Admissible Pruning Strategies based on plan minimality for Plan-Space Planning

Subbarao Kambhampati*
Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-5406
Email: rao@asu.edu
WWW: http://rakaposhi.eas.asu.edu/pub/rao/rao.html

January 19, 1996

Abstract

Although plan-space planners have been shown to be flexible and efficient in plan generation, they do suffer from the problem of ''looping'' -- that is, they may spend an inordinate amount of time doing locally seemingly useful but globally useless refinements. In this paper, I review the anatomy of looping and argue that looping is intimately tied to the production of non-minimal solutions. I then propose two classes of admissible pruning techniques based on the notion of plan minimality. I show that the first one is admissible for planners which do not protect their establishments, but allow a precondition to be reestablished any number of times. The second one is admissible for planners which protect their establishments through causal links. I also discuss the complexity of the proposed pruning strategies, and their potential applications.

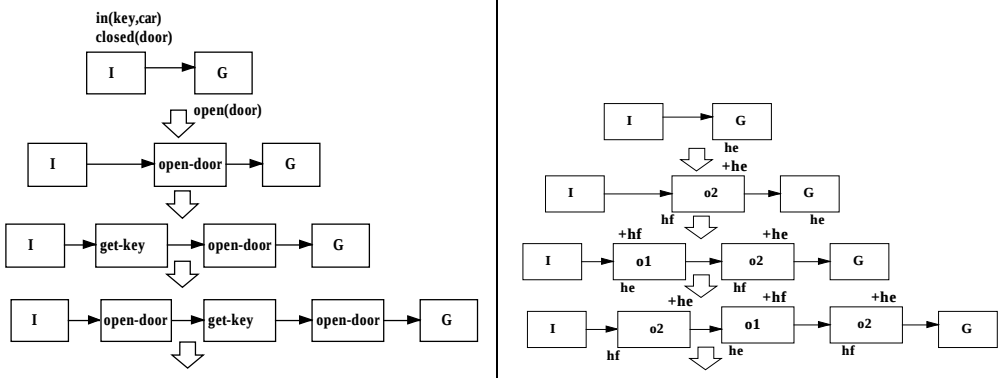
*This paper is a revised and extended version of a paper that was presented at IJCAI-95 []. This research is supported in part by NSF research initiation award (RIA) IRI-9210997, NSF young investigator award (NYI) IRI-9457634 and ARPA/Rome Laboratory planning initiative grant F30602-93-C-0039. I thank Eric Cohen, Yong Qu, Suresh Katukam and Laurie Ihrig for helpful comments.

1 Introduction

Domain independent classical planning techniques come in two main varieties -- those that search in the space of world states and those that search in the space of plans. The conventional wisdom of the planning community, supported to a large extent by the recent analytical and empirical studies [1, 15], holds that searching in the space of plans provides a more flexible and efficient framework for planning.

Despite its many perceived advantages, plan-space planning techniques still lag behind state-space planning techniques in terms of search control and pruning heuristics. In particular, an important property of state-space planners is that given any domain that has only a finite number of distinct states, the planner will terminate in finite time on any problem, whether or not the problem has a solution. In contrast, since the ground operator sequences form a power sequence over the set of operators in the planning domain, the complete search space of a plan-space planner can be infinite even for domains with finite number of actions. Consequently, a plan-space planner trying to solve an unsolvable problem may never halt without looping checks, even when the the corresponding state-space planner will terminate. As an example, consider the problems shown Figure 1. In the **car-door** example [8], the infinite regress occurs because the planner subgoals on having keys to open the door, and subgoals back on opening the door to get the keys (which are inside the car). A similar regress happens in the **hf/he** example, where the goal is to achieve he, given an empty initial state. The domain contains two operators O_1 and O_2. O_1 has a precondition he, deletes he and gives hf. O_2 has a precondition hf, deletes hf, and adds he.

The only way to avoid such looping is to intelligently prune unpromising search paths. Although there exist a variety of techniques for pruning search branches in state space planning (including the state-loop, goal-loop and inconsistent-state heuristics; see Section 7), these loop control techniques turn out to be either inapplicable, or inadmissible for plan-space planners [8] (following [5], we consider a pruning technique admissible if it does not affect the planner's ability to find all minimal solutions for any given problem). Of course, as long as the planning problem is solvable (i.e., has at least one solution plan), and the underlying planner uses an admissible search strategy to navigate the space of partial plans, then theoretically at least, lack of pruning strategies may not directly affect the efficiency of the planner. Instead, the increased branching factor, and the commitment of the state-space planners tend to dominate over the larger search space size of the plan-space planners. This explains why we typically



(a) **car-door** example

(b) **hf/he** example

Figure 1: Examples of a looping.

find plan-space planners faring better than state space planners in many domains [1, 15].

Despite this, pruning strategies for plan-space planners are still very important for several reasons. Even with best-first search, looping can cause serious problems when faced with unsolvable problems. Further, many practical planners use depth-first rather than best-first search strategies for efficiency purposes. Looping can significantly affect the efficiency of depth-first search regimes. The final, and often overlooked, need for pruning strategies has got to do with the importance of failures in learning [10]. Most speedup learning strategies to improve planning performance learn from the failures encountered in plan generation. Existence of a variety of pruning techniques provides a rich opportunity for the learner to learn from the pruned branches. Although best-first search strategies may be able to avoid the unpromising branches, and depth-limited depth first strategies may be able to avoid infinite looping, neither of them provide any guidance for the learner. In the presence of learning strategies, the cost of pruning techniques also tends to be less of a concern. In particular, it is possible to use the pruning techniques strategically by combining them with a depth-limited search, and applying them only to the plans that cross the depth-limits (see [10] for a demonstration of the effectiveness of this approach).

Despite their importance, very little work has been done towards formulation

and evaluation of pruning techniques for plan-space planning. Most existing plan-space planners prune a plan when the constraints on the partial plan are mutually inconsistent. This guarantees that the partial plan cannot be refined into a complete solution. Unfortunately, pruning inconsistent plans alone is not effective in many situations. To see this consider the hf/he example in Figure 1. Clearly, this problem has no solutions. Both forward and backward state-space planners will recognize this immediately and terminate. Specifically, the forward state-space planners terminate because no operators are applicable from the initial state. The backward state-space planners terminate after finding a state loop. In contrast, plan-space planners fail to terminate on this problem. Specifically, they will add O_2 to give he, add O_1 to give the precondition hf of O_1, add another O_1 to achieve the precondition he of O_2 and so on. At no point in this process is the resulting partial plan inconsistent, so it cannot be pruned by the consistency checks. Similar situation occurs in the car-door example.

Although this example might look contrived at first glance, many realistic domains do exhibit this type of behavior at least on some branches of the search space. For example, even if hf or he is present in the initial state of the above example, the search space still contains an infinitely looping branch. Similar looping also occurs when the planner attempts to generate a plan for opening the door of a car when the keys are inside the car and the door is closed [8], or plans for clearing a block by putting another block on top of it and then removing it.

We note that a hallmark of the looping behavior in the examples above is that in all cases the planner continues to add steps into an partial plan which seems to already contain too many steps. Although we cannot be sure that the refinement will not eventually lead to a solution, it is likely that no minimal solution will result from those branches. Intuitively, a plan is a minimal solution to a problem, if no strict subplan of it (derived by removing some steps from the original plan) is also a solution.

The discussion above suggests pruning partial plans which are unlikely to lead to *minimal solutions* for the problem. Since every solvable problem must have a minimal solution, a complete planner need only generate all minimal solutions for any given planning problem. While we can recognize and discard non-minimal solution plans (c.f. [6]), discarding complete non-minimal plans *after* they are generated by the planner, is an ineffective pruning strategy. Rather than wait until non-minimal solutions are generated, we would like to prune incomplete plans that are likely to lead to non-minimal complete plans.[1] Doing this, while retaining

[1]One might wonder as to why there are non-minimal plans in the search space to begin with.

soundness and completeness of the underlying planner, turns out to be a tricky proposition.

In this paper, I describe two techniques for pruning partial plans based on minimality considerations. The first one generalizes the notion of minimality to incomplete partial plans, and uses this notion to prune partial plans that are non-minimal. I will show that this technique is admissible for non-causal link planners, such as UA and TO [15, 3], which continue to refine a partial plan as long as there are preconditions that are not necessarily true (even if it means working on a precondition more than once). Unfortunately, however, it is not applicable to planners which use causal links [13], and do not work on any precondition more than once. To prune based on minimality for systematic causal link planners such as SNLP [13], I develop another strategy that uses the causal links to decide if any portion of the plan is ''taking'' more conditions than it is ''giving.''

The rest of this paper is organized as follows: The next section introduces some terminology for plan space planning. Section 3, reviews some existing pruning techniques. In section 4, I describe a technique called Nsat-prune that generalizes the notion of plan minimality to incomplete plans, and uses it as a basis to prune plans. I will show that it is admissible for non-causal link planners, but inadmissible for causal link planners. In Section 5, I discuss a pruning strategy called Cutset-prune that is applicable for causal link planners. Section 7 discusses related work and Section 8 summarizes the paper's contributions. The appendix contains complete proofs of all theorems.

