On the Implementation and Evaluation of AbTweak

Qiang Yang *
University of Waterloo
Canada

Josh D. Tenenberg †
Indiana University at South Bend
USA

Steven Woods ‡
University of Waterloo
Canada

April 19, 1995

Abstract

In this paper we describe the implementation and evaluation of the AbTweak planning system, a test bed for studying and teaching concepts in partial-order planning, abstraction, and search control. We start by extending the hierarchical, precondition-elimination abstraction of Abstrips to partial-order-based, least-commitment planners such as Tweak. The resulting system, AbTweak, is used to illustrate the advantages of using abstraction to improve the efficiency of search. We show that by protecting a subset of abstract conditions achieved so far, and by imposing a bias on search toward deeper levels in a hierarchy, planning efficiency can be greatly improved. Finally, we relate AbTweak to other planning systems Snlp, Alpine and Sipe by exploring their similarities and differences.

*Computer Science Department. Waterloo, Ont. Canada, N2L 3G1. Tel: (519)888-4567 X4716. Fax: (519)885-1208. Email: qyang@after.logos.uwaterloo.ca

†Department of Mathematics and Computer Science, 1700 Mishawaka Avenue, South Bend, IN 46617. Tel: (219) 237-4525. Email: tenenberg@acm.org

‡Computer Science Department. Waterloo, Ont. Canada, N2L 3G1. Email: sgwoods@after.logos.uwaterloo.ca

1 Introduction

Abstraction has been used as a method to reduce the computational complexity of classical planning. A large number of problem-solving systems, including GPS [Newell and Simon 1972], ABSTRIPS [Sacerdoti 1974], LAWLY [Siklossy and Dreussi 1973], NOAH [Sacerdoti 1977], NONLIN [Tate 1977], MOLGEN [Stefik 1981], SOAR [Unruh and Rosenbloom 1989], SIPE [Wilkins 1984], and ALPINE [Knoblock 1991], have been developed based on the concept of abstraction.

Despite the large number of proposed systems mentioned above, few of them are actually available for experimentation in abstraction. Systems such as GPS, LAWLY, MOLGEN and NOAH exist only as descriptions in several classical articles. ABSTRIPS and ALPINE are based on STRIPS, a special type of subgoal-based and total-order planning, and cannot be used to handle the now popular partial-order planning problems. Other systems such as NONLIN and SIPE are mega-systems that have many other built-in mechanisms, making the systematic study of abstraction in isolation of other techniques difficult.

In this paper, we present an implemented test bed for studying abstraction with nonlinear planning, combining ABSTRIPS-style abstraction with TWEAK-style partial-order planning. The result is an abstract, partial-order planning system called ABTWEAK[1]. This implemented software system is in the public domain, and has been requested by more than 100 universities and industries around the world. We will describe and justify our design decisions made in the implementation of ABTWEAK and explain a number of useful properties demonstrated by the system.

To improve search efficiency, ABTWEAK is also equipped with a number of built-in search control techniques that take advantage of an abstraction hierarchy. One such technique is called *Left-Wedge*, a search heuristic for speeding up abstract planning. Another is the use of *primary effects*, which restricts the branching factor of search dramatically. Although primary effects do not have to be coupled with abstraction, we argue that they are especially effective when used together. The efficiency gain associated with these techniques is verified in several empirical tests. In addition, we relate ABTWEAK to two other well-known systems, SIPE, which predates ABTWEAK, and SNLP which postdates it. In each case, we point out their similarities and differences and argue about their relative advantages. The organization of the paper is as follows. In Section 2, we present a planning formalism based on the TWEAK language, and demonstrate our extension to ABTWEAK. Section 3.1 presents the monotonic goal protection heuristic, Section 3.2 discusses the LEFT-WEDGE heuristic for controlling abstract search, and Section 3.3 shows how domain-dependent heuristics can be effectively combined with abstract planning. An empirical evaluation of the heuristics is done in Section 4, and a comparison to related work is made in Section 5. Finally, conclusions are given in Section 6.

[1]ABTWEAK is available via WWW at http://after.logos.uwaterloo.ca/abtweak or anonymous ftp on after.logos.uwaterloo.ca.

2 TWEAK and ABTWEAK

In this section, we provide a formal foundation for the rest of the paper, by reviewing the TWEAK plan language, and introducing the ABTWEAK plan language.

2.1 TWEAK

[Chapman 1987] provides a formalization of a least commitment, nonlinear planner, TWEAK, similar to but predating the recent work on the SNLP planner [McAllester and Rosenblitt 1991]. TWEAK extends STRIPS [Fikes and Nilsson 1971] by allowing for

1. a partial temporal ordering on the operators in a plan,
2. partial constraints on the binding of variables (*codesignations*) of the operators.

A TWEAK plan thus represents a space of STRIPS plans: all totally ordered, fully ground plans that satisfy the ordering and codesignation constraints.

Formally, a TWEAK system is a pair $\Sigma = (L, O)$. L is a restricted language consisting of a finite number of predicate symbols, infinitely many constant and variable symbols, and negation. The set of *terms* of L is the constants unioned with the variables. The set of *atoms* is all expressions of the form

$$P(x_1, \ldots, x_n),$$

where P is an n-ary predicate and the x_i are terms. The *ground* atoms are the atoms where all terms are constants. The *literals* (also called *propositions*) include all atoms and their negations. Further, for any literal p, $\neg\neg p$ is equivalent to p. O is a set of operator templates (referred to simply as operators). Associated with each operator a is a set of precondition literals, P_a, and effect literals, E_a.

Chapman did not give a formal definition of a TWEAK plan [Chapman 1987]. Because this concept is very important in defining a number of others later in the paper, we formally define it as follows.

Definition 2.1 *A plan* Π *is a triple* $(\text{Operators}_\Pi, \prec_\Pi, \text{Co\&Nonco}_\Pi)$*, where*

- Operators_Π *is a set of operators, which are copies of operator templates, in which the template variables have new, unique names.*
- $\prec_\Pi$*, the temporal constraints, is a binary relation on* Operators_Π *such that the transitive closure of* $\prec_\Pi$ *is a partial order (irreflexive, asymmetric, transitive),*
- $\text{Co\&Nonco}_\Pi$*, the codesignation constraints, is a pair of binary relations on the terms in* L*, with* $\approx_\Pi$ *being the positive codesignations, and* $\not\approx_\Pi$ *being the non-codesignations.* $\approx_\Pi$ *is an equivalence relation.* $\text{Co\&Nonco}_\Pi$ *is further constrained so that*

Consistency: *if* $(x \approx_\Pi y)$*, then it is not the case that* $(x \not\approx_\Pi y)$*, for any terms* x *and* y*, and*

Uniqueness of Names: *it is not the case that* $(c \approx_\Pi d)$*, for any 2 constants* c *and* d*.*

The plan subscripts to $\approx$, $\not\approx$, and $\prec$ will be dropped if the plan to which these relations refer is clear from context. In addition, we will use $\neg$ before an expression to mean "it is not the case that." For example, $\neg(a \approx b)$ is to be read "it is not the case that $(a \approx b)$." Note that $\neg(a \approx b)$ is *not* equivalent to $(a \not\approx b)$. Further, we extend $\approx$ and $\not\approx$ to literals in the standard way. That is, two literals codesignate if the predicates have the same number of arguments, and all corresponding arguments in the two literals codesignate. Likewise, two literals non-codesignate if the predicates have a different number of arguments, or if one or more of the corresponding arguments non-codesignate.

With the above definition, we can now restate several terminologies used in [Chapman 1987] formally. A *complete plan* Π is a plan where $\prec_\Pi$ is a linear ordering on Operators_Π, and $\text{Co\&Nonco}_\Pi$ is such that every variable in every operator of Operators_Π codesignates with some constant. A plan *completion* of Π refers to any complete plan Π' that satisfies the constraints of Π.

An operator α *asserts* literal p if there exists $q \in E_\alpha$ such that p and q codesignate, and *denies* p if its negation is asserted. A *state* is defined as a set of ground atoms in L. An input problem is taken to be a pair $\rho = (I, G)$, where I is a state (the *initial* state), and G is a set of propositions, (the *goal*).

