Limits for Compact Representations of Plans

Christer Bäckström and **Peter Jonsson**
Department of Computer Science, Linköping University
SE-581 83 Linköping, Sweden
christer.backstrom@liu.se peter.jonsson@liu.se

Abstract

Most planning formalisms allow instances with shortest plans of exponential length. While such instances are problematic, they are usually unavoidable and can occur in practice. There are several known cases of restricted planning problems where plans can be exponential but always have a compact (ie. polynomial) representation, often using recursive macros. Such compact representations are important since exponential plans are difficult both to use and to understand. We show that these results do not extend to the general case, by proving a number of bounds for compact representations of plans under various criteria, like efficient sequential or random access of actions. Further, we show that it is unlikely to get around this by reformulating planning into some other problem. The results are discussed in the context of abstraction, macros and plan explanation.

1 Introduction

It is well known that many AI planning problems exhibit problem instances where the shortest plans are of exponential length, measured in the size of the instance. The reason is that instances in common planning formalisms are actually compact descriptions of exponential state-transition graphs. For instance, if we have n binary variables, the corresponding state space has 2^n states. It is easy to define planning instances where the shortest solutions must pass every state. This paper analyses the possibility of working with such instances efficiently, by compact representations of plans, or otherwise. The results are also important for plan explanations, in the sense of a planner that provides an explanation for the plan it generates. Bidot et al. (2010) suggests that it is important for planning systems (and other AI systems) to be able to explain their plans and decisions to the user, or else the user may not trust the system. Similarly, Southwick (1991) writes:

> There seems to be a general agreement amongst those involved in KBS research that in order to be useful, a system must be able to explain its reasoning to a user.

Although we will not consider any advanced explanation methods, as they do, our results have implications for what is possible to explain meaningfully.

The fact that plans may have exponential length has haunted planning reasearch for many decades, and there are many opinions about what it means and what to do about it. One way out is to say that exponential solutions are unrealistic and never occur in practice. For instance, we usually do not expect planning for a robot or an industrial plant to result in an exponential plan. On the other hand, certain archetypical AI problems are puzzle-type problems, like Towers of Hanoi. While such problems are not typical engineering applications, they pose challenges to AI research. Towers of Hanoi is not a difficult problem in principle—any common planner can solve it, if given enough time and memory. However, solving the problem is perhaps not the most interesting thing to do with it. While solutions for this problem are exponential, they can always be described compactly as recursive schemas. It would undoubtedly be a more intelligent behaviour of a planner to generate such a schema than to just list the sequence of necessary moves, and one may think of such a schema as a higher-level explanation of the plan. Clearly, the need for finding such structure in plans, that make it possible to explain them simply and compactly, increases with the size of the plans. In fact, in this case we may not even be interested in executing the plan, but just that the system can explain how the plan works.

This kind of puzzle-like problems is not restricted only to artificial problems intended to test or demonstrate intelligent behaviour. Subproblems of this kind occur also in many other domains, but their small size might often make them a minor problem. However, many technical systems today are not large and slow mechanical systems, but extremely fast hardware and/or software systems with a vast number of state variables. They can execute very long sequences of steps/instructions in short time. For such a system it can be important to verify that it can reach a desired state or, conversely, that it cannot reach certain forbidden states. Such problems are often solved by model checking techniques. It is well known that there are close ties between planning and model checking, and that model-checking traces can be viewed as plans and vice versa (Edelkamp, Leue, and Visser 2007). The number of steps (or clock cycles) can be exponential in the number of state variables, but an exponential-size plan/trace is not of much use to an engineer—it is an almost impossible task to analyse and understand such a plan. If the planning/verifying system could autonomously find

repetitive patterns, and even recursive repetitive patterns, in the plan, then it would be considerably easier to understand what happens and why. In fact, it may not be interesting to execute the plan, even in a simulator, so a compact understandable explanation of the plan may be the actual goal.

Recursive schemas, as previously mentioned, correspond to the concept of recursive macros, sometimes used in planning. For instance, Jonsson and Bäckström (1998) presented a class of planning instances where deciding if a solution exists is a polynomial-time problem, but optimal solutions may be of exponential size. Giménez and Jonsson (2008) showed that plans for this class always have a polynomial-size representation using recursive macros. Jonsson (2009) later demonstrated similar results for other classes. Although these particular classes of planning instances may not be of much practical use, the principle of compressing the solution using recursive macros is an interesting tool both for planning and explanation. This prompts the obvious question whether exponential plans can always be compressed using recursive macros (or any other method). We show that this is unlikely, no matter what type of compact representation we try to use (macro plans, finite automata or whatever). Furthermore, the results are not only, or even primarily, about representing very long plans, but implicitly also about structure in plans that scale exponentially even if short.

The remainder of the paper is organized as follows. Section 2 introduces basic notation and concepts. We then first ask, in Section 3, whether all (optimal) plans for an instance can have a compact (ie. polynomial size) representation? We find that the answer is no; it is not possible, neither by macros nor any other method. In Section 4 we restrict the question to whether there exists at least one plan for each instance having a compact representation? We show that such representations do not generally exist if they must also satisfy some useful access criterion. As a last resort we analyse, in Section 5, whether we can get around the problem by reformulating planning to some other problem? Also this is answered negatively. If we actually ask for a plan for the original problem, then the problem is inherently intractable also when using reformulation. If looking only at the decision problems, it seems not possible to make planning simpler either, by reformulation. The paper ends with a three-part discussion in Section 6. We first discuss the connections between recursive macros, abstraction and causal graphs. Then we discuss the implications of our results for plan explanation, rather than planning. Finally, we discuss some related results in the literature for more expressive formalisms, and also note that our results hold even for restricted cases, such as propositional STRIPS with unary actions.

2 Preliminaries

Here we briefly define some concepts used in the paper.

2.1 STRIPS Planning

Although we consider planning in general terms, not restricted to any particular planning language, all theorems will be proven for propositional STRIPS or restricted variants thereof, in order to make the results strong. We assume the reader to be familiar with STRIPS and just note that we consider one of several equivalent variants of propositional STRIPS (Bäckström 1995).

