Computational Intelligence, Volume 10, Number 2, 1994

EXPLOITING CAUSAL STRUCTURE TO CONTROL RETRIEVAL AND REFITTING DURING PLAN REUSE

SUBBARAO KAMBHAMPATI

Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287, U.S.A.

The ability to reuse existing plans to solve new planning problems can enable a domain-independent planner to improve its average case efficiency by exploiting the problem distribution and avoiding repetition of planning effort. The pay-off from plan reuse, however, crucially depends on finding effective solutions to two important underlying control problems: (i) controlling the retrieval of an appropriate plan and mapping to be reused in a new situation, and (ii) controlling the modification (refitting) of the retrieved plan so as to minimize perturbation to the applicable parts of the plan. This paper is concerned with the development of efficient domain-independent solutions to these two problems. For the retrieval, it provides a domain independent similarity metric that utilizes the plan causal dependency structure to estimate the utility of reusing a given plan in a new problem situation. For the refitting, it presents a minimum-conflict heuristic, again based on the causal dependency structure of the plan, to conservatively control the modification. The paper also discusses the implementation and evaluation of these strategies within the PRIAR plan modification framework.

Key words: planning, plan reuse, plan retrieval, plan modification, similarity metrics, adaptation, case-based planning.

1. INTRODUCTION

The utility of reusing existing plans to solve new planning problems has been realized early in planning research, and has more recently received considerable attention in the planning and learning communities. By exploiting problem distribution and avoiding repetition of planning effort, reuse promises significant savings in the average case efficiency of planning. The payoff from plan reuse does, however, crucially depend on both the the ability to retrieve an appropriate plan to be reused and the ability to efficiently modify the retrieved plan to solve the new problem. Indiscriminate retrieval strategies can degrade a planner's performance by making it spend an inordinate amount of time in the retrieval phase (thereby offsetting any potential savings from reuse) and/or retrieve an inappropriate reuse candidate. Similarly, once a candidate plan is selected, the planner should be capable of modifying the retrieved plan with less computational effort than would be required to solve the new problem from scratch. To ensure this, the refitting process should be conservative and retain all applicable parts of the retrieved plan; otherwise modification could degenerate into planning from scratch. Thus, for reuse to lead to realistic improvements in planning performance, effective solutions must be devised to control the retrieval and refitting stages.

For retrieval to be effective, the similarity metrics used should be capable of evaluating the ease of modifying an existing plan to solve the new problem. This cannot in general be done by merely measuring surface similarity between the problems. Instead, the retrieval strategies need to estimate the expected match between the plans for the two problem situations. Similarly, for refitting to be effective, parts of the retrieved plan that are already applicable in the current problem situation should be left undisturbed as much as possible. However, strict protection of all the applicable parts of the old plan is not a desirable solution, as it may in general lead to both inefficiency and loss of completeness (cf. Waldinger 1977). What we need instead are control strategies capable of comparing the relative disturbances caused by various modifications and selecting the best.

[0]Parts of this work have previously been presented at IJCAI-89 and AAAI-90.

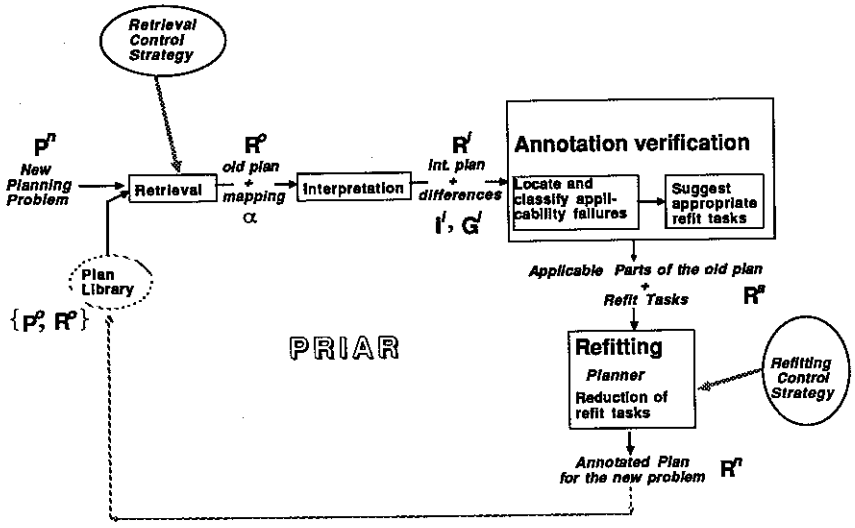


FIGURE 1. Schematic overview of PRIAR.

In this paper, we present domain-independent techniques for controlling retrieval and refitting based on plan causal dependency structures. For mapping and retrieval, we present a class of efficient domain independent similarity metrics that utilize the causal structure of a plan to estimate the utility of reusing it in a new problem situation. For refitting control we develop "minimum-conflict" type heuristic strategies, which exploit the causal structure to guide the modification so as to cause least amount of disturbance to the applicable parts of the plan.

Our techniques are developed and implemented in a framework for plan modification called PRIAR, which allows flexible and conservative modification of plans generated by a hierarchical nonlinear planner (Kambhampati and Hendler 1992; Kambhampati 1989). Figure 1 shows the schematic overview of the PRIAR plan modification framework. In this framework, the causal and teleological structure of generated plans are represented in a form of explanation of correctness of the plan called the "validation structure." Individual planning decisions made during the generation of the plan are justified in terms of their relation to the validation structure. Modification is characterized as a process of detecting and removing inconsistencies in the validation structure resulting from the externally imposed constraints. The cost of the modification process depends upon the number and type of these inconsistencies. Repair actions utilize the dependency structures to transform a completed plan with an inconsistent validation structure into a partially reduced plan with a consistent validation structure. The repair of these inconsistencies involves removing unnecessary parts of the plan and adding new nonprimitive tasks to the plan to establish missing or failing validations. The resultant partially reduced plan is then sent to the planner for full reduction, and a completed plan is produced. This last stage is referred to as the "refitting" stage.

Within the PRIAR modification framework, the problem of retrieval and mapping can be characterized as that of selecting a reuse candidate with the lowest estimated cost of modification to solve the given problem. Similarly, the refitting control can be characterized as the problem of guiding the planner conservatively while it refines the partially reduced plan provided to it by the modification process. In particular, this refinement process should be

controlled so as to cause the least amount of disturbance to the parts of the previous plan that are already found to be applicable in the new situation by the annotation verification process.

