Incremental Planning

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

Abstract. Ambros-Ingerson and Steel suggested to interleave planning and execution through incremental planning, *ie.* using a planner that can output valid prefixes of the final plan before it has finished planning. This method could considerably bring down the time lost in planning, especially in dynamic domains, where replanning has to occur frequently. We improve on the basic idea, avoiding certain problems, by presenting an incremental planner with provable properties for a restricted class of planning problems, the 3S class. Finding out whether a 3S instance is solvable or not is computationally tractable, despite the fact that generating a plan is inherently intractable. By first testing whether an instance is solvable or not, we can avoid outputting prefixes of invalid plans in the latter case. Furthermore, making the reasonable assumption that natural problems have non-exponential-size solutions, we can also plan efficiently in practice since we need not waste time on non-solvable instances.

1 Introduction

Clasical planning is sometimes considered inadequate for coping with dynamic worlds and execution failures. First, planning is computationally expensive in the general case; plan existence for STRIPS-style formalisms is undecidable in the first-order case [Chapman, 1987] and PSPACE-complete in the propositional case [Bylander, 1994]. This is a problem in time-critical applications since we have to wait for the planner to generate the whole plan before we can start executing it. Second, when the execution of an action fails, or other changes in the world force us to some unexpected world state, we have to reinvoke the planner to find a new plan from the current world state to the goal. This is known as *replanning* and is often considered an infeasible method since it can be as costly as planning. So called *reactive planning* is sometimes considered a better alternative, but it must be noted that, from a computational perspective, a reactive planner could not perform any better than a classical planning/replanning system unless we allow it to solve a relaxed version of the problem.

Ambros-Ingerson & Steel [Ambros-Ingerson and Steel, 1987] have addressed the problems above by suggesting to interleave planning with execution. Their idea is to use a planner which will as soon as possible output a *prefix* of the final solution—that is, a set of actions which are the first actions of the final solution. We can, thus, immediately start executing this plan prefix and the planner will concurrently continue to generate the rest of the solution and output successive prefixes for execution whenever possible. This approach has two advantages. First, even if it takes a long time to generate the whole solution, we can hope to start executing a prefix within a reasonably short time. In most applications, we can expect action execution to take place on a relatively much slower time scale than planning, so outputting prefixes now and then will keep the plan executor busy and not much time will be lost in planning. Second, if we have to replan, we only have to wait for the first prefix of the new plan before we can, once again, start execution. That is, for each failure, we only lose the time it takes to generate a prefix of the new plan. We will refer to this method as *incremental planning.*

There is an obvious problem with incremental planning: In general, there is no guarantee that the planner can output a prefix before it has generated the whole solution. Even worse, before the planner has terminated we cannot know if there is a solution at all. Hence, if the planner outputs a prefix, we cannot know whether it is a prefix of a solution or not. Executing the prefix of a non-solution 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 prefix may prevent us from reaching the alternative goal.

Now, to be more precise, we have to distinguish between the *plan existence* problem, *ie.* finding out whether a solution exists or not, and the *plan generation* problem, *ie.* actually generating a plan. While other authors in the literature have restricted themselves to analysing the complexity of plan existence only, under various restrictions, we have also analysed the corresponding plan generation problems [Bäckström and Nebel, 1993, Jonsson and Bäckström, 1994a]. It turned out that while plan existence is typically only conjectured intractable (*ie.* NP- or PSPACE-complete), plan generation is often inherently intractable since optimal plans may be of exponential length. There are even a few problem classes where we could not determine the complexity of plan existence, although plan generation is inherently intractable. This raised the question of whether there might be problems exhibiting tractable plan existence but intractable plan generation—a question which we answer positively in this paper by presenting the *3S* class, which exhibits this property.

Incremental planning is obviously attractive for the 3S class, since we can efficiently find out in advance whether an instance has a solution or not. Having first verified that a solution exists, we can be assured both that all prefixes are indeed valid prefixes of a solution and that we will not waste time on the intractable plan generation problem for a non-solvable instance. We augment this theoretical result with presenting an incremental planning algorithm for the 3S class. The planning algorithm is proven correct and runs in polynomial time in the length of the solution, which we will refer to as *solution-polynomial time.*

The remainder of the paper is organized as follows. Section 2 recapitulates the propositional STRIPS formalism, Section 3 defines the problem class 3S and Section 4 proves that plan existence is tractable for 3S. Section 5 presents the incremental planning algorithm for the 3S class and, finally, Section 6 discusses the results and future work.

2 Basic Formalism

We base our work in this paper on the propositional STRIPS formalism with negative goals [Bylander, 1994], which is equivalent to most other variants of propositional STRIPS [Bäckström, 1995].

Definition 2.1 Given a set of operators $\mathcal{O}$, we define the set of all operator sequences over $\mathcal{O}$ as $Seqs(\mathcal{O}) = \{\langle\rangle\} \cup \{\langle o\rangle;\omega | o \in \mathcal{O} \text{ and } \omega \in Seqs(\mathcal{O})\}$, where ; is the sequence concatenation operator.

