A Theory of Plan Modification

Subbarao Kambhampati[*]

Center for Design Research and Department of Computer Science
Stanford University
Bldg. 530, Duena Street, Stanford CA 94305-4026
e-mail: *rao@sunrise.stanford.edu*

Abstract

We present a theory of plan modification applicable to hierarchical nonlinear planning. Our theory utilizes the validation structure of the stored plans to yield a flexible and conservative plan modification framework. The validation structure, which constitutes a hierarchical explanation of correctness of the plan with respect to the planner's own knowledge of the domain, is annotated on the plan as a by-product of initial planning. Plan modification is characterized as a process of removing inconsistencies in the validation structure of a plan when it is being reused in a new (changed) planning situation. The repair of these inconsistencies involves removing unnecessary parts of the plan and adding new non-primitive tasks to the plan to establish missing or failing validations. The resultant partially reduced plan (with a consistent validation structure) is sent to the planner for complete reduction. We discuss the development of this theory in PRIAR system, and characterize its completeness, coverage, efficiency and limitations.

1. Introduction

The ability to flexibly and conservatively modify existing plans to make them conform to the constraints of a new of changed planning situation is very useful in plan reuse, replanning and incremental planning. While the value of such capability has been acknowledged early in planning research [5, 7], the strategies developed were inflexible in that they could reuse or modify a given plan in only a limited number of situations, and could deal with only a limited variety of applicability failures. There was no general framework for conservatively modifying an existing plan to fit it to the constraints of a new problem situation. A major shortcoming with these approaches was that the stored plans did not represent enough information about the internal dependencies of the plan to permit flexible modification. For example, reuse based on macro-operators [5] built from sequences of primitive plan steps was unable to modify intermediate steps of the macro-operator, as macro-operators did not represent the intermediate decisions and dependencies corresponding to their internal steps. Even in cases where the need for the dependency information was recognized (e.g. [4, 22]), a systematic representation and utilization of such structures in plan reuse and modification was not attempted.

We present a theory of plan modification that allows flexible and conservative modification of plans generated by a hierarchical nonlinear planner. Hierarchical planning is a prominent method of abstraction and least-commitment in domain-independent planning [3]. Our theory of plan modification proposes *validation structure* as a way of representing the internal dependencies of a hierarchical plan and provides algorithms for annotating the validation structure on the plans during plan generation. It systematically explores the utility of the annotated validation structure in guiding and controlling all the processes involved in flexible plan reuse and modification. The PRIAR plan modification system [9, 11, 10, 12] is our implementation of this theory.

The plan modification problem that is addressed in PRIAR is the following: *Given* (*i*) a planning problem P^n (specified by a partial description of the initial state I^n and goal state G^n), (*ii*) an existing plan R^o (generated by a hierarchical nonlinear planner), and the corresponding planning problem P^o, *Produce* a plan for P^n by minimally modifying R^o.

In the PRIAR reuse framework, the internal dependencies of a hierarchical plan which are relevant to guide its reuse and modification are formalized as the *validation structure* of the plan. The validation structure can be seen as a form of hierarchical explanation of correctness for the plan with respect to the planner. Individual tasks of the hierarchical plan are annotated with information about their role in the plan validation structure. PRIAR provides efficient algorithms for acquiring these annotations as a by-product of planning.

When an existing plan is being reused in a new planning situation, the applicability failures, the redundancies, and the shortcomings that may arise in the process are formally characterized as *inconsistencies* in the plan's validation structure. Reuse in the PRIAR framework is formally seen as a process of repairing the inconsistencies in the validation structure of a given plan when it is mapped into the new problem situation. Given the new problem P^n, and an *annotated* plan R^o, PRIAR's reuse process proceeds in the following steps:

(1) Mapping and Interpretation: An appropriate mapping α between the objects of $[P^o, R^o]$ and P^n is computed, and R^o is mapped into P^n with it. Next, some important differences between P^o and P^n are marked. The resulting interpreted plan, R^i, is typically a plan with an inconsistent validation structure.

(2) Annotation Verification: The inconsistencies in the validation structure of R^i are located, and appropriate repairs are

---

[*] 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.

suggested. The repairs include removing parts of R^i that are unnecessary and adding non-primitive tasks (called *refit tasks*) to establish any required new validations. The resulting annotation-verified plan R^a will have a consistent validation structure but is typically only partially reduced. It consists of all the applicable parts of R^i and any refit tasks which are introduced.

(3) Refitting: The refit tasks specified during the annotation verification phase constitute sub-planning problems for the hierarchical planner. The refitting process involves reducing them with the help of the planner. Conservatism is ensured through the use of a heuristic control strategy which minimizes the disturbance to the applicable parts of R^a during this process.

Computational savings stem from the fact that the cost of solving the sub-planning problems during refitting is on the average much less than the cost of solving the entire planning problem from scratch. This is supported by the results of the empirical studies in blocks world, which showed that plan modification proveds 20-98% savings (corresponding to speedup factors of 1.5 to 50) over pure generative planning.

