Planning with Reduced Operator Sets

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

Abstract

Classical propositional STRIPS planning is nothing but the search for a path in the state-transition graph induced by the operators in the planning problem. What makes the problem hard is the size and the sometimes adverse structure of this graph. We conjecture that the search for a plan would be more efficient if there were only a small number of paths from the initial state to the goal state. To verify this conjecture, we define the notion of *reduced* operator sets and describe ways of finding such reduced sets. We demonstrate that some state-of-the-art planners run faster using reduced operator sets.

Keywords: STRIPS planning, preprocessing

Introduction

A major focus of research in AI planning has been on improving the efficiency of planners, in particular STRIPS planners. The STRIPS planning problem is, loosely speaking, characterised by the assumptions that operator effects are deterministic and unconditional, and that the initial state is completely known. Under these assumptions, the planning problem reduces to that of finding a path in the state-transition graph induced by the operators. What makes the problem still hard is the size, and perhaps sometimes adverse structure, of this graph.

However, the use of the STRIPS representation also gives the state-transition graph structure which can be exploited to gain efficiency. The use of representations such as planing graphs (Blum & Furst 1997), of state-of-the-art SAT technology (Kautz & Selman 1996), and of automatic extraction of heuristic information and domain invariants from the problem representation (Bonet, Loerincs, & Geffner 1997; Fox & Long 1998) has meant that STRIPS planners today solve problems that seemed impossible not too long ago.

Based on the view of planning as the search for a path in the state-transition graph, we conjecture that planning will, at least in some instances, be faster if the density of the graph is reduced; ideally, we would like to reduce the graph to a minimal spanning tree (actually, a minimal equivalent digraph). To motivate this, perhaps controversial, conjecture, consider that the complexity of graph search is $O(b^d)$, where b is the branching factor and d the solution depth. For search in a spanning tree of the graph, the complexity is $O(a^c)$, where $a < b$, since the tree has fewer edges, and $c \geqslant d$, since the shortest path to a solution in the graph may not be available in the tree. Clearly there is a gain if the decrease in branching factor is large and the increase in depth small.

Our belief that the state-transition graph of a planning problem is often more dense than necessary comes from the observation that some operators in some domains are, in a certain sense, redundant. Consider, for example, the well known Blocksworld domain, and the operator **move**(**A**,**B**,**C**), meaning "move block **A** from on top of block **B** to block **C**". This operator is redundant, since we can replace any occurence of this operator in a plan by the sequence **move**(**A**,**B**,**Table**), **move**(**A**,**Table**,**C**).

To put this conjecture to the test, we define the notion of operator redundancy and construct a method to find and remove redundant operators or operator instances from a planning problem, creating what we call a *reduced* problem. Though an efficient algorithm for computing at least an approximation of the minimal equivalent digraph exists, it is of no use to us since it works on the graph explicitly. We need methods that can be applied on the level of description at which the problem is given, *i.e.* to the set of operators. The reduction process we use preserves solution existence, *i.e.* the reduced planning problem has a solution if and only if the original problem was solvable. We have implemented the reduction in the form of a preprocessor and compared the runtimes of several planners on reduced planning problems to that on the original problems. In some cases, considerable speed-up was observed.

Related Work

Examples of the use of domain analysis techniques to improve the performance of planners are not rare in the literature. Well known examples are the automatic generation of abstraction hierarchies (Knoblock 1994) and the postponement of threats with the aid of operator graphs (Peot & Smith 1993). The information provided by these methods, however, is useful only to planners of a certain design.

The method suggested by Nebel *et al.* (1997) for removing irrelevant facts and operators from planning problems is not in this way specific to a certain planner or kind of planner. This technique has been shown to yield quite impressive results, although only for planners whose performance is impaired by irrelevant facts. It must also be noted that the method is not solution-preserving.

The "redundant sequence of actions" (RSA) concept defined by Scholz (1999) is very similar to our notion of redundancy, although it is restricted only to sequences of length 2. Scholz, however, makes use of it in a way different from ours, by integrating it as a constraint into the planning mechanism to exclude plans that contain the, longer, redundant sequence.

A number of other transformations on planning problems have been suggested, but aimed to reduce an expressive representation language into a simpler language rather than to increase the performance of the planner. Interesting examples are (Gazen & Knoblock 1997), (Koehler *et al.* 1997) and (Bäckström 1995).

Definition of Reduced Operator Sets

We begin this section by defining the syntax and semantics of the STRIPS planning problem. We avoid the elaborate STRIPS semantics defined by Lifschitz (1986) in favor of an essentially propositional formulation, similar to *e.g.* McAllester & Rosenblitt (1991) and Bylander (1994). We assume complete knowledge of the initial state and complete determinism in the effects of all operators. We also restrict operator preconditions and goals to contain only positive literals, *i.e.* atoms.

