Appeared in Working notes of AAAI 1990 Spring Symposium on Casebased Reasoning

A Classification of Plan Modification Strategies Based on Coverage and Information Requirements[*]

Subbarao Kambhampati

Center for Design Research and Department of Computer Science
Stanford University
Stanford CA 94305
email: *rao@sunrise.stanford.edu*

1. Introduction

The ability to modify an existing plan to make it conform to the constraints of a new (or changed) planning situation is important for plan reuse, replanning and case-based planning. While significant amount of past work addressed this problem (e.g., [2, 3] [5, 1] [13] [17], [16]), there is still a lack of understanding of the general types of modification strategies and their coverage. For example, while it is generally agreed that the information about the internal structure of the plan is important to facilitate flexible plan modification, the general issue of relation between the types of remembered information and the modification/reuse methodologies they entail has not been adequately addressed. In the course of our research on developing PRIAR, a framework for flexible plan reuse in hierarchical planning [11, 12, 10], we have developed a classification of plan modification strategies based on the nature of the modifications that are allowed by them, and the types of information they require to facilitate those modifications. The motivation behind it was to classify and better understand the previous work on plan modification, and to characterize PRIAR's approach in that spectrum. We present this classification in the current paper, with the belief that it can help in better understanding the coverage and capabilities of various existing plan modification strategies, and in formulating the directions for further research. (Much of the discussion in this paper generalizes in part also to reuse and modification of other dependent structures such as designs and problem solutions.)

Let us start by describing two dimensions along which the information requirements of the plan modification strategies can be classified: the two dimensions of classification of the represented information: *functionality* and *ease of acquisition*. ‘‘Functionality’’ refers to the coverage, i.e., the different types of plan modification the represented information facilitates. ‘‘Ease of acquisition’’ is concerned about the methods by which such information can be acquired, and their complexity. As will become clear later on, there is often a trade-off involved between these two.

[*]The support of the Defense Advanced Research Projects Agency and the U.S. Army Engineer Topographic Laboratories under contract DACA76-88-C-0008, and that of Office of Naval Research under contract N00014-88-K-0620 are gratefully acknowledged.

The generally accepted answer to the question of representation of a stored plan is to ‘‘*remember everything that went into the formation of the plan*.’’ This was first suggested by Carbonell in his derivational analogy work [2]. For plan reuse, this has been interpreted to mean that the stored plan should represent a comprehensive ‘‘derivational trace’’ of the plan [13]. Here we show that the derivational trace typically combines two fundamentally different classes of information, which vary along the dimensions of functionality and ease of acquisition. We refer to these two types as the *correctness explanation* and the *decision rationale*. The first consists of information about the internal dependencies of the plan that effectively provide an explanation of the correctness (or the validity) of that plan. This itself can be sub-divided based on whether the explanation is with respect to the planner's domain model or with respect to some external domain model. The second type, decision rationale, consists of information about the rationale behind the individual planning choices. In the rest of the paper, we shall discuss the functionality and the ease of acquisition of these two broad categories of information, and list previous approaches to plan modification that can be categorized into these classes (Table 1 summarizes the exising work in this classification). Along the way, we shall address the importance of correctness explanations in guiding flexible plan modification, introduce the notion of ‘‘correctness with respect to domain models at various levels of abstraction,’’ and motivate its utility in guiding plan modification.

2. Modification based on Correctness Explanations

One type of information that has been found to be of significant use during plan reuse and modification is the explanation of the correctness or validity of the plan. Such information helps the planner compute the ramifications of local modifications on the overall correctness of the plan, and can also help it guide the modification so as to cause least amount disturbance to the original plan [12]. Various implemented systems (e.g., [5, 17, 16]) including PRIAR [11] utilize this type of information to guide plan modification. Here we shall argue that the information regarding the correctness of the plan is *necessary* for ensuring the flexibility of plan modification. We will show that such information can be further divided into two broad sub-categories based on whether the explanation is with respect to the planner's (problem-solver's) knowledge of the domain, or whether it is with respect to some external domain model. We shall see that this categorization allows us (*i*) to better compare the approaches of CHEF [5] and GORDIUS [16] to those of PRIAR [11] and SIPE [17], and (*ii*) to suggest ways in which their approaches can be combined. We will also discuss the utility of hybrid approaches that can exploit both types of correctness explanations.