This paper concentrates mainly on the development of the basic modification framework; the details of mapping and refitting control strategies can be found in [9, 12, 10]. The organization of this paper is as follows: Rest of this section provides some necessary preliminaries of hierarchical planning. Section 2 presents the notion of plan validation structure, explains the motivation behind remembering it along with each generated plan, and presents a scheme for annotating it on the plan. Section 3 develops the basic modification processes, and explains how they utilize the plan validation structure. Section 4 contains a discussion completeness, coverage and efficiency of PRIAR modification theory and section 5 provides a brief discussion of related work.

1.1. Preliminaries

This paper makes extensive use of the concepts of hierarchical planning paradigm. A good introduction to this methodology can be found in [3]. Some well known hierarchical planners include NOAH [17], NONLIN [20] and SIPE [21]. In hierarchical planning, a partial plan is represented as a *task network.* A task network is a 3-tuple $\langle T, O, \Pi \rangle$, where T is a collection of tasks, O defines a partial ordering over T, and Π is a set of *protection intervals*. A protection interval is a 3-tuple $\langle E, t_1, t_2 \rangle$, where $t_1, t_2 \in T$, E is an effect of t_1, and E has to necessarily persist up to t_2 [3]. Planning proceeds by selecting a task from the current task network and reducing it with the help of a task reduction schema to more concrete subtasks. This reduction may introduce some harmful interactions with the existing protection intervals, which are handled by introducing additional partial ordering relations among the tasks.

The planner cannot reduce certain distinguished tasks of the domain called *primitive tasks.* (It is assumed that the planner "knows" how to execute such tasks.) Further, if all the required effects of a task are already true in a given partial plan, then that task does not have to be reduced any further (such tasks are called *phantom goals* [3]). A task network is said to represent a *completed plan* when none of its tasks have to be reduced further.

The hierarchical development of a plan $P: \langle T, O, \Pi \rangle$ is captured by its *hierarchical task network* (abbreviated as HTN) . A HTN is a 3-tuple, $\langle P: \langle T, O, \Pi \rangle, T^*, D \rangle$, where T^* is a superset of T, and D defines a set of parent-child relations among the tasks of T^*. (The immediate children of a task t are the tasks that resulted from its reduction during planning.) For convenience, we will be referring to T^*, the tasks of the HTN, also as its nodes. We shall refer to the number of leaf nodes in a HTN ($|T|$) as the length of the corresponding plan, and denote it by N_P. For the sake of uniformity, we shall assume that the HTN has two special primitive nodes $n_I, n_G \in T^*$, corresponding respectively to the input state and the goal state of the planning problem. We shall use the notation "$n_1 < n_2$" (where n_1 and n_2 are nodes of HTN) to indicate that n_1 is ordered to *precede* n_2 in the partially orderered plan represented by the HTN. Similarly, "$n_1 > n_2$" denotes that n_1 is ordered to *follow* n_2, and "$n_1 \,//\, n_2$" denotes that there is no ordering relation between the two nodes (n_1 is parallel to n_2). The set consisting of a node n and all its descendents in the HTN is called the sub-reduction of n, and is denoted by $R(n)$. Following [3, 20], we also distinguish two types of plan applicability conditions: the preconditions (such as $Clear(A)$ in the blocks world) which the planner can achieve, and the filter conditions (such as $Block(A)$ in the blocks world) which the planner cannot achieve. Finally, we shall use the notation "$F \vdash f$" to indicate that f deductively follows from the set of facts in F.

2. Validation Structure and Annotations

2.1. Validation Structure

§2.1. Validation: A *validation* is a 4-tuple $\langle E, n_s, C, n_d \rangle$, where n_s and n_d are leaf nodes belonging to the HTN, and the effect E of node n_s (called the *source*) is used to satisfy the applicability condition C of node n_d (called the *destination*). C and E are referred to as the *supported condition* and the *supporting effect* respectively of the validation. As a necessary condition for the existence of a validation v, the partial ordering among the tasks in HTN must satisfy the relation $n_s < n_d$. The *type* of a validation is defined as the type of the applicability condition that the validation supports (one of *filter condition, precondition, phantom goal*). Notice, from section 1.1 that every validation $v: \langle E, n_s, C, n_d \rangle$ corresponds to a protection interval [3] $\langle E, n_s, n_d \rangle$. This correspondence implies that there will only be a finite set of validations corresponding to a given HTN representing the development of a plan; we shall call this set V. (If ξ is the maximum number of applicability conditions for any action in the domain, then $|\mathbf{V}|$ is $O(\xi N_P)$ [10].)

Figure 1 shows the validation structure of the plan for solving a block stacking problem 3BS (also shown in the figure). Validations are represented graphically as links between the effect of the source node and the condition of the destination node. (For the sake of exposition, validations supporting conditions of the type $Block(?x)$ have not been shown in the figure.) For example, $\langle On(B,C), n_{15}, On(B,C), n_G \rangle$ is a validation belonging to this plan since $On(B,C)$ is required at the goal state n_G, and is provided by the effect $On(B,C)$ of node n_{15}.