2 Preliminaries and Terminology

In this section, I review some preliminaries and terminology related to plan space planning algorithms. Readers unfamiliar with plan-space planning might want to consult [9, 15] for more comprehensive reviews. A planning problem is a 3-tuple $\langle I, G, \mathcal{A} \rangle$, where I is the description of the initial state, G is the (partial) description of the goal state, and $\mathcal{A}$ is the set of actions (also called ''operators''). An action sequence (also referred to as ground operator sequence) S is said to be a solution for a planning problem, if S can be executed from the initial state of

The answer is that non-minimality is a function of the goals of the planner. For example, although plans that lead to state cycles (e.g. the sequence of actions Open the door, Close the door one after other) are non-minimal for any problem containing goals of achievement, they may be minimal if the goal of the planner itself is to exhibit that specific behavior (state sequence) [9]. Thus a general domain independent planner is forced to have such plans in its search space.

the planning problem, and the resulting state of the world satisfies all the goals of the planning problem.

A partial plan for a planning problem $\langle I, G, \mathcal{A} \rangle$ is a 5-tuple $\langle \mathcal{S}, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$ where: $\mathcal{S}$ is the set of steps in the plan; $\mathcal{S}$ contains two distinguished step names s_0 and s_∞. $\mathcal{ST}$ is a symbol table, which maps step names to actions from $\mathcal{A}$. The special step s_0 is always mapped to the dummy operator `start`, and similarly s_∞ is always mapped to `finish`. The effects of `start` correspond to I and the preconditions of `finish` correspond to G. O is a partial ordering relation over $\mathcal{S}$. $\mathcal{B}$ is a set of codesignation (binding) and non-codesignation (prohibited binding) constraints on the variables appearing in the preconditions and post-conditions of the operators. $\mathcal{L}$ is a set of causal links. A causal link $s \xrightarrow{p} s'$ is a commitment that the precondition p of the step s' will be supplied by an effect of the step s, and that p will be preserved in the interval between s and s' (i.e., no step that deletes p will be allowed to intervene between s and s').

A ground linearization of a partial plan is a permutation on the fully instantiated steps of the plan, that is consistent with all the orderings and bindings of the plan. A partial plan is said to be **complete** if all of its ground linearizations correspond to action sequences which are solutions to the planning problem. A complete plan is also called a **solution plan**. A partial plan that is not complete is said to be an **incomplete plan**.

Two partial plans P_1 and P_2 are said to be **equivalent** if there is a bijective mapping from the steps of P_1 to the steps of P_2, such that under that mapping the orderings, bindings, causal links and the symbol tables of the plans are equal. A partial plan P_1 is said to be a subplan of another partial plan P_2, if P_1 is equivalent to a plan P_2' derived from P_2 by removing some steps, and the ordering, binding, causal link relations involving those steps. A plan P is said to be **minimal** if it is complete, and no subplan of P is complete.

A precondition p of a step s in a partial plan is said to be **necessarily true** [3] (or satisfied) if in every ground linearization of the plan, there is some step s' that precedes s and gives p, and no step between s' and s deletes p. Similarly, if at least one ground linearization satisfies these conditions, then p is said to be possibly true. If all preconditions of all the steps are necessarily true, then the plan is complete. For actions whose preconditions and effects are function-less first order literals (called the TWEAK representation) Chapman [3] provides the necessary and sufficient conditions for checking the necessary truth of a precondition in polynomial time.

Plan space planning involves repeatedly selecting and ‘‘establishing’’ a

precondition of a step in the plan, such that it becomes necessarily true. When all preconditions are necessarily true, planning is complete. Since the establishment refinements used by most planners ensure completeness by considering all possible ways of achieving a chosen precondition [9], the order in which different preconditions are selected for establishment (referred to as ''goal selection order'' or ''goal order'') does not matter. However, when an individual partial plan is refined by establishing a specific precondition, the constraints added in that process may undo the establishment of a previously established precondition (thus making it not necessarily true). There are two general approaches for handling this -- some planners, popularly called goal protection planners or **causal link planners**, post constraints (called causal links) to protect their past establishments [13]. In particular, if the planner uses the effects of step s' to make the condition p true at step s, it posts a causal link $s' \xrightarrow{p} s$ on the partial plan. This constraint ensures that no step s'' that can delete p can possibly come between s' and s. Because of this, a causal link planner will never undo an establishment that it has made, and thus never has to work on the same precondition more than once. In [9], we point out that such protection strategies lead to reduction of redundancy in the search space. Examples of this class of planners include SNLP [13] and UCPOP [16]. A second class of planners, such as TWEAK [3], UA and TO [15], which may be called **non-causal link planners**, do not protect their establishments, but allow re-establishment of a precondition that was previously established, and was subsequently undone. We will see that these two classes of planners need differing types of pruning strategies.

3 Review of existing pruning techniques

Most existing techniques for pruning plans in plan-space planning attempt to show that the constraints in the partial plan are mutually inconsistent. If the inconsistency proof uses only the constraints of the plan, then we call the pruning technique ''domain independent.'' Such domain independent techniques include showing that the ordering constraints have a cycle, or showing that the binding constraints are unsatisfiable. The former can be done with the help of a topological sort algorithm in $O(n^2)$ time for an n step plan. The complexity of binding consistency check depends on whether the variables have finite or infinite domains. In the former case, the consistency check is NP-Complete, while in the later case it can be done in $O(n^3)$ time for an n variable plan. Finally, we can also prune a plan if

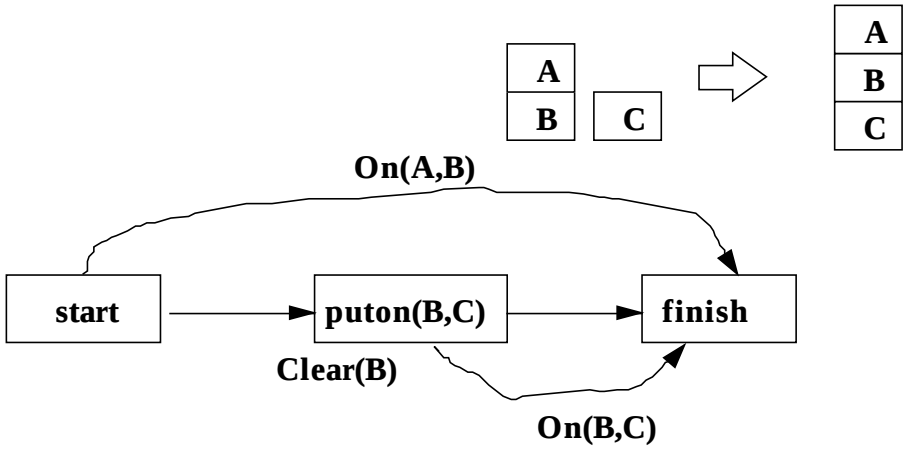


Figure 2: Example illustrating the use of domain specific inconsistency checks as a way of pruning loops.

there exists no ground linearization that is safe with respect to all the causal links of the plan. This check is useful when the underlying planners use causal links, but postpone resolution of the conflicts with the causal links [9, 17]. Checking the link inconsistency of an arbitrary partial plan is NP-complete [17].

Sometimes, the constraints on the plan themselves may not be inconsistent, but may be inconsistent together with some implicit domain knowledge. Admissible pruning is possible even in such situations, as long as relevant knowledge about the domain is available. The pruning techniques are then called ‘‘domain-dependent.’’ Often such domain dependent pruning techniques can prune a plan earlier than the domain independent ones. An an example consider the simplified blocks-world problem of achieving $On(A, B) \wedge On(B, C)$ starting from an initial state where A is on B, and C is on the table [10]. A a causal link planner such as SNLP [13] may generate the partial plan shown in Figure 2 in solving this problem:
In this plan, the goal condition $On(A, B)$ is being established from the effects of the initial state, and a new step $Puton(B, C)$ is added to achieve the second goal $On(B, C)$. Given the blocks world domain axiom that a block cannot both be clear and support another block, we can show that this partial plan cannot be refined into a solution (even though its constraints by themselves are not inconsistent). To see this, consider the state of the world preceding $Puton(B, C)$ in any eventual

solution plan. This state should contain both $Clear(B)$ (which is a precondition of the action), and $On(A, B)$ (which is protected by the causal link $s_0 \overset{On(A,B)}{\rightarrow} s_\infty$). The infeasibility of this can be detected with the help of the plan structure, and the appropriate blocks world domain axiom.

Although applying this strategy at every refinement could be costly, in [10], we show that combining this strategy with an explanation based learning framework can significantly improve planning performance.

4 Minimality based pruning for non-causal link planners

As I discussed earlier, pruning techniques based on constraint inconsistency alone are not enough to stop looping in many situations. In this section, I will develop a technique for pruning partial plans that are *likely* to lead to non-minimal solutions. To do this, we need to extend the notion of minimality to cover incomplete plans.