For the retrieval problem, we provide a partial solution 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 the new problem by analyzing how the internal dependencies of that plan will be affected in the new problem situation. In particular, we provide a domain-independent similarity metric based on the plan validation structure to measure the utility of modifying a given plan to solve the new problem. The key insight used in these metrics is that the modification cost depends on the number and types of inconsistencies caused by the new problem specification in the validation structure of the retrieved plan. These metrics are domain-independent and take the surface specification as well as the structure of the solution into account. They thus strike a balance between the purely syntactic feature-based retrieval methods, and the methods which require a comparison of the solutions of the new and old problems to guide the retrieval (e.g., Carbonell 1983).

For the refitting control problem, we provide minimum-conflicts type heuristic search control strategies that compare the relative disturbances caused by various refitting choices, and select the one causing the least disturbance. The strategies use the validation structure of the plan to measure the disturbance (interactions) caused by each refitting choice. While most minimum-conflict type heuristics tend to concentrate on minimizing the *number* of conflicts, our strategies use the plan validation structure to also weight the conflicts in terms of the estimated difficulty of repairing them.

The rest of this paper provides a description and evaluation of the retrieval and refitting control strategies. Section 2 contains a review of the previous work that addressed these problems. Section 3 presents a brief overview of the PRIAR modification framework. Section 4 develops and analyzes a computational measure of similarity based on plan validation structure to select mappings and retrieval candidates. Section 5 develops and analyzes a validation-structure-based refitting control strategy. Sections 4 and 5 are largely self-contained and include discussion and evaluation of the respective techniques. Section 6 briefly discusses our experience in applying these techniques to guide incremental plan maintenance in a manufacturing planning domain. Section 7 provides a summary of the paper's main contributions.

2. RELATED WORK

In this section, we will briefly review previous research that addressed the issues of retrieval and refitting during plan reuse and motivated our work.

2.1. Mapping and Retrieval

The problem of retrieval and mapping has received considerable attention in the case-based reasoning and analogical problem-solving communities (Gentner 1983; Kolodner 1983; Hammond 1990; Carbonell 1986). Most existing retrieval methods fall into two broad categories. The first can be characterized as *feature based* and includes the majority of existing methods. Feature based retrieval methods attempt to judge similarity by matching the syntactic specifications of the new problem and the existing problems. For most problems, however, the surface features of the problem specification are not very predictive of the structure of the solution. Thus, feature based similarity measures often fail to accurately estimate the cost involved in modifying a given plan to solve the new problem. Some methods deal with this problem by supplementing the problem specification with information about the saliency of

individual features and doing a weighted feature match. There are two problems with this approach. First, saliency information used by these methods is often very domain specific, making the retrieval techniques themselves very domain dependent. Second, the saliency of a specification feature may in general change from problem to problem—what is very important for one problem may not be so in solving another problem. Thus, even if domain dependent saliency measures are available, they may still be ineffective in guiding retrieval since saliency should ideally be judged with respect to a specific problem and its plan.

The methods in the second group, such as Carbonell's derivational analogy (Carbonell 1986), try to deal with the limitations of feature based retrieval by comparing the derivation of the solutions of the existing problems to the derivation of the new problem. Since new problems have no solution to begin with, these methods depend on solving the new problem at least partially so that they can compare derivations. This makes them too costly for general-purpose retrieval.

The similarity measure 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 stage. 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. It can be best understood as a saliency-based feature matching, where the saliency information is not domain specific but is instead culled from the causal dependency structure of the plan for solving the particular problem.

The principal 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 (Hammond 1990) 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 causal dependency 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 (Kambhampati 1990).

2.2. Refitting Control

Any technique that attempts to find a solution for a new problem by modifying an existing solution, or tries to improve an existing solution by debugging it, has to be concerned about the number of changes done to the chosen solution. Ideally, such a technique should be *conservative* in that it should modify, debug, or extend the existing solution without disturbing the parts of the solution that are already applicable in the new situation. There is a long history of planning systems that use minimization of perturbation to the overall plan as a basis for modification and repair of plans. The general strategy for affecting such localization of search process is to develop a measure of perturbation, and rank various choices in the search space in terms of the expected perturbation. Several different realizations of this basic strategy, usually called the minimum-conflict type heuristics, have been proposed. Hammond's case-based planner, CHEF (Hammond 1990), uses the explanation of an execution time failure to suggest various minimally interactive ways of repairing that failure and then uses domain dependent heuristics to select among the repair strategies. The debugger in Simmons's GTD system (Simmons 1988) selects among possible repairs by doing a causal simulation of the plan with the suggested repairs, followed by an assessment of the global effect of the

suggested repair on the final outcome of the plan in terms of the "bugs" they cause in the plan structure. Other systems, such as PRIDE (Mittal and Araya 1986) and CAS (Turner 1987) store specific handcoded strategies for repairing individual failing preconditions, and use them to guide refitting. More recently, Zweben *et al.* (1990) describe the application of min-conflict heuristics to dynamic revision of schedules. Min-conflict heuristics have also been widely applied in repair-based approaches to constraint satisfaction problems.[1] Examples include Minton *et al.*'s local-search-based methods for constraint satisfaction problems (Minton *et al.* 1990), and the recent work by Selman *et al.* (1992) for solving satisfiability problems.

PRIAR's refitting control strategy is an attempt to provide a domain-independent realization of a minimum conflict heuristic search control strategy for plan modification in hierarchical planning. In particular, the different choices for modifying (extending) the plan are ranked by the amount of interactions they would introduce into the plan being reused. The interactions are measured efficiently with the help of the causal dependency structure of the plan. Unlike most of the techniques described above, which are only concerned with minimizing the number of conflicts, PRIAR also weights the inconsistencies in terms of the estimated difficulty of repairing the inconsistency. Further, PRIAR employs a consistency check based on the domain axioms to get a more realistic estimate of the conflicts caused by a particular modification choice (see Section 5.4). In this latter respect PRIAR's realization of the minimum conflicts heuristic also bears similarity to the "consistency" based heuristics for guiding planning such as the temporal coherence heuristic of Drummond and Currie (1989) and the loop control heuristics of Feldman and Morris (1990). Such heuristics try to direct search away from the paths that may require extensive interaction resolution and/or backtracking by anticipating harmful interactions with the help of a domain-constraint-based consistency analysis.

3. PRELIMINARIES AND OVERVIEW OF PRIAR PLAN MODIFICATION FRAMEWORK

In this section, we establish some terminology for hierarchical nonlinear planning and briefly describe how plans are represented and modified in PRIAR. In hierarchical planning, plans are represented as partially ordered networks of tasks at varying levels of abstraction. 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. Planning is considered complete when all the tasks in the plans are either *primitive* (tasks that cannot be decomposed any further) or *phantom* (tasks whose intended effects are achieved as side effects of some other tasks). Some well-known hierarchical planners include NOAH (Sacerdoti 1977), NONLIN (Tate 1977), and SIPE (Wilkins 1984).

