Towards Efficient Universal Planning—A Randomized Approach

Peter Jonsson, Patrik Haslum and Christer Bäckström
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
{petej,pahas,cba}@ida.liu.se

31st August 2000

Abstract

One of the most widespread approaches to reactive planning is Schoppers' universal plans. We propose a stricter definition of universal plans which guarantees a weak notion of soundness, not present in the original definition, and isolate three different types of completeness that capture different behaviors exhibited by universal plans. We show that universal plans which run in polynomial time and are of polynomial size cannot satisfy even the weakest type of completeness unless the polynomial hierarchy collapses. By relaxing either the polynomial time or the polynomial space requirement, the construction of universal plans satisfying the strongest type of completeness becomes trivial. As an alternative approach, we study *randomized* universal planning. By considering a randomized version of completeness and a restricted (but nontrivial) class of problems, we show that there exists randomized universal plans running in polynomial time and using polynomial space which are sound and complete for the restricted class of problems. We also report experimental results on this approach to planning, showing that the performance of a randomized planner is not easily compared to that of a deterministic planner.

1 Introduction

In recent years reactive planning has been proposed as an alternative to classical planning, especially in rapidly changing, dynamic domains. Although this term has been used for a number of more or less related approaches, these have one thing in common: There is usually very little or no planning ahead. Rather the idea is centered around the stimulus-response principle — prompt reaction to the input. One of the most well-known methods for reactive planning is the *universal plans* by Schoppers [1987]. A universal plan is a function from the set of states into the set of operators. Hence, a universal plan does not generate a sequence of operators leading from the current state to the goal state as a classical planner; it decides after each step what to do next based on the current state.

Universal plans have been much discussed in the literature. In a famous debate [Ginsberg, 1989b, Schoppers, 1989, Ginsberg, 1989a, Schoppers, 1994], Ginsberg criticized the approach while Schoppers defended it[1]. Based on a counting argument, Ginsberg claims that almost all (interesting) universal plans takes an infeasibly large amount of space. Schopper's defense has, to a large extent, built on the observation that planning problems are structured. According to Schoppers, this structure can be exploited in order to create small, effective universal plans. We refrain from going into the details of this debate and merely note that both authors have shown great ingenuity in their argumentation. However, from the standpoint of formal rigour, these papers do not settle the question. One of the few authors that has treated universal plans from a formal, complexity-theoretic point of view is Selman [1994]. He shows that the existence of small (polynomially-sized) universal plans with the ability to generate *minimal* plans implies a collapse of the polynomial hierarchy. Since a collapse of the polynomial hierarchy is widely conjectured to be false in the literature [Johnson, 1990, Papadimitriou, 1994], the existence of such universal plans seems highly unlikely.

In our opinion, one of the problems with universal plans is the generality of the definition, which makes formal analysis hard or even impossible. Therefore, we begin this article by giving a stricter definition of universal plans, a definition that embodies the notion of *soundness*. In addition, we

[1]This list is not exhaustive. Other authors, such as Chapman [1989], have joined the discussion. However, it seems that the main combatants have been Schoppers and Ginsberg.

supply three different criteria of *completeness*. These notions of completeness capture different desirable properties of universal plans. For example, A-completeness states that if the problem has a solution, then the universal plan will find a solution in a finite number of steps. Our first result says that universal plans which run in polynomial time and are of polynomial size cannot satisfy even this weakest type of completeness[2]. However, by relaxing either the polynomial time requirement or the polynomial space requirement, it becomes trivial to construct universal plans that satisfy the strongest type of completeness. Also in this case, the result holds for severely restricted problems.

As an alternative, we propose to give the universal plans access to a random source, making universal planning probabilistic. This forces us to redefine completeness in a way that takes the randomization into account. Even after these changes to the universal planning paradigm, it is impossible to provide efficient universal plans for the general planning problem, but for a certain subclass of problems we show that there exists sound and complete randomized universal plans running in polynomial time and using polynomial space. It should be noted that this class is not trivial; the planning problem is PSPACE-complete, *i.e.*, as hard as the unrestricted problem.

We have implemented such a randomized planner (which we call STOCPLAN) and compared it to a deterministic planner (GRAPHPLAN) on a number of domains. The experimental results are inconclusive; no planner is consistently faster than the other, and no single domain characteristic can reliably predict STOCPLAN's performance. However, we present a plausible hypothesis.

The article is organized as follows: We begin by defining the basic STRIPS formalism in Section 2. We investigate deterministic universal planning in Section 3 and randomized universal planning in Sections 4 and 5, theoretically and empirically. The article is concluded with a brief discussion of the results. Section 3 is a revised version of the conference paper [Jonsson and Bäckström, 1996].

2 Basic Formalism

We base our work in this article on the propositional STRIPS formalism with negative goals (PSN, for short [Bylander, 1994]), which is equivalent to most

[2]Under the assumption that the polynomial hierarchy does not collapse.

other variants of propositional STRIPS [Bäckström, 1995].

Definition 1 An instance of the *PSN planning problem* is a quadruple $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ where

- $\mathcal{P}$ is a finite set of *atoms*;
- $\mathcal{O}$ is a finite set of *operators* where $o \in \mathcal{O}$ has the form $Pre \Rightarrow Post$ where
  - Pre is a satisfiable conjunction of positive and negative atoms in $\mathcal{P}$, respectively called the *positive preconditions* ($pre^+(o)$) and the *negative preconditions* ($pre^-(o)$);
  - $Post$ is a satisfiable conjunction of positive and negative atoms in $\mathcal{P}$, respectively called the *positive postconditions* ($add(o)$) and the *negative postconditions* ($del(o)$);
- $\mathcal{I} \subseteq \mathcal{P}$ denotes the *initial state*;
- and $\mathcal{G} = \langle \mathcal{G}^+, \mathcal{G}^- \rangle$ denote the *positive* and *negative goal*, respectively, satisfying $\mathcal{G}^+, \mathcal{G}^- \subseteq \mathcal{P}$ and $\mathcal{G}^+ \cap \mathcal{G}^- = \varnothing$.

A *PSN structure* is a tuple $\Phi = \langle \mathcal{P}, \mathcal{O} \rangle$ where $\mathcal{P}$ is a set of atoms and $\mathcal{O}$ is a set of operators over $\mathcal{P}$. We denote the negation of an atom by overlining it. As an example, the operator o defined as $\overline{p} \Rightarrow q, \overline{r}$ satisfies $pre^+(o) = \varnothing$, $pre^-(o) = \{p\}$, $add(o) = \{q\}$ and $del(o) = \{r\}$.

Definition 2 For a given PSN structure $\Phi = \langle \mathcal{P}, \mathcal{O} \rangle$, let the set of states $\mathcal{S} = 2^{\mathcal{P}}$ and the extended set of states $\mathcal{S}_\perp = 2^{\mathcal{P}} \cup \{\perp\}$; $\perp$ is used to represent undefinedness. The *update* operator $\oplus : \mathcal{S}_\perp \times \mathcal{O} \to \mathcal{S}_\perp$ is defined as

$$\begin{aligned} \perp \oplus o &= \perp \\ S \oplus o &= \begin{cases} (S - del(o)) \cup add(o) & \text{if } pre^+(o) \subseteq S \wedge pre^-(o) \cap S = \varnothing \\ \perp & \text{otherwise} \end{cases} \end{aligned}$$

for each operator $o \in \mathcal{O}$ and state $S \in \mathcal{S}_\perp$. We say that an operator o is *admissible* in a state $S \in \mathcal{S}_\perp$ iff $S \oplus o \neq \perp$.

Given a set of operators $\mathcal{O}$, let $Seqs(\mathcal{O})$ denote the set of all operator sequences over $\mathcal{O}$. A sequence $\langle o_1, \ldots, o_n \rangle \in Seqs(\mathcal{O})$ of operators is called a *PSN plan* (or simply plan) over Φ. A sequence is admissible in a state S iff

$$((\ldots(S \oplus o_1)\ldots) \oplus o_n) \neq \perp$$

and we say that a plan *solves* the problem instance Π iff it is admissible in the initial state, $\mathcal{I}$, and

$$\mathcal{G}^+ \subseteq ((\ldots(\mathcal{I} \oplus o_1)\ldots) \oplus o_n)$$
$$((\ldots(\mathcal{I} \oplus o_1)\ldots) \oplus o_n) \cap \mathcal{G}^- = \varnothing$$

both hold.

Finally, we define the computational problems under consideration.

Definition 3 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \langle \mathcal{G}^+, \mathcal{G}^- \rangle \rangle$ be a given PSN instance. The *plan generation problem* (PG) is to find some $\omega \in \mathit{Seqs}(\mathcal{O})$ s.t. ω is a solution to Π or answer that no such ω exists. The *bounded plan generation problem* (BPG) takes an integer $K \geq 0$ as additional parameter and the object is to find some $\omega \in \mathit{Seqs}(\mathcal{O})$ s.t. ω is a solution to Π of length $\leq K$ or answer that no such ω exists.

3 Universal Plans

The material on deterministic universal plans is collected in this section. Subsection 3.1 contains the basics of universal planning and Subsection 3.2 identifies different types completeness for universal plans. The existence of universal plans being complete in a very strong sense is discussed in Subsection 3.3 and such universal plans are shown to be infeasible in Subsection 3.4.

3.1 Preliminaries

Universal plans are defined as follows by Ginsberg [1989b].

> A universal plan is an arbitrary function from the set of possible situations S into the set of primitive actions A.

Using the terminology we have adopted in this article results in the following equivalent definition.

Definition 4 Given a PSN structure $\Phi = \langle \mathcal{P}, \mathcal{O} \rangle$, a universal plan is a function from the set of states $2^{\mathcal{P}}$ into the set of operators $\mathcal{O}$.

This very general notion of universal plans is difficult to use as a basis for formal analysis. We would like, for example, to discuss the issues of correctness and resource consumption. The next definition captures the notion of soundness for a universal plan.

For a given PSN structure $\Phi = \langle \mathcal{P}, \mathcal{O} \rangle$, let $\mathcal{O}^+ = \mathcal{O} \cup \{o_\perp, o_\top\}$ and extend the update operator so that

$$\begin{aligned} S \oplus o_\perp &= \perp \\ S \oplus o_\top &= S \end{aligned}$$

for every state $S \in \mathcal{S}_\perp$. The "special" operators $o_\perp$ and $o_\top$ should not be considered as operators in the sense of Definition 1 but rather as two completely new symbols without internal structure. They will be used by the universal plans for "communication with the environment".