Definition 1 (STRIPS Syntax and Semantics)
A STRIPS operator, o, is a triple (P_o, A_o, D_o) where each component is a set of atoms, representing preconditions, added atoms and deleted atoms, respectively. We assume that operators are consistent, *i.e.* that $A_o \cap D_o = \emptyset$.

A state, s, is represented by the set of atoms that are true in the state. We say that an operator o is applicable in s iff $P_o \subseteq s$. The state resulting from applying the operator o in state s is $(s - D_o) \cup A_o$. We also say a sequence of operators, $o_1, \ldots, o_n$, is applicable in s iff o_1 is applicable in s and for each $i > 1$, o_i is applicable in the state resulting from executing $o_1, \ldots, o_{i-1}$.

A planning problem instance is a triple $P = (O, I, G)$, where O is a set of STRIPS operators, I is the set of initially true atoms and G the set of goal atoms. We say a sequence of operators is a plan for P if the sequence is applicable in I and all atoms $g \in G$ hold in the state resulting from executing the sequence.

When the operator set is clear from the context, we abbreviate a problem instance (I, G). We also denote by $D(P)$ the domain of the problem instance, *i.e.* the set of all atoms mentioned in P.

In a planning problem, it is often the case that certain combinations of atoms will never occur together in any reachable state; they are in some sense inconsistent. This property of atom sets will be important in our definition of redundancy. Strictly logically speaking, a set of atoms can of course not be inconsistent. Therefore, the inconsistency relation can not be determined from the domain and operator set alone.

Definition 2 (The Inconsistency Relation)
For a set of atoms A and an atom a, we say that a is *inconsistent* with A if there is no state s reachable from I such that $A \subset s$ and $a \in s$. We denote by $INC(A)$ the set of atoms that are inconsistent with A.

An approximation of $INC(A)$ can be computed using a fixpoint computation similar to that of Bonet and Geffner (1999). It is approximate in the sense that we compute a subset, possibly equal, of $INC(A)$. This approximation is sound, but not always complete.

The next two definitions provide the foundations for our definition of the concept of operator redundancy.

Definition 3 (Cumulative Effects)
We define the *cumulative* preconditions, positive and negative effects of a sequence of STRIPS operators, denoted $C_P(\cdot)$, $C_A(\cdot)$ and $C_D(\cdot)$ respectively, inductively as

$$\begin{array}{l} C_P(o) = P_o \\ C_A(o) = A_o \\ C_D(o) = D_o \end{array} \qquad (i)$$

and

$$\begin{array}{l} C_P(o_1, \ldots, o_n, o) = \\ \quad C_P(o_1, \ldots, o_n) \cup (P_o - C_A(o_1, \ldots, o_n)) \\ C_A(o_1, \ldots, o_n, o) = \\ \quad (C_A(o_1, \ldots, o_n) - D_o) \cup A_o \\ C_D(o_1, \ldots, o_n, o) = \\ \quad (C_D(o_1, \ldots, o_n) - A_o) \cup D_o \end{array} \qquad (ii)$$

iff $C_D(o_1, \ldots, o_n) \cap P_o = \emptyset$, and undefined otherwise.

The cumulative preconditions and effects of a sequence of operators summarise the conditions under which the sequence will be applicable and the effects of applying it, respectively, much in the same way as the preconditions and effects of a single operator do. Therefore, it provides a natural way of defining precisely when an operator may be replaced by a sequence of alternative operators, *i.e.* when the operator is redundant.

Definition 4 (The Implements Relation)
We say a sequence of operators $o_1, \ldots, o_n$ *implements* an operator o, iff

(i) o does not occur in the sequence, (*ii*) $C_P(o_1, \ldots, o_n) \subseteq P_o$,

(*iii*) $A_o \subseteq C_A(o_1, \ldots, o_n)$, and

(*iv*) $C_D(o_1, \ldots, o_n) \subseteq (D_o \cup INC(P_o))$.

Definition 5 (Redundant, Reduced)
For a set of operators O and an operator $o \in O$, we say that o is *redundant* with respect to O iff there exists some sequence of operators in $O - \{o\}$ which implements o.

We call the set O *reduced* iff there are no redundant operators in O.

