Mapping and Retrieval During Plan Reuse: A Validation Structure Based Approach

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

Effective mapping and retrieval are important issues in successful deployment of plan reuse strategies. In this paper we present a domain independent strategy for ranking a set of plausible reuse candidates in the order of cost of modifying them to solve a new planning problem. The cost of modification is estimated by measuring the amount of disturbance caused to the *validation structure* of a reuse candidate if it were to be reused in the new problem situation. This strategy is more informed than the typical feature based retrieval strategies, and is more efficient than the methods which require partial knowledge of the nature of the plan for the new problem situation to guide the retrieval process. We discuss the implementation of this retrieval strategy in PRIAR, a framework for flexible reuse and modification in hierarchical planning.

1. Introduction

The utility of reusing existing plans to solve new planning problems has been realized early in the planning research, and has more recently received considerable attention in case-based reasoning [6, 1]. A major obstacle to successful deployment of plan reuse schemes has been the problem of plan retrieval and mapping. Retrieving an appropriate plan that can be efficiently reused in the current problem situation, and choosing an appropriate mapping between the objects of the existing plan and the objects of the new planning problem is generally very hard [15]. The payoff from plan reuse depends crucially on both the cost of retrieval and mapping, and the the cost of modifying the retrieved plan to solve the new problem. Thus, for reuse to be effective, the similarity metrics used should be capable of evaluating the ease of modifying an existing plan to solve the new problem, rather than merely measuring surface similarity. For efficiency reasons, most typical retrieval strategies employ straight forward feature matching or domain dependent indexing schemes. Such schemes do not accurately reflect the cost of reusing the retrieved plan in the new problem situation. To be effective, such strategies should also be able to take into account the expected match between the plans for the two problem situations.

While the above limitations have been recognized, many of the proposed alternatives failed to be cost-effective as they require comparison of the solution derivations for the old and new problems (e.g., [2]). In this paper, we provide a partial solution for the retrieval and mapping problem that does not depend on any prior knowledge of the plan for solving the new problem. It makes an informed estimate of the cost of modifying a given plan to solve a new problem by analyzing how the internal dependencies of that plan will be affected in the new problem situation. It utilizes a novel representation of the internal dependency structure of the plan for this purpose. This strategy has been implemented in PRIAR, a framework for flexible reuse and modification of plans [10, 9, 11, 13].

PRIAR utilizes plan *validation structure*, a systematic internal dependency representation of hierarchically generated plans, to guide and control all phases of reuse of a given plan in a new problem situation. In PRIAR, plan modification is seen as a process of repairing the inconsistencies in the validation structure of an existing plan, when it is interpreted in a new problem situation. The cost of modification process depends upon the number and type of these inconsistencies. The retrieval strategy estimates the number inconsistencies for each reuse candidate, and rank orders the candidates based on that estimate. This ordering is facilitated through the development of the notion of the *plan kernel* of a plan. The plan kernel provides a way of encapsulating the validation dependencies between a plan and its problem specification. In the rest of the paper, we briefly discuss PRIAR's validation structure based plan reuse framework, and describe its plan kernel based ordering strategy for choosing among reuse candidates.

2. Overview of PRIAR Plan Modification Framework

2.1. Validation Structure

In PRIAR framework the building blocks of the stored plan dependency structure are validations. A validation is a 4-tuple $\langle E, n_s, C, n_d \rangle$ where the effect E of the task n_s (called source node) in the hierarchical task network (HTN) is used to satisfy (support) the condition C of task n_d (called destination node).

* 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.

For any plan synthesized by a hierarchical planner, there are a finite set of validations, corresponding to the *protection intervals* [3] that are maintained during planning; we denote this set by V. The individual validations are classified depending on the type of the conditions they support. Figure 1 shows the validation structure of the plan for solving the blocks world problem 3BS (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 the condition $On(B,C)$ is required at the goal state n_G, and is provided by the effect $On(B,C)$ of node n_{15}.

The uniqueness of PRIAR framework is in the way the plan validations are stored on the HTN. Each task n in the HTN is annotated with the set of validations that are supplied by, consumed by, or necessarily preserved by the tasks belonging to the sub-reduction (hierarchical wedge) rooted at n. We call these the external effect conditions (e-conditions), external preconditions (e-preconditions) and persistence conditions (p-conditions) respectively of task n. The annotations on a node encapsulate the node's role in the validation structure of the plan. These annotations are computed efficiently for each node in the HTN in a bottom-up, breadth-first fashion at the planning time. In [10], we provide a $O(N^2)$ algorithm (where N is the length of the plan) for doing this.