§2.2. Inconsistencies and Consistency of Validation Structure: A validation $v: \langle E, n_s, C, n_d \rangle$ is considered a *failing validation* if either $E \notin \mathit{effects}(n_s)$ or when there exists a node $n \in$ HTN such that n possibly falls between n_s and n_d. A validation $v: \langle E, n_s, C, n_d \rangle$ is considered an *unnecessary validation* iff the node n_d does not require the condition C. (This could happen, for example, if a goal of the plan is no longer necessary in the current problem situation.) Finally, we say that there is a

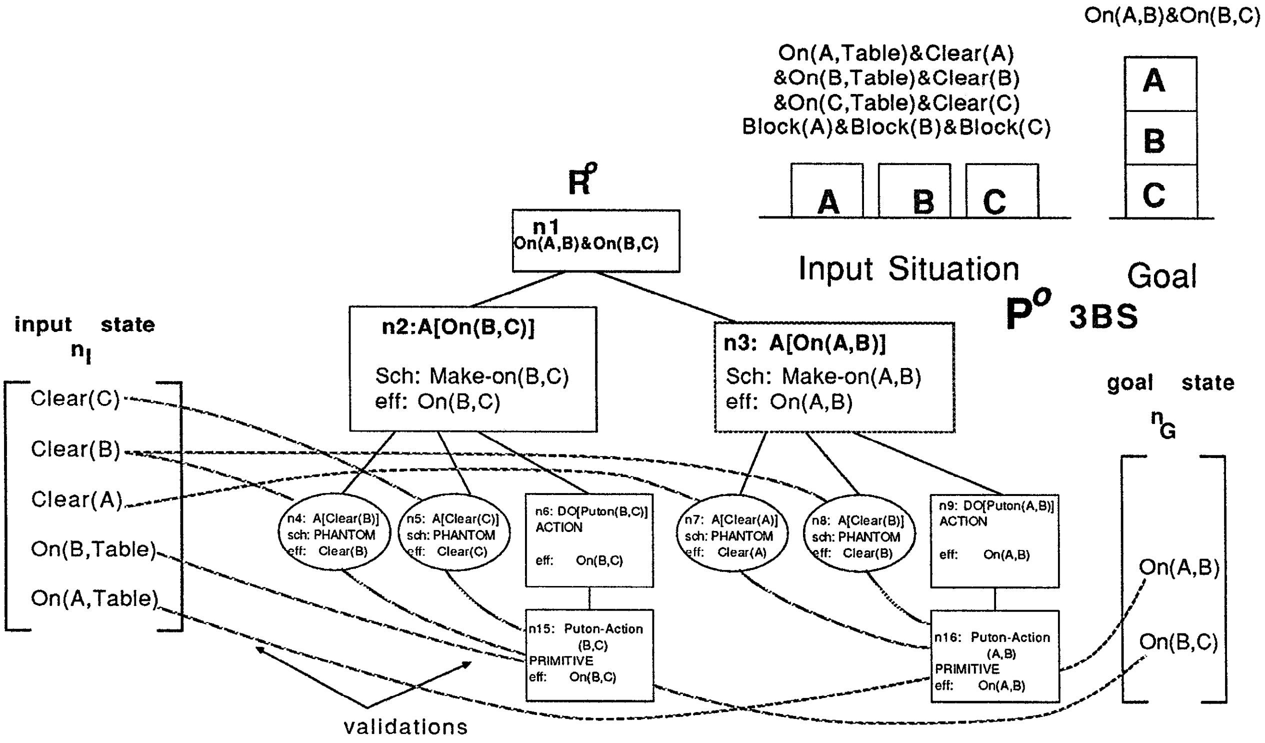

Figure 1. Validation structure of 3BS plan

missing validation corresponding to a condition, node pair $\langle C', n' \rangle$ of the HTN iff $\nexists v: \langle E, n_s, C, n_d \rangle$ *s.t.* $C = C' \wedge n_d = n'$.

The unnecessary, missing or failing validations in a HTN will be referred to as *inconsistencies* in its validation structure. An HTN is said to have a *consistent validation structure* if it does not have any inconsistencies. From these definitions, it should be clear that in a HTN with a consistent validation structure, each applicability condition of a node (including each goal of n_G) will have a non-failing validation supporting it. (Thus, a completely reduced HTN with a consistent validation structure constitutes a valid executable plan.)

2.2. Annotating Validation Structure

Having developed the notion of validation in a plan, our next concern is representing the validation structure of the plan locally as annotations on individual nodes of a HTN. The intent is to let these annotations encapsulate the role played by the sub-reduction below that node in the validation structure of the overall plan, so that they can help in efficiently gauging the effect of any modification at that node on the overall validation structure of the plan. We achieve this as follows: For each node $n \in$ HTN we define the notions of (i) *e-conditions*(n), which are the externally useful validations supplied by the nodes belonging to $R(n)$ (the sub-reduction below n) (ii) *e-preconditions*(n), which are the externally established validations that are consumed by nodes of $R(n)$, and (iii) *p-conditions*(n), which are the external validations of the plan that are required to persist over the nodes of $R(n)$.