For simplicity, the goal G can be represented by a special operator $\mathcal{G}$, where $P_\mathcal{G} = E_\mathcal{G} = G$. The initial state I can likewise be viewed as a special operator $\mathcal{I}$, with $P_\mathcal{I} = \emptyset$ and $E_\mathcal{I} = I$. These two operators will be an element of each plan Π, under the constraint that, for every other operator $\alpha \in \text{Operators}_\Pi$, $(\mathcal{I} \prec_\Pi \alpha)$ and $(\alpha \prec_\Pi \mathcal{G})$.

A complete plan implicitly defines a sequence of states obtained by applying each fully instantiated operator in the sequence specified by the ordering relation. A complete plan is *correct* if for every operator, every precondition is satisfied in the state in which the operator is applied. A complete plan *solves* a problem if the plan is correct, and the goal is satisfied in the final state. A plan Π (not necessarily complete) is correct if every completion is correct, and Π solves a problem if every completion solves the problem. An equivalent definition of correctness, which we have adopted in our algorithm (see Appendix C), is Chapman's *Modal Truth Criterion (MTC)* [Chapman 1987], that has the additional advantage that it is polynomially checkable.

In the balance of the paper, we will apply Chapman's modal necessity operator ($\Box$) to propositions that are true, (or constraints that hold) in *every* completion of a given plan. Likewise, the possibility operator ($\Diamond$) denotes that a proposition (or constraint) is true in *some* completion of a given plan. Note that since plan completions are obtained by adding operators and constraints, a constraint (e.g., $a \prec b$) is necessarily true in Π if it is an element of one of the constraint sets of Π, and possibly true if its inverse (e.g., $b \prec a$) is not an element of one of the constraint sets.

2.2 AbTweak

In Abstrips, Sacerdoti developed an elegant means for generating abstract problem spaces by assigning criticality values (an integer between 0 and k, for some small k) to preconditions, and abstracting at level i by eliminating all preconditions having criticality less than i. This is formalized as follows.

A k level AbTweak system is a triple $\Sigma = (L, O, crit)$, where L and O are defined as for Tweak, and *crit* is a function mapping preconditions to non-negative integers:

$$crit : \bigcup_{\alpha \in O} P_\alpha \rightarrow \{0, 1, \ldots, k-1\}.$$

Intuitively, *crit* is an assignment of criticality values to each proposition appearing in the precondition of an operator. Let α be an operator. We take ${}_iP_\alpha$ to be the set of preconditions of α which have criticality values of at least i:

$${}_iP_\alpha = \{p \mid p \in P_\alpha \ and \ crit(p) \geq i\}.$$

${}_i\alpha$ is operator a with preconditions ${}_iP_\alpha$ and effects E_α. Let the set of all such ${}_i\alpha$ be ${}_iO$. This defines a TWEAK system on each level i of abstraction:

$${}_i\Sigma = (L, \ {}_iO).$$

We extend this notation to plans. Given plan Π, ${}_i\Pi$ is plan Π where the operators are all drawn from ${}_iO$.

As with Abstrips, AbTweak performs its search level by level. At each level i, a correct plan Π' at level $i+1$ is taken as input. The process of taking a plan at a higher level and transforming it into a correct one at the current level is called *refinement*. The operators in Π' are converted to their corresponding ones at level i. Then a search is performed to refine Π' to a correct solution at the current level, where the correctness criterion is based on Chapman's MTC [Chapman 1987]. The iteration starts at the highest level with an initial plan $(\mathcal{I}, \mathcal{G})$, and terminates when a correct plan is found. The algorithm can be found in Appendix C.

In the following sections, we explore how to improve the search efficiency using several different methods: by protecting a subset of abstract conditions achieved so far, by biasing search toward deeper levels in a hierarchy, and by appropriately exploiting domain-dependent heuristics.

3 Search Control Strategies in AbTweak

3.1 Protection

Plan construction is a goal-directed process. Given a set of goals, a planner selects a goal or an operator precondition that has not yet been achieved, and attempts to find operators that can "establish" it. The need for goal protection arises because an operator for achieving one goal may inadvertently undo or reachieve an already achieved goal. When a goal or a precondition is protected, no subsequent operators are allowed to be added to a plan that undo or reachieve this goal.

We first formalize the notion of establishment of a goal or precondition. Suppose that we have a correct plan produced by Tweak. Then in this plan, every precondition p of every operator must have an operator before it that *asserts* p. More precisely,

Definition 3.1 *Let* Π *be a plan. Operator* α *establishes proposition* p *before operator* β *(*$\text{Establishes}(\alpha, \beta, p)$*) if and only if*

1. $\Box(\alpha \prec \beta)$,
2. $\exists u \in E_\alpha . \Box(u \approx p)$, *and*
3. $\forall \alpha' \in \text{Operators}_\Pi, \forall u' \in E_{\alpha'}$, *if* $(\alpha \prec \alpha' \prec \beta)$, *then* $\neg(u' \approx p)$.

The final condition ensures that α is the last such operator that asserts p. Notice that this is a restatement within the Tweak representation of Sussman's *ontological structure* [Sussman 1973], Tate's *goal structure* [Tate 1977], and *causal links* [McAllester and Rosenblitt 1991].

Notice that in a Tweak plan the operators are partially ordered. Thus, it is possible that a precondition may have several establishers. If a precondition has more than one establisher, we call the set of all establishment relations in the plan the *establishment set*[2].

Adapting the definition from [McAllester and Rosenblitt 1991], we say that c is a *possible threat* to $\text{Establishes}(a, b, p)$ if c is an operator other than a or b that is possibly between a and b and that either possibly asserts (a *positive* threat) or possibly denies p (a *negative* threat). Thus, given a possible threat, in some completion, c occurs between a and b and asserts or denies p. If c is necessarily between a and b, and either necessarily asserts or necessarily denies p, then c is a necessary threat. By definition, given a necessary threat, in every completion c occurs between a and b and asserts or denies p. Relating this to Chapman's terminology [Chapman 1987], a possible negative threat is also called a *clobberer*.

Possible threats can be made *safe* [McAllester and Rosenblitt 1991] by adding ordering constraints making c be either before a or after b (demotion and promotion, [Chapman 1987]), or by adding non-codesignation constraints making c not assert or deny p (separation). If

[2] A similar notion was defined by [Kambhampati 1992].

a plan is obtained from Π by adding new operators and/or constraints, then it is called a *successor* plan of Π.

Definition 3.2 (Protection) *Precondition p of β is protected in plan Π if and only if in no successor plan of* Π every *establishment in the establishment set of p for β is necessarily threatened by an operator.*

In other words, any successor plan containing an establishment set of a protected goal in which every establishment is necessarily threatened (i.e., a protection *violation*), is pruned from the search tree.

The level by level protection of precondition establishments used in AbTweak is called *monotonic protection.*

Definition 3.3 (Monotonic Protection (MP)) *A planner enforces* monotonic protection *if, for every plan Π and every level i, every precondition of an operator in Π at level i is protected during refinement at every lower level.*

A refinement of an abstract plan obtained with the monotonic property enforced is called a *monotonic refinement* [Knoblock, Tenenberg and Yang. 1991]. We use the term monotonic since once an operator or constraint is added at a particular abstract level, it is never removed during refinement at less abstract levels. Note that monotonic protection does not preserve *every* establishment relation constructed so far; rather, it is guaranteed that at least *one* of the abstract establishments for each precondition is protected. It also means that while planning, no establishment relations *on the current level* are protected. In the rest of the paper, the term AbTweak *with MP* refers to the abstract planner AbTweak augmented with the addition of monotonic protection at every level. According to the definition, AbTweak with MP *prunes* all plans with monotonic violations. The question emerges as to whether the resulting planner is still complete. The following theorem answers the question. It states that AbTweak with MP does not lose the completeness of search, and that this property holds regardless of the specific abstraction hierarchy used.