Using the task annotations introduced before, PRIAR also defines the notion of the *validation state* preceding and following each primitive executable action in the plan. They specify the set of validations that should hold at each point during the plan execution for the rest of the plan to have a consistent validation structure (thereby guaranteeing its successful execution). Of particular interest for the PRIAR retrieval strategy are the validation state following the initial node in the HTN, denoted by $A^s(n_I)$, and the one preceding the goal node, denoted by $A^p(n_G)$. The former contains all validations which are provided by the initial node, n_I and the latter contains all the validations which are consumed by the goal node, n_G.

The annotated validation structure effectively provides a hierarchical explanation of correctness of the plan with respect to the planner's knowledge of the domain. In PRIAR, the validation structure is used (*i*) to locate the parts of the plan that would have to be modified, (*ii*) to suggest appropriate modification actions, (*iii*) to control the modification process such that it changes the existing plan minimally to make it work in the new situation, and (*iv*) to assist in plan mapping and retrieval.

2.2. Plan Modification via Annotation-Verification

Given a plan to be reused to fit the constraints of a new problem situation, PRIAR first maps the plan into the new problem situation. This process, known as *interpretation*, marks the differences between the plan and the problem situation. These differences in turn are seen to produce *inconsistencies* in the plan validation structure (such as missing, failing, or redundant validations). In PRIAR framework, a plan is modified in response to inconsistencies in its validation structure. PRIAR uses a process called *annotation-verification* to suggest appropriate modification to the plan for removing those inconsistencies from the validation structure of the plan. These domain independent modifications depend on the type of the inconsistencies and