In the PRIAR framework (Kambhampati and Hendler 1992; Kambhampati 1989), a plan is formally represented by a structure called a "hierarchical task network" (HTN). A HTN is a 3-tuple:

$$\langle P : \langle T, O \rangle, T^*, D \rangle$$

where P is a partially ordered plan such that

(i) T is the set of tasks of the plan P. (We will use the names *tasks*, *steps*, and *nodes* interchangeably to denote the members of T.) T contains two distinguished tasks n_I and n_G (standing for the input and goal state specification, respectively). Each task has

[1] In these problems, one starts with a complete but possibly inconsistent assignment for all variables, and incrementally changes the assignments to make it consistent. Unlike planning, where the intermediate repairs can introduce new subgoals, the repairs in constraint satisfaction problems do not change the size of the problem.

a set of applicability conditions and effects associated with it. They are represented as unquantified literals in first order predicate calculus. Following Charniak and McDermott (1984) and Tate (1977), we 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 should not achieve.

(ii) O defines a partial ordering on T. We shall use the notation "$n_1 < n_2$" (where $n_1, n_2 \in T$) to indicate that n_1 is ordered to *precede* n_2 according to this partial ordering. 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).

(iii) T^* is the union of tasks in T and their ancestors.

(iv) D defines a set of parent, child relations among the tasks of T^* such that n_c is a child of a task n_p if n_c is introduced into the plan as a result of reducing n_p. The set consisting of a node n and all its descendants (i.e., its children, their children, and so on) in the plan is called the subreduction of n, and is denoted by $R(n)$.

The notation "$F \vdash f$" is used to indicate that f directly follows from the set of facts in F (i.e., $\exists f' \in F$ such that f' codesignates with f), and the notation "$F \vdash^d f$" to indicate that f deductively follows from the facts in F and the domain axioms. For example, in the blocks world, if $F = \{On(A, B), Clear(C)\}$ then $F \vdash Clear(C)$, and $F \vdash^d \neg Clear(B)$ (since $On(?x, ?y) \supset \neg Clear(?y)$ is a domain axiom). Finally, the modal operators "$\Box$" and "$\Diamond$" denote necessary and possible truth of an assertion. (An assertion is said to be necessarily true in the situation preceding a step s in the plan, if it is true in every ground linearization of that plan.)

The causal structure of the plan is represented by a set of producer-consumed dependency links among the tasks of T. A dependency link, referred to as a *validation* of the plan, is a 4-tuple $\langle E, n_s, C, n_d \rangle$ where the effect E of the task $n_s \in T$ (called source node) is used to satisfy (support) the condition C of task $n_d \in T$ (called destination node). We can show that for any correct plan there exist a finite set of such validation links; we denote this as $\mathbf{V}$. The individual validations are further distinguished based on the type of conditions they support and the level at which they are introduced into the plan.

Figure 2 shows the validation structure of the plan for solving the blocks world problem 3BS (shown in the figure). Validations are represented graphically as dashed lines between the effect of the source node and the condition of the destination node. (To simplify the diagram, validations supporting conditions of the type $Block(?x)$ are not 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}.

Let $\mathbf{V}$ be the set of validations of a hierarchical task network (HTN), and I and G be the initial and goal state specifications of the HTN. We define the correctness of the HTN in terms of its set of validations in the following way:

- For each $g \in G$, and each applicability condition C of the tasks $t \in T$, there exists a validation $v \in \mathbf{V}$ supporting that goal or condition. If this condition is not satisfied, the plan is said to contain *missing validations*.
- None of the plan validations are violated. That is $\forall v : \langle E, n_s, C, n_d \rangle \in \mathbf{V}$, (i) $E \in$ *effects*(n_s) and (ii) $\nexists n \in T$ s.t. $(\Diamond(n_s < n < n_d)) \wedge$ *effects*$(n) \vdash \neg C$. If this constraint is not satisfied, then the plan is said to contain *failing validations*.

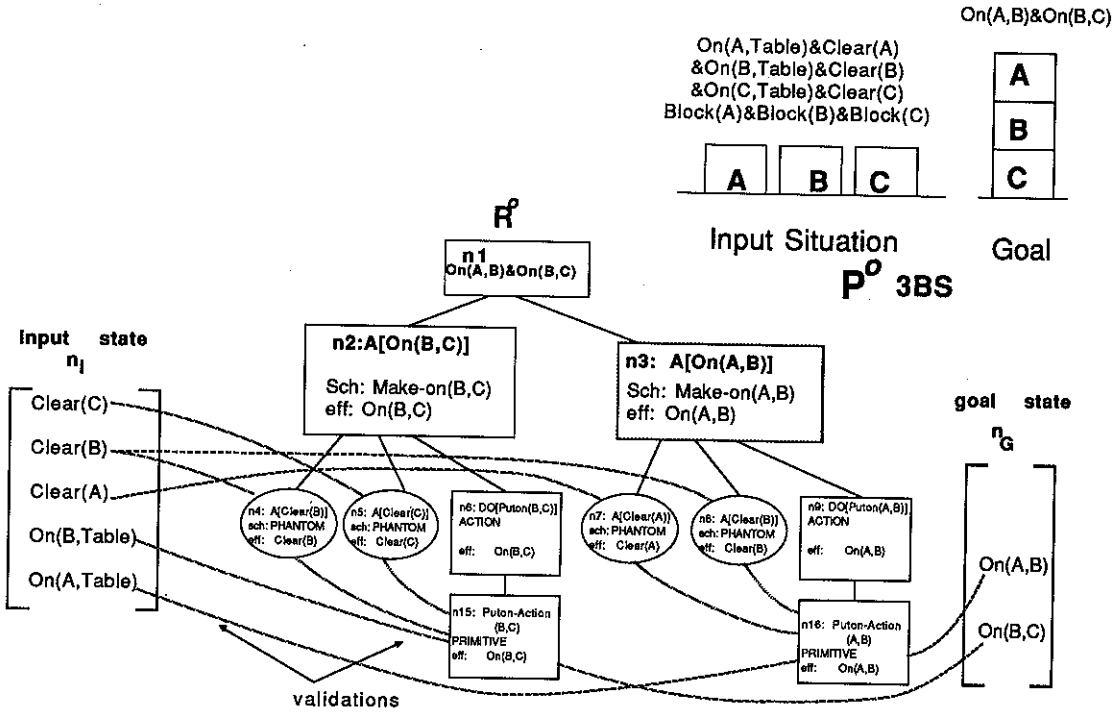


FIGURE 2. 3BS plan and its validation structure.