Type of Information		Functionality Served	Ease of Acquisition
I.	**Explanation of Correctness**	*Retrieval, Location and Repair of Applicability Failures, Control of Modification.*	Can be acquired *on-line* during planning, or *off-line* after the planning through a domain model based causal simulation.
	(a). Correctness w.r.t. planner [*Validation Structure*]	*Ensures correctness relative to the planner's domain knowledge.* In other words, modified plans stay in the *deductive closure* of the planner's domain knowledge.	Can be annotated by the planner as a *by-product of* planning (e.g., PRIAR [11], SIPE [17], NONLIN [4])
	(b). Correctness w.r.t. external domain models [*Causal Explanation*]	*Ensures correctness relative to the external domain model.* In other words, modification here can take care of failures that the planner itself cannot detect or correct. Thus, modified plans may lie outside of the deductive closure of the planner's own domain knowledge.	Off-line causal simulation with the help of external domain models (e.g., CHEF [5], GTD [16]).
II.	**Rationale behind decisions** [*Justification for choosing a particular planning decision over the other possible alternatives.*]	*Can guide modification to avoid previous failures, achieve optimality etc.*	These structures have to be annotated by the planner or captured interactively (e.g., [3], [14]).

Table 1. Classification of Plan Modification Strategies based on Functionality and Ease of Acquisition of information represented on the stored plans

2.1. Correctness with respect to Planner

Explanation of correctness of the plan in terms of the planner's own knowledge of the domain can be provided (annotated) by the planner as a by-product of the initial plan generation. To the extent that such information helps in judging the correctness of a plan relative to the planner's own domain knowledge, it can also help in ensuring the correctness of the modification with respect to that knowledge. Since the result of modification lies in the deductive closure of planner's domain knowledge, the modification process itself can be integrated with the generative planning. (A point to bear in mind here is that the planner's own domain model may be incorrect and incomplete; more about that below.)

In our work with PRIAR, we formalized this type of explanation as the *validation structure* of the plan. The validation structure essentially encodes the information about the protection intervals of the plan locally on each task of the hierarchical task network underlying the development of the plan. Thus, it effectively provides an explanation of correctness of the plan at different levels of abstraction. Plan modification is seen as a process of removing the inconsistencies in the validation structure of the plan[1]. We showed that validation structure can be used not

[1] Such as repairing failing validations, establishing missing validations, and removing redundant validations; see [11] for details.

only for efficient localization of plan applicability failures (as was demonstrated by previous work), but also to control the refitting and retrieval phases [11, 12]. Other work that uses correctness explanations with respect to planner to guide plan modification includes [7, 4, 17]; for a detailed comparison of these systems with PRIAR, see [11]. We believe that in comparison to earlier approaches, PRIAR represents a more systematic exploration of the utility of annotated correctness explanations (validation structure) in guiding and controlling various phases of plan reuse and modification.

2.2. Correctness with respect to External Domain Models

The correctness explanation can also be provided with respect to an external domain model, which may be arbitrarily deep and may explicitly state assumptions that the planner's own model might have left implicit (to improve the efficiency of planning[2]). Such an explanation is typically acquired as a result of causal simulation, after the planning. Since the explanation of correctness is not limited by the the planner's domain model, it can also help in the detection and correction of bugs that arise due to incorrectness and incompleteness of the planner's own limited domain model. The trade-off is the increased complexity of plan modification (see below). Hammond's case-based planner CHEF [5], and Simmons' Generate-Test-Debug planner GORDIUS [16] are examples of systems that do plan modification using this type of correctness explanation.

An important source of complexity here is that modification cannot be integrated with planning, as the planner cannot carry out the debugging of plans that are not faulty with respect to its model of the domain. As a result, more complex domain independent debugging techniques would have to be employed to carry out the modification.[3]

2.3. Combining the Correctness Explanations: Hybrid Models

Though most previous plan modification/debugging approaches used either one or the other of the above two types of correctness explanations, the use of one does not necessarily preclude the use of the other. A way of exploiting both types of explanations would be the following: When the specification of the plan changes, first, PRIAR type validation structures can be used to efficiently produce a plan at the level of correctness of the planner through reuse. If during subsequent execution, that plan is found to fail due to incorrectness or incompleteness of the planner's domain model, a GTD style debugging of that plan can be attempted with the help of a correctness explanation relative to an external domain model. In a CHEF type framework[4], this allows us to first ensure the correctness of the modification with respect to planner, before embarking on a more costly debugging stage. This in turn lets the debugger concentrate on only those bugs that the planner itself cannot detect or repair, allowing for improvement of average case efficiency of plan modification.