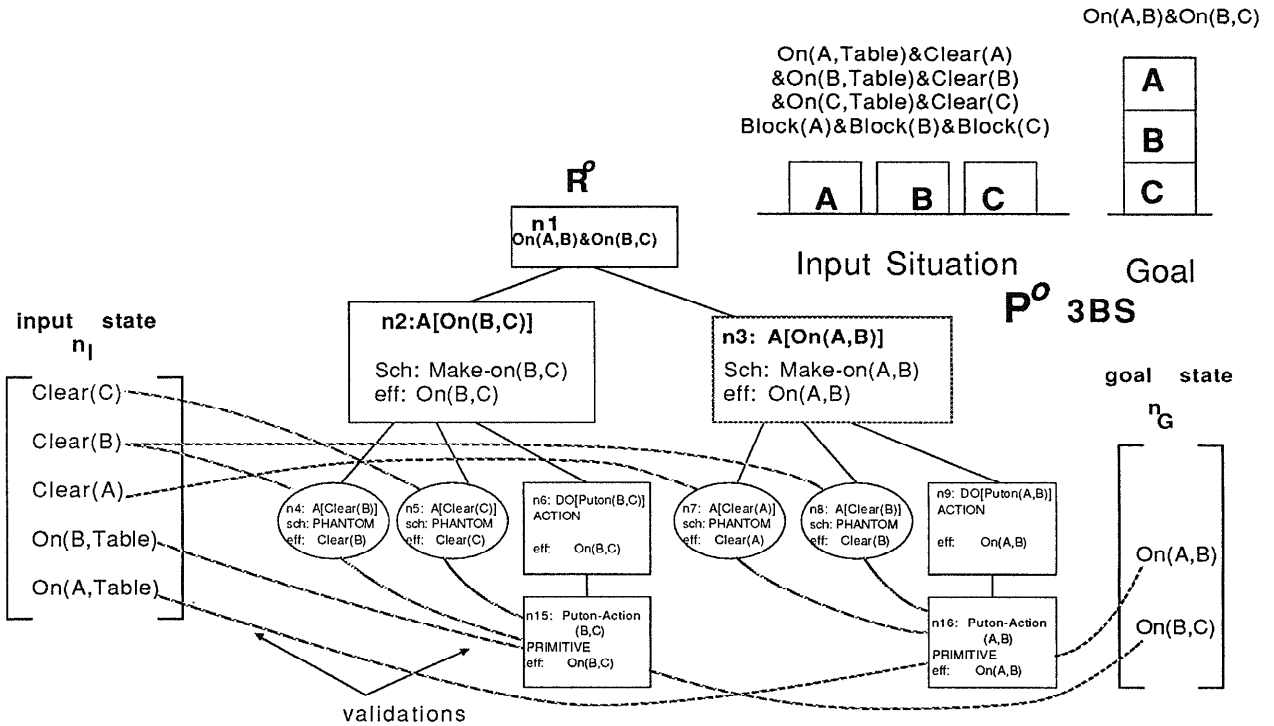

Figure 1. Validation structure of 3BS plan

involve removal of redundant parts of the plan, exploitation of any serendipitous effects of the changed situation to shorten the plan, and addition of high level refit tasks to re-establish any failing validations. At the end of the annotation-verification, which is a polynomial time process process [10], PRIAR will have a *partially reduced* plan with a *consistent* validation structure. PRIAR's hierarchical nonlinear planner accepts this partially reduced plan and produces a completely reduced HTN. The planner uses a conservative heuristic search control strategy called *task kernel-based ordering* (see [9]) to control this process of refitting, so as to localize the modification to the plan and preserve as many of its applicable portions as possible. (Further details of PRIAR modification process can be found in a companion paper [13] in this proceedings.)

3. Mapping and Retrieval in PRIAR

In this section we will assume that PRIAR is given a new problem $P^n=[I^n,G^n]$ and a set of reuse candidates $\{\langle R^o,\alpha\rangle\}$, where R^o is an existing plan, and α is a mapping between the objects of R^o and P^n. PRIAR currently performs a partial unification on the goals of R^o and P^n to get an initial set of reuse candidates. This strategy is described in [10], and is not of particular interest for the purposes of this paper. Here we will concentrate on strategies that exploit the validation structure of the plans to efficiently rank the initial set of reuse candidates in the order of the cost of modifying them to solve the new problem. The heuristic cost metric of each reuse candidate $\langle R^o,\alpha\rangle$ is computed in terms of the number and type of validations of R^o that will be failing in P^n, for the mapping α. We define the notion of *plan kernel* of a reuse candidate to facilitate efficient computation of this cost metric. The degree of match between the plan kernel of the reuse candidate and the new problem situation will be used to guide the ordering.

3.1. Plan Kernels

The plan kernel of a stored plan R^o, $PK(R^o)$, is intended to encapsulate the dependencies between R^o and the features of its input and goal specification. We will formulate it as a collection of validations of R^o that are supported by or supporting the features of the input and goal states of the plan. These validations are further divided into three categories based on the expected difficulty of re-establishing them, in the event that the input and goal state features on which they are dependent no longer hold in the new planning situation. Thus, we define it as a three tuple

$$PK(R^o) = \langle g\text{-features}, f\text{-features}, pc\text{-features}\rangle$$

where the g-features, f-features and pc-features are in turn defined as follows:

g-features (Goal Features): These correspond to the validations of R^o that directly support its goals. Thus

$$g\text{-features}(PK(R^o)) = A^p(n_G)$$

(where $A^p(n_G)$ is as defined in the previous section.)

f-features (Filter Features): These correspond to the validations supported by the input specification of R^o to either the filter conditions (the unachievable applicability conditions) of the plan, or the phantom nodes that achieve some main goal of R^o. Thus, a validation $v:\langle E,n_I,C,n_d\rangle$ belongs to f-$features(PK(R^o))$, iff $v \in A^s(n_I)$ and either C is a filter condition or $\exists v':\langle E',n_{d'},C',n_G\rangle \in A^p(n_G)$ such that $n_{d'}=n_d \wedge C=C'$

pc-features (Precondition Features): These correspond to the the validations supported by the input specification of R^o that support either the preconditions of some node of R^o, or the phantom nodes achieving the preconditions of some node of R^o. In the current framework, these will essentially be all the validations of $A^s(n_I)$ that are not included in the f-features of the plan kernel. That is, $pc\text{-features}(PK(R^o)) =$

$$\{v \mid v \in A^s(n_I) \wedge v \notin f\text{-features}(PK(R^o))\}$$

Based on the above definition, the plan kernel of a plan can be computed in a straightforward fashion from the initial validation state $A^s(n_I)$ and the final validation state $A^p(n_G)$ of that plan. As an example, the plan kernel of 3BS plan shown in figure 1 will be:

