Evaluating the Utility of Goal Protection and Search Strategies in Abstract Planning

Steven Woods* Qiang Yang† Josh D. Tenenberg‡

Abstract

To solve a complex problem using abstraction, one would like to protect parts of the abstract solution that has already been completed. However, it has not been well understood how this protection can be done profitably; some subgoal protection may indeed result in a decrease in efficiency.

The Monotonic Property has been proposed as a form of goal protection in abstract planning. In this paper, we show that there are different versions of this property, each with a different effect on planning performance and with different applicability to planning domains. Furthermore, we present a series of empirical tests of the utility of the property, and of different search strategies for abstract planning.

If goal protection is considered as a constraint for search within one level of abstraction, then search across different levels of abstraction is another problem that needs to be addressed. Less abstract solutions are generally closer to a concrete level solution than more abstract ones. We take advantage of this fact by developing a new, abstract planning control strategy known as Left-Wedge. This strategy adds a depth-first flavor to a complete search strategy. The relative benefit of this approach under various criticality assignments demonstrated through comparisons with breadth-first strategy.

Topic: Planning, Scheduling, and Reasoning about Action.
Subtopic: Planning with Abstraction
Secondary Topic: Automated Reasoning.

*Computer Science Department, University of Waterloo, Waterloo, Ont., Canada, N2L 3G1,
†Computer Science Department, University of Waterloo Waterloo, Ont., Canada, N2L 3G1,
‡Computer Science Department, University of Rochester, Rochester, New York, 14627

1 Introduction

In solving complex planning problems, one would like to distinguish parts that are crucial and difficult to solve, from those that are less important and easy to solve. To achieve this purpose, research in abstract planning has been particularly active, which ranges from the exploration of different types of abstraction systems [2, 3, 8, 9, 11], to the formal characterization of the intuition behind abstraction in planning[4, 6, 10, 13].

Despite the differences between the different approaches, a common feature dictates most of the systems. That is, planning is done in a top-down manner, starting from the most important and difficult level of abstraction, down to the most basic, concrete level. In this respect, two important questions in search control are yet to be answered. The first open question concerns the control of search *within* each individual level of abstraction, and the second is related to the coordination of search *across* different levels of abstraction.

The first open question is intimately related to the formalization of the intuition behind successful abstraction hierarchies and systems. When an abstract solution is formed, a set of causal relations among the operators in the solution is also known. Search within one level is often guided by this causal structure in the abstract solution. In protecting the subgoal establishment structure already achieved at the higher levels of abstraction, one imposes constraints on planning in the current level. To maintain search efficiency, one has to make decisions regarding what subgoals to protect, how to, and when to protect them. For example, is it more advantageous to backtrack when some previously satisfied goals are in danger of being undone, or to simply allow this destruction to occur, and re-plan for those goals later?

The second question involves search across different levels of abstraction. This search must be performed carefully, in order to preserve completeness. Yet, there is no guarantee that a complete search with abstraction will outperform one without abstraction. Thus, it is also important to investigate search strategies with which an abstract search outperforms one without abstraction.

This paper reports an empirical study in addressing the above two problems. In addressing the subgoal protection problem within each level, we investigate the utility of protecting subgoals with a nonlinear, hierarchical planner AbTweak. Across different hierarchies of the same domains, we test several different implementations of the *Monotonic Property* (MP) [13], a property which allows for the application of goal protection within abstract planning while still allowing a search strategy to remain complete. The results demonstrate that depending on the search strategies used across different levels of abstraction, protecting higher level subgoals may not always improve the efficiency of planning. With analysis of these results, we point out several ways of making the improvement possible.

In addition, we show how to conduct a complete search across different levels with certain depth-first search flavors. This new search strategy, called the *Left-Wedge* search, assigns a higher priority in search to a plan that is more refined than others. For some hierarchies, search using the left-wedge strategy is constrained to occur within a considerably smaller

frontier while preserving completeness. As a result, search efficiency with abstraction is dramatically improved.