Definition 5 Let $\Phi = \langle \mathcal{P}, \mathcal{O} \rangle$ be a PSN structure and let $\mathcal{G}$ be a goal over $\mathcal{P}$. A *sound universal plan* $U_\mathcal{G}$ for the goal $\mathcal{G}$ is a function mapping $\mathcal{S}_\perp$ to $\mathcal{O}^+$ such that

1. for every $S \in \mathcal{S}_\perp$, if $U_\mathcal{G}(S) = o \in \mathcal{O}$ then o is admissible in S;
2. for every $S \in \mathcal{S}_\perp$, $U_\mathcal{G}(S) = o_\top$ iff S satisfies $\mathcal{G}$;

The first point of the definition says that if the universal plan generates an operator, then this operator is executable in the current state. This restriction seems to have been tacitly assumed in the literature. The second point tells us that the special operator $o_\top$ is generated if and only if the universal plan is applied to a state satisfying the goal state. Thus, $o_\top$ is used by $U_\mathcal{G}$ to report success. The reason for introducing the operator $o_\top$ is to avoid the generation of new operators when the current state satisfies the goal state. The special operator $o_\perp$, on the other hand, indicates that the universal plan cannot handle the current state. This can, for instance, be due to the fact that the goal state is not reachable from the current state. Note that no operator is admissible in $\perp$ so $U_\mathcal{G}$ must generate $o_\perp$ whenever applied to $\perp$. Henceforth, we will use the term universal plan as an abbreviation for sound universal plan.

3.2 Properties of Universal Plans

We continue by defining some properties of universal plans. For a universal plan $U_\mathcal{G}$ we use the notation $U_\mathcal{G}^K(S)$ to denote $U_\mathcal{G}(S_K)$ where S_K is inductively defined:

$$S_K = \begin{cases} S & \text{if } K = 0 \\ S_{K-1} \oplus U_{\mathcal{G}}(S_{K-1}) & \text{otherwise.} \end{cases}$$

Definition 6 Let $\mathcal{X}$ be a set of PSN structures. We say that $\mathcal{X}$ admits

- *acceptance-complete* universal plans (A) iff for every $\Phi \in \mathcal{X}$ and goal $\mathcal{G}$ over Φ, there exists a universal plan $U_{\mathcal{G}}$ such that for every $S \in \mathcal{S}$, if $\langle \mathcal{P}, \mathcal{O}, S, \mathcal{G} \rangle$ is solvable, then there exists an integer K such that $U_{\mathcal{G}}^K(S) = o_\top$;

- *rejection-complete* universal plans (R) iff for every $\Phi \in \mathcal{X}$ and goal $\mathcal{G}$ over Φ, there exists a universal plan $U_{\mathcal{G}}$ such that for every $S \in \mathcal{S}$, if $\langle \mathcal{P}, \mathcal{O}, S, \mathcal{G} \rangle$ is not solvable, then there exists an integer K such that $U_{\mathcal{G}}^K(S) = o_\perp$;

- *poly-time* universal plans (P_T) iff there exists a polynomial p such that for every $\Phi \in X$ and every goal $\mathcal{G}$ over Φ, there exists a universal plan $U_{\mathcal{G}}$ with running time bounded by $p(|\Phi|)$.

- *poly-space* universal plans (P_S) iff there exists a polynomial q such that for every $\Phi \in X$ and every goal $\mathcal{G}$ over Φ, there exists a universal plan $U_{\mathcal{G}}$ such that the size of $U_{\mathcal{G}}$ *and* the size of the auxiliary memory used by $U_{\mathcal{G}}$ is bounded by $q(|\Phi|)$.

For the sake of brevity, we will use the terms A- and R-completeness for acceptance- and rejection-completeness, respectively. By saying that $\mathcal{X}$ admits, for example, A-complete *and* poly-time universal plans, we mean that there exists a polynomial p such that for each $\Phi \in \mathcal{X}$ and goal state $\mathcal{G}$ over Φ, there exists a acceptance-complete universal plan running in time bounded by $p(|\Phi|)$.

It makes sense to say that a single universal plan $U_{\mathcal{G}}$ is A-complete or R-complete with respect to a PSN structure Φ. However, it does not make sense to say that it is poly-time or poly-space for obvious reasons; the input to $U_{\mathcal{G}}$ is of fixed size so it is trivially poly-time and poly-space.

A minimal requirement on universal plans is that they are A-complete so we are guaranteed to find a solution within a finite number of steps if there is one. Note that if an A-complete universal plan is not R-complete then $U_{\mathcal{G}}^K(S)$ can differ from $o_\perp$ for all K if $\mathcal{G}$ is not reachable from S. R-completeness is, thus, desirable but not always necessary. In domains such as blocksworld, where we know that a solution exists in advance, R-completeness is of minor interest. To have R-completeness without A-completeness is useless since we can trivially construct universal plans that are R-complete for all problems; simply let $U_{\mathcal{G}}(S) = o_\perp$ for all $S \in \mathcal{S}_\perp$.

The definition of poly-time universal plans should be quite clear while the definition of the poly-space restriction may need further explanation. The first part of the definition ensures that $U_{\mathcal{G}}$ can be stored in a polynomially-bounded memory. The second part guarantees that any computation will use only a polynomially-bounded amount of auxiliary memory. Hence, we can both store and run the algorithm in a memory whose size is bounded by a polynomial in the size of Φ. This restriction excludes algorithms using extremely large fixed data structures as well as algorithms building such structures during run-time.

In certain cases, a stronger form of R-completeness may be needed.

Definition 7 Let $\mathcal{X}$ be a set of PSN structures. We say that $\mathcal{X}$ admits *strongly rejection-complete* universal plans (R^+) iff for every $\Phi \in \mathcal{X}$ and goal $\mathcal{G}$ over Φ, there exists a universal plan $U_{\mathcal{G}}$ such that for every $S \in \mathcal{S}$ such that $\langle \mathcal{P}, \mathcal{O}, S, \mathcal{G} \rangle$ is not solvable, $U_{\mathcal{G}}(S) = o_\perp$;

The motivation for introducing strong R-completeness is simple. If the universal plan outputs operators, we cannot know whether they will lead to a solution or not. Executing such operators is not advisable, since we may wish to try planning for some alternative goal if there is no solution for the first one. However, executing the "invalid" operators may prevent us from reaching the alternative goal.

3.3 Existence of $\mathbf{P}_T\mathbf{AR}^+$ and $P_S\mathbf{AR}^+$ universal plans

The next theorem shows that the class of all PSN structures admits AR^+-complete universal plans which are either poly-time or poly-space. In the next section, we will show that this class does not admit even A-complete universal plans which are simultaneously poly-time and poly-space.

Theorem 8 The class of all PSN structures admits $P_T AR^+$-universal plans and $P_S AR^+$-universal plans.

Proof: Arbitrarily choose a PSN structure $\Phi = \langle \mathcal{P}, \mathcal{O} \rangle$ and goal state $\mathcal{G}$ over $\mathcal{P}$. First, we show that there exists an AR^+-universal plan $U_{\mathcal{G}}$ whose running time is bounded by some fixed polynomial in $|\mathcal{P}|$. Then, we show that there exists a poly-space AR$^+$-universal plan $U'_\mathcal{G}$ which is of constant size and uses $O(p(|\langle\mathcal{P},\mathcal{O}\rangle|)^2)$ auxiliary space for some fixed polynomial p.

Construction of $U_\mathcal{G}$: We define a function $f : \mathcal{S}_\perp \to \mathcal{O}^+$ as follows. For each $K \geq 1$ and $S \in \mathcal{S}$ such that $\langle\mathcal{P},\mathcal{O},S,\mathcal{G}\rangle$ has a shortest solution of length K, choose an $o \in \mathcal{O}$ such that $\langle\mathcal{P},\mathcal{O},S\oplus o,\mathcal{G}\rangle$ has a shortest solution of length $K-1$. Denote this operator o_S and let

$$f(S) = \begin{cases} o_\perp & \text{if } \langle\mathcal{P},\mathcal{O},S,\mathcal{G}\rangle \text{ is not solvable} \\ o_\top & S \text{ satisfies } \mathcal{G} \\ o_S & \text{otherwise} \end{cases}$$

Clearly, for every $S \in \mathcal{S}$ there exists an integer K such that if $\langle\mathcal{P},\mathcal{O},S,\mathcal{G}\rangle$ is solvable then $U^K_\mathcal{G}(S) = o_\top$. Otherwise, $U_\mathcal{G}(S) = o_\perp$. Consequently, f is both A-complete and strongly R-complete. The proposed construction of the function f is obviously of exponential size. However, it can be arranged as a balanced decision tree of depth $|\mathcal{P}|$ and be accessed in polynomial time. Consequently, we have constructed $U_\mathcal{G}$.

Construction of $U'_\mathcal{G}$: Consider a forward-chaining PSN planning algorithm P that is sound, complete and generates shortest plans. We modify the algorithm to output only the first operator of the plan that leads from S to $\mathcal{G}$. Since a plan might be of exponential size this cannot necessarily be implemented in polynomial space. However, we can guess the plan one operator at a time and compute the resulting state after each action, using only polynomial space. Hence, this modified planner can be represented by a nondeterministic algorithm using $O(p(|\langle\mathcal{P},\mathcal{O}|))$ space for some fixed polynomial p. Thus, by Savitch's [1970] theorem, it can also be represented by a deterministic algorithm that uses $O(p(|\langle\mathcal{P},\mathcal{O}|)^2)$ space. This modified planner can be the same for all problems simply by giving the PSN structure Φ and the goal state $\mathcal{G}$ as additional inputs. Hence, it is of constant size, *i.e.*, its size does not depend on the size of the given PSN structure. Consequently, we can disregard the size of the planner and we have constructed a poly-space universal plan. (Observe that the soundness of P implies soundness of $U'_\mathcal{G}$ if we modify $U'_\mathcal{G}$ to generate $o_\top$ whenever the current state satisfies the goal state.)

The planner P is complete and generates minimal plans. Hence, if the shortest plan from the current state S to the goal state $\mathcal{G}$ is of length L, the length of the shortest plan from $S \oplus U'_\mathcal{G}(S)$ to $\mathcal{G}$ is $L-1$. By this observation and the fact that P is complete, A-completeness of $U'_\mathcal{G}$ follows.

Finally, if there is no plan from the current state to the goal state, the planner will fail to generate even the first operator. In this case we simply output $o_\perp$ and strong R-completeness follows. $\Box$

Constructions similar to those used in the previous proof have been presented by Baral and Tran [1998]; our research were done independently around the same time.

It is crucial that the planner used in the previous theorem generates shortest plans, otherwise, we cannot guarantee A-completeness. We illustrate this with a small, contrived example.