Theorem 3.4 *If a solution plan exists, then a search path to one such plan also exists in the search space of* AbTweak *with MP.*

Proof: (Sketch)

Consider a version of the Tweak planner which protects *all* establishments constructed so far. Let's denote this planner by Tweak' (Tweak' is a variant of Snlp[Barrett and Weld 1992]). It can be shown by induction on the number of operators in a plan that Tweak' is complete. The monotonic protection used in AbTweak protects only a subset of the establishments protected by Tweak'. Thus, given the completeness of Tweak', AbTweak must also be complete. □

Based on this theorem, AbTweak with MP can eventually find a solution if one exists.

3.2 The LEFT-WEDGE Search Control Strategy

So far, we have identified a universal property, the monotonic property, for all abstraction hierarchies. This property guarantees that a sequence of monotonic refinements exists for any hierarchy. However, the way in which one goes about searching for such a sequence is not obvious. Many different search control strategies exist, each resulting from a different way of coordinating search in the abstract plan space and search during plan refinement.

An intuitively obvious choice of control is to use a strategy that is complete on each level of abstraction. This is especially appealing, since it is not difficult to specify complete control strategies for TWEAK, either using a complete state-space search procedure such as A*, breadth-first search, or the procedure provided by [Chapman 1987]. Using this approach, if a plan is formed on abstraction level i, then it is passed down to the level below. At level $i-1$, all the conditions of criticalities no less than $i-1$ are planned for. The process continues, until either a correct plan is formed at the base level, or it is found that a plan cannot be made correct at a level. Then the planner backtracks to the level immediately above the current one, and tries to find an alternative solution.

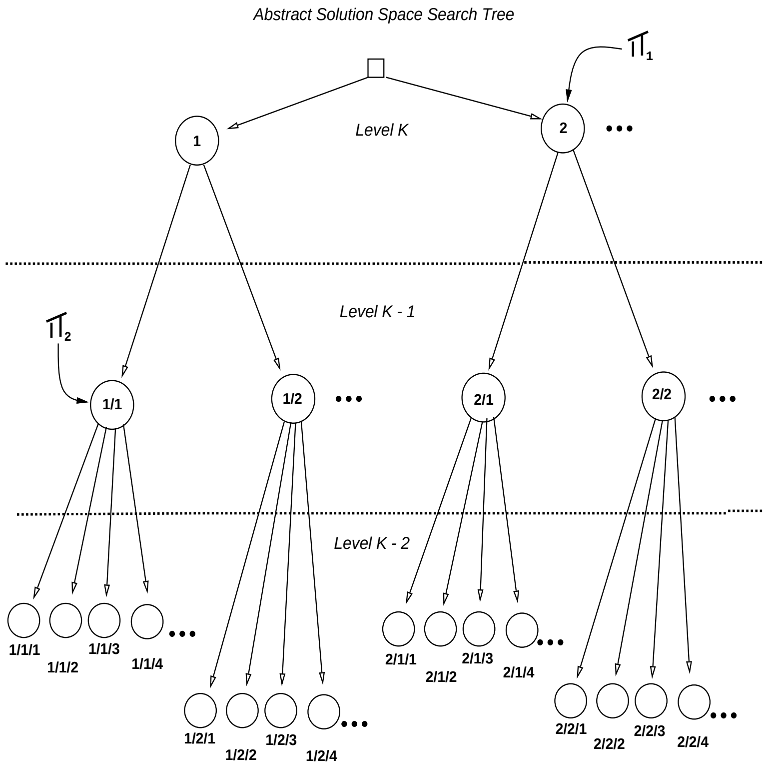

Figure 1: Representing the abstract solution space

The fact that each level is complete may lead one into believing that the above control structure is also complete. Unfortunately, this is not the case in general. The reason is that a complete search strategy for any given level is only guaranteed to find a single solution at that level. But this first solution might not be monotonically refinable. Incompleteness might result if either searching for a refinement never terminates, or the strategy does not search the space of alternative abstract solutions.

ABTWEAK's search strategy does not suffer from these drawbacks, but rather interleaves its effort between expanding *downwards* by refining abstract solutions to lower level ones, and *rightwards* by finding more solutions at each particular level of abstraction. When no preference is given to either direction of growth, the interleave between the two expansions is similar to Cantor's complete enumeration of rational numbers (see, for example, [Dodge 1969]). The fact that solutions to solvable problems will eventually be found follows directly from Cantor's proof on the completeness of diagonal enumeration.

However, the degree in which a search strategy tends to favor either direction of growth is an important aspect governing search performance. In Figure 1 the abstract solution search space is shown. This figure shows the relationship between search within an abstract level and search across abstract levels. Each node in the figure represents a solution, that is, a plan, found within a particular abstraction level. It is possible that there may exist multiple solutions within a particular level of abstraction. The leftmost solutions shown in Figure 1 represent the first or "simplest" solutions found within that abstract level. Subsequent, more complex solutions found appear in a left to right fashion. In Figure 1, the solution nodes are labeled such that the abstraction ancestry of that solution can be seen. For example, at level $k-1$, the leftmost node is labeled "1/1", indicating that this is the first solution found at level $k-1$, and this plan is a refinement of the first solution at level k.

Breadth-first search shows no preference for concrete level plans over abstract plans. However, all solutions are at the concrete level, and therefore, effort expended toward exploring alternatives at the abstract level is solely for the sake of completeness. This completeness, however, exacts a heavy computational cost, since computational effort expended in searching at the abstract level is at the expense of further effort to refine plans. We introduce an alternative strategy, which we call LEFT-WEDGE, that allows for plunging more deeply into the search space along the "leftmost" frontier, yet still remaining complete. The motivation for this strategy rests on our intuition that the intent of criticalities is to impose an order on the solution of subgoals. A well chosen abstraction hierarchy would be one in which the choices made at the abstract level serve as fixed constraints throughout the planning, *and never need to be retracted.* Thus, a solution strategy that exploits such a hierarchy would prefer expanding plan refinements over plan alternatives, (downwards to rightwards) under the assumption that correct initial choices of abstract plan steps will rarely require the refinement of abstract alternatives.

As an example of LEFT-WEDGE, consider Figure 1 again. Plan Π_2 at level k is expanded with the same preference as plans 1/1/1 through 1/1/4. In this way, the search space grows deeper much more quickly on the leftmost branches than the right, with the frontier taking on the characteristic *left wedge* shape for which the strategy is named. Assuming that each plan Π has a certain cost, $cost(\Pi)$, the LEFT-WEDGE strategy can be implemented as follows.

The search strategy used by ABTWEAK always selects a plan Π with minimal cost to refine next. The LEFT-WEDGE heuristic can be implemented by modifying the cost function as follows. For each plan Π at a certain level i, an additional value is subtracted from the cost function, where the value depends on the level of abstraction i:

$$newcost(\Pi) = cost(\Pi) - lw(crit(\Pi))$$

The function $lw(i)$ is any monotonically decreasing function of i, such that $lw(k-1) = 0$, for a hierarchy with k levels of abstraction. Then search in ABTWEAK is guided by the *newcost* function; a plan with the minimal *newcost* value is always selected next for refinement. Experiments using this strategy, and comparisons with breadth-first search, are described in Section 4.

3.3 Using Primary Effects in ABTWEAK

In ABTWEAK it is also possible to make a distinction between *primary effects* and those effects which are not primary for each operator [Sacerdoti 1974, Minton 1990, Knoblock 1991]. For example, in the robot task planning domain to be described, the primary effect of pushing a box from one room to another is just that the box be in the destination room. Any additional effects, such as that the robot is also in the destination room are considered secondary. This distinction indicates to the planner that to move the robot around, the push-box operator should *not* be used. It should be used only for moving boxes around, not for any side-effects that might result.

The application of primary effects corresponds to reducing the branching factor of the search space, since operators are only considered as plan steps when the current subgoal is a primary effect. In fact, without the application of this heuristic, many trivial problems cannot be solved by ABTWEAK.

In our use of ABTWEAK we have found that often primary effects combine quite naturally with the use of abstraction. The following theorem shows that if all predicates associated with primary effects of an operator are assigned criticality values at least as high as that for the side effects, then the completeness of abstract planning is preserved.

