s, there are 2^i subsets, including s itself, so there can be at most 2^i incoming edges, including a loop at s. Hence, a safe upper bound on the number of edges is

$$|E| \leq \sum_{i=0}^{v} \binom{v}{i} 2^i.$$

Using the binomial formula

$$(a+b)^v = \sum_{i=0}^{v} \binom{v}{i} a^{v-i} b^i$$

and setting $a = 1$ and $b = 2$ we get

$$|E| \leq \sum_{i=0}^{v} \binom{v}{i} 2^i = \sum_{i=0}^{v} \binom{v}{i} 1^{v-i} 2^i = (1+2)^v = 3^v$$

i.e. $|E| \leq 3^v = 2^{v \log 3}$. Also note that this state space is isomorphic to the lattice of the subset relation over V. We thus get that

$$|E| + |S| \log |S| \leq 3^v + 2^v \log 2^v \in O(3^v).$$

Using Dijkstra's algorithm, we can thus achieve the following upper bounds.

Proposition 6. *The following upper bounds hold:*

1. $\mathrm{COP}(\mathrm{SAS}^+)$ *can be solved in time* $d^{2v} \cdot poly(||\mathbb{P}||)$ *and space* $d^{2v} \cdot poly(||\mathbb{P}||)$.
2. $\mathrm{COP}(\mathrm{PSN})$ *can be solved in time* $4^v \cdot poly(||\mathbb{P}||)$ *and space* $4^v \cdot poly(||\mathbb{P}||)$.
3. $\mathrm{COP}(\mathrm{PSN}^*_{*+})$ *can be solved in time* $3^v \cdot poly(||\mathbb{P}||)$ *and space* $3^v \cdot poly(||\mathbb{P}||)$.

3.3 Tighter Upper Bounds for Restricted Cases

The previous bounds are based on the worst case where the state-transition graph is the complete graph. If we instead consider graphs with some maximum outdegree δ, then we get $|E| \leq d^v \cdot \delta$. We obviously have $\delta \leq a$, i.e. the number of actions is an upper bound on the outdegree. The opposite does not hold, though; the maximum outdegree only gives the trivial bound that $a \leq d^v \cdot \delta$. It is a reasonable assumption in many practical cases that the maximum number of actions that can be applied in a state is restricted by some polynomial in v and d. Indeed, this is always the case when the instance size is polynomially restricted in v and d. This assumption greatly improves the upper time bound for planning. However, we can make the assumption even more general by allowing a polynomial number of exceptions to this restriction, as the following theorem demonstrates. Note that this theorem does not assume that the domain size, d, is constant.

Theorem 7. *Let p and q be polynomials. Let C be a class of SAS$^+$ instances such that at most $q(dv)$ states in the state-transition graph have more than $p(dv)$ outgoing edges. Then* $\mathrm{COP}(C)$ *can be solved in time* $d^{(1+\varepsilon)v} \cdot poly(||\mathbb{P}||)$ *for all* $\varepsilon > 0$.

Proof. The state space has d^v states, so the maximum outdegree is d^v. At most $q(dv)$ states can have outdegree larger than $p(dv)$, so these states can contribute at most $q(dv)d^v$ edges. The remaining $d^v - q(dv)$ states can have outdegree $p(dv)$, at most, so they contribute at most $\big(d^v - q(dv)\big)p(dv) \leq d^v p(dv)$ edges. In total, we get $|E| \leq q(dv)d^v + d^v p(dv) = \big(p(dv) + q(dv)\big)d^v$. Let $c \geq 1$ and $k \geq 1$ be constants such that $p(dv) + q(dv) \leq c(dv)^k$, i.e. $|E| \leq c(dv)^k d^v$. As previously noted, Fredman and Tarjan's (1987) variant of Dijkstra's algorithm can be implemented to run in time $(|E| + |S| \log |S|) \cdot \mathrm{poly}(||\mathbb{P}||)$ for arbitrary costs. We first note that

$$|S| \log |S| = d^v \log d^v = d^v \log 2^{v \log d} = d^v v \log d \leq (dv)d^v,$$

so we get that

$$\begin{aligned} |E| + |S| \log |S| &\leq c(dv)^k d^v + (dv)d^v \leq 2c(dv)^k d^v = 2cd^{k+v}v^k \\ &= 2cd^{k+v}d^{k \log_d v} = 2cd^{v+k \log_d v + k} \leq 2cd^{(1+\varepsilon)v} \end{aligned}$$

for all $\varepsilon > 0$ and large v. Since we can build the state-transition graph in time $|S| \cdot \mathrm{poly}(||\mathbb{P}||) = d^v \cdot \mathrm{poly}(||\mathbb{P}||)$, it follows that we can solve $\mathrm{COP}(C)$ in time $d^{(1+\varepsilon)v} \cdot \mathrm{poly}(||\mathbb{P}||)$ for all $\varepsilon > 0$. □

This theorem is a considerable improvement over the previous general upper bounds, improving the exponential factor by $d^{(1-\varepsilon)v}$. The following corollary is immediate.

Corollary 8. *Let p be a polynomial and let C be the class of all SAS$^+$ instances such that $a \leq p(dv)$. Then* $\mathrm{COP}(C)$ *can be solved in time $d^{(1+\varepsilon)v} \cdot poly(||\mathbb{P}||)$ for all $\varepsilon > 0$.*

Planning classes not satisfying this restriction are rare in the literature, so it seems reasonable to assume that most, or all, 'natural' planning problems can be solved in time $d^{(1+\varepsilon)v} \cdot \mathrm{poly}(||\mathbb{P}||)$ for all $\varepsilon > 0$. For instance, any natural generalization of a typical logistics scenario to arbitrary size would have the number of actions bounded by some polynomial in the numbers of packages, trucks and locations.

A number of other results also follow from Theorem 7.

Corollary 9. *Let $k > 0$ be a constant. Let C_k be the class of all SAS$^+$ instances $\mathbb{P}$ where no action has more than k defined effects. Then* $\mathrm{COP}(C_k)$ *can be solved in time $d^{(1+\varepsilon)v} \cdot poly(||\mathbb{P}||)$ for all $\varepsilon > 0$.*

Proof. For each state s, we can simultaneously change at most k variables. There are $\binom{v}{k}$ choices of k variables. For each variable, there are at most $d - 1$ choices of new value so there are less than d^k choices of value combinations for the k variables. Since we may also change fewer than k variables, the maximum number of outgoing edges from s is less than

$$\sum_{i=1}^{k} \binom{v}{i} d^i < \sum_{i=1}^{k} v^i d^i = \sum_{i=1}^{k} (dv)^i < k(dv)^k,$$

which is polynomial in dv. Hence, Theorem 7 applies to C_k. □

Corollary 10. $\mathrm{COP}(\mathrm{SAS}^+\text{-U})$ *can be solved in time $d^{(1+\varepsilon)v} \cdot poly(||\mathbb{P}||)$ for all $\varepsilon > 0$.*

Proof. Immediate since SAS$^+$-U is the class C_1 in Corollary 9. □

Corollary 11. $\mathrm{COP}(\mathrm{PSN}^*_k)$ *can be solved in time $2^{(1+\varepsilon)v} \cdot poly(||\mathbb{P}||)$ for all $\varepsilon > 0$.*

Proof. Immediate since PSN^*_k is the class C_k in Corollary 9 restricted to $d = 2$. □

Corollary 12. *Let p be a polynomial. Let C_p be the class of all SAS$^+$ instances $\mathbb{P}$ where for all $v \in V$ and all $d \in D$, there are at most $p(dv)$ actions a such that* $\mathrm{eff}(a)[v] = d$. *Then* $\mathrm{COP}(C_p)$ *can be solved in time $d^{(1+\varepsilon)v} \cdot poly(||\mathbb{P}||)$ for all $\varepsilon > 0$.*

Proof. There are dv combinations of variables and variables values, so there can be at most $dv \cdot p(dv)$ actions. Hence, Corollary 8 applies. □

Corollary 13. $\mathrm{COP}(\mathrm{SAS}^+\text{-P})$ *can be solved in time $d^{(1+\varepsilon)v} \cdot poly(||\mathbb{P}||)$ for all $\varepsilon > 0$.*

Proof. Immediate from Corollary 12. □

It is known that LOP is **PSPACE**-complete for SAS$^+$-U, SAS$^+$-S and SAS$^+$-B, while it is only known to be **NP**-hard for SAS$^+$-P (Bäckström & Nebel, 1995). It is also known that the parameterised complexity of LOP, using the plan length as parameter, is **W**[2]-complete for SAS$^+$-S and SAS$^+$-B, **W**[1]-complete for SAS$^+$-U and in **FPT** for SAS$^+$-P (Bäckström et al., 2015). These two different types of analyses thus give different separations, giving a more fine-grained picture when combined. The results above add to this, showing that LOP for SAS$^+$-U and SAS$^+$-P can be solved in time $d^{(1+\varepsilon)v} \cdot \mathrm{poly}(||\mathbb{P}||)$ for all $\varepsilon > 0$, while no such results are known for SAS$^+$-S and SAS$^+$-B. This separation is not as sharp, of course, since it is only based on the absence of such results for the latter classes.

3.4 Choice of Algorithms for Upper Bounds

There are implementations of Dijkstra's algorithm that are even faster than the one by Fredman and Tarjan (1987) under various restrictions on the costs or on the edge set. For instance, Thorup (2000) presents an implementation that runs in time $O(|E| \log\log |E|)$, which is faster than time $O(|E| + |S| \log |S|)$ for very sparse graphs. It does not work for rational edge costs, but it holds for integer costs and certain other numeric domains, for instance, IEEE 754 floating-point numbers[4]. Furthermore, using any of these faster implementations does not give much improvement in the result of Theorem 7. In particular, the factor $\mathrm{poly}(||\mathbb{P}||)$ may dominate factors like $\log |S|$ and $\log\log |E|$, so a more detailed analysis of this polynomial and an appropriate choice of datastructures would be necessary. If we only want to solve PSAT, then we could use standard breadth-first search. This algorithm runs in time $O(|E| + |S|)$ since it does not need any priority queue, or similar structure, to keep track of the length or cost of paths. However, this would only help for very sparse graphs and it will still take time $|S| \cdot \mathrm{poly}(||\mathbb{P}||)$ to build the graph. Furthermore, we will later see that Theorems 7 and 37 together result in tight upper and lower bounds for PSAT, LOP and COP for a large number of important subclasses of SAS$^+$ and PSN. A different choice of algorithm could not improve this result, even for PSAT.

All these upper bounds assume that we have explicit access to the state-transition graph of the instance. We can also view the instance $\mathbb{P}$ as a compact representation of its own state-transition graph in the sense of Galperin and Wigderson (1983), where we can check in time $\mathrm{poly}(||\mathbb{P}||)$ if an edge exists between any two states. This is, of course, also how most AI search algorithms work. Dijkstra's algorithm can also be implemented to use an implicit graph representation by assuming a default distance value of $+\infty$ for all states that we have not yet encountered. Whenever we follow an edge to a state s, we check if this state is already encountered or not. This requires a separate data structure to keep track of the encountered states. If $|E| \leq \mathrm{poly}(dv) \cdot d^v$, as in the proof of Theorem 7, then we can find the next adjacent state and check if it is already encountered in amortized time $\mathrm{poly}(dv)$, so the upper bound in the theorem will still hold. The time complexity of standard AI search algorithms is usually given as $O(b^h)$, where b is the branching factor and h is the height (or depth) of the tree of visited search nodes.[5] This is a very coarse bound, especially since it is often difficullt to find good bounds on the parameters b and h. In the worst case, all actions are applicable in a state, giving a branching factor of $b = a$. Furthermore, the shortest plan can be of length $2^v - 1$ in the worst case. This gives an upper bound of time $O(b^h) = O(a^{2^v-1})$, which is clearly not very useful. In the case of heuristic search algorithms, like A^*, one sometimes uses an 'average' or 'effective' branching factor (Russell & Norvig, 1995), but the value of this cannot be determined a priori since it depends on

4. If we use floating-point numbers we may no longer solve COP exactly due to rounding errors.
5. To be precise, we also need a factor $\mathrm{poly}(||\mathbb{P}||)$, which is usually omitted in the literature.

the actual search tree for an instance rather than the instance itself. This can still be useful for comparing algorithms, but it is not very useful for analysing the complexity of problems.

4. ETH-based Subexponential Lower Bounds

In this section, we will prove that a number of restricted planning problems that are known to be **NP**-hard also cannot be solved in subexponential time, unless the ETH is false. We first consider the Psat and LOP problems for the subclasses of PSN previously considered by Bylander (1994), i.e. classes defined by restricting the number and polarity of literals in action preconditions and effects. Then we consider Psat and LOP for the subclasses of SAS$^+$ defined by the PUBS restrictions previously considered by Bäckström and Nebel (1995). We finally demonstrate how this type of lower bounds can also be proven for classes of planning problems defined by restrictions on the causal graph.

A few remarks are in place, before carrying on to the results. There are exponential lower bounds for plan generation in the literature. For instance, Bäckström and Klein (1991) proved that plan generation for SAS$^+$-PUB requires time $\Omega(2^v)$ in the worst case. This was proven by showing that the shortest plans are of length $\Omega(2^v)$ in the worst case, so it takes time $\Omega(2^v)$ just to output the solution. Such problems are referred to as *inherently intractable*. The lower bounds we will present in this section apply to the decision problems Psat and LOP, that is we will show that we cannot even decide if a plan (of certain length) exists in subexponential time, which is a considerable strengthening compared to exponential time for generating a plan. Furthermore, none of the proofs in this section relies on instances with exponential-length solutions; all proofs use instances where the shortest plans are of linear length in the number of variables.

4.1 Lower Bounds for PSAT Based on Restricting Preconditions and Effects

The problems Psat(PSN$^1_{1+}$) and Psat(PSN$^{1+}_2$) are both known to be **NP**-hard (Bylander, 1994, Corollary 3.6, Footnote 4). We will strengthen these results below by also proving that neither can be solved in subexponential time (in either the number of variables or the number of actions), unless the ETH is false.