§2.3. E-Conditions (External Effect Conditions): The e-conditions of a node n correspond to the validations supported by the effects of any node of $R(n)$ which are used to satisfy applicability conditions of the nodes that lie outside the sub-reduction. Thus, *E-conditions*(n) =

$$\{v_i: \langle E, n_s, C, n_d \rangle \,|\, v_i \in \mathrm{V};\ n_s \in R(n);\ n_d \notin R(n)\ \}$$

For example, the e-conditions of the node n_3 in the HTN of figure 1 contains just the validation $\langle On(A,B),\ n_{16},\ On(A,B),\ n_G \rangle$ since that is the only effect of $R(n_3)$ which is used outside of $R(n_3)$. The e-conditions provide a way of stating the externally useful effects of a sub-reduction. They can be used to decide when a sub-reduction is no longer necessary, or how a change in its effects will affect the validation structure of the parts of the plan outside the sub-reduction.

§2.4. E-Preconditions (External Preconditions): The e-preconditions of node n correspond to the validations supporting the applicability conditions of any node of $R(n)$ that are satisfied by the effects of the nodes that lie outside of $R(n)$. Thus, *E-preconditions*(n) =

$$\{v_i: \langle E, n_s, C, n_d \rangle \,|\, v_i \in \mathrm{V};\ n_d \in R(n);\ n_s \notin R(n)\ \}$$

For example, the e-preconditions of the node n_3 in the HTN of figure 1 will include the validations $\langle Clear(A),\ n_I,\ Clear(A),\ n_7 \rangle$ and $\langle Clear(B),\ n_I,\ Clear(B),\ n_8 \rangle$. The e-preconditions can be used to locate the parts of rest of the plan that will become unnecessary or redundant, if the sub-reduction below this node is changed.

§2.5. P-Conditions (Persistence Conditions): P-conditions of a node n correspond to the protection intervals of the HTN that are external to $R(n)$, and have to persist over some part of $R(n)$ for the rest of the plan to have a consistent validation structure. We define them in the following way:

A validation $v_i: \langle E, n_s, C, n_d \rangle \in \mathrm{V}$ is said to *intersect* the sub-reduction $R(n)$ below a node n (denoted by "$v \otimes R(n)$") if there exists a leaf node $n \in R(n)$ such that n possibly falls between n_s and n_d (for some total ordering of the tasks in the HTN). Using the definition of validation [10], we have $v_i : \langle E, n_s, C, n_d \rangle \otimes R(n)$ iff

$$\begin{bmatrix} \exists n' \in R(n) \text{ s.t. } children(n') = \emptyset \wedge \\ (n_s < n' < n_d \vee n_s \,//\, n' \vee n_d \,//\, n') \end{bmatrix}$$

A validation $v_i : \langle E, n_s, C, n_d \rangle \in \mathrm{V}$ is considered a *p*-condition of a node n iff v_i *intersects* $R(n)$ and neither the source nor the destination of the validation belong to $R(n)$. Thus, *P-conditions*(n) =

$$\{ v_i : \langle E, n_s, C, n_d \rangle \mid v_i \in \mathrm{V};\ n_s, n_d \notin R(n);\ v_i \otimes R(n) \}$$

From this definition, it follows that if the effects of any node of the $R(n)$ violate the validations corresponding to the *p*-conditions of n, then there will be a potential for harmful interactions. As an example, the *p*-conditions of the node n_3 in the HTN of figure 1 will contain the validation $\langle On(B,C), n_{15}, On(B,C), n_G \rangle$ since the condition $On(B,C)$, which is achieved at n_{15} would have to persist over $R(n_3)$ to support the condition (goal) $On(B,C)$ at n_G. The *p*-conditions help in gauging the effect of changes made at the sub-reduction below a node on the validations external to that sub-reduction. This is of particular importance in localizing the refitting [9].

2.3. Computing Annotations

In the PRIAR framework, at the end of a planning session, the HTN showing the development of the plan is retained, and each node of the HTN is annotated with the following information: (1) *Schema*(n), the schema instance that reduced node n (2) *e-preconditions*(n) (3) *e-conditions*(n), and (4) *p-conditions*(n). The node annotations are computed in two phases: First, the annotations for the leaf nodes of the HTN are computed with the help of the set of validations, V, and the partial ordering relations of HTN. Next, using the relations between the annotations of a node and its children (which can be easily derived from the definitions of the previous section; see [10]), the annotations are propagated to non-leaf nodes in a bottom up breadth-first fashion. The exact algorithms are given in [10], and are fairly straightforward to understand given the development of the previous sections. The time complexity of annotation computation is $O(N_P^2)$, where N_P is the length of the plan (number of leaf nodes in the HTN).

While the procedures discussed above compute the annotations of a HTN in one-shot, often during plan modification, PRIAR needs to add and remove validations from the HTN one at a time. To handle this, PRIAR also provides algorithms to update node annotations consistently when incrementally adding or deleting validations from the HTN. These are used to re-annotate the HTN and to maintain a consistent validation structure after small changes are made to the plan. They can also be called by the planner any time it establishes or removes a new validation (or protection interval) during the development of the plan, to dynamically maintain a consistent validation structure. The time complexity of these algorithms is $O(N_P)$ [10].