Definition 2.2 An instance of the *PSN planning problem* is a quadruple $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*{}^+, s_*{}^- \rangle \rangle$ where

- $\mathcal{P}$ is a finite set of *atoms*;
- $\mathcal{O}$ is a finite set of *operators* of the form $\langle pre^+, pre^-, add, del, name \rangle$, where $pre^+, pre^- \subseteq \mathcal{P}$ denote the *positive* and *negative precondition* respectively, satisfying $pre^+ \cap pre^- = \varnothing$, $add, del \subseteq \mathcal{P}$ denote the *positive* and *negative postcondition* (add and delete list) respectively, satisfying $add \cap del = \varnothing$, and $name$ is a unique identifier;
- $s_0 \subseteq \mathcal{P}$ denotes the *initial state* and $s_*{}^+, s_*{}^- \subseteq \mathcal{P}$ denote the *positive* and *negative goal* respectively, satisfying $s_*{}^+ \cap s_*{}^- = \varnothing$;

The unique identifier for each operator is not technically necessary but it will simplify the forthcoming proofs. For $o = \langle pre^+, pre^-, add, del, name \rangle \subseteq \mathcal{O}$, we write $pre^+(o)$, $pre^-(o)$, $add(o)$, $del(o)$ and $name(o)$ to denote pre^+, pre^-, add, del and $name$ respectively. A sequence $\langle o_1, \ldots, o_n \rangle \in Seqs(\mathcal{O})$ of operators is called a *PSN plan* (or simply plan) over Π. We can now define when a plan solves a planning instance.

Definition 2.3 The ternary relation $Valid \subseteq Seqs(\mathcal{O}) \times 2^{\mathcal{P}} \times (2^{\mathcal{P}} \times 2^{\mathcal{P}})$ is defined s.t. for arbitrary $\langle o_1, \ldots, o_n \rangle \in Seqs(\mathcal{O})$ and $S, T^+, T^- \subseteq \mathcal{P}$, $Valid(\langle o_1, \ldots, o_n \rangle, S, \langle T^+, T^- \rangle)$ iff either

1. $n = 0$, $T^+ \subseteq S$ and $T^- \cap S = \varnothing$ or
2. $n > 0$, $pre^+(o_1) \subseteq S$, $pre^-(o_1) \cap S = \varnothing$ and $Valid(\langle o_2, \ldots, o_n \rangle, (S - del(o_1)) \cup add(o_1), \langle T^+, T^- \rangle)$.

A plan $\langle o_1, \ldots, o_n \rangle \in Seqs(\mathcal{O})$ is a *solution* to Π iff $Valid(\langle o_1, \ldots, o_n \rangle, s_0, \langle s_*^+, s_*^- \rangle)$.

We can now formally define the planning problems that we will consider in this paper.

Definition 2.4 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^- \rangle \rangle$ be a given PSN instance. The *plan existence problem* is to decide whether there exists or not exists some $\omega \in Seqs(\mathcal{O})$ s.t. ω is a solution to Π. The *plan generation problem* is to find some $\omega \in Seqs(\mathcal{O})$ s.t. ω is a solution to Π or answer that no such ω exists.

3 The 3S Class

We begin by defining *dependency graphs* on planning instances. Such a graph represents for each atom p, which other atoms we will possibly have to add or delete in order to add or delete p. The idea is not new; a more restricted variant is used by Knoblock [Knoblock, 1994] in his ALPINE system.

Definition 3.1 Let $p \in \mathcal{P}$ and let $Q \subseteq \mathcal{P}$. Then, $Affects(p) = \{o \in \mathcal{O} | p \in add(o)$ or $p \in del(o)\}$ and $Affects(Q) = \bigcup_{q \in Q} Affects(q)$.

Definition 3.2 For a given PSN instance $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^- \rangle \rangle$, we define the corresponding *dependency graph* $DG(\Pi)$ as a directed labelled graph $DG(\Pi) = \langle \mathcal{P}, \mathcal{D} \rangle$ with vertex set $\mathcal{P}$ and arc set $\mathcal{D}$ such that for all $p, q \in \mathcal{P}$,

- $\langle p, +, q\rangle \in \mathcal{D}$ iff there exists an operator $o \in \mathit{Affects}(q)$ such that $p \in \mathit{pre}^{+}(o)$
- $\langle p, -, q\rangle \in \mathcal{D}$ iff there exists an operator $o \in \mathit{Affects}(q)$ such that $p \in \mathit{pre}^{-}(o)$.
- $\langle p, \sim, q\rangle \in \mathcal{D}$ iff there exists an operator $o \in \mathcal{O}$ such that $p, q \in \mathit{add}(o) \cup \mathit{del}(o)$ and $p \neq q$.

An example of an dependency graph for some Π with $\mathcal{P} = \{A, \ldots, I\}$ can be found in Figure 1. For example, we can see that there exists some operator affecting both A and B and that I is not dependent of the other atoms in any way. We continue by defining three classes of atoms, namely *static*, *irreversible* and *reversible* atoms. The intuition behind these classes is that a static atom must not or cannot be added or deleted, an irreversible atom can be added or deleted but not both and a reversible atom can be both added and deleted.

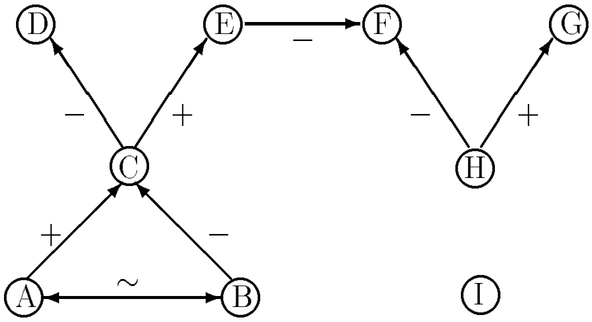

Figure 1: An example dependency graph.