Construction 14. *Let $\mathbb{F}$ be a 3-CNF formula with variables $x_1, \ldots, x_n$ and with clauses $c_1, \ldots, c_m$, where each clause c_j is of the form $\{l^1_j, l^2_j, l^3_j\}$. Construct a corresponding* PSN$^1_{1+}$ *instance $\mathbb{P} = \langle V, A, s_I, s_G \rangle$ as follows:*

- $V = \{f_i, t_i \mid 1 \leq i \leq n\} \cup \{y_j \mid 1 \leq j \leq m\}$;
- *A contains the actions*
  - $\mathit{setf}_i : \{\overline{t_i}\} \Rightarrow \{f_i\}$, *for all i $(1 \leq i \leq n)$,*
  - $\mathit{sett}_i : \{\overline{f_i}\} \Rightarrow \{t_i\}$, *for all i $(1 \leq i \leq n)$ and*
  - $\mathit{vfy}^k_j : \{\hat{l}^k_j\} \Rightarrow \{y_j\}$, *for all j, k $(1 \leq j \leq m,\ 1 \leq k \leq 3)$,*
    where $\hat{l}^k_j = f_i$ if $l^k_j = \overline{x_i}$ and $\hat{l}^k_j = t_i$ if $l^k_j = x_i$;
- $s_I = \varnothing$ *and* $s_G = \{y_1, \ldots, y_m\}$.

Note that we assume all clauses have exactly three literals in this construction and in many that follow, but this is only for simplicity, and not a necessary restriction.

Theorem 15. $\textsc{Psat}(\text{PSN}^1_{1+})$ *cannot be solved in time* $2^{o(v)}$ *or time* $2^{o(a)}$ *unless the* ETH *is false.*

Proof. Let $\mathbb{F}$ be an arbitrary 3-CNF formula with n variables and m clauses and let $\mathbb{P}$ be the corresponding PSN instance according to Construction 14. Then $\mathbb{P}$ is a PSN^1_{1+} instance with $v = 2n + m$ variables and $a = 2n + 3m$ actions. Each variable x_i in $\mathbb{F}$ is encoded as two variables f_i and t_i in $\mathbb{P}$, where f_i true represents that x_i is false and t_i true represents that x_i is true. A plan for $\mathbb{P}$ can freely set either f_i or t_i to true, but not both. Hence, a plan chooses a (partial) truth assignment for variables $x_1, \ldots, x_n$. For each clause c_j of $\mathbb{F}$, there is a corresponding variable y_j and three actions, one for verifying each of the literals in c_j. A plan must contain at least one such action for each clause of $\mathbb{F}$ in order to set all y_j variables. Hence, Construction 14 is a polynomial reduction from 3-SAT to $\textsc{Psat}(\text{PSN}^1_{1+})$. The theorem follows by applying Lemma 5 separately to both measures v and a. ☐

Construction 16. *Let* $\mathbb{F}$ *be a 3-CNF formula with variables* $x_1, \ldots, x_n$ *and with clauses* $c_1, \ldots, c_m$*, where each clause* c_j *is of the form* $\{l^1_j, l^2_j, l^3_j\}$*. Construct a corresponding* PSN^{1+}_2 *instance* $\mathbb{P} = \langle V, A, s_I, s_G\rangle$ *as follows:*

- $V = \{e_i, f_i, t_i \mid 1 \leq i \leq n\} \cup \{y_j \mid 1 \leq j \leq m\}$;
- *A contains the actions*
  - $setf_i : \{e_i\} \Rightarrow \{\overline{e_i}, f_i\}$, *for all* i $(1 \leq i \leq n)$,
  - $sett_i : \{e_i\} \Rightarrow \{\overline{e_i}, t_i\}$, *for all* i $(1 \leq i \leq n)$ *and*
  - $vfy^k_j : \{\hat{l}^k_j\} \Rightarrow \{y_j\}$, *for all* j, k $(1 \leq j \leq m,\ 1 \leq k \leq 3)$, *where* $\hat{l}^k_j = f_i$ *if* $l^k_j = \overline{x_i}$ *and* $\hat{l}^k_j = t_i$ *if* $l^k_j = x_i$;
- $s_I = \{e_1, \ldots, e_n\}$ *and* $s_G = \{y_1, \ldots, y_m\}$.

Theorem 17. $\textsc{Psat}(\text{PSN}^{1+}_2)$ *cannot be solved in time* $2^{o(v)}$ *or time* $2^{o(a)}$ *unless the* ETH *is false.*

Proof. Construction 16 is analogous to Construction 14, except that it uses the extra e_i variables to prevent a plan from setting both f_i and t_i to true. Hence, Construction 16 is a polynomial reduction from 3-SAT to $\textsc{Psat}(\text{PSN}^{1+}_2)$ such that $v = 3n+m$ and $a = 2n+3m$, so the theorem follows from Lemma 5. ☐

It is further known that the problems $\textsc{Psat}(\text{PSN}^{*+}_{*+})$, $\textsc{Psat}(\text{PSN}^0_*)$ and $\textsc{Psat}(\text{PSN}^{*+}_1)$ can be solved in polynomial time (Bylander, 1994, Thm. 3.7, 3.9, Footnote 4). Hence, we have a complete classification of $\textsc{Psat}$ for all PSN classes defined by the number and polarity of preconditions and effects of actions, each being classified as either tractable or not solvable in subexponential time (unless the ETH is false).

4.2 Lower Bounds for LOP Based on Restricting Preconditions and Effects

It is sufficient to consider LOP for monotone PSN, since LOP is hard already for very restricted such classes. Even LOP(PSN^{1+}_{1+}) is **NP**-complete (Bylander, 1994, Corollary 4.3) and cannot be approximated within a constant (Betz & Helmert, 2009, Thm. 2). We will complement these result by an ETH-based lower bound.

Construction 18. *Let $\mathbb{F}$ be a 3-CNF formula with variables $x_1, \ldots, x_n$ and with clauses $c_1, \ldots, c_m$, where each clause c_j is on the form $\{l^1_j, l^2_j, l^3_j\}$. Construct a corresponding PSN^{1+}_{1+} instance $\mathbb{P} = \langle V, A, s_I, s_G \rangle$ as follows:*

- $V = \{f_i, t_i, s_i \mid 1 \leq i \leq n\} \cup \{y_j \mid 1 \leq j \leq m\}$;
- *for all i $(1 \leq i \leq n)$, A contains the actions*
  - $set^f_i : \varnothing \Rightarrow \{f_i\}$,
  - $set^t_i : \varnothing \Rightarrow \{t_i\}$,
  - $sets^f_i : \{f_i\} \Rightarrow \{s_i\}$ *and*
  - $sets^t_i : \{t_i\} \Rightarrow \{s_i\}$.

  for all j, k $(1 \leq j \leq m,\ 1 \leq k \leq 3)$, A contains the action
  - $vfy^k_j : \{\hat{l}^k_j\} \Rightarrow \{y_j\}$, *where* $\hat{l}^k_j = f_i$ *if* $l^k_j = \overline{x_i}$ *and* $\hat{l}^k_j = t_i$ *if* $l^k_j = x_i$;
- $s_I = \varnothing$ *and* $s_G = \{s_1, \ldots, s_n\} \cup \{y_1, \ldots, y_m\}$.

Theorem 19. LOP(PSN^{1+}_{1+}) *cannot be solved in time $2^{o(v)}$ or time $2^{o(a)}$ unless the* ETH *is false.*

Proof. Let $\mathbb{F}$ be a 3-CNF formula and let $\mathbb{P}$ be the corresponding PSN instance according to Construction 18. We claim that $\mathbb{F}$ is satisfiable if and only if $\mathbb{P}$ has a plan of length $2n + m$.

$\Rightarrow$: Suppose α is a satisfying assignment for $\mathbb{F}$. Construct a plan ω as follows. For each variable x_i, let ω contain set^f_i followed by $sets^f_i$ if $\alpha(x_i) = 0$ and otherwise let ω contain set^t_i followed $sets^t_i$. Then, for each clause $c_j = \{l^1_j, l^2_j, l^3_j\}$, there is at least one k such that α makes l^k_j true. Choose such a k and add action vfy^k_j at the end of ω. Clearly, ω is a plan for $\mathbb{P}$ of length $2n + m$.

$\Leftarrow$: Suppose ω is a plan for $\mathbb{P}$ of length $2n + m$. It must contain n actions setting the s_i variables and m actions setting the y_j variables. In order to set the s_i variables, it must also set either of f_i and t_i for each i, but it cannot set both since there can only be n such actions in total. Hence, ω corresponds to a satisfying assignment.

It follows that Construction 18 is a polynomial reduction from 3-SAT to LOP(PSN^{1+}_{1+}) (asking for a plan of length $2n + m$, or less) such that $v = 3n + m$ and $a = 4n + 3m$, so the theorem follows from Lemma 5. □

Theorem 19 covers all cases of monotone planning, except when actions have no preconditions at all. In the case of no preconditions, we will not use the ETH, but a conjecture about the k-Set Cover problem, which is defined as follows:

> k-Set Cover
> *Instance:* A set S, a set C of subsets of S such that $|c| \leq k$ for all $c \in C$ and an integer $t \geq 1$.
> *Question:* Does S have a cover of size t, i.e. is there a subset $C' \subseteq C$ such that $\bigcup_{c \in C'} c = S$ and $|C'| \leq t$?

Note that k-Set Cover is a restriction of Set Cover where all sets have size k at most, just as k-SAT is a restriction of SAT where all clauses have size k at most. Cygan, Dell, Lokshtanov, Marx, Nederlof, Okamoto, Paturi, Saurabh, and Wahlström (2016) conjectured that k-Set Cover cannot be solved in time $2^{o(n)}$ unless the Seth is false, where $n = |S|$.

Theorem 20. LOP(PSN$^0_{k+}$) *cannot be solved in time* $2^{o(v)}$ *unless* k-Set Cover *can be solved in time* $2^{o(n)}$.

Proof. Proof by polynomial reduction from k-Set Cover. Given an instance $\mathbb{I} = \langle S, C \rangle$ of k-Set Cover, construct a LOP(PSN$^0_{k+}$) instance $\mathbb{P} = \langle V, A, s_I, s_G \rangle$, where $V = S$, A contains the action $a_c : \varnothing \Rightarrow \{x \mid x \in c\}$ for every $c \in C$, $s_I = \varnothing$ and $s_G = S$. Clearly, $\mathbb{P}$ has a plan of length t if and only if $\mathbb{I}$ has a cover of size t. □

It is known that LOP(PSN$^0_{3+}$) is **NP**-complete and that LOP(PSN$^0_{2+}$) is in **P** (Bylander, 1994).[6] The latter result can be generalized to hold also for COP.

Theorem 21. COP(PSN$^0_{2+}$) *is in* **P**.

Proof sketch. Equivalent to the Minimum Edge Cover problem, which can be solved in polynomial time for real costs (White, 1971).[7] □

Theorems 19–21 provide a complete classification of LOP and COP for all PSN classes defined by the number and polarity of preconditions and effects of actions, each being classified as either tractable or not solvable in subexponential time (unless the ETH, or in some cases the Seth, is false).

4.3 Lower Bounds for PSAT and LOP Based on the PUBS Restrictions

We will now prove ETH-based subexponential lower-bound results for SAS$^+$ planning for all **NP**-hard combinations of the PUBS restrictions. We will need the following construction, which can be used to encode disjunctions in SAS$^+$-PUB instances.

Construction 22. *(Bäckström et al., 2015, in the proof of Lemma 2) An OR gate g with two inputs x_1, x_2 and output o can be encoded as a SAS$^+$-PUB instance $\mathbb{P}_{OR} = \langle V, D, A, s_I, s_G \rangle$ as follows:*

- $V = \{x_1, x_2, o, o_1, o_2, i_1, i_2\}$;
- *A contains the actions*
  - $a_o : \{o_1 = 1, o_2 = 1\} \Rightarrow \{o = 1\}$,

6. The **NP**-completeness result for LOP(PSN$^0_{3+}$) is originally from Erol, Nau, and Subrahmanian (1991).
7. White (1971) only mentions the result, which appeared in his doctoral thesis. See Plesník (2001) for a survey of algorithms and results for the Minimum Edge Cover problem.

   – $a_{o_1} : \{i_1 = 1, i_2 = 0\} \Rightarrow \{o_1 = 1\}$,
   – $a_{o_2} : \{i_1 = 0, i_2 = 1\} \Rightarrow \{o_2 = 1\}$,
   – $a_{i_1} : \varnothing \Rightarrow \{i_1 = 1\}$,
   – $a_{i_2} : \varnothing \Rightarrow \{i_2 = 1\}$,
   – $a_{v_1} : \{x_1 = 1\} \Rightarrow \{i_1 = 0\}$ *and*
   – $a_{v_2} : \{x_2 = 1\} \Rightarrow \{i_2 = 0\}$*;*

- $s_I[x_1]$ *and* $s_I[x_2]$ *are arbitrary and* $s_I[v] = 0$ *for all other* $v \in V$*;*
- $s_G[o] = 1$ *and* s_G *is otherwise undefined.*

We refer to Bäckström et al. (2015, proof of Lemma 2) for a correctness proof, but intuitively the construction works as follows. The instance $\mathbb{P}_{\mathrm{OR}}$ implements the logical or function $o = x_1 \vee x_2$, where o_1, o_2, i_1 and i_2 are internal variables. There is no plan for $\mathbb{P}_{\mathrm{OR}}$ if both x_1 and x_2 are initially false. If x_1 is true in s_I, then $a_{i_1}, a_{o_1}, a_{v_1}, a_{i_2}, a_{o_2}, a_o$ is a plan for $\mathbb{P}_{\mathrm{OR}}$, and if x_2 is true in s_I, then $a_{i_2}, a_{o_2}, a_{v_2}, a_{i_1}, a_{o_1}, a_o$ is a plan for $\mathbb{P}_{\mathrm{OR}}$. If both x_1 and x_2 are true in s_I, then both these sequences are plans for $\mathbb{P}_{\mathrm{OR}}$. We can now prove the following lower-bound results.