[2] The rationale is that it may be more efficient on the average to synthesize a plan with the help of a simple if incomplete domain model and then to test it with respect to a deeper and complete external domain model, than to synthesize plans with a complete domain model; see [16] for further explanation.

[3] Simmons [16] points out that the overall success of GTD system is predicated on the generator's ability to produce plans that are typically correct, thereby reducing the need for debugging with the help of external domain models.

[4] For our purpose, CHEF's case-base of plans plus its plan modification strategies can be seen to constitute its plan generator.

2.3.1. Modification in the presence of Multiple Incomplete Domain Models

In many realistic domains, the planner is an embedded module, and has to of necessity interact with other modules that have more knowledge about certain aspects of the domain. Plan modification in such situation presents additional complexities. For example, in process planning for manufacturing, many important planning decisions arise from geometric considerations. If the domain model of the planner were to directly model such considerations, the resultant planner would be unacceptably inefficient. In general there may be several other considerations of this nature influencing a planner operating in a realistic domain (e.g., fixture considerations in process planning). One way of looking at them is to see those considerations as arising from specialized but incomplete models of the domain. For example, in process planning, the geometric considerations may be seen as stemming from the geometric simulation model of the domain. Note that these are not necessarily passive *testers*, as some level of understanding of these considerations is required to be able to carry out any useful planning at all. Plan modification in such situations provides special challenges as the correctness of the plan cannot be explained without addressing these other specialist modules. change.

There are two ways of dealing with the problem—the first, embraced by several existing systems (e.g., [8]), is to *anticipate* all the interactions and the corresponding linearizations with the help of some collection of expert rules, before the planning starts[5]. While this allows the planner to side-step the costly interaction detection and resolution processes, it leaves the planner with little understanding of the internal dependencies of the resultant plan (as it has no understanding of why certain planning decisions, such as linearizations, were made). Consequently, during reuse when the specification of the problem change (there by changing the anticipated interactions and linearizations), the planner cannot decide how much of its original plan will survive, and which parts need to be changed in the new situation. This inhibits flexible reuse of the plan.

A second way is to build into the planner an understanding of these interactions at some level of abstraction so that the planner, while incapable of discovering interactions by itself, can at least compute the ramifications of those interactions on its plan. In the event of specification change, such a capability would help it avoid redoing everything from scratch by flexibly reusing any applicable parts of the plan. For example, in process planning, the planner might use a predicate such as $Clear{-}Access(?feature)$ as a precondition of various actions to model tool accessibility of the feature. It may not be able to deduce the truth or falsity of the $Clear{-}Access$ predicate based solely on its domain model, but may have to rely on the geometric simulator to compute the truth of this predicate. Notice that this leads to two levels of explanation of correctness

[5] this has obvious similarities to the approach taken by CHEF [5].

of the plan—when the specification of the problem changes, the planner by itself cannot completely determine if its plan is correct—it will have to depend on the geometric simulator to compute the truth of the predicates such as *Clear−Access*. However, the planner can use the change in the truth of *Clear−Access* predicate to realize which exact parts of its plan will get affected and which will stay unaffected. The critical question concerns the feasibility of coming up with such consistent abstractions. Currently, we have started investigating this in the process planning domain.

2.4. Heuristic Modifications and Correctness

A common theme in the previous section was that the modification is best guided by an explanation of correctness of the plan with respect to some systematically abstracted model of the domain—either encoded in the planner, or outside of it. In contrast, most of the work in case-based plan modification, utilizes heuristic modification strategies. Such strategies are typically based on local transformations of the plan that are known to be harmless in the particular domain, or based on some other general knowledge of the domain (see [9] for a taxonomy of such modification strategies). To the extent that they represent externally acquired expertise (rather than cached generalizations from previous correctness-based modifications), it is difficult to say anything systematic about the correctness of the resultant modified plan. For example, a heuristic modification might introduce some new ‘bugs’ into the plan which may be impossible to detect in the absence of a causal dependency structure of the plan. Thus, either these strategies have to be supplemented by a simulation-debugging stage, or they should be restricted to domains where execution time failures are permissible.

