Search Reduction in Planning with Primary Effects

Eugene Fink *
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
eugene@cs.cmu.edu

Qiang Yang *
Computer Science Department
University of Waterloo
Waterloo, Ont., Canada N2L3G1
qyang@logos.uwaterloo.ca

Abstract

The use of primary effects in planning is an effective approach for reducing search costs, closely related to abstraction planning. However, there has been little analysis of planning with primary effects and few experimental results. In this paper we demonstrate the connection between primary effects and abstraction hierarchies and provide analytical and empirical results on the efficiency of planning with primary effects. First, we show how the relationship between primary effects and abstraction can be used for automatically selecting primary effects and generating good abstraction hierarchies. Then we analytically demonstrate that the use of primary effects may lead to an exponential reduction of the running time and identify factors that influence the efficiency of planning with primary effects. Finally, we describe experiments that confirm our analysis and demonstrate the practical efficiency of using primary effects.

1 Introduction

Planning with primary effects is an effective approach for reducing search costs. The idea of this approach is to select *primary* effects among the effects of each operator and to use an operator only when we need to achieve one of its primary effects. A *primary-effect restricted* planner never inserts an operator into a plan in order to achieve any of the side effects of the operator. For example, the primary effect of lighting a fireplace is to heat the house. If we have lamps in the house, we should view illumination of the room as a side effect of using a fireplace. We would not use a fireplace just to illuminate the room.

The use of primary effects is closely related to abstraction planning, since both methods are based on the same notion of difference in importance between different features of the problem domain. For example, the priorities in our fireplace domain can be described in terms of an abstraction hierarchy: we may put the room temperature on the higher level of the hierarchy, and the illumination on the lower level. Then an abstraction planner

*The first author is supported by Carnegie Mellon University. The second author is supported in part by grants from the Natural Sciences and Engineering Research Council of Canada and ITRC.

first searches for a plan for heating the house and then refines this plan by adding lighting operations.

The advantages of using primary effects in planning are well-known. If a planner considers only operators whose primary effects match a current goal, the branching factor of search can be reduced. This may result in an exponential reduction of running time. For this reason, primary effects are used by many implemented planning systems, such as SIPE [Wilkins, 1988], PRODIGY [Carbonell *et al.*, 1990], and ABTWEAK [Yang and Tenenberg, 1990]. Interestingly, primary effects also play an important role in systems that automatically generate abstraction hierarchies. Recent work [Knoblock, 1991], [Fink and Yang, 1992] has shown that the use of primary effects allows us to increase the number of abstraction levels, resulting in hierarchies that improve the efficiency of abstract planning.

Recently we formalized the notion of planning with primary effects and developed two algorithms, MARGIE [Fink and Yang, 1992] and Prim-TWEAK [Fink and Yang, 1993], for automatically selecting primary effects of operators.

1.1 Overview of the Results

The purpose of this paper is three-fold. First, we illustrate the relationship between primary effects and abstraction. We show that (1) the knowledge of primary effects can be used in generating abstraction hierarchies (Section 2.2) and (2) an abstraction-generating system, ALPINE [Knoblock, 1991], can be used for automatically selecting primary effects of operators (Section 2.3).

Second, we analyze benefits of planning with primary effects (Section 3). We pinpoint the factors that determine the efficiency of a primary-effect restricted planner and derive the conditions under which the use of primary effects improves planning efficiency. Our analysis demonstrates that the use of primary effects may result in an exponential reduction of planning time.

Third, we describe experiments on planning with primary effects (Section 4). The experiments confirm the results of our analysis and demonstrate practical efficiency of primary-effect restricted planning.

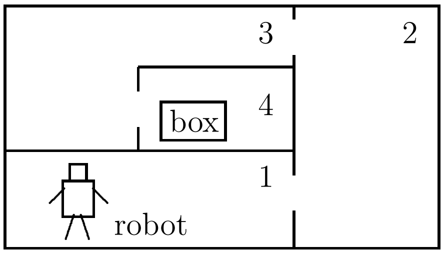
2 Planning with Primary Effects

In this section we describe the use of primary effects in planning, show their close connection with abstraction, and outline an algorithm for automatically selecting primary effects of operators.

2.1 Motivating Example

Consider a planning domain with a robot, four rooms, and a box (see Table 1). The robot may go between two rooms connected by a door, and it may carry the box. If two rooms are separated by a wall, the robot may break through the wall to create a new door.

To describe a current *state* of the domain, we have to specify the location of the robot and the box, and pairs of rooms connected by doors. This may be done with three predicates,



operator	**preconditions**	**effects**	**cost**
go(x, y)	robot-in(x), door(x, y)	robot-in(y), $\neg$robot-in(x)	2
carry-box(x, y)	robot-in(x), box-in(x) door(x, y)	robot-in(y), box-in(y) $\neg$robot-in(x), $\neg$box-in(x)	3
break(x, y)	robot-in(x)	robot-in(y), $\neg$robot-in(x) door(x, y)	4

Table 1: Robot domain

robot-in(x), *box-in*(x), and *door*(x, y). Literals describing a current state of the domain may be obtained from these predicates by substituting specific room numbers for x and y. For example, the literal *robot-in*(1) means that the robot is in Room 1, *box-in*(4) means that the box is in Room 4, and *door*(1,2) means that Room 1 and Room 2 are connected by a doorway.