Theorem 23. Psat*(SAS⁺-PUB) cannot be solved in time* $2^{o(v)}$ *or time* $2^{o(a)}$ *unless the* ETH *is false.*

Proof. Instance $\mathbb{P}$ in Construction 14 is a SAS⁺-UB instance, but it is not post-unique since there are three actions with the same effect for each clause variable y_j. These three actions together simulate the disjunction in the clause. The disjunction $l_j^1 \vee l_j^2 \vee l_j^3$ can be computed in two steps, as $((l_j^1 \vee l_j^2) \vee l_j^3)$. This is equivalent to simulating a 3-input OR gate with two 2-input OR gates, i.e. we need two copies of the OR gate in Construction 22. By replacing the three actions for each clause by two copies of the OR gate construction, we get a modified construction which is a polynomial reduction from 3-SAT to Psat(SAS⁺-PUB). Each OR gate introduces 4 new internal variables, and the output of the first OR-gate in each pair is an input to the second one, thus also being a new variable, so we get 9 extra variables per clause. Furthermore, each OR gate introduces 7 new actions, so we get 14 new action per clause, replacing the 3 previous ones. Since Construction 14 has $2n + m$ variables and $2n + 3m$ actions, the modified construction has $v = 2n + 10m$ variables and $a = 2n + 14m$ actions. The result now follows from Lemma 5. □

Corollary 24. Psat(SAS⁺-PBS) *cannot be solved in time* $2^{o(v)}$ *or time* $2^{o(a)}$ *unless the* ETH *is false.*

Proof. There is a polynomial reduction from LOP(SAS⁺-PUB) to LOP(SAS⁺-PBS) that increases the number of variables by a factor 2 and retains the number of actions (Bäckström & Nebel, 1995, Proof of Theorem 9). The reduction works also for Psat so the result follows from Theorem 23 since $2^{o(2v)} = 2^{o(v)}$. □

It has remained an open question in the literature whether Psat(SAS⁺-PUB) and Psat(SAS⁺-PBS) are **NP**-hard, while the corresponding LOP problems are known to be **NP**-hard (Bäckström & Nebel, 1995). Since the two preceeding proofs use polynomial reduction from 3-SAT we can settle this question affirmatively as a spin-off result.

Corollary 25. PSAT(SAS$^+$-PUB) *and* PSAT(SAS$^+$-PBS) *are* **NP**-*hard.*

It still remains an open question whether these problems are also in **NP**.

While both PSAT and LOP are **NP**-hard for these cases, the problems differ for combinations US and UBS; PSAT(SAS$^+$-US) is in **P** (Bäckström & Nebel, 1995, Thm. 10, 11) but LOP(SAS$^+$-UBS) is **NP**-complete (Bäckström & Nebel, 1995, Lemma 2). We can immediately strenghten the latter result to a lower-bound result.

Corollary 26. *(To Theorem 19)* LOP(SAS$^+$-UBS) *cannot be solved in time* $2^{o(v)}$ *or time* $2^{o(a)}$ *unless the* ETH *is false.*

These results imply a complete classification of PSAT and LOP for all combinations of the PUBS restrictions, classifying each as either tractable or not solvable in subexponential time (unless the ETH is false).

4.4 Lower Bounds Based on Restricting Causal Graphs

The *causal graph* of a planning instance describes certain types of variable dependencies of a planning instance, and has frequently been exploited for identifying easy subclasses or for classifying the complexity of planning classes (cf. Giménez & Jonsson, 2009; Helmert, 2006; Jonsson & Bäckström, 1998; Katz & Domshlak, 2010b; Williams & Nayak, 1997).

Definition 27. *The causal graph for a SAS$^+$ instance* $\mathbb{P} = \langle V, D, A, s_I, s_G \rangle$ *is the directed graph* $CG(\mathbb{P}) = \langle V, E \rangle$ *where for all* $u, v \in V$*,* $\langle u, v \rangle \in E$ *if and only if both* $u \neq v$ *and there is some* $a \in A$ *such that* $u \in vars\big(\mathrm{pre}(a)\big) \cup vars\big(\mathrm{eff}(a)\big)$ *and* $v \in vars\big(\mathrm{eff}(a)\big)$.

We will only briefly consider causal graphs and give an example of lower-bound results for instances with quite restricted causal graphs.

Theorem 28. *If* LOP(PSN^{1+}_{1+}) *can be solved in time* $2^{o(v)}$ *or time* $2^{o(a)}$ *for instances where the causal graph is acyclic, bipartite and has degree 3 and depth 2, then the* ETH *is false.*

Proof. We will show that the proof of Theorem 19 applies also in this case, by showing that all SAS$^+$ instances that need to be considered in that proof have a causal graph satisfying the preconditions of this theorem. Let $\mathbb{F}$ be a 3-CNF formula and let $\mathbb{P}$ be the corresponding PSN^{1+}_{1+} instance according to Construction 18. The causal graph $CG(\mathbb{P})$ for $\mathbb{P}$ contains the following edges:

- $\langle f_i, s_i \rangle$ and $\langle t_i, s_i \rangle$ for all i $(1 \leq i \leq n)$;
- $\langle f_i, y_j \rangle$ for all i, j $(1 \leq i \leq n, 1 \leq j \leq m)$ such that $\overline{x_i} \in c_j$ and $\langle t_i, y_j \rangle$ for all i, j $(1 \leq i \leq n, 1 \leq j \leq m)$ such that $x_i \in c_j$.

It is immediate that $CG(\mathbb{P})$ is acyclic, bipartite and has depth 2. All s_i vertices have two incoming edges and no outgoing edges. All y_j vertices have at most three incoming edges and no outgoing edges. The proof of Theorem 19 relies on Lemma 5, which exploits Lemma 4, so we can assume that no variable occurs in more than 3 clauses of $\mathbb{F}$. If all 3 occurences of a variable have the same polarity, then it is redundant, so we may assume that no variable occurs more than twice with the same polarity. It follows that each f_i and t_i vertex have no incoming edges, one edge to vertex s_i and at most two edges to y_j vertices. It follows that $CG(\mathbb{P})$ has degree 3, at most. $\square$

It should be noted that all SAS$^+$ instances with an acyclic causal graph must satisfy restriction U, so the class of all such instances can be solved in time $d^{(1+\varepsilon)v} \cdot \text{poly}(||\mathbb{P}||)$ for all $\varepsilon > 0$, according to Corollary 10. As two further examples, it is known that Psat is **NP**-hard when the causal graph is either an out-star (aka. fork) or an in-star (aka. inverted fork) (Domshlak & Dinitz, 2001). For both these cases, there are reductions from 3-SAT where v and a are linear combinations of n and m (Bäckström & Jonsson, 2013, Lemma 4, 5). Hence, we can apply Lemma 5 (in this article) and it follows that neither problem can be solved in time $2^{o(v)}$ or $2^{o(a)}$ unless the ETH is false.

5. Subexponential Solvability for Sublinear Number of Actions

In their studies of subexponential solvability of CSP problems, De Haan et al. (2015) found a phase-transition phenomenon as follows.

Proposition 29. *(De Haan et al., 2015, Proposition 6) The restriction of CSP to instances in which the number of tuples is $o(n)$ is solvable in subexponential time, and unless the* ETH *fails, the restriction of CSP to instances in which the number of tuples is $\Omega(n)$ is not solvable in subexponential time.*

This result demonstrates a sharp transition in complexity between having a sublinear number of tuples and having a linear number. In this section, we will demonstrate some analogous results for planning. We start with the following result, for the case where the number of actions is linear in v.

Corollary 30. *(To Theorem 19)* LOP(PSN$^{1+}_{1+}$) *cannot be solved in time $2^{o(v)}$, unless the* ETH *is false, even if $a \in \Theta(v)$ and the optimal plan length is in $\Theta(v)$.*

Proof. The proof of Theorem 19 is based on a reduction where $a \in \Theta(v)$ and the optimal plan length is in $\Theta(v)$. □

Note that we write $\Theta(v)$ instead of $\Omega(v)$ to emphasize that a linear number of actions is sufficient for the hardness results.

We will then consider the three classes PSN$^{*+}_{*+}$, PSN$^{*}_{k+}$ and PSN$^{*}_{*+}$, and show that for all three, COP can be solved in subexponential time in v if the number of actions is sublinear in v. Together with Corollary 30 this establishes corresponding phase-transition results since PSN$^{1+}_{1+}$ is a subclass of all three classes. We will tacitly use the observation that $||\mathbb{P}||$ is polynomial in v if $a \in o(v)$.

For the first class, we start with an observation about problem Psat(PSN$^{*+}_{*+}$), which is in **P** (Bylander, 1994, Footnote 4). The greedy algorithm in Figure 1 solves this problem in polynomial time and also returns a plan, when there is one. It is easy to see that this algorithm is correct. We know from Proposition 1.3 that we need not consider plans with more than one occurence of any action. Furthermore, if an action a is valid in a state s, then a must be valid in all states t such that $s \subseteq t$, i.e. a is valid in $s \ltimes \text{eff}(a')$ for all actions a'. Hence, the order of the actions in a plan does not matter as long as every action in the plan is valid, so the greedy strategy will find a plan if there is one.

```
function GreedyPlan(⟨V, A, s_I, s_G⟩)
  s := s_I; ω := ⟨⟩
  while there is some a ∈ A such that pre(a) ⊑ s do
    s := s ⋉ eff(a)
    Add a to the end of ω
    Remove a from A
  if s_G ⊑ s then return ω
  else reject
```

Figure 1: Greedy algorithm for solving Psat(PSN^{*+}_{*+}).

This algorithm for Psat can be used as a subroutine for solving COP, as follows.

Lemma 31. COP(PSN^{*+}_{*+}) *can be solved in time* $2^a \cdot \mathit{poly}(||\mathbb{P}||)$.

Proof. We can solve COP(PSN^{*+}_{*+}) with the following algorithm. Let $\mathbb{P} = \langle V, A, s_I, s_G\rangle$ be a PSN^{*+}_{*+} instance. Enumerate all subsets A' of A. For each subset A', use GreedyPlan to check if there is a plan for $\langle V, A', s_I, s_G\rangle$, and also generate a plan when there is one. Keep track of the cheapest plan found so far (or one of them, if there are more than one) and return it, or reject if there is no plan for any subset.

We will now prove that this algorithm is correct. First suppose $\mathbb{P}$ is unsolvable. Then the instance $\langle V, A', s_I, s_G\rangle$ must be unsolvable for all $A' \subseteq A$, so the algorithm will reject. Instead suppose $\mathbb{P}$ is solvable and let $\omega^* = a^*_1, \ldots, a^*_\ell$ be a shortest cost-optimal plan for $\mathbb{P}$. Then ω^* cannot contain any action more than once according to Proposition 1.3. Let $A^* = \{a^*_1, \ldots, a^*_\ell\}$. Since we enumerate all subsets of A, we will eventually apply GreedyPlan to the instance $\mathbb{P}^* = \langle V, A^*, s_I, s_G\rangle$. Obviously, ω^* is a plan for $\mathbb{P}^*$, so GreedyPlan will find some plan ω' for $\mathbb{P}^*$ and ω' must then also be a plan for $\mathbb{P}$. Suppose $|\omega'| < |\omega^*|$. Then ω' must contain a proper subset of A^* since ω^* contains exactly the actions in A^*. It follows that $c(\omega') \leq c(\omega^*) = c(A^*)$, but this contradicts that ω^* is a shortest cost-optimal plan for $\mathbb{P}$, so it must be the case that ω' contains all actions in A^*. Hence, either $\omega' = \omega^*$ or ω' is a permutation of ω^*. In either case, we have that $c(\omega') = c(\omega^*)$, so GreedyPlan will find a plan with cost $c(\omega^*)$ for $\mathbb{P}^*$. Since ω^* is cost-optimal, there cannot be any $A' \subseteq A$ such that $\langle V, A', s_I, s_G\rangle$ has a cheaper plan than $c(\omega^*)$, so the algorithm will return a cost-optimal plan. Since there are $2^{|A|}$ subsets of A and the greedy algorithm runs in polynomial time, we can solve COP(PSN^{*+}_{*+}) in time $2^{|A|} \cdot \text{poly}(||\mathbb{P}||)$. □

Theorem 32. COP(PSN^{*+}_{*+}) *can be solved in time* $2^{o(v)}$ *if* $a \in o(v)$.

Proof. COP(PSN^{*+}_{*+}) can be solved in time $2^a \cdot \text{poly}(||\mathbb{P}||)$ according to Lemma 31, so if $a \in o(v)$, then COP(PSN^{*+}_{*+}) can be solved in time $2^{o(v)} \cdot \text{poly}(v) = 2^{o(v)}$. □

There is a similar phase transition for actions with arbitrary preconditions, if we limit their effects to a constant number of variables.

Theorem 33. COP(PSN^{*}_{k+}) *can be solved in time* $2^{o(v)}$ *if* $a \in o(v)$.

Proof. With a actions, we can set at most ka different variables, so $v - ka$ variables are trivially redundant and can be removed from the instance before solving it. We get at most ka remaining variables, but $ka \in o(v)$ since $a \in o(v)$. Hence, we can solve the problem in time $3^{o(v)} \cdot \text{poly}(v) = 3^{o(v)} = 2^{o(v)}$, using Proposition 6. $\square$

If allowing also arbitrary preconditions, we get a somewhat less sharp phase transition.

