Ann Math Artif Intell (2016) 78:157–175
DOI 10.1007/s10472-016-9521-y

Refining complexity analyses in planning by exploiting the exponential time hypothesis

Meysam Aghighi[1] · **Christer Bäckström**[1] · **Peter Jonsson**[1] · **Simon Ståhlberg**[1]

Published online: 29 July 2016

Abstract The use of computational complexity in planning, and in AI in general, has always been a disputed topic. A major problem with ordinary worst-case analyses is that they do not provide any quantitative information: they do not tell us much about the running time of concrete algorithms, nor do they tell us much about the running time of optimal algorithms. We address problems like this by presenting results based on the *exponential time hypothesis (ETH)*, which is a widely accepted hypothesis concerning the time complexity of 3-SAT. By using this approach, we provide, for instance, almost matching upper and lower bounds onthe time complexity of propositional planning.

Keywords Action planning · Computational complexity · Lower bounds · Upper bounds · Exponential time hypothesis

Mathematics Subject Classifications (2010) 68Q17 · 68Q25 · 68T20

1 Introduction

In this article, we aim at making worst-case analyses of propositional planning more useful and comprehensible. One problem with ordinary worst-case analyses is that they do not provide any quantitative information: knowing that a particular computational problem is **PSPACE**-complete does not tell us much about the behaviour of concrete algorithms for

Meysam Aghighi and Simon Ståhlberg are partially supported by the National Graduate School in Computer Science (CUGS), Sweden. Christer Bäckström is partially supported by the Swedish Research Council (VR) under grant 621-2014-4086.

✉ Christer Bäckström
christer.backstrom@liu.se

[1] Department of Computer and Information Science, Linköping University, 581 83 Linköping, Sweden

the problem, nor does it tell us much about the running time of optimal algorithms for the problem. Later on (in Section 5.2), we will see striking examples of instance sets X and Y that are widely separated with respect to their computational complexity (under the assumption that **NP**$\neq$ **PSPACE**), yet almost indistinguishable with respect to upper and lower bounds on worst-case time complexity. Another problem with ordinary worst-case analyses is that they have traditionally been performed on very large and (in some sense) unstructured sets of instances. To exemplify, consider Bylander's results [8]: the complexity analyses are performed on sets of instances characterised by the number of preconditions and effects of actions. This leads to sets of instances that contain a wide variety of problems, which often results in overly pessimistic complexity figures from an application point of view.

Our approach is based on the *exponential time hypothesis* (ETH). This hypothesis was suggested by Impagliazzo and Paturi [15] and it has become an important and widely used assumption when studying the computational complexity of combinatorial problems, cf. the survey by Lokshtanov et al. [22]. It is also gaining more and more popularity when studying central problems in AI such as planning and constraint satisfaction [1, 3, 4, 19, 31]. The theory of **NP**-hardness provides evidence that many computational problems are unlikely to be solvable in polynomial time, but this theory does not give any concrete lower time bounds. To achieve such bounds, we need stronger complexity assumptions and the ETH is such an assumption. Assessing the plausibility of the ETH is difficult due to the same reasons as assessing the plausibility of $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$: our understanding of this kind of complexity questions is currently far too weak. We content ourselves by saying that the ETH has deep connections with many topics in computer science, such as the existence of subexponential algorithms for **NP**-complete problems [16, 18, 25], the complexity and approximability of optimisation problems [9, 23] and parameterised complexity theory [10, 11], and the failure of ETH would have far-reaching consequences.

Complexity research in planning has typically first decided on some class C of planning instances and then derived either a tractability result or a hardness result for that particular class. A common objection to tractability results is that C is too restricted to be relevant for real applications. For hardness results, a common objection is instead that C does not correspond to actual applications and the worst cases of C are thus not relevant.

> If the only problematic cases are the ones that do not seem to occur in practice (for example, if they require inputs that are too large), we can simply decide to eliminate them from consideration (without great loss of generality, presumably).
>
> Levesque [20]

While this quote may seem to support the argument that the worst cases are irrelevant, it is a fallacy to draw that conclusion. A more correct conclusion is that we have studied the wrong class of instances and that the ones occuring in practice only constitute a subset of our class.

This inspires us to ask the following question: if we are provided with a class C of planning instances and a planning algorithm $\mathscr{A}$, what can we say about the behaviour of $\mathscr{A}$ for instances of C? We will exploit the ETH to give a general answer to this question: basically, we divide planning problems (parameterised by the allowed sets of instances) into 'easy' and 'hard', and demonstrate how lower bounds on the time complexity of hard problems can be deduced. We emphasise that the choice of both C and $\mathscr{A}$ is completely arbitrary—any set of planning instances will do and any sound and complete planner will do. In particular, we prove almost matching upper and lower bounds on the time complexity of unrestricted propositional planning.

The remainder of the article is structured as follows. In Section 2 we provide some basic definitions for planning and satisfiability, and discuss the time complexity of algorithms for propositional planning. We also formally define the ETH and use it to prove some results that are needed later on. Section 3 formalises and proves the basic theorem that we have already discussed informally. This is followed (in Section 4) by results concerning lower bounds on the time complexity of propositional planning. The upper and lower bounds presented in Sections 2 and 4 are not particularly close to each other so we take the opportunity to discuss and present ways of tightening such bounds in Section 5: this leads to almost matching upper and lower bounds for propositional planning. The results in Sections 4 and 5 additionally illustrate two different lower bound methods with quite different properties: the first method is always applicable (but may result in inferior bounds) while the second method is only applicable in certain cases (but typically gives stronger bounds). Finally (in Section 6), we use our approach for discussing a recent topic in planning: it has been observed in practice that most planning algorithms seem to be much faster on solvable instances than on unsolvable ones, i.e. they are faster at finding a plan than at finding that there is no plan. We demonstrate that this phenomenon is an artifact of the particular algorithms and not an inherent asymmetry in planning. Let C be a set of planning instances such that the planning problem for C is **NP**-hard. Let C_s contain the solvable instances in C and let $\mathscr{A}$ be a planning algorithm that generates solutions (and not only provides yes/no answers). If the worst-case time complexity of $\mathscr{A}$ is significantly better when applied to the instances in C_s compared to the instances in C, then the ETH is not true. In fact, even if we merely assume that $\mathscr{A}$ can generate plans for all instances in C_s (and may behave erratically otherwise), then there exists a planner $\mathscr{A}'$ that is sound and complete for C and that has the same time complexity as $\mathscr{A}$ up to a polynomial-time factor.

2 Preliminaries

We let $\mathbb{N}$ denote the natural numbers (i.e. the non-negative integers), $\mathbb{Z}$ denote the integers, $\mathbb{Z}_+$ denote the positive integers and $\mathbb{N}_\infty$ denote the set $\mathbb{N} \cup \{\infty\}$, i.e. the natural numbers extended with a symbol to denote infinity. Given some object X, we write $|X|$ to denote its cardinality, and we write $||X||$ to denote its size, i.e. the number of bits in the representation of X. Unless otherwise noted, we will assume that polynomials are non-decreasing functions that grow at least linearly.

A *propositional atom* is a variable that can take one of the truth values **true** or **false**. The *negation* of a propositional atom x is denoted $\overline{x}$. A *literal* is either an atom or the negation of an atom, and negation is extended to literals such that $\overline{\overline{x}} = x$. Let X be a set of propositional atoms. Then $L(X)$ denotes the set of literals over X, i.e. $L(X) = \{x, \overline{x} \mid x \in X\}$. A set $Y \subseteq L(X)$ is *consistent* if either $x \notin Y$ or $\overline{x} \notin Y$ for all $x \in X$. We furthermore define $Y^+ = \{x \in X \mid x \in Y\}$ and $Y^- = \{x \in X \mid \overline{x} \in Y\}$. A set $Z \subseteq X$ of atoms *satsifies* a set $Y \subseteq L(X)$ of literals if both $Y^+ \subseteq Z$ and $Y^- \cap Z = \emptyset$.

2.1 Planning

All results in this article will be stated using propositional STRIPS. In particular, we will use *propositional STRIPS with negative goals* (PSN) [8], which can alternatively be viewed as SAS$^+$ [6] restricted to boolean (i.e. two-valued) variable domains.

A PSN *frame* is a tuple $F = \langle V, A \rangle$ where V is a set of propositional atoms and A is a set of actions. The *state space* is $S(F) = 2^V$ and its members are called *(total) states*.

