A Spectrum of Plan Justifications

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

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

Abstract

This paper formalizes the notion of *justified plans*, which captures the intuition behind "good" plans. A plan is called justified if it does not contain operators that are not necessary for achieving a goal. We explore several different types of justification, present algorithms for removing "useless" operators from a plan, and show that the task to remove *all* useless operators is NP-complete.

1 Introduction

When we search for a plan to achieve certain goals, we wish to find a plan that does not contain "useless" steps. In other words, we wish to optimize the plan by removing all operators that are not necessary for achieving the goals. For example, suppose that we prepare tea by following the plan: "put a tea bag into a cup; boil water in a kettle; pour water into the cup". If later we discover that the water in the kettle is already hot, then the second step of the plan, "boil water", is no longer necessary. After removing the second step, the resulting plan, "put a tea bag into a cup; pour water into the cup", contains fewer steps while still achieving the same goal. The operation of removing useless operators from a plan is known as *justification*. The purpose of our paper is to formalize different ways of performing justification.

The justification algorithms described in this paper may be used to augment a non-optimal planner such as STRIPS [Nilsson, 1980]. Such an augmentation is especially useful for planning with macro operators [Korf, 1985]. Another application of justification algorithms is in reusing old plans. Suppose that we have found a plan for achieving goals G_1 and G_2. Later we may use the same plan to achieve the goal G_1 alone. In this case we wish to find the subset of the initial plan which is "relevant" to achieving G_1, by removing all unnecessary operators. Planning systems that reuse old plans [Hammond, 1986], [Kambhampati and Hendler, 1992], [Hanks and Weld, 1992] can benefit from justification algorithms.

Different definitions of justification were presented in [Tenenberg and Yang, 1990], [Knoblock *et al.*, 1991], and [Bacchus and Yang, 1991], where this notion was used for formalizing several important properties of abstraction hierarchies. A further formalization of justification was presented in [Kambhampati, 1992]. However, these papers did not present an algorithm for removing non-justified operators from a plan.

In this paper we extend and unify the previous work and describe two algorithms for finding justified versions of *nonlinear* plans. First we consider the notion of *backward justified* plans, which guarantees that each operator in the plan establishes a literal necessary for achieving the goal. Then we present *well-justified* plans. Informally, a plan is well-justified if no single operator may be removed from the plan. We compare well-justified and backward justified plans. Finally, we consider the task to find the "best possible" justification of a given plan, that is, a subplan of a given plan that cannot be further optimized by removing any subset of its operators. The task of finding such a subplan is NP-complete. To satisfy the practical need for efficient planning, we present a greedy algorithm that finds a near-perfect justification in polynomial time. Proofs of the italicized statements may be found in [Fink, 1992].

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

2 Backward justification

To formalize the notion of justified plans, we first generalize the concept of establishment defined in [Knoblock *et al.*, 1991] to nonlinear plans.

Let α_1 and α_2 be two operators of a correct linear plan, such that α_1 precedes α_2, and l be a precondition of α_2. We say that α_1 *establishes* l for α_2 if l is an effect of α_1 and no operator between α_1 and α_2 either asserts or removes l. We say that α_1 *possibly establishes* a literal l for α_2 in a nonlinear plan Π if it establishes l for α_2 in at least one linearized version of Π.

Intuitively this means that the precondition l of the operator α_2 is satisfied, and α_1 is the last operator that achieves it.

Definition 1 *Let Π be a correct plan. An operator α of Π is called* backward justified *if α possibly establishes some literal l either for the goal of the plan or for another backward justified operator.*

We say that a plan Π is backward justified if all its operators are backward justified. This definition of justification was used for linear plans in [Knoblock *et al.*, 1991]. It is slightly weaker than the justification used in the ABTWEAK planner [Tenenberg and Yang, 1990]. Intuitively, an operator is backward justified if it establishes some literal necessary for achieving the goal. We call this justification *backward* because it has been mostly used in formalizing backward-chaining planners.

However, backward justified operators are not "truly justified". For example, if a literal is achieved twice in a linear plan, then, by definition, the second operator achieving this literal is backward justified. But in this case the second operator is useless. For example, if the same discovery is made by two independent researches, the second one usually does not get a credit. As another example, suppose that you have a kettle with hot water and an empty cup, and you wish to have a cup of hot water. The following plan achieves the goal.