Lemma 34. *Let $\mathbb{P} = \langle V, A, s_I, s_G \rangle$ be a* PSN *instance. Generating all plans of length ℓ, or less, can be done in time $\ell^2 |A|^\ell \cdot \mathit{poly}(|V|)$.*

Proof. For $|A| \geq 2$ and $\ell \geq 2$, there are at most

$$|A|^0 + |A|^1 + \cdots + |A|^\ell \leq \ell |A|^\ell$$

action sequences of length ℓ, or less. Each plan of length ℓ, or less, can be verified in time $\ell \cdot \text{poly}(|V|)$. Hence, we can generate all plans of length ℓ, or less, in time $\ell |A|^\ell \cdot \ell \cdot \text{poly}(|V|) = \ell^2 |A|^\ell \cdot \text{poly}(|V|)$. $\square$

Theorem 35. *If $a \in o(\frac{v}{\log v})$, then* COP(PSN$^*_{*+}$) *can be solved in time $2^{o(v)}$.*

Proof. According to Proposition 1.3, no action occurs more than once in a shortest cost-optimal plan, so the maximum plan length we need to consider is a. It follows from Lemma 34 that we can generate all plans of length a, or less, in time $a^2 a^a \cdot \text{poly}(v)$, which equals $a^a \cdot \text{poly}(v)$ since $a \in o(v)$. We can solve COP(PSN$^*_{*+}$) by keeping track of the cheapest plan found. We must prove that $a^a \cdot \text{poly}(v) \in 2^{o(v)}$, but $\text{poly}(v) \in 2^{o(v)}$ so it remains to prove that $a^a \in 2^{o(v)}$. Since $a \in o(\frac{v}{\log v})$, it holds for all $c > 0$ that $a < c\frac{v}{\log v}$, for large v. We get

$$a^a = 2^{a \log a} < 2^{(\frac{cv}{\log v}) \log \frac{cv}{\log v}} = 2^{cv \frac{\log \frac{cv}{\log v}}{\log v}}$$

for all $c > 0$ and large v. Choose an arbitrary $c' > 0$. We want to prove that there is a $c > 0$ such that

$$2^{cv \frac{\log \frac{cv}{\log v}}{\log v}} \leq 2^{c'v},$$

that is,

$$cv \frac{\log \frac{cv}{\log v}}{\log v} \leq c'v.$$

We rewrite to

$$cv \frac{\log c + \log v - \log \log v}{\log v} \leq c'v,$$

but

$$\frac{\log c + \log v - \log \log v}{\log v} < 1$$

for $v > 2^c$ so it is sufficient to choose $c = c'$, which is allowed since we only require that $c > 0$. It follows that $a^a \in 2^{o(v)}$, since c' was chosen arbitrarily. We have now shown that $a^a \cdot \text{poly}(v) \in 2^{o(v)}$ and, thus, that COP(PSN$^*_{*+}$) can be solved in time $2^{o(v)}$ when $a \in o(\frac{v}{\log v})$. $\square$

6. Tighter Lower Bounds

The previous lower-bound results in Section 4 are of the form that a certain planning problem cannot be solved in time $2^{o(v)}$ unless the ETH is false. While being a very strong indication of hardness, such results still say very little about how fast we can solve a problem. For instance, there is a huge difference between the functions $2^{0.001v}$ and $2^{0.999v}$, even though both are exponential. In this section, we will demonstrate two different ways to achieve more precise characterizations of the value of c in time bounds of the form $2^{cv} \cdot \text{poly}(||\mathbb{P}||)$. We will first show that $\text{Psat}(\text{PSN}_1^*)$ cannot be solved in time $2^{(1-\varepsilon)v} \cdot \text{poly}(||\mathbb{P}||)$ for any $\varepsilon > 0$, unless the Seth is false, i.e. the value of c is 1. We will then provide a result stating that $\text{LOP}(\text{PSN}_*^{1+})$ cannot be solved faster than time $2^{\frac{v}{2}} \cdot \text{poly}(||\mathbb{P}||)$ unless the Graph Colourability problem can be solved faster than time $2^n \cdot \text{poly}(n)$, which is the best currently known lower bound for this problem. That is, we do not determine a fixed value for c, but a value that depends on the best known value of the constant for Graph Colourability.

6.1 A Lower Bound for PSAT Using the Strong Exponential Time Hypothesis

We will now prove a tighter lower bound for PSN_1^* by using the Seth instead of the ETH. We first recall how to encode counters in PSN (see Bäckström & Jonsson, 2012, for details). Assume we have n variables $x_1, \ldots, x_n$ and let the states represent the integers $0, \ldots, 2^n - 1$ by treating a state as a bit vector with x_1 as the least significant bit. An n-bit binary counter can be implemented by the following n counting actions:

$$a_i : \{x_1, \ldots, x_{i-1}, \overline{x_i}\} \Rightarrow \{\overline{x_1}, \ldots, \overline{x_{i-1}}, x_i\}, \text{ for all } i \ (1 \leq i \leq n).$$

For instance, the plan $a_1, a_2, a_1, a_3, a_1, a_2, a_1$ enumerates the states $0, \ldots, 7$ in numeric order. We can also implement a counter that counts in Gray code, i.e. it enumerates the states in an order such that successive states have Hamming distance 1. This requires $2n$ actions, defined as:

$$s_i : \{\overline{x_i}, x_{i-1}, \overline{x_{i-2}}, \ldots, \overline{x_1}\} \Rightarrow \{x_i\}, \text{ for all } i \ (1 \leq i \leq n),$$
$$r_i : \{x_i, x_{i-1}, \overline{x_{i-2}}, \ldots, \overline{x_1}\} \Rightarrow \{\overline{x_i}\}, \text{ for all } i \ (1 \leq i \leq n).$$

For instance, the plan $s_1, s_2, r_1, s_3, s_1, r_2, r_1$ enumerates the states $0, \ldots, 7$ in order $0, 1, 3, 2, 6, 7, 5, 4$. The Gray-code counter has the property that it is always a PSN_1^* instance. Otherwise, it is sufficient for our purposes to note that both counters can enumerate all integers $0, \ldots, 2^n - 1$ with a plan of length $2^n - 1$.

The following construction is a polynomial reduction from 3-UNSAT to Psat(PSN) (Aghighi et al., 2016b, Lemma 10). Of importance to us is that it encodes a 3-CNF formula with n variables as a PSN instance with $v = n + 1$ variables.

Construction 36. *(Aghighi et al., 2016b, Construction 9) Let $\mathbb{F}$ be a 3-CNF formula with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$. Assume without loss of generality that $\mathbb{F}$ contains no clause c_j that is a tautology, i.e. both x_i and $\overline{x_i}$ appear in c_j for some $1 \leq i \leq n$. We construct a corresponding* PSN *instance $\mathbb{P} = \langle V, A, s_I, s_G \rangle$ as follows:*

- *Let $V = \{x_1, \ldots, x_{n+1}\}$.*

- *Let $a_1, \ldots, a_{n+1}$ denote the actions in an $n+1$-bit binary counter over the variables $x_1, \ldots, x_{n+1}$. For each clause $c_j = (l_1 \vee l_2 \vee l_3)$ of $\mathbb{F}$, where $1 \leq j \leq m$, define $T_j = \{\overline{l_1}, \overline{l_2}, \overline{l_3}\}$. Let $A = \{\, a_{i,j} \mid 1 \leq i \leq n+1,\ 1 \leq j \leq m \,\}$, where the actions are defined as $a_{i,j} : \mathrm{pre}(a_i) \cup T_j \Rightarrow \mathrm{eff}(a_i)$. One may view $a_{i,j}$ as action a_i extended with preconditions expressing that "clause j is not satisfied by $x_1, \ldots, x_n$".*

- *Let $s_I = \varnothing$ and $s_G = \{\overline{x_1}, \ldots, \overline{x_n}, x_{n+1}\}$.*

Intuitively, the reduction works as follows. Interpreting the states as integers, we ask for a plan from state 0 to state 2^n. This plan must enumerate all combinations of values for $x_1, \ldots, x_n$, i.e. it enumerates all assignments for $\mathbb{F}$. For each state, exactly one of the original counter actions is applicable, so one of the m new variants of this action must be applicable, which means that at least one clause must be false in this state. This is only possible if $\mathbb{F}$ is unsatisfiable, since there is otherwise some state where all clauses are true and the counter will get stuck there. This construction is illustrated in Figure 2.

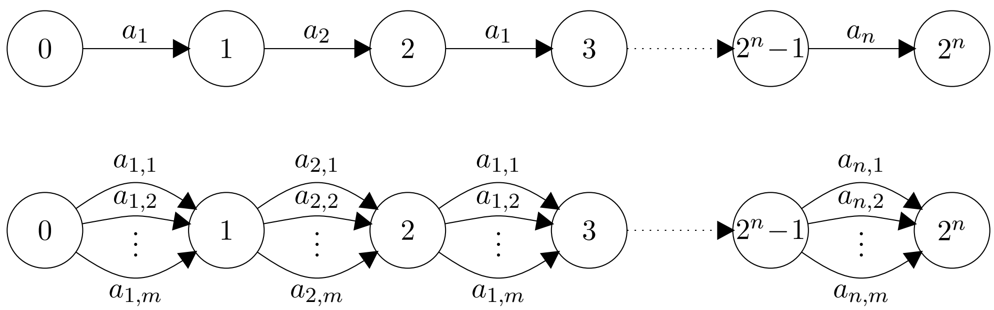

Figure 2: Illustration of an n-bit binary counter counting from 0 to 2^n (top) and of Construction 36 (bottom).

We make the following two observations about this construction.

1. It is straightforward to extend it to CNF formulae with unrestricted clause size. (The original construction was restricted to 3-CNF in order to achieve a property that is of no interest to us here.)

2. All counter actions are treated in the same way, i.e. each one is replaced with m new actions with preconditions added in a systematic way that does not depend on the original action. Hence, the construction works also with a Gray-code counter, if we adjust the goal appropriately, which makes $\mathbb{P}$ a PSN_1^* instance.

We can now prove the following result.

Theorem 37. *If* $\textsc{Psat}(\mathrm{PSN}_1^*)$ *can be solved in time* $2^{(1-\varepsilon)v} \cdot$ *poly*$(||\mathbb{P}||)$ *for some* $\varepsilon > 0$, *then the* $\textsc{Seth}$ *is false.*

Proof. We consider the modified variant of Construction 36 that allows unrestricted clause size and uses a Gray-code counter. This is obviously a polynomial reduction from UNSAT to PSAT(PSN_1^*). Assume the SETH holds. Let p be a polynomial such that the reduction takes time $p(||\mathbb{F}||)$, or less, for each formula $\mathbb{F}$. Suppose there is an $\varepsilon > 0$ such that PSAT(PSN_1^*) can be solved in time $O\big(2^{(1-\varepsilon)v} \cdot q(||\mathbb{P}||)\big)$, for some polynomial q. Obviously, $||\mathbb{P}|| \leq p(||\mathbb{F}||)$ so we can solve UNSAT in time $O\big(p(||\mathbb{F}||) + 2^{(1-\varepsilon)(n+1)} \cdot q(p(||\mathbb{F}||))\big)$, since $v = n+1$. That is, we can solve UNSAT in time $O\big(2^{(1-\varepsilon)(n+1)} \cdot q(p(||\mathbb{F}||))\big) = O\big(2^{(1-\varepsilon)n} \cdot q(p(||\mathbb{F}||))\big)$. Since the SETH holds, by assumption, there are constants $s_3, s_4, s_5, \ldots$ as specified in Definition 2 such that $\lim_{k\to\infty} s_k = 1$. We can, thus, choose an integer k such that $1 - \varepsilon < s_k$. Since we can solve UNSAT in time $O\big(2^{(1-\varepsilon)n} \cdot q(p(||\mathbb{F}||))\big)$, we can also solve k-UNSAT in time $O\big(2^{(1-\varepsilon)n} \cdot q(p(||\mathbb{F}||))\big)$. However, $m \in \Theta(n^k)$ for all k-CNF formulae $\mathbb{F}$, so it follows that $||\mathbb{F}||$ is polynomial in n. Hence, we can solve k-UNSAT in time $O(2^{(1-\varepsilon)n})$, which contradicts the SETH since $1-\varepsilon < s_k$. It follows that our assumption must be wrong and that PSAT(PSN_1^*) cannot be solved in time $O\big(2^{(1-\varepsilon)v} \cdot \mathrm{poly}(||\mathbb{P}||)\big)$ for any $\varepsilon > 0$, unless the SETH is false. □

Combining this result with Theorem 7 yields very tight upper and lower bounds for a very large number of subclasses. In particular we note the following corollaries.

Corollary 38. *For all $k \geq 1$,* COP(PSN_k^*) *can be solved in time $2^{(1+\varepsilon)v} \cdot \mathit{poly}(||\mathbb{P}||)$ for all $\varepsilon > 0$, but* PSAT(PSN_k^*) *cannot be solved in time $2^{(1-\varepsilon)v} \cdot \mathit{poly}(||\mathbb{P}||)$ for any $\varepsilon > 0$, unless the* SETH *is false.*

Proof. Combine Corollary 11 and Theorem 37. □

Corollary 39. COP(SAS^+-UB) *can be solved in time $2^{(1+\varepsilon)v} \cdot \mathit{poly}(||\mathbb{P}||)$ for all $\varepsilon > 0$, but* PSAT(SAS^+-UB) *cannot be solved in time $2^{(1-\varepsilon)v} \cdot \mathit{poly}(||\mathbb{P}||)$ for any $\varepsilon > 0$, unless the* SETH *is false.*

Proof. Combine Corollary 10 and Theorem 37. □

6.2 A Functional Lower Bound for LOP Using Graph Colourability

All previous lower-bound results are based on assuming that the ETH, or even the SETH, holds. We will now demonstrate a different way to achieve lower-bound results, where the constant c in the exponent depends on how fast we can solve some other problem, in this case GRAPH COLOURABILITY, which is defined as follows.

> GRAPH COLOURABILITY
> *Instance:* A graph $G = \langle V, E\rangle$ and a positive integer $k \leq |V|$.
> *Question:* Is G k-colourable, i.e. is there a function $f : V \to \{1, \ldots, k\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

The best known upper bound for the GRAPH COLOURABILITY problem is time $2^n \cdot \mathrm{poly}(n)$, where $n = |V|$, (Björklund, Husfeldt, & Koivisto, 2009, Proposition 1), and it is considered an important open question whether a faster algorithm can exist (Impagliazzo & Paturi, 2013). We will now exploit this as an alternative condition for proving lower-bound results for planning.

Theorem 40. *If* LOP(PSN_*^{1+}) *can be solved in time* $2^{\frac{cv}{2}} \cdot poly(v)$ *for some* $c > 0$*, then* GRAPH COLOURABILITY *can be solved in time* $2^{cn} \cdot poly(n)$*.*

Proof. Proof by reduction from GRAPH COLOURABILITY to LOP(PSN_*^{1+}). Let $\mathbb{I} = \langle G, k\rangle$ be an instance of GRAPH COLOURABILITY, where $G = \langle V, E\rangle$ is a graph and $k > 0$ is an integer. Assume $V = \{v_1, \ldots, v_n\}$. Construct a corresponding LOP(PSN_*^{1+}) instance $\mathbb{I}' = \langle \mathbb{P}', k'\rangle$ as follows. Let $\mathbb{P}' = \langle V', A', s_I{}', s_G{}'\rangle$, where

- $V' = \{v_1, \ldots, v_n, e_1, \ldots, e_n\}$;
- A' contains the actions
  - $a_s : \varnothing \Rightarrow \{e_1, \ldots, e_n\}$, and
  - $a_i : \{e_i\} \Rightarrow \{v_i\} \cup \{\overline{e_j} \mid \{v_i, v_j\} \in E\}$ for all $v_i \in V$;
- $s_I = \varnothing$ and $s_G{}' = \{v_1, \ldots, v_n\}$.

Let $k' = n + k$.

A plan colours the vertices in phases, one colour in each phase. The phases are separated by occurences of action a_s, which switches to the next colour. Variable v_i is true if vertex v_i has been coloured and variable e_i is an enabling variable, signalling that v_i may be coloured for the moment. The actual colour of a vertex is only implicit in the plan, and not explicitly represented. At the start of each phase, any node can be coloured and colouring a node immediately blocks its neighbours from being coloured in the same phase. Note that an already coloured node can be recoloured in a later phase, although this may result in a plan that is not length optimal. We now claim that $\mathbb{I}$ is k-colourable if and only if $\mathbb{I}'$ has a plan of length k'.