Theorem 3.5 (Primary Effect Theorem) *Suppose that an abstraction hierarchy satisfies the condition that, for every operator α, for every primary effect p of α and side effect q of α,* $\text{crit}(p) >= \text{crit}(q)$. *If* TWEAK *with primary effects can find a solution plan to a problem, then using the hierarchy,* ABTWEAK *with MP and primary effects can also find a solution to the problem.*

Proof: First, consider a two level system. We want to show that if a solution exists in which every operator achieves a precondition or goal using a primary effect, then the abstract version of the plan also has the property if all primary effect predicates have the same or higher criticality values. Let Π be a plan in which every operator achieves a precondition or goal using a primary effect. In its abstract version ${}_i\Pi$ we can remove all unnecessary operators and those operators that achieve a precondition using one of its side effects. Let the resulting plan be Π'. We want to show that Π' is also correct. Suppose Π' is incorrect. This must be due to the removal of an operator which only achieves a precondition q as a side effect. Let this operator be α. Since in Π the operator α achieves other preconditions using both primary and side effects, it must be the case that one of the preconditions p that is removed during the abstraction process is one of α's primary effects. Since p is removed while q is not, it must be true that $crit(p) < crit(q)$. However, this contradicts with requirement that all primary effect predicates receive higher criticality values.

It is easy to extend the theorem to more than two levels. □

4 Empirical Tests

Above we have described the monotonic property for search control within a level of abstraction, LEFT-WEDGE as a control strategy for search across multiple levels of abstraction, and the use of primary effects in ABTWEAK. While we are able to show that both methods guarantee completeness for ABTWEAK, it is difficult, if not impossible, to conduct a theoretical analysis of their effectiveness in search reduction. An alternative then, is to test ABTWEAK empirically.

Both ABTWEAK and TWEAK have been implemented in Allegro Common Lisp, on a Sun SparcStation II. A detailed explanation of the implementation can be found in [Woods 1991]. In the implementation, we have paid special attention in making sure that the two planners share key subroutines, so the comparison in their performances can be fair. We have also conducted experiments in two domains, the Towers of Hanoi domain, and a robot task planning domain from ABSTRIPS[Sacerdoti 1974]. The full operator descriptions for the Towers of Hanoi domain appear in Appendix A. Appendix B lists the operators and language used in the robot task planning domain.

In choosing our experimental domains and abstraction hierarchies in these domains, we are particularly interested in the following problem features:

Inter-level Subgoal Interaction; Can plan refinements at a lower level add or delete a higher level establishment?

Conjunctive Goals; Does a domain contain conjunctive goals in the planning problems?

Solution Length; How do the solution lengths increase with the number of goals?

4.1 Testing the Towers of Hanoi Domain

4.1.1 Domain Description

For the first set of experiments, we have chosen to use the Towers of Hanoi (TOH) domain. In the 3-disk version of the domain, four predicates are used to describe the states. These are `IsPeg, OnSmall, OnMedium, OnLarge`. If a hierarchy is built based on assigning a distinct criticality value to each of the predicates, then 24 different hierarchies exist. Out of the 24 hierarchies, only one has been extensively tested in the past with linear, abstract planners [Knoblock 1991]. This hierarchy corresponds to assigning criticality values in the following way: $crit(\texttt{ISPEG}) = 3, crit(\texttt{OnLarge}) = 2, crit(\texttt{OnMedium}) = 1, crit(\texttt{OnSmall}) = 0$. In order to fully investigate the effects of different control strategies on search efficiency as a function of the hierarchy used, we have tested all possible permutations of the hierarchies. For ease of exposition, we use ILMS to represent the above hierarchy. Similarly, SMLI represents the hierarchy with the reverse order of criticality assignment.

This domain is of interest to us because of the problem features we listed above. The planning problem typically consists of conjunctive goals, and the solution length increases exponentially with the number of goals. Many non-hierarchical planners consider this problem hard to solve. In addition, there exists a clear distinction of different degrees of importance among the domain conditions, and the interactions between these conditions can be carefully controlled to generate meaningful results. For example, in the ILMS hierarchy no low-level action can achieve or deny a high-level subgoal. To see this, observe that at the `OnLarge` level only the `OnLarge` goals are achieved. At the next level down, while achieving the `OnMedium` subgoals a `MoveMedium` operator could not add or delete an `OnLarge` subgoal. Thus, while refining an abstract plan there is no monotonic violation. This *inter-level interaction* feature, however, can be easily varied, by using a different hierarchy. If we use the IMLS hierarchy, then in the refinement of an `OnMedium`-level abstract plan new `MoveMedium` operators will be introduced. These operators will cause the higher level establishments of the `OnMedium` subgoals to be threatened.

4.1.2 Evaluating LEFT-WEDGE

Comparing with breadth-first search Our first test compares ABTWEAK using breadth-first across abstraction levels, and LEFT-WEDGE, respectively. Both versions of ABTWEAK are associated with MP, and used the most intuitive hierarchy ILMS. The results are shown in Table 1. It is evident from the table that search time and space is greatly reduced when using the LEFT-WEDGE strategy. For comparison purposes, in the table we also included data obtained using two other hierarchies which are small variations of the ILMS hierarchy.

Comparing with depth-first search ¿From the above data it is evident that the LEFT-WEDGE strategy easily defeats breadth-first search. We now show that LEFT-WEDGE can

Hierarchies	Expanded		CPU seconds	
	Left-Wedge	Breadth-first	Left-Wedge	Breadth-first
ILMS	57	471	38.2	252.2
IMLS	86	166	38.1	60.4
MILS	94	295	81.3	235.9

Table 1: Comparing AbTweak with and without Left-Wedgein the TOH domain.

Preconditions	Effects	Cost
`Move-Medium-and-Large`(x,y)		

`OnLarge(x)`, `OnMedium(x)`
`not OnSmall(x)`, `not OnSmall(y)`
`IsPeg(x)`, `IsPeg(y)`
	`OnLarge(y)`, `OnMedium(y)`
	`not OnLarge(x)`, `not OnMedium(x)`
		2

Table 2: An operator in extended TOH domain.

avoid a major pitfall of depth-first search across different levels of abstraction.

To see this, consider an extended TOH domain where we add an additional operator `Move-Medium-and-Large`(x, y), for simultaneously moving both the medium disk and the large disk from peg x to y. This operator is shown in Table 2. With this extension, and with the hierarchy ILMS, there are now two interesting abstract plans at `OnLarge` level. The first one is the usual `MoveLarge(peg1,peg3)`. This abstract plan refines to a 7-step solution plan as obtained in the original TOH domain.

The second abstract solution, `Move-Medium-and-Large(peg1,peg3)`, has a cost of 2. Although the cost of this abstract plan is higher than the first one, the plan refines to a less costly concrete level plan:

1. `MoveSmall(peg1,peg2)`
2. `Move-Medium-and-Large(peg1, peg3)`
3. `MoveSmall(peg2,peg3)`

This plan has a cost of 4.

Note that if depth-first search was used across abstraction levels, then the cheapest abstract solution `MoveLarge(peg1,peg3)` would have been committed to by the planner, resulting in a plan which is more costly to execute and to find. This phenomenon would occur even if the hierarchy satisfied the so-called *downward solution property* [Bacchus and Yang 1994], whereby every abstract solution could be monotonically refined to the next lower level. This worst-case phenomenon was first noticed by Backstrom and Jonsson[Backstrom and Jonsson 1995].

Hierarchy	Expanded		CPU seconds	
	LEFT-WEDGE	Depth-first	LEFT-WEDGE	Depth-first
ILMS	13	142	1.1	70.3

Table 3: Comparing LEFT-WEDGE and depth-first search in extended TOH domain.

Hierarchies	Expanded		CPU seconds	
	With MP	Without MP	With MP	Without MP
ILMS	57	57	38.2	30.5
IMLS	86	1009	38.1	1470.5
MILS	94	1008	81.3	2278.8

Table 4: Comparing ABTWEAK with and without MP, in the original TOH domain.