Each action $a \in A$ has a *precondition* $\text{pre}(a) \subseteq L(V)$ and an *effect* $\text{eff}(a) \subseteq L(V)$, which are both consistent. The notation $a : X \Rightarrow Y$ defines an action a with $\text{pre}(a) = X$ and $\text{eff}(a) = Y$. For all $s, t \in S(F)$ and $a \in A$, we say that

- action a is *valid* in s if s satisfies $\text{pre}(a)$ and
- action a is *from s to t* if a is valid in s and $t = (s \cup \text{eff}(a)^+) \setminus \text{eff}(a)^-$.

A sequence $\omega = \langle a_1, \ldots, a_\ell \rangle$ of actions in A is a *plan* from $s_0 \in S(F)$ to $s_\ell \in S(F)$ if either

- ω is the empty sequence and $s_0 = s_\ell$ or
- there are $s_1, \ldots, s_{\ell-1} \in S(F)$ such that a_i is from s_{i-1} to s_i for all i $(1 \leq i \leq \ell)$.

A PSN *instance* is a tuple $P = \langle V, A, s_I, s_G \rangle$ such that $F = \langle V, A \rangle$ is a PSN frame, $s_I \in S(F)$ and $s_G \subseteq L(V)$ is consistent. We tacitly assume that instances and frames do not contain superfluous variables, i.e. every variable appears in the precondition or effect of at least one action. A *plan* for P is a plan from s_I to some $s \in S(F)$ that satisfies s_G. Let PSN denote the set of PSN instances. For any subset $C \subseteq$ PSN of planning instances, we define the *Plan Existence* (PE) problem as follows.

PE(C)
INSTANCE: A planning instance $P \in C$.
QUESTION: Does P have a plan?

We can alternatively view the PE problem as graph search. The *state-transition graph* $G(F)$ for a frame $F = \langle V, A \rangle$ is a directed graph defined as $G(F) = \langle S, E \rangle$, where $S = S(\langle V, A \rangle)$ and E contains all edges $\langle s, t \rangle$ such that there is some action in A from s to t. This is extended to instances such that if $P = \langle V, A, s_I, s_G \rangle$ is a PSN instance, then $S(P) = S(\langle V, A \rangle)$ and $G(P) = G(\langle V, A \rangle)$. One typically wants F to be a succinct representation of the graph $G(F)$. However, the size of F and $G(F)$ may be of the same order: for instance, there may be one action in A for each edge in E.

Let $P = \langle V, A, s_I, s_G \rangle$ be an instance of PE(PSN) and assume, without loss of generality, that the goal state s_G is a total state.[1] Also let $G(P) = \langle S, E \rangle$ be the state-transition graph of P. Obviously, every path from s_I to s_G in $G(P)$ corresponds directly to a plan for P. Hence, P is solvable if and only if there is a path from s_I to s_G in $G(P)$.

There are two obvious choices when it comes to representing planning instances: either the preconditions and effects of actions are represented as lists of defined variables and their values or as bitvectors over all variables. We will exclusively consider the list representation in the sequel: the common situation is that actions depend on and changes a small number of variables compared to the total number of variables.

2.2 Planning algorithms

In general, every planner for PSN must run in worst-case exponential time if we require it to output a solution, since there are planning instances with exponentially long shortest solutions. If the planner is only required to solve the decision problem, PE, then this simple observation is not useful for saying anything about the running time of the planner. For example, the 3S class [17] has instances with exponentially long shortest solutions, but PE(3S) can be solved in polynomial time.

[1] This can always be achieved by adding one propositional atom and one action.

We continue by demonstrating a straightforward upper bound for PE based on search in the state-transition graph. Searching state-space graphs in planning requires somewhat more care than for graph searching in general. We may assume that all actions in A are unique so every action must correspond to a unique combination of preconditions and effects. The set $L(V)$ of literals over V contains $2|V|$ literals, so there are less than $2^{2|V|}$ consistent subsets of $L(V)$. Hence, there are at most $2^{2|V|}$ different preconditions and at most $2^{2|V|}$ different effects, which results in at most $2^{2|V|} \cdot 2^{2|V|} = 2^{4|V|}$ unique actions. That is, the size of A, and thus also the size of P, is not necessarily polynomially bounded in the number of variables for a class of planning instances. However, we know that $|A| < ||A|| < ||P||$ so the instance size is always a safe, albeit crude, upper bound on the number of actions.

For simplicity, we may consider representing $G(P)$ using adjacency lists and start by constructing an array for S and initialising all adjacency lists, which takes time $O(|S|) = O(2^{|V|})$. Then note that every state in S can have at most one outgoing arc for every action, i.e. each adjacency list is of length $|A|$, at most. We can compute the adjacency list for each state $s \in S$ by checking s against every action $a \in A$. If a is valid in s, then it is trivial to compute the unique state $t \in S$ such that a is from s to t, and insert an arc from s to t in the adjacency list of s, if there is not already such an arc. Checking an action against a state takes polynomial time in the instance size $||P||$ and $|A| \leq |P| \leq ||P||$. Hence, there is some polynomial p such that this procedure takes $O(p(||P||)$ time for each state. Constructing $G(P)$ thus takes total time $O\left(|S| + |S| \cdot p(||P||)\right) = O\left(|S| \cdot p(||P||)\right) = O\left(2^{|V|} \cdot p(||P||)\right)$.

Checking if there is a path from s_I to s_G in $G(P)$ can be done by depth-first or breadth-first search in time

$$\begin{aligned} O\left((|S| + |E|) \cdot q(||P||)\right) &\subseteq O\left((|S| + |S|^2) \cdot q(||P||)\right) \subseteq \left(|S|^2 \cdot q(||P||)\right) \\ &\subseteq O\left(2^{2|V|} \cdot q(||P||)\right) \end{aligned}$$

for some polynomial q.

Recall that we require that every variable is used (i.e. appears in either the preconditions or the effects) by at least one action. This implies that we have the name of each variable appear in at least one action which requires at least $|V| \log |V|$ bits. Thus,

$$|V| \log |V| \leq ||A|| \leq ||P||$$

for sufficiently large $|V|$. We conclude that we can assume $|V| \leq ||P||/\Psi$ for some arbitrarily chosen constant $\Psi \geq 1$ without loss of generality. The method of solving PE(PSN) by first constructing $G(P)$ and then search for a path in it thus has an upper bound of time

$$O\left(2^{|V|} \cdot p(||P||) + 2^{2|V|} \cdot q(||P||)\right) \subseteq O\left(2^{||P||/\Psi} \cdot p(||P||) + 2^{2||P||/\Psi} \cdot q(||P||)\right)$$

that equals $O\left(2^{||P||/(\Psi/2)} \cdot r(||P||)\right)$ for some polynomial r. From now on and throughout the article, fix $\Psi \geq 1$ and let $\Phi = \Psi/2$.

The algorithm proposed above uses an exponential amount of memory. By using Savitch's theorem [26], the amount of memory can be lowered. Savitch showed that there exists an algorithm $\mathscr{A}_S$ 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 $O\left(\log^2(|U|)\right)$ 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 only thing one has to keep in mind when using this time bound is that we must be able to check whether two vertices are connected or not in polynomial time (in the size of the graph). Problem PE can thus be solved by asking if there is a plan of length $k = |S| = 2^{|V|}$.

If we assume an implicit graph representation where vertex adjacency can be checked in $p(||P||)$ time for some polynomial p (such as PSN), we can thus solve PE in time

$$|S|^{O(\log k)} = |S|^{O(\log |S|)} = (2^{|V|})^{O(\log 2^{|V|})} = (2^{|V|})^{O(|V|)} = 2^{O(|V|^2)} \subseteq 2^{O((||P||/\Psi)^2)}$$

using space $O(\log^2 |S|) = O(\log^2 2^{|V|}) = O(|V|^2) \subseteq O((||P||/\Psi)^2)$.

2.3 Satisfiability and the exponential time hypothesis

We will use several different variants of the boolean satisfiability problem during the course of this article. In its basic form it is defined as follows.