Definition 1 (Nsatplan**)** *Given an incomplete plan $\mathcal{P}$, we define* Nsatplan($\mathcal{P}$) *as the plan derived from $\mathcal{P}$ by removing all the preconditions of all the steps of $\mathcal{P}$ (including s_∞) that are not necessarily true in $\mathcal{P}$.*

By definition, Nsatplan is a complete plan (since all its preconditions are necessarily true).

Figure 3 illustrates the Nsatplan definition in the car-door example. The partial plan on the top has several steps. Except of the precondition $have(key)$ of the first `open-door` step, all the other step preconditions are made necessarily true by the effects of the preceding steps. Thus, the Nsatplan of this plan, shown in the bottom, contains all preconditions except $have(key)$ condition of the first step.

Using the notion of Nsatplan, we now generalize the notion of plan minimality to incomplete plans as follows:

Definition 2 (Minimal incomplete plans) *An incomplete plan $\mathcal{P}$ is said to be minimal if the complete plan* Nsatplan($\mathcal{P}$) *is minimal.*

The car-door plan at the top of Figure 3 is non-minimal by this definition, since the Nsatplan will give $open(door)$ at goal step, even if we remove the last two steps of the plan.

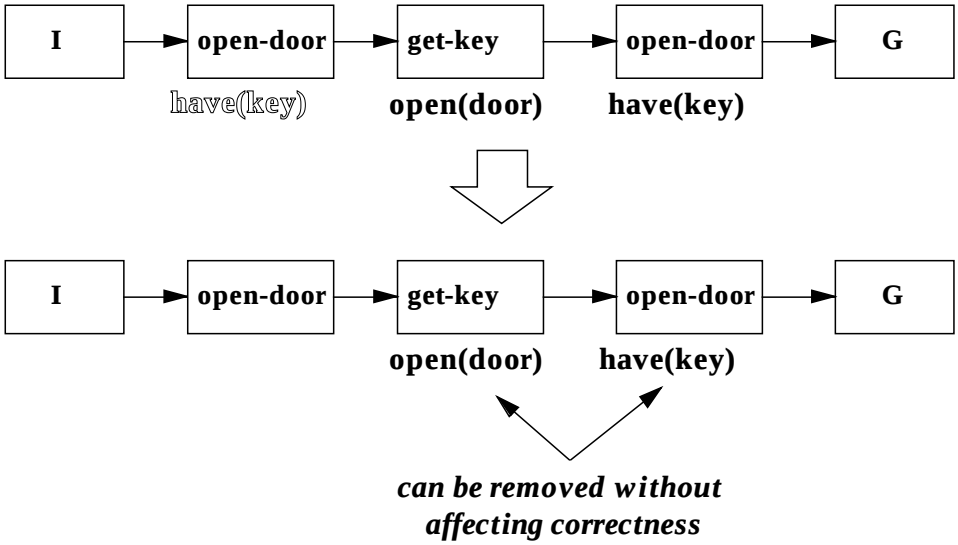


Figure 3: Example illustrating the operation of Nsat-prune

Note that this definition subsumes the definition of minimality of complete plans, since if $\mathcal{P}$ is a complete plan, then Nsatplan($\mathcal{P}$) = $\mathcal{P}$. Armed with this definition, we now have a plausible technique for search reduction via pruning of non-minimal incomplete plans:

Pruning Strategy 1 (Nsat-prune**)** *Prune any partial plan $\mathcal{P}$ from the search space, if* Nsatplan($\mathcal{P}$) *is not a minimal plan.*

This technique clearly does ensure termination in the case of the hf/he example discussed in the introduction. In particular, as soon as the partial plan

$$s_0 \prec s_3{:}O_2 \prec s_2{:}O_1 \prec s_1{:}O_2 \prec s_\infty$$

is produced, Nsat-prune prunes it, (since the Nsatplan does not contain the precondition $hf@s_3$, and consequently, $s_3{:}O_2$ itself is enough to give he to s_∞, making the other two steps redundant), thereby avoiding the looping behavior. Figure 3 shows another application of Nsat-prune.

There are however two important considerations remaining: We need to make sure that Nsat-prune is admissible, i.e., it does not affect the completeness of the underlying planner. We also need to look at the costs associated with the pruning strategy itself. I will look at these in turn.

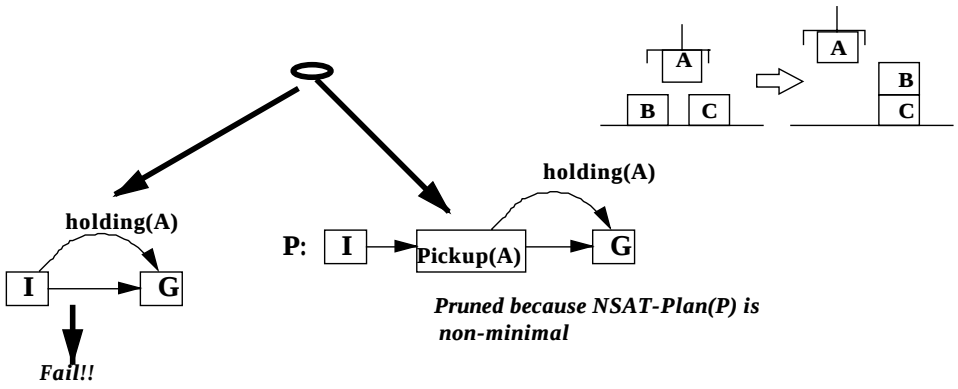


Figure 4: Example showing that Nsat-prune is inadmissible for causal link planners such as SNLP

Intuitively, the reason we believe Nsat-prune may be admissible is because given any solvable planning problem, and a minimal solution P_s for the problem, it is possible in principle to come up with a sequence of refinements of the null plan such that each refinement is a minimal incomplete plan, and the refinements culminate in P_s. Intuitively, this holds because we can always remove steps from P_s in such a way that each resultant incomplete plan is a minimal incomplete plan. So, adding the steps in reverse order of their removal gives us a sequence of nonminimal plans.

Although a sequence of refinements comprising exclusively of minimal incomplete plans exists, it does not guarantee that such a refinement sequence will actually exist in the search space of a specific refinement planner. In particular, most plan-space planners refine partial plans by concentrating on establishing the preconditions of individual steps [9]. The order in which preconditions are considered for establishment is called the *goal order*. We can show that any planner that considers all possible goal orderings will eventually generate the sequence of minimal refinements leading to the solution, and is thus complete. However, since the refinements used in plan-space planning are complete for *any* goal ordering [9], most plan-space planners *do not* backtrack on goal ordering.

I will look at the admissibility of Nsat-prune for two broad classes of plan-space planners discussed in Section 2 -- causal link planners which protect their establishments, and non-causal link planners which reestablish each precondition as many times as necessary.

Causal-Link Planners: Nsat-prune leads to loss of completeness for causal-link planners such as SNLP, and UCPOP. To see this, consider the blocks world

example shown in Figure 4. In this problem the robot arm is holding block A in the initial situation, and blocks B and C are on the table. The goal state is to have B on C, and hold the block A. Suppose the causal link planner, say SNLP, works on this problem by first working on the precondition $holding(A)@s_\infty$. In this case, it generates two refinements, one in which $holding(A)$ is established from the initial state, and the second in which $holding(A)$ is established with the help of a new step `pickup(A)`. It is easy to see that although the former is a minimal incomplete plan, it cannot be refined into a complete plan since in the final plan initial state cannot give $holding(A)$ to the goal state. The second plan, containing `pickup(A)`, is a non-minimal incomplete plan by our definition.[2] and is thus pruned. This leads to loss of completeness.

Non-Causal Link Planners: The reason for inadmissibility of Nsat-prune for causal-link planners is that there is a refinement of the non-minimal incomplete plan $s_0 \prec s_1$: `pickup(A)` $\prec s_\infty$, viz., $s_0 \prec s_4$: `putdown(A)` $\prec s_3$: `pickup(B)` $\prec s_2$: `Stack(B,C)` $\prec s_1$: `pickup(A)` $\prec s_\infty$, which is a minimal solution. Since causal link planners work on each precondition at most once, and protect the establishments in each search branch, pruning the non-minimal plan effectively prunes all the minimal solutions under that non-minimal plan from the search space.

The foregoing discussion raises the possibility that Nsat-prune may be admissible for non-causal link planners that do not protect establishments, and work on preconditions as many times as required until the plan becomes complete. Let us consider the case of a non-causal link planner, such as TWEAK [3], UA or TO [15] solving the above blocks world problem. To begin with, standard implementations of TWEAK [3] and UA [15] work only on preconditions that are not necessarily true, and thus will not even consider $holding(A)$ for establishment at the beginning. However, we argue in [9] that this goal selection strategy is a design choice orthogonal to their non-causal link nature. So, let us consider what happens if they do select $holding(A)$. Once again TWEAK/UA generates the two refinements one in which the initial state establishes the condition, and the other in which the step `pickup(A)` establishes the condition. The latter plan will still be non-minimal incomplete plan and will be pruned. Unlike the

[2]To see this, note that only the precondition $holding(A)@s_\infty$ is necessarily true [3]. The Nsatplan for this plan is thus going to have the same steps and orderings, but will have the single precondition $holding(A)@s_\infty$. The Nsatplan is non-minimal since its only goal can also be satisfied by simple establishment from the initial step, thus making the `pickup(A)` redundant and removable.

causal-link planners however, the non-causal link planners can refine the former plan into a solution. In particular, when working on $On(B, C)$, the planner adds the step `stack(B,C)`, and then the steps `pickup(B)` to give $holding(B)$ to `stack(B,C)`, and `putdown(A)` to give $armempty$ to `holding(B)`. It can be seen that every one of these refinements produces minimal incomplete plans. At this point, the planner notices that the condition $holding(A)$ is no longer true at the goal state, and adds the step `pickup(A)` after `stack(B,C)` to establish it, thus giving the minimal complete solution to the problem.