The organization of the paper is as follows. We first discuss the two distinct search strategies used as the basis for experiments. Next, we describe the monotonic properties inherent in abstract planning. Within the given context, we then describe the experiments completed, and evaluate our results with an eye towards explaining planning behaviour under various search strategies and applications of monotonic properties. We conclude summarizing the most interesting aspects of our experiments.

2 Abstraction via Precondition-Elimination

The representation of AbTweak is built upon that of the nonlinear, least-commitment planner Tweak[1]. An operator in Tweak is defined by a set of precondition literals and effect literals. A plan, Π, is defined as a triple (A, B, C), where A is a set of operators, B is a partial order on A, and C is a set of codesignation and noncodesignation constraints. A detailed discussion of AbTweak can be found in [13]. Below, we use Tweak's definition for *necessary* ($\Box$) and *possible* ($\Diamond$) constraints on operator ordering or variable bindings.

Given a domain definition in terms of operator set O, an abstraction hierarchy can be constructed by assigning criticalities to the precondition literals. Let a be an operator. We take ${}_iP_a$ to be the set of preconditions of a which have criticality values of at least i. Then a i^{th} level hierarchy can be defined by replacing each operator a's precondition by ${}_iP_a$.

3 Search Within Each Level of Abstraction

3.1 Monotonic Property

The purpose of subgoal protection is to ensure that the previously achieved tasks are not undone when completing a plan. In doing this, one would like to use subgoal establishment structure as a constraint on search, while preserve completeness of the systems. Towards this goal, Yang and Tenenberg [13] define the *Monotonic Property*.

In a correct plan Π, if an operator α necessarily achieves a precondition p of an operator β, and if no other operators necessarily between α and β does so, then we say α *establishes* p for β, or $\mathrm{Est}(\alpha, \beta, p)$.

Informally, a refinement of an abstract plan is *monotonic*, if the subgoal-establishment structure of the abstract plan is preserved in the refinement. More precisely, let Π_a be a plan at abstract level i for solving goal G, and Π be a plan at level $i-1$ for solving the same goal. Suppose that both plans are *justified*, in that every operator in them directly or indirectly achieve a goal. Then Π is a *monotonic refinement* of Π_a if the abstract version of Π is Π_a. For example, let Π be `go-to(door), open(door), go-between(room1, room2)` for getting into `room2`, and the abstract plan Π_a be `go-between(room1, room2)` for solving

the same goal. Then Π is a monotonic refinement of Π_a. It has been shown that in every abstraction hierarchy based on precondition elimination, if there is a correct plan at the base-level, then one of the abstract plans can monotonically refine to a solution at the base level [13].

The monotonic property imposes constraints on search providing the ability to backtrack on protection violations. A violation occurs when another operator γ is inserted between α and β, where $\text{Est}(\alpha, \beta, p)$, and γ necessarily asserts p. It was shown that backtracking on protection violations in this manner retains completeness in a search strategy [13].

3.2 Search Pruning using the Monotonic Property

3.2.1 Strong vs Weak Monotonic Property

Due to the nonlinear nature of plans, it is possible that a precondition p of an operator β is established by two or more unordered operators α_1 and α_2 in the plan. In this case, two versions of monotonic property can be designed:

Strong Monotonic Property (SMP): When refining an abstract plan, protect *all* establishment relations, including both $\text{Est}(\alpha_1, \beta, p)$, and $\text{Est}(\alpha_2, \beta, p)$.

Weak Monotonic Property (WMP): When refining an abstract plan, protect at least one of establishment relations, i.e., either $\text{Est}(\alpha_1, \beta, p)$, or $\text{Est}(\alpha_2, \beta, p)$, but not both.

As suggested by their names, SMP imposes a stronger constraint on search, by backtracking on *any* monotonic violation. On the other hand, WMP backtracks only when *all* abstract establishment relations are violated. One would be tempted to conclude that the strong version is always superior.