Definition 1. An *atom* is a binary variable and a *literal* is either an atom x or its negation $\overline{x}$ (which has the opposite value of x). An *instance* of the STRIPS *planning problem* is a quadruple $\mathbb{P} = \langle V, A, I, G \rangle$, where V is a set of atoms, $I, G \subseteq V$ are the *initial* and *goal descriptions* and A is a set of *actions*. Each action in A is a tuple $\langle \mathrm{pre}, \mathrm{post} \rangle$, where pre and post are sets of literals over V. For action definitions, we use the notation $name : \mathrm{pre} \Rightarrow \mathrm{post}$, where $name$ is used to refer to the action. A *plan* is an action sequence that solves the instance, ie. if executing the sequence, starting in a state satisfying I, it ends up in a state satisfying G.

The following construction will be frequently used.

Construction 1. An n-bit binary counter can be encoded in STRIPS by letting $V = \{x_1, \ldots, x_n\}$ and A have n actions s.t. for all i, where $1 \leq i \leq n$,

$$a_i : \overline{x_i}, x_{i-1}, \ldots, x_1 \Rightarrow x_i, \overline{x_{i-1}}, \ldots, \overline{x_1}.$$

2.2 Computational Complexity

First a note on notation, by $|X|$ we mean the number of elements of an object X and by $||X||$ we mean its size, ie. the number of bits of its representation. We will also make heavy use of the 3SAT problem and advice-taking Turing machines, and, thus, briefly recapitulate their definitions.

Definition 2. The *3SAT problem* consists of instances on the form $C = \{c_1, \ldots, c_m\}$ where each c_i, for $1 \leq i \leq m$, is called a *clause* and is a set of exactly three literals over some universe of binary variables. The instance C is *satisfiable* if there exists some assignment of truth values to the variables used in C s.t. at least one literal is true in each c_i, for $1 \leq i \leq m$, and it is otherwise *unsatisfiable*.

Deciding satisfiability for 3SAT is NP-complete, while deciding unsatisfiability is coNP-complete.

Definition 3. An *advice-taking Turing machine* M has an associated sequence $a_1, a_2, a_3, \ldots$ of *advice strings*, a special advice tape and an *advice function* a, from the natural numbers to the advice sequence, s.t. $a(n) = a_n$. On input x the advice tape is immediately loaded with $a(||x||)$. After that M continues in the normal way, except that it also has access to the advice written on the advice tape.

If there exists a polynomial p s.t. $||a(n)|| \leq p(n)$, for all $n > 0$, then M is said to use *polynomial advice*. The complexity class P/poly is the set of all decision problems that can be solved on some advice-taking TM that runs in polynomial time using polynomial advice. This can be extended s.t., for instance, NP/poly is defined by TMs runnning in non-deterministic polynomial time using polynomial advice.

Note that the advice depends only on the size of the input, not its content, and need not even be computable.

3 Representing Arbitrary Plans Compactly

Knowing that exponential plans can sometimes be represented compactly, we might ask if all exponential plans have such compact encodings. Consider the most simple compact

notation possible, an index number i for each plan for a particular instance. Since instances may have infinitely many plans, due to cycles in the state-transition graph, we consider optimal plans only.

Such an index could not be used instead of an actual plan, but it could be a way to store exponential plans efficiently. Consider two extreme cases. The first case is purely theoretical—an oracle reads the planning instance and immediately returns a huge data structure which we can index to get the actual plan we want. The second case is possible in practice—we store the planning instance together with the plan index and the planning algorithm. Assuming that the algorithm is deterministic and can enumerate all optimal plans, we can always find the ith plan again, by using the planning algorithm. While this is very inefficient and may seem ridiculous at first sight, it could have practical use. Suppose we frequently analyse systems having many exponential plans and want to store these plans, or some of them, for future reference. Also suppose we store some relevant properties of the plans along with their indices, so we can know which plans are interesting to retrieve in certain cases. If many plans need to be stored, but very few of them need ever be retrieved again, then the method described could pay off in saved disk space, under the condition that the plan indices are small compared to the actual plans.

Unfortunately, there is no guarantee that even such an index is much smaller than the plan itself—a polynomial number of bits may not be sufficient for the index.

Theorem 1. *There are* STRIPS *instances where the number of optimal plans requires an exponential number of bits to be represented.*

Proof. We first prove that there are n-variable STRIPS instances having $2^{O(2^n)}$ unique optimal solutions. Consider the n-bit binary counter in Construction 1, but add an extra atom y and define two families of incrementing actions s.t. for all i, where $1 \leq i \leq n$,

$$a_i : \overline{x_i}, x_{i-1}, \ldots, x_1 \Rightarrow x_i, \overline{x_{i-1}}, \ldots, \overline{x_1}, \overline{y}$$
$$b_i : \overline{x_i}, x_{i-1}, \ldots, x_1 \Rightarrow x_i, \overline{x_{i-1}}, \ldots, \overline{x_1}, y$$

This extra atom does not matter to the counter, but doubles the state space Also note that the number of actions is linear in n, so the size of the whole instance is polynomial in n. Clearly, all optimal plans counting from 0 to $2^n - 1$ are of length $2^n - 1$. However, there are 2^{2^n-1} such plans achieving the goal (half of them resulting in y true). Since an m-bit number can distinguish between at most 2^m different objects, a string of length $\geq 2^n - 1$ bits is necessary to index the optimal plans for this instance. □

Although the atom y is redundant in this particular example the whole construction could be a part of a larger instance, where y does have a purpose.

4 Representing one Plan Compactly

We now know that we cannot, in general, compress arbitrary exponential plans to subexponential size. But what if we do not get to choose ourselves which plan to use? The previous result still leaves open the possibility that a small fraction of solutions for a planning instance could have compact representations. However, the planner (or an oracle or whatever) would then have to choose for us which plan to present us with a compact representation of. Suppose a planner could actually do this, how would we make use of it? If we still need the actual plan itself, we cannot avoid its exponential size, so the interesting cases seem to be if we could at least access useful information efficiently.