Example 9 Consider the PSN structure $\Phi = \langle \mathcal{P}, \mathcal{O} \rangle = \langle \{p, q\}, \{p^+, q^+, q^-\} \rangle$ where the operators are defined as $p^+ = (\overline{p} \Rightarrow p)$, $q^- = (q \Rightarrow \overline{q})$ and $q^+ = (\overline{q} \Rightarrow q)$.

Let $\mathcal{I}_1 = \{q\}$, $\mathcal{I}_2 = \varnothing$, $\mathcal{G} = \langle \{p\}, \varnothing \rangle$, $\Pi_1 = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}_1, \mathcal{G} \rangle$ and $\Pi_2 = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}_2, \mathcal{G} \rangle$. The shortest plan for both Π_1 and Π_2 is $\langle p^+ \rangle$. Assume a planning algorithm A that generates the plan $\omega_1 = \langle q^-, p^+ \rangle$ for Π_1 and $\omega_2 = \langle q^+, p^+ \rangle$ for Π_2. A universal plan $U_\mathcal{G}$ based on A would then satisfy $U_\mathcal{G}(\mathcal{I}_1) = q^+$ and $U_\mathcal{G}(\mathcal{I}_2) = q^-$. Consequently, $U_\mathcal{G}^K(\mathcal{I}_1) = q^+$ for odd K and $U_\mathcal{G}(\mathcal{I}_1) = q^-$ for even K. In other words, the universal plan will toggle q forever. Hence, $U_\mathcal{G}$ is not A-complete.

If $\mathcal{S}$ is a set of planning problems such that BPG[3] can be solved in polynomial-time, $\mathcal{S}$ admits $\mathrm{P}_{T,S}\mathrm{AR}^+$ universal plans, by Theorem 8. For planning problems such that PG is polynomial but BPG is not, the theorem does not apply. This method for constructing universal plans is pointed out by Selman [1994] but he does not explicitly state that generating the shortest plan is necessary. The question whether classes of planning problems where PG is polynomial but BPG is not admits $\mathrm{P}_{T,S}\mathrm{AR}^+$ universal plans remains open.

3.4 Non-existence of $\mathrm{P}_{T,S}\mathrm{A}$ universal plans

To show that the class of all PSN structures does not admit A-complete universal plans that are both poly-time and poly-space, we will use *advice-taking* Turing machines [Johnson, 1990].

[3] Recall that BPG and PG denote the bounded and unbounded plan generation problem respectively.

Definition 10 An *advice-taking* Turing machine is a TM T that has associated with it a special "advice oracle" A, which is a (not necessarily computable) function. Let x be an arbitrary input string and let $|x|$ denote the size of x. When T is applied to x, a special "advice tape" is automatically loaded with $A(|x|)$ and from then on the computation proceeds as normal, based on the two inputs, x and $A(|x|)$. An advice-taking Turing machine uses *polynomial advice* iff its advice oracle satisfies $|A(n)| \leq p(n)$ for some fixed polynomial p and all nonnegative integers n. The class P/poly is the set of languages defined by polynomial-time advice-taking TMs with polynomial advice.

Advice-taking TMs are very powerful. They can, for instance, compute certain undecidable functions. Despite their apparent power, it is highly unlikely that all problems in NP can be solved by polynomial-time TMs using polynomial advice.

Theorem 11 If NP $\subseteq$ P/poly, then the polynomial hierarchy collapses into Σ_2^p [Karp and Lipton, 1982]. Furthermore, the polynomial hierarchy collapses into the complexity class $\mathrm{ZPP}^{\mathrm{NP}}$ [Köbler and Watanabe, 1999].

$\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$ is a complexity class in the second level of the polynomial hierarchy [Johnson, 1990], $\mathrm{ZPP}^{\mathrm{NP}} \subseteq \Sigma_2^p$ and this inclusion is believed to be strict. A collapse of the polynomial hierarchy is widely conjectured to be false in the literature [Johnson, 1990, Papadimitriou, 1994]. Our proofs rely on the following construction.

Lemma 12 Let $\mathcal{F}_n$ be the set of all 3SAT [Garey and Johnson, 1979] instances with n variables. For every n, there is a PSN structure $\Theta_n = \langle \mathcal{P}, \mathcal{O} \rangle$ and a goal state $\mathcal{G}_n$ such that for every $F \in \mathcal{F}_n$, there exists an $\mathcal{I}_F$ with the following property: $\Pi_F = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}_F, \mathcal{G}_n \rangle$ is a planning instance which is solvable iff F is satisfiable. Furthermore, any solution to Π_F must have a length less than or equal to $8n^3 + 2n$.

Proof: Let $U = \{u_1, \ldots, u_n\}$ be the set of variables used by the formulae in $\mathcal{F}_n$. Observe that there can only be $(2n)^3$ different clauses in any formula in $\mathcal{F}_n$. Let $\mathcal{C} = \{C_1, \ldots, C_{8n^3}\}$ be an enumeration of the possible clauses over the variable set U. Let $\mathcal{P} = \{T(i), F(i), C(j) | 1 \leq i \leq n,\ 1 \leq j \leq 8n^3\}$. The atoms will have the following meanings: $T(i)$ is true iff the variable u_i is true, $F(i)$ is true iff the variable u_i is false and $C(j)$ is true iff the clause C_j is satisfied. For each variable u_i, two operators are needed:

- $\overline{T(i)}, \overline{F(i)} \Rightarrow T(i)$,
- $\overline{T(i)}, \overline{F(i)} \Rightarrow F(i)$.

That is, $T(i)$ can be made true iff $F(i)$ is false and vice versa. In this fashion, only one of $T(i)$ and $F(i)$ can be true. For each case where a clause $C(j) \in \mathcal{C}$ contains a variable u_i, the first operator below is needed: for a negated variable $\neg u_i$, the second operator is needed:

- $T(i), \overline{C(j)} \Rightarrow C(j)$,
- $F(i), \overline{C(j)} \Rightarrow C(j)$.

We specify the goal such that $\mathcal{G}_n = \langle \mathcal{G}_n^+, \mathcal{G}_n^- \rangle = \langle \{C_1, \ldots, C_{8n^3}\}, \varnothing \rangle$. Let $F \in \mathcal{F}$. We want to construct an initial state $\mathcal{I}_F$ such that $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}_F, \mathcal{G}_n \rangle$ is solvable iff F is satisfiable. Let $\mathcal{I}_F = \{C(j) | C(j) \notin F\}$. Clearly, every $C(j)$ can be made true iff a satisfying assignment for F can be found. Finally, it is easy to see that any solution to Π_F must be of length $\leq 8n^3 + 2n$ since we have exactly $8n^3 + 2n$ atoms and each atom can be made true at most once. □

Lemma 13 If, for every integer $n \geq 1$, there exists a polynomial advice function that allows us to solve Π_F for all $F \in \mathcal{F}_n$ in polynomial time, then the polynomial hierarchy collapses.

Proof: By Lemma 12, Π_F is solvable iff F has a satisfying truth assignment. If there exists a polynomial advice function that allows us to solve Π_F for all $F \in \mathcal{F}_n$ in polynomial time, then NP $\subseteq$ P/poly so, by Theorem 11, the polynomial hierarchy collapses into Σ_2^p and $\mathrm{ZPP}^{\mathrm{NP}}$. □

We can now prove our non-existence theorem.

Theorem 14 The class of all PSN structures does not admit universal plans which are A-complete, poly-time and poly-space at the same time.

Proof: We show that if there exists a poly-time and poly-space A-universal plan $U_{\mathcal{G}_n}$ for Θ_n, $n \geq 1$, then the polynomial hierarchy collapses.

Assume $U_{\mathcal{G}_n}$ to be such a universal plan for Θ_n, and consider the algorithm A in Figure 1. $U_{\mathcal{G}_n}$ is sound so it must generate an operator that is admissible in the given state or generate one of the special operators $o_\perp, o_\top$. Hence, by Lemma 12, the **repeat** loop can iterate at most $8n^3 + 2n$ times before o equals either $o_\perp$ or $o_\top$. We have assumed that $U_{\mathcal{G}_n}$ is a polynomial-time

Algorithm A.
Input: A 3SAT formula F with n variables.
$S \leftarrow \mathcal{I}_F$
repeat
    $o \leftarrow U_{\mathcal{G}_n}(S)$
    $S \leftarrow S \oplus o$
until $o \in \{o_\perp, o_\top\}$
if $o = o_\top$ **then accept**
else reject

Figure 1: The algorithm used in the proof of Theorem 14.

algorithm so algorithm A runs in polynomial time. We show that algorithm A accepts iff F has a satisfying truth assignment. The if-part is trivial by noting that if F has a satisfying truth assignment, then the algorithm accepts by A-completeness. For the only-if part, assume that the algorithm accepts. Then $U_{\mathcal{G}_n}$ has returned the operator $o_\top$ when applied to some state S. By Definition 5, $U_{\mathcal{G}_n}(S) = o_\top$ iff S satisfies $\mathcal{G}_n$, and consequently, F is satisfiable by Lemma 12. Hence, the algorithm accepts iff F is satisfiable and rejects iff F is not satisfiable. Furthermore, $U_{\mathcal{G}_n}$ is a polynomial advice function since we have restricted $U_{\mathcal{G}_n}$ to be of polynomial size and the theorem follows by Lemma 13. □

The generality of this theorem has to be emphasized. Recall that an advice is an *arbitrary* function from the size of the input; it does not even have to be computable. Hence, there does not exist any mechanism whatsoever that is of polynomial size and uses only polynomial time which has the ability to solve problems like those exhibited in the previous theorem. Methods that have been proposed to reduce the size of universal plans, such as the *variables* introduced by Schoppers [1994], cannot change this fact.

Moreover, observe that Theorem 14 applies even to a class of severely restricted PSN structures. The restrictions are, among others, that there are no negative postconditions and each operator has at most two preconditions. Since there are no negative postconditions, this restricted class is in NP [Bylander, 1994]. Consequently, it is a class with considerably less expressive power than the general PSN planning problem which is PSPACE-complete (under the plausible assumption that NP $\neq$ PSPACE). Yet, poly-time and poly-space A-universal plans do not exist for this class of planning problems. Note that this is not caused by the existence of exponential-size minimal plans since all minimal plans in this class are polynomially bounded.

Finally, we would like to compare Theorem 14 with Selman's [1994] negative result. He shows that the class of all PSN structures does not admits $\mathrm{P}_{T,S}\mathrm{A}$ universal plans which generate the shortest possible solution where our results shows that this class does not even admit $\mathrm{P}_{T,S}\mathrm{A}$ universal plans generating *any* solution.