Notice that this was possible because there are two refinement paths from the null plan to the minimal solution, one through the non-minimal incomplete plan $s_0 \prec s_1$: `pickup(A)` $\prec s_\infty$ and the other through the minimal incomplete plan $s_0 \prec s_\infty$. Although Nsat-prune prunes the former refinement path, it leaves undisturbed a path through the latter, thus retaining completeness. In particular, we have the following theorem:

Theorem 1 Nsat-prune *is an admissible pruning strategy for UA and TO [15]*

The proof of the theorem follows from an extension of the proofs of completeness of UA and TO planners [15]. In particular, we can show that given an incomplete plan $\mathcal{P}$ such that Nsatplan($\mathcal{P}$) is minimal, and a minimal complete plan $\mathcal{P}^s$, there exists a one-step TO/UA refinement of $\mathcal{P}$, $\mathcal{P}'$, such that Nsatplan($\mathcal{P}'$) is minimal, and $\mathcal{P}'$ is a subplan of $\mathcal{P}^s$. Given this, and the fact that the null plan $\mathcal{P}_\emptyset$ is a minimal incomplete plan, the theorem follows by induction on the number of steps in the plan. Perhaps surprisingly, the theorem above does not hold for TWEAK [3]. Here is a counter example:

Example showing that Nsat-prune **is inadmissible for TWEAK:** Consider a domain that has two operators O_1 and O_2. O_1 has an effect p. O_2 has an effect q, but it deletes p. Neither operator has any precondition. Our problem is to achieve $p \wedge q$ starting from an empty initial state. Suppose TWEAK decides to work on the goal p first. It produces the single plan $s_0 \prec s_1: O_1 \prec s_\infty$. Next, it works on the goal q, and produces the single plan $s_0 \prec \binom{s_1:O_1}{s_2:O_2} \prec s_\infty$. At this point, the Nsat-prune strategy will prune this plan (since $p@s_\infty$ is no longer necessarily true in the plan, and removing it makes the step O_1 redundant, and thus the Nsatplan non-minimal), leading to the loss of completeness.

The example does not pose a problem for UA since UA orders O_1 with respect to O_2 as soon as O_2 is introduced (resulting in two partial plans, one of which is pruned by Nsat-prune, while the other leads to the solution. The reason turns out to be that in TWEAK, a condition that is not necessarily true may be possibly

true, while in UA and TO a condition is either necessarily true or necessarily false (Minton et. al. term this property ''unambiguousness'' [15]).

4.1 Cost of Nsat-prune

The cost of Nsat-prune depends on the cost of constructing the Nsatplan from a given incomplete plan, and the cost of checking the minimality of a complete plan. Constructing the Nsatplan involves checking the necessary truth of each precondition of the plan, and can be done in polynomial time for plans containing actions in TWEAK representation [3]. Checking whether a given complete plan is minimal is unfortunately NP-hard even for plans in TWEAK action representation (see [6] for a proof). However, it is possible to formulate weaker conditions that provide necessary but insufficient conditions for minimality. Two such constraints are unbloated plans and greedily justified plans. A complete plan is called **unbloated** or *well-justified* plans (c.f. [14]) if it is not possible to remove a single step from the plan while keeping the plan correct. We can check whether a plan is unbloated in time polynomial in the length of the plan. In particular, if n is the number of steps in the plan, we can check unbloatedness in $O(n^4)$ time for TWEAK action representation. An unbloated plan may still be non-minimal because a non-singleton subset of steps can be removed from it without losing correctness. Fink and Yang describe a set of stronger conditions for minimality check called *greedily justified plans*.[3] A plan is not greedily justified, if it is possible to keep the plan complete after removing a single step s, and all the steps whose preconditions become unsatisfied because of the removal of s, as well as those whose precondition become unsatisfied because of the removal the second set of steps and so on iteratively, until all the remaining steps have their preconditions satisfied. Greedy justification can be checked in $O(n^5)$ time for TWEAK action representation.

In Section 6, we will show that the use of weaker checks of non-minimality is quite effective in reducing looping in many domains. In fact, the results demonstrate a meta-reasoning tradeoff inherent in using minimality based pruning--although the use of stronger minimality checks increases the pruning power of Nsat-prune, it does not necessarily result in increased performance improvements.

[3] Fink and Yang also consider unbloated plans in their paper; they calle these *well-justified* plans

5 Minimality-based pruning for causal link planners

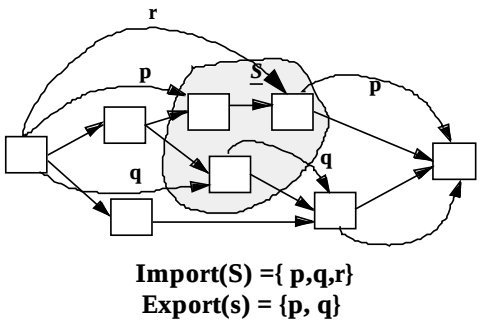
In the previous section I showed that Nsat-prune is inadmissible for causal link planners such as SNLP. As noted there, the lack of redundancy in the search space of causal link planners means that a partial plan cannot be pruned as long as it can *eventually* be refined into a minimal solution.

To find a pruning strategy for such planners, we take a different route that involves using the causal dependencies encapsulated by the causal links. The causal links can be used to understand the role played by any part of the plan in ensuring the completeness of the overall plan. In particular, given a set of steps in the plan, the causal links of the plan which originate from steps outside of this set, but terminate on steps in this set can be seen as the conditions being *imported* by this set of steps. We can define the exports of a set of steps in the plan in an analogous way. Imports and exports can be used to determine the non-minimality of a complete plan. Informally, we can prune a plan whenever some part of the plan ''imports'' more conditions than it ''exports.'' We can formalize the notion of imports subsuming exports, in terms of domination. A set $\mathcal{L}$ of causal links is said to **dominate** another set $\mathcal{L}'$ if for every causal link $s_i' \xrightarrow{c'} s_j'$ in $\mathcal{L}'$, there exists a causal link $s_i \xrightarrow{c} s_j$ in $\mathcal{L}$, such that c and c' necessarily codesignate.

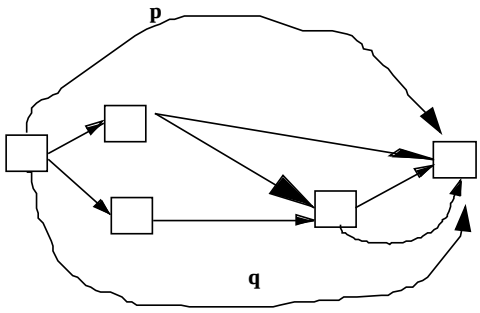
Consider the example plan shown in Figure 5. The shaded section of the plan imports a set of links that dominate the set of links it exports. In such cases, it is clear that the shaded section can be removed from the plan without affecting its correctness. Specifically, we can do this by removing the import links and redirecting the export links. This leads to the following relatively straightforward lemma:

Lemma 1 *Any complete plan P is non-minimal if there exists a subset $S' \subset \mathcal{S}$ of the steps of P such that the set of causal links given by the steps of S' to steps in $\mathcal{S} - S'$ is dominated by the set of causal links taken by the steps of S' from the steps in $\mathcal{S} - S'$.*

Intuitively, this lemma follows because we can remove the steps in S' from the plan P without affecting the correctness of P. In particular, any condition $p@s_e$ (where $s_e \in \mathcal{S} - S'$) that was being supported by a step in S', can still be supported from a step in $\mathcal{S} - S'$. Conceptually, we can understand this in terms of an editing operation on the causal links, such that every pair of causal links $s' \xrightarrow{p} s_e$ and $s_e' \xrightarrow{p} s''$ (where $s', s'' \in S'$ and $s_e, s_e' \in \mathcal{S} - S'$) are replaced by the single link $s_e' \xrightarrow{p} s_e$, thus bypassing the steps in S'.



(a) A complete plan, which has a set of steps
whose exports are dominated by their imports



(b) Plan after link redirection

Figure 5: The plan shown on the left is non-minimal because the shaded parts of the plan have imports that dominated their exports. The plan on right shows the complete plan that results from removing the shaded parts and redirecting the causal links appropriately.

The question is how are we to generalize this observation so that it can apply to *incomplete* plans? The straightforward application of this observation to incomplete plans will not work. For example, we cannot prune an incomplete plan just because the imports of a section of that plan dominate its exports, since the exports might grow as the planning continues, and more effects of this step are used to establish preconditions elsewhere. Figure 6 illustrates this.