In addition, we introduce a condition of nonredundancy as follows:

- For each validation $v : \langle E, n_s, C, n_d \rangle \in \mathbf{V}$, there exists a chain of validations the first of which is supported by the effects of n_d and the last of which supports a goal of the plan. If this constraint is not satisfied, then the plan is said to contain *unnecessary validations*.

A plan that does not contain any missing, failing, or unnecessary validation is said to have consistent validation structure. The missing, failing, and unnecessary validations defined above are collectively referred to as *inconsistencies* in the plan validation structure. Note that this definition of correctness is applicable to both completely and partially reduced HTNs. In particular, a completely reduced plan with a consistent validation structure constitutes a valid executable plan.

To facilitate efficient reasoning about the correctness of the plan, and to guide incremental modification, we characterize the role played by the individual steps of the plan and the planning decisions underlying the development of the plan in terms of their relation to the validation structure of the plan. We accomplish this by annotating the individual nodes of the HTN of the plan with the set of validations that encapsulate the role played by the subreduction below that node in the validation structure of the overall plan. In particular, for each task $n \in T^*$, we define the notions of e-conditions, e-preconditions, and p-conditions as follows: The e-conditions of a task represent the set of validations that it or its descendants in the HTN provide to the rest of the plan. If $R(n)$ represents the subreduction of n,[2] e-conditions(n) is given by the set of validations

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

The e-preconditions represent the set of validations that the task or its descendants consume from the rest of the plan. *e-preconditions*(n) is given by the set of validations

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

Finally, the p-conditions represent the validations that should necessarily be preserved by the effects of the task and its descendants to guarantee the correctness of the plan. *p-conditions*(n) is given by the set of validations

$$\{v_i : \langle E, n_s, C, n_d \rangle \mid v_i \in \mathbf{V}; n_s, n_d \notin R(n) \wedge \exists n' \in T \; s.t. \; n' \in R(n) \wedge \Diamond(n_s < n' < n_d)\}$$

where $\Diamond(n_s < n' < n_d)$ is true if n' possibly falls between n_s and n_d for some total ordering of the partially ordered plan. For example, for the node n_3 in Fig. 2, the validation $\langle On(A, B), n_{16}, On(A, B), n_G \rangle$ is an e-condition, the validation $\langle Clear(A), n_I, Clear(A), n_7 \rangle$ is an e-precondition, and the validation $\langle On(B, C), n_{15}, On(B, C), n_G \rangle$ is a p-condition, respectively. The annotations on a node encapsulate the node's role in the validation structure of the plan. These annotations can be computed efficiently for each node in the HTN in a bottom-up, breadth-first fashion at planning time.

Using the task annotations introduced earlier, we also define the notion of the *validation state* preceding and following each primitive executable action in the plan.

$$\textit{Preceding validation state } A^p(n) = \textit{e-preconditions}(n) \cup \textit{p-conditions}(n)$$
$$\textit{Succeeding validation state } A^s(n) = \textit{e-conditions}(n) \cup \textit{p-conditions}(n)$$

[2]Note that $R(n) = \{n\}$ if $n \in T$ (i.e., n is a leaf node in the HTN).

FIGURE 2. 3BS plan and its validation structure.

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 (which in turn guarantees its successful execution, modulo the correctness of the planner's domain model and barring any unexpected events). 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.

During retrieval, it is often useful to know which parts of the initial state specification of a plan are directly involved in supporting a particular validation in the plan. (In Section 4.2, we will see that this helps the retrieval strategy in focusing on only those features of the input specification which will serve some useful purpose in solving the new problem.) For this purpose, we define the notion of *input-support* of a validation $\langle E, n_s, C, n_d \rangle$ as follows:

$$\textit{input-support}(v : \langle E, n_s, C, n_d \rangle) = \begin{cases} v & \text{if } n_s = n_I \\ \bigcup_{v' \in e-preconditions(n_s)} \textit{input-support}(v') & \text{if } n_s \neq n_I \end{cases}$$

For example, in the 3BS plan shown in Fig. 2, we have

$$\textit{input-support}(\langle On(A, B), n_{16}, On(A, B), n_G \rangle) = \begin{pmatrix} \langle Clear(B), n_I, Clear(B), n_8 \rangle \\ \langle Clear(A), n_I, Clear(A), n_7 \rangle \\ \langle On(A, Table), n_I, On(A, Table), n_{16} \rangle \end{pmatrix}$$

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. Figure 1 shows the schematic overview of the PRIAR plan modification framework. 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). Modification is characterized as the process of removing these inconsistencies (caused by the specification of the new problem) from the validation structure of the plan. PRIAR uses a polynomial-time process called *annotation-verification* to locate the inconsistencies and suggest appropriate modifications to remove them from the plan validation structure. These domain-independent modifications depend on the type of the inconsistencies and 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 reestablish any failing validations. After annotation-verification, PRIAR will have a *partially reduced* plan with a consistent validation structure. In the next stage, called the *refitting stage*, PRIAR's hierarchical nonlinear planner accepts this partially reduced plan and reduces it further to produce a completely reduced HTN. To ensure efficiency of reuse, PRIAR employs validation-structure-based strategies for controlling retrieval and refitting. The rest of this paper is devoted to the discussion of these control strategies. The details of the modification process itself can be found in (Kambhampati and Hendler 1992).

4. CONTROLLING MAPPING AND RETRIEVAL

The modification strategy discussed in the previous section enables a planner to flexibly reuse a given plan in a new problem situation. To build an effective plan reuse framework around this modification strategy, we still need a methodology for selecting an appropriate plan to be reused given a new planning problem. PRIAR's retrieval method works by matching the causal dependency structure of the existing plans to the new problem situation to estimate the cost of modifying that plan to fit the specification of the new problem situation. Formally, given a 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, the objective of PRIAR retrieval strategy is to select the candidate that can solve the new problem with the lowest expected modification cost.[3] PRIAR does this by estimating the inconsistencies that would be caused in the validation structure of the reuse candidate if it is used in the new problem situation. In particular, it defines the notion of *plan kernel* for every stored plan and ranks the reuse candidates by the degree of match between the candidate and the new problem situation. The following sections provide the details of this strategy.

4.1. 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 reestablishing 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 Section 3).

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 $f\text{-}features(PK(R^o))$, 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 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,

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

[3] PRIAR currently performs a partial unification on the goals of R^o and P^n to get an initial set of reuse candidates (see Section 4.5).