The operations performed by the robot, such as moving between rooms or carrying the box, are called *operators*. Each operator is described by a *set of preconditions*, a *set of effects*, and a real-valued *cost* (see Table 1). The *preconditions* of an operator are the conditions that must hold before the execution of the operator, and the *effects* are the results of the execution. If a literal l is an effect of some operator, we say that this operator *achieves* l.

A *plan* is a sequence of operators that achieves some desired goal. For example, assume that the initial state is as shown in Table 1, and we wish to bring both the robot and the box into Room 3. This goal may be achieved by the plan (*break*(1,4), *carry-box*(4,3)). We define the cost of a plan as the sum of the costs of its operators. The cost of our plan is $4 + 3 = 7$. An *optimal plan* is the cheapest possible plan that achieves the goal.

If an operator has several effects, we may choose certain "important" effects among them and insert the operator into a plan only for the sake of these effects. The chosen important effects are called *primary*. The other effects are called *side* effects.

For example, consider the following selection of primary effects:

Selection 1	
Operators	**Prim. Effects**
go(x, y)	robot-in(y)
carry-box(x, y)	box-in(y)
break(x, y)	door(x, y)

Assume that the initial state is as shown in Table 1 and the robot must go into Room 3. The robot may achieve this goal by breaking through the wall between Rooms 1 and 3. However, since changing the location of the robot is *not* a primary effect of *break*, a planner will not consider this possibility. Instead, it will find the plan (*go*(1,2), *go*(2,3)).

2.2 Primary Effects in Abstraction Planning

Intuitively, the use of primary effects and planning in an abstraction hierarchy are based on the same notion of different importance of literals in a problem domain. In abstraction planning, we put important literals on higher levels of a hierarchy, while in primary-effect restricted planning we select important literals as primary effects.

This intuition can be illustrated by the ALPINE algorithm [Knoblock, 1991], which uses the knowledge of primary effects of operators to generate an abstraction hierarchy. The algorithm can extract the information about importance of literals from a selection of primary effects and encode this information in the form of a hierarchy. ALPINE represents an importance of every literal in a problem domain by a natural number: the larger the number, the more important the literal. Hierarchies generated by ALPINE are based on the following three constraints:

For every operator *Op* in a problem domain
(1) if $prim_1$ and $prim_2$ are primary effects of *Op*,
  then $Importance(prim_1) = Importance(prim_2)$;
(2) if *prim* is a primary effect of *Op* and *side* is a side effect of *Op*,
  then $Importance(prim) \geq Importance(side)$;
(3) if *prim* is a primary effect of *Op* and *prec* is a precondition of *Op*,
  then $Importance(prim) \geq Importance(prec)$.

These constraints guarantee that when we refine an abstract plan on a lower level of a hierarchy, we cannot change any high-level literal [Knoblock *et al.*, 1991]. This property usually leads to a high efficiency of abstraction planning.

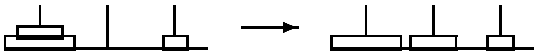
Observe how the first two constraints utilize the information about primary effects of operators. According to Constraint 1, if two literals are primary effects of the same operator then they are equally important. According to Constraint 2, primary effects of an operator are more important than its side effects.

For example, consider our robot domain with primary effects that we selected in the end of Section 2.1 (Selection 1). The following hierarchy of literals in this domain satisfies ALPINE's constraints:

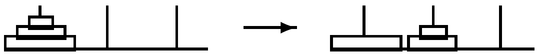
box-in(x)	Level 1 (more important)
robot-in(x) door(x,y)	Level 0 (less important)

In our next example, we generate an abstraction hierarchy for a modified version of the Tower of Hanoi domain. This example will demonstrate that primary effects may correspond to a hierarchy with *more than two* levels.

Consider the Tower of Hanoi with 3 pegs and 3 disks, and suppose that we can use two types of operators (see Figure 1): we can move a single disk from peg to peg, with the usual Tower-of-Hanoi restrictions; or we can move two disks at once, as long as both disks are on



Operator: move-medium(x,y)	
Preconditions	Effects
medium-on(x) not small-on(x) not small-on(y)	medium-on(y) not medium-on(x)



Operator: move-small-and-med(x,y)	
Preconditions	Effects
small-on(x) medium-on(x)	small-on(y) medium-on(y) not small-on(x) not medium-on(x)

Figure 1: Two operators of the Tower of Hanoi domain

Operators	**Prim. Effects**	**Operators**	**Prim. Effects**
move-small	small-on	move-small-and-med	medium-on
move-medium	medium-on	move-small-and-large	large-on
move-large	large-on	move-med-and-large	large-on

Table 2: Primary effects in the extended Tower of Hanoi domain

the same peg and there are no other disks between them. A state of this domain can be encoded by three predicates, *small-on*(x), *medium-on*(x), and *large-on*(x), which show the locations of the small, medium, and large disk. The actions in the domain can be expressed by six operators:

move-small(x, y)	move-small-and-med(x, y)
move-medium(x, y)	move-small-and-large(x, y)
move-large(x, y)	move-med-and-large(x, y)

A formal representation of two of these operators is shown in Figure 1.

Intuitively, moving a larger disk in this domain is harder than moving a smaller one. Thus, when we move two disks at once, we should view the movement of the larger disk as a primary effect of this action. For example, the primary effect of the operator *move-small-and-large* is the position of the large disk, since we would not use this operator if we only wanted to move the small disk. Table 2 shows the selection of primary effects based on this intuition.