$\Rightarrow$: Suppose G has a k-colouring. Then there is a partition $C_1, \ldots, C_k$ of V, such that C_i contains all vertices with colour i, for all i ($1 \leq i \leq k$). Then each C_i is an independent set, i.e. there are no two $v_j, v_h \in C_i$ such that $\{v_j, v_h\} \in E$. Define the action sequence $\omega = a_s, \omega_1, a_s, \omega_2, a_s, \ldots, a_s, \omega_k$, where ω_i contains action a_j for each $v_j \in C_i$ in arbitrary order. The a_s actions guarantee that variables $e_1, \ldots, e_n$ are true at the start of sequence ω_i for each i ($1 \leq i \leq k$). Since there are no two $v_j, v_h \in C_i$ such that $\{v_j, v_h\} \in E$, there is no action in ω_i that sets e_j to false for any $v_j \in C_i$. It follows that all actions in ω_i are valid. Furthermore, the a_s actions are always valid. Since each $v_j \in V$ occurs in exactly one set C_i, it follows that action a_j occurs exactly once in ω for each $v_j \in V$. Hence, the resulting state satisfies s_G and it follows that ω is a plan for $\mathbb{P}'$ of length $n + k = k'$.

$\Leftarrow$: Suppose ω is a plan for $\mathbb{P}'$ of length k' or less. Without losing generality, assume ω is a shortest such plan. Then ω does not contain any successive occurences of action a_s, so it is of the form $\omega = a_s, \omega_1, a_s, \omega_2, a_s, \ldots, a_s, \omega_m$, for some m, where the subplans ω_i do not contain any occurences of action a_s. Since ω must contain at least one occurence of action a_j for each $v_j \in V$, it follows that $|\omega| \geq n+m$, i.e. $m \leq k$ since $|\omega| \leq k' = n+k$. Suppose there is some i ($1 \leq i \leq m$) and two actions a_j, a_h in ω_i such that $\{v_j, v_h\} \in E$. Without losing generality, assume v_j occurs before v_h. Then a_j sets e_h to false, which blocks the execution of a_h. Hence, the assumption must be false and $\{v_j, v_h\} \notin E$ for all $a_j, a_h \in \omega_i$. It follows that G must have an m-colouring, and, thus, also a k-colouring.

It follows that the construction is a polynomial reduction from Graph Colourability to LOP(PSN_*^{1+}). Suppose there is some $c > 0$ such that LOP(PSN_*^{1+}) can be solved in time $2^{\frac{cv}{2}} \cdot \text{poly}(v)$. Since $|V'| = 2|V|$, we can solve Graph Colourability in time $2^{cn} \cdot \text{poly}(n)$. □

This is a tighter bound than the ETH-based ones in Section 4 in the following sense. If LOP(PSN_*^{1+}) can be solved faster than time $2^{\frac{v}{2}} \cdot \text{poly}(v)$, then there is a faster algorithm for Graph Colourability than previously known, i.e. this result is based on an assumption about a specific fixed value for the constant in the exponent. Note that Theorem 19 still applies, i.e. LOP(PSN_*^{1+}) cannot be solved in time $2^{o(v)}$ unless the ETH is false.

7. Time vs. Space in Upper Bounds

The best upper time bounds for hard problems usually assume algorithms that do not run in polynomial space. For instance, the result of Björklund et al. (2009) that Graph Colourability can be solved in time $2^n \cdot \text{poly}(n)$ also requires using space $2^n \cdot \text{poly}(n)$. They also show an upper bound of time $2.2461^n \cdot \text{poly}(n)$ under the additional restriction of polynomial space (Björklund et al., 2009, Proposition 7). Since the proof of Theorem 40 is based on a polynomial reduction from this problem, we immediately get the following result.

Corollary 41. *(To Theorem 40) If* LOP(PSN_*^{1+}) *can be solved in time* $2^{\frac{cv}{2}} \cdot \textit{poly}(v)$ *and polynomial space for some* $c > 0$*, then* Graph Colourability *can be solved in time* $2^{cn} \cdot \textit{poly}(n)$ *and polynomial space.*

We have $2.2461^n = 2.2461^{v/2} = 1.4987^v$, so if we could solve LOP(PSN_*^{1+}) faster than time $1.4987^v \cdot \text{poly}(v)$ using only polynomial space, then there would be a faster polynomial-space algorithm for Graph Colourability than previously known.

There is, thus, often a trade-off between time and space, such that we can choose to use less space at the expense of using more time for finding the solution. This is not only of theoretical interest. For instance, it is quite common in practice that planning and search algorithms run out memory before they find a solution or hit the time limit (if such a limit is set in advance). It would clearly be beneficial in these cases if we could balance time and space in a better way. Another solution to overcome this problem is disk-based search (cf. Korf, 2008), i.e. both the internal memory of the computer and hard disks are used as working memory. Since disks are much slower than internal memory, it is an interesting question if there are algorithms that can trade time for space such that they run in internal memory only while still not being slower than a disk-based algorithm? In practice, it may not be optimal to require that an algorithm runs in polynomial space, but rather very low-order exponential space, but in order to explore the limits of this trade-off, we will focus on planning using only polynomial space.

We know from Section 3.2 that COP(PSN) can be solved in time $4^v \cdot \text{poly}(||\mathbb{P}||)$, using Dijkstra's algorithm, but this result also requires space $4^v \cdot \text{poly}(||\mathbb{P}||)$. Using an implicit representation of the state-transition graph does not help much since we have to store $\Omega(|S|) = \Omega(2^v)$ nodes in the worst case. Most search algorithms, like breadth-first search and A^*, require time $b^h \cdot \text{poly}(||\mathbb{P}||)$ and space $b^h \cdot \text{poly}(||\mathbb{P}||)$, where b is the branching factor and h is the height (or depth) of the explored search tree (cf. Russell & Norvig, 1995)[8]. That is, all these algorithms require exponential space. An exception is depth-first search (DFS), which runs in time $b^h \cdot \mathrm{poly}(||\mathbb{P}||)$ and space $bh \cdot \mathrm{poly}(||\mathbb{P}||)$ (cf. Russell & Norvig, 1995). However, this is still bad in the general case. Since h is the length of a shortest (or cheapest) plan, we get $h = 2^v - 1$ in the worst case, so bh is not polynomial in the instance size.

In this section, we will first show that it is possible to solve even the most general case, i.e. COP(SAS$^+$), in polynomial space, but at the cost of a very generous upper time bound. We will then consider two more restricted cases, COP(PSN$^{*}_{*+}$) and LOP(PSN$^{*+}_{*+}$). Both of these can be solved in time $b^v \cdot \mathrm{poly}(||\mathbb{P}||)$ and polynomial space, using DFS. We will show that the upper time bounds for both these problems can be improved considerably, for moderately large b, by adding some simple pruning heuristics to DFS; COP(PSN$^{*}_{*+}$) can be solved in time $F(v) \cdot \mathrm{poly}(||\mathbb{P}||)$ and polynomial space, while LOP(PSN$^{*+}_{*+}$) can be solved in time $v! \cdot \mathrm{poly}(||\mathbb{P}||)$ and polynomial space. $F(n)$ denotes the nth number in the sequence of *Fubini numbers*, which grows approximately as $2^{\Theta(n \log n)}$.

7.1 General SAS$^+$ Planning in Polynomial Space

We do know that there must exist an algorithm for COP(SAS$^+$) that runs in polynomial space, since the problem is in **PSPACE**, but this does not tell us much about the actual time bounds. Even if using an implicit representation of the state-transition graph, most search algorithms still need an exponential amount of memory. This applies even to depth-first search since there are planning instances with exponentially long shortest solutions. By using Savitch's theorem (Savitch, 1970), the amount of memory can be lowered. Savitch showed that there exists an algorithm $\mathcal{A}_S$ that takes a graph $G = \langle U, E \rangle$ as input and checks whether there exists a path from $u \in U$ to $v \in U$ of length k, or less, using space $O\big(\log^2(|U|)\big)$ and time $|U|^{O(\log k)}$. The time bound did not appear in Savitch's article, but it is a well-known folklore result. The result assumes that we can check whether two vertices are connected or not in polynomial time (in the size of the graph), which is no problem since we can view $\mathbb{P}$ as a compact representation of its state-transition graph, as previously described in Section 3.4.

The following results are straightforward generalisations to SAS$^+$ of previously published results for PSN (Aghighi et al., 2016b). Problem PSat can be solved by asking if there is a plan of length $k = |S| = d^v$, or less, i.e. by solving LOP for this value of k. We can thus solve PSat in time

$$\begin{aligned} |S|^{O(\log k)} \cdot \mathrm{poly}(||\mathbb{P}||) &= (d^{\,v})^{O(\log d^{\,v})} \cdot \mathrm{poly}(||\mathbb{P}||) = (2^{v \log d})^{O(\log 2^{v \log d})} \cdot \mathrm{poly}(||\mathbb{P}||) \\ &= (2^{v \log d})^{O(v \log d)} \cdot \mathrm{poly}(||\mathbb{P}||) = 2^{O(v^2 \log^2 d)} \cdot \mathrm{poly}(||\mathbb{P}||) \end{aligned}$$

using space

$$O(\log^2 |S|) = O(\log^2 d^v) = O(\log^2 2^{v \log d}) = O(v^2 \log^2 d)$$

Savitch's theorem is clearly also useful for problem LOP, since checking whether there exists a plan of length k or less takes time $d^{\,O(v \log k)} = 2^{O(v \log d \log k)}$ and uses space

8. The polynomial factor $\mathrm{poly}(||\mathbb{P}||)$ is usually omitted in the literature, but it is necessary in the general case where it is not necessarily dominated by the factor b^h.

$O(v^2 \log^2 d)$, which can be substantially better than solving Psat when k is moderately large. In the case of COP, the bounds for LOP apply if all actions have cost 1 or more, since $|\omega| \leq c(\omega)$ must hold for all plans ω, and otherwise the bounds for Psat apply.

We conclude by noting that the polynomial factor $\mathrm{poly}(||\mathbb{P}||)$ that we have used to cover the time for checking the action set of an instance $\mathbb{P}$ is sufficient also for verifying actions and adding action costs. Note that Savitch's theorem has been repeatedly applied to planning in the literature for proving membership in **PSPACE**. However, to the best of our knowledge, it has never been used to derive explicit bounds on time and space for planning before the work of Aghighi et al. (2016b).

7.2 COP for Monotone PSN with Arbitrary Preconditions

We will now consider the more restricted class of PSN^{*}_{*+}, i.e. monotone PSN. We know from Proposition 1.3 that no action is required more than once in a length-optimal or shortest cost-optimal plan. Furthermore, such a plan cannot contain any redundant actions, so every action sets at least one variable. Hence, the search depth can be restricted to v, so if we let DFS explore the full search tree to depth v, it will be able to find a shortest (or a cheapest) plan. It follows that DFS can solve COP for monotone PSN in time $b^v \cdot \mathrm{poly}(||\mathbb{P}||)$ and space $bv \cdot \mathrm{poly}(||\mathbb{P}||)$. This time bound is still very bad unless b is small and the space bound is polynomial in $||\mathbb{P}||$ only if b is polynomial in $||\mathbb{P}||$. We will show that with a simple heuristic for pruning the search tree, DFS can solve this problem much faster for moderately large branching factors and in polynomial space.

We note that if an instance is monotone, then every variable that is true in the initial state will remain true throughout a plan. Hence, an action with a negative precondition that is not satsified in the initial state can never occur in a plan, so we can assume all such actions are removed from the instance. Furthermore, we can always assume that the initial state is the empty set, since all variables that are true in the inital state will be true also in the goal state, and they are thus redundant.

We will start by investigating some connections between state sequences and plans for monotone instances. First recall that a *partition* of a set S is a set $P \subseteq 2^S$ of pairwise disjoint non-empty subsets of S such that P covers S, i.e. $\cup_{X \in P} X = S$. The elements in P are called *parts*. An *ordered partition* of S is a sequence $X_1, \ldots, X_n$ of subsets of S such that $\{X_1, \ldots, X_n\}$ is a partition of S, i.e. an ordered partition is a partition with an additional total order on the parts. The Stirling number $\left\{ {n \atop k} \right\}$ of the second kind denotes the number of ways we can partition a set of size n into k parts. Since each partition of size k can be ordered in $k!$ different ways, the total number of ordered partitions of all sizes of a set with n elements is $F(n) = \sum_{k=0}^{n} k! \left\{ {n \atop k} \right\}$, which is known as the nth *Fubini number* (or the nth *ordered Bell number*).

Let $\sigma = s_0, \ldots, s_\ell$ be a strictly monotone state sequence. We define the *difference sequence* $\delta = d_1, \ldots, d_\ell$ of σ such that $d_i = s_i \setminus s_{i-1}$ for all i ($1 \leq i \leq \ell$). Then all sets in δ are non-empty and pairwise disjoint and $d_1 \cup \ldots \cup d_\ell = s_\ell \setminus s_0$, so δ is an ordered partition of $s_\ell \setminus s_0$. In particular, if $s_0 = \varnothing$, then δ is an ordered partition of s_ℓ. If we know the value of s_0, then we can trivially reconstruct σ from δ, so a strictly monotone state sequence and its corresponding difference sequence are equivalent in this respect. Furthermore, if $d'_1, \ldots, d'_m$ is an ordered partition of $V \setminus s_\ell$, then $d_1, \ldots, d_\ell, d'_1, \ldots, d'_m$ is an ordered partition of $V \setminus s_0$.

That is, the difference sequence for a plan can always be extended to an ordered partition of $V \setminus s_0$, i.e. an ordered partition of V if $s_0 = \varnothing$.

We can obviously solve length-optimal planning for a PSN^*_{*+} instance $\mathbb{P} = \langle V, A, \varnothing, s_G \rangle$ as follows. Enumerate all ordered partitions of V. For each such partition $d_1, \ldots, d_\ell$, construct the corresponding state sequence $s_0, \ldots, s_\ell$, which must be strictly monotone. Check if there are actions $a_1, \ldots, a_n$, for some $n \leq \ell$, such that $a_1, \ldots, a_n$ is a plan from s_0 to s_n and $s_G \sqsubseteq s_n$. By choosing the smallest such value for n and keeping track of the shortest plan found for any partition, we can find a length-optimal plan. This runs in polynomial time for each partition, so the whole algorithm runs in time $F(v) \cdot \mathrm{poly}(||\mathbb{P}||)$ and polynomial space. We can even find a cost-optimal plan by always choosing a cheapest possible action in each step. In the earlier conference version of this article (Bäckström & Jonsson, 2016), we presented this algorithm and demonstrated that it has a tighter upper bound than the standard bound $O\big(b^v \cdot \mathrm{poly}(||\mathbb{P}||)\big)$ for DFS, unless the branching factor is small. We will now show that there is no need for such special algorithms, since the same bound can be proven directly using DFS.