3. Modification by Annotation Verification

We will now turn to the plan modification process, and demonstrate the utility of annotated validation structure in guiding plan modification. Throughout the ensuing discussion, we will be following the simple example case of modifying the plan for the three block stacking problem 3BS (*i.e.*, R^o = 3BS) shown on the left side in figure 2 to produce a plan for the four block stacking problem 4BS1 (*i.e.*, P^n = 4BS1) shown on the right side. We shall refer to this as the 3BS→4BS1 example.

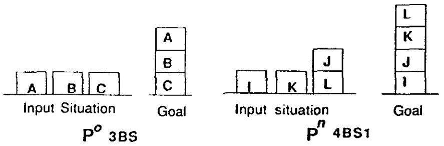

Figure 2. 3BS→4BS1 Reuse problem

3.1. Mapping and Interpretation

In PRIAR, the set of possible mappings between $[P^o, R^o]$ and P^n are found through a partial unification of the goals of the two problems. There are typically several semantically consistent mappings between the two planning situations, and selecting the right mapping can considerably reduce the cost of modification. The mapping and retrieval methodology used by PRIAR [10, 12] achieves this by selecting mappings based on the number and type of inconsistencies that would be caused in the validation structure of R^o. As the details of this strategy are beyond the scope of this paper, for the purposes of this paper, we shall simply assume that such a mapping is provided to us. (It should be noted that this mapping stage will not be required if the objective is to modify an existing plan in response to changes in its own specifications.) Once a mapping α is selected, the interpreted plan R^i is constructed by mapping R^o along with its annotations into the new planning situation P^n, and marking the differences between the specifications of the old and new planning situations. These differences, marked in I^i and G^i, serve to focus the annotation verification procedure on the inconsistencies in the validation structure of the interpreted plan.

In the 3BS→4BS1 example, let us assume that the mapping strategy selects $\alpha = [A \to L, B \to K, C \to J]$ as the mapping from 3BS and 4BS1. With this mapping, $Clear(L)$ is no longer true in the input specification of 4BS1. So it will be marked *out* in I^i. The facts $On(J,L)$, $On(I,Table)$ and $Clear(I)$ are true in 4BS1 but not in 3BS, so they will be marked as *new* facts in I^i. Similarly, as $On(J,I)$ is not a goal of 3BS but is a goal of 4BS1, it will be marked as an *extra goal* in G^i. There are no *unnecessary* goals. At the end of this processing, R^i, I^i and G^i are sent to the annotation verification procedure.

3.2. Annotation Verification and Refit Task Specification

At the end of the interpretation procedure, R^i may not have a consistent validation structure (see §2.2) as the differences between the old and the new problem situations (as marked by the interpretation procedure) may be causing some inconsistencies in the validation structure of R^i. These inconsistencies will be referred to as *applicability failures*, as these are the reasons why R^i cannot be directly applied to P^n. The purpose of the annotation verification procedure is to modify R^i such that the result, R^a, will be a partially reduced HTN with a consistent validation structure.

The annotation verification procedure achieves this goal by first localizing and characterizing the applicability failures caused by the differences in I^i and G^i, and then appropriately modifying the validation structure of R^i to repair those failures.

It groups the applicability failures into one of several classes depending on the type of the inconsistencies and the type of the conditions involved in those inconsistencies. The repairs are suggested based on this classification, and involve removal of unnecessary parts of the HTN and/or addition of non-primitive tasks (called refit tasks) to establish missing and failing validations. The individual repair actions taken to repair the different types of inconsistencies are briefly described below; they make judicious use of the node annotations to modify R^i appropriately (see [10, 12] for the detailed procedures). In [10], we show that the time complexity of the annotation-verification process is polynomial ($O(|V|N_P^3)$) in the length of the plan.

[1] Unnecessary Validations—Pruning Unrequired Parts: If the supported condition C of a validation $v:\langle E, n_s, C, n_d\rangle$ is no longer required, then v can be removed from the plan along with all the parts of the plan whose sole purpose is supplying those validations. The removal can be accomplished in a clean fashion with the help of the annotations on R^i. After removing v validation from the HTN (which will also involve incrementally re-annotating the HTN, see section 2.3), the HTN is checked for any node n_v that has no e-conditions. If such a node exists, then its sub-reduction, $R(n_v)$ has no useful purpose, and thus its nodes can be removed from the HTN. This essentially involves backtracking over the task reductions in that sub-reduction, and removing any ordering relations that were introduced as a result of those reductions. This removal turns the e-preconditions of n_v into unnecessary validations, and they are handled in the same way recursively.

[2] Missing Validations—Adding Tasks for Achieving Extra Goals: An extra goal is any goal of the new problem that is not a goal of the old plan, and thus is un-supported by any validation in R^i. The general procedure for repairing missing validations (including the extra goals, which are considered conditions of n_G) is to create a refit task of the form $Achieve[G]$, and to add it to the HTN in such a way that it follows the initial node n_I, and precedes the node which requires the unsupported condition (in this case n_G). Establishing a new validation in this way necessitates checking to see if its introduction leads to any new failing validations in the plan; the planner's interaction detection routines are used for this purpose. Finally, the annotations of the nodes of the HTN are updated (with the help of incremental annotation procedures) to reflect the introduction of the new validation.