Instead, we need to consider a set of links with respect to a step such that the set will never grow as the plan is refined further. To this end we define the **np-cutset** of a step s in a plan P as the set of causal links out-links(s)$\cup$ necessary-cross-links(s) (where a causal link $s_i \xrightarrow{c} s_j \in \mathcal{L}$ belongs to the **out-links** of a step if $s_i = s$, and belongs to the **necessary-cross-links** of a step s, if s is ordered to come necessarily between s_i and s_j). Similarly, the **pp-cutset** of a step s in a plan P is defined as the set of causal links out-links(s)$\cup$ possible-cross-links(s) (where a link $s_i \xrightarrow{c} s_j \in \mathcal{L}$ belongs to the **possible-cross-links** of step s if s can *possibly* come between s_i and s_j (i.e., it comes between s_i and s_j in at least one linearization).

The **pp-cutset** of a step may reduce in size (as additional orderings are

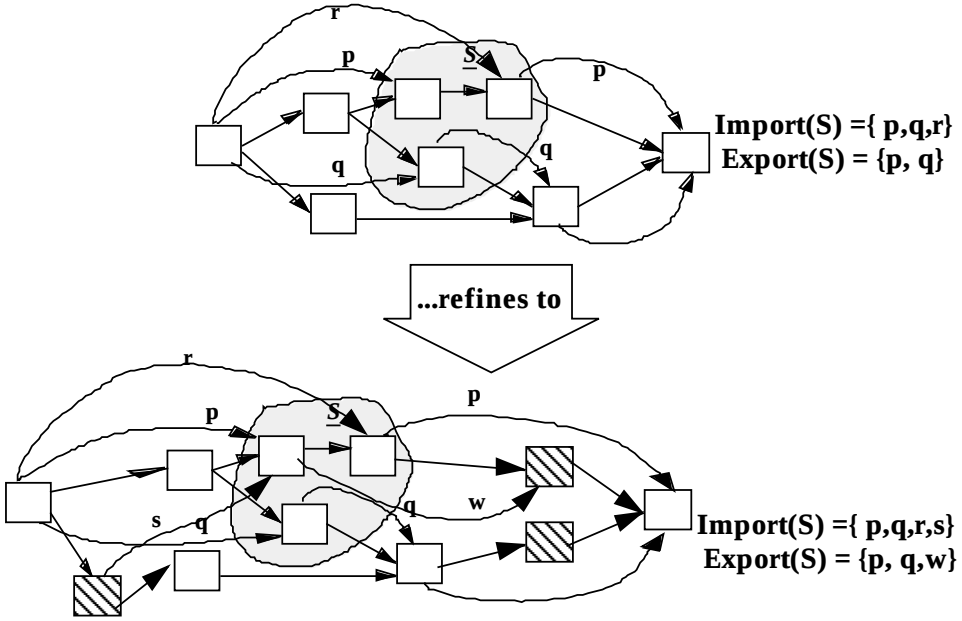


Figure 6: Example showing that incomplete plans cannot be pruned based on domination of exports by imports. The plan at the top has a set of steps (shown in the shaded area), whose imports dominate their exports. However, at the bottom is a refinement of the same plan (with the hatched steps corresponding to newly added steps), where the dominance no longer holds for the steps in the shaded area.

introduced, making steps currently unordered with respect to s to come before it), but the np-cutset will never reduce in size (since refinements can add, but not delete, step and ordering constraints).

In the example plan in Figure 7, the links comprising pp-cutset of the step s'' (shown with the bold border) are shown in dashed lines. The corresponding conditions are P, Q, and R. From the figure, we can also see that the links comprising the np-cutset of the step s' have the conditions P, Q, and R on them. Thus the pp-cutset of s'' is dominated by the np-cutset of s'.

Now suppose that all the remaining open conditions of the plan in Figure 7 are with respect to steps that are necessarily before s'' (equivalently, the preconditions of all the steps s in P such that s can possibly follow s'', have causal links supporting them). In such a case, the pp-cutset of s'' *can never increase in size during planning* (it may reduce in size if the steps that are currently unordered with respect to s'' get ordered to come before s''.) Since the pp-cutset of s'' is currently

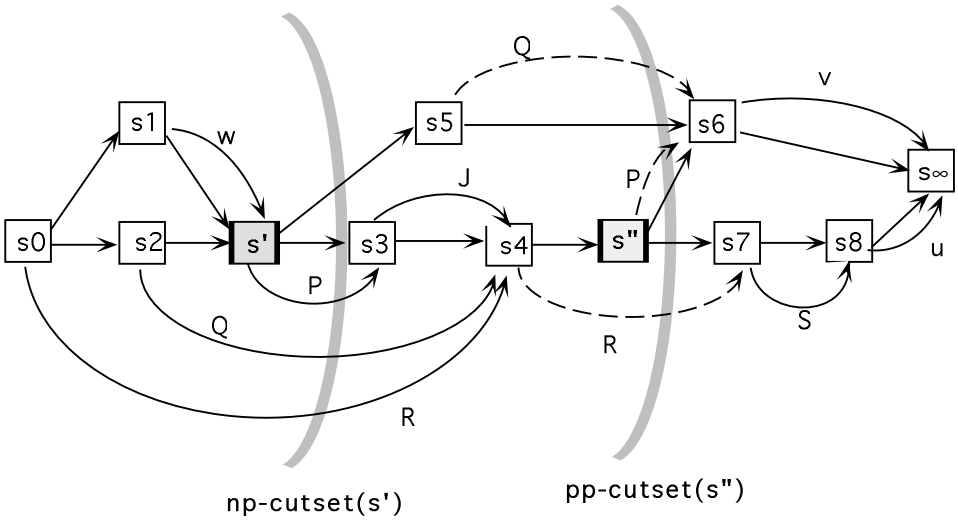


Figure 7: Example illustrating the operation of Cutset-prune strategy. Orderings are shown by straight lines with arrows while the causal links are represented by curved lines, with the condition of the causal link shown explicitly near the link. The **np-cutset** of step s' corresponds to the conditions P,Q and R, while the **pp-cutset** of s'' corresponds to conditions P, Q and R. We assume that there are no open condition corresponding to steps steps s_5, s_6, s_7, s_8 and s_∞. Thus, the **pp-cutset** of s'' cannot increase.

dominated by the np-cutset of s', and the pp-cutset of s'' will not increase, and the np-cutset of a step will not decrease in size, this dominance relation will hold for all refinements of this plan, including any refinements corresponding to complete plans. It is easy to see from the previous discussion that any such complete plans will be non-minimal. Thus, the plan in Figure 7 can be pruned without losing any minimal solutions (complete plans). This pruning strategy is called the Cutset-prune strategy. Formally,

Pruning Strategy 2 (Cutset-prune**)** *Prune any incomplete partial plan P if it has two steps s' and s'' such that (i) s' necessarily precedes s'', (ii) Every open-condition $\langle c, s \rangle$ of the plan is such that s necessarily precedes s'' (iii) np-cutset(s') dominates pp-cutset(s'').*

It is easy to see that Cutset-prune will stop looping in the hf/he example for a causal link planner such as SNLP. In particular, once a plan of type $s_0 \prec s_3{:}O_2 \prec s_2{:}O_1 \prec s_1{:}O_2 \prec s_\infty$ is made, with the causal links $\{s_3 \xrightarrow{he}$

$s_2, s_2 \xrightarrow{hf} s_1, s_1 \xrightarrow{he} s_\infty\}$, the Cutset-prune strategy can prune the plan since pp-cutset of s_1 is dominated by the np-cutset of s_3, and there are no open conditions after s_1. It is also provably admissible for causal link planners. In particular, we have:

Theorem 2 Cutset-prune *is an admissible pruning strategy for planners that protect their establishments via causal links.*

As noted above, the admissibility property follows from the fact that the plans pruned by Cutset-prune will not have any refinements culminating in minimal solutions. We can illustrate it with the example plan shown in Figure 7. Suppose this plan, call it P, is refined into a complete solution plan P'. It is possible to remove the steps s_3, s_4 and s'' (plus a few others) from P' without affecting its correctness. To see this, we start by noting that since there are no open conditions possibly after s'' in P in Figure 7, the only links given by these steps to steps that are possibly after s'' in the complete plan P' are those that they currently give in P. The conditions supported by these links can still be supplied by s', or steps that are necessarily before s'. For example, the link $s_4 \xrightarrow{R} s_7$ can be redirected as $s_0 \xrightarrow{R} s_7$. At the end of this process, the steps s_3 and s_4 will not have any useful causal links emanating from them to steps possibly after s''. Thus, all these steps (and any steps preceding s'' that take any links from them) can be removed without affecting the correctness of P', showing that the plan P' is non-minimal.

Cost of Cutset-prune**:** Since Cutset-prune is quite costly (it needs to potentially look at n^2 pairs of steps, and the check on the cutset domination could be costly when the plan contains variables), it may not be an effective strategy to use at every iteration; it needs to be applied more strategically. Once again, I believe that the primary utility of this strategy will be in terms of its guidance to an underlying learning system (c.f. [10]). In particular, it can be applied to plans crossing depth limits to see if they are provably non-minimal. If so, the explanation of non-minimality can be used to guide an EBL based system to learn effective control rules to avoid the looping branches in the future [10].