SAT
INSTANCE: A boolean formula F on conjunctive normal form, i.e. F is a conjunction of *clauses* where each clause is a disjunction of literals.
QUESTION: Does 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 implies that there are no unused variables, i.e. every variable appears in at least one clause. The problem k-SAT, $k \geq 1$, is the SAT problem restricted to clauses containing at most k literals. The SAT problem is **NP**-complete and so is the k-SAT problem when $k \geq 3$ [12, problem LO1]. We will also consider the complement of SAT (known as UNSAT): an instance of this problem is considered a 'yes'-instance if and only if there does *not* exist a satisfying assignment. The UNSAT problem is **coNP**-complete and so is the UNSAT problem restricted to clauses of length k (which we denote k-UNSAT) whenever $k \geq 3$. The following properties of 3-CNF formulae will be used in the forthcoming proofs. The bounds are straightforward to prove by recalling that F contains no repeated clauses, no empty clauses, and no unused variables.

Proposition 1 *Let F be an arbitrary 3-CNF formula with n variables and m clauses. Then one may assume that*

1. $n \leq 3m$,
2. $m \leq 8n^3$,
3. $||F|| \leq 3m(1 + \log n)$ *and*
4. $||F|| \leq 12m \log m$, *for* $m \geq 2$.

A more precise characterisation of the complexity of k-SAT (compared to merely saying that it is an **NP**-complete problem) is possible by using the *exponential time hypothesis (ETH)* [15, 16]. This hypothesis is a conjecture stated as follows.

Definition 2 For all constant integers $k > 2$, let s_k be the infimum of all real numbers δ such that k-SAT can be solved in time $O(2^{\delta n})$, where n is the number of variables of an instance. The *exponential time hypothesis (ETH)* is the conjecture that $s_k > 0$ for all $k > 2$.

Informally, ETH says that satisfiability cannot be solved in subexponential time. One may equivalently define the ETH for the number of clauses, i.e. replacing $O(2^{\delta n})$ with $O(2^{\delta m})$ in Definition 2, where m is the number of clauses. It is known that the ETH with respect to the number of clauses holds if and only if the ETH with respect to the number of variables holds [16]. Note that since the ETH refers to actual deterministic time bounds it is

possible to swap the answer, i.e. if we could solve k-UNSAT in time $O(f(n))$ for some function f, then we could also solve k-SAT in time $O(f(n))$. Hence, the ETH can equivalently be defined in terms of the k-UNSAT problem.

Since using the ETH offers more precision than polynomial reductions between the usual complexity classes do, we will benefit from also being more precise about how much a reduction blows up the instance size, defining the following measure for this purpose.

Definition 3 Given a reduction ρ from some problem X to some problem Y, we say that ρ has *blow-up* b, for some $b > 0$, if there exists an $n > 0$ such that $||\rho(I)|| \leq ||I||^b$ for all instances I of X such that $||I|| \geq n$.

Clearly, every polynomial-time reduction has bounded blow-up.

The time complexity of a problem is usually defined as a function of the instance size, while the ETH is defined as a function of the number of variables or clauses. This latter allows for a sharper characterisation but it is not quite suitable for planning where the instance size is not necessarily polynomial in the number of variables. We will instead use time bounds on the form $O(2^{n^c})$, where n is the instance size. This approach is closely related to the concept of *power indices* [28–30]: the power index of a problem X equals $\inf\{c \mid \mathsf{X} \in \mathrm{DTIME}(2^{n^c})\}$. Although the power index of SAT is obviously not known, Stearns and Hunt assume that SAT has power index 1, which is known as the *satisfiability hypothesis (SH)*. The SH predates the ETH, but the concepts are clearly connected, with the ETH as the stronger assumption. This is demonstrated by the following lemma which shows that the power index of SAT must be at least 1 (and, thus, exactly 1) if the ETH is true, i.e. the SH is true if the ETH is true.

Lemma 4 3-*SAT and* 3-UNSAT *cannot be solved in time* $O(2^{||F||^c})$ *for any* $c < 1$*, unless the ETH is false.*

Proof Suppose 3-SAT can be solved in time $O(2^{||F||^c})$ for some $c < 1$. We know from Proposition 1 that $||F|| \leq 12m \log m$, for $m \geq 2$. Furthermore, $12m \log m < m^{1+\epsilon}$ for all $\epsilon > 0$ and large m. Choose $\epsilon = 1 - c$, which satisfies that $\epsilon > 0$ since $c < 1$. It then follows from the assumption that 3-SAT can be solved in time $O(2^{||F||^c}) \subseteq O(2^{m^{(1+\epsilon)c}}) = O(2^{m^d})$, where $d = (1+\epsilon)c = (1+\epsilon)(1-\epsilon) < 1$. This contradicts the ETH since 2^{m^d} grows slower than $2^{\delta m}$ for all $\delta > 0$. The case for 3-UNSAT is analogous. □

It is important to note that this lemma only states a one-way relationship between the SH and the ETH. It does state that the ETH must be false if 3-SAT can be solved in time $O(2^{||F||^c})$ for some $c < 1$. However, it does not rule out the possibility that the ETH is still false even if 3-SAT cannot be solved in time $O(2^{||F||^c})$ for any $c < 1$. This is because $||F|| \in \Theta(m \log m)$ and there is no c such that $m \in \Theta((m \log m)^c)$. If $c < 1$, then $(m \log m)^c$ grows strictly slower than m, and if $c \geq 1$, then $(m \log m)^c$ grows strictly faster than m.

3 Restricted planning

We will now address the main question of this article: if we are allowed to choose a class of planning instances and a planning algorithm, what can we say about the time complexity in this case? A general answer to this question will be provided in Theorem 5 below. We

will then (in Sections 4 and 5) demonstrate how this answer can be instantiated to concrete figures in specific cases.

3.1 Main result

Theorem 5 *Let C be a class of* PSN *instances. Then, at least one of the following must hold:*

1. $\mathsf{PE}(C)$ *is neither* ***NP****-hard nor* ***coNP****-hard,*
2. *there exists some $c > 0$ such that $\mathsf{PE}(C)$ cannot be solved in time $O(2^{||P||^c})$ or*
3. *the ETH is not true.*

We will intervene with a brief discussion of this result before moving on to the proof. In case (1), $\mathsf{PE}(C)$ is neither **NP**-hard nor **coNP**-hard. One may suspect that membership in **P** is the most common instance of this case. If so, a tractable fragment of planning has been identified. Assume now that $\mathsf{PE}(C)$ is in **NP** but not a member of **P**, i.e. $\mathsf{PE}(C)$ is an **NP**-intermediate problem. This may appear a bit surprising since almost all complexity results for planning appearing in the literature exclusively deal with "well-known" complexity classes such as **P**, **NP**, and **PSPACE**. However, an infinite number of complexity classes can be captured by propositional planning if we assume that **P** $\neq$ **PSPACE**–see Section 3.2.

In case (2), $\mathsf{PE}(C)$ cannot be solved in time $O(2^{||P||^c})$ for some $c > 0$: this is bad news since it basically implies that $\mathscr{A}$ performs poorly on C. At this point, it is important to recall that the choice of $\mathscr{A}$ and C is completely arbitrary. This, in turn, implies that Levesque's suggestion (as stated in the introductory section) of removing 'unwanted' instances is problematic: in order to get a running time that is in $O(2^{||P||^c})$ for all $c > 0$, we have to find a subset C' of C such that $\mathsf{PE}(C')$ does not belong to the same complexity class as $\mathsf{PE}(C)$. We can restate this as follows: $\mathsf{PE}(C')$ must be easier with respect to *worst-case* computational complexity than $\mathsf{PE}(C)$. In other words, understanding the worst-case computational complexity of planning problems is highly relevant also for understanding the complexity of naturally arising planning problems.

Regarding case (3), it suffices to note that if the ETH were false, then this would have such fundamental consequences for complexity theory in general that both this article and many others in the literature would become obsolete.

Proof of Theorem 5 First suppose that $\mathsf{PE}(C)$ is neither **NP**-hard nor **coNP**-hard. Then we are in case 1 and we are done.

Otherwise, $\mathsf{PE}(C)$ can be assumed to be either **NP**-hard or **coNP**-hard. If there is some $c > 0$ such that $\mathsf{PE}(C)$ cannot be solved in time $O(2^{||P||^c})$, then we are in case 2 and we are done.