Now let us use ALPINE to generate an abstraction hierarchy for this selection of primary effects. Since the position of the medium disk is a primary effect of the operator of *move-small-and-med*, while the position of the small disk is its side effect, Constraint 2 of the ALPINE algorithm produces the following inequality:

$$\textit{Importance}(\textit{medium-on}) \geq \textit{Importance}(\textit{small-on})$$

Similarly, the application of this constraint to the operators *move-small-and-large* and *move-med-and-large* gives us

$\textit{Importance}(\textit{large-on}) \geq \textit{Importance}(\textit{small-on})$, and
$\textit{Importance}(\textit{large-on}) \geq \textit{Importance}(\textit{meduim-on})$

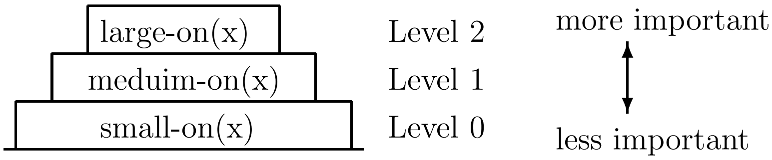


Figure 2: Abstraction hierarchy in the extended Tower of Hanoi domain

These three inequalities suggest the abstraction hierarchy shown in Figure 2. One can verify that this hierarchy satisfies not only Constraint 2, but also Constraints 1 and 3 of ALPINE. Observe that the resulting abstraction hierarchy corresponds to the human intuition and carries the same information about the relative importance of literals as our selection of primary effects.

2.3 Selecting Primary Effects

The utility of primary-effect restricted planning crucially depends on a good selection of primary effects. If primary effects are used to construct an abstraction hierarchy, then the quality of the hierarchy is also determined by the selected effects.

An improper selection of primary effects may result in the loss of completeness of planning. This happens when a primary-effect restricted planner cannot find a plan for a solvable planning problem. For example, if we use the primary effects chosen at the end of Section 2.1 (Selection 1), a planner cannot find a plan for removing the box from Room 4, since $\neg$*box-in* is not a primary effect of any operator.

Most planning systems rely on the user in finding primary effects. For example, the user-specified primary effects are used in PRODIGY [Carbonell *et al.*, 1990] and ABTWEAK [Yang and Tenenberg, 1990]. If the user does not specify primary effects, then by default *all* effects are considered primary.

In [Fink and Yang, 1992] we presented a simple technique for automatically selecting primary effects of operators. To achieve the completeness of planning, we enforce the following two restrictions:

(1) every achievable literal must be a primary effect of some operator, and

(2) every operator must have a primary effect.

Restriction 1 guarantees that if a literal can be achieved at all, then it can be achieved as a primary effect of some operator. If a literal were not a primary effect of any operator, the planner would not find a way of achieving it. Restriction 2 insures that the planner can use all operators of the problem domain. These restrictions are a heuristic rather than a provable guarantee of completeness: they enable us to find near-complete selections of primary effects in most planning domains, but in some situations the completeness is lost [Fink, 1992].

Our algorithm for selecting primary effects according to the two restrictions can be informally described as follows:

1. For every literal l in the problem domain,
   if there are operators that achieve l,
   then make l a primary effect of one of these operators.

2. For every operator *Op* that does not have primary effects yet,
   make one of the effects of *Op* primary.

Notice that for every literal l, there may be several different ways to choose an operator for achieving l at Step 1 of the algorithm. Similarly, at Step 2 we may select different primary effects of every operator *Op* that still does not have primary effects. When making these choices, our algorithm uses the following heuristic:

> When selecting a new primary effect, choose it in such a way as to maximize the number of levels in the hierarchy produced by ALPINE.

Thus, the goal of the algorithm is to find primary effects that increase the quality of the ALPINE-generated hierarchy while preserving the completeness of planning.

For example, suppose we use this technique to select primary effects of operators in our extended Tower of Hanoi domain. At the first step, the algorithm makes sure that every literal is a primary effect of some operator. This may be achieved by selecting the following primary effects:

Operators	**Prim. Effects**
move-small	small-on
move-medium	medium-on
move-large	large-on

After applying ALPINE's constraints to this selection of primary effects, we obtain the following inequality:

$$Importance(large\text{-}on) \geq Importance(medium\text{-}on) \geq Importance(small\text{-}on)$$

This inequality corresponds to the hierarchy shown in Figure 2. However, there are still three operators that do not have primary effects, and on the second step the algorithm selects primary effects of these operators.

Let us consider the processing of the operator *move-small-and-medium*. The effects of this operator are *small-on* and *medium-on*, and one of these effects must be selected as primary. If *small-on* is chosen as a primary effect, then, according to ALPINE's Constraint 2, *small-on* is at least as important as *medium-on*:

$$Importance(small\text{-}on) \geq Importance(meduim\text{-}on)$$

Combining this inequality with the previous one, we get

$$Importance(large\text{-}on) \geq Importance(medium\text{-}on) = Importance(small\text{-}on)$$

and thus the number of abstraction levels is reduced to two.

On the other hand, is we select *medium-on* as a primary effect of *move-small-and-medium*, then ALPINE's constraints give us

$$Importance(medium\text{-}on) \geq Importance(small\text{-}on)$$

This inequality does not add any new constraints and the number of abstraction levels remains three. Since the goal of our algorithm is to maximize the number of abstraction levels, it will select *medium-on* as a primary effect of *move-small-and-medium*.