With its LEFT-WEDGE control strategy, ABTWEAK can avoid this problem. During plan refinement, LEFT-WEDGE interleaves between the refinement of different abstract plans. Although it pays more attention to the lowest-cost abstract plan, the rest of the abstract plans are also given various degrees of attention according to their ranking. In the extended TOH example, ABTWEAK could find the concrete-level solution containing `Move-Medium-and-Large` before encountering the original 7-step solution. Table 3 compares the performance of ABTWEAK using LEFT-WEDGE and depth-first search, respectively.

4.1.3 Evaluating MP

Original Towers of Hanoi Domain When comparing LEFT-WEDGE with and without using MP (see Table 4), the results indicate that, in general, using LEFT-WEDGE with MP works much better than not using MP, especially for those hierarchies which are variations of the best one, ILMS. In the IMLS hierarchy, while there are monotonic violations, the protection strategy of ABTWEAK could quickly terminate the search path containing such violations, resulting in a smaller search tree. For ILMS, however, we notice that not using MP turns out to be slightly better than using MP in terms of CPU time. This is because when using the best hierarchy ILMS there is no monotonic violation. Thus, the CPU time saving does not justify the additional cost in safe-guarding the higher level establishments.

Figure 2 shows the overall result of the comparison in the Towers of Hanoi domain. In this test, ABTWEAK was run with MP and the LEFT-WEDGE control strategy, in the hierarchy ILMS. The figure contrasts TWEAK with ABTWEAK, in terms of the number of states expanded as a function of the solution length. The data in the figure are generated and averaged based on planning with a fixed initial state, and 26 different goal states in this domain. It is clear that AbTweak dramatically outperforms Tweak when the solution length increases.

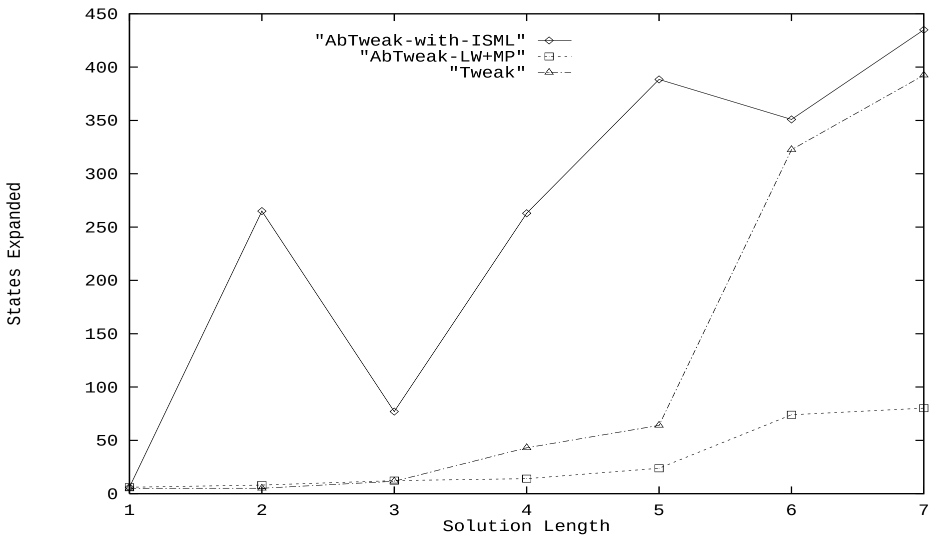

Figure 2: Comparing Tweak with AbTweakin the original TOH domain.

The same figure also compares the performance of the two planners, but using a poorly chosen criticality assignment, namely ISML. The result is that with this hierarchy, AbTweak using both MP and Left-Wedge performs the worst. This result leads us to the conclusion that an arbitrary abstraction hierarchy is not necessarily good. To improve performance using abstraction, one has to be very careful in the choice of both the abstraction hierarchy and the search strategies guiding the abstract search. This result serves as a strong motivation for much of the current research in finding syntactic criteria for good abstraction hierarchies. Examples of such current work can be found in [Bacchus and Yang 1994], [Knoblock, Tenenberg and Yang. 1991] and [Knoblock 1991].

Modified Towers of Hanoi Domain To test the MP thoroughly, we are also interested in experimenting in a domain where no hierarchies with the downward-refinement property (DRP) can be found. With this in mind we have designed a modified Towers of Hanoi domain, where the large and medium disks are replaced by two medium disks, **`Medium1`** and **`Medium2`**. It is further assumed that a medium disk cannot be moved into a peg where another medium disk is currently located. The operators of this domain include the usual ones for moving each individual disk, and one for moving both medium disks from one peg to another, as long as the two medium disks are initially on the same peg.

The operators for moving a medium disk and for moving two medium disks are listed in Table 5.

Preconditions	Effects	Cost
MoveMedium1(x,y)		
`IsPeg(x)`, `IsPeg(y)` `OnMedium1(x)`, `not OnMedium2(y)` `not OnSmall(x)`, `not OnSmall(y)`	`OnMedium1(y)`, `not OnMedium1(x)`	1
MoveMediums-1-2(x,y)		
`IsPeg(x)`, `IsPeg(y)` `OnMedium1(x)` `OnMedium2(x)`, `not OnSmall(x)` `not OnSmall(y)`	`OnMedium1(y)`, `not OnMedium1(x)` `OnMedium2(y)`, `not OnMedium2(x)`	2

Table 5: Operators in the modified TOH domain.

Hierarchies	Expanded		CPU seconds	
	With MP	Without MP	With MP	Without MP
IMMS	27	69	1.5	4.7
ISMM	70	1001	5.3	235.2
IMSM	84	1001	9.6	283

Table 6: Comparing AbTweak with and without MP, in the modified TOH domain.

This domain has three natural hierarchies, corresponding to the three different and non-symmetric criticality assignments to predicates `OnMedium1(x)`, `OnMedium2(x)` and `OnSmall(x)`. We call these hierarchies IMMS, ISMM, and IMSM, respectively.

As noted before, a feature of this domain is that none of the intuitive hierarchies has the downward refinement property (DRP), whereby every abstract solution can be monotonically refined to the next lower level. An abstract plan for moving any individual medium disk would result in a deadlock at a lower level, and similarly moving the small disk from peg1 directly to peg3 is not compatible with any operator for moving the medium disks. This feature complements the original TOH domain, in which the hierarchy ILMS satisfies the DRP.

The results of running AbTweak with and without MP are shown in Table 6. Both versions of AbTweak utilized the Left-Wedge control strategy. The results of the table clearly shows that, in the absence of a hierarchy that satisfies the DRP, using MP in this domain outperforms not using MP.

4.1.4 Summary: insights gained from the experiments

Our experience so far could be summed up as follows:

1. A planner searching in an abstraction hierarchy should balance between two distinct search behaviors, the generation of alternative solution plans within a level, and the pursuit of refinement of a given abstract solution across different levels. Any arbitrary commitment to either strategy would have some undesirable effects. LEFT-WEDGE strategy can be seen as a middle ground between the two, offering an intermediate approach to counter the shortcomings of both depth-first search and breadth-first search.

2. A function of the abstract solutions is to constrain the scope of refinements at lower levels of abstraction. Monotonic protection (MP) can be seen as an implementation of this function. The effectiveness of this protection strategy is demonstrated in the TOH domain and the modified TOH domain, where we have seen that using MP has resulted in better performance than not using MP.

4.2 Robot Task Planning Domain

We have also run 50 tests of ABTWEAK with the hierarchy in Table 7. In this domain there is a robot that can move between a number of connected rooms. Between any two rooms there may be a door, which can be open or closed. In addition, there are also boxes which the robot can push from one location to another. This domain is chosen because there is a large number of different conjunctive goals, which can be obtained by specifying the final locations of different boxes and the robot. Unlike the TOH domain the solution lengths for most problems that we tested increase *linearly* with the number of goals. In addition, using the abstraction hierarchy there are plenty of monotonic violations. For example, the `open-door` action subgoals on the `Robot-At` subgoal, which could potentially introduce threats to the higher level establishments.