In case 3, the remaining possibility is that there is some algorithm $\mathscr{A}$ that can solve $\mathsf{PE}(C)$ in time $O(2^{||P||^c})$ for all $c > 0$. First consider the case where $\mathsf{PE}(C)$ is **NP**-hard. Then there is a polynomial reduction ρ from 3-SAT to $\mathsf{PE}(C)$, i.e. there is some polynomial p and some b such that ρ has blow-up b and can be computed in time $p(||F||)$. There is then an algorithm $\mathscr{B}$ for 3-SAT that works by first computing the PSN instance $\rho(F)$ and then applying $\mathscr{A}$ to $\rho(F)$. Choose c such that $0 < c < 1/b$. It then follows from the assumption that $\mathscr{B}$ solves 3-SAT in time

$$O\left(p(||F||) + 2^{(||F||^b)^c}\right) = O\left(p(||F||) + 2^{||F||^{bc}}\right) = O\left(2^{||F||^{bc}}\right).$$

However, this contradicts the ETH according to Lemma 1, since $bc < b(1/b) = 1$.

Instead suppose that PE(C) is **coNP**-hard. Then there is a polynomial reduction ρ from 3-UNSAT to PE(C) and the rest of the proof is analogous to the previous case, since Lemma 1 applies also to 3-UNSAT. □

Theorem 5 may appear slightly disappointing for at least two reasons: (1) it is not exclusively about the planning problem since it can easily be adapted to almost any computational problem and (2) it does not take any concrete algorithm into account. Reason (1) is interesting from several viewpoints. It is quite obvious that Theorem 5 is applicable to a wide range of problems. In fact, it can be adapted to every **NP**- or **coNP**-hard problem with virtually no changes of its proof. However, this is not the case for the results appearing later in this article; in fact, most of the proofs intimately use the fact that we are studying the planning problem and not an arbitrary **NP**- or **coNP**-hard problem. For instance, we will (in Section 4) exhibit a particular polynomial-time reduction from 3-UNSAT to PE(PSN) with blow-up 3. Obtaining similar reductions to some other problem X poses certain problems. First of all, most **NP**- and **coNP**-hardness proofs are not based on direct reductions from 3-(UN)SAT—such proofs are typically based on (sometimes quite long) chains of reductions from the (UN)SAT problem to X. The blow-up of such a chain can be huge (and thus lead to weak results) and, more importantly, immensely difficult to calculate. The close connections between *propositional* logic and *propositional* planning made this potentially difficult problem quite manageable.

Reason (2) is, fortunately, not a reason for disappointment. Theorem 5 is providing a lower bound and *no* algorithm can beat this bound unless the ETH is false. Thus, there is no need to discuss any particular algorithm. This fact must not be interpreted wrongly: even though no given planner $\mathscr{A}$ can beat the lower bound, planner $\mathscr{A}$ can have a time complexity that is significantly *worse* than this bound. Thus, the choice of planner is not at all unimportant when facing concrete planning instances.

While case 2 of this theorem states that there exists some constant c such that PE(C) cannot be solved in time $O(2^{||P||^c})$, it does not say anything about concrete values of c. Typically, we want to know the value of c, or at least we need some estimation of the value, since there is a huge difference between algorithms that run in, say, time $O(2^{||P||^{0.01}})$ and algorithms that run in time $O(2^{||P||^{0.99}})$. The proof hints at one way to determine actual values for c, though, based on the blow-up of reductions. This can be formalised as follows.

Corollary 6 *Let C be a class of* PSN *instances. If the ETH holds and* PE(C) *is* ***NP****-hard, then* PE(C) *cannot be solved in time $O(2^{||P||^{1/c}})$ for any $c > b_*$ where*

$$b_* = \inf\{b \mid \textit{there is a poly-time reduction from 3-SAT to PE}(C) \textit{ w. blow-up } b\}.$$

The analogous result holds when PE(C) *is* ***coNP****-hard.*

Proof Suppose problem PE(C) can be solved in time $O(2^{||P||^{1/c}})$ for some $c > b_*$. Choose a polynomial-time reduction from 3-SAT to PE(C) with blow-up b such that $b_* < b < c$. Such a reduction must exists due to the definition of b_*. By assumption we can thus solve 3-SAT in time $O(2^{(||F||^b)^{1/c}}) = O(2^{||F||^{b/c}})$, but this contradicts the ETH according to Lemma 4, since $b/c < 1$. □

While this corollary establishes a relationship between the blow-up of reductions and the constant c, it still only states that c and b_* are related. Hence, we will (in Section 4) demonstrate how one can proceed to achieve actual values for c. It is quite natural that

computing exact values for b_* and c is often a difficult problem. We stress that finding exact values is not always a necessity—estimates of b_* and c still provide working lower bounds albeit not optimal bounds.

3.2 Capturing complexity classes with planning classes

For all that we know, it could be the case that only a finite number of complexity classes can be captured by planning. In fact, earlier results point in this direction: it is extremely rare that any other classes than **P**, **NP**, and **PSPACE** appear in complexity classifications of planning. However, this is not the case—planning captures an enormous amount of different computational problems and complexity classes if we assume that $\mathbf{P} \neq \mathbf{PSPACE}$. We will see below that any complexity class $\mathcal{C}$ such that $\mathbf{P} \subseteq \mathcal{C} \subseteq \mathbf{PSPACE}$ and that is closed under polynomial reductions can be described by a class of planning instances. Furthermore, every computational problem X in **PSPACE** can be captured by the plan existence problem!

Bylander has shown that PE(PSN) is **PSPACE**-complete [8]. This was accomplished by devising a polynomial-time reduction from Turing machine acceptance with polynomially bounded tape. Let M be a Turing machine where the tape length is bounded by a polynomial q, i.e. the number of tape cells is bounded by $q(|x|)$ where x is the input string. Let $\rho(M, q, x)$ denote the corresponding PSN instance that has a solution if and only if M accepts input x using at most $q(|x|)$ tape cells. According to Bylander's result, ρ is polynomial-time computable. The reason that we make the polynomial q explicit, instead of implicit in M, is that Bylander's reduction must have explicit access to it. Now, consider the following set of PSN instances:

$$C_M^q = \{\, \rho(M, q, x) \mid x \text{ is an input string for } M \,\}.$$

We say that ρ has property (*) if

1. for every $P \in C_M^q$, there exists exactly one string x such that $P = \rho(M, q, x)$ and
2. for every $P \in C_M^q$, one can in polynomial time compute the x such that $P = \rho(M, q, x)$.

Property (*) is not a very prohibitive assumption—think of PDDL[2] encodings of Bylander's reduction. Every such reduction has property (*) since every variable is given an explicit name. Thus, given an instance of C_M^q, we can extract the unique input string in polynomial time by looking at the variables corresponding to the tape. Henceforth, let ρ denote this particular reduction.

Theorem 7 *Let X be an arbitrary computational problem in **PSPACE**. Then there exists a class C_X of* PSN *instances such that X and $\mathsf{PE}(C_\mathsf{X})$ are polynomial-time equivalent problems.*

Proof The problem X can be solved by a Turing machine M_X with tape length bounded by a polynomial q (since X is in **PSPACE**). Let $C_\mathsf{X} = \{\, \rho(M_X, q, x) \mid x \text{ is an input string} \}$. Obviously, there exists a polynomial-time reduction from X to $\mathsf{PE}(C_\mathsf{X})$. There is also a polynomial-time reduction in the other direction: arbitrarily choose $P \in C_\mathsf{X}$ and compute

[2]PDDL (the *Planning Domain Definition Language*) is a commonly used specification language for planning instances.

(in polynomial time) the input string x such that $\rho(M_{\mathsf{X}}, q, x) = P$. Then, P has a solution if and only if M_{X} accepts x. □

Corollary 8 *Let $\mathcal{C}$ be a complexity class in **PSPACE** such that $\mathcal{C}$ is closed under polynomial-time reductions and $\mathcal{C}$ contains complete problems under polynomial-time reductions. Then, there exists a set of* PSN *instances C such that $\mathsf{PE}(C)$ is complete for $\mathcal{C}$.*

4 A concrete lower bound

The results in the previous section are general and Corollary 6 only states that there exists a relationship between the lower bound and the blow-up of reductions. Hence, we will give a more concrete example in this section, demonstrating how one can determine the actual blow-up of a reduction and use this to instantiate the previous result. More precisely, we will present a reduction from 3-UNSAT to PE(PSN).