Items (*i*)-(*iii*) of definition 4 are rather straightforward. Since the intention is to eliminate the operator o by substituting for it the sequence $o_1, \ldots, o_n$, we have to require that o is not part of the sequence, that the sequence is applicable whenever o is, and that it makes true at least the same atoms as does o. Finally, we have to ensure that the substituted sequence does not delete anything that would not have been deleted by o. The natural way to state this requirement, $C_D(o_1, \ldots, o_n) \subseteq D_o$, turns out to be to restrictive. To capture the intuitive meaning of operator redundancy, we have also to allow that the sequence $o_1, \ldots, o_n$ deletes some atoms that o does not, as long as those atoms are such that can not be true in any state where o is applicable.

Example 1 Consider the Blocksworld domain, and the operator **move**(**A**,**B**,**C**), meaning "move block **A** from on top of block **B** to block **C**". We would expect this operator to be redundant, since we can replace any occurence of this operator in a plan by the sequence **move**(**A**,**B**,**Table**), **move**(**A**,**Table**,**C**). The cumulative effects of this sequence, however, include the deletion of the atom **on_table**(**A**), which is not deleted by the operator **move**(**A**,**B**,**C**). But, for any normal initial state, this atom is inconsistent with **on**(**A**,**B**), which is a precondition of the replaced operator.

In principle, there is no reason why we should not make similar exceptions in items (*ii*) and (*iii*). We could allow the implementing sequence $o_1, \ldots, o_n$ to require additional precondition atoms, or to fail to add some of the atoms added by the operator o, as long as those atoms were "entailed" by the preconditions of o and the set of atoms added by o (or required and not deleted), respectively. The reasons for not doing so are two: First, there appears to be no easy way to compute this entailment relation between atoms, and second, judging from our experiments it does not seem necessary.

We do, however, use a simple analysis to detect static atoms (atoms that can never change truth value from their value in the initial state) and remove not only operators whose preconditions are statically false but also statically true atoms from the preconditions of remaining operators. Although this analysis is incomplete, it can help in finding redundant operators.

Example 2 In the Grid domain there is a robot which can move between adjacent nodes, arranged in a grid, provided the node moved to is not locked (denoted by the atom **open**(n)). A node which is initially locked can later become unlocked, but a node which is initially unlocked can never become locked.

Because the statically true **open**(n) preconditions are removed, some instances of the **move** operator can be implemented by a "semi-circular" sequence of three move operators. For example, **move(n0-0,n0-1)** could be implemented by the sequence **move(n0-0,n1-0)**, **move(n1-0,n1-1)**, **move(n1-1,n0-1)**), provided **open**(**n1-0**) and **open**(**n1-1**) are statically true.

Correctness and Complexity

In this section, we show that reducing an operator set preserves solution existence, *i.e.* it does not render any planning problem that was previously solvable insolvable. We also show that in general, the problem of computing a reduced set of operators is very hard. In the next section, we describe the more efficient approximate methods that we have used in our experiments.

Lemma 1
Let $o_1, \ldots, o_n$ be a sequence of operators such that the cumulative preconditions and effects are defined, and let s be a state such that

$$C_P(o_1, \ldots, o_n) \subseteq s. \qquad (i)$$

Then the operator sequence is applicable in s and the resulting state is

$$s' = (s - C_D(o_1, \ldots, o_n)) \cup C_A(o_1, \ldots, o_n). \qquad (ii)$$

Proof: By induction on n, the length of the sequence. For $n = 1$, the cumulative preconditions and effects of the sequence are the preconditions and effects of its single operator, and the claims follow immediately from definition 1.

Assuming the lemma holds for $n \leqslant k$, consider a sequence $o_1, \ldots, o_k, o_{k+1}$. From definition 3(*ii*), we see that the cumulative preconditions of an operator sequence are non-decresing w.r.t. appending an operator, *i.e.*

$$C_P(o_1, \ldots, o_k) \subseteq C_P(o_1, \ldots, o_k, o_{k+1}) \qquad (iii)$$

This is because $C_P(o_1, \ldots, o_k, o_{k+1})$ is the union of the cumulative preconditions of the sequence $o_1, \ldots, o_k$ and a, possibly empty, subset of the preconditions of the appended operator, o_k. Therefore, it follows from (*i*) that

$$C_P(o_1, \ldots, o_k) \subseteq s \qquad (iv)$$

which, by the induction assumption, implies that the sequence $o_1, \ldots, o_k$ is applicable in s, and that the resulting state is

$$s'' = (s - C_D(o_1, \ldots, o_k)) \cup C_A(o_1, \ldots, o_k) \qquad (v)$$

Since the cumulative preconditions of $o_1, \ldots, o_k, o_{k+1}$ are defined, we must have that $C_D(o_1, \ldots, o_k) \cap P_{o_{k+1}} =$