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 Fig. 2 will be:

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

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.*

4.2. Plan-Kernel-Based Ordering of Reuse Candidates

Plan kernels can be used to develop an efficient similarity metric to rank a set of reuse candidates in the order of the cost of modifying them to solve the new problem. As defined earlier, a *reuse candidate* consists of a plan R^o, and a mapping α between the objects of that plan and the new problem situation. The plan kernel of a reuse candidate $\langle R^o, \alpha\rangle$ is obtained by substituting α in $PK(R^o)$. To obtain an initial set of reuse candidates to be ranked, PRIAR performs a partial unification on the goals of the new problem, and those of the stored plans (see Section 4.5).

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. In Figure 3, 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 a given reuse candidate in the new problem situation by estimating the number of inconsistencies caused by the new problem specification in the validation structure of the reuse candidate.

While ranking the reuse candidates with respect to the f-features and pc-features of the plan kernel, only the validations that ultimately support a goal of the reused plan which matches with some goal of the new problem (under the given mapping) should be considered. This is because the rest of the validations of $A^s(n_I)$ do not serve any useful purpose in the new problem and will most probably be pruned from the final plan. As an example, suppose that we are judging the appropriateness of reusing the 3BS plan, shown in Fig. 2, in a new problem situation, and we find that under the chosen mapping there is no match for the goal $On(A, B)$

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

Step 0. Compute the plan kernels of the reuse candidates by translating the plan kernels of the corresponding plans, using the mapping associated with the reuse candidate. That is, $PK(\langle R^o, \alpha\rangle) = R^o \cdot \alpha$, where "$\cdot$" refers to the operation of object substitution.

Step 1. Rank reuse candidates 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

$$|\{g \mid g \in G^n \wedge (\nexists v : \langle E, n_d, C, n_G\rangle \in g\text{-}features(PK(\langle R^o, \alpha\rangle))\, s.t.\, C = g)\}|$$

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. In case of a tie in step 1, rank the best candidates further based on the number of f-features of their plan kernels that indirectly support some matched goal of the new problem, but do not hold in the input specification of the new problem. Thus the cost function for this layer is given by

$$|\{v \mid v : \langle E, n_I, C, n_d\rangle \in f\text{-}features(PK(\langle R^o, \alpha\rangle)) \wedge v \in GSV(PK(\langle R^o, \alpha\rangle) \wedge E \notin I^n\}|$$

where $GSV(PK(\langle R^o, \alpha\rangle))$ is the set of validations in $PK(\langle R^o, \alpha\rangle)$ that indirectly support the g-feature validations which match with some goal of P^n under the mapping α (see below). It is given by Section 3:

$$GSV(PK(\langle R^o, \alpha\rangle)) = \{input\text{-}supports(v) \mid v : \langle E, n_d, C, n_G\rangle \in g\text{-}features(PK(\langle R^o, \alpha\rangle)) \wedge \exists g \in G^n\, C = g\}$$

Step 3. In case of a tie in step 2, rank the best-ranked candidates further ranked by the number of pc-features of their plan kernels that indirectly support some matched goal of the new problem, but do not hold in the input state of the new problem. Thus, the cost function for this layer is given by

$$|\{v : \langle E, n_I, C, n_d\rangle \mid v \in pc\text{-}features(PK(\langle R^o, \alpha\rangle)) \wedge v \in GSV(PK(\langle R^o, \alpha\rangle) \wedge C \notin I^n\}|$$

FIGURE 3. Plan-kernel-based ordering.

of 3BS in the new problem. 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 (since n_3 has no useful purpose), thus making its failure inconsequential. Steps 2 and 3 of the ranking procedure shown in Fig. 3 use the notion of *input-supports* of a validation (as defined in Section 3) to avoid this type of cost overestimation. 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.

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 reestablishing them in the new problem situation. One 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, we expect that less effort is required to reachieve the failing preconditions. Another heuristic is that in the event of filter condition failure, it is possible to exploit some of the previous planning effort (e.g., effort expended in establishing the e-preconditions of the subreduction being replaced) in the new planning situation. For this reason, the failing filter conditions are considered less costly to handle than new goals.

4.3. Example

Figure 4 shows the initial and goal state specification of the blocks world problem 4BS1, and lists four possible reuse candidates for that problem. For each reuse candidate, the figure shows the initial and goal state specifications, the mapping between the candidate and the 4BS1 problem, and the plan kernel of the reuse candidate.[4] To simplify the drawings, the figure shows the validations of the plan kernels only by their supporting effects. Similarly, it also omits assertions of type $Block(?x)$ (as well as the validations of the type $\langle Block(?x), -, -, - \rangle$) from the specifications of the problems.

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 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. Since each reuse candidate has two goals and all of them match exactly two goals of the 4BS problem, it can be easily seen that $GSV(.)$ for each reuse candidate will be same as all the validations supported by its initial state (i.e., $A^s(n_I)$). Thus the check $v \in GSV(.)$ in the next two layers of the ranking procedure in Fig. 3 is trivially satisfied.

At the second layer, the f-feature $\langle On(K, J), -, -, - \rangle$ 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 $On(K, J) \notin I^n$. (This basically means that the top-level phantom goal of this reuse candidate has to be reestablished if we want to use it to solve P^n.) Similarly, the f-feature $\langle Pyramid(L), -, -, - \rangle$ of the plan kernel of the reuse candidate $\langle 3BS\text{-}Pyramid, [A \rightarrow L, B \rightarrow K, C \rightarrow J] \rangle$ is not preserved since $Pyramid(L) \notin I^n$. If we want to solve $4BS1$ by this reuse candidate, the subreduction dependent on this filter condition would have to be replaced. Further, the f-feature $\langle On(J, Table), -, -, - \rangle$[5] of the plan kernel of the reuse candidate $\langle 3BS, [A \rightarrow K, B \rightarrow J, C \rightarrow I] \rangle$ is not preserved since $On(J, Table) \notin I^n$. 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

[4] As an exercise, the reader may 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.

[5] We follow the convention of (Tate 1977) 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.