4 Randomized Universal Planning: Theory

To overcome the negative results in the previous section, one can basically do three things:

1. Give the universal plan access to more or other computational resources,
2. use some other notion of completeness, or
3. only consider a restricted class of problems.

We will combine all these ideas in this section. By giving the universal plans access to a random source (which forces us to modify our notion of completeness) and concentrating on PSN structures having a certain symmetry property, we show that there exists a large, non-trivial class of planning problems which admits efficient randomized universal plans.

This section is organized as follows: Subsection 4.1 introduces the concepts of randomized completeness and symmetric PSN structures while Subsection 4.2 settles the existence of of randomized universal plans under certain restrictions. Complexity aspects of the problem of deciding symmetry are discussed in Subsection 4.3, and of the planning problem in the symmetric case in Subsection 4.4.

4.1 Randomized completeness and symmetric PSN structures

We assume that the random source is being accessed by *coin tosses*, that is, the universal plan can at any time during its execution toss an unbiased coin and receive a random bit. To take full advantage of the introduction of a random source, a new concept of completeness is needed. Thus, we make the following definition.

Definition 15 A universal plan $U_{\mathcal{G}}$ for a PSN structure $\Phi = \langle \mathcal{P}, \mathcal{O} \rangle$ is *complete in the randomized sense with parameter p* (C_p), $0 \leq p \leq 1$, iff for every $S \in \mathcal{S}$ there exists an integer K such that $U^K_{\mathcal{G}}(S) = o_\top$ or $U^K_{\mathcal{G}}(S) = o_\perp$, and the following holds:

- if $U^K_{\mathcal{G}}(S) = o_\top$ then $\langle \mathcal{P}, \mathcal{O}, S, \mathcal{G} \rangle$ has a solution;
- if $U^K_{\mathcal{G}}(S) = o_\perp$, then $\langle \mathcal{P}, \mathcal{O}, S, \mathcal{G} \rangle$ has no solution with probability $\geq p$.

The probability is taken over all possible coin tosses made by $U_{\mathcal{G}}$. Let $\mathcal{X}$ be a set of PSN structures and assume that there exists a C_p-complete universal plan for each member of $\mathcal{X}$. In this case, we say that $\mathcal{X}$ *admits* C_p-universal plans.

Comparing the notion of C_p-completeness with A- and R-completeness, we see that if $U^K_{\mathcal{G}}(S) = o_\top$, then we know for sure that there exists a plan. This is of course inevitable since we insist that $U_{\mathcal{G}}$ has to be sound. If $U^K_{\mathcal{G}}(S) = o_\perp$, then there is no plan with probability $\geq p$. However, there is a positive probability that there is a plan, albeit a small one, $\leq 1 - p$. Thus, the answer $o_\perp$ does not completely rule out the existence of a plan, it merely tells us that the existence of a plan is highly unlikely.

We continue by defining the class of *symmetric* PSN structures.

Definition 16 A PSN structure $\Phi = \langle \mathcal{P}, \mathcal{O} \rangle$ is *symmetric* iff there for every $S \in \mathcal{S}$ and every operator o which is admissible in S exists an operator o' such that $((S \oplus o) \oplus o') = S$. Let SYM denote the set of symmetric PSN structures.

Symmetric planning problems have previously been considered by several authors, *e.g.* Williams and Nayak [1997] and Jonsson and Bäckström [1998]. Several standard examples such as blocksworld [Sacerdoti, 1975] and Towers-of-Hanoi [Green, 1969] are also symmetric under suitable encodings.

The complexity of deciding whether a given PSN structure is symmetric or not is investigated in Subsection 4.3. The main result is the following:

Theorem 17 The problem of deciding whether a PSN structure is symmetric or not is coNP-complete.

Even though the class of symmetric PSN structures may seem severely restricted, it is by no means trivial. In fact, deciding the plan existence problem in symmetric PSN structures is as hard as for arbitrary PSN instances, as shown in Subsection 4.4:

Theorem 18 The plan existence problem is PSPACE-complete for symmetric PSN instances.

4.2 Existence of $\mathbf{P}_{T,S}\mathbf{C}_p$ universal plans

We are now ready to show that the set of symmetric PSN instances admits $\mathrm{P}_{T,S}\mathrm{C}_p$ universal plans for any choice of $0 \leq p < 1$. These universal plans are of an extremely simple type; they perform a *random walk* in the state space.

Definition 19 Let $G = \langle V, E \rangle$ be an arbitrary undirected graph. A *random walk on* G is a sequence $v_1, v_2, \ldots$ of nodes in V such that v_{i+1} is chosen randomly from the neighbours of v_i, *i.e.* the set $\{w \mid \{v_i, w\} \in E,\ v_i \neq w\}$, and each node in this set is an equally likely choice.

Given an undirected graph $G = \langle V, E \rangle$, we represent the edges as unordered pairs of nodes. This implies that $|E| \leq |V|^2/2$.

Lemma 20 Let $G = \langle V, E \rangle$ be an arbitrary undirected graph, v a node in V and $d(v)$ the degree of v, *i.e.* the number of nodes adjacent to v. The expected number of steps a random walk starting in v needs to take before returning to v is $2|E|/d(v)$.
Proof: See for instance Papadimitriou [1994] or Motwani and Raghavan [1995]. □

Theorem 21 Let $G = \langle V, E \rangle$ be an arbitrary undirected graph and $v, w \in V$ two nodes in G. If there exists a path from v to w in G, then the expected number of steps a random walk starting in v needs to take before reaching w is no more than $|V| \cdot |E|$.
Proof: Whenever the walk is in a node v' of degree $d(v')$ which lies on this path, the next step in the walk will with probability $1/d(v')$ be a step "in the right direction", *i.e.* to the next node in the path. If any other neighbour of v' is chosen, the walk will after on average $2|E|/d(v')$ steps return once more to v' and try again, and after on average $\frac{1}{2}d(v')$ tries it will chose the "right" neighbouring node. Thus, the expected number of steps needed to take one step along the path is $2|E|/d(v') \cdot \frac{1}{2}d(v') = |E|$. Since the shortest path from v to w can be at most $|V|$ steps long, the expected total number of steps to complete the path and reach w is no more than $|V| \cdot |E|$. □

Theorem 22 (*Markov's inequality*) If X is a stochastic variable taking non-negative values and $\mathcal{E}(X)$ is the expected value of X, then for any $k > 0$, $\mathbf{Prob}[X \geq k \cdot \mathcal{E}(X)] \leq 1/k$.

Combining the two theorems, we get the following corollary.

Corollary 23 Let $G = \langle V, E \rangle$ be an arbitrary undirected graph, choose two nodes $v, w \in V$ and a number $0 \leq p < 1$, and make a random walk of length $(1/(1-p)) \cdot |V||E|$ on G starting in node v. Then,

1. if the random walk reaches w, then there exists a path from v to w;

2. if the random walk does not reach w, then there is no path from v to w with probability $\geq p$.

Proof: If the random walk reaches w, then trivially there exists a path from v to w.

Assume that the random walk does not reach w but there exists a path from v to w. Let X denote the stochastic variable telling us how long the random walk must be to reach w. Obviously, X is a random variable taking only non-negative values. By Theorem 22, we have that

$$\mathbf{Prob}\left[X \geq \frac{1}{1-p} \cdot \mathcal{E}(X)\right] \leq 1-p$$

and by Theorem 21,

$$\mathbf{Prob}\left[X \geq \frac{1}{1-p} \cdot |V||E|\right] \leq 1-p.$$

That is, the probability that the random walk must take more than $\frac{1}{1-p} \cdot |V||E|$ steps to reach w is less than $1-p$. Thus, if the random walk does not reach w, there is no path from v to w with probability $\geq p$. □

The randomized universal plan that we propose is shown in Figure 2. The next theorem shows that this algorithm is sound and C_p-complete for symmetric PSN structures.

Theorem 24 For any $0 \leq p < 1$, the class of symmetric PSN instances admits universal plans satisfying $\mathrm{P}_{T,S}\mathrm{C}_p$.

Proof: Let $\Phi = \langle \mathcal{P}, \mathcal{O} \rangle$ be an arbitrary symmetric PSN structure, and $\mathcal{G}$ a goal state over $\mathcal{P}$. Consider the algorithm in Figure 2. Assume that the memory of the algorithms is initially loaded such that $z = 0$. Given a state $\mathcal{I}$ over $\mathcal{P}$, the sequence $\mathcal{I}, U_{\mathcal{G}}(\mathcal{I}), U_{\mathcal{G}}^2(\mathcal{I}), \ldots$ is a random walk on the graph

$$G = \langle 2^{\mathcal{P}}, \{\{v, w\} \mid \exists o \in O \text{ such that } o(v) = w \text{ and } v \neq w\}\rangle.$$

(We refer to as G as the *state-transition graph.*) Since Φ is symmetric, G is an undirected graph. Furthermore, $|V| = 2^{|\mathcal{P}|}$ and $|E| \leq |V|^2/2 = 2^{2|\mathcal{P}|-1}$. Since the algorithm keeps track of how many times it has been invoked, it will return either $o_\perp$ or $o_\top$ after at most $\frac{1}{1-p} \cdot 2^{|\mathcal{P}|+2|\mathcal{P}|-1} \geq \frac{1}{1-p} \cdot |V||E|$ steps. By Corollary 23,

1. if a state S satisfying $\mathcal{G}$ is visited, then there is a path from the initial state $\mathcal{I}$ to a state S which satisfies the goal $\mathcal{G}$;

2. if no state satisfying $\mathcal{G}$ is visited, then there is no path from $\mathcal{I}$ to such a state with probability at least p.

Consequently, $U_{\mathcal{G}}$ is complete in the randomized sense with parameter p. To see that $U_{\mathcal{G}}$ is P_T and P_S, one merely has to note that

1. it is possible to uniformly choose a member of a set S by tossing $O(\log(|S|))$ coins, and that

2. the value of p is not part of the input so the memory needed by *count* is fixed.

$\square$

It may seem like a major problem that the universal plan must be invoked as many as $1/(1-p) \cdot 2^{3|\mathcal{P}|-1}$ times before $o_\perp$ is outputted. We can improve this bound somewhat, as discussed below, but not drastically so in the general case since there are symmetric PSN instances having exponentially long shortest solutions, as the next theorem shows.