Definition 3.3 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^-\rangle\rangle$ be a PSN instance and let $p \in \mathcal{P}$. Then, p is *static in* Π iff (1) $p \notin s_0$ and there does not exist an $o \in \mathcal{O}$ such that $p \in \mathit{add}(o)$ or (2) $p \in s_0$ and there does not exist an $o \in \mathcal{O}$ such that $p \in \mathit{del}(o)$ or (3) $p \notin s_0$ and $p \in s_*^-$ and there does not exist an $o \in \mathcal{O}$ such that $p \in \mathit{del}(o)$ or (4) $p \in s_0$ and $p \in s_*^+$ and there does not exist an $o \in \mathcal{O}$ such that $p \in \mathit{add}(o)$.

Case (1) says that there does not exist any operator that can add p and since $p \notin s_0$, p cannot occur in any state that any plan solving Π might achieve. Case (2) is analogous to case (1). Case (3) says that if we add p, then we cannot delete it again. But since $p \in s_*^-$ we must delete p if we have added it. Hence, p cannot be added by any plan solving Π. Case (4) is analogous to case (3). So, if an atom p is static in Π, then no plan solving Π can add or delete p.

Definition 3.4 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^-\rangle\rangle$ be a PSN instance. An atom $p \in \mathcal{P}$ is *reversible in* Π iff for all $o \in \mathcal{O}$, whenever $p \in \mathit{add}(o)$ then there exists an $o' \in \mathcal{O}$ such that $p \in \mathit{del}(o)$ and vice versa. Moreover, p is *symmetrically reversible in* Π iff p is reversible and for all $o \in \mathcal{O}$, whenever $p \in \mathit{add}(o)$ then there exists an $o' \in \mathcal{O}$ such that $p \in \mathit{del}(o)$, $\mathit{pre}^{+}(o) = \mathit{pre}^{+}(o')$ and $\mathit{pre}^{-}(o) = \mathit{pre}^{-}(o')$.

If an atom p is reversible in Π, then plans that solve Π can contain both operators adding p and operators deleteing p. If an atom is symmetrically reversible, then we can always delete it under the same conditions as we can add it and vice versa.

Definition 3.5 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^-\rangle\rangle$ be a PSN instance. An atom $p \in \mathcal{P}$ is *irreversible in* Π iff it is not static in Π and not reversible in Π.

If an atom p is irreversible in Π, then plans that solve Π must contain an operator that adds or deletes p, but not both.

Definition 3.6 Let $G = \langle V, E\rangle$ be a directed labelled graph and $G' = \langle V, E'\rangle$ be its undirected counterpart, that is, let $E' = \{(v, x, w), (w, x, v) | (v, x, w) \in E\}$. Then, for $v, w \in V$, w is *weakly reachable* from v iff there exists a path from v to w in $G' = \langle V, E'\rangle$.

Definition 3.7 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^-\rangle\rangle$ be a PSN instance, $DG(\Pi) = \langle V, E\rangle$ and let $p \in \mathcal{P}$. Furthermore, let $Q_+^p = \{q|(p, +, q) \in E\}$, $Q_-^p = \{q|(p, -, q) \in E\}$, $DG_+^p(\Pi) = \langle V, E - \{(p, +, x) \in E | x \in V\}\rangle$ and $DG_-^p(\Pi) = \langle V, E - \{(p, -, x) \in E | x \in V\}\rangle$. Then, we can divide $\mathcal{P}$ into the following three sets:

1. $\mathcal{P}_+^p = Q_+^p \cup \{q|q$ is weakly reachable from some $r \in Q_+^p$ in $DG_+^p(\Pi)\}$.
2. $\mathcal{P}_-^p = Q_-^p \cup \{q|q$ is weakly reachable from some $r \in Q_-^p$ in $DG_-^p(\Pi)\}$.
3. $\mathcal{P}_0^p = \{q \in \Pi' | q$ is not weakly reachable from p in $DG(\Pi)\}$.

Consider the dependency graph in Figure 1 and the vertex C. We can see that $Q_+^C = \{E\}$ and consequently, $\mathcal{P}_+^C = \{E\} \cup \{F, G, H\} = \{E, F, G, H\}$. Analogously, $Q_-^C = \{D\}$ and $\mathcal{P}_-^C = \{D\} \cup \varnothing = \{D\}$. Also note that $\mathcal{P}_0^C = \{I\}$. Obviously, $\mathcal{P}_+^p \cap \mathcal{P}_0^p = \varnothing$ and $\mathcal{P}_-^p \cap \mathcal{P}_0^p = \varnothing$ for all choices of p but, in the general case, $\mathcal{P}_+^p$ and $\mathcal{P}_-^p$ are not disjoint. This observation leads to the next definition:

Definition 3.8 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^-\rangle\rangle$ be a PSN instance. An atom $p \in \mathcal{P}$ is *splitting in* Π iff $\mathcal{P}_+^p$ and $\mathcal{P}_-^p$ are disjoint.