[3] Failing Validations: The facts of I^i which are marked "*out*" during the interpretation process, may be supplying validations to the applicability conditions or goals of the interpreted plan R^i. The treatment of such failing validations depends upon the types of the conditions that are being supported by the validation. We distinguish three types of validation failures—validations supporting preconditions, phantom goals and filter conditions respectively—and discuss each of them in turn below[1].

(3.i) Failing Precondition Validations: If a validation supporting a precondition of some node in the HTN is found to be failing, because its supporting effect E is marked *out*, it can simply be reachieved. The procedure involves creating a refit task, $n_v{:}Achieve[E]$, to re-establish the validation v, and adding it to the HTN in such a way that it follows the source node and precedes the destination node of the failing validation. The validation structure of the plan is updated so that the failing validation will be replaced by an equivalent validation to be supplied by n_v. Finally, the annotations on the other nodes of the HTN are adjusted incrementally to reflect this change.

(3.ii) Failing Phantom Validations: If the validation supporting a phantom goal node is failing, then the node cannot remain phantom. The repair involves undoing the phantomization, so that the planner would know that it has to re-achieve that goal. Once this change is made, the failing validation is no longer required and can be removed.

(3.iii) Failing Filter Condition Validations: In contrast to the validations supporting the preconditions and the phantom goals, the validations supporting failing filter conditions cannot be reachieved by the planner. Instead, the planning decisions which introduced those filter conditions into the plan have to be undone. That is, if the validation $v:\langle E, n_s, C, n_d\rangle$ supporting a filter condition C of a node n_d is failing, and n' is the ancestor of n, whose reduction introduced C into the HTN originally, then the sub-reduction $R(n')$ has to be replaced, and n' has to be re-reduced with the help of an alternate schema instance. So as to least affect the validation structure of the rest of the HTN, any new reduction of n' is be *expected* to supply (or consume) the validations previously supplied (or consumed) by the replaced reduction. Any validations not supplied by the new reduction would have to be re-established by alternate means, and the validations not consumed by the new reduction would have to be pruned. Since there is no way of knowing what the new reduction will be until refitting time, this processing is deferred until that time.

[4] P-Phantom-Validations—Exploiting Serendipitous Effects: It is possible that some of the validations that R^i establishes via step addition can be established directly from the interpreted initial state, thus shortening the plan. Such validations, called *p-phantom validations*, are located by collecting validations whose source node is *not* n_I, and checking to see if their supporting effects are now true in the *new* facts of I^i. For each p-phantom validation, PRIAR checks to see if an equivalent validation can actually be established from the initial state, n_I without introducing new interactions (and thereby causing substantial revisions) in the plan. If so, the p-phantom validation becomes redundant, and is treated as an unnecessary validation. The parts of the plan that are currently establishing this validation are pruned from the HTN, thus effectively shortening the plan.

Example: Figure 3 shows R^a, the HTN produced by the annotation verification procedure for the 3BS→4BS1 example. The input to the annotation verification procedure is the interpreted plan R^i discussed in section 3.1. In this example, R^i contains a failing phantom validation and a missing validation corresponding to an extra goal. The goal $On(J,I)$ of G^i is an extra goal, and is not supported by any validation of the HTN. So, the refit task $n_{10}{:}Achieve[On(J,I)]$ is added to the task network, in parallel to the existing plan, such that $n_I < n_{10} < n_G$. n_{10} now supplies the validation $\langle On(J,I), n_{10}, On(J,I), n_I\rangle$ to the goal $On(J,I)$. Next, the fact $Clear(L)$, which is marked *out* in I^i, causes the validation $\langle Clear(L), n_I, Clear(L), n_7\rangle$ supporting the phantom goal node n_7 to fail. So, the phantom goal node n_7 is converted into a refit task to be reduced. It no longer needs the failing phantom validation from n_I. Notice that the HTN shown in this figure corresponds to a partially reduced task network which consists of the applicable parts of the old plan and the two refit tasks suggested by the annotation verification

---

[1] In NONLIN terminology [19] the precondition validations support the "unsupervised conditions" of a schema, while the phantom goal validations support the "supervised conditions" of a schema.