Let $\mathbb{P} = \langle V, A, s_I, s_G \rangle$ be a PSN instance. The DFS search tree for $\mathbb{P}$ consists of a set of search nodes and a set of edges that form a directed out-tree. Each node contains a state and each edge is labelled with an action. Define a successor function Succ such that for every state s, $\mathrm{Succ}(s)$ contains all successors of s, i.e. all tuples $\langle a, t \rangle$ such that t is a state and a is an action from s to t. The search tree can be defined recursively as follows: Let the root node N_0 contain the initial state s_I. For each node N with associated state s, define a new node N' with state t and an edge $\langle N, a, N' \rangle$ for each tuple $\langle a, t \rangle \in \mathrm{Succ}(s)$. While all search nodes are unique, several nodes may contain the same state. Note that there is a one-to-one correspondence between the branches in the search tree and the possible walks in the state-transition graph, starting at s_I.

Since we will only consider the special case of monotone PSN, it holds that $s \subseteq t$ whenever $\langle a, t \rangle \in \mathrm{Succ}(s)$. In general, the search tree is infinite and the search algorithm implements a goal test, for terminating recursion, and possibly also pruning heuristics. In order to simplify the proofs, we will instead let the successor function implement these directly, resulting in a finite search tree, and we will see that it is safe to make the following assumptions for the cases we consider:

A1: If $s_G \sqsubseteq s$, then $\mathrm{Succ}(s) = \varnothing$.

A2: if $\langle a, t \rangle \in \mathrm{Succ}(s)$, then $s \subset t$,

A3: For all states s and t, there is at most one action a such that $\langle a, t \rangle \in \mathrm{Succ}(s)$.

A1 implements the goal test, by forcing a branch to terminate whenever reaching a state that satisfies the goal. This implies that every branch in the search tree can contain at most one goal state, which must then be a leaf. This is safe since we can never find a shorter or cheaper plan by adding more actions to a plan. A2 says that we only consider actions that change some variable, i.e. we ignore loops in the state-transition graph. This is safe since loops correspond to redundant actions, that can never make a plan shorter or cheaper. A3 is the assumption that we never need to consider more than one action for any edge in the state-transition graph, which is safe for our purposes. For length-optimal planning, it does not matter which action we choose, and for cost-optimal planning, we can choose any cheapest action. These assumptions make the successor function systematic in the sense that no two branches in the search tree can contain the same state sequence, as part 2 of the following lemma states.

Lemma 42. *Let $\mathbb{P}$ be a PSN^*_{*+} instance and let T be the search tree for $\mathbb{P}$ generated by a succesor function Succ satisfying assumptions A1–A3. Then*

1. *The sequence of states along any branch in T is strictly monotone and*
2. *Every branch in T has a unique state sequence.*

Proof. 1. Let $N_0, \ldots, N_m$ be the search nodes along some branch in T, where N_0 must be the root node of the search tree. Let $s_0, \ldots, s_m$ be the corresponding states associated with the nodes and let $a_1, \ldots, a_m$ be the labels of the edges along the branch. Then for all i ($1 \leq i \leq m$), it holds that $\langle a_i, s_i \rangle \in \mathrm{Succ}(s_{i-1})$ and, thus, also that $s_{i-1} \subset s_i$ by assumption A2.

2. Let $\eta = N_0, \ldots, N_m$ and $\eta' = N'_0, \ldots, N'_n$ be two distinct branches in T. Let $\sigma = s_0, \ldots, s_m$ be the states associated with the nodes in η and let $\sigma' = s'_0, \ldots, s'_n$ be the states associated with the nodes in η'. Suppose $\sigma = \sigma'$. Then $m = n$. Let i be the maximum index such that $N_i = N'_i$. There is always such a choice of i since $N_0 = N'_0$ must be the root node of T. It follows that $N_0, \ldots, N_i = N'_0, \ldots, N'_i$ since T is an out-tree, i.e. the two branches must share exactly the first $i+1$ nodes. It must also be the case that $i < m$, since the two branches are distinct. We have $s_{i+1} = s'_{i+1}$ by assumption, so it must be the case that $\mathrm{Succ}(s_i)$ contains two tuples $\langle a_{i+1}, s_{i+1} \rangle$ and $\langle a'_{i+1}, s_{i+1} \rangle$ such that $a_{i+1} \neq a'_{i+1}$ since $N_{i+1} \neq N'_{i+1}$. However, this contradicts assumption A3, so it follows that $a_{i+1} = a'_{i+1}$ and, thus, that $N_{i+1} = N'_{i+1}$, which contradicts the choice of i. It follows that $\sigma \neq \sigma'$. This proof is illustrated in Figure 3. □

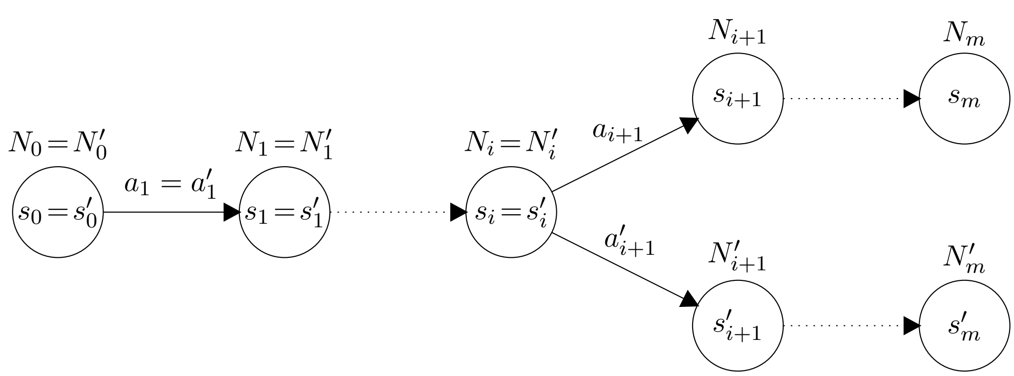

Figure 3: Illustration of the proof of Lemma 42.2.

We see from the proof of property 2 of this lemma that the two nodes N_{i+1} and N'_{i+1}, that are both successsors to the split point at node $N_i = N'_i$, cannot contain the same state. However, nodes further down the two branches may do so. For instance, it could be the case that $a_{i+1} = a'_{i+2}$ and $a'_{i+1} = a_{i+2}$. Then both a_{i+1}, a_{i+2} and a'_{i+1}, a'_{i+2} must be plans from

$s_i = s'_i$ to s_{i+2}, which implies that $s'_{i+2} = s_{i+2}$. Hence, in the proof above (and in Figure 3) it could be the case that only s_{i+1} and s'_{i+1} differ and $s_{i+2}, \ldots, s_m = s'_{i+2}, \ldots, s'_m$, i.e. the two branches differ in only one position. Since it still holds that any two different branches must differ in at least one position, we are guaranteed that no two branches contain the same state sequence. Because of property 2 in this lemma, we can also ignore the actual search nodes of a branch and identify it uniquely by its state sequence. That is, we will refer to a branch in the search tree as a sequence $s_0, \ldots, s_\ell$ of states with an associated sequence $a_1, \ldots, a_\ell$ of actions, meaning that $\langle a_i, s_i \rangle \in \mathrm{Succ}(s_{i-1})$ for all i ($1 \leq i \leq \ell$).

For cost-optimal PSN^*_{*+} planning, we will consider the following successor function which satisfies assumptions A1–A3.

Succ_1: For every state s, $\mathrm{Succ}_1(s)$ is defined as follows: If $s_G \sqsubseteq s$, then $\mathrm{Succ}_1(s) = \varnothing$. Otherwise, for every state t such that $s \subset t$, if A contains some action from s to t, then $\mathrm{Succ}_1(s)$ contains $\langle a, t \rangle$ for an arbitrary cheapest such action a.

Example 43. *Consider the following five actions: $a_1 : \{v_1\} \Rightarrow \{v_2\}$, $a_2 : \{v_1\} \Rightarrow \{v_3\}$, $a_3 : \{v_2\} \Rightarrow \{v_3\}$, $a_4 : \{v_1\} \Rightarrow \{v_4\}$, $a_5 : \{v_2\} \Rightarrow \{v_4\}$. Define the cost function c as $c(a_1) = c(a_2) = c(a_3) = c(a_4) = 1$ and $c(a_5) = 2$. Let $s = \{v_1, v_2\}$ and consider the set $\mathit{Succ}_1(s)$. All five actions are applicable in s. Action a_1 results in state s, so $\langle a_1, s \rangle$ is not in $\mathit{Succ}_1(s)$. Both of the actions a_2 and a_3 result in state $\{v_1, v_2, v_3\}$, so either of $\langle a_2, \{v_1, v_2, v_3\} \rangle$ and $\langle a_3, \{v_1, v_2, v_3\} \rangle$ must be in $\mathit{Succ}_1(s)$, but not both. The choice is arbitrary since a_2 and a_3 have the same cost. Both the actions a_4 and a_5 result in state $\{v_1, v_2, v_4\}$. However, since a_5 has a higher cost than a_4, we must choose to include $\langle a_4, \{v_1, v_2, v_4\} \rangle$ but not $\langle a_5, \{v_1, v_2, v_4\} \rangle$ in $\mathit{Succ}_1(s)$. Hence, we can arbitrarily choose $\mathit{Succ}_1(s)$ as either $\{\langle a_2, \{v_1, v_2, v_3\} \rangle, \langle a_4, \{v_1, v_2, v_4\} \rangle\}$ or $\{\langle a_3, \{v_1, v_2, v_3\} \rangle, \langle a_4, \{v_1, v_2, v_4\} \rangle\}$.*

The following lemma shows that even though the search tree generated by Succ_1 may not contain all cost-optimal plans for a solvable instance, it will contain at least one of them, which is sufficient for solving cost-optimal planning.

Lemma 44. *Let $\mathbb{P}$ be a PSN^*_{*+} instance. If $\mathbb{P}$ is solvable, then the search tree generated by successor function Succ_1 contains a branch with a cost-optimal plan for $\mathbb{P}$.*

Proof. Suppose $\mathbb{P}$ is solvable. Let $\omega^* = a^*_1, \ldots, a^*_\ell$ be a shortest cost-optimal plan for $\mathbb{P}$ and let $\sigma = s_0, \ldots, s_\ell$ be the state sequence of ω^*. It must then hold that $s_G \sqsubseteq s_\ell$, but $s_G \not\sqsubseteq s_{\ell-1}$, and also that σ is strictly monotone, according to Proposition 1.2. It further follows from the definition of Succ_1 that for each i ($1 \leq i \leq \ell$), $\mathrm{Succ}_1(s_{i-1})$ contains $\langle a_i, s_i \rangle$ for some action a_i from s_{i-1} to s_i such that $c(a_i) \leq c(a^*_i)$. Hence, the search tree contains a branch with state sequence σ and a plan $\omega = a_1, \ldots, a_\ell$ from s_0 to s_ℓ, i.e. ω is a plan for $\mathbb{P}$, such that $c(\omega) \leq c(\omega^*)$. However, since ω^* is cost-optimal by assumption, it must be the case that $c(\omega) = c(\omega^*)$, i.e. ω is a cost-optimal plan for $\mathbb{P}$. □

Let $\sigma = s_0, \ldots, s_\ell$ be a state sequence over a variable set V such that $s_0 = \varnothing$ and let $\delta = d_1, \ldots, d_\ell$ be the difference sequence of σ. We define the corresponding *augmented difference sequence* δ_A as $\delta_A = d_1, \ldots, d_\ell$ if $s_\ell = V$ and otherwise as $\delta_A = d_1, \ldots, d_\ell, d'$, where $d' = V \setminus s_\ell$. We note that δ_A is always an ordered partition of V and that two different state sequences can never have the same augmented difference sequence.

Theorem 45. *DFS with successor function $Succ_1$ solves cost-optimal planning for* PSN^*_{*+} *in time $F(v) \cdot poly(||\mathbb{P}||)$ and space $poly(||\mathbb{P}||)$.*

Proof. Let $\mathbb{P}$ be a PSN^*_{*+} instance. Generate the whole search tree T for $\mathbb{P}$ using DFS with successor function Succ_1. This is possible since T is finite. Keep track of the cheapest plan for $\mathbb{P}$ found in any branch and return this plan. Reject if no plan was found. It follows from Lemma 44 that this method is correct.

Every branch has a unique state sequence according to Lemma 42. Hence, every branch also has a unique augmented difference sequence. Since every augmented difference sequence is an ordered partition of V, there cannot be more branches in T than there are ordered partitions of V, i.e. there can be at most $F(v)$ branches. The set of successors of a node can be computed in polynomial time and space in the instance size. Since we use DFS we only need to keep track of the nodes along the current branch, there are at most v such nodes, and the cheapest plan found, which is of length v at most. Hence, this algorithm runs in time $F(v) \cdot \mathrm{poly}(||\mathbb{P}||)$ and space $\mathrm{poly}(||\mathbb{P}||)$. □

The standard upper bound of time $O\big(b^v \cdot \mathrm{poly}(||\mathbb{P}||)\big)$ for DFS still holds, so we will compare this bound with our new bound. Let the polynomial p be an upper bound on the time spent in each node. Then the standard upper bound is time $O\big(b^v \cdot p(||\mathbb{P}||)\big)$ and the new upper bound is time $O\big(F(v) \cdot p(||\mathbb{P}||)\big)$. We will now show that the new bound is tighter unless the branching factor is very small. We are only interested in an estimate of when this occurs, so for large values of v, we can ignore constant factors, that is, we want to find the value of b such that $b^v \cdot p(v) < F(v) \cdot p(v)$. Note that p is the same polynomial in both cases. The actual polynomial is implementation dependent, but for any fixed implementation we have the same polynomial in both bounds. Hence, the actual polynomial is irrelevant so we want to find when $b^v < F(v)$.