$\emptyset$; thus, for any atom $a \in P_{o_{k+1}}$, $a \in s''$ if $a \in s$ or $a \in C_A(o_1, \ldots, o_k)$. If $a \notin C_A(o_1, \ldots, o_k)$, then $a \in C_P(o_1, \ldots, o_k, o_{k+1})$, by definition 3(*ii*), and therefore $a \in s$ by (*i*) above. Consequently, $P_{o_{k+1}} \subseteq s''$, which shows that the entire sequence $o_1, \ldots, o_k, o_{k+1}$ is applicable in s. Next, note that

$$\begin{aligned} s' = & \; (((s - C_D(o_1, \ldots, o_k)) \cup C_A(o_1, \ldots, o_k)) \\ & \; -D_{o_{k+1}}) \cup A_{o_{k+1}} \\ = & \; ((s - (C_D(o_1, \ldots, o_k) \cup D_{o_{k+1}})) \cup \\ & \; (C_A(o_1, \ldots, o_k) - D_{o_{k+1}})) \cup A_{o_{k+1}} \\ = & \; (s - ((C_D(o_1, \ldots, o_k) - A_{o_{k+1}}) \cup D_{o_{k+1}})) \cup \\ & \; ((C_A(o_1, \ldots, o_k) - D_{o_{k+1}}) \cup A_{o_{k+1}}) \\ = & \; (s - C_D(o_1, \ldots, o_k, o_{k+1})) \cup \\ & \; C_A(o_1, \ldots, o_k, o_{k+1}) \end{aligned}$$

which proves the identity (*ii*) above. □

Lemma 2
Let $P = (O, I, G)$ be a planning problem instance, $o \in O$ a redundant operator and $o_1, \ldots, o_n$ a sequence of operators that implements o. Then, in any state s reachable from I,

(*i*) if o is applicable in s, then $o_1, \ldots, o_n$ is applicable in s, and

(*ii*) if s' is the result of applying o in s and s'' the result of applying $o_1, \ldots, o_n$ in s, then $s' \subseteq s''$.

Proof: Since, by definition of the implements relation, $C_P(o_1, \ldots, o_n) \subseteq P_o$, claim (*i*) follows by Lemma 1. Also by Lemma 1, applying $o_1, \ldots, o_n$ in s results in

$$s'' = (s - C_D(o_1, \ldots, o_n)) \cup C_A(o_1, \ldots, o_n) \qquad (iii)$$

Consider an atom $a \in s' = (s - D_o) \cup A_o$. We then have that $a \in s''$ iff either $a \in s$ and $a \notin C_D(o_1, \ldots, o_n)$ or $a \in C_A(o_1, \ldots, o_n)$. In the first case, a can not be in $INC(P_o)$, since because o is applicable in s we must have $P_o \subseteq s$, and thus $a \notin C_D(o_1, \ldots, o_n) \subseteq D_o \cup INC(o)$, which means $a \in s''$. In the second case, we have because $A_o \subseteq C_A(o_1, \ldots, o_n)$ that $a \in C_A(o_1, \ldots, o_n)$ and therefore $a \in s''$. This gives (*ii*). □

Proposition 3
For any planning problem instance, $P = (O, I, G)$, removing a redundant operator from O preserves solution existence.
Proof: Consider a plan $o_1, \ldots, o_m$ for P and suppose operator o_i in this plan is redundant. Since o_i is redundant, there exists by definition a sequence of operators $o'_1, \ldots, o'_n$ which implements o_i.

Because $o_1, \ldots, o_m$ is a plan for P, the sequence must be applicable in I, which also means o_i must be applicable in the state resulting from executing $o_1, \ldots, o_{i-1}$. Therefore, by Lemma 2, $o'_1, \ldots, o'_n$ is also applicable in this state, and in the state resulting from executing the sequence in place of o_i at least the atoms that would have been true after o_i had been executed hold. Since preconditions are only positive, having more atoms true in a state can not make any operator following o_i in the plan inapplicable, and neither can it make any goal atom in the final state false. □

Thus, an operator set O can be transformed to a reduced set O' by enumerating the operators in the set and successively removing any operator that is redundant with respect to the operators currently remaining in the set. The resulting reduced set is, however, not unique; depending on the order of enumeration, different reduced sets can be obtained. Moreover, not all reduced subsets of O are equal in size.