The term representation is used in a loose sense here, but need not really be precisely defined. It suffices to note that any representation needs both some kind of data structure and some kind of access algorithm, with the extreme cases being either a vector of data with the trivial access algorithm or an algorithm that embeds all the data.

What could it mean to access a compact representation efficiently? We will investigate two such criteria. The first one is that we can efficiently retrieve the actions of the actual plan sequentially. Efficient can here mean either *polynomial delay*, ie. each action takes at most polynomial time in the size of the instance to retrieve, or *incremental polynomial time*, which is like polynomial delay but measured in the size of the input and the size of the output generated so far, ie. we get more time for each action at the end of the plan than in the beginning. The second criterion is that any action in the actual plan can be random accessed in polynomial time, in the size of the instance.

Theorem 2. *If there is an algorithm that for any solvable* STRIPS *instance can either generate a plan sequentially in incremental polynomial time or random access any action in some plan in polynomial time, then* $\mathrm{P} = \mathrm{NP}$.

(Proofs of Theorems 2 and 4 appear later in this section). This is a very strong criterion, since we require one single algorithm for all instances. A more relaxed variant is to ask for a compact representation of some plan for each instance.

Definition 4. Let p, q be arbitrary polynomials, $\mathbb{P}$ an arbitrary solvable planning instance, ω a plan for $\mathbb{P}$ and ρ some representation of ω, then

1) ρ is *compact* iff $||\rho|| \leq p(||\mathbb{P}||)$,

2) ρ is *sequential-access* iff it can retrieve all actions in ω sequentially in incremental polynomial time and

3) ρ is *random-access* iff, given an arbitrary index i, $1 \leq i \leq |\omega|$, ρ can retrieve action i in ω in time $\leq q(||\rho||)$.

A representation ρ is a *compact sequential-access representation* (CSAR) iff it satisfies conditions 1 and 2, and it is a *compact random-access representation* (CRAR) iff it satisfies conditions 1 and 3.

Note that this definition does not even assume that the representations are computable, but only that they exist.

Theorem 3. *If every solvable* STRIPS *instance* $\mathbb{P}$ *has at least one plan* ω *with a corresponding* CSAR ρ *that can be verified in polynomial time, then* $\mathrm{NP} = \mathrm{PSPACE}$.

Proof. Deciding if a STRIPS instance has a plan can be done by guessing a CSAR ρ for some such plan and then verify ρ in polynomial time. It follows that $\mathrm{NP} = \mathrm{PSPACE}$ since deciding plan existence is PSPACE-complete for STRIPS (Bylander 1994). □

Theorem 4. *If every solvable* STRIPS *instance $\mathbb{P}$ has at least one plan ω with a corresponding* CRAR *ρ, then the polynomial hierarchy collapses.*

Theorems 2 and 3 also impliy the analogous results for polynomial delay, since this is a stricter criterion than incremental polynomial time. Also note that neither of the three theorems above require that we ask for optimal plans.

4.1 Proof of Theorem 2

The proof first requires some additional theory.

Definition 5. For all $n > 0$, let $X_n = \{x_1, \ldots, x_n\}$ be a standardized set of variables and let $m(n)$ be the number of possible 3-literal clauses over X_n. Let $c_n^1, c_n^2, \ldots, c_n^{m(n)}$ be some systematic enumeration of these clauses and C_n the set of them all. Each clause defines three literals s.t. $c_n^i = \{\ell_i^1, \ell_i^2, \ell_i^3\}$[1]. Further, let $C_n^0, C_n^1, \ldots, C_n^{2^{m(n)}-1}$ be a systematic enumeration of all subsets of C_n, and let $\mathbb{S}_n^i = \langle X_n, C_n^i \rangle$, for $0 \leq i < 2^{m(n)}$. Also define the set $E_n = \{e_n^1, e_n^2, \ldots, e_n^{m(n)}\}$ of atoms and its subsets $E_n^i = \{e_n^j \mid c_n^j \in C_n^i\}$, for $0 \leq i < 2^{m(n)}$.

The sequence $\mathbb{S}_n^0, \mathbb{S}_n^1, \ldots, \mathbb{S}_n^{2^{m(n)}-1}$ is a systematic enumeration of all possible 3SAT instances over n variables, and hence equivalent to the usual definition of 3SAT[2]. Since $m(n) < 8n^3$, the enumerations of C_n and E_n can be chosen to be polynomial-time computable, and we assume some such enumerations have been fixed from now on. We also note that a set E_n^i uniquely identifies the clause set C_n^i. 3SAT instances can now be encoded as STRIPS instances as follows.

Construction 2. Given an $n > 0$, construct the STRIPS instance $\mathbb{P}_n^i = \langle V_n, A_n, E_n^i, \{goal\} \rangle$ s.t. $V_n = X_n \cup E_n \cup \{cts, ctu, goal, inc\} \cup \{v_0, \ldots, v_{m(n)}\}$ and A_n has the actions specified in Table 1.

Lemma 1. *For each 3SAT instance $\mathbb{S}_n^i$, a corresponding solvable* STRIPS *instance $\mathbb{P}_n^i$, according to Construction 2, can be constructed in polynomial time and has the property: if $\mathbb{S}_n^i$ is satisfiable, then every plan for $\mathbb{P}_n^i$ starts with action acs and otherwise every plan starts with action acu.*

Proof. Solvability and polynomial-time construction is trivial. To prove the property, we first note that the initial state contains only atoms from E_n. As previously noted, each subset E_n^i of E_n uniquely identifies the 3SAT instance $\mathbb{S}_n^i$, by telling which clauses in C_n are 'enabled' in $\mathbb{S}_n^i$.