The atom C in Figure 1 is a splitting atom because $\mathcal{P}_+^C \cap \mathcal{P}_-^C = \{E, F, G, H\} \cap \{D\} = \varnothing$. Another example is the atom F where both $\mathcal{P}_+^F$ and $\mathcal{P}_-^F$ equal $\varnothing$. Intuitively, if an atom p is splitting then the problem instance can be split into three subproblems which can be solved independently and the sets $\mathcal{P}_+^p, \mathcal{P}_-^p$ and $\mathcal{P}_0^p$ tells us which atoms that belongs to which subproblem. As a convention, we usually drop "in Π" when dealing with types of atoms if Π is clear from the context. We can now define the 3S class of planning problems.

Definition 3.9 3S is the set of PSN instances having acyclic dependency graphs and where every atom is static, symmetrically reversible or splitting.

Note that if $DG(\Pi)$ is acyclic for some PSN instance $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^-\rangle\rangle$, then Π is *unary*, that is, $|add(o) \cup del(o)| = 1$ for every $o \in \mathcal{O}$. Hence, every 3S instance is unary. It should also be noted that the restrictions on the atomic level are not orthogonal. For example, an atom can be splitting and symmetrically reversible at the same time.

4 Polynomial-time Plan Existence

In this section, we show that the plan existence problem for instances in 3S is polynomial while the plan generation problem is provably intractable. However, we begin by defining some concepts that will facilitate the forthcoming proofs.

Definition 4.1 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^-\rangle\rangle$ be a PSN instance, p be a member of $\mathcal{P}$ and let $s, s^+, s^- \subseteq \mathcal{P}$. Then s is *compatible* with $\langle s^+, s^-\rangle$ wrt p iff (1) $p \in s$ and $p \notin s^-$ or (2) $p \notin s$ and $p \notin s^+$.

Loosely speaking, an initial state s_0 is compatible with a goal state $\langle s_*^+, s_*^- \rangle$ wrt. p if we do not have to add or delete p in order to satisfy the goal. We continue by defining a function $\Cap$ for restricting operators and planning instances to limited sets of atoms. We also define a function for recreating operators that have been restricted by $\Cap$.

Definition 4.2 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^- \rangle \rangle$ be a PSN instance, $o \in \mathcal{O}$ and $\mathcal{P}' \subseteq \mathcal{P}$. Then, $o \Cap \mathcal{P}'$ (the *restriction of o to $\mathcal{P}'$*) is the operator $\langle add(o) \cap \mathcal{P}', del(o) \cap \mathcal{P}', pre^+(o) \cap \mathcal{P}', pre^-(o) \cap \mathcal{P}', name(o) \rangle$. We define $\Cap$ for a set $\mathcal{O}' \subseteq \mathcal{O}$ of operators in the following way: $\mathcal{O}' \Cap \mathcal{P}' = \{o \Cap \mathcal{P}' | o \in \mathcal{O}'\}$. Finally, we define $\Cap$ for a PSN problem instance Π such that $\Pi \Cap \mathcal{P}' = \langle \mathcal{P}', \mathcal{O} \Cap \mathcal{P}', s_0 \cap \mathcal{P}', \langle s_*^+ \cap \mathcal{P}', s_*^- \cap \mathcal{P}' \rangle \rangle$

Definition 4.3 Let o be an operator and $\mathcal{O}$ a set of operators. Then, $\downarrow^{\mathcal{O}}(o)$ is defined as the unique operator $o' \in \mathcal{O}$ such that $name(o) = name(o')$. We generalize $\downarrow^{\mathcal{O}}$ to operate on plans in the obvious way, namely $\downarrow^{\mathcal{O}}(\langle o_1, \ldots, o_n \rangle) = \langle \downarrow^{\mathcal{O}}(o_1), \ldots, \downarrow^{\mathcal{O}}(o_n) \rangle$.

Observe that the previous definition is sound since we have assumed that every operator has a unique name in every operator set. In the next definition, we provide a method for removing certain operators from a planning instance.

Definition 4.4 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^- \rangle \rangle$ be a PSN instance, $\mathcal{O}' \subseteq \mathcal{O}$ and $p \in \mathcal{P}$. Then, $R^+(p, \mathcal{O}') = \{o \in \mathcal{O}' | p \notin pre^+(o)\}$ and $R^-(p, \mathcal{O}') = \{o \in \mathcal{O}' | p \notin pre^-(o)\}$. We also define R^+ and R^- for PSN problem instances the obvious way; namely $R^+(p, \Pi) = \langle \mathcal{P}, R^+(p, \mathcal{O}), s_0, \langle s_*^+, s_*^- \rangle \rangle$ and $R^-(p, \Pi) = \langle \mathcal{P}, R^-(p, \mathcal{O}), s_0, \langle s_*^+, s_*^- \rangle \rangle$.

We can view $R^+(p, \Pi)$ as the problem instance Π with all operators o such that $p \in pre^+(o)$ is removed and $R^-(p, \Pi)$ as Π with all operators o such that $p \in pre^-(o)$ is removed. Finally, we define a well-known graph-theoretic concept.

Definition 4.5 Let $G = \langle V, E \rangle$ be a directed (labelled) graph. A vertex $v \in V$ is *minimal* iff there does not exist any $e \in E$ ending in v.