Unfortunately, SMP cannot guarantee completeness in general. This can be shown by the following counter-example. Suppose for a planning problem, the only solution at the base level is a linearized plan, "$\Pi = \alpha_1, \alpha_2, \beta$." Also suppose that in the abstract plan both α_1, and α_2 are establishers for γ. Note that a nonlinear, least-committed planner does not compute every linearization of a correct abstract plan. Thus, protecting both establishment structures, the planner can never find the solution Π. On the other hand, with WMP the completeness can be shown to hold, since $\text{Est}(\alpha_2, \beta, p)$ is never violated. Thus, in our experiments presented below, WMP version is used.

3.2.2 Two versions of the WMP

To apply WMP in a straightforward fashion, one ensures for each abstract precondition p that, whenever a new operator is inserted into a plan, or a set of constraints is imposed upon a plan, one of the establishment relations for p still holds. More precisely, let $\text{Est}(\alpha, \beta, p)$ hold in a abstract plan. In its refinement, let γ be an operator inserted *necessarily* between

α and β. If γ necessarily asserts or denies p, then a monotonic violation occurs. This version of the WMP is call *necessary* weak monotonic property, or N-WMP.

With more domain knowledge, one can do better. Then a *possible* monotonic violation occurs for $\text{Est}(\alpha, \beta, p)$, whenever an operator γ is inserted necessarily between α and β, such that γ asserts q, and q possibly codesignates with p. For example, suppose a robot can only hold one thing at a time, and in an abstract plan, an operator α =`pickup(cup)` establishes the precondition `holding(cup)` for β =`put-on-table(cup)`. If the operator γ =`pickup(x)` is inserted between α and β, a possible monotonic violation occurs.

If certain predicates of a domain satisfy the following condition, then prevention of possible monotonic violation guarantees completeness, and the hierarchy is said to have *possible* weak monotonic property (P-WMP). The condition is as follows:

> Let r be a predicate. For two different sets of parameters X and Y, $r(X)$ and $r(Y)$ cannot hold at the same time.

It can be shown that given this condition, any possible weak monotonic violation in a plan Π implies that every descendent of Π will eventually lead to a monotonic violation. This condition is true for many predicates. In the robot example above, `hold` is one such predicate. Moreover, in the Towers of Hanoi example to be shown later, the predicates `ons, onm, and onb` are all of this type.

The advantage of P-WMP over N-WMP is that it allows pruning earlier in search, thus limiting the branching factor of search even more.

4 Search Across Abstraction Levels

We are interested in finding complete search strategies to coordinate planning at different levels of abstraction. This is a difficult problem because we would like AbTweak to be both complete, and more efficient than planning without abstraction.

It must be noted that a simple-minded application of Tweak's search strategy at each level of hierarchy is *not* complete across different levels. Tweak is only semi-decidable in the sense that termination is not guaranteed if planning for a given problem which does not have a solution. A particular abstract solution at level k may not be refinable to a solution at level $k-1$. As a result, an abstract strategy that is committed to only an individual level-k solution may never find a concrete level solution, even if the search within each level is complete.

AbTweak's search strategy does not overcommit in this fashion, 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. The degree in which a search strategy tends to favor either dimension of growth is an important aspect governing search performance.

An obvious control strategy for search control is to use breadth-first search. In the search space, a state corresponds to a plan. During each iteration, a state Π is selected

for correctness check using the Modal Truth Criterion. If Π is correct at level i, then all operators in Π are replaced by their corresponding $i-1$-level operators. Otherwise, the plan is modified according to TWEAK's plan modification procedures. The process terminates when a correct plan is found at level-0. The cost of each state in the search tree is simply the total number of operators in the plan.

One problem with the above strictly breadth-first search strategy is that plans at higher levels of abstraction are preferred equally with plans at lower abstraction levels. Another way to organize search is to prefer the selection of a lower level plan *more* than an abstract one, because a less abstract plan is potentially closer to a concrete level solution. The reason for this is that if we simply want to obtain any concrete level solution, it may be beneficial to progress more deeply in the abstract space beneath existing high abstract level solutions, rather than continue to generate more high level possibilities. Towards a more "depth-first" yet complete strategy, we have devised a hybrid strategy which retains the completeness of breadth-first, and also has the advantage of preferring less abstract solutions in a somewhat depth-first manner. This is called the *Left-Wedge* strategy. Given two states Π_1 and Π_2 in the search space, such that Π_2 is at a more abstract level than Π_1, it assigns a higher priority (or less cost) to Π_1 than Π_2, such that for every plan expansion to Π_2, several more expansions are done for Π_1.