We begin by recalling how to encode an n-bit binary counter in PSN [2]. Let $V = \{x_1, \ldots, x_n\}$ and let A contain the n actions

$$c_i : \{x_1, \ldots, x_{i-1}, \overline{x}_i\} \Rightarrow \{\overline{x}_1, \ldots, \overline{x}_{i-1}, x_i\} \quad (1 \leq i \leq n).$$

Then use V and A to construct a PSN instance $P = \langle V, A, s_I, s_G \rangle$ such that $s_I = \emptyset$ and $s_G = \{ x_i \mid 1 \leq i \leq n \}$. This instance is always solvable and it has a shortest plan of length $2^n - 1$ actions that corresponds to a Hamilton path from s_I to s_G in the state-transition graph $G(P)$.

Construction 9 Define a function $\rho : 3 - \text{UNSAT} \to \text{PSN}$ as follows. Let F be an instance of 3-UNSAT with n variables $\{x_1, \ldots, x_n\}$ and m clauses $\{C(F)_1, \ldots, C(F)_m\}$. Assume without loss of generality that F contains no clause $C(F)_j$ that is a tautology, i.e. both x_i and $\overline{x}_i$ appear in $C(F)_j$ for some $1 \leq i \leq n$. We construct a corresponding PSN instance $\rho(F) = P_F = \langle V, A, s_I, s_G \rangle$ as follows:

- Let $V = \{x_1, \ldots, x_{n+1}\}$.
- Let $c_1, \ldots, c_{n+1}$ denote the actions in the $n + 1$-bit counter over the variables $x_1, \ldots, x_{n+1}$. For each clause $C(F)_j = (\ell_1 \vee \ell_2 \vee \ell_3)$ of F, where $1 \leq j \leq m$, define $T_j = \{\overline{\ell}_1, \overline{\ell}_2, \overline{\ell}_3\}$. Let $A = \{ a_{i,j} \mid 1 \leq i \leq n + 1,\ 1 \leq j \leq m \}$, where the actions are defined as $a_{i,j} : \text{pre}(c_i) \cup T_j \Rightarrow \text{eff}(c_i)$. One may view $a_{i,j}$ as action c_i in a binary counter extended with preconditions saying "clause j is not satisfied by $x_1, \ldots, x_n$".
- Let $s_I = \emptyset$.
- Let $s_G = \{\overline{x}_1, \ldots, \overline{x}_n, x_{n+1}\}$.

This construction is indeed a polynomial reduction from 3-UNSAT to PE(PSN) and we now determine a concrete upper bound for its blow-up.

Lemma 10 *The function ρ in Construction 9 is a polynomial reduction from problem 3-*UNSAT *to PE(*PSN*) with blow-up 3 at most.*

Proof We first prove that the function ρ is a polynomial reduction from 3-UNSAT to PE(PSN). Let F be a 3-UNSAT instance and let $P_F = \rho(F)$ be the corresponding PSN instance according to Construction 9. If we treat the variables $x_1, \ldots, x_{n+1}$ as encoding a binary number, then every plan must count through all numbers from 0 to 2^n, since

these numbers correspond to the initial and goal states and there are only counting actions. Furthermore, for each state $s \in S(P_F)$, only one of the different types $c_1, \ldots, c_{n+1}$ of counter actions is valid in s and there are m variants of each such action, one for each clause of F. Hence, for each step in a plan for P_F, there are m different actions to choose from. An action corresponding to a clause $C(F)_j$ of F has a precondition that requires all literals in $C(F)_j$ to be false, i.e. the action is valid only if $C(F)_j$ is false in the current variable assignment specified by variables $x_1, \ldots, x_n$. This means that for each assignment, i.e. for each state, it is necessary to find an action corresponding to a clause of F that is false for that assignment. In other words, a plan for P_F must verify that for every assignment to $x_1, \ldots, x_n$, at least one clause of F is false, i.e. that F is unsatisfiable. Hence, ρ is a reduction from $3-$UNSAT to PE(PSN) and it is obvious that it is polynomial-time computable.

We then prove that ρ has blow-up at most 3. Let F be a 3-UNSAT instance with n variables and m clauses. Without losing generality, assume that $m \geq 3$ and $n \geq 2$. We know from Proposition 1 that $n \leq 3m$ and that F requires $||F|| \leq 3m(1 + \log n)$ bits to represent. However, it is reasonable to assume that all clauses contain exactly three literals, so we can assume equality, i.e. $||F|| = 3m(1 + \log n)$. Let $\rho(F) = P_F = \langle V, A, s_I, s_G \rangle$ be the corresponding PSN instance according to Construction 9. Then $|V| = n + 1$ and $|A| = m|V| = m(n + 1)$.

Each set of literals is represented as a list of literals. Each literal must uniquely identify a variable and its polarity, which requires $\log |V| + 1$ bits. A consistent literal set thus requires at most $|V|(\log |V| + 1)$ bits. For a total state it is sufficient to list the variables that are true, but for simplicity we overestimate this to get $|V|(\log |V| + 1)$ bits also in this case. For the set of variables, V, we just list all variables where each variable requires $\log |V|$ bits to identify. For the action set, A, we list all actions and represent the precondition and effect for each one. We thus get:

$$\begin{aligned} ||V|| &= |V| \cdot \log |V|, \\ ||A|| &= \textstyle\sum_{a \in A} (||\text{pre}(a)|| + ||\text{eff}(a)||) \\ &\leq |A| \cdot (|V|(\log |V| + 1) + |V|(\log |V| + 1)) \\ &\leq m|V| \cdot 2|V|(\log |V| + 1) \\ &= 2m|V|^2(\log |V| + 1), \\ ||s_I||, ||s_G|| &\leq |V| \cdot (\log |V| + 1). \end{aligned}$$

It is now straightforward to verify that $||P_F|| \leq 3m(n + 1)^2 \cdot (\log(n + 1) + 1)$.

To prove that the blow-up is at most 3, we need to prove that $||P_F|| \leq ||F||^3$ for sufficiently large instances. We have assumed that $||F||^3 = (3m(\log n + 1))^3 = 27m^3(\log n + 1)^3$. We also know that $n \leq 3m$, so $3m(n+1)^2 \leq 48m^3$ (since $m \geq 2$). Hence,

$$\frac{||P_F||}{||F||^3} = \frac{3m(n + 1)^2 \cdot (\log(n + 1) + 1)}{(3m(\log n + 1))^3} \leq \frac{48m^3 \cdot (\log(n + 1) + 1)}{27m^3 \cdot (\log n + 1)^3} \leq 1,$$

which holds since $n \geq 2$. Note that this result is not sensitive to the constant in the expression for $||F||$; if we underestimate $||F||$ with the value $m(\log n + 1)$, i.e. only assuming at least one literal per clause, then the inequality would still hold for sufficiently large values of n. It follows that the construction has blow-up at most 3. □

We can now plug this blow-up figure into Corollary 6 to get a concrete lower bound for PSN planning.

Theorem 11 *PE(*PSN*) takes time* $\Omega(2^{||P||^{1/3}})$ *to solve, unless the ETH is false.*

Proof We know from Lemma 10 that there exists a polynomial-time reduction ρ from 3-UNSAT to PE(PSN) with blow-up 3. Hence, we know that $b_* \leq 3$ for the constant b_* in Corollary 6. It thus follows from this corollary that PE(PSN) takes time $\Omega(2^{||P||^{1/3}})$ to solve, unless the ETH is false. □

After having seen how to use Theorem 5 for analysing the time complexity of planning problems, it is not very hard to investigate the time complexity of, for instance, standard benchmark problems. An excellent starting point is the article by Helmert [13]. Let us consider the benchmark problem GRID for instance. Helmert shows **NP**-hardness in Theorem 30 via a reduction from 3-SAT. The blow-up of this reduction can be determined and it immediately gives a $c > 0$ such that GRID cannot be solved in time $O(2^{||P||^c})$ unless the ETH is false. Let us consider the TRANSPORT problem instead. Here, Helmert shows **NP**-hardness (in Theorem 14) by a reduction from the HAMILTONIAN PATH problem. In this case, we need to analyse two reductions: some reduction from 3-SAT to HAMILTONIAN PATH (Theorem 7.46 in Sipser [27] presents one possible reduction) followed by Helmert's reduction. Once again, we can obtain a $c > 0$ such that TRANSPORT cannot be solved in time $O(2^{||P||^c})$ unless the ETH is false. Note that the value of c established in this way may not be optimal.