We claim that the PE-3S algorithm which is presented in Figure 2 solves the plan existence problem in polynomial time for problem instances in 3S. To prove the claim, we need the following three lemmata.

Lemma 4.6 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^- \rangle \rangle \in$ 3S and let $\mathcal{P}' = \mathcal{P} - \{p\}$ for some arbitrary $p \in \mathcal{P}$. Then, $\Pi \Cap \mathcal{P}', R^+(p, \Pi) \Cap \mathcal{P}', R^-(p, \Pi) \Cap \mathcal{P}' \in$ 3S.

```
function PE-3S(Π) : boolean (* Π = ⟨P, O, s_0, ⟨s_*^+, s_*^-⟩⟩ *)
if P = ∅ then return true
else
    choose an atom p that is minimal in DG(Π)
    if p is static then
        if s_0 is not compatible with ⟨s_*^+, s_*^-⟩ wrt. p
         then return false
        elsif p ∉ s_0 then return PE-3S(R^+(p, Π) ⋒ (P − {p}))
        else return PE-3S(R^-(p, Π) ⋒ (P − {p}))
    else return PE-3S(Π ⋒ (P − {p}))
```

Figure 2: The PE-3S algorithm.

Proof: If $\Pi' = \Pi \Cap \mathcal{P}'$, then $DG(\Pi')$ is acyclic and every atom in Π' is either static, symmetrically reversible or splitting since $\Pi \in$ 3S. Assume $\Pi' = R^+(p, \Pi) \Cap \mathcal{P}'$. $DG(\Pi')$ is acyclic since $DG(\Pi)$ is acyclic. Choose an arbitrary $q \in \mathcal{P}'$. If q is static, then q is static in Π' because $|\mathcal{O}'| \leq |\mathcal{O}|$. If q is symmetrically reversible, then q is either reversible or static in Π'. This follows from the fact that if some add operator that affects p is removed by R^+, then the corresponding delete operator is removed as well. Finally, if q is splitting, then q is still splitting in Π' because $|\mathcal{O}'| \leq |\mathcal{O}|$. The case when $\Pi' = R^-(p, \Pi \Cap \mathcal{P}')$ is analogous. □

Lemma 4.7 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^- \rangle \rangle \in$ 3S. Then, PE-3S(Π) returns true if Π has a solution.

Proof: Suppose there exists a plan ω that solves Π, but PE-3S(Π) returns false. We show that this is impossible by induction over $|\mathcal{P}|$:
Basis step: $|\mathcal{P}| = 0$. PE-3S(Π) returns true by definition.
Induction hypothesis: Suppose the lemma holds for $|\mathcal{P}| \leq k$, $k \geq 0$.
Induction step: We want to show that the lemma holds for $|\mathcal{P}| = k + 1$. We have four cases:

1. PE-3S returns false in line 7. Obviously, Π does not have any solution. Contradiction.

2. PE-3S returns false in line 8. Since we cannot add p, we must check if $R^+(p, \Pi) \Cap (\mathcal{P} - \{p\})$ has any solution. By Lemma 4.6, $R^+(p, \Pi) \Cap (\mathcal{P} - \{p\}) \in$ 3S so, by the induction hypothesis, we can do this with the PE-3S procedure. Hence, if PE-3S($R^+(p, \Pi) \Cap (\mathcal{P} - \{p\})$) returns false, Π does not have any solution. Contradiction.

3. PE-3S returns false in line 9. Analogous to the previous case.

4. PE-3S returns false in line 10. By Lemma 4.6, $\Pi \Cap (\mathcal{P} - \{p\}) \in$ 3S. Hence, by the induction hypothesis, we can check whether $\Pi \Cap (\mathcal{P} - \{p\})$ has a solution or not with PE-3S. If $\Pi \Cap (\mathcal{P} - \{p\})$ has no solution, then Π has no solution. Contradiction. □

Lemma 4.8 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^- \rangle \rangle \in$ 3S Then, Π has a solution if PE-3S(Π) returns true.

Proof: Assume PE-3S(Π) returns true. We show the lemma by induction over $|\mathcal{P}|$:
Basis step: If $|\mathcal{P}| = 0$, then PE-3S(Π) returns true in line 2. Obviously, $\langle\rangle$ is a valid plan for Π so the lemma holds in this case.
Induction hypothesis: Suppose the lemma holds for $|\mathcal{P}| \leq k$, $k \geq 0$.
Induction step: We want to show that the lemma holds for $|\mathcal{P}| = k + 1$. Let p be the minimal atom in $DG(\Pi)$ that PE-3S chooses in line 7 and let $\mathcal{P}' = \mathcal{P} - \{p\}$. (Note that the algorithm always can choose such a p since $DG(\Pi)$ is acyclic). We have three cases:

1. p is static. Since PE-3S(Π) returns true, s_0 is compatible with $\langle s_*^+, s_*^- \rangle$ wrt. p. Hence, PE-3S must return true in line 10 or 11. Both $R^+(p, \Pi) \Cap \mathcal{P}'$ and $R^-(p, \Pi) \Cap \mathcal{P}'$ are members of 3S by Lemma 4.6. So, by the induction hypothesis, there exists a valid plan ω for $R^+(p, \Pi) \Cap \mathcal{P}'$ or $R^-(p, \Pi) \Cap \mathcal{P}'$. Since p is static, $\downarrow^{\mathcal{O}}(\omega)$ is a valid plan for Π.