1. Pour water into the cup.
2. Put the cup into a microwave.

The second operator *is* backward-justified, because it makes the water hot, while no other operator *after* it achieves the same goal of making the water hot in the final state. However, this operator still may be skipped, because the water was already hot before its execution. Thus, the second operator is not "truly justified".

The advantage of the backward justification is that it can be computed with a small running time.

Backward_Justification(Π)
1. let $\overline{\Pi}$ be some linearized version of Π;
2. **for** $\alpha :=$ (end of $\overline{\Pi}$) **downto** (beginning of $\overline{\Pi}$) **do**
   begin
3.   *Justified* := *False*;
4.   **for** each $l \in Eff(\alpha)$ **do**
5.   **if** (α establishes l for some α_1 or for the goal)
6.   **then** /* α is justified */ *Justified* := *True*;
7.   **if** *Justified*=*False* /* α is not justified */
8.   **then** remove α from the plan Π;
   end

Table 1: Finding a backward justified subplan of Π

Observe that if α is the last non-backward-justified operator in some linearized version of a plan, it does not establish any literal for any other operator, and thus may be removed without violating the correctness of the plan. Also, after its removal all backward justified operators are still backward justified, and all non-backward-justified operators are still non-backward-justified. Thus, we may remove non-backward-justified operators from a plan, one by one, until we remove all of them. It may be shown that *the resulting plan is backward justified and still achieves the goal*. The algorithm in Table 1 removes non-backward-justified operators from a plan in $O(E \cdot |\Pi|^2)$ time, where $|\Pi|$ is the number of operators in Π, and E is the number of effects of all operators in the plan, $E = \sum_{\alpha \in \Pi} |Eff(\alpha)|$.

The backward justification not only characterizes backward-chaining planners but also relates to execution monitoring. For example, STRIPS' triangle table [Nilsson, 1980] is computed via a backward justification algorithm. Furthermore, the use of the table itself is a kind of backward justification, where an entire tail of a plan is justified by considering the current situation as the first operator.

3 Well-justification

Definition 2 *An operator is* well-justified *if and only if we cannot remove it from the plan without violating the correctness of the plan.*

A plan is well-justified if all its operators are well-justified. This definition captures the intuition behind "good" plans, in terms of individual operators: a well-justified plan does not contain any operator that is not necessary for achieving the goal.

For example, suppose that one has a kettle of cold water and needs a cup of hot water. The following plan would lead to the desired result.

1. Boil water by putting the kettle onto a stove.
2. Pour the water into a cup.
3. Put the cup into a microwave.

This plan is not well-justified, because either the first or third operator may be skipped without violating the correctness. Thus, the plan has two well-justified subplans: one of them consists of the first two operators, and the other consists of the last two.

Recall that if a plan Π is not backward justified, then its last non-backward-justified operator may be removed without violating the correctness of Π, which means that Π is not well-justified either. Thus, *any well-justified plan is backward justified*. In other words, well-justification is stronger than backward justification. It is also stronger than the justification used in ABTWEAK [Tenenberg and Yang, 1990].

We found an algorithm [Fink and Yang, 1992] that computes a well-justified version of a given plan in $O(P \cdot |\Pi|^4)$ time, where $|\Pi|$ is the number of operator in Π, and P is the number of preconditions of all operators in the plan, $P = \sum_{\alpha \in \Pi} |Pre(\alpha)|$.

4 Perfect justification

While well-justified plans cannot contain unnecessary operators, they still may contain unnecessary *groups of operators*. This means that while no single operator may be eliminated from the plan, several operators may be eliminated *together*. For example, consider the following plan of boiling water:

1. Fill a cup with water.
2. Empty the cup.
3. Fill the cup with water again.
4. Put the cup into a microwave.

This plan is well-justified: we cannot skip *operator 2*, because then we could not fill the cup again; and we cannot skip *operator 3*, because the cup has to be full when we put it into a microwave. However, we may skip *operators 2* and *3* together. To formalize this observation, we introduce the notion of a *perfectly justified* plan.

Intuitively, a plan is perfectly justified if no subset of its operators may be removed from the plan. In other words, this is the "best possible" justification.

Definition 3 *A correct plan* Π *is called* perfectly justified *if no subset of its operators may be removed from* Π *without violating the correctness of* Π.

Just by definition perfect justification is stronger than all justifications discussed above. Unfortunately, a perfect justification of a given plan cannot be found in polynomial time.

```
Greedy_Justify_Checking(Π,α)
remove α from Π;
repeat
  Illegals := "the set of illegal operators of Π";
  remove all operators of the set Illegals from Π
until Π does not contain illegal operators;
if Π still achieves the goal
  then return("Π is optimization of initial plan")
  else return("α is greedily justified")
```

Table 3: Checking if α is greedily justified

Theorem 1 *The task of finding a perfect justification of a given plan is NP-hard, even for linear plans.*

A proof may be found in [Fink, 1992].