The actions are partitioned into three groups, the first containg actions acs and acu, which set atoms cts and ctu, and the other two blocks consist of actions requiring either of these atoms to be true. Obviously any plan must start with either acs or acu, and since they block each other, only one

[1]We sometimes omit index n, when it can be assumed obvious from context, and thus write ℓ_i^k rather than and $\ell_{n,i}^k$.

[2]Technically speaking, this is a redundant encoding of 3SAT, since it allows instances that specify more variables than are used in the clauses. This is harmless, however, since all non-redundant instances remain.

$acs\colon \overline{ctu} \Rightarrow cts$ $acu\colon \overline{cts} \Rightarrow ctu$
$aset_i\colon cts, \overline{v_0} \Rightarrow x_i$ $avt_0\colon cts \Rightarrow v_0$ $avt_j^0\colon cts, \overline{e_n^j}, v_{j-1}, \Rightarrow v_j$ $avt_j^k\colon cts, e_n^j, v_{j-1}, \ell_j^k \Rightarrow v_j$ $ags\colon cts, v_{m(n)} \Rightarrow goal$
$avf_j\colon ctu, \overline{inc}, e_n^j, \overline{\ell_j^1}, \overline{\ell_j^2}, \overline{\ell_j^3} \Rightarrow inc$ $aix_i\colon ctu, inc, \overline{x_i}, x_{i-1}, \ldots, x_1 \Rightarrow \overline{inc}, x_i, \overline{x_{i-1}}, \ldots, \overline{x_1}$ $agu\colon ctu, inc, x_1, \ldots, x_n \Rightarrow goal$

Index ranges: $1 \leq i \leq n$, $1 \leq j \leq m(n)$ and $1 \leq k \leq 3$.

Table 1: Actions for Construction 2.

of them can appear in a plan, making cts and ctu mutually exclusive. Hence, in the first action the plan commits to verifying either satisfiability (starting with action acs) or unsatisfiability (starting with action acu).

Wlog. we assume the plan is optimal. If it verifies satisfiability, it must be

$$\langle acs, \underbrace{aset_{i_1}, \ldots, aset_{i_h}}_{\text{assign}}, avt_0, \underbrace{avt_1^{k_1}, \ldots, avt_{m(n)}^{k_{m(n)}}}_{\text{verify}}, ags \rangle.$$

The assign block has $h \leq n$ actions that set a satisfying assignment for $x_1, \ldots, x_n$. The verify block consists of one action $a = avt_j^{k_j}$ for each clause c_n^j. If c_n^j is enabled (e_n^j true), then $1 \leq k_j \leq 3$ and a verifies that ℓ_j^k in c_n^j is true for the assignment. Otherwise, if c_n^j is disabled, then $k_j = 0$, so $a = avt_j^0$ which skips over c_n^j without verifying anything. The planner has thus chosen: 1) to verify that $\mathbb{S}_n^i$ is satisfiable, 2) a satisfying assignment and 3) one literal for each clause as a witness that it is true under this assignment.

If the plan instead verifies unsatisfiability, then it must be $\langle acu, b_0, a_1, b_1, a_2, b_2, \ldots, a_h, b_h, agu \rangle$, which, apart from acu and agu, can be viewed as two interleaved sequences $\alpha = \langle a_1, \ldots, a_h \rangle = \langle aix_1, aix_2, aix_1, aix_3, \ldots, aix_1 \rangle$ and $\beta = \langle b_0, b_1, \ldots, b_h \rangle$. Sequence α is a binary counter sequence, counting from 0 to $h = 2^n - 1$, thus enumerating all 2^n possible truth assignments to $x_1, \ldots, x_n$. Sequence β has one action b_i for each truth assignment, where $b_i = avf_j$ for some enabled clause c_n^j that is false for the current assignment. Hence, the plan verifies that none of all possible truth assignments can satisfy $\mathbb{S}_n^i$, providing a witness for each.

The plan can be of the first form only if $\mathbb{S}_n^i$ is satisfiable and on the second form only if $\mathbb{S}_n^i$ is unsatisfiable, so the first action in the plan can be used to tell which is the case. □

This can be understood by viewing the planner as a theorem prover, outputting first a theorem (the first action in the plan) and then a proof of the theorem (the rest of the plan).

Proof. (of Theorem 2) Suppose there is an algorithm with either sequential or random access as stated in the precondition of the theorem. We can then solve any 3SAT instance

$abi\colon \overline{svi}, \overline{sva}, \overline{sia}, \overline{sii}, \overline{sti} \Rightarrow svi, \overline{t}$
$aba\colon svi, \overline{sia} \Rightarrow sva, \overline{f}, v_0, \overline{v_1}, \ldots, \overline{v_{m(n)}}$
$avt_j^k\colon sva, \overline{v_j}, v_{j-1}, e_n^j, \ell_j^k \Rightarrow v_j$ $avf_j\colon sva, \overline{v_j}, v_{j-1}, e_n^j, \overline{\ell_j^1}, \overline{\ell_j^2}, \overline{\ell_j^3} \Rightarrow v_j, f$ $avs_j\colon sva, \overline{v_j}, v_{j-1}, \overline{e_n^j} \Rightarrow v_j$
$aaf\colon sva, v_{m(n)}, f \Rightarrow \overline{sva}, sia$ $aat\colon sva, v_{m(n)}, \overline{f} \Rightarrow \overline{sva}, sia, t$
$aix_i\colon sia, \overline{x_i}, x_{i-1}, \ldots, x_1 \Rightarrow \overline{sia}, x_i, \overline{x_{i-1}}, \ldots, \overline{x_1}$ $arx\colon sia, x_n, \ldots, x_1 \Rightarrow \overline{sia}, \overline{svi}, sti, \overline{x_n}, \ldots, \overline{x_1}$
$ais\colon sti, t \Rightarrow \overline{sti}, sii$ $aiu\colon sti, \overline{t} \Rightarrow \overline{sti}, sii$
$aii_j\colon sii, \overline{e_n^j}, e_n^{j-1}, \ldots, e_1 \Rightarrow \overline{sii}, e_n^j, \overline{e_n^{j-1}}, \ldots, \overline{e_n^1}$ $ari\colon sii, e_n^{m(n)}, \ldots, e_n^1 \Rightarrow goal$

Index ranges: $1 \leq i \leq n$, $1 \leq j \leq m(n)$ and $1 \leq k \leq 3$.