For large n, $F(n)$ can be approximated as $\frac{n!}{2(\ln 2)^{n+1}}$ (Sprugnoli, 1994) and $n!$ can be approximated as $\sqrt{2\pi n}(\frac{n}{e})^n$ (using Stirling's formula). Combining these, we can approximate $F(v)$ as

$$\frac{\sqrt{2\pi v}(\frac{v}{e})^v}{2(\ln 2)^{v+1}} = \frac{\sqrt{2\pi}v^{\frac{1}{2}}(\frac{v}{e\ln 2})^v}{2(\ln 2)} = \frac{\sqrt{2\pi}}{2(\ln 2)}v^{\frac{1}{2}}(\frac{v}{e\ln 2})^v$$
$$= \frac{\sqrt{2\pi}}{2(\ln 2)}2^{\frac{1}{2}\log v}2^{v\log\frac{v}{e\ln 2}} = \frac{\sqrt{2\pi}}{2(\ln 2)}2^{\frac{1}{2}\log v + v\log\frac{v}{e\ln 2}}$$

Since $b^v = 2^{v\log b}$, we get that $b^v < F(v)$ approximately when

$$2^{v\log b} < 2^{\frac{1}{2}\log v + v\log\frac{v}{e\ln 2}}$$

i.e. when

$$v\log b < \frac{1}{2}\log v + v\log\frac{v}{e\ln 2}.$$

For large v, this occurs approximately when $b < \frac{v}{e\ln 2} \approx 0.53v$.

The worst possible branching factor is $b = 2^v - 1$, which occurs if there is an action to every successor node of the initial state. In this case, we get $b^v = (2^v - 1)^v$, i.e. $b^v \in \Theta(2^{v^2})$, which is much larger than $F(v)$. This case is somewhat extreme, but even if the instance size is polynomially restricted in v, we could have a branching factor that is polynomial in v, i.e. much larger than $0.53v$. Furthermore, even if the actual branching factor is small, it can be difficult to determine a tight bound for it a priori. In contrast, our new upper bound is independent of the branching factor and can be determined directly from the instance. We finally note that the new time bound is still much worse than if using Dijkstra's algorithm, since $F(v)$ is approximately in $2^{\Theta(v \log v)}$.

7.3 LOP for Monotone PSN with Positive Preconditions

We can modify the previous method to show an even better upper bound for the case where also the preconditions are positive.

Let s and t be states. Then t is a *subset-maximal result* of s if (1) $s \neq t$, (2) there is some action a from s to t and (3) there is no action a' from s to some state t' such that $t \subset t'$. Let $\omega = a_1, \ldots, a_\ell$ be a plan from s_0 to s_ℓ with state sequence $s_0, \ldots, s_\ell$. Then ω is a *state-maximal plan* if s_i is a subset maximal result of s_{i-1} for all i ($1 \leq i \leq \ell$).

Lemma 46. *Let $\mathbb{P}$ be a PSN^{*+}_{*+} instance. If $\mathbb{P}$ has a plan of length ℓ from a state s to a state t, then there is a state-maximal plan of length ℓ, or less, from s to some state t' such that $t \subseteq t'$.*

Proof. Let s and t be two arbitrary states and let $\omega = a_1, \ldots, a_\ell$ be a plan from s to t. Let $s_0, \ldots, s_\ell$ be the state sequence of ω, i.e. $s_0 = s$ and $s_\ell = t$. Proof by induction over the plan length.

Base case: If ω is the empty plan, then $s_\ell = s_0$. This plan is already state maximal since we cannot reach any other state without using at least one action.

Induction: Suppose the claim holds for all plans of length k, for some $k > 0$. Let $\ell = k + 1$. There are two cases:

(1) Suppose that $s_0 = s_1$. Then $a_2, \ldots, a_\ell$ is a plan from s_0 to s_ℓ of length $\ell - 1 = k$, so it follows from the induction hypothesis that there is a state-maximal plan from s_1 to some state t such that $s_\ell \subseteq t$. This plan must then also be a state-maximal plan from s_0 to t.

(2) Instead suppose that $s_0 \neq s_1$. Then it must hold that $s_0 \subset s_1$, so there must be some action a'_1 from s_0 to some state t_1 such that $s_1 \subseteq t_1$ and t_1 is a subset-maximal result of s_0. Since $\mathrm{pre}(a_2) \subseteq s_1$ and $s_1 \subseteq t_1$, it follows that $a_2, \ldots, a_\ell$ is a plan from t_1 to some state t_ℓ with state sequence $t_1, \ldots, t_\ell$ such that $s_i \subseteq t_i$ for all i ($1 \leq i \leq \ell$). It thus follows from the induction hypothesis that there is a state-maximal plan $\omega' = a'_2, \ldots, a'_m$ from t_1 to some state t' such that $t_\ell \subseteq t'$ and $m \leq \ell$. Hence, $a'_1, \ldots, a'_m$ is a state-maximal plan from s_0 to t', where $s_\ell \subseteq t'$ and $m \leq \ell$. This case is illustrated in Figure 4. □

We will use the following successor function for length-optimal PSN^{*+}_{*+} planning.

Succ$_2$: For every node s in the search tree, $\mathrm{Succ}_2(s)$ is defined as follows: If $s_G \sqsubseteq s$, then $\mathrm{Succ}_2(s) = \varnothing$. Otherwise, for every state t that is a subset maximal result of s, let $\mathrm{Succ}_2(s)$ contain $\langle a, t \rangle$ for an arbitrary action from s to t.

Example 47. *Consider the following four actions: $a_1 : \{v_1\} \Rightarrow \{v_3\}$, $a_2 : \{v_1\} \Rightarrow \{v_3, v_4\}$, $a_3 : \{v_2\} \Rightarrow \{v_3, v_4\}$ and $a_4 : \{v_1\} \Rightarrow \{v_5\}$. Let $s = \{v_1, v_2\}$ and conisder the set Succ$_2(s)$. All four actions are applicable in s. Action a_1 results in state $\{v_1, v_2, v_3\}$, actions a_2 and a_3 both*

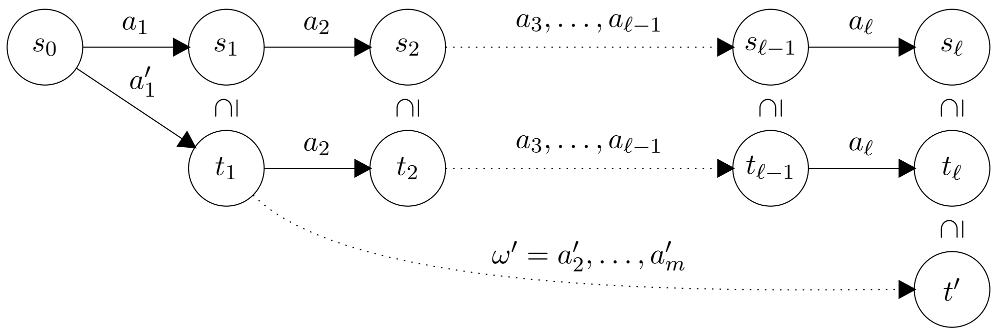

Figure 4: Illustration of case 2 in the induction step in the proof of Lemma 46.

result in state $\{v_1, v_2, v_3, v_4\}$ *and action* a_4 *results in state* $\{v_1, v_2, v_5\}$*. State* $\{v_1, v_2, v_3\}$ *is not a subset-maximal result of* s *since* $\{v_1, v_2, v_3\} \subseteq \{v_1, v_2, v_3, v_4\}$*. Neither of the states* $\{v_1, v_2, v_3, v_4\}$ *and* $\{v_1, v_2, v_5\}$ *is a subset of the other, so they are both subset-maximal results of* s*. We can arbitrarily choose* $Succ_2(s)$ *as either* $\{\langle a_2, \{v_1, v_2, v_3, v_4\}\rangle, \langle a_5, \{v_1, v_2, v_5\}\rangle\}$ *or* $\{\langle a_3, \{v_1, v_2, v_3, v_4\}\rangle, \langle a_5, \{v_1, v_2, v_5\}\rangle\}$*.*

We note that Succ_2 satisfies assumptions A1–A3, so Lemma 42 holds also for Succ_2. Also note that if $\mathrm{Succ}_2(s)$ contains both $\langle a, t\rangle$ and $\langle a', t'\rangle$, then neither $t \subseteq t'$ nor $t' \subseteq t$ hold since both t and t' are subset maximal results of s.

The following lemma shows that even though the search tree generated by Succ_2 may not contain all length-optimal plans for a solvable instance, it will contain at least one of them, which is sufficient for solving length-optimal planning.

Lemma 48. *Let* $\mathbb{P}$ *be a* PSN^{*+}_{*+} *instance. If* $\mathbb{P}$ *is solvable, then the search tree for* $\mathbb{P}$ *generated by successor function* $Succ_2$ *contains a branch with a length-optimal plan for* $\mathbb{P}$*.*

Proof. Suppose $\mathbb{P}$ is solvable and let T be the search tree for $\mathbb{P}$ generated by Succ_2. Let $\omega^* = a^*_1, \ldots, a^*_\ell$ be a length-optimal plan for $\mathbb{P}$, with state sequence $\sigma^* = s^*_0, \ldots, s^*_\ell$. It follows from Lemma 46 that there is a state-maximal plan $\omega = a_1, \ldots, a_n$ from $s_0 = s^*_0$ to some state s_n such that $n \leq \ell$ and $s^*_\ell \subseteq s_n$. Let $\sigma = s_0, \ldots, s_n$ be the state sequence of ω. Since ω is state-maximal, it holds for all i $(1 \leq i \leq n)$ that s_i is a subset maximal result of s_{i-1}, i.e. there is some a'_i such that $\langle a'_i, s_i\rangle \in \mathrm{Succ}_2(s_{i-1})$. It follows that some branch in T contains the state sequence σ and the plan $\omega' = a'_1, \ldots, a'_n$, which is a plan from s_0 to s_n and, thus, also a plan for $\mathbb{P}$ since $s_G \subseteq s^*_\ell \subseteq s_n$. Furthermore, it must be the case that $n = \ell$ since ω^* was assumed length optimal, so ω' is a length-optimal plan for $\mathbb{P}$. □

Let $V = \{v_1, \ldots, v_n\}$ and let $\delta = d_1, \ldots, d_m$ be an ordered partition of V. We say that a permutation π of the index set $\{1, \ldots, n\}$ *respects* δ if the following holds: For all i, j $(1 \leq i, j \leq m)$, all $v_x \in d_i$ and all $v_y \in d_j$, if $i < j$, then $\pi(x) < \pi(y)$. We define $\tau(\delta)$ as the set of all permutations of V that respect δ. Note that $\tau(\delta)$ must be non-empty.

Example 49. *Let* $V = \{v_1, \ldots, v_5\}$ *and let* $\delta = d_1, d_2, d_3$ *be an ordered partition of* V *such that* $d_1 = \{v_3, v_5\}$*,* $d_2 = \{v_1, v_4\}$ *and* $d_3 = \{v_2\}$*. Then the permutation* v_5, v_3, v_1, v_4, v_2 *respects* δ*, but the permutation* v_3, v_1, v_5, v_4, v_2 *does not respect* δ *since* $v_5 \in d_1$ *and* $v_1 \in d_2$ *but* v_1 *is ordered before* v_5*.*

We can now prove the following upper bound.

Theorem 50. *DFS with successor function $Succ_2$ solves length-optimal planning for* PSN^{*+}_{*+} *in time* $v! \cdot poly(||\mathbb{P}||)$ *and space* $poly(||\mathbb{P}||)$.

Proof. Let $\mathbb{P}$ be a PSN^{*+}_{*+} instance. Generate the whole search tree T for $\mathbb{P}$ using DFS with successor function Succ_2. This is possible since T is finite. Keep track of the shortest plan for $\mathbb{P}$ found in any branch and return this plan. Reject if no plan was found. Lemma 48 guarantees that this method is correct.

Let $\sigma = s_0, \ldots, s_m$ and $\sigma' = s'_0, \ldots, s'_n$ be the states along two arbitrary distinct branches in the search tree. Then $\sigma \neq \sigma'$ according to Lemma 42. Obviously, $s_0 = s'_0$ since both branches start at the root node. Let i be the maximal index such that $s_0, \ldots, s_{i-1} = s'_0, \ldots, s'_{i-1}$. It is not possible that $m = n = i - 1$ since $\sigma \neq \sigma'$. Hence, $\mathrm{Succ}_2(s_{i-1}) = \mathrm{Succ}_2(s'_{i-1})$ must be non-empty, so no branch can stop at the node with state $s_{i-1} = s'_{i-1}$. It follows that $m \geq i$, $n \geq i$ and $s_i \neq s'_i$. That is, $s_{i-1} = s'_{i-1}$ is the branching point after which the two branches start to differ. Let $\delta = d_1, \ldots, d_m$ be the augmented difference sequence for σ and let $\delta' = d'_1, \ldots, d'_n$ be the augmented difference sequence for σ', i.e. both are ordered partitions of V. Then $d_1, \ldots, d_{i-1} = d'_1, \ldots, d'_{i-1}$. The successor function Succ_2 guarantees that $s_i \not\subseteq s'_i$ and $s'_i \not\subseteq s_i$, so it follows that $d_i \not\subseteq d'_i$ and $d'_i \not\subseteq d_i$. Without losing generality, assume $|d_i| \leq |d'_i|$. Define $k = |d_1| + \cdots + |d_{i-1}| = |d'_1| + \cdots + |d'_{i-1}|$. Let π be any permutation in $\tau(\delta)$ and let π' be any permutation in $\tau(\delta')$. There must be some variable $v_x \in d_i$ such that $v_x \notin d'_i$, since $d_i \not\subseteq d'_i$. Hence, π must satisfy that $k < \pi(x) \leq k + |d_i|$. Obviously, $v_x \in d'_j$ for some $j > i$, since $d_1, \ldots, d_{i-1} = d'_0, \ldots, d'_{i-1}$, so π' must satisfy that $\pi'(x) > k + |d'_i| \geq k + |d_i| \geq \pi(x)$. Hence, π and π' cannot be the same permutation. Since π and π' were chosen arbitrarily it further follows that $\tau(\sigma)$ and $\tau(\sigma')$ do not overlap. Since also σ and σ' were chosen arbitrarily, it follows that no permutation can respect more than one branch in the tree. We thus conclude that there can be at most $v!$ branches in the tree since there are $v!$ permutations of $\{1, \ldots, v\}$. It follows that DFS with Succ_2 runs in time $v! \cdot \mathrm{poly}(||\mathbb{P}||)$ and space $\mathrm{poly}(||\mathbb{P}||)$. $\square$

An analogous analysis to the one for DFS with Succ_1 yields that the new upper bound for DFS with Succ_2 is tighter than the standard upper bound for DFS approximately when $b < 0.37v$.