Theorem 25 For all $n > 1$, there is some PSN instance $\Pi_n = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ with $|\mathcal{P}| = n$ such that $\langle \mathcal{P}, \mathcal{O} \rangle$ is symmetric and such that all plans solving Π are of length $\Omega(2^n)$.
Proof: Define the PSN instance $\Pi_n = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ as

- $\mathcal{P} = \{p_1, \ldots, p_n\}$;

- $\mathcal{O} = \{o_1^+, o_1^-, \ldots, o_n^+, o_n^-\}$ where for $1 \leq i \leq n$:

Universal plan $U_{\mathcal{G}}$.
Input: A state $S \subseteq \mathcal{P}$
if $z = 0$ **then** $z \leftarrow 1$ and $count \leftarrow 0$
else $count \leftarrow count + 1$
if $\mathcal{G}$ is satisfied by S **then** output $o_{\top}$
elsif $count > \frac{2^{3|\mathcal{P}|-1}}{1-p}$ **then** output $o_{\perp}$
else
  begin
    $R \leftarrow \{S' \mid \exists o \in \mathcal{O} \text{ s.t. } S' = S \oplus o \text{ and } S' \neq S\}$
    uniformly choose $S' \in R$
    output an operator o such that $S' = S \oplus o$
  end

Figure 2: The randomized universal plan algorithm.

  - $o_i^+ = (\overline{p_1}, \ldots, \overline{p_{i-2}}, p_{i-1}, \overline{p_i} \Rightarrow p_i)$;
  - $o_i^- = (\overline{p_1}, \ldots, \overline{p_{i-2}}, p_{i-1}, p_i \Rightarrow \overline{p_i})$.

- $\mathcal{I} = \varnothing$;
- $\mathcal{G} = \langle \{p_n\}, \{p_1, \ldots, p_{n-1}\} \rangle$.

Bäckstrom and Nebel [1995] have shown that the shortest solution for Π_n has length $\Omega(2^n)$. To show that $\langle \mathcal{P}, \mathcal{O} \rangle$ is symmetric, we reason along the same lines as in the proof of Theorem 18: Let S be an arbitrary state such that an operator o_i^+ is admissible in S. Then, it is easy to see that o^- is admissible in $S \oplus o^+$, and that if o^- is admissible in S, then o^+ is admissible in $S \oplus o^-$. $\square$

In our formulation of $U_{\mathcal{G}}$ above, we have used the most pessimistic bound on $|E|$, namely $|E| = |V|^2/2$. One way to reduce the number of times that $U_{\mathcal{G}}$ has to be invoked is to give a better estimate of $|E|$. For instance, note that $|E| \leq |V| \cdot |\mathcal{O}|$, since there can not be more ways to leave a state than there are operators. Since in general, $|\mathcal{O}| \ll |V|$, the bound $|V| \cdot |V| \cdot |\mathcal{O}| \leq 2^{2\mathcal{P}} \cdot |\mathcal{O}|$ is often much better than the estimate used above. We can reduce our estimate of $|E|$ even further by considering that not all operators are applicable in all states. An operator o with $|pre^+(o)| + |pre^-(o)| = n$ preconditions is applicable in only $2^{\mathcal{P}-n}$ of the $2^{\mathcal{P}}$ possible states.

4.3 Complexity of Deciding Symmetry

This section contains the proof of Theorem 17.

Lemma 26 Deciding the validity of DNF formulae is a coNP-complete problem.

Proof: Deciding the satisfiability of CNF formulae is an NP-complete problem [Garey and Johnson, 1979] which implies that the complement of this problem is coNP-complete (since the complement of any NP-complete problem is coNP-complete, *cf.* [Papadimitriou, 1994]).

Now, the complementary problem is to decide whether a CNF formula F is false for every assignment to its variables. This is equivalent to the problem of deciding whether the formula $\neg F$ is true for every assignment, *i.e.*, whether $\neg F$ is valid or not. By distributing $\neg$ over F (a task which obviously can be accomplished in polynomial time), we end up with a DNF formula F' which is valid iff $\neg F$ is valid iff F is not satisfiable. Thus, deciding the validity of DNF formulae is a coNP-complete problem. $\Box$

We can now prove Theorem 17, *i.e.*, show that it is a coNP-complete problem to decide whether a PSN structure is symmetric or not.

Proof (of Theorem 17)**:** Let $\Phi = \langle \mathcal{P}, \mathcal{O} \rangle$ be an arbitrary PSN structure. Membership in coNP follows from the observation that if Φ is *not* symmetric, then there exists a state S over $\mathcal{P}$ and an $o \in \mathcal{O}$ (which is admissible in S) such that no operator in $\mathcal{O}$ can transform the state $S \oplus o$ back to S. Given such a state and an operator, this property can be checked in polynomial time which implies that testing if Φ is not symmetric is in NP. Hence, testing if Φ is symmetric is in coNP.

To show hardness for coNP, we exhibit a polynomial time reduction from the problem of deciding validity of DNF formulae. Let $F = C_1 \vee \ldots \vee C_k$ be an arbitrarily chosen DNF formula over the propositions $p_1, \ldots, p_n$. Construct a PSN structure $\langle \mathcal{P}, \mathcal{O} \rangle$ as follows:

- $\mathcal{P} = \{X, p_1, \ldots, p_n\}$;
- $\mathcal{O} = \{o_X, o_1, \ldots, o_k\}$, where
  - $o_X = (\overline{X} \Rightarrow X)$,
  - $o_i = (X, C_i \Rightarrow \overline{X})$, for each clause C_i. Note that C_i is a conjunction of literals.

Clearly, this transformation can be carried out in polynomial time. We claim that $\langle \mathcal{P}, \mathcal{O} \rangle$ is symmetric iff F is valid.

First, assume that F is not valid. Then there exists an assignment M from $\{p_1, \ldots, p_n\}$ to $\{\mathbf{T}, \mathbf{F}\}$ such that F evaluates to false under M. Consider the state $S = \{p_i \mid M(p_i) = \mathbf{T}\}$, the operator o_X and the state $S' = S \oplus o_X$. Since M does not satisfy F, none of the operators in $o_1, \ldots, o_n$ is applicable in S' which implies that $\langle \mathcal{P}, \mathcal{O} \rangle$ is not symmetric.

Assume instead that F is valid and arbitrarily choose a state S over $\mathcal{P}$ and an operator o which is applicable in S. If $X \in S$, then in $o \in \{o_1, \ldots, o_n\}$ so $S \oplus o = S - \{X\}$ and $(S - \{X\}) \oplus o_X = S$. If $X \notin S$, then $o = o_X$ and $S \oplus o_X = \{X\} \cup S$. Since F is valid, at least one operator o_i, $1 \leq i \leq k$ is applicable in $\{X\} \cup S$ so $(S \oplus o_X) \oplus o_i = S$ and $\langle \mathcal{P}, \mathcal{O} \rangle$ is symmetric. $\square$

Despite the hardness of testing symmetry, the structure inherent in many problems gives us a method for determining symmetry efficiently. Recall, for instance, the proofs of Theorems 18 and 25. The instances studied there have the property that for any operator $X, \overline{p} \Rightarrow p$ (where X denotes a set of preconditions not including p or $\overline{p}$), there exists an "undo" operator $X, p \Rightarrow \overline{p}$ and vice versa[4]. Clearly, this property implies symmetry and it can easily be checked in polynomial time. Also note that this property can be generalized (in the obvious way) to operators having arbitrarily many postconditions.

4.4 Complexity of Symmetric Planning

In this Subsection, we prove Theorem 18. First, we introduce the concept of *symmetric Turing machines* (as defined by Lewis and Papadimitriou [1982]), with the aid of *peeking Turing machines*. Then, the acceptance problem for polynomially space-bounded symmetric TMs is shown to be PSPACE-hard and reduced to the plan existence problem for symmetric PSN instance. The reduction is similar to (but considerably more complex than) the reduction used by Bylander [1994] to show PSPACE-hardness of unrestricted PSN planning.

A *peeking* TM (PTM) is a 7-tuple $\langle K, \Sigma, \Sigma_0, k, \Delta, s, F \rangle$, where

- K is a finite set of *states*;

[4]Pairs of operators having this property are said to be *symmetrically reversible* in [Jonsson and Bäckström, 1998].

- Σ is a finite alphabet (the *tape alphabet*);
- $\Sigma_0 \subseteq \Sigma$ is the *input alphabet*;
- $k > 0$ is the number of tapes;
- $s \in K$ is the *initial state*;
- $q_F \subseteq K$ is the set of *final states*; and
- Δ is a finite set of transitions (which are to be defined below).

What is unusual about PTMs is the nature of the transitions; they enable the machine to "peek" one square to the right or left while moving to the right or left, respectively. Formally speaking, a *transition* is of the form $(p, t_1, \ldots, t_k, q)$, where p and q are states, k is the number of tapes, and $t_1, \ldots, t_k$ are *tape triples*. A tape triple is either of the form (ab, D, cd) where $a, b, c, d \in \Sigma$ and D is $+1$ or -1, or of the form $(a, 0, b)$, where $a, b \in \Sigma$.

We begin by an informal description of the one-tape case. A transition of the form $(p, (a, 0, b), q)$ means that when M is in state p and scanning a symbol a, it may rewrite a as b and move into state q, without moving the tape head. A transition of the form $(p, (ab, +1, cd), q)$ means that when M is in state p, scanning symbol a, and the square just to the right of the scanned square contains symbol b, M may rewrite these two squares to contain symbols c and d, respectively, and move the tape head one step to the right. Similarly, a transition $(p, (ab, +1, cd), q)$ signifies a potential left movement of the tape head, except that now the scanned symbol must be b and the one to its left a; these are rewritten as d and c, respectively.

For multitape PTMs, the specified preconditions of each tape triple must be met on each corresponding tape in order for the transition to be applicable and the corresponding actions to be taken.

We set aside a *blank* symbol #, assumed to belong to the tape alphabet of every PTM and to the input alphabet of none. A *configuration* of $M = \langle K, \Sigma, \Sigma_0, k, \Delta, s, F \rangle$ is then a $2k+1$-tuple $(q, w_1, h_1, \ldots, w_k, h_k)$, where $q \in K$, $h_j \in \mathbb{N}$ and w_i is a function from $\mathbb{N}$ to Σ such that $w_j(i) = \#$ for all but finitely many $i \in \mathbb{N}$. The function w_j specifies the contents of tape j, while h_j specifies the position of the head on that tape.

Let $\mathcal{C}(M)$ denote the set of all configurations of M. It is straightforward to define formally what it means for a configuration to *yield* another, based on the informal description of transitions above. We write $C \vdash_M C'$ to denote that $C, C' \in \mathcal{C}(M)$ and C yields C'.