In general, the Left-wedge abstract search strategy is expected to work well if a base-level solution tends to have simple, short abstractions. One can view the search tree as organized by ordering the shortest solutions on each level in a left-to-right way. Then search basically proceeds along the left wedge of the search tree, the tip of the wedge being deeper towards its left side.

5 Experiments

5.1 Domains

Several domains have been tested with ABTWEAK, including the 3-disk Towers of Hanoi (Hanoi-3), the Blocks World Domain [7], and Sacerdoti's robot and box domain [8] (see [12]). Because of the space limitation, only Hanoi-3 results are presented.

Hanoi-3 is formulated as follows. There are three pegs and three disks, of different sizes. A disk can only be placed on a peg, or a disk bigger than itself. Initially, all disks are on `Peg1`. In the goal state, all three disks are on `Peg3`. The predicates used in representing the domain are listed in Table 1.

This domain has been extensively tested in the past with linear, abstract planners [5]. Most researchers have based their tests on one obvious hierarchy, $criticality(ISPEG) = 3$, $criticality(OnBig) = 2$, $criticality(OnMedium) = 1$, $criticality(OnSmall) = 0$. In order to fully test the utility of the monotonic property, and search strategy, we have tested all possible permutations of the hierarchies. For ease of exposition, we use *ibms* to represent the

Predicate	Meaning
Ispeg(x)	x is a peg
OnBig(x)	location of the big disk
OnMedium(x)	location of the medium disk
OnSmall(x)	location of the small disk

Table 1: Towers of Hanoi

above hierarchy. Similarly, *smbi* represents the hierarchy with the reverse order of criticality assignment.

5.2 Explanation of the Figures

Our results can be divided into two categories: those demonstrating the usefulness of monotonic properties in restricting search, and those comparing the left-wedge and breadth-first search strategies. Results are shown in Figures 1 through 10.

Figures 1 to 4 and 7 to 10 show the number of state expansions required by AbTweak to find a solution in the Hanoi-3 domain as a function of the abstraction hierarchy used. Figures 1 to 4 show the number of expansions required by breadth-first search (1) without any monotonic property, (2) with P-WMP, and (3) with N-WMP. Figures 7 to 10 contrast breadth-first and left-wedge strategies without using monotonic properties. Figure 5 contrasts breadth-first and left-wedge AbTweak with P-WMP. Figure 6 compares breadth-first and left-wedge with and without monotonic properties.

5.3 The Utility of The Monotonic Properties

We can summarize our experimental results supporting the application of the monotonic property in abstract planning as follows:

1. Using the monotonic property to constrain search is almost always advantageous, for hierarchies that are intuitively good. An intuitively good hierarchy solves a difficult problem, such as `OnBig`, first, and the easy parts like `OnSmall` the last. The better performance can be observed from Figures 1-4. For example, in Figure 1, breadth-first search using either N-WMP or P-WMP outperforms one without MP pruning in 5 of 6 cases: (*ibms ibsm imbs imsb isbm*, excluding *ismb*). This can be explained in two ways. First, using the MP reduces the branching factor of search. Second, when pruning is done, it is unlikely that a branch leading to a solution will be cut off during breadth-first search. The reason for the second justification is that for most problems, it is more straightforward to find a base-level solution from an abstract one, without violating any abstract causal relations.

2. Breadth-first search is more efficient with P-WMP than with N-WMP (see Figures 1-4).

   The superiority of P-WMP over N-WMP can be explained by the fact that the former cuts a subtree of the search space off much earlier than by the latter. This is because every search path descending from a plan that violates P-WMP, will eventually be terminated as a result of violating N-WMP.