Example 3 Consider the operator set

$$\begin{aligned} O \; = \{ \; & o_1 : \; (\{p\}, \{q\}, \{p\}), \\ & o_2 : \; (\{q\}, \{p\}, \{q\}), \\ & o_3 : \; (\{p\}, \{r\}, \{p\}), \\ & o_4 : \; (\{r\}, \{p\}, \{r\}), \\ & o_5 : \; (\{q\}, \{r\}, \{q\}), \\ & o_6 : \; (\{r\}, \{q\}, \{r\}) \quad \} \end{aligned}$$

A reduced subset of O is $\{o_1, o_2, o_3, o_4\}$ (o_2, o_3 implements o_5 and o_4, o_1 implements o_6). Another is $\{o_1, o_4, o_5\}$ (o_5, o_4 implements o_2, o_1, o_5 implements o_3 and o_4, o_1 implements o_6). Yet another is $\{o_2, o_3, o_6\}$.

Complexity

In the worst case, deciding if an operator, o, is redundant with respect to a set of operators, O, is as hard as solving the general planning problem. To see this, consider any planning problem $P = (O, I, G)$ and create the operator o with

$$\begin{aligned} P_o &= I \\ A_o &= G \\ D_o &= D(P) - G \end{aligned}$$

It is easily seen that a plan for P implements o, and thus that o is redundant with respect to O if and only if P has a solution.

Finding a minimal reduced subset of a set of operators is an even harder problem.

Proposition 4
For a given set of operators O, finding a minimal reduced set $O' \subseteq O$ is NP-hard, even with the assumption that the redundancy of an operator $o \in O$ with respect to some subset of O can be decided in polynomial time.
Proof: By reduction from the minimal equivalent digraph problem, which is stated as follows (Garey & Johnson 1979): Given a directed graph $G = (V, E)$, find the smallest subset $E' \subseteq E$ such that, for every ordered pair of vertices $v, v' \in V$, the graph $G' = (V, E')$ contains a (directed) path from v to v' if and only if G does.

For a given graph, $G = (V, E)$, we create a set of planning operators over a domain equal to V, the vertices of G. For each edge $(v, v') \in E$, we create the operator o, with $P_o = \{v\}$, $A_o = \{v'\}$ and $D_o = \{v\}$. Let O be the set of all such operators. Deciding whether an operator $o \in O$ is redundant can be decided in polynomial time by breadth-first search, since the length of any path in G is no more than V. Because the construction yields a one-to-one correspondence between the operators in O and the edges of G, a minimal reduced set $O' \subset O$ corresponds to minimal a subgraph $G' = (V, E')$ of G satisfying the requirement of path equivalance.

Finally, note that the construction is linear in the size of E. □

Note again that this shows the problem of finding a minimal reduced set to be NP-hard *under the assumption that the redundancy of an operator can be decided in polynomial time.* The problem of deciding redundancy is, as shown above, actually PSPACE-hard, which means that finding a minimal set is certainly harder, though exactly how much harder we can not say.

Computing Reduced Sets

Computing a minimal reduced set of operators breaks down in two problems, both of which were in the preceeding section shown to be hard. Fortunately, we can use simple and efficient approximations to both problems, and still achieve acceptable results.

Detecting Redundancy

Faced with a set of operators, O, and an operator $o \in O$, how do we determine if O is redundant, *i.e.* if there exists a sequence of operators that implements o? To a first approximation, we can state this as a planning problem: Find a plan for initial state P_o and goal A_o, utilising only operators in $O - \{o\}$. This approach, however, fails to take into account item (*iv*) of definition 4; we need to enforce the constraint that the plan *preserves* all atoms not in $D_o \cup INC(P_o)$. To achieve this, we search in the space of operator sequences of increasing length and compute the cumulative preconditions and effects directly according to definition 3. The search is guided by an automatically extracted heuristic function, like that of (Bonet, Loerincs, & Geffner 1997). To further reduce the branching factor of the search, we make use of the fact that the cumulative preconditions of a sequence of operators are non-decreasing with respect to appending an operator (*cf.* equation (*iii*) in the proof of Lemma 1) and therefore any operator sequence whose cumulative preconditions are not a subset of P_o can never be extended to one which implements o.

The sequence implementing a redundant operator can be up to $2^{|D(P)|}$ steps long. We can obtain a sound but incomplete algorithm by limiting the length of sequences considered. In practice, the implementing sequences appear for the most part to be very short. Table 1 shows, for a number of problems in the domains where we have found redundancy, the number of instantiated operators (excluding operators that can never be applied due to unreachable preconditions) and the number of redundant operators found with varying bounds on the length of the implementing sequence. Each operator has been tested for redundancy with respect to the set of all other operators and therefore it is not necessarily the case that all the redundant operators can be removed.