8. Discussion

We have presented a variety of results concerning the time and space complexity of propositional planning during the course of this article. Most of the results bring up issues that warrant further discussion and suggest directions for future research. We discuss some of these issues below.

For problems that are not solvable in polynomial time, one usually resorts to alternative methods, like polynomial-time approximation algorithms or heuristic search, hoping that this will perform satisfactorily in practice. However, with modern computers it is becoming increasingly popular to consider also algorithms running in superpolynomial time. Preferrably, such an algorithm should still run in subexponential time. It is then interesting to know whether such an algorithm can exist or not, thus asking for the type of lower-bound results we derive in this paper. Examples of planning classes that are solvable in subexponential time and not likely to be polynomial-time solvable are identified in an article by Bäckström and Jonsson (2013, Thm. 24). There are also cases in the literature where one considers algorithms, and even approximation algorithms, that require low-order exponential time (Cygan, Kowalik, & Wykurz, 2009). In such cases, the performance is very sensitive to the constant in the exponent. It is thus important to find restricted planning classes where we can lower the constant even more than in Theorem 7.

We believe that studying the time complexity of monotone planning is important, and we find it highly conceivable that monotone planning is substantially easier than general planning despite the fact that our upper bounds are not so well separated (time $3^v \cdot \text{poly}(||\mathbb{P}||)$ for monotone PSN vs. time $4^v \cdot \text{poly}(||\mathbb{P}||)$ for general PSN) . We base this suspicion on the following fact: we have no strong lower bounds whatsoever for monotone PSN while we have almost matching upper and lower bounds for large fragments of general PSN.

While upper-bound results sometimes take also space into account, lower-bound results generally refer to time only, making no additional restrictions on space. Having seen in Sec. 7 how additional space restrictions can affect the upper bound, it is an obvious question to ask if additional space bounds could also strengthen the lower-bound results upwards. Another but related question is whether the DFS-based upper bounds for monotone planning presented in Sec. 7 can be improved. As pointed out above, we do not have any lower bound results that preclude this. One possible way of improving these algorithms is to reduce the search space by exploiting partial order reduction techniques (Wehrle & Helmert, 2012; Wehrle, Helmert, Alkhazraji, & Mattmüller, 2013).

We have demonstrated that PSAT has approximately the same time complexity as SAT for large classes of planning instances. One should be aware that this primarily applies to the decision problems: if we also ask for a solution, then the two problems may behave very differently. The reason for this is the size of the output. A solvable instance of SAT with n variables always has a solution of size $O(n)$. Solvable instances of PSAT, on the other hand, may have solutions as large as $\Theta(2^v \cdot \log a)$ if we represent solutions in the natural way. If we allow 'non-standard' representations of plans, then this gap may be decreased. Consider, for instance, the class 3S of PSN instances (Jonsson & Bäckström, 1998). This class has the property that it is possible to decide in polynomial time if an instance is solvable, but the shortest plans may be of exponential length in the number of variables. While this may look as preventing the possibility of generating plans in polynomial time, this is not the case: Jonsson (2009) proved that it is possible to generate polynomial-size macro representations of plans for the 3S class in polynomial time. Thus, it may be relevant to consider different compact plan representations for different purposes and the amount of time needed for generating plans in a particular representation. It is not likely that useful compact representations exist in the general case but they do exist in certain restricted cases. General discussions of such representations can be found in the literature (Bäckström & Jonsson, 2012; Bäckström, Jonsson, & Jonsson, 2012).

Our analysis of LOP in Section 5 is similar in spirit to recent analyses of lower bounds for constraint satisfaction problems (De Haan et al., 2015). It is interesting to note that they prove a case where CSP can be solved in time $2^{o(n)}$ if $m \in o(n)$, but cannot be solved in time $2^{o(n)}$ if $m \in \Omega(n)$ and the ETH holds, where n is the number of variables and m the number of constraint tuples. Although there are no immediate connections, this is a sharp easy-hard transition of the same type as indicated by our results in Section 5. Obviously the ratio a/v is crucial here. This has similarities to the phenomenon of phase transitions for **NP**-complete problems, which was pioneered by Cheeseman, Kanefsky, and Taylor (1991) and has remained an active research area ever since. For instance, in the case of k-SAT, the phase transition occurs at a particular value of the ratio m/n for each k, such that instances around this ratio are likely to be hard and the probability of hard instances is very low for other values of the ratio. While the vast majority of work in this area has been empirical, the exact values of the phase transitions for k-SAT have been determined analytically (Achlioptas & Moore, 2006). There are a number of phase transition results for planning described in the literature (Bylander, 1996; Rintanen, 2004; Cohen & Beck, 2017). However, these are all transitions of the type easy-hard-easy. Our transition is of the type easy-hard and is, thus, more similar to the type of transitions for resolution proofs studied by Achlioptas, Beame, and Molloy (2004).

Acknowledgements

Bäckström is partially supported by the Swedish Research Council (VR) under grant 621-2014-4086.

References

Achlioptas, D., Beame, P., & Molloy, M. S. O. (2004). A sharp threshold in proof complexity yields lower bounds for satisfiability search. *J. Comput. Syst. Sci.*, *68*(2), 238–268.

Achlioptas, D., & Moore, C. (2006). Random k-SAT: Two moments suffice to cross a sharp threshold. *SIAM J. Comput.*, *36*(3), 740–762.

Aghighi, M., & Bäckström, C. (2015). Cost-optimal and net-benefit planning - A parameterised complexity view. In *Proc. 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina*, pp. 1487–1493.

Aghighi, M., & Bäckström, C. (2016). A multi-parameter complexity analysis of cost-optimal and net-benefit planning. In *Proc. 26th International Conference on Automated Planning and Scheduling, (ICAPS 2016), London, UK.*, pp. 2–10.

Aghighi, M., Bäckström, C., Jonsson, P., & Ståhlberg, S. (2016a). Analysing approximability and heuristics in planning using the exponential-time hypothesis. In *Proc. 22nd European Conference on Artificial Intelligence (ECAI 2016), The Hague, The Netherlands*, pp. 184–192.

Aghighi, M., Bäckström, C., Jonsson, P., & Ståhlberg, S. (2016b). Refining complexity analyses in planning by exploiting the exponential time hypothesis. *Ann. Math. Artif. Intell.*, *78*(2), 157–175.

Aghighi, M., Jonsson, P., & Ståhlberg, S. (2015). Tractable cost-optimal planning over restricted polytree causal graphs. In *Proc. 29th AAAI Conference on Artificial Intelligence (AAAI 2015), Austin, TX, USA*, pp. 3225–3231.

Bäckström, C., & Jonsson, P. (2012). Algorithms and limits for compact plan representations. *J.Artif. Intell. Res.*, *44*, 141–177.

Bäckström, C., Jonsson, A., & Jonsson, P. (2012). Macros, reactive plans and compact representations. In *Proc. 20th European Conference on Artificial Intelligence (ECAI 2012), Montpellier, France*, pp. 85–90.

Bäckström, C., & Jonsson, P. (2013). A refined view of causal graphs and component sizes: SP-closed graph classes and beyond. *J. Artif. Intell. Res.*, *47*, 575–611.

Bäckström, C., & Jonsson, P. (2016). Upper and lower time and space bounds for planning. In *Proc. 22nd European Conference on Artificial Intelligence (ECAI 2016), The Hague, The Netherlands*, pp. 716–724.

Bäckström, C., Jonsson, P., Ordyniak, S., & Szeider, S. (2015). A complete parameterized complexity analysis of bounded planning. *J. Comput. Syst. Sci.*, *81*(7), 1311–1332.

Bäckström, C., & Klein, I. (1991). Planning in polynomial time: The SAS-PUBS class. *Comput. Intell.*, *7*, 181–197.

Bäckström, C., & Nebel, B. (1995). Complexity results for SAS$^+$ planning. *Comput. Intell.*, *11*, 625–656.

Betz, C., & Helmert, M. (2009). Planning with h^+ in theory and practice. In *Proc. 32nd Annual German Conference on Artificial Intelligence (KI 2009), Paderborn, Germany,*, pp. 9–16.

Björklund, A., Husfeldt, T., & Koivisto, M. (2009). Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, *39*(2), 546–563.

Brafman, R. I., & Domshlak, C. (2003). Structure and complexity in planning with unary operators. *J. Artif. Intell. Res.*, *18*, 315–349.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artif. Intell.*, *69*(1-2), 165–204.

Bylander, T. (1996). A probabilistic analysis of propositional STRIPS planning. *Artif. Intell.*, *81*(1-2), 241–271.

Cheeseman, P., Kanefsky, B., & Taylor, W. M. (1991). Where the really hard problems are. In *Proc. 12th International Joint Conference on Artificial Intelligence (IJCAI 1991) Sydney, Australia*, pp. 331–340.

Cohen, E., & Beck, J. C. (2017). Problem difficulty and the phase transition in heuristic search. In *Proc. 31st AAAI Conference on Artificial Intelligence (AAAI 2017), San Francisco, CA, USA*, pp. 780–786.

Cygan, M., Dell, H., Lokshtanov, D., Marx, D., Nederlof, J., Okamoto, Y., Paturi, R., Saurabh, S., & Wahlström, M. (2016). On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, *12*(3), 41:1–41:24.

Cygan, M., Kowalik, L., & Wykurz, M. (2009). Exponential-time approximation of weighted set cover. *Inf. Process. Lett.*, *109*(16), 957–961.

de Haan, R., Kanj, I. A., & Szeider, S. (2015). On the subexponential-time complexity of CSP. *J. Artif. Intell. Res.*, *52*, 203–234.

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

Erol, K., Nau, D., & Subrahmanian, V. S. (1991). Complexity, decidability, and undecidability results for domain-independent planning. Tech. rep. CS-TR-2797, Dept. of Computer Science, University of Maryland, College Park, MD, USA.

Flum, J., & Grohe, M. (2006). *Parameterized Complexity Theory*, Vol. XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer, Berlin.

Fredman, M. L., & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, *34*(3), 596–615.

Galperin, H., & Wigderson, A. (1983). Succinct representations of graphs. *Inform. Control*, *56*(3), 183–198.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

Giménez, O., & Jonsson, A. (2009). Planning over chain causal graphs for variables with domains of size 5 is NP-hard. *J. Artif. Intell. Res.*, *34*, 675–706.

Helmert, M. (2006). The fast downward planning system. *J. Artif. Intell. Res.*, *26*, 191–246.

Impagliazzo, R., & Paturi, R. (2001). On the complexity of k-SAT. *J. Comput. Syst. Sci.*, *62*(2), 367–375.

Impagliazzo, R., & Paturi, R. (2013). Exact complexity and satisfiability - (invited talk). In *Proc. Parameterized and Exact Computation - 8th International Symposium, (IPEC 2013), Sophia Antipolis, France, Revised Selected Papers*, pp. 1–3.

Impagliazzo, R., Paturi, R., & Zane, F. (2001). Which problems have strongly exponential complexity?. *J. Comput. Syst. Sci.*, *63*(4), 512–530.

Jonsson, A. (2009). The role of macros in tractable planning. *J. Artif. Intell. Res.*, *36*, 471–511.

Jonsson, P., & Bäckström, C. (1998). Tractable plan existence does not imply tractable plan generation. *Annals Math. Artif. Intell.*, *22*(3-4), 281–296.

Katz, M., & Domshlak, C. (2008). New islands of tractability of cost-optimal planning. *J. Artif. Intell. Res.*, *32*, 203–288.

Katz, M., & Domshlak, C. (2010a). Implicit abstraction heuristics. *J. Artif. Intell. Res.*, *39*, 51–126.

Katz, M., & Domshlak, C. (2010b). Optimal admissible composition of abstraction heuristics. *Artif. Intell.*, *174*(12-13), 767–798.

Katz, M., & Keyder, E. (2012). Structural patterns beyond forks: Extending the complexity boundaries of classical planning. In *Proc. 26th AAAI Conference on Artificial Intelligence (AAAI 2012), Toronto, ON, Canada*, pp. 1779–1785.

Korf, R. E. (2008). Linear-time disk-based implicit graph search. *J. ACM*, *55*(6), 26:1–26:40.

Kronegger, M., Pfandler, A., & Pichler, R. (2013). Parameterized complexity of optimal planning: A detailed map. In *Proc. 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013), Beijing, China*, pp. 954–961.

Lokshtanov, D., Marx, D., & Saurabh, S. (2011). Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, *105*, 41–72.

Plesník, J. (2001). Minimum cost edge subset covering exactly k vertices of a graph. *J. Comb. Optim.*, *5*(3), 275–286.

Rintanen, J. (2004). Phase transitions in classical planning: An experimental study. In *Proc. 14th International Conference on Automated Planning and Scheduling (ICAPS 2004), Whistler, BC, Canada*, pp. 101–110.

Russell, S. J., & Norvig, P. (1995). *Artificial intelligence - A modern approach: The intelligent agent book*. Prentice Hall series in artificial intelligence. Prentice Hall, Upper Saddle River, NJ.

Savitch, W. (1970). Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, *4*(2), 177–192.

Sprugnoli, R. (1994). Riordan arrays and combinatorial sums. *Disc. Math.*, *132*(1-3), 267–290.

Stearns, R. E. (1994). Turing award lecture: It's time to reconsider time. *Commun. ACM*, *37*(11), 95–99.

Stearns, R. E., & Hunt, III, H. B. (1990). Power indices and easier hard problems. *Math, Syst. Theory*, *23*(4), 209–225.

Thorup, M. (2000). On RAM priority queues. *SIAM J. Comput.*, *30*(1), 86–109.

Wehrle, M., & Helmert, M. (2012). About partial order reduction in planning and computer aided verification. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012), Atibaia, São Paulo, Brazil*, pp. 297–305.

Wehrle, M., Helmert, M., Alkhazraji, Y., & Mattmüller, R. (2013). The relative pruning power of strong stubborn sets and expansion core. In *Proc. 23rd International Conference on Automated Planning and Scheduling, (ICAPS 2013), Rome, Italy*, pp. 251–259.

White, L. J. (1971). Minimum covers of fixed cardinality in weighted graphs. *SIAM J. Appl. Math.*, *21*(1), 104–113.

Williams, B. C., & Nayak, P. P. (1997). A reactive planner for a model-based executive. In *Proc. 15th International Joint Conference on Artificial Intelligence (IJCAI 1997), Nagoya, Japan*, pp. 1178–1185.