3. Without using MP, the criticality assigned to object-type predicates such as `IsPeg` is crucial in search efficiency (see Figures 2-4). Overall, it seems better to place these types of predicates at the highest level of abstraction. However, MP tends to stabilize the effect of criticalities on search (see Figures 2-4).

4. We can see a general rule emerging from the results in Figure 5, that the fewer the number of monotonic violations, the better the performance in search with an abstraction hierarchy. In a sense, MP measures the number of attempts we make at a particular level of abstraction to undo the work done at the previous level. Exceptions to this general rule can occur when operators added into a plan have beneficial side-effects, or in instances where a simple solution can be found more quickly generating a successor which violates the monotonic property.

5. Figure 6 summarizes the results of tests with and without using the MP. The first column lists six hierarchies, and the rest the number of states expanded under a particular search strategy.

   From the table, it is clear that *ibms*, *ibsm* and *imbs* are three hierarchies that demonstrate better performances than the rest. This phenomenon can be easily explained by the fact that, for example, in ibms, the abstract solution at `OnBig` level corresponds the *first* abstract solution. However, in *ismb*, the abstract solution refineable, at the `OnSmall` level, to the base-level one actually corresponds to the fourth alternative solution at that level. As a result, the search space with *ibms* has a much smaller branching factor than ismb. Thus, intuitively, *ibms* is a good abstraction hierarchy. Similarly, hierarchies *ibsm* and *imbs* are close to be good ones. On the other hand, the other three are far from optimal. This is because they all solve easier problems (involving the placement of the small disk) first, and solve the harder ones (involving the placement of medium or big disks) the last.

   The following conclusion can be drawn from Figure 6:

   (a) For good abstraction hierarchies, using the MP is clearly better than without using the MP on average cases. For example, the average number of expansions using *ibms*, *ibsm*, and *imbs* under breadth-first and P-WMP is 450. The average for the other three hierarchies under breadth-first, and without using MP is 711.

(b) For good abstraction hierarchies, using Left-Wedge strategy and P-WMP is an overall winner. For example, Left-Wedge plus P-WMP for the first three hierarchies gives an average of 222 number of expansions, as opposed to 711 for breadth-first.

(c) However, for bad hierarchies, the use of MP with either breadth-first search or Left-Wedge dramatically decreases search efficiency. For example, for *ismb*, either breadth-first or Left-Wedge plus MP couldn't even finish in 6000 expansions.

Thus, goal-protection in abstract planning improves efficiency only when the hierarchy used is intuitively good.

5.4 Left Wedge Refinement Utility

We can summarize our experimental results supporting the use of our left-wedge search strategy in abstract planning as follows:

1. The results of planning using the left-wedge approach (Figure 6 to 10) indicate quite a dramatic improvement over breadth-first for certain criticality assignments. In Figure 6, this can be observed under Left-Wedge without using MP for hierarchies *isbm*, *ismb*, *ibms* and *ibsm*. However, no improvement, or even a decrease in performance is seen for certain other criticality assignments, notably *imbs* and *imsb*.

   It is also interesting to note that, for hierarchies (*isbm* and *ismb*, Figures 9 and 10) that perform very poorly with breadth-first, the left-wedge strategy achieves good results.

2. Left-Wedge tends to stabilize the effect of criticalities on search (see Figures 7, 9 and 10). Although this effect can be easily explained for intuitively good hierarchies (i.e., *bms*), it is still not clear why the stability exists in bad hierarchies such as *sbm* and *smb*.

5.5 Comparing TWEAK with ABTWEAK

The utility of abstract search will not be completely understood without also comparing it with search without abstraction. TWEAK expanded 379 nodes when solving the Hanoi-3 problem. However, ABTWEAK without using the monotonic property does worse; in hierarchy *ibms*, 471 nodes are expanded. But, if P-WMP is added, ABTWEAK with the hierarchy *imbs* expanded only 149 nodes, more than twice as efficient as TWEAK.

When compared with Left-Wedge, the difference is more dramatic. With good hierarchies, e.g., *ibms*, the number of nodes expanded is only 57, more than six times more efficient than TWEAK.

6 Figures

These figures refer to experiments performed using the Towers of Hanoi domain for three pegs.