In the tests, we used the primary effects heuristic. Without it many simple problems were not solvable within the given time bounds by any of the planners that we tested. Five different planning problems of each length were solved using TWEAK, ABTWEAK with breadth-first, and ABTWEAK with both MP and the LEFT-WEDGE control strategy. Both planners in this domain used primary-effects as a domain-dependent heuristic to restrict the branching factor of search. Figure 3 shows the number of states expanded as a function of solution length. It is clear that ABTWEAK with MP and the LEFT-WEDGE control strategy dramatically outperforms both TWEAK and ABTWEAK with only the breadth-first control strategy.

5 Relation to Other Work

We consider ABTWEAK as a clean theoretical and experimental tool to study abstraction and search. In this section, we discuss how our work is compared to other closely related efforts in planning and abstraction.

Criticality	Predicate
4	`Box-Inroom` and other sort-type predicates.
3	`Robot-Inroom`
2	`Box-At`
1	`Robot-At`
0	`Open`

Table 7: Criticality assignments for the Robot Task Planning Domain.

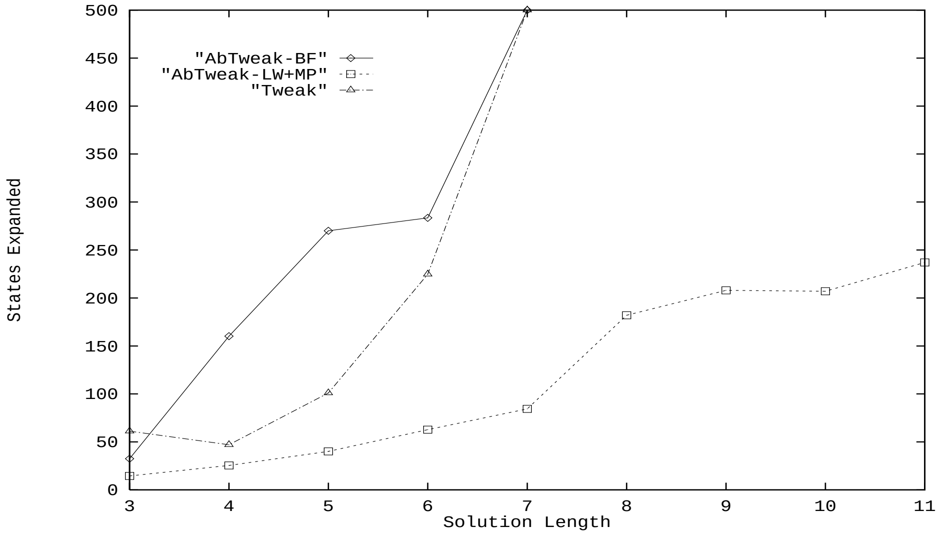

Figure 3: Comparing TWEAK with ABTWEAK in the robot task planning domain.

5.1 Relationship to SIPE

SIPE is an abstract planning system based on hierarchical task networks (HTN) [Wilkins 1984]. HTN planners represent actions as *schemas*. Each schema has a set of preconditions and effects, just like an ABTWEAK operator. Unlike ABTWEAK, each schema also specifies other information useful for planning. An important piece of information specifies how one action can be realized through many subactions. For example, a schema for fetching an object may consist of the subactions for sensing and picking up the object. The subactions are associated with the ordering constraints and the binding constraints imposed on variables. In addition, an action schema specifies the range of conditions in which they are to be protected during planning.

As pointed out by Wilkins, the search component of SIPE performs two types of planning activities. SIPE either expands an action node by replacing it by its subactions at a different level of detail, or it inserts subactions at the same level of detail. The former is an expansion across different *abstraction levels*, while the latter across different *planning levels*. As an example, consider planning to get a robot from one room to another. Suppose the planner already has a plan for crossing different rooms. If lower level conditions, such as changing the location of the robot within a room, are introduced into the plan, then the plan is expanded across abstraction levels. However, the process of inserting more operators to go from one location to the next in the same room is plan expansion at different planning levels.

Comparing SIPE to ABTWEAK, we see a striking similarity in their search behavior. Although ABTWEAK does not use action schemas to expand a plan, it implicitly accomplishes the process of expansion via planning from the first principles. In other words, the plan expansion process of SIPE at different planning levels is paralleled by the plan refinement process of ABTWEAK within a given abstraction level. Both systems work at different levels of abstraction, and compose plans in a top-down manner. Furthermore, both systems protect abstract level goal conditions during planning, although the protection is done under different names — In ABTWEAK an interval of protection is called an *establishment relation*, while in SIPE it is known as a *protect-until* slot.

Based on the above discussion, we can now summarize the major similarities between ABTWEAK and SIPE as follows:

Monotonic Protection The action schemas in SIPE specify a set of *protect-until* slots. Associated with each slot is a condition to be protected during planning, and each slot specifies the range where the condition is to be maintained. As we noted above, it is not hard to see that the protect-until construct of SIPE is very similar to the *establishment relations* in ABTWEAK. Both systems use this information to control search. In particular, SIPE periodically applies a set of verification routines, known as *Critics*, to a plan. If any of the protected conditions is found to be violated by the Critics, then SIPE prunes that plan from the search space. In essence, this is exactly the same operation as the monotonic protection used in ABTWEAK.

Primary Effects To improve its search efficiency, SIPE specifies the main purpose of each action schema, in a manner similar to the primary effect heuristic used in ABTWEAK. In particular, each action schema has one or more main *purposes*, which are the primary effects of the action. During planning, the schema is inserted in a plan only for its stated purpose. All other effects are derived through a special causal reasoner and domain axioms; these being the *side* effects of the action. The goal of restricting the effects of actions in this way, is exactly the same as using primary effects in ABTWEAK: they are both for cutting down the branching factor of search, and reducing the total amount of time used in reasoning about side effects.

Left-Wedge Search To maintain its heuristic adequacy, SIPE uses a depth-first search method for selecting the next plan to expand. The type of depth-first search used in

SIPE is guided by the abstraction levels, in a manner similar to our LEFT-WEDGE strategy in ABTWEAK.

Given these similarities, the results obtained from experimenting with ABTWEAK can directly benefit the use of abstraction in SIPE.

5.2 The Ordered Monotonic Property

A closely related work to ABTWEAK is Knoblock's ALPINE system [Knoblock 1991], which is an extension of Sacerdoti's ABSTRIPS and Siklossy and Dreussi's LAWLY[Siklossy and Dreussi 1973]. ALPINE differs from ABTWEAK in two significant ways. First, ALPINE is a strictly *linear* planning system, and has no capability for nonlinear planning. Second, Knoblock's focus with ALPINE was in automating the generation of abstraction hierarchies, and not on how to make efficient use of a given hierarchy, as was our intent. In ALPINE, all generated hierarchies satisfy the *ordered monotonicity* property (OM), which is defined in [Knoblock 1991, Knoblock, Tenenberg and Yang. 1991] roughly as

> *Every* refinement of an abstract plan leaves *all* high-level literals unchanged.

This implies that for any OM hierarchy, it is impossible to generate monotonic violations of abstract preconditions during refinement at lower levels. This property can be guaranteed by a set of syntactic conditions that relate the operator schemas to the literals in the domain language. The syntactic conditions can then be used in the design of an algorithm that generates abstraction hierarchies possessing the OM property [Knoblock, Tenenberg and Yang. 1991]. For example, in the Towers of Hanoi domain, the hierarchy ILMS is ordered. Experiments reported in [Knoblock 1991] demonstrate that in several domains, planning with the abstraction hierarchy generated by ALPINE clearly improves planning efficiency.

Range of Applicability

In comparing ALPINE's ordered monotonicity (OM) property and the monotonic property (MP) method used in ABTWEAK, we note that OM is much stronger than MP, and thus is satisfied by fewer domains. In fact, OM requires that a refinement leave intact *all* higher-level literals, even those that are not part of the abstract plan being refined. Furthermore, this restriction must hold for *every* refinement. In many cases, the OM property is so strong that it can only be satisfied by trivial hierarchies, i.e., the hierarchy often collapses to a single level. In contrast, MP can be applied to every hierarchy, whether they satisfy OM or not. In addition, MP as defined by ABTWEAK can affect higher-level literals, just as long as it does not affect the higher-level literals appearing in the particular abstract plan being refined.