5 An improved lower bound

By combining the upper bounds presented in Section 2.1 with Theorem 11, we know that PSN planning can be performed in time $O(2^{||P||/\Phi})$ where $\Phi \geq 1/2$ but it cannot be performed in time $O(2^{||P||^{1/3}})$ if the ETH holds. We study lower bounds for domain-independent planning in greater detail in this section. We show that PE(PSN) cannot be solved in time $O(2^{||P||^c})$ for any $c < 1$ and this bound closely matches the upper bound for PE(PSN). The lower bound on PSN planning that was shown in Theorem 11 was obtained by analysing the blow-up of a particular reduction from 3-UNSAT to PE(PSN). This way of obtaining lower bounds is very general—it is applicable whenever one is considering (**co**)**NP**-hard problems—but it is sometimes difficult to achieve tight bounds. There are alternative ways of obtaining lower bounds, though, and we will illustrate one of them. While the ETH is intrinsically based on the time complexity of 3-SAT, there are other computational problems that can be used equally well, i.e. there are other problems X such that if X can be solved in subexponential time, then the ETH is false. Sometimes X exhibit properties that simplify the identification of lower bounds.

We consider lower bounds based on the number of variables in Section 5.1 and we use these bounds for proving results based on instance size in Section 5.2.

5.1 Lower bound in the number of variables

In the sequel, we will exploit a particular constraint satisfaction problem (CSP) for obtaining an improved lower bound on the complexity of PE(PSN). Let Γ be a set of finitary relations

over some domain D; such sets are known as *constraint languages*. We define $\mathsf{CSP}(\Gamma)$ to be the following computational problem.

$\mathsf{CSP}(\Gamma)$
INSTANCE: A tuple $\langle V, C\rangle$ where V is a finite set of variables and C is a finite set of *constraints*, i.e. objects $R(v_{i_1}, \ldots, v_{i_k})$ where $R \in \Gamma$, $\{v_{i_1}, \ldots, v_{i_k}\} \subseteq V$ and k equals the arity of R.
QUESTION: Is there a function $f : V \to D$ such that $\langle f(v_{i_1}), \ldots, f(v_{i_k})\rangle \in R$ for every constraint $R(v_{i_1}, \ldots, v_{i_k})$ in C?

Given a $\mathsf{CSP}(\Gamma)$ instance $\langle V, C\rangle$, we say that $\langle V, C\rangle$ has *degree*[3] d if each variable appears in at most d different constraints. Given a constraint language Γ, we let $\mathsf{CSP}(\Gamma)$-d denote the $\mathsf{CSP}(\Gamma)$ problem restricted to instances of degree d. Let R denote the 6-ary relation $\{(1, 0, 0, 0, 1, 1), (0, 1, 0, 1, 0, 1), (0, 0, 1, 1, 1, 0)\}$. We have the following result.

Theorem 12 (Jonsson et al. [18, Thm. 6.2]) *$\mathsf{CSP}(\{R\})$-2 can be solved in time $2^{\epsilon \cdot n}$ (where n is the number of variables) for every $\epsilon > 0$ if and only if the ETH is false.*

We have chosen to work with $\mathsf{CSP}(\{R\})$-2 because of its simplicity: we only need to consider a single relation and we know the exact value of the degree bound. However, many other degree-bounded constraint problems would, in principle, have worked equally well for obtaining the lower bound.

Note that every instance $\langle V, C\rangle$ of $\mathsf{CSP}(\{R\})$-2 satisfies $|C| \leq 2|V|$: assume we have a set of constraints $\{C_1, \ldots, C_n\}$ and we want to minimise the number of variables used in these constraints. To fill "all variable slots", we make pairs of the constraints and fill them with one variable per pair, i.e.

$$C_1(x_1, \ldots, x_1), C_2(x_1, \ldots, x_1), C_3(x_2, \ldots, x_2), C_4(x_2, \ldots, x_2), \ldots$$

Hence, $n/2$ variables are needed and $|C| = 2|V|$ under the safe assumption that n is even.

We now present a polynomial-time reduction from $\mathsf{CSP}(\{R\})$-2 to PE(PSN). We exploit the fact that $\mathsf{CSP}(\{R\})$-2 instances contain a fairly small number of constraints in order to produce PSN instances that contain a small number of variables and actions.

Let $T = \langle V, C\rangle$ be an arbitrary instance of $\mathsf{CSP}(\{R\})$-2. Assume $V = \{v_1, \ldots, v_n\}$ and $C = \{C_1, \ldots, C_m\}$. We construct a PSN instance $P_T = \langle W, A, s_I, s_G\rangle$ based on $\langle V, C\rangle$. We avoid notational inconveniences by not giving concrete names to the actions in A; we simply refer to them by their preconditions and effects. Let $W = \{a_1, \ldots, a_n\} \cup \{b_1, \ldots, b_m\} \cup \{c\}$. For each variable $v_i \in V$, introduce the action $\overline{c} \Rightarrow a_i$. For each $C_j \in C$, do the following. Assume for simplicity that $C_j = R(v_1, \ldots, v_6)$ and introduce the following three actions:

1. $a_1, \overline{a}_2, \overline{a}_3, \overline{a}_4, a_5, a_6 \Rightarrow b_j, c$
2. $\overline{a}_1, a_2, \overline{a}_3, a_4, \overline{a}_5, a_6 \Rightarrow b_j, c$
3. $\overline{a}_1, \overline{a}_2, a_3, a_4, a_5, \overline{a}_6 \Rightarrow b_j, c$

Finally, let $s_I = \{\overline{w} \mid w \in W\}$ and $s_G = \{b_j \mid 1 \leq j \leq m\}$. It is not hard to verify that $\langle W, A, s_I, s_G\rangle$ has a solution if and only if $\langle V, C\rangle$ has a solution, $\langle W, A, s_I, s_G\rangle$ can be constructed in polynomial time in $||\langle V, C\rangle||$, and that $|W| \leq 3|V| + 1$ since $|C| \leq 2|V|$.

Assume now that PE(PSN) can be solved in $2^{|V|^c}$ time for some $c < 1$. This implies that $\mathsf{CSP}(\{R\})$-2 can be solved in $2^{(3|V|)^c}$ time. This implies, in turn, that $\mathsf{CSP}(\{R\})$-2 can be

[3]The reader should be aware that other notions of degree appear in the literature.

solved in time $2^{\epsilon \cdot |V|}$ for all $\epsilon > 0$, since $\epsilon |V| > (3|V|)^c$ for large $|V|$ when $c < 1$. Hence, the ETH does not hold due to Theorem 12. We have shown the following result.

Lemma 13 PE(PSN) *cannot be solved in time* $2^{|V|^c}$ *for any* $c < 1$ *unless the ETH is false.*

One should note that this result holds for much more restricted sets of planning instances than PSN. For example, the actions have only positive effects, which implies that the optimal solutions are very short since no variable is required to change more than once at most. One should also note that there probably are no "useful" reductions from CSP($\{R\}$)-2 to every **NP**-hard planning problem PE(C) in the sense that they beat the bounds that can be obtained by the blow-up method in Section 4.

5.2 Lower bound in the instance size

The bound given in Section 5.1 measures the size of the instance in the number of variables. We continue by presenting bounds that are based on the usual instance measure, i.e. the number of bits needed to represent the instance.

Let $T = \langle V, C \rangle$ be an arbitrary instance of CSP($\{R\}$)-2 and let P_T be the corresponding planning instance $\langle W, A, s_I, s_G \rangle$. Recall that we have $|V|$ actions for setting the variables and at most $3|C| \leq 6|V|$ actions for checking the clauses. It follows that $||W|| = |W| \cdot \log |W|$,

$$||A|| \leq \sum_{a \in A} (||\text{pre}(a)|| + ||\text{eff}(a)||) \leq D' \cdot |V| \cdot \log |W|,$$

(for some constant D') and $||s_I||, ||s_G|| \leq |W| \cdot (\log |W| + 1)$. By recalling that $|W| \leq 3|V| + 1$, it follows that $||P_T|| \leq D \cdot |V| \log(|V|)$ for sufficiently large n and some sufficiently large constant D.