Table 2: Actions for Construction 3.

$\mathbb{S}_n^i$ in polynomial time by asking the algorithm for the first action of some plan for the corresponding instance $\mathbb{P}_n^i$ and tell from this action whether $\mathbb{S}_n^i$ is satisfiable. However, this implies that P = NP. □

4.2 Proof of Theorem 4

This proof also requires additional theory.

Construction 3. Given an $n > 0$, construct a STRIPS instance $\mathbb{P}_n = \langle V_n, A_n, \varnothing, \{goal\}\rangle$ s.t. $V_n = X_n \cup E_n \cup \{v_0, \ldots, v_{m(n)}\} \cup \{svi, sva, sia, sii, sti, t, f, goal\}$ and A_n has the actions specified in Table 2.

Lemma 2. *For all $n > 0$, instance $\mathbb{P}_n$ according to Construction 3 can be constructed in time polynomial in n and has the properties: 1) $\mathbb{P}_n$ always has at least one solution and 2) there exist constants a_n and b_n s.t. for every i, where $0 \leq i < 2^{m(n)}$, we can tell from position $b_n i + a_n$ in any plan for $\mathbb{P}_n$ if $\mathbb{S}_n^i$ is satisfiable or not.*

Proof. (Sketch) Construction 2 allows plans of two types, either choosing an assignment and then verifying all clauses by chaining, or enumerating all assignments and demonstrate one false clause for each. Construction 3 mixes these methods. Any plan enumerates all assignments, and for each, it walks through all clauses by chaining. For each enabled clause it demonstrates either a true literal or that no literal is true, and it skips over disabled clauses. Atoms f and t keep track of whether all clauses were true for some assignment, in which case the instance is satisfiable.

An extra counter, using $e_n^1, \ldots, e_n^{m(n)}$, enumerates all possible subsets E_n^i of E_n, thus implicitly enumerating all 3SAT instances $\mathbb{S}_n^0, \ldots, \mathbb{S}_n^{2^{m(n)}-1}$. This counter constitutes an 'outer loop', so for each E_n^i, all possible assignments for $x_1, \ldots, x_n$ are tested as described above.

Setting $a_n = 2^n(m(n)+3)+2$ and $b_n = a_n + 1$ satisfies the claim, since the action at position $b_n i + a_n$ is ais if $\mathbb{S}_n^i$ is satisfiable and aiu if it is unsatisfiable. □

Proof. (of Theorem 4) Suppose all solvable STRIPS instances $\mathbb{P}$ have at least one plan with a corresponding CRAR. For each $n > 0$ and instance $\mathbb{P}_n$ according to Construction 3, choose one such plan ω_n and some corresponding CRAR ρ_n.

Construct an advice-taking TM M, with input $\mathbb{I}_n^i = \langle \mathbb{P}_n, i\rangle$, for $n > 0$, $0 \leq i < 2^{m(n)}$ and i in binary using $m(n)$ bits. Clearly, $||\mathbb{I}_n^i||$ is strictly increasing and depends only on n, so let $s_n = ||\mathbb{I}_n^i||$ (for arbitrary i). Define the advice function a s.t. $a(s_n) = \rho_n$. Since M can simulate whatever algorithm is used to access ρ_n, it follows from the assumptions that M can find action $b_n i + a_n$ in ω_n in polynomial time and return yes if this action is ais, and otherwise no.

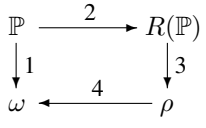
Given an arbitrary $\mathbb{S}_n^i$, construct $\mathbb{I}_n^i$ in polynomial time and then run M on this instance. By construction, M answers yes iff $\mathbb{S}_n^i$ is satisfiable. However, M runs in polynomial time using polynomial advice and solves satisfiability for 3SAT. Hence, NP $\subseteq$ P/poly, which is impossible unless the polynomial hierarchy collapses at level 2 (Karp and Lipton 1980, Theorem 6.1). □

5 Problem Reformulation

Having now concluded that there seems to be little hope that plans can be compactly represented in general, we turn to the idea of problem reformulation, to see if that can be of any help. While this may seem out of place in this context, it is, to the contrary, a quite logical step to take. So far, we have only analysed planning problems and plans, and that is what the results hold for. It is not obvious that, or when, the results hold also when planning instances are solved by reformulating them to instances of some other problem, so it is hypothetically possible that we could get around the problems with this approach. However, to say something useful and relevant about this, it is not sufficient to look only at naive approaches, such as polynomial reductions, so we will investigate a stronger criterion.

The basic idea of reformulation is to transform a planning instance to another instance, either another planning instance or an instance of some other problem. For reformulation to be useful, the solution for the new instance must be of use to solve the original instance, and something should be gained. Often, reformulation is used with the intention that the overall process is faster than solving the original instance directly. Common variants are to reformulate planning into SAT, CSP, model checking or another planning problem. Reformulation to SAT was first suggested by Kautz and Selman (1992) and is still a popular approach to planning. Long et al. (2002) discuss reformulation for planning in general and Edelkamp et al. (2007) discuss the connections between model checking and planning.

The reformulation process can be viewed as follows:



A planning instance $\mathbb{P}$ has a solution ω that we can find using ordinary planning, indicated by arrow 1. Solving $\mathbb{P}$ via

reformulation instead follows the path 2,3,4. First $\mathbb{P}$ is reformulated into a new instance $R(\mathbb{P})$ (of some problem), then this instance is solved resulting in a solution ρ and, finally, ρ is transformed back into a solution ω for $\mathbb{P}$.