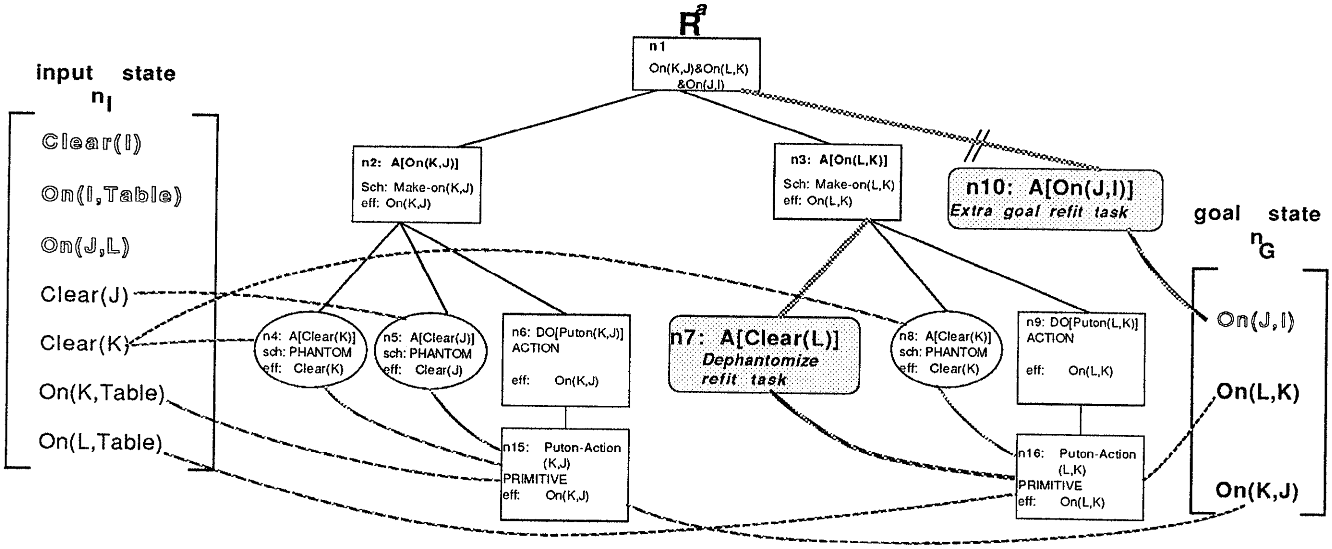

Figure 3. Annotation-Verified plan for 3BS→4BS1

procedure. It has a consistent validation structure, but it contains two unreduced refit tasks n_{10} and n_7 which have to be reduced.

3.3. Refitting

To produce an executable plan for P^n, R^a (the HTN after the annotation verification process) has to be completely reduced. This process, called refitting, essentially involves reduction of the refit tasks that were introduced into R^a during the annotation verification process. The responsibility of reducing the refit tasks is delegated to the planner by sending R^a to the planner. An important difference between refitting and from-scratch (or generative) planning is that in refitting the planner starts with an already partially reduced HTN. For this reason, solving P^n by reducing R^a is less expensive on the average than solving P^n from scratch.

The procedure used for reducing refit tasks is fairly similar to the one the planner normally uses for reducing non-primitive tasks (see section 1.1), with the following important difference. An important consideration in refitting is to minimize the disturbance to the applicable parts of R^a during the reduction of the refit tasks. To ensure this *conservatism* of refitting, the default schema selection procedure is modified in such a way that for each refit task, it selects a schema instance that is expected to give rise to the least amount of disturbance to the validation structure of R^a. The annotated validation structure of the plan helps in this selection by estimating the effect of reduction at a refit task on the rest of the plan. A detailed presentation of this heuristic control strategy is beyond the scope of this paper; the interested reader is referred to [9, 10]. Once the planer selects an appropriate schema instance in this way, it reduces the refit task by that schema instance in the normal way, detecting and resolving any interactions arising in the process.

Example: Figure 4 shows the hierarchical task reduction structure of the plan for the 4BS1 problem that PRIAR produces by reducing the annotation-verified task network (shown in Figure 3). (The top down hierarchical reductions are shown in left to right fashion in the figure. The dashed arrow lines show the temporal precedence relations developed between the nodes of the HTN.) The shaded nodes correspond to the parts of the interpreted plan R^i that survive after the annotation verification and refitting process. The white nodes represent the refit tasks added during the annotation verification process, and their subsequent reductions. In the current example, the refitting control strategy recommends that the planner reduce the refit task $A[Clear(L)]$ by putting J on I rather than putting J on *Table* or on K. This decision in turn leads to a shortened plan by allowing the extra goal refit task $A[On(J,I)]$ to be achieved by phantomization.

4. Completeness, Coverage and Efficiency

Completeness : The validation structure based modification is complete in that it will correctly handle all types of applicability failures that can arise during plan modification, and provide the planner with a partially reduced HTN with a consistent validation structure. In particular, our definition of inconsistencies (see §2.2) captures all types of applicability failures that can arise due to a change in the specification of the problem; and our annotation verification procedure provides methods to correctly modify the plan validation structure to handle each type of inconsistency (see section 3.2), without introducing any new inconsistencies into the HTN (a proof is provided in [10].)

Coverage : The validation structure developed here covers the internal dependencies of the plans produced by most traditional

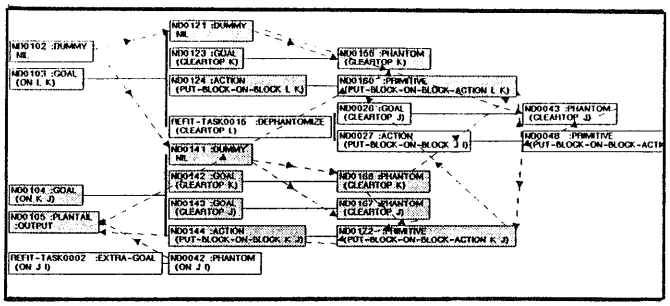

Figure 4. The plan produced by PRIAR 3BS→4BS1

hierarchical planners. The captured dependencies can be seen as a form of explanation of correctness of the plan with respect to the planner's own domain model. By ensuring the consistency of the validation structure of the modified plan, PRIAR guarantees correctness of the modified plan with respect to the planner. However, it should be noted that as the dependencies captured by the validation structure do not represent any optimality considerations underlying the plan, the optimality of modification is not guaranteed. Further, since the modification is integrated with the planner, failures arising from the incorrectness or incompleteness of the planner's own domain model will not be detected or handled by the modification theory. Of course, these should not be construed as limitations of the theory, as its goal is to improve the average case efficiency of the planner.

