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:** ftp://rakaposhi.eas.asu.edu/pub/rao/rao.html

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.

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; 14], 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.

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 6), 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 find plan-space planners faring better than state space planners in many domains [1; 14].

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; 16]. 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; 16] 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

[*]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.

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 following simple example.

The hf/he Example: The problem 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.

Clearly, this problem has no solutions. Both forward and backward state-space planners will recognize this immediately and terminate.[1] 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.

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.[2] Doing this, while retaining soundness and completeness of the underlying planner, turns out to be a tricky proposition.

[1]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.

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

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 [14; 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 6 discusses related work and Section 7 summarizes the paper's contributions. Due to space limitations, I provide sketches of all proofs. The complete proofs may be found in [11].

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; 14] 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 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.

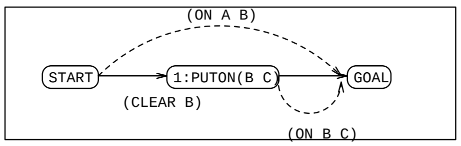
A precondition p of a step s in a partial plan is said to be **necessarily true** [3] (or satisfied) if in every ground linearization of the plan, there is some step s' that precedes s and gives p, and no step between s' and s deletes p. Similarly, if at least one ground linearization satisfies these 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 [15]. A second class of planners, such as TWEAK [3], UA and TO [14], 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 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 following partial plan 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 $s_0 \xrightarrow{On(A,B)} s_\infty$). 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).

As an example, consider the blocks world problem where 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. Consider the incomplete plan

$$\mathcal{P} : s_0 \prec s_1 \colon \texttt{pickup(A)} \prec s_\infty,$$

where the preconditions of the plan are $On(B, C)@s_\infty$, $holding(A)@s_\infty$, $handempty@s_1$, $clear(A)@s_1$. Of these, 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 $holding(A)@s_\infty$.

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 blocks world plan in the above example is non-minimal by this definition, since the only precondition of its Nsatplan $holding(A)@s_\infty$ is satisfied even if we remove the step s_1: pickup(A).

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

$$s_0 \prec s_3 \colon O_2 \prec s_2 \colon O_1 \prec s_1 \colon O_2 \prec s_\infty$$

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.

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.

Intuitively, the reason we believe Nsat-prune may be admissible is because of the following (fairly obvious) theorem:

Theorem 1 *Given any solvable planning problem,* P, *for every minimal solution $\mathcal{P}$, there exists a sequence of partial plans $\mathcal{P}_1, \mathcal{P}_2..\mathcal{P}_n$ such that $\mathcal{P}_1$ is the null plan, $\mathcal{P}_n$ is equivalent to $\mathcal{P}$, and $\mathcal{P}_{i+1}$ can be derived from $\mathcal{P}_i$ by establishing some precondition $\langle p, s\rangle$ in $\mathcal{P}_i$, and for every plan $\mathcal{P}_i$ in the sequence,* Nsatplan($\mathcal{P}_i$) *is a minimal plan.*

Although this theorem says that a sequence of refinements comprising exclusively of minimal incomplete plans exists, it does not guarantee that such a refinement sequence exists for every possible goal selection order. In particular, using this theorem, we can show that any planner that backtracks over all possible goal orderings (i.e., the order in which the goals are established) will be complete. However, since the refinements used in plan-space planning are complete for *any* goal ordering, 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 discussed above. 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, 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_∞, 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 [14] solving the above blocks world problem. When the planner works on the precondition $holding(A)$,[3] once again it 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 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 solu-

[3]Although non-causal link planners such as TWEAK [3] and UA [14] are traditionally formulated as planners that work only on preconditions that are not necessarily true, this is a design choice orthogonal to their non-causal link nature; see [9].

tion, one through the non-minimal incomplete plan $s_0 \prec s_1{:}\texttt{pickup(A)} \prec s_\infty$ 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 2 Nsat-prune *is an admissible pruning strategy for UA and TO [14]*

The proof of the theorem follows from an extension of the proofs of completeness of UA and TO planners [14]. 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: 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'' [14]).

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. Yang and Fink [6] provide a series of polynomial time necessary but insufficient conditions for plan minimality. It should be easy to see that instantiations of Nsat-prune that use such weaker conditions will be admissible in all the cases where the pruning strategies based on full minimality are admissible.

In [4], it is shown 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.

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, and prune a plan when some part of it ''takes'' more causal links than it ''gives''. Recall that a causal link $s_i \xrightarrow{c} s_j$ can be seen as a commitment that the step s_i gives condition c to the step s_j. 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. We start by noting 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'.

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 set of links given by a step s is dominated by the set of links taken by it, since the former set might grow as the planning continues, and more effects of this step are used to establish preconditions elsewhere.

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).

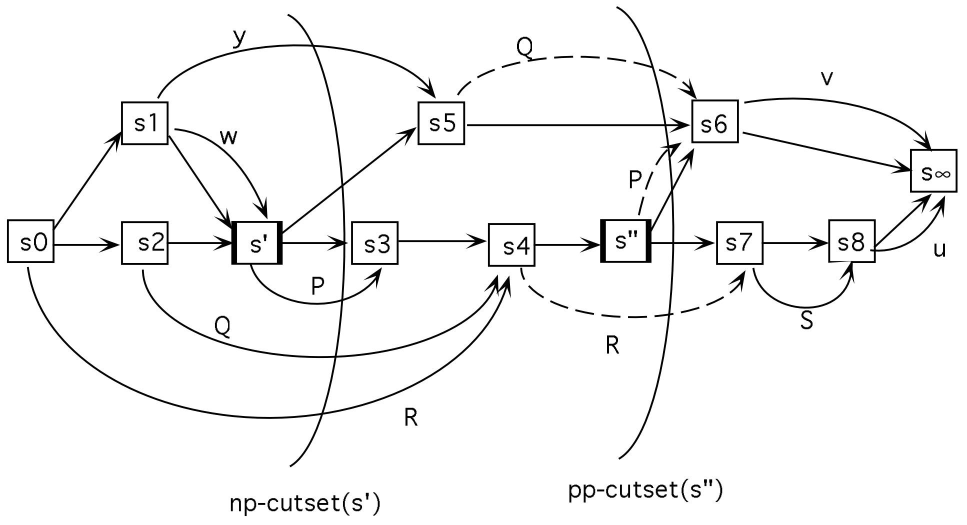


Figure 1: Example illustrating the operation of Cutset-prune strategy. Orderings are shown by straight lines with arrows while the causal links are represented by curved lines, with the condition of the causal link shown explicitly near the link. We assume that there are no open condition corresponding to steps steps s_5, s_6, s_7, s_8 and s_∞.

The **pp-cutset** of a step may reduce in size (as additional orderings are 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 1, 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 1 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 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 that any such complete plans will be non-minimal by Lemma 1. Thus, the plan in Figure 1 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 3 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 1. 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 1, 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; 16].

6 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 [18] 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.

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.

7 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.

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. ASU-CSE-TR 94-002. To appear in *Artificial Intelligence* special issue on Planning and Scheduling. 1995.

[10] S. Katukam and S. Kambhampati. Learning explanation based search control rules for partial order planning. In *Proc. AAAI-94*, 1994.

[11] S. Kambhampati. Admissible pruning Strategies based on Plan minimality for plan-space planning: The details. ASU CSE Tech. Report., 1995 (in preparation).

[12] S. Kambhampati and B. Srivastava.[4] 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] 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.

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

[16] Y. Qu and S. Kambhampati. Learning Control-rules of expressive plan-space planners: Factors affecting the performance. ASU-CSE-TR 94-006.

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

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

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