6 Empirical Evaluation

To understand the utility of minimality based pruning strategies in planning, we conducted empirical studies. The studies were done on a version of TWEAK

planning algorithm implemented on top of Barrett and Weld's SNLP code [1]. To check the cost vs. benefit ratios afforded by the stronger vs. weaker pruning checks, we implemented three different pruning checks:

BLOAT-PRUNE Prunes incomplete plans that are non-minimal according to the unbloatedness criterion

GREEDY-PRUNE Prunes incomplete plans that are non-minimal according to the greedy justification criterion

HYBRID-PRUNE First applies the BLOAT-PRUNE pruning strategy to the given incomplete plan. If the plan survives BLOAT-PRUNE, then it applies the GREEDY-PRUNE pruning strategy applied to it.

The BLOAT-PRUNE strategy is weaker than GREEDY-PRUNE in that it prunes fewer plans (but does so at a lower cost). The hybrid strategy HYBRID-PRUNE is going to prune exactly the same plans as GREEDY-PRUNE. However, it is expected to be cheaper than GREEDY-PRUNE when most of the pruned plans are bloated plans.

To reduce the performance penalties stemming from the cost of pruning strategies, we use two simple heuristics to restrict the use of pruning checks:

- Pruning checks are done when the node is about to be expanded (and not when it is generated). This way, we avoid the cost of pruning plans which would not any way be considered by the base-level search.
- Pruning checks are restricted to only those plans which were obtained through step addition refinement on their immediate parent. This makes sense since the minimality is defined in terms of plan steps.

Our first experiments were done in an artificial domain called ART-MD-RD, which is defined as follows:

For even i $(Defstep\ A_i\ \underline{precond}:\ I_i, he\ \underline{add}:\ G_i, hf\ \underline{delete}:\ \{I_j | j < i\} \cup \{he\})$
For odd i $(Defstep\ A_i\ \underline{precond}:\ I_i, hf\ \underline{add}:\ G_i, he\ \underline{delete}:\ \{I_j | j < i\} \cup \{hf\})$

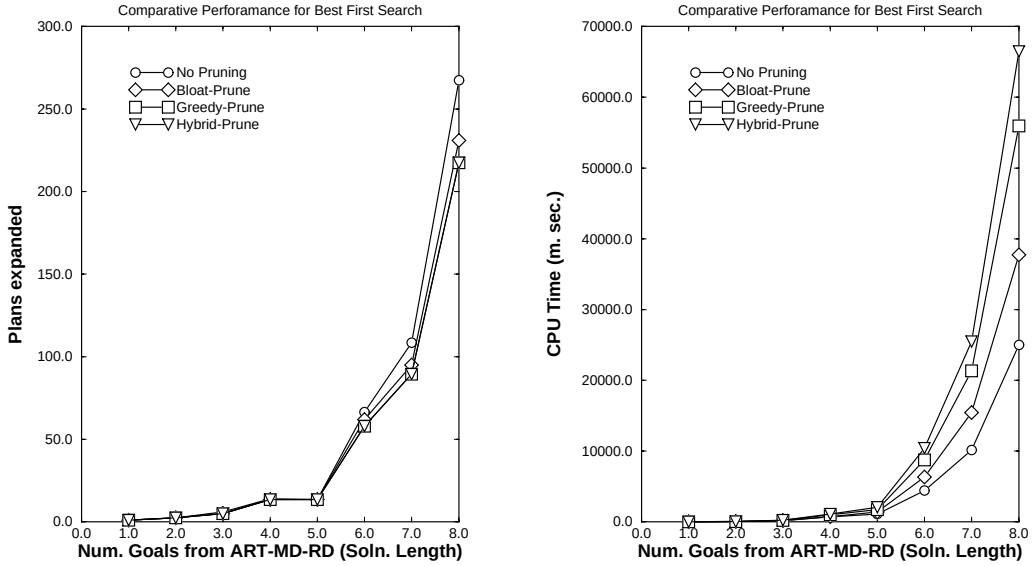


Figure 8: Performance of Pruning strategies in ART-MD-RD domain with Best First Search

This domain is selected because of its potential to cause looping -- it is possible to add odd numbered and even numbered actions indefinitely to provide an alternating chain of hf and he conditions respectively. We have experimented with two search strategies: (i) a best-first strategy with a ranking function that is the sum of the plan-length, and the number of unsatisfied preconditions in the plan and (ii) a *depth-limited* depth-first strategy. Based on our arguments in Section 1, we would expect the effect of pruning strategies to be much more pronounced in the depth-first regime. We looked at *solvable* problems ranging in size from 1 to 8 goals selected from the set $\{G_1 \cdots G_8\}$. In each problem two different goal orderings -- one a FIFO goal ordering and the other a LIFO goal ordering -- were used to solve each problem, and the performance statistics were averaged over. If any of the goal orderings fail to solve the problem within the given resource limits (300 cpu sec. on Sparc-II), the plan is considered unsolved. To understand the costs and benefits of pruning, we compared both the number of nodes generated and the total time (pruning plus node generation) taken for finding the complete (solution) plan. The plots in Figures 8 and 9 compare the performance of the planner under the various search and pruning strategies.

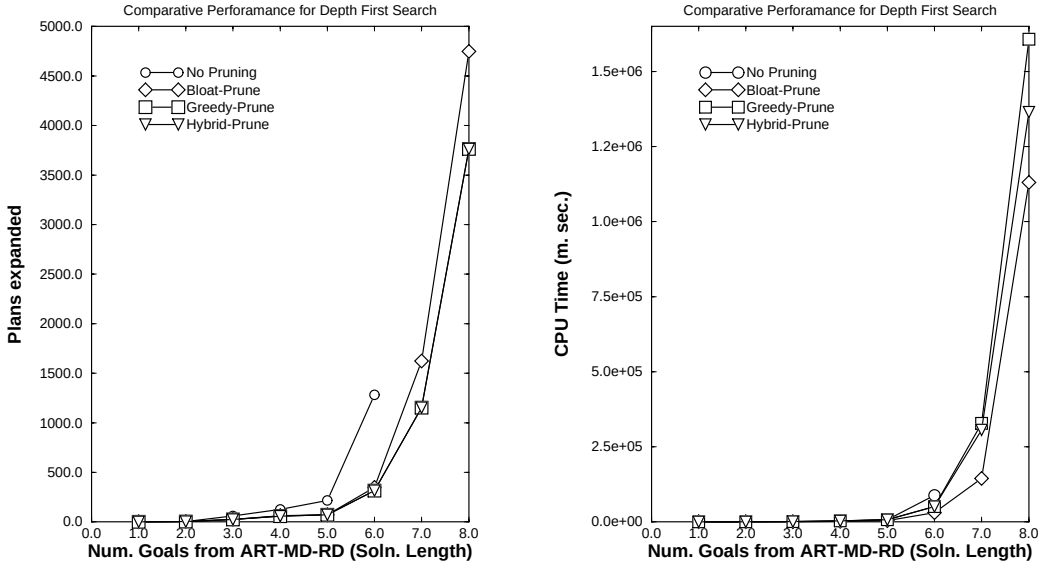


Figure 9: Performance of Pruning strategies in ART-MD-RD domain with Depth First Search

From the plots, we observe the following patterns: For the case of best-first search (Figure 8), we find that the addition of pruning strategies slightly reduces the number of plans expanded. However, this reduction is not enough to offset the cost of pruning strategies, and the planning time *increases* with pruning strategies. In the case of depth-first search (Figure 9) however, the situation is markedly different. Both the number of plans expanded and the cpu. time taken reduce significantly in the presence of pruning strategies. In fact, without the pruning strategies, the planner fails to solve the problems with 7 and 8 goals, within the cpu. time limit.

In terms of the relative utility of the pruning strategies, we find that although the stronger pruning strategies GREEDY-PRUNE and HYBRID-PRUNE reduce the the number of plans expanded, as compared to the weaker BLOAT-PRUNE strategy, they fail to bring about corresponding improvements in the planning times. In fact, in terms of cpu time, GREEDY-PRUNE and HYBRID-PRUNE do worse than BLOAT-PRUNE in both best-first and depth-first strategies. This indicates that the presence of a meta-reasoning tradeoff between the strength of the minimality check and its cost-benefit ratio. As expected, the HYBRID-PRUNE strategy expands the same number of plans as GREEDY-PRUNE. However, it improves cpu time in

the case of depth-first search.

Although bloated pruning strategy does prevail over the other two strategies for solvable problems, the situation changes when we go to unsolvable problems. In particular, we noticed that both the base level planning strategy and the bloated pruning strategy fail to terminate within a reasonable time for even the smallest unsolvable problems in this domain. However, the use of hybrid and/or greedy justification based pruning strategies allow the planner to terminate on these problems. We can understand this by noting that in the case of unsolvable problems, the planner is forced to look at *all* the search branches and prune each of them, before it can terminate. In this case, the residual plans that are left unpruned by bloated pruning strategy keep the planner from terminating. In the case of solvable problems however, the planner is able to get out of these residual branches either with the help of best-first heuristic, or with the help of depth-limit, and still find the solution branch.