2. p is reversible. We know that p is not static so PE-3S must return true in line 13. Since p is minimal in $DG(\Pi)$, there exists operators o^+, o^- that adds p and deletes p having no preconditions at all. Hence, we can add and delete p freely. By Lemma 4.6, $\Pi \Cap \mathcal{P}' \in$ 3S so by the induction hypothesis, there exists a valid plan ω for $\Pi \Cap \mathcal{P}'$. Consequently, there exists a plan ω' for Π. (Simply by inserting o^+ before every operator in $\downarrow^{\mathcal{O}}(\omega)$ that needs p to be true and inserting o^- before every operator in $\downarrow^{\mathcal{O}}(\omega)$ that needs p to be false. Possibly, we also have to insert some operator last in the plan to ensure that the goal state is satisfied.)

3. p is irreversible. We begin by showing that p is splitting. Assume p is not splitting. Since $\Pi \in$ 3S and p is not static, p must be symmetrically reversible. Then p is not irreversible, so p is splitting. Consequently, PE-3S must return true in line 13. By Lemma 4.6, $\Pi' = \Pi \Cap \mathcal{P}' \in$ 3S and by the induction hypothesis, there exists a plan ω that solves Π'. Since p is splitting, $\mathcal{P}^p_+, \mathcal{P}^p_-$ and $\mathcal{P}^p_0$ are disjoint. Form the following three subinstances: $\Pi'_+ = \Pi \Cap \mathcal{P}^p_+, \Pi'_- = \Pi \Cap \mathcal{P}^p_-, \Pi'_0 = \Pi \Cap \mathcal{P}^p_0$. Assume that $p \in s_0$. As we know that p is not static, there exists an operator o^- that deletes p. Furthermore, we know that $\mathcal{P}^p_+, \mathcal{P}^p_-$ and $\mathcal{P}^p_0$ are disjoint, so we can reorder ω to the plan $\omega' = (\omega_+; \omega_-; \omega_0)$ where ω_+ solves Π'_+, ω_- solves Π'_- and ω_0 solves Π'_0. As a consequence, $\omega'' = (\downarrow^{\mathcal{O}}(\omega_+); o^-; \downarrow^{\mathcal{O}}(\omega_-); \downarrow^{\mathcal{O}}(\omega_0))$ is a valid plan solving Π. The case when $p \notin s_0$ is analogous. □

We are now able to prove that the plan existence problem for instances in 3S is polynomial.

Theorem 4.9 Let $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s^+_*, s^-_* \rangle \rangle \in$ 3S. Then, whether Π has a solution or not can be decided in polynomial time.

Proof: The recursion depth of PE-3S is bounded above by $|\mathcal{P}|$ since the number of atoms decreases strictly for each recursive level. Hence, PE-3S must eventually return true or false. By Lemmata 4.7 and 4.8, Π has a solution iff PE-3S(Π) returns true. Conversely, Π lacks a solution iff PE-3S(Π) is false. It remains to show that PE-3S runs in polynomial time for every Π. Constructing $DG(\Pi)$ and performing the different tests takes only polynomial time. We have already shown that we PE-3S will make less than $|\mathcal{P}|$ recursive calls. Consequently, PE-3S runs in polynomial time. □

Now, we turn our attention to the complexity of plan generation for instances in 3S. In the next theorem, we show that there exists instances having exponentially sized minimal solution in 3S.

Theorem 4.10 For all $n > 0$, there is some instance $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s^+_*, s^-_* \rangle \rangle \in$ 3S such that $|\mathcal{P}| = n$ and all minimal plans solving Π are of length $2^n - 1$.

Proof sketch: The SAS-PUB instance in Bäckström & Nebel [Bäckström and Nebel, 1993] which have exponentially sized minimal solution can trivially be converted into a 3S instance with the desired properties. □

So, plan existence is polynomial for instances in 3S while plan generation takes exponential time in the worst case.

5 Incremental Planning

Since Lemma 4.8 is constructive, it allows us to devise a planner that generates a solution plan whenever one exists. By exploiting our knowledge of the structure of the plan, we can even construct a *incremental* planner, *ie.* a planner that attempts outputting an executable prefix of the final solution plan before the whole of this solution is completed. This algorithm, IP-3S, appears in Figure 3. To be able to output prefixes before the whole solution is computed, we use *streams*. The primitive *output* puts one or more elements on the stream and the function *read* returns the first element of the stream and removes it.

The ability of IP-3S to produce executable prefixes stems from two facts: (1) if an atom p is splitting, then the problem can be divided into three subproblems which can be solved independently and (2) *Interweave* does not have to wait for its incoming stream to be closed before it can begin putting operators on its output stream. To take full advantage of the prefix-generation, a certain amount of parallelism is needed. We do not want to wait until a recursive call in line 15 of IP-3S is completed before we begin to process the output on the stream from this call. Instead, we want the recursive call to be a new process that executes concurrently with the process that called it.

It should be noted that $\downarrow$ has to be redefined in a straight-forward way to operate on streams in order to make IP-3S work correctly. The IP-3S algorithm clearly follows the cases in the proof of Lemma 4.8 so the following theorem follows immediately.