$$PK(3BS) = \left[\begin{array}{ll} g\text{-features:} & \left\{\begin{array}{l}\langle On(A,B),n_{16},On(A,B),n_G\rangle \\ \langle On(B,C),n_{15},On(B,C),n_G\rangle\end{array}\right. \\ f\text{-features:} & \left\{\begin{array}{l}\langle Block(B),n_I,Block(B),n_{15}\rangle \\ \langle Block(B),n_I,Block(B),n_{16}\rangle \\ \langle Block(A),n_I,Block(A),n_{16}\rangle \\ \langle Block(C),n_I,Block(C),n_{15}\rangle \\ \langle On(B,Table),n_I,On(B,?x),n_{15}\rangle \\ \langle On(A,Table),n_I,On(A,?x),n_{16}\rangle\end{array}\right. \\ pc\text{-features:} & \left\{\begin{array}{l}\langle Clear(B),n_I,Clear(B),n_4\rangle \\ \langle Clear(B),n_I,Clear(B),n_8\rangle \\ \langle Clear(C),n_I,Clear(C),n_5\rangle \\ \langle Clear(A),n_I,Clear(A),n_7\rangle\end{array}\right.\end{array}\right]$$

Notice that the different features of the problem specification enter the plan kernel only by virtue of the validations that they provide to the plan. Moreover, if any features support multiple validations, they enter the plan kernel once for each of these validations. For example, the features $Block(B)$ and $Clear(B)$ enter $PK(3BS)$ more than once. Thus, the number of times a feature enters the plan kernel, and the type of validations it supports implicitly reflect the relative importance of that feature during retrieval.

3.2. Plan Kernel Based Ordering of Reuse Candidates

The degree of match between the plan kernel of a reuse candidate and the input and goal specification of a new planning problem gives a rough indication as to how much of that plan would be applicable to the new problem and as to what type of validation failures would arise when it is reused in the new problem situation. Since the refitting cost depends to a large extent on the number and type of validation failures, it is reasonable to use this match to estimate the amount of modification that would be needed for reuse. Below we describe a three layered ordering procedure to rank a set of reuse candidates with the help of their plan kernels. The procedure measures the difficulty of reusing the given plan in the new problem situation by estimating the number of inconsistencies that will arise in the validation structure of a plan because of the problem differences.

Given. The new problem $P^n=[I^n,G^n]$, and a set of reuse candidates $\{\langle R^o,\alpha\rangle\}$.

Step 0. The plan kernels of the reuse candidates are computed by translating the plan kernels of the corresponding plans, using the mapping. That is, $PK(\langle R^o,\alpha\rangle) = R^o \cdot \alpha$, where "."

refers to the operation of object substitution.

Step 1. The reuse candidates are ranked based on the number of goals of P^n that will not be supported by the g-features of the plan kernels of individual candidates. The cost function for this layer of ordering will be given by

$$\left|\left\{ g \mid g \in G^n \wedge \left[\begin{array}{l} \exists v : \langle E, n_d, G, n_G \rangle \in \\ \quad g\text{-}features(PK(\langle R^o, \alpha \rangle)) \text{ s.t. } G \vdash g \end{array}\right] \right\}\right|$$

(where $F \vdash f$ is true iff f can be deductively inferred from F and the domain axioms.) Based on this step, the best candidates are those which will need to achieve the least number of extra goals to be reused in the new problem situation.

Step 2. If more than one candidate is ranked best by the ordering of step 1, then the best candidates are ranked further based on the number of f-features of their plan kernels that do not hold in the input specification of the new problem. Thus the cost function for this layer is given by

$$\left| \{ v \mid v : \langle E, n_I, C, n_d \rangle \in f\text{-}features(PK(\langle R^o, \alpha \rangle)) \wedge I^n \not\vdash C \} \right|$$

Step 3. Finally, the best ranked candidates in the ordering of step 2 are further ranked by the number of *pc*-features of their plan kernels that do not hold in the input state of the new problem. Thus, the cost function for this layer is given by

$$\left| \{ v : \langle E, n_I, C, n_d \rangle \mid v \in pc\text{-}features(PK(\langle R^o, \alpha \rangle)) \wedge I^n \not\vdash C \} \right|$$

The best ranked reuse candidates at the end of this three layer ordering procedure are returned as the preferred candidates for reuse in solving P^n.

Remarks: Even if an old plan matches all the goals of P^n, it may still be an inappropriate candidate for solving P^n since the methods it uses to achieve those goals may not be applicable in the new problem situation. This, for example, is the case when a filter condition of the schema used to reduce the task achieving a goal in R^o is no longer true in P^n, or the phantom goal node that achieved a goal in R^o can no longer stay phantom in P^n. The ranking with respect to f-features, carried out in step 2, essentially attempts to prefer candidates which will not have validations supporting such failing filter conditions or failing phantom goals.