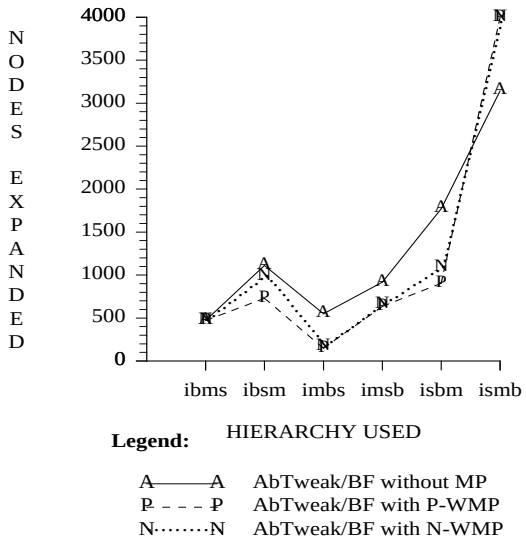

Figure 1: Expansions, vary hierarchy

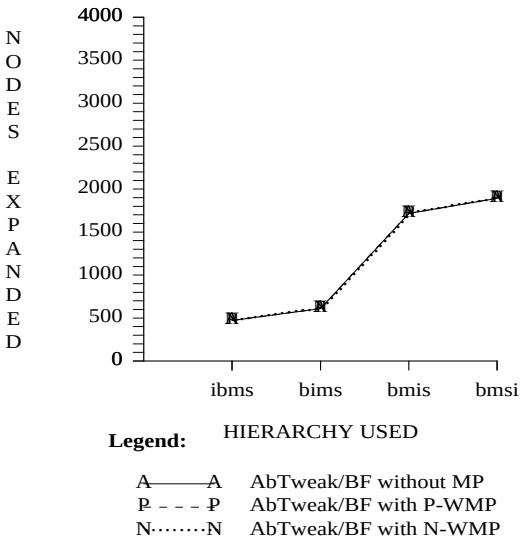

Figure 2: BMS expansions, vary IsPeg

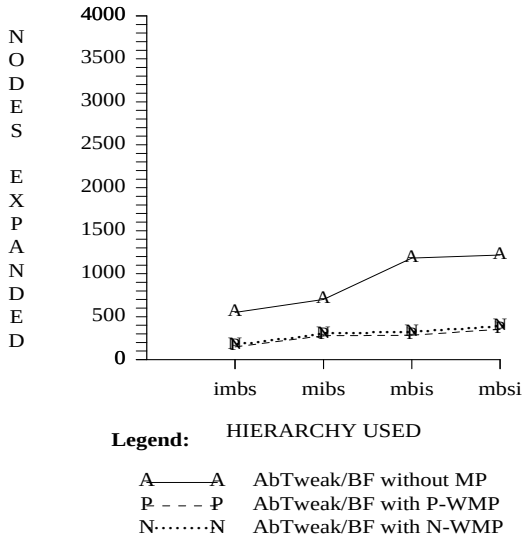

Figure 3: MBS expansions, vary IsPeg

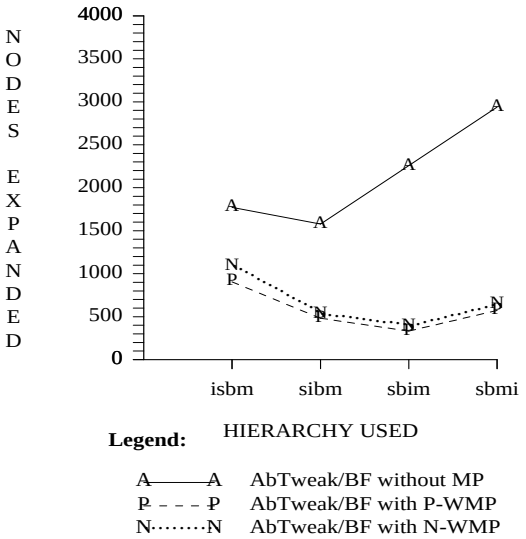

Figure 4: SBM expansions, vary Ispeg

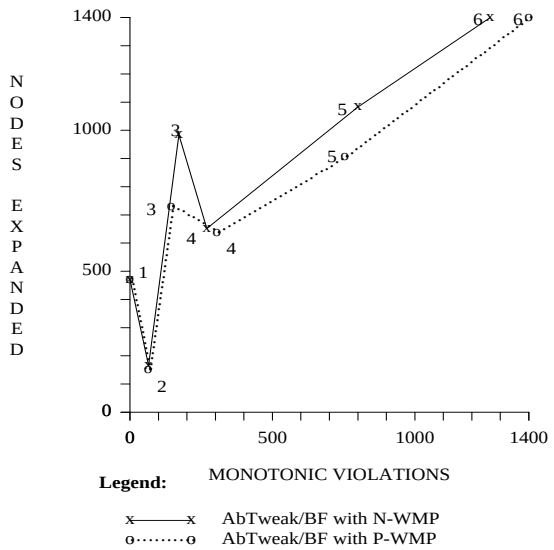

Figure 5: BF Expansions : 1-IBMS, 2-IMBS, 3-IBSM, 4-IMSB, 5-ISBM, 6-ISMB

	Breadth	Breadth P-WMSP	Left-Wedge	Left-Wedge P-WMSP
IBMS	471	471	57	57
IBSM	1112	729	828	531
IMBS	550	149	1009	78
IMSB	918	636	5170	2672
ISBM	1771	904	168	5232
ISMB	3142	>6000	963	>6000