Our position is that in general the heuristic modification strategies should be supplemented by explanations of correctness of the underlying plans, so that the ramifications of local modifications can be computed by the planner if needed. This is in contrast to Hammond's position [6][6]. While his argument is based on the apparent complexity of modification based on causal models, our position is motivated by a desire to demarcate the applicability of purely heuristic modification strategies. We believe that acquiring a set of robust and useful heuristic modification strategies which are guaranteed to give rise to successful plans is generally infeasible[7], and thus to avoid frequent execution time errors, or the costly simulation and debugging sessions, the modification process should also have access to the internal dependency structure of the plan. We are not arguing for inclusion of detailed causal models during plan modification; that will obviously be inefficient. As we pointed out in section 2.3.1, we believe that the right way of dealing with ‘‘complex causal models’’ during plan modification is to abstract the causal model in such a way that the planner/modifier has an understanding of the correctness (internal effects) of its plan at *some level of abstraction*.

[6] In [6], Hammond argues ‘‘ · · · New plans are built from old plans. But the internal effects of these plans need not be known and need not be analyzed by the planner.’’

[7] To achieve robustness, such strategies have to either carefully restrict allowable modifications to the plan specification (before they resort to from-scratch planning), or rely on retrieval strategies that are able to provide a very closely applicable plan for any given new problem. What we are addressing here is a *flexible* modification strategy that does not impose such limitations [11].

3. Exploiting Rationale Behind Planning Choices

We have discussed several types of dependency structures that represent explanation of the correctness of a stored plan. One of the limitations of the correctness explanations is that while they point out why a planning decision (such as a schema instance selection) is valid, they do not explicate the rationale behind that decision choice. In other words, they do not give any information regarding the space of possible decision choices at the time that decision was made, or regarding the reasons for choosing that particular decision from among the competing ones. The correctness explanations also do not address the global optimality considerations underlying the plan. Often a particular planning decision may have been made based on optimality considerations, and when an existing plan is being reused in a new situation, it may be correct, but no longer optimal in that situation. The decision rationale information of the type outlined above, when properly represented, can be useful in guiding the modification so as to avoid previous failures, or to take optimality considerations into account during plan modification.

To make this distinction clear, let us consider the example of modifying a process plan for making two holes h_1 and h_2 on a metal stock. Suppose that in the original plan, both the holes are being made with spade drilling operation even though h_2 could have been made with a less expensive twist drilling operation. Such a decision may have been made to ensure global optimality of the plan by avoiding a tool change. When this plan is being used in a slightly different situation, the planner may find that h'_1, the hole corresponding to h_1 in the new situation, cannot be made with spade drilling. This should not only change the process plan for hole h'_1, but should also change the process plan for the hole h'_2, so that it will now be made with twist drilling instead of the more expensive spade drilling operation. (Even this might not necessarily ensure the global optimality of the resultant plan; see below.)

Neither of the two types of correctness explanations discussed previously capture such optimality considerations underlying the plan. (From the point of view of correctness, in the above example, the decision to make h_2 with the help of twist drilling and the decision to make it with spade drilling are both equally viable.) Such decision rationale information has to be either annotated by the planner, or be interactively captured from a teacher. The problem with the first alternative is that there are very few planners which do any kind of intelligent deliberation on the decision choice. Further, while it is possible to remember previous failures, often (particularly in nonlinear planning) pinpointing the exact planning decision which lead to a particular type of failure (and subsequent backtracking) might require a costly analysis. The second alternative of interactive capture of decision rationale is an even harder problem, as the captured rationale has to be in a form that can be understood and utilized by the planner (see [15]). It is thus not surprising that there have not been any implemented systems which systematically represent decision rationale information, or utilize it to automate plan reuse and modification.