Completeness

If a hierarchy satisfies OM, then a solution path exists on which no violation of high level conditions can occur. However, this does not imply that every search path in the abstract search space leads to a solution. If a search path does not lead to a solution, then backtracking has to be invoked in order to find the solution. However, as we pointed out in Section 3.2, refinement of an abstract solution may take forever without reaching either a dead end or a solution. Therefore, even if a hierarchy satisfies the OM property, performing a depth-first search across abstraction levels still leads to incompleteness. Likewise, both ABSTRIPS and LAWLY are likely to be incomplete because they also perform depth-first search across abstraction levels. One way to solve the incompleteness problem is to apply the LEFT-WEDGE search method used in ABTWEAK. In this way, completeness is retained and search can be made more efficient than a blind breadth-first method.

5.3 Relationship with SNLP

Following the development of ABTWEAK [Yang, Tenenberg and Woods 1991, Woods 1991], McAllester and Rosenblitt [McAllester and Rosenblitt 1991] developed a partial-order planning algorithm, which was later implemented and tested in [Barrett and Weld 1994], that uses goal protection (or causal-link protection) as the central search principle.

SNLP's protection policy differs from ABTWEAK's in two ways. First, in SNLP, *every* established precondition is protected during subsequent planning. In contast, ABTWEAK protects only a subset of important conditions achieved so far. Second, an establishment is protected in SNLP by guarding against not only negative threats, but also positive threats. ABTWEAK, however, only protects against negative threats. Recent studies [Knoblock and Yang 1994] and [Kambhampati, Knoblock and Yang 1994] have shown that, due to these differences, depending on the problem domain, it is possible for one planner to significantly outperform the other.

While SNLP removes threats as they appear, a planner proposed by Peot and Smith [Peot and Smith 1993, Smith and Peot 1993] defers the resolution of some threats to achieve the best performance. Their empirical result supports the intuition that it is not always better to protect all establishments eagerly. This observation is consistent with the monotonic protection approach in ABTWEAK, in that only the most important conflicts with abstract establishments are resolved first. Conflicts with conditions of lower level of importance are resolved later.

6 Conclusions

This research is aimed at developing a test bed for experimenting with abstract planning. The resulting system ABTWEAK is equipped with several heuristics for abstract search, including the use of monotonic goal protections in planning, the LEFT-WEDGE heuristic, and methods of using primary effects in abstract planning. We have also empirically evaluated these methods.

ABTWEAK has already inspired several other important developments in planning. Experimentation with ABTWEAK helped discover an important property, the *Downward Refinement Property*, for ranking abstraction hierarchies [Bacchus and Yang 1994]. Algorithms for automatically constructing abstraction hierarchies are integrated into a system known as HighPoint. The protection strategy in ABTWEAK led to research on the application of constraint satisfaction techniques to conflict resolution system [Yang 1992]. The use of primary effects in ABTWEAK also inspired work on automatically learning primary effects for domain operators [Fink and Yang 1993]. The LEFT-WEDGE strategy was combined with an abstract constraint problem solver for spatial pattern recognition [Woods 1993].

Acknowledgement

We thank Craig Knoblock and the anonymous referees for many useful comments. This work was supported in part by research grants to Qiang Yang, from the Natural Sciences and Engineering Research Council of Canada, and by grants to Josh D. Tenenberg in part from ONR research grant no. N00014-90-J-1811, Air Force - Rome Air Development Center research contract no. F30602-91-C-0010, and Air Force research grant no. AFOSR-91-0108. Steven Woods would like to thank both the Commonwealth Scientific and Industrial Research Organization (CSIRO) Australia, the Department of National Defence (Defense Research Establishment Valcartier) Canada, and NSERC for support of this research.

References

[Bacchus and Yang 1994] Fahiem Bacchus and Qiang Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, Vol. 71, No. 1. pages 43-100, Nov. 1994.

[Backstrom and Jonsson 1995] Christer Backstrom and Peter Jonsoon Planning with Abstraction Hierarchies can be Exponentially Less Efficient. To appear in *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (IJCAI-95), Montreal, Canada, August 1995.

[Barrett and Weld 1992] Anthony Barrett and Dan Weld. Partial order planning: Evaluating possible efficiency gains. Technical Report 92-05-01, University of Washington, Department of Computer Science and Engineering, Seattle, WA 98105, 1992.

[Barrett and Weld 1994] Anthony Barrett and Dan S. Weld. Partial order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1994.

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

[Dodge 1969] Clayton W. Dodge. *Numbers and Mathematics.* Prindle, Weber & Schmidt, Inc., Boston, MA, 1969.

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

[Fink and Yang 1993] Eugene Fink and Qiang Yang. Characterizing and automatically finding primary effects in planning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1374 – 1379, Morgan Kaufmann Publishers, Inc., August 1993.

[Kambhampati 1992] Subbarao Kambhampati. Characterizing multi-contributor causal structures for planning. In *Proceedings of the First International Conference on AI Planning Systems*, pages 116–125, San Mateo, CA, 1992. Morgan Kaufmann Publishers, Inc.

[Kambhampati, Knoblock and Yang 1994] Subbarao Kambhampati, Craig Knoblock, and Qiang Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. Accepted by Artificial Intelligence Journal, special issue on planning and scheduling, 1994.

[Knoblock, Tenenberg and Yang. 1991] Craig Knoblock, Josh Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 692-698. Anaheim, CA, 1991.

[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 and Yang 1994] Craig A. Knoblock and Qiang Yang. Evaluating the tradeoffs in partial-order planning algorithms. In *Proceedings of the Canadian Artificial Intelligence Conference, (AI 94)*, pages 279–286, 1994.

[McAllester and Rosenblitt 1991] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 634–639, San Mateo, CA, 1991. Morgan Kaufmann Publishers, Inc.

[Minton 1990] Steve Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42:363–391, 1990.

[Newell and Simon 1972] Allen Newell and Herbert A. Simon. *Human Problem Solving.* Prentice Hall, Enlewood Cliffs, NJ, 1972.

[Peot and Smith 1993] Mark A. Peot and David E. Smith. Threat-removal strategies for partial-order planning. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pages 492–499, Washington, DC, 1993.

[Sacerdoti 1974] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Sacerdoti 1977] Earl Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier, 1977.

[Siklossy and Dreussi 1973] L. Siklossy and J. Dreussi. An efficient robot planner which generates its own procedures. In *Proceedings of the 3rd IJCAI*, pages 423–430, 1973.

[Smith and Peot 1993] David E. Smith and Mark A. Peot. Postponing threats in partial-order planning. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pages 500–506, Washington, DC, 1993.

[Stefik 1981] Mark Stefik. Planning with constraints. *Artificial Intelligence*, 16(2):111–140, 1981.

[Sussman 1973] G.A. Sussman. *A Computational Model of Skill Acquisition.* M.I.T. AI Lab Memo no. AI-TR-297, 1973.

[Tate 1977] Austin Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 888–893, San Mateo, CA, 1977. Morgan Kaufmann Publishers, Inc.

[Unruh and Rosenbloom 1989] Amy Unruh and Paul S. Rosenbloom. Abstraction in problem solving and learning. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 681–687, Detroit, MI, 1989.

[Wilkins 1984] David Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, Vol. 22, pages 269-301, 1984.

[Woods 1993] Steven Woods. A method of interactive recognition of spatially defined model deployment templates using abstraction. In *Proceedings of the 1993 DND Workshop on Advanced Technologies in Knowledge Based Systems and Robotics*, Pages 665-675, Department of National Defence, Government of Canada, November 1993.

[Woods 1991] Steven G. Woods. An implementation and evaluation of a hierarchical nonlinear planner. Master's thesis, Computer Science Department, University of Waterloo, 1991.

[Yang 1992] Qiang Yang. A theory of conflict resolution in planning. *Artificial Intelligence*, 58(1-3), pp. 361-392, 1992.

[Yang, Tenenberg and Woods 1991] Qiang Yang, Josh Tenenberg, and Steve Woods. Abstraction in nonlinear planning. Technical Report CS 91-65, University of Waterloo, Department of Computer Science, Waterloo, Ontario, Canada N2L 3G1, 1991.