We also tested our pruning strategies in blocks world domain; the plots in Figure 10 show the performance of the various pruning strategies in this domain. As can be seen, the general patterns observed in ART-MD-RD domain also holds in the blocks world.

6.1 Experiments with Cutset-prune strategy

We have also experimented with Cutset-prune strategy on top of SNLP implementation, but the results were not as promising. In particular, we found that although Cutset-prune is able to prevent looping in unsolvable problems, its effectiveness is dependent on the goal selection order. In particular, as formulated, the problem with Cutset-prune technique is that it is not applicable unless there exists a step s'' such that all the open conditions are necessarily before it. The frequency with which this happens will depend on the type of goal selection strategy used. In particular, strategies that have FIFO flavor, and work on establishing all top level goals, before working on the subgoals spawned by those establishments, will lead to prunable plans faster than those which have a LIFO flavor. On the other hand, looping occurs more readily in LIFO goal selection strategies than it does in FIFO orders. We have considered relaxing the pruning strategy by using the operator graph of the domain [17]. Specifically, the operator graph structure can be used to compute an optimistic approximation of the **pp-cutset** of the steps -- that is, what are all the *possible* open conditions following s'' that s'' and its possible predecessor steps can support. If the optimistic cutset is in itself

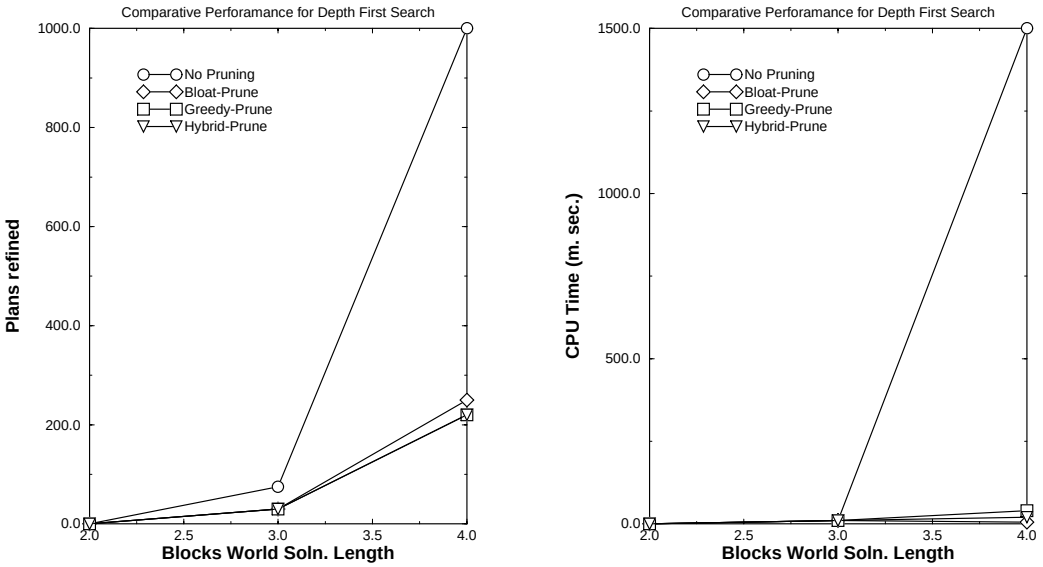


Figure 10: Performance of Pruning strategies in Blocks-world domain with Depth First Search

dominated, then we can be sure that Cutset-prune will eventually hold. Although initial results in this directions were encouraging, we suspended further work as Peot and Smith [18] recently reported an incremental pruning strategy, also based on plan minimality, that seems to improve performance more effectively (see Section 8 for further discussion on this).

7 Related Work

As mentioned earlier, most state space planners, including STRIPS [7] and PRODIGY [2] use state loop and goal loop based pruning strategies. State-loop heuristics prune any path that visits the same world state more than once. Goal-loop heuristics are used in state-space planners that use means-ends analysis, or planners that do backward search in the space of states. These techniques prune any search path where in an attempt to achieve some goal g, the planner spawns a set of subgoals that include g. The state-loop techniques are inapplicable for plan-space planners which do not keep track of world-state during planning. The goal loop strategies turn out to be inadmissible in general (c.f. [8]). In [12],

we present a generalized planner called UCP that combines both state-space and plan-space approaches within a single framework. UCP does have the ability to use the state loop and goal loop pruning, as well as the minimality pruning strategies described in this paper.

Morris et. al. [8] discuss a way of using filter conditions to avoid certain types of looping in partial order planners. Their method depends on an *a priori* analysis of the domain operators to identify the preconditions which should not be expanded. Even this analysis allows loop control in only a very restricted class of situations. Both the Nsat-prune and the Cutset-prune strategies will stop looping on all the examples described in their paper, without requiring any special analysis of filter conditions.

Drummond and Currie [5] describe a pruning strategy called temporal coherence heuristic, which is analogous to the inconsistent state heuristics used by backward state-space planners. This method essentially constrains the planner to work on the goals in the reverse order of their achievement, and thus is complete only when the planner backtracks on all goal orderings. Murray and Yang [19] describe empirical studies that show that the increased branching factor caused by backtracking over goal orderings is typically not offset by the temporal coherence based pruning. In contrast, the minimality based pruning strategies described in our work are complete for a much larger class of planners.

The notion of minimality of complete plans has been around for some time. Often, the proofs of completeness of plan-space planners use the minimality notions to show that the planners are complete for all minimal plans. The particular minimality criterion used in many of these proofs corresponds to what we called ‘‘unbloated plans’’ in this paper (the term first appears in the completeness proof for PEDESTAL [14]). As I mentioned earlier, Fink and Yang [6] formulate a set of tractable necessary but insufficient conditions for checking non-minimality of partial plans. They however do not use their notions of minimality in improving planning performance. As I discussed in Section 4, before the notions of minimality of complete plans can be used to develop pruning techniques, they must first be generalized to incomplete plans.

8 Conclusion

In this paper, I addressed the problem of ‘‘looping,’’ motivated the need for pruning techniques to avoid looping, and showed that looping is intimately tied

to the production of non-minimal solutions. I then proposed two classes of admissible pruning techniques based on the notion of plan minimality. The first one, based on the notion of non-minimal incomplete plans, is admissible for non-causal link planners such as UA and TO, which don't protect goals, but allow reestablishment of goals. The second one is based on the notion of cutset of the causal dependency graph of a plan, and is admissible for causal link planners such as SNLP which protect establishments through causal links. I also discussed the complexity, and utility of the pruning strategies. The development here also brings out the interplay between the redundancy in the search space of a planner and the admissibility of different pruning strategies for it.

Limitations and Future Work: There are several ways the pruning techniques described in this paper can be extended and made more effective. To begin with, checking non-minimality of plans at every refinement could be computationally costly. So, it would be useful to bias the planner in such a way that it will avoid the branches that will lead to non-minimal plans. There are two possible ways of doing this. The first is to make the planner ''learn'' to avoid non-minimal plan generation. Specifically, we are looking into using failure-driven explanation-based learning techniques [10] for this purpose. The idea is to use the proof of non-minimality of the plan as the ''explanation of failure'' of the pruned plan, and use the regression and propagation process of EBL to learn search control rules.

A second approach for biasing the planner against non-minimal plan generation is to make the pruning strategies themselves incremental. In other words, rather than wait until the partial plan becomes provably non-minimal and then prune it, we could try to ''black list'' partial plans that are likely to become non-minimal. Recently, Smith and Peot [18] described a method that ''suspends'' working on recursive preconditions in a partial plan (roughly speaking, an open condition $c@s$ is considered recursive, if it is part of a causal chain that contains another step precondition $c@s'$, that has already been achieved). As planning proceeds, some suspended preconditions may have to be reinstated (because they can also potentially take part in a non-recursive causal chain). Smith and Peot provide experimental results that show that this technique significantly improves the performance of SNLP. It will be intersting to see how their general approach can be adapted to non-causal link planners.

A Proofs

Theorem 3 Nsat-prune *is an admissible pruning strategy for UA and TO [15]*

Proof: Let $[I, G]$ be a planning problem. We need to prove that for every minimal solution P_s of this problem, there exists a sequence of UA/TO refinements that will pass through minimal incomplete plans, and terminate with P_s.