Similarly, the algorithm selects *large-on* as a primary effect of the operators *move-small-and-large* and *move-med-and-large*, and thus it generates the selection of primary effects shown in Table 2.

As another example, our algorithm would select the following primary effects in the robot domain:

Selection 2	
Operators	**Prim. Effects**
go(x, y)	robot-in(y), ¬robot-in(x)
carry-box(x, y)	box-in(y), ¬box-in(x)
break(x, y)	door(x, y)

An increase in the number of abstraction levels due to a proper selection of primary effects leads to improving the efficiency of abstract planning. Interestingly, our experiments [Fink and Yang, 1992] show that the use of selected primary effects improves the performance even without using abstraction. This happens because primary effects carry the same information about importance of literals as the corresponding ALPINE's hierarchy, and thus a primary-effect restricted planner implicitly uses this abstraction hierarchy.

2.4 Completeness and Cost Increase

While the use of primary effects chosen by the above algorithm may improve the efficiency of planning, it can also cause two serious problems. First, the resulting selection of primary effects is not immune to the loss of completeness. Second, the algorithm does not take into account the costs of operators, and planning with the selected primary effects may result in finding a non-optimal solution. This happens because primary effects place a bias in directing the search for a solution. If not set properly, the bias can favor a search toward a costly solution. For example, the goal of moving the robot from Room 1 to Room 4 may be achieved by the operator *break*(1,4), the cost of which is 4. However, a primary-effect restricted planner will not consider this possibility, since the new position of the robot in not a primary effect of *break*. Instead, the planner will find the plan (*go*(1,2), *go*(2,3), *go*(3,4)), with a cost 6. In this example, the ratio of the cost of the primary-effect restricted solution to the cost of the optimal solution is $6/4 = 1.5$. This ratio is called the *cost increase* of the planning problem.

To address the problem of completeness and optimality, we present an algorithm that tests whether a given choice of primary effects guarantees the completeness of planning and estimates the maximal cost increase.

Recall that planning with primary effects is complete if a primary-effect restricted planner can find a plan for every solvable problem. Completeness is lost when we can achieve a goal as a side effect of some operator, but cannot find a primary-effect restricted plan that achieves this goal. To guarantee completeness, we must insure that such a situation cannot happen. That is, we have to make sure that *for every operator there is always a primary-effect restricted plan that achieves its side effects.*

Now suppose that a somewhat stronger condition holds: for some constant C, every operator Op can always be replaced by a primary-effect restricted plan *whose cost is at most* $C \cdot cost(Op)$. Then the cost increase due to the use of primary effects is never larger than C. Thus, this stronger condition not only insures completeness, but also guarantees a limited cost increase for all possible problems in a planning domain. Below, we summarize our observations:

> **Completeness and Limited Cost Increase:**
> Suppose that for every operator *Op* and every initial state S satisfying the preconditions of *Op*, there is a primary-effect restricted plan $\mathcal{P}$ with the initial state S such that
> (1) $\mathcal{P}$ achieves the side effects of *Op* and
> (2) $cost(\mathcal{P}) \leq C \cdot cost(Op)$.
> Then for any solvable planning problem, there is a primary-effect restricted solution at most C times more expensive than an optimal solution.

A formal proof and discussion of this observation can be found in [Fink and Yang, 1993]. We use this result to design a learning algorithm that adds new primary effects to a selection to make sure that primary-effect restricted planning is complete and that the cost increase is within the user-specified bound. For each operator *Op* in the problem domain, the learner generates several states that satisfy the preconditions of *Op* and checks whether the side effects of *Op* can be achieved in all these states. If not, the algorithm "promotes" some side effects of *Op* to primary effects. Informally, this learning algorithm can be described as follows:

1. Ask the user for the maximal allowed cost increase, C.
2. For each operator *Op* in the problem domain, repeat "several" times:
   (a) Pick at random a state S that satisfies the preconditions of *Op*.
   (b) Try to find a primary-effect restricted plan $\mathcal{P}$ such that
      ∘ $\mathcal{P}$ achieves the side effects of *Op* from the initial state S, and
      ∘ the cost of $\mathcal{P}$ is at most $C \cdot cost(Op)$.
   (c) If such a plan is not found,
      then make one of the side effects of *Op* be a primary effect.

In [Fink and Yang, 1993] we presented a formal description of this algorithm and estimated the number of different initial states that must be considered for every operator to insure a high probability of completeness.

As an example, suppose we apply this learning algorithm to our robot domain, with primary effects selected by the algorithm of Section 2.2 (Selection 2). The learner is likely to notice that the effect *robot-in*(4) of the operator *break*(1,4) cannot be efficiently achieved by a primary-effect restricted plan. After noticing it, the algorithm will add a new primary effect to the *break* operator, producing the following selection:

Selection 3

Operators	**Prim. Effects**
go(x, y)	robot-in(y), $\neg$robot-in(x)
carry-box(x, y)	box-in(y), $\neg$box-in(x)
break(x, y)	door(x, y), robot-in(y)

3 Analysis of the Search Reduction

In this section we present an analytical comparison of planning efficiency with and without the use of primary effects. Our evaluation of the planning time is based on analyzing the search space of backward-chaining planning systems, such as PRODIGY, TWEAK, and SNLP. This analysis is an approximation based on several simplifying assumptions about properties of planning domains. It is similar to the analysis of planning with macro operators in [Korf, 1987] and the analysis of hierarchical problem solving in [Knoblock, 1991a]. Even though most real domains do not satisfy the restrictive assumptions of our formal analysis, the experimental results in Section 4 demonstrate that planning algorithms usually behave as predicted by the analysis.