The original derivational analogy framework [2] proposed that this type of information be captured. However, since then there has not been any systematic characterization of the nature and utility of such structures. Carbonell and Veloso's recent proposal for integrating derivational analogy into a problem-solving framework [3] takes some preliminary steps towards enumerating the types of decision rationale information that can be captured automatically. However, it does not specify the details of the possible uses that are envisaged for such information. PRIAR currently does not capture or utilize this type of decision rationale information. Making it do so is an important research direction. An avenue for potential research is to extend the PRIAR validation structure such that it will also include information about ‘‘optimality validations,’’ i.e., validations whose failure will not necessarily undermine the executability of the plan, but would necessitate a re-examination of the plan optimality. Upon failure of such validations, the modification strategies would have to look at possible ways of regaining the local and global optimality of the plan. Unlike correctness, optimality is a predominantly global consideration. To formalize such notions of incremental restoration of optimality, more systematic notions of quasi-optimality (such as reusing as much of the current plan as possible by finding the modifications that can be merged with the actions of the existing plan) have to be formulated.

4. Summary

In this paper, based on our experience in designing PRIAR plan reuse and modification framework, we attempted a categorization of classes of plan modification strategies based on their coverage and information requirements. We refined the notion of ‘‘derivational trace’’ of a plan in guiding plan modification, and motivated the utility of correctness explanations with respect to domain models at various levels of abstraction. This allowed us to classify various previous approaches to plan modification, to compare their coverage, and to suggest directions for further research.

References

1. R. Alterman, ‘‘An Adaptive Planner’’, *Proceedings of 5th AAAI*, 1986, 65-69.
2. J. G. Carbonell, ‘‘Derivational Analogy and its Role in Problem Solving’’, *Proceedings of AAAI*, Washington D.C., 1983, 64-69.
3. J. Carbonell and M. Veloso, ‘‘Integrating Derivational Analogy into a General Problem Solving Architecture’’, *Proceedings of Case-Based Reasoning Workshop*, 1988, 104-121.
4. L. Daniel, ‘‘Planning: Modifying non-linear plans’’, DAI Working paper 24, University of Edinburgh, December 1977. (Also appears as ‘‘Planning and Operations Research,’’ in *Artificial Intelligence: Tools, Techniques and Applications*, Harper and Row, New York, 1983).
5. K. J. Hammond, ‘‘CHEF: A Model of Case-Based Planning’’, *Proceedings of 5th AAAI*, 1986, 267-271.
6. K. Hammond, ‘‘Case-Based Planning: Viewing Planning as a Memory Task’’, *Proceedings of Darpa workshop on Case-Based Reasoning*, 1988, 17-20.
7. P. J. Hayes, ‘‘A Representation for Robot Plans’’, *Proceedings of 4th IJCAI*, 1975.
8. C. Hayes, ‘‘Using Goal Interactions to Guide Planning’’, *Proceedings of Sixth AAAI*, 1987, 224-228.
9. T. R. Hinrichs, ‘‘Strategies for Adaptation and Recovery in a Design Problem Solver’’, *Proceedings of 2nd DARPA workshop on Case-Based Reasoning*, 1989, 115-118.
10. S. Kambhampati and J. A. Hendler, ‘‘Flexible Reuse of Plans via Annotation and Verification’’, *Proceedings of 5th IEEE Conf. on Applications of Artificial Intelligence*, 1989, 37-44.
11. S. Kambhampati, ‘‘Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach’’, CS-Tech. Rep.-2334, CAR-Tech. Rep.-469, Center for Automation Research, Department of Computer Science, University of Maryland, College Park, MD 20742, October 1989. (Ph.D. Dissertation).
12. S. Kambhampati and J. A. Hendler, ‘‘Control of Refitting during Plan Reuse’’, *11th International Joint Conference on Artificial Intelligence*, Detroit, Michigan, USA, August 1989, 943-948.
13. J. L. Kolodner, ‘‘Case-Based Problem Solving’’, *Proceedings of the Fourth International Workshop on Machine Learning*, University of California, Irvine, June 1987, 167-178.
14. J. Mostow and M. Barley, ‘‘Automated Reuse of Design Plans’’, *Proceedings of International Conference on Engineering Design*, 1987.
15. J. Mostow, ‘‘Design by Derivational Analogy: Issues in the Automated Replay of Design Plans’’, Rutgers University ML-Tech. Rep.-22, March 1987. (To appear in Artificial Intelligence Journal).
16. R. Simmons and R. Davis, ‘‘Generate, Test and Debug: Combining Associational Rules and Causal Models’’, *Proceedings of 10th IJCAI 10* (1987), 1071-1078.
17. D. E. Wilkins, ‘‘Recovering from execution errors in SIPE’’, *Computational Intelligence 1* (1985).