Obviously, reformulation cannot help us when plans must be exponential. Even if steps 2 and 3 were polynomial and ρ of polynomial size, it would necessarily take exponential time to transform ρ into ω, because ω must be exponential. That is, the problem is inherently intractable, whichever method we use to solve it. However, reformulation could potentially speed things up, if ρ could somehow be used directly as a solution for the original problem. While such cases might perhaps occur, it is not the general case.

A special case, though, is if we consider the decision problem, rather than the generation problem, ie. we are not asking for a plan, but whether a plan exists or not. In this case we can use the solution for $R(\mathbb{P})$ directly, since decision problems have only two possible answers, yes and no. We may thus escape the inherent intractability. Since no exponential solution is generated, reformulation could potentially be more efficient. We know that the decision problem for STRIPS is PSPACE-complete in the general case (Bylander 1994, Theorem 3.1). If the reformulated problem were easier to solve, then it could be beneficial to first reformulate $\mathbb{P}$ to $R(\mathbb{P})$ and ask if that instance has a solution or not. Then it would be possible to check if there is a solution at all before embarking on generating a possibly exponentially long plan. (Cf. the case of 3S (Jonsson and Bäckström 1998), where plans may be of exponential size but it is always possible to decide in polynomial time if there is a plan.) It thus seems like the case of reformulating decision problems is the most interesting one to look at, and if that does not give any improvement, then there can hardly be any improvement for plan generation via reformulation either.

Let $\mathcal{S}$ denote the decision problem for STRIPS. The following two results are trivial, but illustrative.

Theorem 5. *a) There exists a decision problem $\mathcal{X}$ and a function R s.t. it holds for all $\mathbb{P} \in \mathcal{S}$ that $R(\mathbb{P}) \in \mathcal{X}$ and that $\mathbb{P}$ and $R(\mathbb{P})$ have the same answer. b) If there is some complexity class C, some decision problem $\mathcal{X} \in C$ and a polynomial-time computable function R s.t. it holds for all $\mathbb{P} \in \mathcal{S}$ that $R(\mathbb{P}) \in \mathcal{X}$ and that $\mathbb{P}$ and $R(\mathbb{P})$ have the same answer, then* PSPACE $\subseteq C$.

Proof. a) Let $\mathcal{X} = \mathcal{S}$ and R the identity function. b) Immediate, since R is a polynomial reduction from $\mathcal{S}$ to $\mathcal{X}$. □

In both cases we reformulate a PSPACE-complete problem into a PSPACE-complete problem, which is not very interesting. If we are to prove anything better, we must obviously look for an $\mathcal{X}$ and an R with more useful restrictions.

It is important to note that when reformulating planning into some NP-complete problem, for instance SAT, this does not magically make planning NP-complete. The reason that STRIPS planning is PSPACE-complete is that it allows exponential solutions. As soon as we restrict the solutions to be bounded by some fixed polynomial, planning belongs in NP. Furthermore, encodings of planning instances in SAT, typically use atoms to encode what actions appear at each position in the plan, ie. an exponential number of extra atoms are required in the general case. Hence, either our problem was already in NP, or we have to blow up the instance exponentially when reformulating to SAT. In the latter case, the complexity results are no longer comparable. Also note, that if we deliberately restrict ourselves to ask only if there is a plan of a certain length or shorter, then we are actually solving a restricted version of the optimization problem, and also in this case, planning itself would be no harder. In fact, it seems most unlikely that planning in general could be reformulated into a problem in NP.

Definition 6. Given a STRIPS instance $\mathbb{P} = \langle V, A, I, G\rangle$, define the alternative notation $\mathbb{P} = \langle \Phi, \Psi\rangle$, where $\Phi = \langle V, A\rangle$ and $\Psi = \langle I, G\rangle$. Let $\mathcal{X}$ be some decision problem. A *reformulation* of $\mathcal{S}$ into $\mathcal{X}$ is a pair $\langle R, r\rangle$ of functions that maps every instance $\mathbb{P} = \langle \Phi, \Psi\rangle \in \mathcal{S}$ to a corresponding instance $\mathbb{X} = R(r(\Phi), \Psi) \in \mathcal{X}$ s.t. $\mathbb{P}$ and $\mathbb{X}$ have the same answer. $\langle R, r\rangle$ is a *polynomial reformulation* iff there are also some fixed polynomials p, q s.t. 1) $||r(\Phi)|| \leq p(||\Phi||)$ and 2) R is computable in time $\leq q(||r(\Phi)|| + ||\Psi||)$.

We thus consider a reformulation that involves two functions, R and r. Function r is the main reformulation function, intended to reformulate the difficult part of the instance. We do not even require this function to be computable, only that it exists. Function R is then used to transform the initial and goal descriptions into something similar that the new instance can use, and combine this with the result delivered by r into a proper instance of $\mathcal{X}$.

Theorem 6. *There is no polynomial reformulation of $\mathcal{S}$ to some $\mathcal{X} \in$ NP, unless the polynomial hierarchy collapses.*

Proof. Suppose $\langle R, r\rangle$ is such a reformulation. For arbitrary $n > 0$, let $\Phi^u_n = \langle V_n, A_n\rangle$ as in Construction 2, but without action acs, and let $\Psi^i_n = \langle E^i_n, \{goal\}\rangle$, for $0 \leq i < 2^{m(n)}$. It follows trivially from the proof of Lemma 1 that instance $\mathbb{P}^i_n = \langle \Phi^u_n, \Psi^i_n\rangle$ has a solution iff $\mathbb{S}^i_n$ is unsatisfiable (note that the 'SAT part' of the instance is 'disarmed').