Flexibility and Efficiency: In the worst case, when none of the steps of R^o are applicable in the new situation, annotation verification will return a degenerate HTN containing refit tasks for all the goals of P^n. In such extreme cases PRIAR may wind up doing a polynomial amount of extra work compared to a pure generative planner. In other words, the worst case complexity of plan modification remains same as the worst case complexity of generative planning. However, on the average, PRIAR will be able to minimize the repetition of planning effort (thereby accrue possibly exponential savings in planning time) by providing the planner with a partially reduced HTN, and conservatively controlling refitting such that the already reduced (applicable) parts of R^a are left undisturbed.

The claims of flexibility and average case efficiency are also supported by the empirical evaluation experiments that were conducted on PRIAR. The plot in figure 5 shows the computational savings achieved when different blocks world problems are solved from scratch and by reusing a range of existing blocks world plans (see [10] for the details of the experimental strategy). For example, the curve marked 7BS1 shows the savings afforded by solving a particular seven-block problem by reusing several different blocks world plans (indicated on the x-axis). The relative savings over the entire corpus of experiments ranged from 30% to 98% (corresponding to speedup factors of 1.5 to 50) with the highest gains shown for the more difficult problems tested. These results also showed that as the size of P^n increases, the computational savings afforded by PRIAR stay very high for a range of reused plans with varying amount of similarity; consider, for example, the plot for the 12BS1 problem in the figure. This latter behavior lends support to the claim of flexibility of the modification framework.

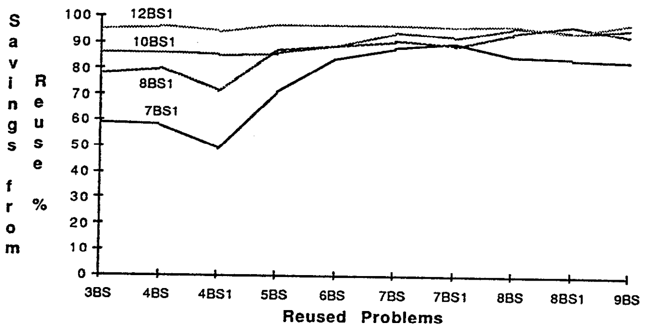

Figure 5. Variation of performance with problem size and similarity

5. Related Work

Representations of plan internal dependency structure have been used by several planners previously to guide plan modification (e.g., the triangle tables and the macro operators of [5] and [8]; the decision graphs of [7] and [4]; the plan rationale representation of [22]). However, our work is the first to systematically characterize the nature of such dependency structures and their role in plan modification. It subsumes and formalizes the previous approaches, provides a better coverage of applicability failures, and allows the reuse of a plan in a larger variety of new planning situations. Unlike the previous approaches, it also explicitly focuses on the flexibility and conservatism of the plan modification. The modification is fully integrated with the generative planning, and aims to reduce the average case cost of producing correct plans. In this sense, PRIAR's strategies are complementary to the plan debugging strategies proposed in GORDIUS [18] and CHEF [6], which use an explanation of correctness of the plan with respect to an external (deeper) domain model (generated through a causal simulation of the plan) to guide the debugging of the plan and to compensate for the inadequacies of the planner's own domain model. Similarly, PRIAR's validation structure based approach to plan modification stands in contrast to other approaches which rely on domain dependent heuristic modification of the plan (e.g. [6, 1, 16]). Our approach of grounding plan modification on validation structure guarantees the correctness of the modification with respect to planner's domain model and reduces the need for a costly modify-test-debug type approach.

6. Conclusion

Our theory of plan modification utilizes the validation structure of the stored plans to yield a flexible and conservative plan modification framework. The validation structure, which constitutes a hierarchical explanation of correctness of the plan with respect to the planner's own knowledge of the domain, is annotated on the plan as a by-product of initial planning. Plan modification is characterized as a process of removing inconsistencies in the validation structure of a plan, when it is being reused in a new (changed) planning situation. The repair of these inconsistencies involves removing unnecessary parts of the HTN, and adding new high-level tasks to it to re-establish failing validations. The resultant partially reduced HTN (with a consistent validation structure) is given to the planner for complete reduction. We discussed the development of this theory in PRIAR system, and characterized its completeness, coverage, efficiency and limitations. This theory provides unified treatment for plan modification involved in replanning, plan reuse and incremental planning.

Acknowledgements

Jim Hendler, Lindley Darden and Larry Davis have influenced the development of the ideas presented here. Jack Mostow and Austin Tate provided useful comments on previous drafts. All three AAAI referees provided very informative reviews. To all, my thanks.

References

[*Please consult the list of references under the companion article* **"Mapping and Retrieval During Plan Reuse: A Validation Structure Based Approach,"** *also in these proceedings.*]