Problem	# operators	# redundant $\leqslant 5$	$\leqslant 3$	$\leqslant 2$
`blocks-11`	1210	990	990	990
`blocks-15`	3150	2730	2730	2730
`logistic.a`	210	12	12	12
`logistic.b`	192	24	24	24
`logistic.c`	256	24	24	24
`grid-1`	2384	1866	1858	1800
`grid-2`	4140	3346	3334	3240

Table 1: Number of redundant operators found with bounded search.

The reader should also keep in mind that the inconsistency relation we compute and make use of when testing for redundancy is only an approximation of the one described in definition 2.

Approximating the Minimal Set

Although the minimal equivalent digraph problem is NP-hard, there exists a polynomial time approximation algorithm guaranteed to return a solution within 1.75 of the optimal (Khuller, Raghavachari, & Young 1995). However, the algorithm is polynomial (slightly above linear) time in the size of the graph. It also requires space linear in the size of the graph. Since for planning problems the state-transition graph is exponential in size, this makes it unusable in practice.

We use instead a simple greedy algorithm, which removes each redundant operator as soon as it is found and checks for redundancy with respect to the currently remaining set of operators. The redundancy check uses bounded search, so it is incomplete.

In the worst case, this may result in removing only a small fraction of the potentially redundant operators. Consider the graph in figure 1; the edges of a minimal set are indicated by dashed lines. However, a non-minimal reduced set can be formed by deleting only the two edges between any pair of adjacent nodes.

Table 2 shows, for the same collection of planning problems, the number of operators in a minimal reduced set and the number of operators in the reduced set found by our algorithm. Clearly, the algorithm achieves a far from optimal result, but, as the results of the next section show, in practice this seems to have relatively little effect.

Experiments

To test the conjecture that planning with a reduced operator set yields an increase in efficiency, we implemented a preprocessor[1] which converts a planning problem specified in PDDL to an equivalent problem with a reduced operator set and compared the runtimes of

[1]The software is available from the authors upon request.

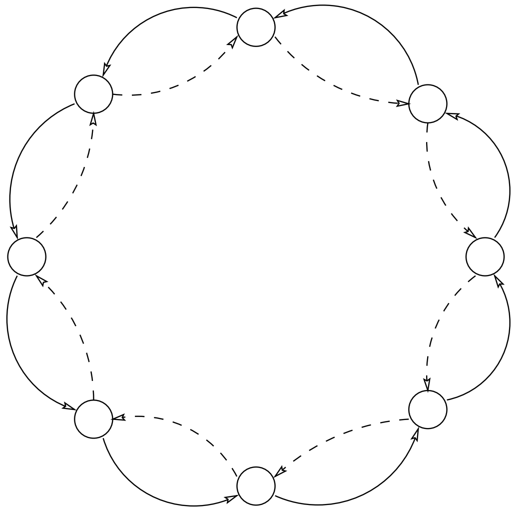

Figure 1: A graph on which the greedy approximation performes badly.

Problem	# operators	reduced	minimal
`blocks-11`	1210	220	220
`blocks-15`	3150	420	420
`logistic.a`	210	206	204
`logistic.b`	192	180	176
`logistic.c`	256	244	240

Table 2: Number of operators in reduced and minimal reduced sets.

several different planners on reduced problems to that on the same problems without reduction. The planners we used were all competitors in the STRIPS track of the AIPS98 planning competition, where they all performed well. Two of them, STAN (Fox & Long 1998) and BLACKBOX (Kautz & Selman 1999), are in part based on the use of planning graphs, but the third, HSP (Bonet, Loerincs, & Geffner 1997), is not. We also included GRAPHPLAN in the test suite.

The results are summarised in table 3. The times shown are the average of a varying number of trials. HSP and BLACKBOX show large variations in runtime from one trial to another, up to $\pm 17\%$ and $\pm 42\%$, respectively (with the exception of the non-reduced version of the `logistic.c` problem, on which BLACKBOX shows a deviation of over 200% from the average). For BLACKBOX in particular, the deviation measured as a percentage of average tends to be greater the longer the average runtime. For problems solved within a few seconds, the largest deviations were only a few percent. The other planners consistently stay within a few percent of the average. A dash indicates that the program exhausted the available memory, or crashed.

Because the preprocessor works with instantiated (propositional) formulas and operators, the reduced domain descriptions are also propositional. As already mentioned, the preprocessor also finds and eliminates static propositions, actions that are never applicable, *etc.* To make the comparison fair, the non-reduced version of each problem was also instantiated, and the same static analysis applied. The planners used in the experiments are likely not designed for propositional input (normally, the domain description is parameterised and the planner handles instantiation internally), which may have been the cause of some problems. Notably, we could not run HSP on the Grid domain, or any planner on the `grid-2` problem.