3.1 Assumptions of the analysis

A planning algorithm determines a search space, which may be characterized as a tree [Minton *et al.*, 1991]. Each node in the search space corresponds to an intermediate (possibly incorrect) plan found in the process of planning. A total-order planner, such as ABSTRIPS or PRODIGY, creates a new node by inserting a new operator into a plan. A partial-order planner, such as TWEAK, may create a node by inserting a new operator or by imposing a constraint on the order of executing old operators. The running time of a planner is proportional to the number of nodes in the search space.

We assume that the planner expands nodes in the best-first order on the cost of a plan. This means that it always expands the node with the cheapest plan.

When inserting a new operator to achieve some literal l, we may use any operator whose primary effect is l. To estimate the number of applicable operators, we assume that, for all literals l, the number of operators achieving l as a primary effect is the same. We define P as $P = \sum_{Op} |\textit{Prim-Eff}(Op)|$. That is, we count the number of primary effects, $|\textit{Prim-Eff}(Op)|$, of each operator Op and define P as the sum of these numbers for all operators. The number of literals in the problem domain is denoted by L. Then the number of operators achieving some particular literal l as a primary effect is $\frac{P}{L}$.

After inserting some operator into a partial-order plan, we may have to impose new constraints on the order of operators in the plan. Let b denote the number of different ways of imposing these ordering constraints. Thus, we can create $b\frac{P}{L}$ different plans by adding an operator that achieves l and imposing necessary ordering constraints. A best-first planner considers all these plans, and therefore the total number of new nodes created after inserting an operator is $b\frac{P}{L}$. This value represents the branching factor of inserting an operator into a primary-effect restricted plan.

Similarly, we assume that the number of all operators achieving l is the same for all literals, and define E by $E = \sum_{Op} |\textit{Effects}(Op)|$, where $|\textit{Effects}(Op)|$ is the number of all effects of Op. Then the branching factor of inserting an operator by a planner that does not use primary effects is $b\frac{E}{L}$.

Finally, we assume that all operators have the same cost. All assumptions of our analysis are listed in Table 3, and the notation used in the analysis is summarized in Table 4.

1. The nodes are expanded in the best-first order on the cost of a plan.
2. The number of nodes expanded by imposing ordering constraints after inserting an operator is always the same, b.
3. For every literal l, the number of operators with an effect l is the same.
4. For every l, the number of operators with a primary effect l is the same.
5. All operators have the same cost.

Table 3: Assumptions of the analysis

P	the number of primary effects of all operators, $P = \sum_{Op} \lvert \textit{Prim-Eff}(Op) \rvert$
E	the number of all effects of all operators, $E = \sum_{Op} \lvert \textit{Effects}(Op) \rvert$
L	the number of literals in the problem domain
b	the number of different ways to impose ordering constraints after inserting an operator
n	the size of the optimal solution of a planning problem
C	the maximal cost increase of planning with primary effects
R	the ratio of running times of planning with and without primary effects

Table 4: Summary of the notation

3.2 Conditions of the Search Reduction

First, we consider planning without primary effects. Assume that the optimal solution of a planning problem contains n operators. Recall that the branching factor of inserting an operator is $b\frac{E}{L}$. Thus, the number of nodes with one-operator plans is $b\frac{E}{L}$, the number of nodes with two-operator plans is $(b\frac{E}{L})^2$, etc. The total number of nodes is

$$1 + b\frac{E}{L} + (b\frac{E}{L})^2 + ... + (b\frac{E}{L})^n = \frac{(b\frac{E}{L})^{n+1} - 1}{b\frac{E}{L} - 1} \tag{1}$$

If the maximal cost increase equals C, then the number of operators in the solution found by a primary-effect restricted planner is at most $C \cdot n$. Since the branching factor of inserting an operator in primary-effect restricted planning is $b\frac{P}{L}$, the number of nodes in the search space of a primary-effect restricted planner is *at most*

$$\frac{(b\frac{P}{L})^{C \cdot n+1} - 1}{b\frac{P}{L} - 1} \tag{2}$$

Let R denote the ratio of running times of planning with and without primary effects. Since the running time is proportional to the number of nodes in the search space, R is determined by the ratio of the search-space sizes:

$$R \leq \frac{((b\frac{P}{L})^{C \cdot n+1} - 1)/(b\frac{P}{L} - 1)}{((b\frac{E}{L})^{n+1} - 1)/(b\frac{E}{L} - 1)} = O\left(\frac{(b\frac{P}{L})^{C \cdot n}}{(b\frac{E}{L})^n}\right) = O\left(\left(\frac{(b\frac{P}{L})^C}{b\frac{E}{L}}\right)^n\right) \tag{3}$$

Let us denote the base of the exponent in Formula 3 by r:

$$r = \frac{(b\frac{P}{L})^C}{b\frac{E}{L}} = \frac{L}{E \cdot b} \cdot \left(\frac{P \cdot b}{L}\right)^C \tag{4}$$

Then Formula 3 may be rewritten as $R = O(r^n)$. If $r < 1$, *the saving in running time grows exponentially with the size of the optimal solution, n.* The smaller the value of r, the greater the saving.