Construct a non-deterministic advice-taking TM M with input $\mathbb{I}^i_n = \langle \Phi^u_n, i\rangle$, for all $n > 0$ and $0 \leq i < 2^{m(n)}$, representing i in binary using $m(n)$ bits. Clearly, $||\mathbb{I}^i_n||$ is strictly increasing and depends only on n, so let $s_n = ||\mathbb{I}^i_n||$ (for arbitrary i). Define the advice function a s.t. $a(s_n) = r(\Phi^u_n)$. Let M first compute Ψ^i_n from $\mathbb{I}^i_n$, and then compute $\mathbb{X}^i_n = R(a(s_n), \Psi^i_n) = R(r(\Phi^u_n), \Psi^i_n)$, both in polynomial time since $a(s_n)$ is given for free as advice. By assumption, $\mathbb{X}^i_n \in \mathcal{X}$ and has answer yes iff $\mathbb{P}^i_n$ has a solution. Also by assumption, $\mathcal{X} \in$ NP, so M can solve $\mathbb{X}^i_n$ by guessing a solution and verifying it in polynomial time. Hence, deciding if $\mathbb{P}^i_n$ has a solution is in NP/poly.

For arbitrary $\mathbb{S}^i_n$, compute $\mathbb{I}^i_n$ in polynomial time. M answers yes for $\mathbb{I}^i_n$ iff $\mathbb{S}^i_n$ is unsatisfiable, but unsatisfiability for 3SAT is coNP-complete, so coNP $\subseteq$ NP/poly. This is impossible unless the polynomial hierarchy collapses to level 3 (Yap 1983, Lemma 7 + Theorem 2). □

This result can be pushed arbitrarily high up in the polynomial hierarchy, thus making it unlikely that planning could be reformulated to anything simpler at all.

Corollary 1. *There is no polynomial reformulation $\langle R, r\rangle$ of $\mathcal{S}$ to some decision problem $\mathcal{X} \in \Sigma_k^p$, for $k > 1$, unless the polynomial hierarchy collapses to level $k + 2$.*

Proof. (Sketch) Construction 2 encodes both existential quantification (choosing a truth assignment in the sat part) and universal quantification (enumerating all truth assignments in the unsat part). Hence, it is straightforward to modify it to an analogous construction for QBF formulae with k alternations. Given that, the rest of the proof is analogous to the proof of Theorem 6, but M must use an oracle for Σ_{k-1}^p. The same argument leads to $\Pi_k^p \subseteq \Sigma_k^p/\mathrm{poly}$, which is impossible unless the polynomial hierarchy collapses to level $k + 2$ (Yap 1983, Lemma 7 + Theorem 2). □

6 Discussion

The discussion section is divided into three parts. First we discuss the use of macros for storing plans compactly, and the related concepts of state abstraction and causal graphs. We then discuss what implications the results in this paper may have on the problem of explaning plans. We conclude with a discussion on some other topics of the paper.

6.1 Macros, Abstraction and Causal Graphs

A macro is a sequence of two or more actions, treated as one single action corresponding to such a subplan. Recursive macros may also contain other macros, not only actions, making them a powerful tool. Restricted planning problems have been demonstrated in the literature where optimal plans may be of exponential size, but can always be represented in polynomial size using recursive macros (Giménez and Jonsson 2008; Jonsson 2009). The concept of macros is closely related to state abstraction, as Knoblock noted (1993, pp. 110–111). Indeed, the refinement process in the hierarchical state abstraction process can result in a small initial abstract plan being successively refined into exponentially larger plans (Bäckström and Jonsson 1995).

Consider a counter that counts in Gray code, ie. successive numbers have Hamming distance 1. An n-bit Gray code counter can be encoded in STRIPS as follows: let $V = \{x_1, \ldots, x_n\}$ and let A contain the $2n$ actions

$$s_i\colon \overline{x_i}, x_{i-1}, \overline{x_{i-2}} \ldots, \overline{x_1} \Rightarrow x_i$$
$$r_i\colon x_i, x_{i-1}, \overline{x_{i-2}} \ldots, \overline{x_1} \Rightarrow \overline{x_i}$$

We note that in this case, all actions are unary, ie. have only one atom each in their postconditions. A plan for a 5-bit Gray counter counting from 0 to 16 looks like

$$\Big\langle \underbrace{\underbrace{\underbrace{s_1, s_2, r_1}_{m_2^s}, s_3, \underbrace{s_1, r_2, r_1}_{m_2^r}}_{m_3^s}, s_4, \underbrace{\underbrace{s_1, s_2, r_1}_{m_2^s}, r_3, \underbrace{s_1, r_2, r_1}_{m_2^r}}_{m_3^r}}_{m_4^s}, s_5 \Big\rangle,$$

where we have indicated the following macro choice:

$$m_2^s = (s_1, s_2, r_1), \quad m_2^r = (s_1, r_2, r_1),$$
$$m_3^s = (m_2^s, s_3, m_2^r), \quad m_3^r = (m_2^s, r_3, m_2^r),$$
$$m_4^s = (m_3^s, s_4, m_3^r).$$

The plan can thus be compactly represented as $\langle m_4^s, s_5\rangle$, where the macro m_4^s can be thought of as the root of a macro tree which expands recursively into a macro-free sequence. In general, plans for such Gray counters can be represented by linear-size macro plans by adding a linear number of macros. The same holds also for binary counters, so both are examples of planning instances where plans can be exponential but always have a compact representation. However, we also know from the results in this paper that, in the general case, it is not possible to find a compact representation of exponential plans, by macros or whatever method. It would thus be interesting to find useful criteria for when compact representations exist and not. This problem is out of the scope of this paper, so let us just make a few observations.

Knoblock (1993) defined the concept of a causal graph for a planning instance, which he used as a guidance for finding abstraction hierarchies. This concept has later been used also to define classes of tractable planning problems, without considering abstraction (see Chen and Giménez (2010) for a survey). We refer the reader to Knoblock for definitions, but simply put, the causal graph induces an order on the atoms based on the action definitions, such that if $x \leq y$ and not $y \leq x$, then x does not depend on y and can be put on a lower abstraction level than y.

If applying this concept to examples in this paper, we find that instances according to Construction 2 have causal graphs where everything is related to everything, making the whole graph one big component. That is, it would not be possible to form any abstraction hierarchies for it based on causal graphs. On the other hand, the Gray counter has a nice linear causal graph, with no cycles at all, making it trivial to solve efficiently with abstraction. Unfortunately, the binary counter, that also allows for compact representations of its plans, does not have such a nice causal graph. It has one big component, just as Construction 2. This is not surprising, however, since causal graphs can be cycle free only if all actions are unary. We conclude that the causal graph is not a sufficient, or even necessarily useful, criterion for when plans have compact representations.