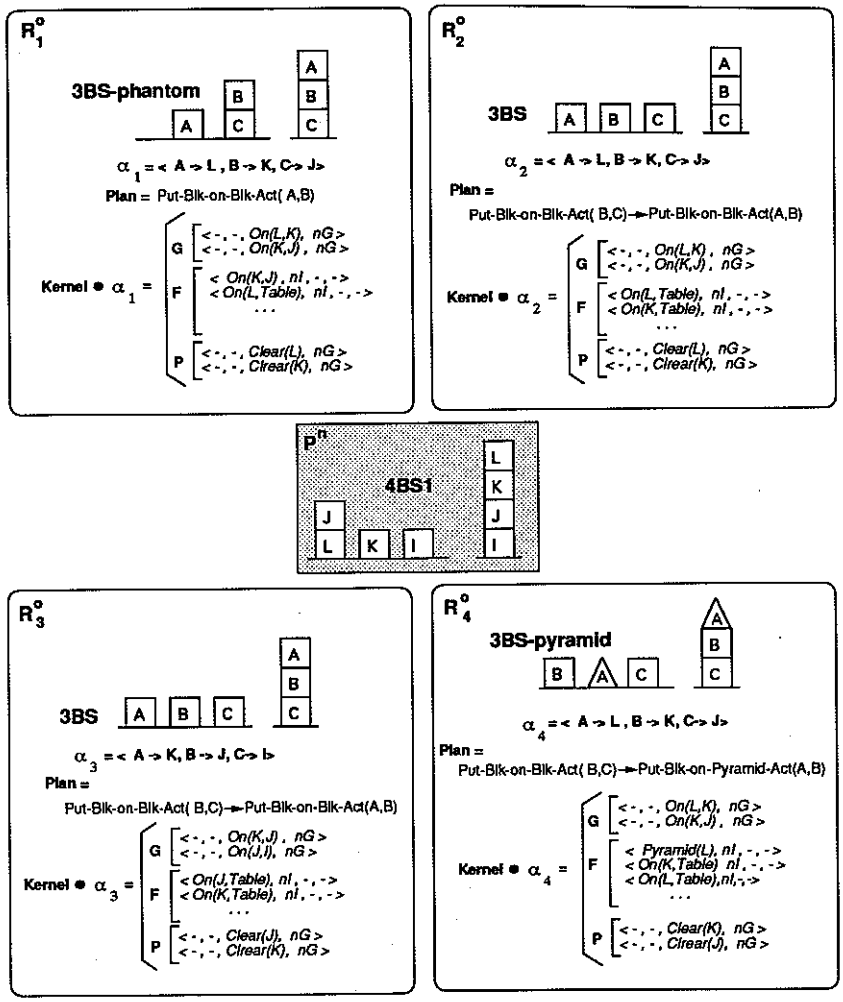


FIGURE 4. Example illustrating the retrieval strategy.

of the same (three blocks) plan. We have seen that the ordering prefers one of the mappings over the other.

4.4. Refinements to Plan-Kernel-Based Ordering

The *informedness* of the ordering procedure presented in Section 4.2 can be further improved by exploiting the hierarchical structure of the plan. In particular, we can distinguish among the validations by the reduction level at which it was first introduced into the HTN. For example, in Fig. 2, 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 schemata are written in such a way that the more important validations are established

at higher levels, while the establishment of less important validations is delegated to lower levels. Thus, the notion of the *level* of a validation can be used as a domain independent measure of saliency of the features supporting that validation.[6]

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 4.2 would now become $\sum_{\Delta} level(v)$, where

$$\Delta \equiv \{v : \langle E, n_I, C, n_d \rangle \mid v \in f\text{-}features(PK(\langle R^o, \alpha \rangle)) \wedge C \notin I^n\}$$

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 replacement of a larger subplan (the subplan rooted at n_3) than the latter (which only leads to the replacement of the subplan rooted at n_9; see Fig. 2).

4.5. Obtaining the Initial Set of Reuse Candidates

The strategy we discussed above shows how to pick a reuse candidate from an initial set of plausible ones. PRIAR uses a simple goal-based retrieval method to generate a set of plausible reuse candidates. The goals of the new problem P^n are variablized and are matched with the goals of the plans in the library. The matching is done by a modified unification algorithm that allows partial unification of formulas with differing number of conjuncts, returning an object mapping (binding list) and the number of goals unified for each partial unification. To avoid any unsupported generalization, we require that different variables be bound to different constants. Thus, the matching algorithm enforces an implicit unique noncodesignating variables constraint (that is, each variable is required to bind to a distinct constant) to ensure that only one-to-one mappings between the objects of P^n and R^o are generated. Currently, PRIAR includes only the candidates that satisfy the maximum number of goals in the set of plausible reuse candidates. These will then be ranked by the plan-kernel-based ordering discussed above.[7]

For example, when the 3BS problem of Fig. 2 is matched with the 4BS1 problem (in the same figure) the partial unification gives two reuse candidates based on this plan, $\langle 3BS, [A \rightarrow L, B \rightarrow K, C \rightarrow J] \rangle$ and $\langle 3BS, [A \rightarrow K, B \rightarrow J, C \rightarrow I] \rangle$. Each of these candidates matches two goals of 4BS1.

4.6. Discussion and Evaluation of Retrieval Control Strategy

The retrieval techniques described here have been implemented within the PRIAR reuse framework and have been used in solving a variety of problems in the blocks world. The evaluation trials consisted of solving blocks world problems by reusing a range of similar to dissimilar stored plans. In each trial, statistics were collected regarding the amount of effort involved in solving each problem from scratch versus solving it by modifying a given plan. Approximately 80 sets of trials were conducted over a variety of problem situations and problem sizes. A comprehensive listing of these statistics can be found in Kambhampati (1989).

[6]We assume that domain schemas having this type of abstraction property are 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.

[7]Notice that this step essentially carries out step 1 of the ordering procedure.

TABLE 1. Sample Statistics for PRIAR Reuse.

$\mathbf{R^o \rightarrow P^n}$	$\mathbf{P^n}$ from Scratch	Reuse $\mathbf{R^o}$	Savings (%)
3BS→4BS1	[**4.0***s*, 12*n*, 5*i*]	[**2.4***s*, 4*n*, 1*i*]	**39**
3BS→S5BS1	[**12.4***s*, 17*n*, 22*i*]	[**5.2***s*, 8*n*, 12*i*]	**58**
5BS→7BS1	[**38.6***s*, 24*n*, 13*i*]	[**11.1***s*, 12*n*, 19*i*]	**71**
4BS1→8BS1	[**79.3***s*, 28*n*, 14*i*]	[**22.2***s*, 18*n*, 18*i*]	**71**
5BS→8BS1	[**79.3***s*, 28*n*, 14*i*]	[**10.1***s*, 14*n*, 7*i*]	**87**
6BS→9BS1	[**184.6***s*, 32*n*, 17*i*]	[**18.1***s*, 17*n*, 17*i*]	**90**
10BS→9BS1	[**184.6***s*, 32*n*, 17*i*]	[**6.5***s*, 5*n*, 2*i*]	**96**
4BS→10BS1	[**401.5***s*, 36*n*, 19*i*]	[**52.9***s*, 30*n*, 33*i*]	**86**
8BS→10BS1	[**401.5***s*, 36*n*, 19*i*]	[**14.5***s*, 12*n*, 7*i*]	**96**
3BS→12BS1	[**1758.6***s*, 44*n*, 23*i*]	[**77.1***s*, 40*n*, 38*i*]	**95**
5BS→12BS1	[**1758.6***s*, 44*n*, 23*i*]	[**51.8***s*, 32*n*, 26*i*]	**97**
10BS→12BS1	[**1758.6***s*, 44*n*, 23*i*]	[**21.2***s*, 13*n*, 7*i*]	**98**