Assume that PE(PSN) can be solved in time $2^{||P||^c}$ for some $c < 1$. This implies that CSP($\{R\}$)-2 can be solved in time $2^{(D \cdot |V| \log |V|)^c}$. If $d = 1 + \frac{1-c}{2}$, then $|V| \log |V| \in O(|V|^d)$ and CSP($\{R\}$)-2 can be solved in time $2^{(D \cdot |V|^d)^c}$, too. We see that

$$2^{(D \cdot |V|^d)^c} = 2^{(D^c \cdot |V|^{c \cdot \left(1 + \frac{1-c}{2}\right)})}$$

and $c \cdot (1 + \frac{1-c}{2}) = \frac{3c - c^2}{2} < 1$ since $0 < c < 1$. This implies, in turn, that problem CSP($\{R\}$)-2 can be solved in time $2^{\epsilon \cdot |V|}$ for all $\epsilon > 0$. Hence, the ETH does not hold due to Theorem 12 and we have proved the following result.

Theorem 14 *The problem* PE(PSN) *can be solved in time* $2^{||P||/\Phi}$ *while* PE(PSN) *cannot be solved in time* $2^{||P||^c}$ *for any* $c < 1$ *unless the ETH is false.*

Since $\Phi \geq 1/2$ it follows that PE(PSN) can be solved in time $4^{||P||}$ while it cannot be solved in time $2^{||P||^c}$ for any $c < 1$ (unless the ETH is false). The second statement implies that PE(PSN) cannot be solved in time $4^{||P||^c}$ for any $c < 1$ so the bounds are matching each other quite closely.

We conclude this section by discussing some consequences of the near-optimal bounds provided by Theorem 14. By this theorem, we know that PE(PSN) cannot be solved in time $2^{||P||^c}$ for any $c < 1$. This lower bound holds even if we only consider instances having very short plans, i.e. plans ω of length $\leq |V|$. This is a direct implication of the definition of P_T where each variable can change at most one time. Define

$$X = \{P_T \mid T \text{ is an instance of CSP}(\{R\})\text{-2}\}$$

and note that PE(X) is **NP**-complete, PE(X) can be solved in time $O(2^{||P||/\Phi})$ and PE(X) cannot be solved in time $2^{||P||^c}$ for any $c < 1$.

Arbitrarily choose $Y \subseteq$ PSN such that PE(Y) is **PSPACE**-complete: one may, for instance, let Y denote the set of planning instances that results from the reduction in the proof of Theorem 3.1 in Bylander [8]. Let $Z = X \cup Y$. We see that PE(Z) can be solved in $2^{||P||/\Phi}$ time but it cannot be solved in $2^{||P||^c}$ time for any $c < 1$. Viewed slightly differently (as pointed out above), we know that PE(Z) can be solved in time $4^{||P||}$ but PE(Z) cannot be solved in time $4^{||P||^c}$ for any $c < 1$. Thus, PE(X) and PE(Z) belong to different complexity classes (under the assumption that **NP** $\neq$ **PSPACE**) but they are almost indistinguishable from the viewpoint of time complexity. Thus, we cannot, in the general case, get much concrete information about running times from statements like "PE(Z) is complete for complexity class C" when **NP** $\subseteq C$.

Another consequence of Theorem 14 is that the upper bound is obtained by basic depth or breadth search without using any heuristics at all. This may seem counterintuitive—should not the use of powerful heuristics be highly important when considering large complex instances? The answer is an emphatic "no". Domain-independent heuristic guidance can be very powerful on small and medium-size instances. For large instances, the amount of help that can be provided diminishes drastically: the advantages of heuristically pruning the search tree are too small in comparison to the enormous size of the search tree. The situation may be very different when considering restricted sets of planning instances and heuristics which are tailored to these restrictions since, in that case, Theorem 14 is not applicable.

6 Solvable vs. Unsolvable instances

An asymmetry in handling solvable and unsolvable instances has received attention in the literature recently [5, 7, 14, 24]: it has been observed that planners are often very good at finding plans but less good at verifying that no plan exists. From a theoretical point of view, this is a somewhat surprising anomaly. If we consider a tractable class of planning problems, then it is easy both to find a plan and to find that there is no plan. On the other hand, if we are faced with an **NP**-hard class, then it cannot be the case that all solvable instances are easy, and only some of the unsolvable instances are hard, as the following result shows.

Theorem 15 *Let $\mathscr{A}$ be a planning algorithm. Let C be a class of* PSN *instances such that* PE(C) *is* ***NP****-hard and let ρ be a polynomial-time reduction from 3-*SAT *to* PE(C) *with blow-up b. If algorithm $\mathscr{A}$ can generate a plan for each solvable instance in C in $2^{||P||^c}$ steps for some c such that $0 < c < 1/b$, then the ETH is not true.*

Note that the behaviour of $\mathscr{A}$ only needs to be correct for solvable instances. For unsolvable instances, it may give incorrect answers or not even terminate.

Proof Suppose there is some algorithm $\mathscr{A}$ that can generate a plan for each solvable instance of C in $2^{||P||^c}$ steps for some c such that $0 < c < 1/b$. Let F be an arbitrary 3-SAT instance and let $P_F = \rho(F)$ be the corresponding PSN instance. Simulate $\mathscr{A}$ on P_F for $2^{||P_F||^c}$ steps. If P_F is solvable, then $\mathscr{A}$ will return a correct plan for P_F, since it has to be correct for all solvable instances. On the other hand, if P_F is not solvable, then there are three possibilities:

1. $\mathscr{A}$ answers that P_F has no plan,

2. $\mathscr{A}$ does not halt within $2^{||P||^c}$ steps or
3. $\mathscr{A}$ returns an output string that is not a plan for P_F.

Obviously, if $\mathscr{A}$ does not return any output string (cases 1 and 2), then P_F has no solution. It remains to distinguish between a solvable instance and case 3. This can be done by checking whether the output string returned by $\mathscr{A}$ is a plan for P_F or not. The output string can contain at most $2^{||P_F||^c}$ symbols, since $\mathscr{A}$ does not have time to produce a longer output string, so there can be at most this many actions in the string. Each step of a plan can be verified in time $p(||P_F||)$ for some fixed polynomial p. Hence, the output from $\mathscr{A}$ can be checked in time $O(p(||P_F||) \cdot 2^{||P_F||^c})$. The construction of P_F guarantees that $\mathscr{A}$ outputs a correct plan for P_F if and only if F is satisfiable. Hence, we can check whether F is satisfiable or not in time

$$O\left(2^{||P_F||^c} + p(||P_F||) \cdot 2^{||P_F||^c}\right) \subseteq O\left((1 + p(||F||^b)) \cdot 2^{(||F||^b)^c}\right) \subseteq O\left(2^{||F||^{bc+\epsilon}}\right)$$

for all $\epsilon > 0$. Furthermore, $bc < 1$, since $c < 1/b$, so we can choose ϵ such that $0 < \epsilon < 1 - bc$. However, then $bc + \epsilon < 1$, which contradicts the ETH according to Lemma 4. □

We note that using the ETH is crucial in establishing the previous result. Assume that we, for instance, would like to use the $\mathbf{P} \neq \mathbf{NP}$ hypothesis instead of the ETH. Then, we would need to prove the following:

> Let $\mathscr{A}$ be a planning algorithm. Let C be a class of PSN instances such that $\mathsf{PE}(C)$ is **NP**-hard and let ρ be a polynomial-time reduction from 3-SAT to $\mathsf{PE}(C)$ with blow-up b. If algorithm $\mathscr{A}$ can generate a plan for each solvable instance in C in $2^{||P||^c}$ steps for some c such that $0 < c < 1/b$, then $\mathbf{P} = \mathbf{NP}$.

However, this statement is not true in general. Let X be any **NP**-complete problem. By Theorem 7, there exists a class C_X of PSN instances such that X and $\mathsf{PE}(C_\mathsf{X})$ are polynomial-time equivalent problems. This implies that $\mathsf{PE}(C_\mathsf{X})$ is **NP**-complete and there exists a polynomial-time reduction from 3-SAT to $\mathsf{PE}(C_\mathsf{X})$ with blow-up b. Assume now that $\mathscr{A}$ can generate a plan for each solvable instance in C in $2^{||P||^c}$ steps for some c such that $0 < c < 1/b$. By arguing as in the proof of Theorem 15, this merely implies that $\mathsf{PE}(C_\mathsf{X})$ can be solved in $2^{||P||^{bc}}$ steps where $0 < bc$. This fact *does not* imply that $\mathsf{PE}(C_\mathsf{X})$ can be solved in polynomial time, though, since the function $2^{||P||^{bc}}$ (with $0 < bc$) grows faster than every polynomial. Thus, we cannot draw the conclusion that $\mathbf{P} = \mathbf{NP}$.