Formula 4 shows that $r < 1$ when $\frac{L}{E \cdot b} \cdot \left(\frac{P \cdot b}{L}\right)^C < 1$. Solving this inequality w.r.t. C, we conclude that the use of primary effects improves the efficiency when

$$C < \frac{\log E + \log b - \log L}{\log P + \log b - \log L} \tag{5}$$

From the above formulas we can make some general conclusions. First, observe that, since $E > P$, the right-hand side of Inequality 5 is larger than 1. Thus, *if the greatest cost increase C equals 1, the use of primary effects always reduces the running time.* (However, it may not be the maximal possible reduction.)

Next, recall that the base of the exponent in Formula 4, $\frac{P \cdot b}{L}$, is the branching factor of the primary-effect restricted search, and therefore it is larger than 1. Thus, r increases with an increase of C. This implies that *the time saving increases with a decrease of the cost increase C.* Also, r increases with an increase of P, and therefore *the time saving increases with a decrease of the number of primary effects P.*

However, there is a tradeoff between the number of primary effects and the cost increase: as we select more primary effects, P increases, while the cost increase C decreases. To minimize r, we have to strike the right balance between C and P. This may be done by the learning algorithm outlined in Section 2.4 [Fink and Yang, 1993].

4 Experimental Results

We have shown analytically that the use of primary effects produces an exponential reduction in the size of the search space. In this section, we present a series of experiments that confirm this result and demonstrate the effectiveness of planning with primary effects.

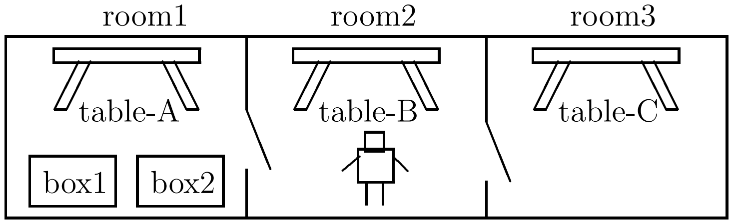
We used the ABTWEAK planning system [Yang and Tenenberg, 1990] in our experiments. The primary effects were selected by the learning algorithm outlined in Section 2.4. We performed our experiments with a modified version of the robot domain and with a series of artificial domains similar to those described in [Barrett and Weld, 1992].

In what follows we first demonstrate the effectiveness of our algorithm on the robot domain. Then we describe the family of artificial domains used to perform controlled experiments and show how the efficiency of primary-effect restricted planning in these domains depends on (1) the sizes of solution plans and (2) the number of different ways to achieve the same literal.

4.1 Experiments in a Robot Domain

In this section we describe the performance of our algorithm in a simple robot world, a modified version of the robot domain from [Fikes and Nilsson, 1971] (see Table 5).

The robot can move between rooms, open and close doors, carry boxes, and climb tables (with or without a box). The predicates in the domain include predicates indicating the



Goal	Solution Size	CPU time, ms		Branch. Factor		Expanded Nodes	
		w/ prim	w/o	w/ prim	w/o	w/ prim	w/o
robot-on(table-B)	1	50	50	1.0	1.5	2	2
status(door12, open)	2	183	217	1.4	2.0	7	7
robot-in(room1)	3	267	350	1.3	1.7	9	11
box-at(box1, door12)	5	800	1933	1.3	2.1	26	56
robot-on(table-C)	5	783	1200	1.3	2.3	26	36
box-on(box1, table-A)	5	800	1250	1.3	2.0	181	368
box-at(box2, table-B)	7	6550	14450	1.3	2.1	26	38

Table 5: Planning time with and without primary effects in the Robot Domain

location of the robot, e.g., *robot-in*(room3) and *robot-on*(table-C), predicates indicating the locations of the boxes, e.g., *box-in*(box1, room1) and *box-at*(box1, door12), and predicates indicating the status of the doors, e.g., *status*(door12, open).

The time of learning primary effects in this domain was 2980 CPU msec. Table 5 shows the performance of the ABTWEAK planner without primary effects ("w/o") and with the use of the learned primary effects ("w/ prim") when achieving different goals. The initial state in all cases is as shown in the picture, with all doors being closed. As can be seen from the table, the use of primary effects considerably improves the efficiency of ABTWEAK.

4.2 Artificial Domains

We ran a series of experiments with a family of artificial domains similar to those described in [Barrett and Weld, 1992]. These problem domains have a number of features that can be varied independently, enabling us to perform controlled experiments.

A problem in our domains is defined by m initial-state literals, $init_1, init_2, .., init_m$, and m goal literals, $goal_1, goal_2, .., goal_m$. There are also m operators, $Op_1, .., Op_m$. Each operator Op_i has the single precondition $init_i$. Op_i removes the literal $init_{i-1}$ and establishes the goal literals $goal_{i \,\mathbf{mod}\, m}, goal_{(i+1) \,\mathbf{mod}\, m}, .., goal_{(i+k-1) \,\mathbf{mod}\, m}$. Thus, each operator has $(k+1)$ effects: one negative effect and k positive effects. Our artificial domains allow us to vary the following problem features:

Goal Size The number of goal literals, m, can be changed. The size of an optimal solution changes in proportion to the number of goal literals.

Effect Overlap The *effect overlap*, k, is the average number of operators establishing the same literal. In terms of the previous section, $k = E/L$.

Cost Variation The *cost variation* is the statistical *coefficient of variation* of the costs of operators, defined as $(S_{cost}/\overline{cost})$, where S_{cost} is the standard deviation of the costs, and $\overline{cost}$ is their mean. Intuitively, the cost variation determines the relative difference between costs of different operators.