We also ran STAN and BLACKBOX on *minimal* reduced versions of the `logistic.b` and `logistic.c` problems, devised by hand. The results are shown in table 4.

Table 5 shows the time taken to preprocess the prob-

STAN

Problem	Operator set	
	original	reduced
`blocks-11`	2.4 sec.	0.2 sec.
`blocks-15`	1 min. 41 sec.	0.9 sec.
`logistic.a`	3.0 sec.	1.7 sec.
`logistic.b`	6.1 sec.	0.6 sec.
`logistic.c`	–	8.0 sec.
`grid-1`	3.3	1.6

BLACKBOX

Problem	Operator set	
	original	reduced
`blocks-11`	9.0 sec.	0.4 sec.
`blocks-15`	–	4.3 sec.
`logistic.a`	2.0 sec.	1.2 sec.
`logistic.b`	13.1 sec.	2.0 sec.
`logistic.c`	12 min. 2 sec.	5.3 sec.
`grid-1`	7.5 sec.	10.1 sec.

GRAPHPLAN

Problem	Operator set	
	original	reduced
`blocks-11`	3.7 sec.	0.3 sec.
`blocks-15`	–	4.5 sec.
`logistic.a`	7 min. 25 sec.	3 min. 15 sec.
`logistic.b`	8 min. 32 sec.	1 min. 38 sec.
`logistic.c`	–	–
`grid-1`	3.9 sec.	3.5 sec.

HSP

Problem	Operator set	
	original	reduced
`blocks-11`	0.9 sec.	0.6 sec.
`blocks-15`	5.6 sec.	1.7 sec.
`logistic.a`	1.2 sec.	1.9 sec.
`logistic.b`	1.1 sec.	0.9 sec.
`logistic.c`	1.5 sec.	2.1 sec.

Table 3: Runtimes using original and reduced operator sets.

Problem	STAN	BLACKBOX
`logistic.b`	0.6 sec.	2.0 sec.
`logistic.c`	3.7	3.4 sec.

Table 4: Runtimes using minimal reduced operator sets.

Problem	Time	Inc.	Red.
`blocks-11`	6.4 sec.	33%	59%
`blocks-15`	51.5 sec.	37%	60%
`logistic.a`	0.4 sec.	31%	23%
`logistic.b`	0.5 sec.	17%	17%
`logistic.c`	0.7 sec.	25%	25%
`grid-1`	1 min. 52 sec.	30%	65%

Table 5: Total preprocessing time and percent spent computing atom inconsistency and reduced operator set.

lems used in the experiments. Besides the total time, the percentage spent in computing the atom inconsistency relation and the reduction of the operator set is given.

Discussion of the Results

Though the results of the experiments for the most part support our conjecture, there were also some surprises. First, redundancy is not as common as we initially believed. In the standard benchmark domains, including the domains from the AIPS'98 competition, we found only three to contain redundant operators: the Blocksworld, Logistics and Grid domains.

Second, not all planners benefit equally from reduction of the operator set. This we expected, but that the plangraph-based planners seem to gain the most from reduction is a bit surprising, as the plans found using a reduced operator set are longer, both in number of actions and number of time steps, than the plans found using the original set. It is for this reason we included GRAPHPLAN in the test suite, since the tendency that appears in the results of STAN and BLACKBOX is exhibited even more clearly by GRAPHPLAN.

There seems to be no simple relationship between the number of operators removed and the decrease in planning time. The greatest gain was shown in the Logistics domain, were the reduced set contains only a few operators less than the original. The experiments with minimal reduced operator sets indicate that minimality is not as important as irredundancy, though the data set is really too small to draw any general conclusion on this question.

Sources of Redundancy in the Example Domains

We can distinguish two different kinds of redundant operators in the example domains. The first kind is distinguished by the fact that they are not a necessary part of the implementing sequence for some other redundant operator, which means all redundant operators of this kind can be removed. We find examples of this kind in the Blocksworld domain, where all operators that move a block from on top of another block directly to another block are redundant (as shown in example 1) and all such operators can be removed.

The other kind is a subset of operators which "implement each other"; we find an example in the Logistics domain, where all instances of the `fly-airplane` operator are redundant, *w.r.t* the set of all other instances, since any airplane can reach any airport as long as the remaining operators form a strongly connected spanning subgraph between the cities. However, obviously not all of them can be removed.