For each string w over Σ_0, we define the *initial configuration with input* w as $I_M(w) = (s, \hat{w}, 0, \hat{e}, 0, \dots, \hat{e}, 0)$; by $\hat{w}$ we mean the function from $\mathbb{N}$ to Σ such that $\hat{w}(i)$ is the i:th symbol of w for $1 \leq i \leq |w|$, and $\hat{w}(i) = \#$ for $i = 0$ and $i > |w|$. The function is defined such that $\hat{e}(i) = \#$ for all $i \in \mathbb{N}$. A *final* configuration is one whose state component is in F. We say that M accepts w if $I_M(w) \vdash_M C_1 \vdash_M \dots \vdash_M C$ for some final configuration C.

To define symmetric PTMs, we first define the *inverse* δ^{-1} of a transition δ: If $\delta = (p, t_1, \dots, t_k, q)$, then $\delta^{-1} = (q, t_1^{-1}, \dots, t_k^{-1}, p)$, where for $1 \leq i \leq k$. If $t_i = (\alpha_i, D_i, \beta_i)$, then $t_i^{-1} = (\beta_i, -D_i, \alpha_i)$. The PTM M is *symmetric* iff $\delta^{-1} \in \Delta$ whenever $\delta \in \Delta$. This implies that if $C \vdash_M C'$, then $C' \vdash_M C$ for all $C \in \mathcal{C}(M)$.

Theorem 27 [Lewis and Papadimitriou, 1982] Let S be any function from $\mathbb{N}$ to $\mathbb{N}$. If a language L is accepted in space S by a k-tape symmetric PTM, $k > 2$, then L is accepted in space S by a 2-tape symmetric PTM.

Lemma 28 The class of languages accepted by symmetric PTMs operating in polynomial space is PSPACE-hard to recognize.
Proof: Given an arbitrary function $S : \mathbb{N} \to \mathbb{N}$, we define

1. DSPACE(S) as the languages accepted by deterministic TMs operating in space S;

2. DSPACE$_P(S)$ as the languages accepted by deterministic PTMs operating in space S;

3. SSPACE$_P(S)$ as the languages accepted by symmetric PTMs operating in space S.

Lewis and Papadimitriou [1982] have shown that

$$\text{DSPACE}_P(S) \subseteq \text{SSPACE}_P(S).$$

Furthermore, it is easy to see that DSPACE$(S) \subseteq$ DSPACE$_P(S)$ (by simply not taking advantage of the possibility to "peek"). We therefore have

$$
\begin{aligned}
\textsc{Pspace} &= \bigcup_{k=1}^{\infty} \textsc{Dspace}(n^k) \\
&\subseteq \bigcup_{k=1}^{\infty} \textsc{Dspace}_P(n^k) \\
&\subseteq \bigcup_{k=1}^{\infty} \textsc{Sspace}_P(n^k)
\end{aligned}
$$

which concludes the lemma. □

Proof (of Theorem 18): With the aid of the previous lemma and theorem, we can show that the plan existence problem is PSPACE-complete for symmetric PSN instances. Membership in PSPACE follows from the fact that the plan existence problem for (unrestricted) PSN instances is in PSPACE. We show hardness for PSPACE by a reduction from the language recognition problem for symmetric PTMs operating in polynomial space.

Let M be an arbitrary polynomial-space bounded symmetric TM and let $x = x_1 x_2 \ldots x_n$ be an input string of length $|x| = n$. Assume that the total number of tape cells used by M is bounded by some polynomial p in $|x|$. We introduce propositional atoms as follows:

$in_1(i, x)$: symbol x is in tape 1's cell i, $1 \leq i \leq p(|x|)$;

$in_2(i, x)$: symbol x is in tape 2's cell i, $1 \leq i \leq p(|x|)$;

$pos(i)$: M is reading tape cell i;

$state(q)$: M is in state q;

$accept$: M accepts the input.

By Lemma 27, it is sufficient to consider two tapes. To simplify the presentation, we only demonstrate the encoding for the case when M is a 1-tape TM; thus we replace $in_1(i, x)$ and $in_2(i, x)$ with $in(i, x)$.

If q_0 is the initial state of M, we define the initial and goal state as

$$
\begin{aligned}
\mathcal{I} &= \{state(q_0), pos(1), in(1, x_1), \ldots, in(n, x_n), in(n+1, \#), \\
&\qquad \ldots, in(p(|x|), \#)\} \\
\mathcal{G} &= \{accept\}.
\end{aligned}
$$

We continue by showing how the transitions of M can be encoded by operators. Consider a transition of the type $(p, (ab, +1, cd), q)$, and assume that $a \neq c$, $b \neq d$ and that the tape head is in position i. Such a transition is represented by the operator

$$\begin{aligned} t^{+} \quad : \quad & pos(i), state(p), in(i,a), in(i+1,b), \overline{pos(i+1)}, \overline{state(q)}, \\ & \quad \overline{in(i,c)}, \overline{in(i+1,d)} \Rightarrow \\ & \overline{pos(i)}, \overline{state(p)}, \overline{in(i,a)}, \overline{in(i+1,b)}, pos(i+1), state(q), \\ & \quad in(i,c), in(i+1,d). \end{aligned}$$

The preconditions may seem puzzling; why introduce the negative preconditions $\overline{pos(i+1)}$, $\overline{state(q)}$, $\overline{in(i,c)}$ and $\overline{in(i+1,d)}$, since we can, for instance, never reach a state such that $in(i,a)$ and $in(i,c)$ holds simultaneously. However, this is only correct under the assumption that we start in the initial state as defined above, which is something that we cannot guarantee. As will become apparent later on, these extra preconditions are needed to make the resulting planning problem symmetric.

By the symmetry of M there also exists a transition $(q, (cd, -1, ab), p)$. Assuming the tape head is in position $i+1$, we represent this transition by the operator

$$\begin{aligned} t^{-} \quad : \quad & pos(i+1), state(q), in(i,c), in(i+1,d), \overline{pos(i)}, \overline{state(p)}, \\ & \quad \overline{in(i,a)}, \overline{in(i+1,b)} \Rightarrow \\ & \overline{pos(i+1)}, \overline{state(q)}, \overline{in(i,c)}, \overline{in(i+1,d)}, pos(i), state(p), \\ & \quad in(i,a), in(i+1,b). \end{aligned}$$

It should be obvious that if t^{+} is applicable in S, then t^{-} is applicable in $S \oplus t^{+}$ and $S = (S \oplus t^{+}) \oplus t^{-}$. In other words, the resulting set of operators is symmetric.

To exemplify why the negative preconditions are needed, define n^{+} and n^{-} as t^{+} and t^{-} but without these preconditions. Assume we are in the "strange" state

$$\begin{aligned} I \quad = \quad & \{pos(i), state(p), in(i,a), in(i+1,b), pos(i+1), state(q), \\ & in(i,c), in(i+1,d)\}. \end{aligned}$$

We then have

$$I \oplus n^{+} = \{pos(i+1), state(q), in(i,c), in(i+1,d)\}$$

and

$$(I \oplus n^+) \oplus n^- = \{pos(i), state(p), in(i,a), in(i+1,b)\}$$

which does not equal I. This situation is prevented by the negative preconditions since t^+ is not applicable in the state I.

The negative preconditions must be introduced with a certain amount of care. Consider the transition $(p, (ab, +1, ad), q)$. The straightforward definition of the corresponding operator is

$$\begin{aligned} t^+ \ : \ & pos(i), state(p), in(i,a), in(i+1,b), \overline{pos(i+1)}, \overline{state(q)}, \\ & \quad \overline{in(i,a)}, \overline{in(i+1,d)} \Rightarrow \\ & \overline{pos(i)}, \overline{state(q)}, in(i,a), \overline{in(i+1,b)}, pos(i+1), state(q), \\ & \quad in(i,a), in(i+1,d) \end{aligned}$$

This operator is not applicable in any state since we have the contradictory preconditions $in(i,a)$ and $\overline{in(i,a)}$. The correct encoding is

$$\begin{aligned} t^+ \ : \ & pos(i), state(p), in(i,a), in(i+1,b), \overline{pos(i+1)}, \overline{state(q)}, \\ & \quad \overline{in(i+1,d)} \Rightarrow \\ & \overline{pos(i)}, \overline{state(p)}, \overline{in(i+1,b)}, pos(i+1), state(q), in(i+1,d) \end{aligned}$$

Having seen these examples, it is easy to define operators for the other types of transitions and to extend the construction to the multi-tape case. By Theorem 27, we only have to consider two tapes which simplifies the definition of operators considerably.

M accepts its input iff it reaches a state $q_F \in F$. Introduce the following operators for each $q_F \in F$:

$$\begin{aligned} a^+ \ &: \ state(q_F), \overline{accept} \Rightarrow state(q_F), accept \\ a^- \ &: \ state(q_F), accept \Rightarrow state(q_F), \overline{accept} \end{aligned}$$

By adding both these operators we preserve the symmetry condition and enable the goal state to be reached iff M accepts its inputs. Since the previously introduced operators precisely encode the transitions of M, M accepts input w iff the corresponding PSN instance has a solution.

Finally, we have to show that this is a polynomial-time transformation. This is, however, easy, by noting that the number of propositions is at most

$$k \cdot |\Sigma| \cdot p(|x|) + p(|x|) + |K| + 1$$

where k is the number of tapes, and the number of operators is at most $|\Delta| \cdot p(|x|) + |F|$. $\Box$

5 Randomized Universal Planning: Experiments

We have implemented a planner, which we call Stocplan, based on the randomized universal planning algorithm presented in Figure 2. To turn the universal plan algorithm into a traditional planner, we invoke the algorithm repeatedly until it returns either $o_\top$, indicating a plan exists, or $o_\perp$, indicating a plan is not likely to exist.

To experimentally evaluate Stocplan, we tested it and compared it to a deterministic planner on a number of domains. The experiments were not designed to test a particular hypothesis, but are rather exploratory in nature. The questions we primarily had in mind were:

1. How does Stocplan compare to a traditional, deterministic planner?
2. What characteristics of the problem domain are crucial for Stocplan's performance?

For comparison, we chose the planner Graphplan [Blum and Furst, 1997], since it is widely acknowledged as one of the fastest propositional planners available. However, a number of circumstances make the comparison somewhat unfair:

1. Graphplan always finds a shortest plan if the given planning instance has a solution while Stocplan finds a (not necessarily optimal) plan with a certain probability; the former task may very well be harder.
2. Stocplan can only solve symmetric planning instances.
3. Graphplan can only deal with conjunctions of positive literals in operator preconditions; this limitation can be circumvented using a standard transformation, but doing so enlarges the domain (*i.e.* the number of propositions).

It is our hope that the results are illustrative despite these imperfections.

5.1 Experiment design

We measured the runtime of the two planners on a number of instances of different planning problems. All problem instances were solvable. All trials were performed on a SUN Sparcstation 10[5] and with a time limit of 300 seconds (= 5 minutes).