Table 1 presents representative statistics from the experiments. Each entry in the table shows the cost of solving a problem P^n from scratch, and solving it by reusing a given plan R^o. It compares planning times (measured in *cpu seconds*), the number of task reductions (denoted by xn), and the number of detected interactions (denoted by xi), for from-scratch planning and for planning with reuse. Since all the problems are drawn from the blocks world, where the goal assertions are all comprised of *clear* and *on* literals, there is a significant amount of surface level similarity between any given plan and R^o and a new problem P^n. This in turn gives rise to multiple reuse candidates in each case, making it critical to select among the reuse candidates. The retrieval strategy described in Fig. 3 is used to choose among these reuse candidates.[8]

The problems 3BS, 4BS, 6BS, 8BS, etc., are block stacking problems with three, four, six, eight, etc., blocks, respectively, on the table in the initial state, and stacked on top of each other in the final state. Problems 4BS1, 5BS1, 6BS1, etc., correspond to blocks world problems where all the blocks are in some arbitrary configuration in the initial state, and stacked in some order in the goal-state. In particular, the entry 3BS→4BS1 in Table 1 corresponds to the example discussed in the previous sections. A complete listing of the test problem specifications, as well as results of over 80 test runs, can be found in Kambhampati (1989). The last column of the table presents the computational savings gained through reuse as compared to from-scratch planning (as a percentage of from scratch planning time). Since the efficiency of reuse depends on the appropriateness of the chosen reuse candidate, these statistics provide a limited form of empirical support regarding the efficacy of the validation-structure-based retrieval.

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.

[8]The initial set of candidate mappings was generated by goal unification as described in Section 4.5.

Finally, the details of the three-layered ordering strategy in Fig. 3 are not meant to be too rigid. Indeed, it is possible to come up with alternatives that involve combining the cost metrics of the different layers in some weighted fashion before ranking the reuse candidates. Our main contribution to the retrieval problem is in elucidating a clear framework for using the causal dependency structure of stored plans to judge the utility of reusing them to solve new problems. We have shown that within this framework it is possible to formulate a variety of efficient and domain-independent similarity metrics with varying cost/benefit trade-offs.

In this section, we have discussed the problem of retrieval without addressing the problem of organization of stored plans in the library. In general, however, the efficiency of a retrieval strategy cannot be measured in isolation, as it is fundamentally influenced by the way plans are stored in the library. For example, since the stored plans may have significant amount of structural overlap, storing them in an unorganized fashion could lead to a considerable amount of redundant matching during the retrieval phase. To avoid this, the organization strategies should be able to group "similar" plans together. A first step is to variablize the plans before storing them (so that we do not store two instances of the same plan); explanation-based generalization techniques (*cf* Mitchell, Keller, and Kedar-Cabelli 1986) are typically used for this purpose. The PRIAR modification strategy, based as it is on a systematic characterization of the explanation of correctness of the plan, is amenable to integration with explanation-based generalization techniques. In particular, we have recently developed a family of provably sound algorithms for generalizing partially ordered plans based on the validation structure representations discussed in Section 4.1 (Kambhampati and Kedar 1992). These techniques are currently being integrated into the PRIAR reuse framework to improve the storage and retrieval costs associated with plan reuse (Kambhampati 1992). Further reduction in redundancy requires utilization of hierarchical memory organization frameworks (such as discrimination networks) to group overlapping plans together. The natural abstraction inherent in the plans generated in the hierarchical planning regime may provide support for this type of abstraction, although this remains to be investigated (Kambhampati 1992).

5. CONTROLLING REFITTING

Once an appropriate reuse candidate has been chosen by the retrieval procedure, PRIAR typically needs to modify it to solve the new problem. As we have mentioned earlier, this process consists of two stages. The first is the *annotation verification* stage, which locates the places where the reuse candidate needs to be modified and suggests modifications to be carried out. The outcome of this stage is typically a partially reduced plan for solving the new problem. The *refitting* stage takes this partially reduced plan and completes the reduction process. Consider, once again, the reuse example in Fig. 4, which involves solving the 4BS1 problem by reusing a set of existing plans. In Section 4.3, we have seen that the retrieval control strategy recommends the 3BS plan, along with the mapping $\alpha = [A \rightarrow L, B \rightarrow K, C \rightarrow J]$, as the preferred reuse candidate in this example. This becomes the input to the annotation verification process. Figure 5 shows the partially reduced plan resulting from the annotation verification process for this example (Kambhampati and Hendler 1992). It contains all the applicable parts of the 3BS reuse candidate plus high-level refit tasks added to handle any unsatisfied goals. The objective of the refitting stage is then to accept this partially reduced plan and complete the reduction of the refit tasks. To ensure overall savings from plan reuse, this reduction process should be localized so as to cause least disturbance to the rest of the applicable parts of the annotation verified plan (denoted by R^a). This, in short, is the problem of refitting control.