Jonsson (2009) has, however, defined a refined version of the causal graph. We refer to Jonsson for details, and just note that if using such refined causal graphs, then both the Gray counter and the binary counter have nice linear graphs, while Construction 2 still has a one-component graph. Perhaps such refined causal graphs could provide some information on when plans can be compressed and not?

6.2 Implications for Plan Explanation

For plan explanation, the results are not necessarily as bad as for planning. Consider for instance a plan for an instance of Construction 2. In the case where the 3SAT instance is unsatisfiable, the whole plan but a few actions consists of an alternating sequence on the form $\langle a, b, a, b, a, b, \ldots\rangle$, where a denotes either of the actions $aix_1, \ldots, aix_n$ and b denotes either of the actions $avf_1, \ldots, avf_m$. The first group are actions that together implement an increment function, and thus all serve the same purpose. Similarly, the second group consists of actions that all serve the purpose of verifying that

some clause is false. For the purpose of explanation, it seems useful to replace the actual actions with such abstract explanations of their functions. This abstract sequence seems easier to understand, and it also allows using recursive macros to compress it, which might further enhance its explaining power. However, in this particular case, it would probably be even more useful if the planner could present the whole sequence as a for loop, or similar.

This essentially boils down to partitioning the set of actions into equivalence classes such that each such class consist of actions that can be meaningfully seen as implementing the same concept. It seems both interesting and important to investigate how and when one can partition the set of actions into equivalence classes useful for such abstractions.

In the context of reformulation, we briefly discussed the possibility whether the solution for the reformulated problem may occasionally be possible to use as a solution for the original problem. While that seems an exception, at best, for planning it might perhaps be a more fruitful approach for plan explanation in some cases.

6.3 Variants of the Results

Liberatore (2005) has also studied the problem of representing plans compactly, and has presented results similar to our theorems in Section 4. However, he assumes a very powerful circuit-based planning formalism, so it is not obvious to what extent his results carry over to less expressive formalisms. Our theorems thus improve upon his results, showing that such representations are unlikely to exist even for simple languages like propositional STRIPS.

The concept of unary actions, that is, actions that have only one atom each in their postconditions, has been studied in the literature. While it may seem to be a very limiting restriction, it has been shown that restricted cases with unary actions are sufficient for practical use in on-board controllers for spacecrafts (Brafman and Domshlak 2003). For the general case, Bylander (1994) has shown that planning remains PSPACE-complete under this restriction. While only implicit in his proofs, it follows that all planning instances must be possible to transform into equivalent instances having only unary actions. Bäckström (1992, Proof of Theorem 6.6) demonstrates a constructive such transformation, replacing each non-unary action with several actions, that are defined so they must always be executed together (thus resembling a macro). All theorems in this paper still hold if making such a transformation on the planning instances used, but we sometimes have to add dummy atoms to make all actions have the same number of postconditions. It seems reasonable that our proofs could be analogously modified to hold also for many other restriction of STRIPS.

Despite the theoretical limits we prove, there are both theoretical and practical cases in the literature where abstraction, recursive macros and reformulation have proven successful. Why and when such methods work seems an important and interesting question for further study.

References

Bäckström, C., and Jonsson, P. 1995. Planning with abstraction hierarchies can be exponentially less efficient. In *Proc. 14th Int'l Joint Conf. Artif. Intell. (IJCAI'95), Montreal, Canada*, 1599–1605.

Bäckström, C. 1992. *Computational Complexity of Reasoning about Plans*. PhD diss., Linköping University, Linköping, Sweden.

Bäckström, C. 1995. Expressive equivalence of planning formalisms. *Artif. Intell.* 76(1-2):17–34.

Bidot, J.; Biundo, S.; Heinroth, T.; Minker, W.; Nothdurft, F.; and Schattenberg, B. 2010. Verbal plan explanations for hybrid planning. In *24th MKWI related PuK-workshop: Planung/Scheduling und Konfigurieren/Entwurfen (PuK'10)*, 2309–2320.

Brafman, R. I., and 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.

Chen, H., and Giménez, O. 2010. Causal graphs and structurally restricted planning. *J. Comput. Syst. Sci.* 76(7):579–592.

Edelkamp, S.; Leue, S.; and Visser, W. 2007. Summary of Dagstuhl seminar 06172 on directed model checking. In *Directed Model Checking*, number 06172 in Dagstuhl Seminar Proceedings. Dagstuhl, Germany.

Giménez, O., and Jonsson, A. 2008. The complexity of planning problems with simple causal graphs. *J. Artif. Intell. Res.* 31:319–351.

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

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

Karp, R. M., and Lipton, R. J. 1980. Some connections between nonuniform and uniform complexity classes. In *Proc. 12th ACM Symp. Theory Comput. (STOC'80), Los Angeles, CA, USA*, 302–309.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *Proc. 10th European Conf. Artif. Intell. (ECAI'92), Vienna, Austria*, 359–363.

Knoblock, C. A. 1993. *Generating Abstraction Hierarchies: An Automated Approach to Reducing Search in Planning*. Norwell, MA: Kluwer Academic Publishers.

Liberatore, P. 2005. Complexity issues in finding succinct solutions of PSPACE-complete problems. *ArXiv publication* abs/cs/0503043.

Long, D.; Fox, M.; and Hamdi, M. 2002. Reformulation in planning. In *5th Int'l Symp. Abstraction, Reformulation and Approximation (SARA'02), Kananaskis, AB, Canada*, volume 2371 of *LNCS*, 18–32. Springer.

Southwick, R. W. 1991. Explaining reasoning: an overview of explanation in knowledge-based systems. *Knowledge Eng. Rev.* 6:1–19.

Yap, C.-K. 1983. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.* 26:287–300.