The implicit levels of importance attached to the validations at different layers of the plan kernel can be justified in terms of the computational effort needed for re-establishing them in the new problem situation. The main heuristic is that a significant amount of task reduction and interaction resolution would be required to generate subplans to achieve goals of the new problem that are not supported by the retrieved plan, or to replace subplans of the retrieved plan with failing filter conditions, in contrast to the effort required to reachieve the failing preconditions. The second heuristic is that in the event of filter condition failure, it is possible that to exploit some of the previous planning effort (e.g., effort expended in establishing the e-preconditions of the sub-reduction being replaced) in the new planning situation (see [10]). For this reason, the failing filter conditions are considered less costly to handle than new goals.

3.3. Example

Figure 2 shows the initial and goal state specification of 4BS1, a blocks world problem and lists four possible reuse candidates for that problem. For each reuse candidate, the figure shows the the initial and goal state specifications, the mapping between the candidate and the 4BS1 problem, and the plan kernel of the reuse candidate (as an exercise, compare the plan kernels of the reuse candidates $\langle 3BS, [A \rightarrow L, B \rightarrow K, C \rightarrow J] \rangle$ and $\langle 3BS, [A \rightarrow K, B \rightarrow J, C \rightarrow I] \rangle$ with $PK(3BS)$ specified previously). For ease of exposition, the figure shows the validations of the plan kernels only by their supporting effects. It does not include facts of type $Block(?x)$ in the specifications of the problems. Similarly, it also does not explicitly show the validations of the type $\langle Block(?x), -, -, - \rangle$.

When these four reuse candidates are ordered with the help of their plan kernels, at the first layer the g-features of the plan kernels of all the reuse candidates fail to satisfy one goal each of the new problem (4BS1). Thus, they are all deemed equally costly at this layer, and all the candidates move to the ordering at the next layer.

At this layer, the f-feature $\langle On(B,C), -, -, - \rangle$ of the plan kernel of $\langle 3BS\text{-}Phantom, [A \rightarrow L, B \rightarrow K, C \rightarrow J] \rangle$ is not preserved in the input state of the new problem (4BS1) as $I^n \not\vdash On(B,C)$. (This basically means that the top level phantom goal of this reuse candidate has to be re-established if we want to use it to solve P^n.) Similarly, the f-feature $\langle Pyramid(L), -, -, - \rangle$ of the

4BS1:
Init-state(I^n): $On(I,Table)$, $On(K,Table)$, $On(L,Table)$, $On(J,L)$, $Clear(J)$, $Clear(K)$, $Clear(I)$,
Goal-state(G^n): $On(L,K)$, $On(K,J)$, $On(J,I)$

Reuse Candidates $\{\langle R^o, \alpha \rangle\}$

⟨3BS-Phantom, $[A \rightarrow L, B \rightarrow K, C \rightarrow J]$⟩:
Init-state: $On(A,Table)$, $On(B,C)$, $On(C,Table)$, $Clear(A)$, $Clear(B)$
Goal-state: $On(A,B)$, $On(B,C)$
Plan: Put-Block-on-Block-Action(A,B)
PK(3BS-Phantom)·$[A \rightarrow L, B \rightarrow K, C \rightarrow J]$:
[g-features: $\langle On(L,K),-,-,- \rangle$ $\langle On(K,J),-,-,- \rangle$
f-features: $\langle On(K,J),-,-,- \rangle$ $\langle On(L,Table),-,-,- \rangle$
pc-features: $\langle Clear(L),-,-,- \rangle$ $\langle Clear(K),-,-,- \rangle$]

⟨3BS, $[A \rightarrow L, B \rightarrow K, C \rightarrow J]$⟩:
Init-state: $On(A,Table)$,$On(B,Table)$,$On(C,Table)$,$Clear(A)$,$Clear(B)$,$Clear(C)$
Goal-state: $On(A,B)$, $On(B,C)$
Plan: Put-Block-on-Block-Action(B,C)→Put-Block-on-Block-Action(A,B)
PK(3BS)·$[A \rightarrow L, B \rightarrow K, C \rightarrow J]$:
[g-features: $\langle On(L,K),-,-,- \rangle$ $\langle On(K,J),-,-,- \rangle$
f-features: $\langle On(L,Table),-,-,- \rangle$ $\langle On(K,Table),-,-,- \rangle$
pc-features: $\langle Clear(L),-,-,- \rangle$ $\langle Clear(K),-,-,- \rangle$ $\langle Clear(J),-,-,- \rangle$]