Figure 6: Summary of BF and LW expansions

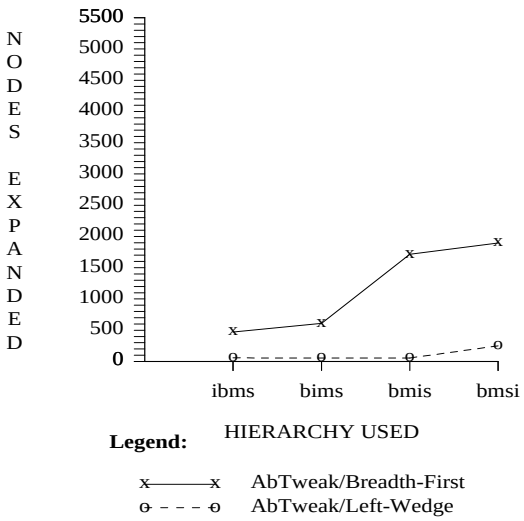

Figure 7: BMS expansions: BF,LW

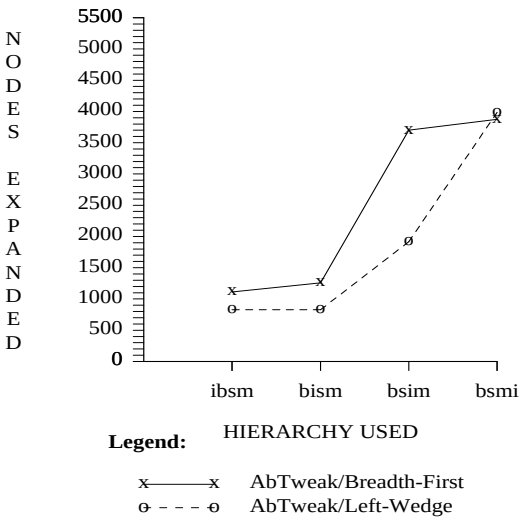

Figure 8: BSM expansions: BF,LW

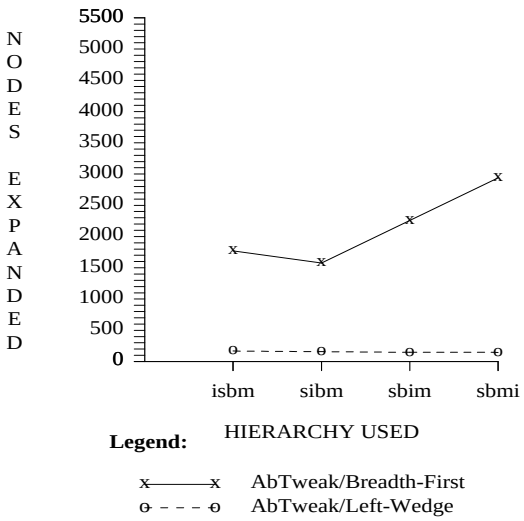

Figure 9: SBM expansions: BF, LW

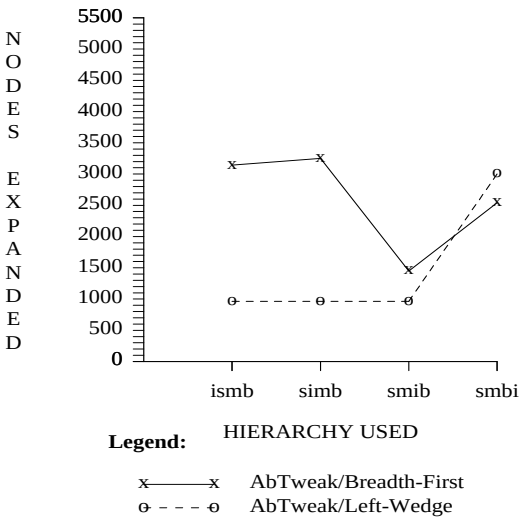

Figure 10: SMB expansions: BF, LW

7 Summary and Future Work

The experiences and lessons learned from our experiments can be summarized as follows:

1. The application of the monotonic property is a form of goal protection in abstract planning. However, if one protects too much, as in the case of SMP, completeness is not preserved. The use of the other versions of the WMP , including both P-WMP and N-WMP, in abstract planning with good hierarchies and using a breadth-first search strategy, tend to outperform breadth-first without the application of MP. This is true for most criticality assignments, with only 2 exceptions out of 24 tests. Furthermore, with a slightly domain-dependent implementation of WMP, i.e. P-WMP, AbTweak always outperforms the domain-independent version of WMP, i.e. N-WMP.

2. When abstract planning takes on a depth-first flavor, as seen in the Left-Wedge strategy, the application of MP is useful only when utilizing *good* abstraction hierarchies. On the other hand, with *bad* hierarchies, Left-Wedge searches more than with a breadth-first strategy.

3. The number of monotonic violations is an indication of the relative *goodness* of a hierarchy. A hierarchy is generally indicated to be *better* in terms of predicted search performance if it demonstrates fewer monotonic violations when planning under a complete search strategy.