An immediate consequence of the proof of Theorem 15 is the following.

Corollary 16 *Let $\mathscr{A}$ be a planning algorithm and let C be a class of* PSN *instances. If algorithm $\mathscr{A}$ can generate a plan for each solvable instance in C in $f(n)$ steps, then $\mathsf{PE}(C)$ can be solved in $(p(n) + 1) \cdot f(n)$ steps for some polynomial p that does not depend on C.*

In other words, if $\mathscr{A}$ is a planner that can generate plans for all solvable instances in C, then there exists a planner $\mathscr{A}'$ that is sound and complete for C and that has the same time complexity as $\mathscr{A}$ up to a polynomial factor. With this result in mind, one may speculate why planners apparently are better at analysing solvable instances than unsolvable instances in empirical evaluations. One possible explanation is that the development of planners has to a large extent been spurred by the international planning competitions [21]. However, the backside of this development is that these competitions, and thus also most planners, has been heavily focused on instances that are guaranteed to be solvable, that is, the planners and methods used are getting increasingly faster at finding solutions but not on verifying

that no solutions exist. Another possible explanation is that the test cases that have been used are not sufficiently large. Corollary 16 is an asymptotic result and a planner may very well not behave as expected when given instances that are too small.

7 Conclusions

Planning is an attractive but underused problem in theoretical computer science. It is straightforward and natural to model most problems in **PSPACE** as planning problems and we have shown that for every 'reasonable' complexity class in **PSPACE** there is a matching subproblem of PSN planning. Planning thus has the modelling power to make 'fine distinctions' between problems.

It is nowadays common to base lower bounds for problems on the assumption that the ETH is true, which typically results in a bound that is a function of the number of variables, or similar. This is not sufficient for planning, though, since the size of planning instances is not always polynomially bounded in the number of variables. Hence, we have demonstrated two methods for deriving lower bounds of the form $2^{||P||^c}$ for the time complexity of planning, both exploiting the assumption that the ETH is true. The first method works for any class of planning instances and is based on relating the constant c to the minimum blow-up in instance-size for polynomial reductions. This allowed us to determine a lower bound of $\Omega(2^{||P||^{1/3}})$ for general PSN planning, unless the ETH is false. The second method is based on a reduction from a CSPproblem with lower-bound properties analogous to the SAT problem. This method works only for a subclass of PSN planning, but gives the sharper lower bound that this subclass cannot be solved in time $4^{||P||^c}$ for any $c < 1$, unless the ETH is false, a bound that tightly matches the upper bound $O(4^{||P||})$. Considering the beneficial properties of planning mentioned above, this subclass of PSN is, thus, a very attractive problem to make reductions from when proving lower bounds for other problems.

We have finally considered the issue of solvable vs. unsolvable planning instances. Due to the International Planning Competitions (IPC), there has been strong focus on developing planners that are good at finding a solution that is known to exist, while not being equally good at determining that there is no solution. We have shown that there there is no such imbalance in theory, if looking at the worst-case time complexity of the planning problem. Finding that there is no solution is at most a polynomial factor harder than generating a solution, when there is one. We are likely to see a similar shift in performance of actual planning algorithms in the future, since the IPC has recently held its first competition on unsolvable and mixed solvable/unsolvable instances.

References

1. Bäckström, C., Jonsson, P.: All PSPACE-complete planning problems are equal but some are more equal than others. In: Proceedings 4th Annual Symposium on Combinatorial Search (SOCS-11), pp. 10–17, Barcelona, Spain (2011)
2. Bäckström, C., Jonsson, P.: Algorithms and limits for compact plan representations. J. Artif. Intell. Res. **44**, 141–177 (2012)

3. Bäckström, C., Jonsson, P.: A refined view of causal graphs and component sizes: SP-closed graph classes and beyond. J. Artif. Intell. Res. **47**, 575–611 (2013)
4. Bäckström, C., Jonsson, P., Ordyniak, S., Szeider, S.: A complete parameterized complexity analysis of bounded planning. J. Comput. Syst. Sci., 1311–1332 (2015)
5. Bäckström, C., Jonsson, P., Ståhlberg, S.: Fast detection of unsolvable planning instances using local consistency. In: Proceedings 6th Annual Symposium on Combinatorial Search (SOCS-13), pp. 29–37, WA, USA (2013)
6. Bäckström, C., Nebel, B.: Complexity results for SAS+ planning. Comput. Intell. **11**, 625–656 (1995)
7. Bogomolov, S., Magazzeni, D., Podelski, A., Wehrle, M.: Planning as model checking in hybrid domains. In: Proceedings 28th AAAI Conference on Artificial Intelligence (AAAI-14), pp. 2228–2234, QC, Canada (2014)
8. Bylander, T.: The computational complexity of propositional STRIPS planning. Artif. Intell. **69**(1-2), 165–204 (1994)
9. Chen, J., Chor, B., Fellows, M., Huang, X., Juedes, D.W., Kanj, I.A., Xia, G.: Tight lower bounds for certain parameterized NP-hard problems. Inf. Comput. **201**(2), 216–231 (2005)
10. Chen, J., Huang, X., Kanj, I., Xia, G.: Strong computational lower bounds via parameterized complexity. J. Comput. Syst. Sci. **72**(8), 1346–1367 (2006)
11. Chen, Y., Grohe, M.: An isomorphism between subexponential and parameterized complexity theory. SIAM J. Comput. **37**(4), 1228–1258 (2007)
12. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness (1979)
13. Helmert, M.: Complexity results for standard benchmark domains in planning. Artif. Intell. **143**(2), 219–262 (2003)
14. Hoffmann, J., Kissmann, P., Torralba, Á.: Distance? Who cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In: Proceedings 21st European Conference on Artificial Intelligence (ECAI-2014), pp. 441–446 (2014)
15. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. **62**(2), 367–375 (2001)
16. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity. J. Comput. Syst. Sci. **63**(4), 512–530 (2001)
17. Jonsson, P., Bäckström, C.: Tractable plan existence does not imply tractable plan generation. Ann. Math. Artif. Intell. **22**(3-4), 281–296 (1998)
18. Jonsson, P., Lagerkvist, V., Nordh, G., Zanuttini, B.: Complexity of SAT problems, clone theory and the exponential time hypothesis. In: Proceedings 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-13), pp. 1264–1277, LA, USA (2013)
19. Kanj, I., Szeider, S.: On the subexponential time complexity of CSP. In: Proceedings 27th AAAI Conference on Artificial Intelligence (AAAI-13), pp. 459–465, WA, USA (2013)
20. Levesque, H.: Logic and the complexity of reasoning. J. Philos. Log. **17**(4), 355–389 (1988)
21. Linares López, C., Celorrio, S.J., Olaya, A.G.: The deterministic part of the seventh international planning competition. Artif. Intell. **223**, 82–119 (2015)
22. Lokshtanov, D., Marx, D., Saurabh, S.: Lower bounds based on the exponential time hypothesis. B. EATCS **105**, 41–72 (2011)
23. Marx, D.: On the optiMality of planar and geometric approximation schemes. In: Proceedings 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS-07), pp. 338–348, RI, USA (2007)
24. Rintanen, J.: Generation of hard solvable planning problems. Technical report TR-CS-12-03, College of Engineering and Computer Science, The Australian National University, Canberra, Australia (2012)
25. Santhanam, R., Srinivasan, S.: On the limits of sparsification. In: Proceeding of the 39th International Colloquium on Automata, Languages, and Programming (ICALP-12), pp. 774–785, Warwick, UK (2012)
26. Savitch, W.: Relationships between nondeterministic and deterministic tape complexities. J. Comput. Syst. Sci. **4**(2), 177–192 (1970)
27. Sipser, M. Introduction to the theory of computation, 2nd edn. Thomson Course Technology (2006)
28. Stearns, R.E.: Turing award lecture: It's time to reconsider time. Commun. ACM **37**(11), 95–99 (1994)
29. Stearns, R.E.: Deterministic versus nondeterministic time and lower bound problems. J. ACM **50**(1), 91–95 (2003)
30. Stearns, R.E., Hunt, III, H.B.: Power indices and easier hard problems. Math. Syst. Theory **23**(4), 209–225 (1990)
31. Traxler, P.: The time complexity of constraint satisfaction. In: Proceeding of the 3rd International Workshop on Parameterized and Exact Computation (IWPEC-08), pp. 190–201, BC, Canada (2008)