Theorem 5.1 Let Π is a soluble instance of 3S, then, IP-3S will generate a plan ω solving Π

Since we can check if $\Pi \in$ 3S has a solution or not in polynomial time, it is not very restrictive that IP-3S requires Π to have a solution in order to work; It would, in fact, be very disappointing if IP-3S generated a large prefix (which we perhaps would start to execute) and then suddenly told us that no solution exists for the instance. We continue with defining a complexity concept for capturing the time complexity of IP-3S.

Definition 5.2 An algorithm runs in *solution-polynomial* time iff its running time is bounded above by some polynomial in the size of the input and in the size of the generated solution.

This concept is similar to the concept *total polynomial time* [Johnson *et al.*, 1988]. By Theorem 4.10, IP-3S cannot run in polynomial time. However, it runs in solution-polynomial time, which is probably as good as we can hope for when dealing with problems having exponentially sized solutions.

Theorem 5.3 IP-3S runs in solution-polynomial time.

Proof sketch: Suppose we want to compute IP-3S(Π) for some arbitrary $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^- \rangle\rangle \in$ 3S and assume that the resulting plan ω has length L. Then, IP-3S will perform less than $|\mathcal{P}|$ recursive calls on non-empty subinstances of the original instance at most. This is trivial if the chosen p is static or reversible and follows from the fact that $\mathcal{P}_+^p$, $\mathcal{P}_-^p$ and $\mathcal{P}_0^p$ are disjoint otherwise. We can over-estimate the consumed time by assuming that every recursive calls works on a plan of length L. The construction of $DG(\Pi)$, finding a minimal p, the different tests and the $\downarrow$ and *Interweave* functions are all bounded by some polynomial p in $|\mathcal{P}|$ and L. Hence, the running time

function IP-3S(Π) : stream (* $\Pi = \langle \mathcal{P}, \mathcal{O}, s_0, \langle s_*^+, s_*^- \rangle \rangle$ *)
if $\mathcal{P} \neq \varnothing$ **then**
    choose an atom p that is minimal in $DG(\Pi)$
    if p is static **then**
        if $p \notin s_0$ **then** $output(\downarrow^{\mathcal{O}}(\text{IP-3S}(R^+(p, \Pi) \Cap (\mathcal{P} - \{p\}))))$
        else $output(\downarrow^{\mathcal{O}}(\text{IP-3S}(R^-(p, \Pi) \Cap (\mathcal{P} - \{p\}))))$
    elsif p is irreversible **then**
        if $p \notin s_0$ **then**
            $output(\downarrow^{\mathcal{O}}(\text{IP-3S}(\Pi \Cap \mathcal{P}_-^p)))$
            $output(o)$ where $o \in \mathcal{O}$ adds p
            $output(\downarrow^{\mathcal{O}}(\text{IP-3S}(\Pi \Cap \mathcal{P}_+^p)))$
            $output(\downarrow^{\mathcal{O}}(\text{IP-3S}(\Pi \Cap \mathcal{P}_0^p)))$
        else the case when $p \in s_0$ is analogous
    else (* p is reversible *)
        $output(Interweave(\downarrow^{\mathcal{O}}(\text{IP-3S}(\Pi \Cap (\mathcal{P} - \{p\}))), p, \mathcal{O}, s_0, \langle s_*^+, s_*^- \rangle))$

function *Interweave*(ω : *stream*, $p, \mathcal{O}, s_0, s_*^+, s_*^-$) : stream
if $p \in s_0$ **then** $added \leftarrow T$
else $added \leftarrow F$
let $o^+, o^- \in \mathcal{O}$ be such that $p \in add(o^+)$ and $p \in del(o^-)$
while ω is not closed **do**
    $o \leftarrow read(\omega)$
    if $p \in pre^+(o)$ **and not** $added$ **then** $output(o^+)$; $added \leftarrow T$
    elsif $p \in pre^-(o)$ **and** $added$ **then** $output(o^-)$; $added \leftarrow F$
    $output(o)$
if $p \in s_*^+$ **and not** $added$ **then** $output(o^+)$
elsif $p \in s_*^-$ **and** $added$ **then** $output(o^-)$

Figure 3: The IP-3S algorithm.

is $L \cdot p(|\mathcal{P}|, L)$ at most and the theorem follows. □

It is important to notice that IP-3S is polynomial in the length of the *generated* plan, not in the shortest possible plan. Hence, it is possible that IP-3S can take exponential time when solving an instance Π though it is possible to solve Π in polynomial time with some other algorithm.

6 Discussion

We have presented a class of planning problems where we can tell in advance, in polynomial time, whether a solution exists or not. We have also presented a provably correct planner for this class that runs in polynomial time in the length of the solution it produces. Furthermore, this planner will, whenever possible, output successive valid prefixes of the final solution for immediate execution, concurrently with the continuing planning process.

This research continues as well as complements our ongoing research into tractable planning, using syntactic restrictions [Bäckström and Nebel, 1993] as well as structural ones [Jonsson and Bäckström, 1994b]. Incremental planning seems to provide one way of tackling non-tractable classes of planning problems, while also making replanning feasible. The variable-graph approach is an obvious continuation of the research into structural restrictions. Interestingly, these graphs can be viewed as a generalization of the dependency graphs Knoblock [Knoblock, 1994] uses for generating abstraction hierarchies, where our graphs contain more information.