We vary these three features in our controlled experiments. While our domains are artificial, they demonstrate some important characteristics of real-world problems. For example, in the real world if the goal size increases, one would expect that the solution plan for achieving the goal should also increase in size. Likewise, if the effect overlap increases then every operator can achieve a larger number of goals, and the solution sizes decrease.

4.3 Varying Solution Sizes

First, we demonstrate that the saving in running time grows exponentially with the solution size. We experiment with effect overlaps 2 and 4. We consider cases where all operators have the same cost, and where costs of operators are distinct (with the cost variation 0.37). The results of experiments are presented in Figure 3. The graphs show the running times of the ABTWEAK planner without primary effects ("w/o prim") and with the use of primary effects ("with prim") for problems with different optimal-solution sizes. (Notice that the time scale is *logarithmic*.) Every point on each figure is the average of five different problems.

The graphs show that, as predicted by our analysis, the saving in planning time exponentially increases with an increase in solution size. A significant improvement is achieved in all four cases.

4.4 Varying Effect Overlap

Next we consider efficiency improvement for different values of the effect overlap. Recall that the effect overlap k is defined as the average number of operators achieving the same literal. We vary the effect overlap k from 2 to 6. (If the effect overlap is 1, then all effects must be selected as primary, and primary-effect restricted planning is equivalent to planning without primary effects.) We experiment with three different cost variations: (1) all operators have the same cost; (2) the costs of operators are distinct, with a small cost variation, 0.37; and (3) the cost of operators are distinct, with a large cost variation, 2.38. Figure 4 shows the running times of ABTWEAK with primary effects ("with prim") and without primary effects ("w/o prim") for every effect overlap under different cost variations.

The graphs show that with an increase of the effect overlap, the time saving of primary-effect restricted planning decreases. A possible explanation of the efficiency decrease for a large overlap is that solutions found by the primary-effect restricted planner are longer than optimal solutions, which results in a larger search depth of planning.

5 Conclusions

We have presented an analytical and empirical evaluation of planning with primary effects. Our analysis demonstrates that the use of primary effects may exponentially improve the