⟨3BS, $[A \rightarrow K, B \rightarrow J, C \rightarrow I]$⟩:
Init-state:$On(A,Table)$,$On(B,Table)$,$On(C,Table)$,$Clear(A)$,$Clear(B)$,$Clear(C)$
Goal-state: $On(A,B)$, $On(B,C)$
Plan: Put-Block-on-Block-Action(B,C)→Put-Block-on-Block-Action(A,B)
PK(3BS)·$[A \rightarrow K, B \rightarrow J, C \rightarrow I]$:
[g-features: $\langle On(K,J),-,-,- \rangle$ $\langle On(J,I),-,-,- \rangle$
f-features: $\langle On(J,Table),-,-,- \rangle$ $\langle On(K,Table),-,-,- \rangle$
pc-features: $\langle Clear(I),-,-,- \rangle$ $\langle Clear(K),-,-,- \rangle$ $\langle Clear(J),-,-,- \rangle$]

⟨3BS-Pyramid,$[A \rightarrow L, B \rightarrow K, C \rightarrow J]$⟩:
Init-state: $On(A,Table)$, $On(B,Table)$, $On(C,Table)$, $Clear(A)$, $Clear(B)$, $Clear(C)$, $Pyramid(A)$
Goal-state: $On(A,B)$, $On(B,C)$
Plan: Put-Block-on-Block-Action(B,C)→Put-Pyramid-on-Block(A,B)
PK(3BS-Pyramid)·$[A \rightarrow L, B \rightarrow K, C \rightarrow J]$:
[g-features: $\langle On(L,K),-,-,- \rangle$ $\langle On(K,J),-,-,- \rangle$
f-features: $\langle Pyramid(L),-,-,- \rangle$ $\langle On(L,Table),-,-,- \rangle$ $\langle On(K,Table),-,-,- \rangle$
pc-features: $\langle Clear(K),-,-,- \rangle$ $\langle Clear(J),-,-,- \rangle$]

Figure 2. Example for Plan Kernel Based Ordering

plan kernel of the reuse candidate $\langle 3BS\text{-}Pyramid, [A \rightarrow L, B \rightarrow K, C \rightarrow J] \rangle$ is not preserved since $I^n \not\vdash Pyramid(L)$. If we want to solve 4BS1 by this reuse candidate, the sub-reduction dependent on this filter condition would have to be replaced. Further, the f-feature $\langle On(J, Table), -, -, - \rangle$[1] of the plan kernel of the reuse candidate $\langle 3BS, [A \rightarrow K, B \rightarrow J, C \rightarrow I] \rangle$ is not preserved since $I^n \not\vdash On(J, Table)$. In contrast, none of the f-features of the reuse candidate $\langle 3BS, [A \rightarrow L, B \rightarrow K, C \rightarrow J] \rangle$ fail to hold in the new problem situation. Thus, this is ranked best by the ordering based on the f-features of the plan kernel. Since this is the only best ranked candidate, the ordering at the third layer is not required and $\langle 3BS, [A \rightarrow L, B \rightarrow K, C \rightarrow J] \rangle$ is returned as the preferred reuse candidate for solving the problem 4BS1.

Notice that the plan kernel based ordering is able to discriminate among these reuse candidates even though all the candidates satisfy the same number of goals of P^n. Further, as we mentioned earlier, it is capable of discriminating among different plans as well as different mappings of the same plan. In the current example, the reuse candidates $\langle 3BS, [A \rightarrow L, B \rightarrow K, C \rightarrow J] \rangle$ and $\langle 3BS, [A \rightarrow K, B \rightarrow J, C \rightarrow I] \rangle$ correspond to two different mappings of the same (three blocks) plan. We have seen that the ordering prefers one of the mappings over the other.