We will prove this by providing a sequence of n refinements with the desired property that will terminate into P_s. Suppose P_s has n steps. Consider the following sequence of n refinements, that start with the null plan, and successively introduce the steps of P_s into the partial plans according to the refinements allowed by UA/TO.

$$P_\emptyset = P_0, P_1, P_2 \cdots P_i, P_{i+1} \cdots P_n = P_s$$

where all plans are subplans of P_s and P_{i+1} is constructed from P_i as follows: If P_{i+1} is not the solution plan, it must have some preconditions of some steps that are not yet necessarily true. Let the goal selected by UA/TO be the condition[4] c at step s'' in P_i. Since P is a strict subplan of P_s, $c@s''$ must also be a precondition in P_s. Let s_{add} be the last step in P_s that adds c before s'' (such a step exists for any partial plan generated by UA/TO planners [15]). We construct P_{i+1} by adding s_{add} to P, and order s_{add} in such a way that the partial ordering relation on P_{i+1} is a subset of the partial ordering relation on P_s. By construction, P_{i+1} is a subplan of P_s. It is also a 1-step refinement of P as done by UA/TO [15]

To show that this specific refinement sequence will survive Nsat-prune we need to show that all the plans in the sequence are minimal incomplete plans. We already know that $P_\emptyset$ is a minimal incomplete plan, and P_s is a minimal plan. To prove that all the plans are minimal, we will use induction. Specifically, our induction hypothesis is that P_i is a minimal incomplete plan. We will need to show that P_{i+1} is also a minimal incomplete plan.

By construction, P_{i+1} has one additional step s_{add} compared to P_i, added to establish the goal condition that is selected by UA/TO for P_i. For P_{i+1} to be non-minimal, there must be a step which can be removed without increasing the number of false preconditions of the plan. Since s_{add} is added to make a

[4] for the purpose of this proof, we will assume that the goal selection strategy picks only those conditions that are not currently necessarily true. This restriction can be relaxed by allowing P' to be the same as P when the selected condition is already necessarily true.

specific false precondition true, removing it will clearly increase the number of false preconditions. Thus, the only source of non-minimality is that some existing step $s' \in P_i$ became redundant when s_{add} got added. This can happen only if the preconditions that s' made true in P are also being either made true or clobbered by s_{add}. This is not possible since by construction s' must have been added into the previous plans only because it serves a specific useful purpose in P_s.

Theorem 4 Cutset-prune *is an admissible pruning strategy for planners that protect their establishments via causal links.*

Proof: Consider a plan P that satisfies the Cutset-prune criterion. Specifically, suppose that P contains two steps s' and s'' such that (a) s' precedes s'' (b) there are no open conditions possibly following s'' and (c) **np-cutset** of s' dominates **pp-cutset** of s''.

Let P_s be a refinement of P such that P_s is a solution to the planning problem. We will show that it is possible to remove steps from P_s, while keeping it a solution (thus it is non-minimal).

Since P_s is a refinement of P, it will have all the constraints of P, plus possibly more. Thus, P_s will have s' and s'' in it. We will first prove a lemma:

Lemma 2 *The dominance between* **np-cutset**(s') *and* **pp-cutset**(s'') *will hold in* P_s

Proof: We will prove this lemma by showing that as P is refined further, **np-cutset**(s') can never reduce in size, while **pp-cutset**(s'') can never increase in size.

To see that **np-cutset**(s') cannot reduce in size, consider a link $s_1 \xrightarrow{p} s_2$ in the np-cutset. Since by definition $s_1 \prec s' \prec s_2$, and refinement operations only add new constraints, without removing any existing ones, $s_1 \prec s' \prec s_2$ will continue to hold in all refinements of P, including P_s, thus making $s_1 \xrightarrow{p} s_2$ a member of the **np-cutset** of s' in P_s. This means that the **np-cutset** of s' does not reduce in size.

To see that **pp-cutset**(s'') does not increase in size, we start by noting that all the open conditions are necessarily preceding s''. Since new causal links can only be added to support open conditions, any causal links added to P as a result of refinement will all have the property that their consumer steps will be necessarily preceding s''. Thus, no new causal links will be added to the **pp-cutset** of s'' as

P is refined to P_s. Further, since the refinement operations can only add new orderings without deleting existing ones, a link that is currently in the **pp-cutset** of s'' will continue to do so as the plan is being refined. Thus, the **pp-cutset** of s'' can only reduce in size.

Since **np-cutset**(s') does not reduce in size, and since **pp-cutset**(s'') does not increase in size, the dominance relation that holds between the two in P will continue to hold in P_s. □

Given the lemma above, we can now show that if P is refined into a solution P_s, then P_s must be a non-minimal solution. For this, we need to show that it is possible to remove steps from P_s without affecting its correctness.

Consider the following transformation: For every casual link $l_1 : s_i \xrightarrow{C} s_j$ in **np-cutset**(s'), consider the corresponding causal link $l_2 : s_1 \xrightarrow{C} s_2$ in **pp-cutset**(s'') (this must exist given the dominance relation between the two). Remove l_1 and l_2 from P_s, and add $l' : s_i \xrightarrow{C} s_2$ in their place. Repeat this process for every causal link in **pp-cutset**(s'').

After the transformation above, there will be no causal link $s_i \xrightarrow{C} s_j$ in the plan such that s_i comes possibly between s' and s'' and s_j comes after s''. This means that s_j as well as every step s_i in the plan such that $s' \prec si \prec s_j$ can be removed from the plan. Let P'_s be the resulting plan.

To show that P'_s is correct, we note that our transformation of P_s into P'_s does not introduce any open conditions (i.e., preconditions of the plan steps that are not supported by causal links). Thus, the only way P'_s could be incorrect is for it to have an unsafe link. Since P'_s does not contain any newer steps not in P_s, all the causal links in P'_s that were present in P_s will continue to be safe in P'_s. This leaves the causal links that were introduced as a part of the link replacement transformation. Suppose the link $l' : s_i \xrightarrow{C} s_2$ is added in place of $l_1 : s_i \xrightarrow{C} s_j$ and $l_2 : s_1 \xrightarrow{C} s2$ (see above). We know that l_1 and l_2 were safe causal links in P_s, and we need to show that l' is a safe causal link in P'_s (i.e., , there are no steps that can come between s_i and s_2 and violate C). For l' to be unsafe, there must exist a step that is possibly between s_i and s_2 and deletes C. Such an s_t must be necessarily after s_j and necessarily before s_1, since otherwise, l_1 and l_2 would have been unsafe. Since l_1 and l_2 must have belonged respectively to the **np-cutset** of s' and **pp-cutset** of s'', it must be the case that s_t is possibly between s' and s''. But, we have already removed all the steps possibly between s' and s'' as part of our transformation. Thus, $s_i \xrightarrow{C} s_2$ cannot be unsafe. Since every open condition still has a causal link and all causal linsk are safe, the plan P'_s is correct. Thus, P_s is

non-minimal.□.

References

[1] A. Barrett and D. Weld. Partial Order Planning: Evaluating Possible Efficiency Gains *Artificial Intelligence*, Vol. 67, No. 1, 1994.

[2] J. Blythe and M. Veloso. An analysis of Search Techniques for a totally-ordered nonlinear planner. In *Proc. 1st Intl. Conf. on AI Planning Systems*, 1992.

[3] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333--377, 1987.

[4] E. Cohen. Understanding the Utility of Pruning by Minimality in partial order planning. MCS Project Report. ASU CSE dept. May 1993.

[5] M. Drummond and K. Currie. Exploiting Temporal Coherence in Nonlinear Plan Construction. In *Computational Intelligence*, Vol. 4, 1988.

[6] E. Fink and Q. Yang. A Spectrum of Plan Justifications. In *Proc. Canadian AI Conference*, 1992.

[7] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251--288, 1972.

[8] R. Feldman and P. Morris. Admissible criteria for loop control in planning. In *Proc. AAAI 1990*.

[9] S. Kambhampati, C. Knoblock and Q. Yang. Planning as Refinement Search: A Unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence* special issue on Planning and Scheduling. Vol. 76, No. 1-2, 1995.

[10] S. Kambhampati, S. Katukam and Y. Qu. Failure Driven Dynamic Search Control for Partial Order Planners: An Explanation Based Approach. *Artificial Intelligence*, To appear. (Available from http://rakaposhi.eas.asu.edu:8001/yochan.html).

[11] S. Kambhampati. Admissible pruning Strategies based on Plan minimality for plan-space planning In *Proc. IJCAI-95*, 1995.

[12] S. Kambhampati and B. Srivastava.[5] Universal Classical Planner: An Algorithm for unifying state-space and plan-space planning. ASU CSE TR 94-002.

[13] D. McAllester and D. Rosenblitt. Systematic Nonlinear Planning. In *Proc. 9th AAAI*, 1991.

[14] D. McDermott. Regression Planning. *Intl. Jour. Intelligent Systems*, 6:357-416, 1991.

[15] S. Minton, J. Bresina and M. Drummond. Total Order and Partial Order Planning: a comparative analysis. Journal of Artificial Intelligence Research 2 (1994) 227-262.

[16] J.S. Penberthy and D. Weld. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *Proc. KR-92*, November 1992.

[17] D.E. Smith and M.A. Peot. Postponing threats in partial-order planning. In *Proc. Eleventh AAAI*, 1993.

[18] D.E. Smith and M.A. Peot. Suspending Recursion in Planning. In *Proc. AIPS-96*, 1996. In press.

[19] Q. Yang and C. Murray. An evaluation of the temporal coherence heuristic in partial-order planning. *Computational Intelligence Journal*, 10(3), 1994.

[5] Technical reports available via URL `ftp://rakaposhi.eas.asu.edu/pub/rao/papers.html`