The runtime for GRAPHPLAN is the mean of two trials. Since this planner is deterministic, the only differences between trials are those caused by "noise" in the environment, which we have minimized as far as possible; the difference was in all cases small compared to the average (at most 8%).

Because STOCPLAN is randomized, to present the "runtime" as a single value would be misleading, no matter how many trials it is the average of. Instead, we take the runtime to be a stochastic variable, X, and hypothesize that it is exponentially distributed. In support of this hypothesis, we can only submit the fact that exponential distribution is natural for stochastic variables representing the time until a certain event occurs, given that the event has a certain probability of occurring at each point in time; this description certainly applies to the runtime of STOCPLAN. The exponential distribution function is

$$F(x|\mu) = 1 - e^{-\frac{x}{\mu}}$$

$F(x|\mu)$ is the probability that an observation of the stochastic variable X will be in the interval $[0, x]$, so observations less than zero have zero probability. The single parameter μ is positive and is also the expected value for a variable of the distribution. A cumulative histogram (from a sample of runs on one of the blocksworld instances) overlayed with the corresponding curve of the distribution function (figure 3) also indicates a likely correspondence.

¿From the experimental data, we calculate (using the MATLAB statistics toolbox[6]) an estimate of μ in the form of a 90% confidence interval, *i.e.* a range of possible values such that the probability of μ being among them is 0.9, given the observed data set. Based on this, we calculate an estimate of the 90th percentile, *i.e.* a value such that the probability of the runtime being less than this value is at least 0.9, given that the runtime is exponentially distributed with a parameter somewhere in the interval. The estimated 90th percentile is the closest we have to "running time", since it is the time we expect we would have to wait in order to be 90% certain that we have not missed a solution. We also compare the estimated value to the measured 90th percentile of the experimental data.

For instances where the percentage of trials solved by STOCPLAN within the limit of 300 seconds is less than 90%, we have not calculated estimates

[5]SUN and Sparcstation 10 are trademarks of SUN Microsystems.

[6]MATLAB is a trademark of MathWorks Inc.

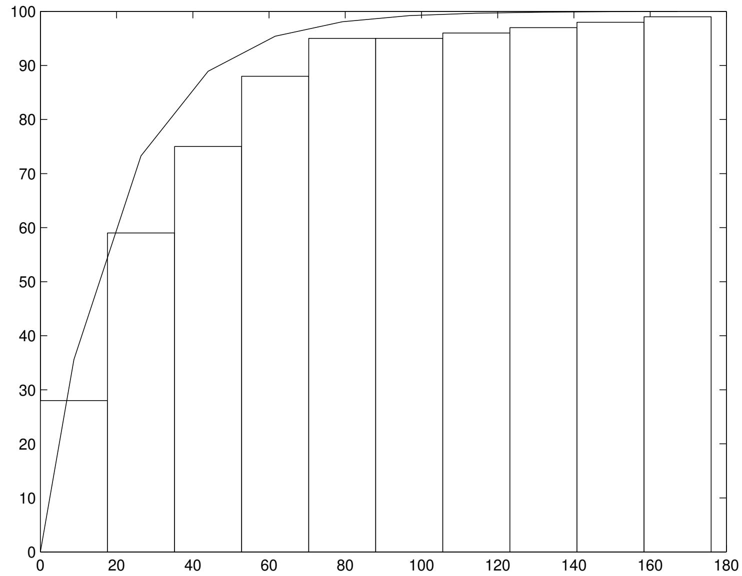

Figure 3: Correspondence between the exponential distribution function (line) and the distribution of the measured runtimes (bars).

of μ or the 90th percentile since those calculations would be biased by the lack of exact data on runtimes exceeding 300 seconds.

5.2 Two Toy Problems

We begin our investigation by considering two toy problems; the traditional blocksworld domain and a puzzle domain involving movable tiles. The blocksworld model uses the single-step operator;

$$\begin{array}{rcl} move(x,y,z) & : & on(x,y), clear(x), clear(z) \Rightarrow \\ & & \overline{on(x,y)}, on(x,z), \overline{clear(z)}, clear(y) \end{array}$$

Special operators are used for the cases when either source or destination is the table.

The tile puzzle consists of an $n \times n$ array of squares with $n^2 - 1$ labeled square tiles laid out on it, as shown in figure 4. When $n = 3$ there are eight tiles, wherefore this problem is also known as the Eight puzzle [Korf, 1987]. Tiles can be moved, vertically or horizontally, into an adjacent square if it is the single empty square. The problem consists in changing the tiles from one

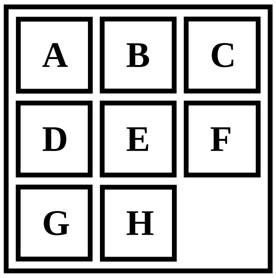

Figure 4: The tile puzzle.

configuration to another. Modeling the problem is straightforward; a proposition $at(l, x, y)$ represents the fact that the tile labeled l is in square (x, y). To avoid negative preconditions the single empty square is also represented as a tile, labeled “Blank”. For example, with $n = 3$, the operator for moving a tile labeled x, where $x \neq$ “Blank”, downwards from the center position would be

$$\begin{aligned} down(x) \quad : \quad & at(x, 2, 2), at(\text{Blank}, 2, 3) \Rightarrow \\ & \overline{at(x, 2, 2)}, \overline{at(\text{Blank}, 2, 3)}, at(x, 2, 3), at(\text{Blank}, 2, 2) \end{aligned}$$

We tried GRAPHPLAN and STOCPLAN on six instances of the blocksworld problem with $n = 8, 9$ and goal configurations chosen to vary properties of the solution, in particular the solution length. For the tile puzzle, we used five instances with $n = 3$ and random goal configurations. Results are presented in tables 1 and 2. Plan length is the shortest plan, found by GRAPHPLAN. Table 1 shows both the number of operators and the number of time steps in the shortest plan; since GRAPHPLAN places operators in parallel whenever possible, the number of time steps may be smaller. In the tile puzzle domain, however, only one operator can be placed in each time step, so the two measures coincide.

In the blocksworld domain, GRAPHPLAN clearly outperforms STOCPLAN. It may appear to do so also on the tile puzzle, but the difference is less and grows lesser as plans grow longer; on the longest example, even the (pessimistic) estimate of the 90th percentile limit is less than the actual running time of GRAPHPLAN.

¿From the results of the blocksworld example with $n = 8$ it appears that STOCPLAN's expected runtime depends both on plan length, *i.e.* the number

n	Plan length*	Graph-plan	Stocplan μ 90% c.i.	Stocplan 90th percentile Estimated	Stocplan 90th percentile Measured	Stocplan % solved
8	8/2	1.2	1.9 – 2.6	6.0	5.0	100.0
	8/8	6.2	6.1 – 8.5	19.6	14.0	100.0
	12/6	4.1	78.8 – 109.6	252.3	229.5	93.0
	12/12	9.6				74.0
9	12/10	6.6				58.0
	14/13	30.4				15.0

* First number is the number of actions, second the number of time steps.

Table 1: Experimental results for the blocksworld domain.

Plan length	Graph-plan	Stocplan μ 90% c.i.	Stocplan 90th percentile Estimated	Stocplan 90th percentile Measured
16	6.1	15.3 – 21.3	49.0	39
22	13.0	20.1 – 28.0	64.3	57.5
22	12.0	25.0 – 34.8	80.2	64
24	20.2	22.0 – 30.6	70.5	61
28	99.7	23.4 – 32.5	74.8	63

Table 2: Experimental results for the tile puzzle domain.

n	GRAPH-PLAN	STOCPLAN			
		μ 90% c.i.	90th percentile		% solved
			Estimated	Measured	
8	10.5	3.5 – 4.0	9.25	7.5	100.0
9	55.5	4.4 – 5.5	12.6	10	100.0
10	*	18.4 – 22.6	52.1	46	100.0
11	*	61.2 – 71.4	173.6	144	99.2
12	*				59.6
13	*				8.0
14	*				0.0

Table 3: Experimental results for the exponential plan domain.

of operators in the shortest plan, and on plan "seriality", *i.e.* the number of sequential time steps needed. This is somewhat surprising, since STOCPLAN examines only totally ordered sequences of operators.

5.3 Exponential Length Plans

To explore the hypothesis that plan length has an influence on the relative performance of the two planners, we go to an extreme; the construction in Theorem 25, which yields planning instances with minimal length $\Omega(2^n)$.

Both planners were tested on instances of this problem ranging in size from 8 up to 14 (on smaller instances, both planners are indistinguishably fast and on larger instances, both fail to yield a solution within the time limit of 300 seconds). The results are presented in table 3. Trials exceeding 300 seconds were aborted, and are marked with an asterisk in the table.

The performance of both planners degrades in a similar way as n grows, but STOCPLAN is clearly able to handle larger instances than GRAPHPLAN.

5.4 The Tunnel Domain

The tunnel domain is an example that has been used in control theory. It consists of a tunnel (see Figure 5) divided into n sections such that the light can be switched on and off independently in each section. The light switches are located at each end of a section. It is also assumed that one can walk through a section only if the light is on in that section. As a typical instance

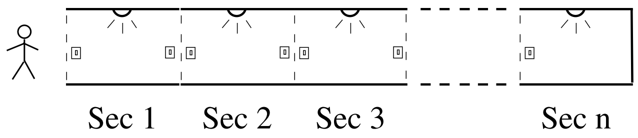

Figure 5: Illustration of the tunnel domain.

n	GRAPH-PLAN	STOCPLAN			
		μ 90% c.i.	90th percentile Estimated	90th percentile Measured	% solved
12	0.8	1.2 – 1.7	4.0	3	100.0
14	1.4	7.7 – 10.7	24.6	19.5	100.0
16	11.7	40.2 – 56.0	128.8	100.5	100.0
18	65.2				73.0
20	*				13.0

Table 4: Experimental results for the tunnel domain.

of this problem assume that all lights are off and the goal is to turn the light in the innermost section on while, in the end, leaving all other lights off. This can be achieved by walking into the tunnel, repeatedly switching on the light in each section until the innermost section is reached, then leaving the tunnel again, repeatedly switching off the light in each section, but leaving the innermost light on.

Modeling this problem is once again straightforward; a proposition $light(i)$ represents the fact that the light is on in section i. The operators

$$
\begin{aligned}
on_i &: light(1), \ldots, light(i-1) \Rightarrow light(i) \\
off_i &: light(1), \ldots, light(i-1) \Rightarrow \overline{light(i)}
\end{aligned}
$$

turn the light in section i on and off, respectively, provided that the light in all sections leading up to section i is on (so one can walk to the switch). The initial state in the problem described above is $\mathcal{I} = \varnothing$ and the goal is $\mathcal{G} = \langle\{light(n)\}, \{light(1), \ldots, light(n-1)\rangle$. Note that a plan for this problem is not particularly long, containing no more than $2n$ operators.