A Operators in the 3-disk Towers of Hanoi Domain

MoveLarge (x y)
Preconditions={IsPeg(x)
IsPeg(y)
¬ OnMedium(x)
¬ OnMedium(y)
¬ OnSmall(x)
¬ OnSmall(y)
OnLarge(x)}
Effects={¬ OnLarge(x)
(OnLarge y)}

MoveMedium (x y)
Preconditions={IsPeg(x)
IsPeg(y)
¬ OnSmall(x)
¬ OnSmall(y)
OnMedium(x)}
Effects={¬ OnMedium(x)
OnMedium(y)}

MoveSmall (x y)
Preconditions={IsPeg(x)
IsPeg(y)
OnSmall(x)}
Effects={¬ OnSmall(x)
OnSmall(y)}

B Operators in the Robot Task Planning Domain.

This appendix lists the operators used in the robot task planning domain. Primary effects of operators are marked by "*."

B.1 Operators for going between rooms

To push a box through a door between 2 rooms.

push-thru-dr (box door-nm from-room to-room door-loc-from door-loc-to robot)
Preconditions={Is-Door(door-nm from-room to-room door-loc-from door-loc-to)
Pushable(box)
Box-Inroom(box from-room)
Robot-Inroom(from-room)
Box-At(box door-loc-from)
Robot-At(door-loc-from)
Open(door-nm) }
Effects={¬ Robot-Inroom(from-room)
Robot-Inroom(to-room)
¬ Box-Inroom(box from-room)
Box-Inroom(box to-room)*
Robot-At(door-loc-to)
Box-At(box door-loc-to)*
¬ Robot-At(door-loc-from)
¬ Box-At(box door-loc-from) }

To go through door from room2 to room1.

go-thru-dr (door-nm from-room to-room door-loc-from door-loc-to)
Preconditions={Is-Door(door-nm from-room to-room door-loc-from door-loc-to)
Robot-Inroom(from-room)
Robot-At(door-loc-from)
Open(door-nm) }
Effects={Robot-At(door-loc-to)*
¬ Robot-At(door-loc-from)
¬ Robot-Inroom(from-room)
Robot-Inroom(to-room)*}

B.2 Operators for going within a room

Operator for going to a location in a room.

goto-room-loc (from to room)
Preconditions={Location-Inroom(to room)
Location-Inroom(from room)
Robot-Inroom(room)
Robot-At(from) }
Effects={¬ Robot-At(from)
Robot-At(to)*}

Operator for pushing box between locations within one room.

push-box (box room box-from-loc box-to-loc robot)
Preconditions={Pushable(box)
Location-Inroom(box-to-loc room)
Location-Inroom(box-from-loc room)
Box-Inroom(box room)
Robot-Inroom(room)
Box-At(box box-from-loc)
Robot-At(box-from-loc) }
Effects={¬ Robot-At(box-from-loc)
¬ Box-At(box box-from-loc)
Robot-At(box-to-loc)
Box-At(box box-to-loc)*}

B.3 Operators for opening and closing doors

To Open a door.

Open (door-nm from-room to-room door-loc-from door-loc-to)
Preconditions={Is-Door(door-nm from-room to-room door-loc-from door-loc-to)
¬ Open(door-nm)
Robot-At(door-loc-from) }
Effects={Open(door-nm)*}

To close a door.

close (door-nm from-room to-room door-loc-from door-loc-to)
Preconditions={Is-Door(door-nm from-room to-room door-loc-from door-loc-to)
Open(door-nm)
Robot-At(door-loc-from) }
Effects={¬ Open(door-nm)*}

C AbTweak Algorithm

C.1 Data Structures and Subroutines

1. OPEN — A priority queue of plans on the frontier of the search tree. The list is sorted in ascending order of the plans' costs, $Cost(\Pi)$.

2. MTC(Π) — A predicate on plans, which is true of Π exactly when Π is necessarily correct.

3. Successors(Π) — A function mapping each plan to a set of successor plans.

C.2 AbTweak

Algorithm AbTweak **(initial, goal):**

$OPEN \leftarrow$ Initial-Plan,
{where Initial-Plan is a plan with two operators, initial and goal).}
Loop
    If $OPEN$ is empty, **Then** exit with failure.
    Else, let $\Pi = First(OPEN)$, and $OPEN \leftarrow Remove(\Pi, OPEN)$.
    Endif
    If $crit(\Pi) = 0$ and $MTC(\Pi) =$ True, **Then** return Π, and exit with success.
    Else, **If** $MTC(\Pi) =$ True, **Then**
        {the plan Π is correct at an abstract level},
        $crit(\Pi) \leftarrow crit(\Pi) - 1$,
        $OPEN \leftarrow Insert(\{\Pi\}, OPEN)$,
        Else,
        {Successor Generation: Plan Π must contain at least one
        precondition that does not necessarily hold.}
        $OPEN \leftarrow Insert(Successors(\Pi), OPEN - First(OPEN))$.
        Endif
    Endif
Endloop

C.3 Successor Generation

Subroutine Successor (Π)
{Comment: The global variable MP is True whenever the Monotonic Protection is used in ABTWEAK.}

$succ := \emptyset$
$successors := \emptyset$
Find a precondition *precond* of an operator *User* in plan Π,
such that *precond* is not necessarily true
If MP = True and *precond* is established for *User* at a level higher than $crit(\Pi)$ **Then**
  For each of the abstract establishment relations of the form $Est_i = \text{Establishes}(\alpha_i, User, precond)$,
    that have been clobbered at the current level,
      $succ := \{(\Pi, Est_i)\} \bigcup succ.$
  Endfor
Else
      Let *Old* be the set of operators in Π whose effects possibly
      establish *precond* for *User*, and let *New* be the set of new
      operators taken from the operator schemas of the domain, that have
      effects which possibly codesignate with *precond*.

      For each operator α in $Old \bigcup New$ **Do**
          (1) Add temporal and codesignation constraints to a copy Π' of Π
          so that for some effect e_α of α, the relation
          $Est = \text{Establishes}(\alpha, User, precond)$ holds
          (2) $succ := succ \bigcup \{(\Pi', Est)\}$
Endif
{ **Declobber** }
For each pair (Π',$Est = \text{Establishes}(\alpha, User, precond)$) in *succ*, **Do**
    If *Est* is clobbered **Then**
      For each clobberer C of *Est*, with a clobbering effect e_C, **Do**
          (1) impose the constraint $C \prec \alpha$, onto a copy Π_1 of Π',
          (2) impose the constraint $User \prec C$, onto a copy Π_2 of Π',
          (3) impose the constraint $e_C \not\approx \neg precond$, onto a copy Π_2 of Π'.
          (4) **For** each $\Pi_i, i = 1, 2, 3$, if the constraints in Π_i are consistent, then
            $successors := successors \bigcup \{\Pi_i\}$.
          Endfor
          {Each copy is a new successor in the search space.}
      Endfor
    Else $successors := successors \bigcup \{\Pi'\}$
    Endif
Endfor

If MP = True **Then** {Monotonic Protection }
For every plan Π' in *successors*, **Do**
If there is some precondition p of an operator β,
with an establishment set S in Π', such that
For every establishment relation Establishes$(\alpha, \beta, p) \in S$
there is an operator γ such that
{Note: This condition defines monotonic violation.}
(1) $(\alpha \prec \gamma \prec \beta)$,
(2) For some effect e_γ of γ,
either $(e_\gamma \approx p)$ or
$(e_\gamma \approx \neg p)$.
Endfor
Then *successors* := *successors* $- \{\Pi'\}$
Endif
Endfor
Endif
Return *successors*

C.4 TWEAK Implementation

TWEAK can be implemented by making the following modifications to the ABTWEAK routines:

1. $crit(\Pi) = 0$, for all Π,
2. In the successor generation part, remove the two monotonic protection components.

List of Symbols

Symbol	Meaning
α, β, γ	Operators
Π	Plan
$\approx$	codesignation constraint
$\not\approx$	non-codesignation constraint
crit	criticality value
P_α	preconditions of α
${}_iP_\alpha$	preconditions with criticalities higher than i
$\neg$	logical negation
MP	monotonic goal protection