3.4. Cost Overestimation in the Ordering Heuristic

A limitation of the plan kernel-based heuristic ordering strategy as discussed in the previous section is that it sometimes overestimates the cost of reusing a plan in the new situation by counting some spurious inconsistencies. This happens when parts of the reuse candidate are rendered redundant in the new problem situation. For example, when the reuse candidate supports some unnecessary goals (the goals which are satisfied by the reuse candidate but are not required in the new planning situation), some of the failing validations may actually be supporting the parts of the plan whose sole purpose is to help achieve the unnecessary goal. Such failing validations should obviously not be counted as inconsistencies, as they can eventually be removed from the HTN and thus do not have to be re-established. As an example, suppose that we are judging the appropriateness of reusing the 3BS plan, shown in figure 1, in a new problem situation where there is no match for the goal $On(A, B)$ (i.e., it is not required). In such a case, the validation $\langle Clear(A), n_1, Clear(A), n_{16} \rangle$ cannot be counted as a failure, even if $Clear(A)$ is not true in the initial state of the new problem—this validation, being an e-precondition of the node n_3:A [$On(A, B)$], will be pruned away eventually thus making its failure inconsequential. Similar situation arises when some of the goals and sub-goals of the reuse candidate are directly satisfied in the input situation of the new problem (see p-phantom validations in [10, 13]). To avoid overestimation in these cases, the parts of the plan that are rendered redundant in the new problem situation would have to be removed, even before reuse candidates are ordered by the plan kernel based ordering. In PRIAR, this type of pruning is done only during annotation verification process, after a reuse candidate is selected (see *prune–validation* process in [10]). Thus, to eventually avoid overestimation of cost, the annotation verification procedure would have to be carried out partially before the reuse candidates are ranked by the plan kernel based ordering. Even though the annotation verification process is only a polynomial time process, we feel that using it during mapping and retrieval stage may still be too expensive. Because of this, we retained the overestimating heuristic in the current implementation of PRIAR. However, this can be changed easily if required.

3.5. Refinements to Plan Kernel Based Ordering

The *informedness* of the ordering procedure presented in section 3.2 can be further improved by exploiting the hierarchical structure of the plan. In particular, the notion of the *level* of a validation can be used as a means of differentiating further among the validations of the individual layers of the plan kernel (in terms of the estimated difficulty of re-establishing them in the event they are not preserved in the new problem situation).

The level of a validation is defined as the reduction level at which that validation is first introduced into the HTN (see [10] for the formalization of this notion). For example, in figure 1, the validation $\langle Block(A), n_1, Block(A), n_{16} \rangle$ is considered to be of a higher level than the validation $\langle On(A, Table), n_1, On(A, Table), n_{16} \rangle$, since the former is introduced into the HTN to facilitate the reduction of task n_3 while the latter is introduced during the reduction of task n_9. A useful characteristic of hierarchical planning is that its domain schemas are written in such a way that the more important conditions are established at higher levels, while the establishment of less important conditions is delegated to lower levels. Thus, the level at which a validation is first introduced into an HTN can be taken to be predictive of the importance of that validation, and the effort required to (re)establish it.[2] The validation levels can be pre-computed efficiently at the time of annotation.

To improve the informedness of the heuristic ordering, we can weight the validations of individual layers by their levels. The cost functions of the ordering procedure will then compute the weighted sum of the number of failing validations. For example, the cost function for the f-feature based ordering step in section 3.2 would now become $\sum_{\Delta} level(v)$, where $\Delta \equiv \{ v : \langle E, n_1, C, n_d \rangle \mid v \in f\text{-}features(PK(\langle R^o, \alpha \rangle)) \wedge I^n \not\vdash C \}$. In the current example, this would mean that the failure of the validation $\langle Block(A), n_1, Block(A), n_{16} \rangle$ would be considered more costly than the failure of the validation $\langle On(A, Table), n_1, On(A, Table), n_{16} \rangle$. This is reasonable since the former necessitates the relacement of a larger sub-plan (the sub-plan rooted at n_3) than the latter (which only leads to the replacement of the sub-plan rooted at n_9; see figure 1).

4. Related Work

Research in analogical problem solving has shown that the appropriateness of reusing a previous problem's solution to solve a new problem cannot be accurately judged through a

[1] We follow the convention of [20] and classify $On(J, ?x)$ as a filter condition rather than a precondition. Some effects of the plan depend on the binding of $?x$ and one way of correctly propagating the effects when the binding of $?x$ changes is to re-reduce the corresponding task.

[2] We assume that domain schemas having this type of abstraction property have been supplied/encoded by the user in the first place. What we are doing here is to exploit the notion of importance implicit in that abstraction.

simple matching of the problem specifications. However, the alternatives, such as the one proposed by Carbonell [2] in derivational analogy, tend to be very costly, as they require that the solution derivations rather than the problem specifications be compared during retrieval. To compare derivations, the new problem would first have to be partially solved by some non-analogical methods. The retrieval method proposed here falls in the middle ground as it does essentially feature based matching, but takes the validation structure of the solution into account during the matching. This latter characteristic gives it the ability to make a more informed estimate of the importance of individual feature matches on the cost of the overall modification.