The planners were compared on instances of this problem ranging in size from 12 up to 20; results are presented in table 4. Since GRAPHPLAN can not deal with negative literals in preconditions or goal, we use the standard

Problem	Reachable states	Plan length	Worst case
Blocksworld, with $n = 8$	$\approx 2^{18}$	12	> 300
Tile puzzle	$\approx 2^{17}$	28	80.2
Exp. length domain, with $n = 10$	2^{10}	1024	52.1
Tunnel domain, with $n = 16$	2^{16}	31	128.8

Table 5: Comparison of some domain characteristics and the estimated running time of STOCPLAN.

transformation of replacing each propositional atom p with two atoms p_T and p_F, with the intended interpretation that p is true when p_T is true and p_F false (and analogously when p is false), on instances for GRAPHPLAN. Note that this does not affect the length of the shortest plan.

Apart from plan length, the tunnel and exponential length plan domains share a great deal of structural similarity. Also, the performance of the two planners degrades in the same way, though in this case STOCPLAN does so more quickly.

5.5 Discussion of the Results

Though example domains have not been chosen in any very principled manner, the presented collection clearly illustrates that neither planner is consistently faster than the other. Plan length appears to be the crucial factor for the relative performance, as can be seen from the results on the exponential length and tile puzzle domains. However, GRAPHPLAN is known to be sensitive to plan length.

Nor to the second question does the results give any conclusive answer. Table 5 shows a comparison between some domain characteristics and the worst estimated running time of STOCPLAN, for some of the examples.

Plan length is obviously not a very important factor; estimated running time in the exponential length domain with $n = 10$, requiring a 1024-step plan, is much less than in the blocksworld domain with $n = 8$, requiring only at most a 12-step plan. Neither is the size of the state space; all instances of the tile puzzle, which have 9^2 propositions and more than 2^{17} reachable states are solved faster than the tunnel instance with $n = 16$, which has only 2^{16} reachable states.

As a tentative explanation, we hypothesize that an important factor is the average ratio of "good" choices of operator to the total number of applicable operators, over the reachable set of states. A choice of operator may be considered "good" in state s if it is possible to reach a goal satisfying state without returning to s; in essence, without backtracking. This can also be characterized as a ratio of the number of paths that reach the goal to the total number of paths to take. In the exponential length domain, for an example of one extreme, there is in each state only two applicable operators, one of which is in the right direction and therefore "good". This can explain why STOCPLAN reaches the goal relatively fast, even though the plan is very long. Blocksworld offers an example of the other extreme; in the worst case there are n blocks to move, $n-1$ places to move each block, and exactly one of them has to be taken for the goal to be reachable without undoing this step. This would also explain why STOCPLAN is faster on problems requiring fewer time steps for the same number of operations. Operations that can be executed in parallel can also be executed serially in any order, increasing the number of choices that lead to the goal.

6 Future Work

As the reader may already have noted, a large number of question are left open by this paper. The following are two of the questions that the authors find interesting.

1. As was pointed out in Section 4, bound on the number of steps needed by the random walk can be substantially reduced by providing sharper estimates of $|V|$ and $|E|$, the number of nodes and edges in the state-transition graph. Some ways of estimating $|E|$ more accurately were discussed, but $|V|$ was estimated only with the worst case bound of $2^{\mathcal{P}}$. Note also that it is sufficient to consider the number of states that are reachable from the initial state in the state-transition graph, so closer estimates of this quantity would also result in improved performance.

   Even more interesting would be methods of estimation that can be run "in parallel" with the randomized planner, improving the two bounds incrementally during the random walk.

   Another way of decreasing the running time is to improve the basic random walk technique. Even though this seems very hard in the general case, there may be domain-dependent heuristics that can speed up the planning process under favourable circumstances.

2. What domain characteristics are crucial to the performance of Stocplan, or randomized planning algorithms in general? To test the hypothesis presented in Section 5.5 empirically, a suitable domain must be found. Since there is a similarity between the idea of "ratio of successful paths" and the concept of trivial and laborious serializability defined by Barret and Weld [1994], symmetric versions of their $D^m S^n$ domains are promising candidates.

   The concept of serializability as originally defined by Korf [1987] is not directly applicable however, since Stocplan does not consider subgoals. As, for example, the results from the tile puzzle domain show, nonserializable problems are not necessarily hard for a randomized planner like Stocplan.

7 Conclusions

We have proposed a stricter definition of universal plans which guarantees a weak notion of soundness not present in the original definition. In addition, we have identified three different types of completeness which capture different behaviors exhibited by universal plans. A-completeness guarantees that if there exists a plan from the current state to the goal state, then the universal plan will find a solution in a finite number of steps. R-completeness is the converse of A-completeness, *i.e.*, if there does not exist a plan from the current state to the goal state, then the universal plan will report this after a finite number of applications. R^+-completeness is a stronger version of R-completeness, stating that if there does not exist a plan from the current state to the goal state, then the universal plan will report this after one application. We have shown that universal plans which run in polynomial time and are of polynomial size cannot be A-complete unless the polynomial hierarchy collapses. However, by dropping either the polynomial time or the polynomial space requirement, the construction of A- and R^+-complete universal plans becomes trivial.

As a complement to the classical universal planning which concentrate on deterministic algorithms, we consider universal plans which have access to a random source. For a randomized version of completeness and a restricted class of problems, we showed that there exists randomized universal plans running in polynomial time and using polynomial space which are sound and complete. We also showed that this class of problems is nontrivial since the planning problem is PSPACE-hard. Experiments with an implementation of the randomized planning algorithm yielded inconclusive results; compared to a deterministic planner we found neither planner to be consistently faster, and we could not conclusively identify any domain characteristics crucial to the performance of the randomized planner.

Acknowledgments

This research has been sponsored by the *Swedish Research Council for the Engineering Sciences* (TFR) under grants Dnr. 93-291, 95-731, 96-737 and 97-301, the *Wallenberg Foundation* and the *ECSEL/ENSYM* graduate studies program.

References

[Allen *et al.*, 1990] J. Allen, J. Hendler, A. Tate, editors, Readings in Planning, San Mateo, CA, 1990.

[Bäckström and Nebel, 1995] C. Bäckström, B. Nebel, Complexity results for SAS^+ planning, *Comput. Intell.* 11(4) (1995) 625–655.

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

[Baral and Tran, 1998] C. Baral, S. C. Tran, Relating theories of actions and reactive control, Linköping Electronic Articles in Computer and Information Science 3(9) (1998).

[Barret and Weld, 1994] A. Barret, D. S. Weld, Partial-order planning: Evaluating possible efficiency gains, *Artif. Intell.* 67 (1994) 71–112.

[Blum and Furst, 1997] A. L. Blum, M. L. Furst, Fast planning through planning graph analysis. *Artif. Intell.* 90(1–2) (1997) 281–300.

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

[Chapman, 1989] D. Chapman, Penguins can make cake, *AI Mag.* 10(4) (1989) 45–50.

[Garey and Johnson, 1979] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.

[Ginsberg, 1989a] M. L. Ginsberg, Ginsberg replies to Chapman and Schoppers, *AI Mag.* 10(4) (1989) 61–62.

[Ginsberg, 1989b] M. L. Ginsberg, Universal planning: An (almost) universally bad idea, *AI Mag.* 10(4) (1989) 40–44.

[Green, 1969] C. Green, Application of theorem proving to planning, in: Proc. 1st Int'l Joint Conf. on Artif. Intell. (IJCAI-69), 1969 pp. 219–239. Reprinted in Allen *et al* [1990], pp. 67–87.

[Johnson, 1990] D. S. Johnson, A catalog of complexity classes, in: Jan van Leeuwen (Ed.), Handbook of Theoretical Computer Science: Algorithms and Complexity, volume A, chapter 2, Elsevier, Amsterdam, 1990, pp. 67–161.

[Jonsson and Bäckström, 1996] P. Jonsson, C. Bäckström, On the size of reactive plans, in: Proc. 13th (US) Nat'l Conf. on Artif. Intell. (AAAI-96), 1996, pp. 1182–1187.

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

[Karp and Lipton, 1982] R. M. Karp, R. Lipton, Turing machines that take advice, Enseign. Math. 28 (1982) 191–209.

[Köbler and Watanabe, 1999] J. Köbler, O. Watanabe, New collapse consequences of NP having small circuits, SIAM J. Comput. 28(1) (1999) 311–324.

[Korf, 1987] R. E. Korf, Planning as search: A quantitative approach, Artif. Intell. 33 (1987) 65–88.

[Lewis and Papadimitriou, 1982] H. R. Lewis, C. H. Papadimitriou, Symmetric space-bounded computation, Theoret. Comput. Sci. 19 (1982) 161–187.

[Motwani and Raghavan, 1995] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.

[Papadimitriou, 1994] C. H. Papadimitriou. Computational Complexity, Addison Wesley, Reading, MA, 1994.

[Sacerdoti, 1975] E. D. Sacerdoti, The non-linear nature of plans, in: Proc. 4th Int'l Joint Conf. on Artif. Intell. (IJCAI-75), 1975, pp. 206–214.

[Savitch, 1970] W. J. Savitch, Relationships between nondeterministic and deterministic tape complexities, J. Comp. System Sci. 4(2) (1970) 177–192.

[Schoppers, 1987] M. J. Schoppers, Universal plans for reactive robots in unpredictable environments, in: Proc. 10th Int'l Joint Conf. on Artif. Intell. (IJCAI-87), 1987, pp. 1039–1046.

[Schoppers, 1989] M. J. Schoppers, In defense of reaction plans as caches. *AI Mag.* 10(4) (1989) 51–62.

[Schoppers, 1994] M. J. Schoppers, Estimating reaction plan size, in: Proc. 12th (US) Nat'l Conf. on Artif. Intell. (AAAI-94), 1994, pp. 1238–1244.

[Selman, 1994] B. Selman, Near-optimal plans, tractability, and reactivity, in: Proc. 4th Int'l Conf. on Principles of Knowledge Repr. and Reasoning (KR-94), 1994, pp. 521–529.

[Williams and Nayak, 1997] B. C. Williams, P. Pandurang Nayak, A reactive planner for a model-based executive, in: Proc. 15th Int'l Joint Conf. on Artif. Intell. (IJCAI-97), 1997, pp. 1178–1185.