We have earlier argued [Bäckström and Nebel, 1993] that planning problems allowing exponential-size optimal solutions should be considered unrealistic.[1] This does not imply that the 3S class is unrealistic, however. It is important to distinguish between the inherent complexity of an application problem and the complexity of the hardest problems allowed by planning formalism *per se.* The only natural examples of problems with exponential-size optimal solutions seem to be artificial puzzles, like Towers of Hanoi, which are deliberatly designed to have this property. Application problems arising 'naturally' in industry *etc.* can be expected not to exhibit this property. In other words, we can expect real problems fitting within the 3S class to have reasonably sized solutions. (Indeed, if we can be sure that all solvable instances we feed the planner have polynomially bounded solutions, then we can actually solve plan generation in polynomial time.) Note, however, that this would not be of much help to us if the 3S class did not allow tractable plan existence, since we would then still face the intractability for the unsolvable instances—not being able to tell in advance that these are unsolvable.

An interesting theoretical consequence of the 3S class is that the relevance of deriving complexity results for plan existence only is questionable since this need not at all be related to the complexity of our ultimate goal—plan generation. However, the 3S class is not only of theoretical interest; it is, in fact, expressive enough for modelling the LEGO car factory, a miniature assembly line for LEGO cars, which is used for undergraduate laborations at the department of EE at Linköping University. This assembly line is in many respects a realistic miniature version of real industrial processes and is described in Klein [Klein *et al.*, 1995].

One problem with the 3S algorithm is that although it runs in polynomial time in the length of the solution it generates, it is not guaranteed to generate an optimal plan. In fact, in the worst case it could happen to generate an exponential plan when there is a short, optimal one. Although we can hope for this not to happen in practice, it seems hard to rule out the possibility by any simple means and this problem arises also for 'standard' general-purpose planners, like TWEAK. However, while such planners can avoid the problem through backtracking, although at a considerably higher cost, this may not be possible if we want to generate prefixes greedily. This problem is not unique for incremental planning, however. An analogous problem arises in state abstraction, where the wrong choice of abstraction hierarchy can force the hierarchical planner to spend exponentially longer time generating an exponentially longer solution than a non-hierarchical planner [Bäckström and Jonsson, 1995]. For incremental planners, there seems to be a tuning factor between outputting prefixes early and guaranteeing reasonably short plans respectively—an interesting challenge for future research.

Another intimately related topic we are currently studying is to determine bounds on the length of solutions before generating them, which could provide a solution to the problem mentioned above. We further plan to exploit the information in the variable-dependency graphs more heavily and also generalize them to multi-valued state variables, since we believe these graphs to provide a major vehicle for future research into tractable planning.

[1] This is simply a specialization of the 'same' claim for problems in general [Garey and Johnson, 1979].

7 Acknowledgements

We would like to thank Christos Papadimitriou and the anonymous referees for discussions and comments which helped improving the paper. The research was supported by the *Swedish Research Council for Engineering Sciences (TFR)* under grants Dnr. 92-143 and Dnr. 93-00291.

References

[Ambros-Ingerson and Steel, 1987] José A. Ambros-Ingerson and Sam Steel. Integrating planning, execution and monitoring. In *Proceedings of the 7th (US) National Conference on Artificial Intelligence (AAAI-88)*, pages 83–88, St. Paul, MN, USA, August 1987. American Association for Artificial Intelligence, Morgan Kaufmann.

[Bäckström and Jonsson, 1995] Christer Bäckström and Peter Jonsson. Planning with abstraction hierarchies can be exponentially less efficient. In Chris Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montréal, PQ, Canada, August 1995. Morgan Kaufmann. To appear.

[Bäckström and Nebel, 1993] Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1430–1435, Chambéry, France, August–September 1993. Morgan Kaufmann.

[Bäckström, 1995] Christer Bäckström. Expressive equivalence of planning formalisms. *Artificial Intelligence, Special Issue on Planning and Scheduling*, 76:1–18, 1995. ARTINT 1234. In Press.

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[Chapman, 1987] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

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

[Johnson *et al.*, 1988] David S Johnson, Mihalis Yannakakis, and Christos H Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, 1988.

[Jonsson and Bäckström, 1994a] Peter Jonsson and Christer Bäckström. Complexity results for state-variable planning under mixed syntactical and structural restrictions. In Philippe Jorrand, editor, *Proceedings of the 6th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA-94)*, Sofia, Bulgaria, September 1994. World Scientific Publishing.

[Jonsson and Bäckström, 1994b] Peter Jonsson and Christer Bäckström. Tractable planning with state variables by exploiting structural restrictions. In *Proceedings of the 12th (US) National Conference on Artificial Intelligence (AAAI-94)*, pages 998–1003, Seattle, WA, USA, July–August 1994. American Association for Artificial Intelligence.

[Klein *et al.*, 1995] Inger Klein, Peter Jonsson, and Christer Bäckström. Tractable planning for an assembly line. In Malik Ghallab, editor, *Current Trends in AI Planning: EWSP'95—3rd European Workshop on Planning*, Assissi, Italy, September 1995. IOS Press. This volume.

[Knoblock, 1994] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.

Note! Most of the authors' publications are available from:
`ftp.ida.liu.se@pub/labs/rkllab/people/{petej,chrba}`