The principle motivation behind our strategy is that mapping and retrieval should be guided by the features of the existing plans that are predictive of the amount of modification required to reuse them in the new problem situation. In this sense, it has some similarities to the CHEF [6] retrieval strategy which gives importance to the features that are predictive of execution time failures and interactions. However, in contrast to CHEF, which *learns* the features predictive of the interactions (through an explanation based generalization of execution time failures), PRIAR uses the existing validation structure of the plan to decide the relative importance of the individual features. To some extent, this difference is a reflection of the differing nature of the tasks that are addressed by the two systems—while PRIAR tries to modify plans in the presence of a generative planner, and ensure correctness of the modification with respect to that planner, CHEF relies on the heuristic modification of the retrieved plans and tests the correctness through a domain model based simulation.

5. Conclusion

Our main contribution to the mapping and retrieval problem is a domain independent heuristic strategy for utilizing the validation structure of the stored plan to decide the appropriateness of reusing it in a new problem situation. The central idea is to estimate the cost of modifying the plan to solve the new problem, and prefer the candidate with least expected modification cost. The modification cost is estimated by measuring the amount of disturbance that would be caused to the validation structure of a reuse candidate in new problem situation. We have discussed the implementation of this retrieval strategy in PRIAR, a framework for flexible reuse and modification of plans. We argued that our strategy is more *informed* than the typical feature based retrieval strategies, and more *efficient* than the methods which require partial knowledge of the nature of the plan for the new problem situation to guide the retrieval process. Retrieving plans based solely on the plan kernel based ordering may still be too expensive when the plan library is very large. In such cases, the initial retrieval of candidate plans, prior to the plan kernel based ordering may have to be based on a domain dependent retrieval strategy. However, the plan kernel based ordering strategy can act in conjunction with such a gross feature-based retrieval strategy to make a more informed estimate of the utility of reusing a plan in the given problem situation.

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. E. Charniak and D. McDermott, "Chapter 9: Managing Plans of Actions", in *Introduction to Artificial Intelligence*, Addison-Wesley Publishing Company, 1984, 485-554.
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. R. Fikes, P. Hart and N. Nilsson, "Learning and Executing Generalized Robot Plans", *Artificial Intelligence 3* (1972), 251-288.
6. K. J. Hammond, "CHEF: A Model of Case-Based Planning", *Proceedings of 5th AAAI*, 1986, 267-271.
7. P. J. Hayes, "A Representation for Robot Plans", *Proceedings of 4th IJCAI*, 1975.
8. M. N. Huhns and R. D. Acosta, "ARGO: A System for Design by Analogy", *IEEE Expert*, Fall 1988, 53-68. (Also appears in Proc. of 4th IEEE Conf. on Appln. of AI, 1988).
9. 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.
10. S. Kambhampati, "Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach", CS-Tech. Rep.-2334, CAR-Tech. Rep.-4698, Center for Automation Research, Department of Computer Science, University of Maryland, College Park, MD 20742, October 1989. (Ph.D. Dissertation).
11. 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.
12. S. Kambhampati, "Mapping and Retrieval during Plan Reuse: A Validation-Structure Based Approach", *Proceedings of Eighth AAAI*, Boston, MA, 1990.
13. S. Kambhampati, "A Theory of Plan Modification", *Proceedings of Eighth AAAI*, Boston, MA, 1990.
14. J. L. Kolodner, "Maintaining Organization in a Dynamic Long-term Memory", *Cognitive Science 7* (1983), 243-280.
15. J. L. Kolodner, "Reconstructive Memory: a Computer Model", *Cognitive Science 7* (1983), 281-328.
16. J. L. Kolodner, "Case-Based Problem Solving", *Proceedings of the Fourth International Workshop on Machine Learning*, University of California, Irvine, June 1987, 167-178.
17. E. D. Sacerdoti, *A Structure for Plans and Behavior*, Elsevier North-Holland, New York, 1977.
18. R. Simmons, "A Theory of Debugging Plans and Interpretations", *Proceedings of 7th AAAI*, 1988, 94-99.
19. A. Tate, "Project Planning Using a Hierarchic Non-Linear Planner", Research Report 25, Department of AI, University of Edinburgh, 1976.
20. A. Tate, "Generating Project Networks", *Proceedings of 5th IJCAI*, 1977, 888-893.
21. D. E. Wilkins, "Domain-independent planning: representation and plan generation", *Artificial Intelligence 22* (1984), 269.
22. D. E. Wilkins, "Recovering from execution errors in SIPE", *Computational Intelligence 1* (1985).