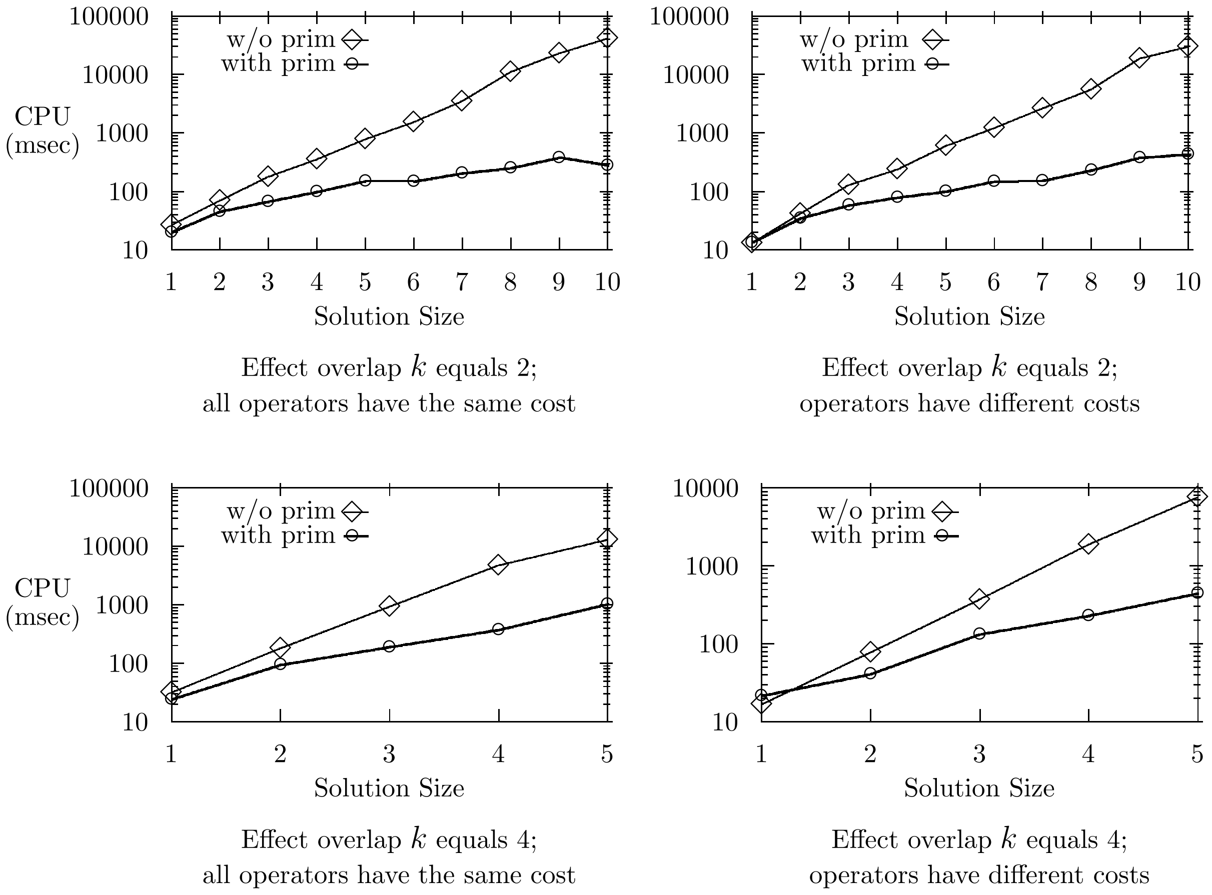


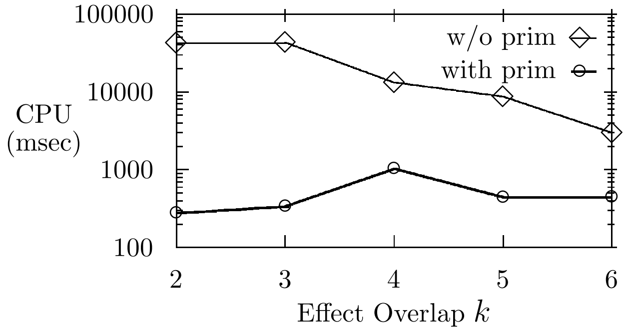
Figure 3: Dependency of planning time on the optimal-solution size

efficiency of planning. However, an improper selection of primary effects may lead to generating non-optimal solutions and increasing the planning time. The crucial factor is the cost increase C. This factor not only determines the optimality of solutions found by a primary-effect restricted planner, but also the efficiency of planning. If $C = 1$, then the use of primary effects always improves efficiency. If $C > 1$, primary-effect restricted planning leads to efficiency improvement only when the number of primary effects is sufficiently small.

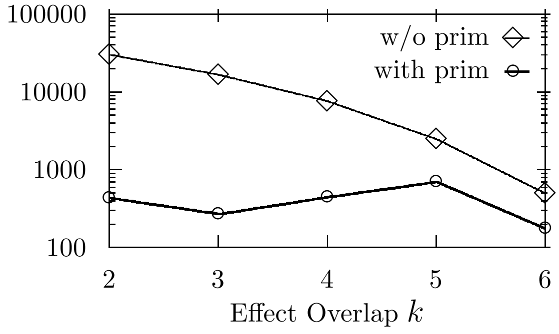
Our experimental results confirm that good selections of primary effects result in considerable efficiency improvement, and that the saving in planning time grows exponentially with the solution size.

5.1 Acknowledgements

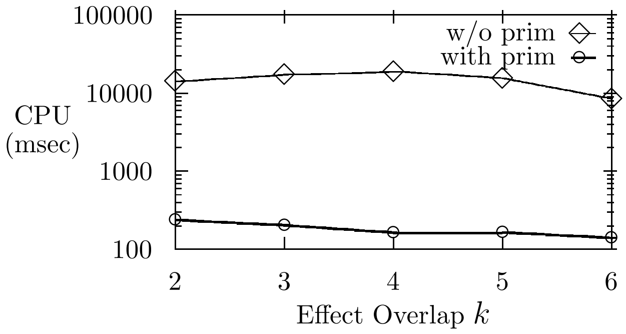
The authors would like to thank Manuela Veloso, Yury Smirnov, Constantine Domashnev, and Robert Monroe for helpful comments and discussions.



All operators have the same cost
(the cost variation is 0)



Operators have different costs,
with the cost variation 0.37



Operators have different costs,
with the cost variation 2.38

Figure 4: Dependency of planning time on the effect overlap

References

[Barrett and Weld, 1992] Anthony Barrett and Dan Weld. Partial order planning: evaluating possible efficiency gains. University of Washington, Department of Computer Science and Engineering, 1992. Tech. Report 92-05-01.

[Carbonell *et al.*, 1990] Jaime G. Carbonell, Craig A. Knoblock, and Steven Minton. PRODIGY: an integrated architecture for planning and learning. In *Architectures for Intelligence*, ed.: Kurt VanLehn, Erlbaum, Hillside, NJ, 1990.

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, pages 189–208, 1971.

[Fink, 1992] Eugene Fink. *Justified plans and ordered hierarchies.* Master's thesis, University of Waterloo, Department of Computer Science, Waterloo, Ontario, Canada. Research Report CS-92-42.

[Fink and Yang, 1992] Eugene Fink and Qiang Yang. Automatically abstracting effects of operators. In *Proceedings of the First International Conference on AI Planning Systems*, pages 243–251, 1992.

[Fink and Yang, 1993] Eugene Fink and Qiang Yang. Characterizing and automatically finding primary effects in planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1374–1379.

[Knoblock, 1991] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1991. Tech. Report CMU-CS-91-120.

[Knoblock, 1991a] Craig A. Knoblock. Search reduction in hierarchical problem solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 686-691, 1991.

[Knoblock *et al.*, 1994] Craig A. Knoblock. Automatically generating abstraction for planing. To appear in *Artificial Intelligence*, 1994.

[Knoblock *et al.*, 1991] Craig Knoblock, Josh Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In *Proceedings of the 9th AAAI*, Anaheim, CA, 1991.

[Korf, 1987] Richard E. Korf. Planning as search: a quantitative approach. *Artificial Intelligence*, 33(1), pages 65–88, 1987.

[Minton *et al.*, 1991] Steven Minton, John Bresina, and Mark Drummond. Commitment strategies in planning: a comparative analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 259–265, 1991.

[Wilkins, 1988] David Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann, CA, 1988.

[Yang and Tenenberg, 1990] Qiang Yang and Josh Tenenberg. ABTWEAK: abstracting a nonlinear, least commitment planner. In *Proceedings of Eighth National Conference on Artificial Intelligence*, pages 923–928, Boston, MA, 1990.