5 Greedy justification

While the task of finding a perfect justification is NP-hard, one can design a greedy algorithm that finds an "almost" perfect justification. To check the usefulness of some operator α in a plan Π, the algorithm proceeds as follows (see Table 3). First, it removes an operator α from the plan. After α has been removed, some operators of Π may become *illegal*, which means that now some of their preconditions are *not* satisfied before their execution. The algorithm removes all illegal operators and examines the resulting plan. If the plan still contains illegal operators, they are also removed. The algorithm repeats this step until all remaining operators are legal. If the plan still achieves the goal, then the initially removed operator α was not useful, and we say that α is not *greedily justified*.

As an example, we consider our water-boiling plan:

1. Fill a cup with water.
2. Empty the cup.
3. Fill the cup with water again.
4. Put the cup into a microwave.

Let us remove *operator 2*. Now *operator 3* is illegal, because we cannot fill a cup which is already full. So, we remove *operator 3*. We are left with the plan

1. Fill a cup with water.
4. Put the cup into a microwave.

which is legal and achieves the goal. Thus, *operator 2* in the initial plan is not greedily justified.

Observe that if an operator α in a plan is not well-justified, and we use the described algorithm to check the usefulness of α, then α is removed at the

type of justification	**running time to find it**	
perfect justification	NP-complete	stronger justification
greedy justification	$O(P \cdot \lvert\Pi\rvert^5)$	↑
well-justification	$O(P \cdot \lvert\Pi\rvert^4)$	
ABTWEAK justification	$O(P \cdot E \cdot \lvert\Pi\rvert)$	↓
backward justification	$O(E \cdot \lvert\Pi\rvert^2)$	weaker justification

Table 2: Types of justifications and the running time to find them

first step of execution, and the remaining plan is legal and achieves the goal. Thus, if an operator is not well-justified, it is not greedily justified either, and therefore *greedy justification is stronger than well-justification.* All operators that are not greedily justified may be removed from a plan in $O(P \cdot |\Pi|^5)$ time.

6 Conclusion

This paper formalizes the intuition behind "good" nonlinear plans. Table 2 presents different types of justification and their corresponding running times. Justification algorithms for linear plans are much faster. For example, a greedily justified version of a linear plan may be found in $O((P+E) \cdot |\Pi|^2)$ time [Fink Yang, 1992].

The table may be viewed as a spectrum of justified plans. On one end of the spectrum plans are backward justified. A backward justified subplan of a given plan is not hard to find, but it may contain some "useless" operators. The other end of the spectrum contains perfectly justified plans. They cannot have any useless operators, but it is NP-hard to find a perfectly justified subplan of a given plan.

References

[Bacchus and Yang, 1991] Fahiem Bacchus and Qiang Yang. The downward refinement property. In *Proceedings of the Twelveth International Joint Conference on Artificial Intelligence*, pages 268–292, 1991.

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

[Fink Yang, 1992] Eugene Fink and Qiang Yang. Formalizing plan justifications. In *Proceeding of the Ninth Conference of the Society for Computational Studies of Intelligence*, pages 9–14, 1992.

[Hammond, 1986] Kristian J. Hammond. CHEF: a model of case-based planning. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 261–271, 1986.

[Hanks and Weld, 1992] Steven Hanks and Daniel S. Weld. Systematic adaptation for case-based planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 96–105, 1992.

[Kambhampati, 1992] Subbarao Kambhampati. Characterizing multi-contributor causal structures for planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 116–125, 1992.

[Kambhampati and Hendler, 1992] Subbarao Kambhampati and James A. Hendler. A validation structure based on theory of plan modification and reuse. *Artificial Intelligence*, 1992, 2-3, vol. 55.

[Korf, 1985] Richard E. Korf. *Learning to solve problems by search for macro operators.* Pitman Publishing, MA, 1985.

[Knoblock *et al.*, 1991] Craig Knoblock, Josh Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In *Proceedings of the Ninth National Conference of Artificial Intelligence*, pages 692–697, 1991.

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

[Minton, 1985] Steven Minton. Selectively Generalizing Plans for Problem-Solving. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985.

[Nilsson, 1980] Nils J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1980.

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