The Grid domain contains both the above sorts of redundacy. All instances of the **pickup-and-loose** operator, which cause the robot to pick up a key and drop another, can be implemented by a two-step sequence (first dropping, then picking up). Some of the **move** operators are redundant, as shown in example 2, but this is redundancy of the second kind; in a "circle" of four adjacent nodes, only one of the four **move** operators can be removed. Similarly, some of the **unlock** operators, which unlock a locked grid node, provided the robot is in an adjacent node, can be implemented by moving the robot to another adjacent node, unlocking from there, and moving the robot back.

Conclusions

We have introduced and defined the concept of redundant operators in planning and reduction of an operator set, and shown how an approximate reduced set can be computed. We have also provided some experimental evidence that reducing the operator set can significantly speed-up planning. What is lacking, however, is an adequate explanation of this phenomenon. According to the argument put forth in the introduction, the time to search for a plan should decrease if the decrease in the branching factor caused by reduction is large and the increase in solution depth is relatively small. In the Logistics domain, where the gain was greatest, the redundant `fly-airplane` operators are certainly few compared to the number of loading and unloading operators. The increase in solution depth for reduced problems in this domain is also significant.

We have made one more observation which may shed some light on this issue. For planners based on planning graphs, the number of "failed levels" (*i.e.* the number of levels in the planning graph between the first where all goal atoms occur non-mutex and the first level where they can actually be achieved) tend to be fewer when using a reduced operator set than when using the original set. Viewing the plan graph construction as being in fact the computation of an admissible (underestimating) heuristic, as suggested by Bonet and Geffner (1999), we may hypothesise that reducing the operator set eliminates some of the "short-cuts" in the state-transition graph, which causes the heuristic to make an estimate closer to the actual cost. The difference in failed levels is, however, quite small (at most two, and usually only one, level).

Another important question concerns the applicability of the reduction method to real-world problems. We have noted that redundant operator sets are not very common in planning problems. For instance, among the problems for the AIPS'98 competition, only the Blocksworld, Logistics and Grid domains were found to contain redundant operators. On the other hand, the detection of redundant operators is a relatively cheap computational task (at least in the approximative sense used in this paper). In the final analysis, it should of course also be kept in mind that while reduction preserves solution existence, it does not preserve solution length: Plans in reduced domains tend to be longer and, from an execution perspective, this is not a trivial price to pay.

Even though this particular method may not be applicable in that many cases, we are quite convinced that preprocessing of planning domains is in general a good idea, and that the range of possible techniques has been far from exhausted. Perhaps results of this paper contribute more to the understanding of the weaknesses of plangraph based planners, and a better way to achieve the same increase in performance is to develop more accurate heuristics.

Acknowledgements

Thanks to Hector Geffner and Sylvie Thiebaux for comments and interesting discussions. We also thank the reviewers for some helpful remarks. This research has been sponsored by the *Wallenberg Foundation*, the *ECSEL/ENSYM* graduate studies program and the *Swedish Research Council for the Engineering Sciences* (TFR) under grant Dnr. 97-301.

References

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

Blum, A., and Furst, M. 1997. Fast planning through graph analysis. *Artificial Intelligence* 90(1–2):281–300.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *Proc. 5th European Conference on Planning.*

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proc. 14th National Conference on Artificial Intelligence.*

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

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research* 9:367 – 421.

Garey, M., and Johnson, D. 1979. *Computers and intractability: A guide to the theory of NP-completeness.* Freeman.

Gazen, B., and Knoblock, C. 1997. Combining the expressivity of UCPOP with the efficiency of Graphplan. In *Proc. 4th European Conference on Planning*, 223–235.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proc. 13th National Conference on Artifical Intelligence.*

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proc. 16th International Joint Conference on Artificial Intelligence.*

Khuller, S.; Raghavachari, B.; and Young, N. 1995. Approximating the minimum equivalent digraph. *SIAM Journal on Computation* 24:859 – 872.

Knoblock, C. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68:243 – 302.

Koehler, J.; Nebel, B.; Hoffman, J.; and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. In *Proc. 4th European Conference on Planning*, 275–287.

Lifschitz, V. 1986. On the semantics of STRIPS. In Georgeff, M., and Lansky, A., eds., *Reasoning about Actions and Plans, Proceedings of the 1986 Workshop.* Morgan Kaufmann.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. 9th National Conference on Artificial Intelligence.*

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *Proc. 4th European Conference on Planning*, 338–350.

Peot, M. A., and Smith, D. E. 1993. Threat-removal strategies for partial-order planning. In *Proc. 11th National Confrence on Artificial Intelligence.*

Scholz, U. 1999. Action constraints for planning. In *Proc. 5th European Conference on Planning.*