Criticality assignments to precondition literals represent one type of goal ordering heuristic. Experiments show that other factors also exist, such as the selection of preconditions at each level of abstraction. Our future work will aim at explaining and analyzing the effect of these other factors on the efficiency of abstract search.

Acknowledgement

Thanks to Verna Friesen and Craig Knoblock for many useful comments. This work is supported in part by NSERC operating grant OGP0089686.

References

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

[2] Jens Christensen. Pablo: A hierarchical planner that generates its own abstraction hieararchies. *Submitted for Publication, Stanford University*, 1990.

[3] Craig A. Knoblock. A theory of abstraction for hierarchical planning. In Paul Benjamin, editor, *Proceedings of the Workshop on Change of Representation and Inductive Bias*, Boston, MA, 1989. Kluwer.

[4] Craig A. Knoblock. Learning effective abstraction hierarchies. In *Proceedings of Eighth National Conference on Artificial Intelligence*, Boston, MA, 1990.

[5] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, Carnegie Mellon University, to appear, 1991.

[6] Richard Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1985.

[7] Nils Nilsson. *Principles of Artificial Intelligence.* Morgan Kaufmann Publishers Inc, 1980.

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

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

[10] Josh Tenenberg. *Abstraction in Planning.* PhD thesis, University of Rochester, Dept. of Computer Science, Rochester, NY, May 1988.

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

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

[13] Qiang Yang and Josh Tenenberg. Abtweak: Abstracting a nonlinear, least commitment planner. In *Proceedings of the 8th AAAI*, Boston, MA, August 1990.