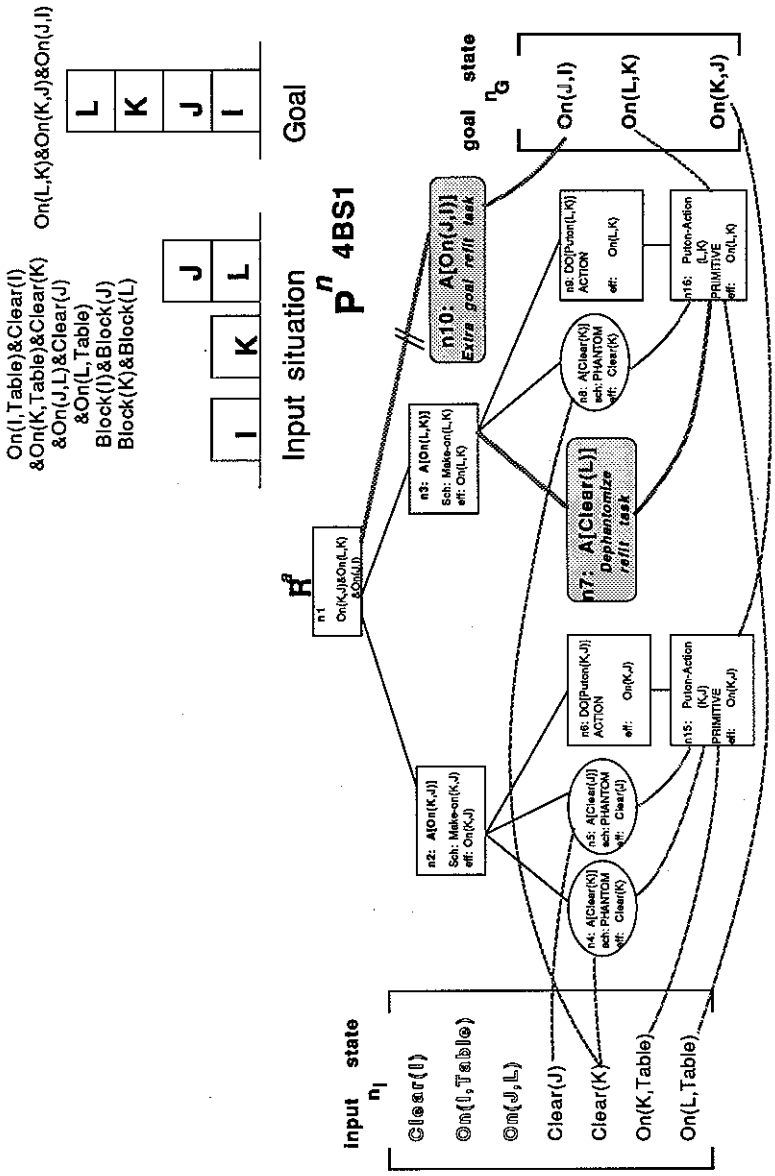


FIGURE 5. Annotation verified plan for 3BS→4BS1 problem.

To ensure such localization of refitting, PRIAR provides a minimum-conflict-type heuristic control strategy for reducing refit tasks. In particular, for each refit task, this strategy chooses the schema instance causing the least number of harmful interactions with the rest of the plan. The interactions are measured in terms of the number and type of inconsistencies in the validation structure of the plan that are caused by the various reduction choices. In the following, we define the notion of the *task kernel* of a refit task and develop a ranking procedure based on it to facilitate conservative control of refitting.

5.1. Representation of the Task Kernel

The task kernel $K(n)$ of a refit task n is intended to encapsulate the potential interactions that a reduction choice at this node will have with the validation structure of the rest of the *annotation-verified* plan R^a. The task kernel of a refit task is intended to encapsulate the information required to measure the amount of disturbance that a reduction choice can cause to the validation structure of the rest of the plan. A new reduction choice at a refit task may neglect establishing an expected validation; it may violate an existing validation of R^a; or it may not consume a validation which it utilized previously. Accordingly, the task kernel has three layers, each corresponding to a way in which the reduction at a refit task can interact with the validation structure of R^a. Formally, we define $K(n)$ as a 3-tuple

$$K(n) = \langle \textit{e-conditions, p-conditions, e-preconditions} \rangle$$

where[9]

$$\begin{aligned} \textit{e-conditions}(K(n)) &= \textit{e-conditions}(n) \\ \textit{p-conditions}(K(n)) &= \textit{p-conditions}(n) \\ \textit{e-preconditions}(K(n)) &= \textit{e-preconditions}(n) \end{aligned}$$

With this formulation, $K(n)$ contains *all* the validations of R^a that might possibly get affected by any reduction done at n.

Consider the refit task *Achieve*[*Clear*(L)] at node n_7 in the annotation-verified plan (R^a) for the 3BS→4BS1 problem we discussed above. The task kernel of n_7 will be[10]

$$K(n_7) = \left[\begin{array}{ll} \textbf{e-conditions}: & \langle \textit{Clear}(L), n_7, \textit{Clear}(L), n_{16} \rangle \\ \textbf{p-conditions}: & \left\{ \begin{array}{l} \langle \textit{Clear}(K), n_1, \textit{Clear}(K), n_8 \rangle \\ \langle \textit{On}(J, I), n_{10}, \textit{On}(J, I), n_G \rangle \\ \langle \textit{On}(K, J), n_{15}, \textit{On}(K, J), n_G \rangle \\ \langle \textit{On}(L, \textit{Table}), n_1, \textit{On}(L, ?x), n_{16} \rangle \end{array} \right. \end{array} \right]$$

As a second example, consider the following hypothetical case from the same example. Suppose that there is a *replace-reduction* refit task at n_3. This might be the case, for example, if in P^n, the object L is a *pyramid* rather than a *block*. This would cause the failure of a filter

[9] We make this differentiation between e-conditions, p-conditions, and e-preconditions of $K(n)$ and those of n because it is possible to give alternative formulations to $K(n)$ (as we do in Section 5.3), where they will not be the same.

[10] The validation supporting the goal $On(J, I)$ occurs in the task kernel because $A[On(J, I)]$ is not yet ordered with respect to the plan, and thus it is a p-condition of $R(n_7)$.

condition validation, and necessitate re-reduction of n_3. The task kernel for this refit task would be

$$K(n_3) = \left[\begin{array}{l} \textbf{e-conditions} : \langle On(L, K), n_3, On(L, K), n_G\rangle \\ \textbf{p-conditions} : \left\{\begin{array}{l}\langle On(K, J), n_{15}, On(K, J), n_G\rangle \\ \langle On(J, I), n_{10}, On(J, I), n_G\rangle\end{array}\right. \\ \textbf{e-preconditions} : \left\{\begin{array}{l}\langle Clear(K), n_I, Clear(K), n_3\rangle \\ \langle Clear(L), n_I, Clear(L), n_3\rangle \\ \langle Block(K), n_I, Block(K), n_3\rangle \\ \langle On(L, Table), n_I, On(L, ?x), n_3\rangle\end{array}\right. \end{array}\right]$$

5.2. Ordering Refitting Choices Using Task Kernels

The main motivation for using task kernels to order reduction choices at a refit task n is to prefer choices that preserve as many of the task kernel validations as possible. This strategy has the following desirable properties:

1. By preserving the *e*-conditions of the task kernel of n, it obviates the need to reestablish the validations that were previously supported by $R(n)$ in R^i. This means that there will not be any need to add additional refit tasks for achieving the unsatisfied *e*-conditions of n during refitting.
2. By preserving the *p*-conditions of the task kernel of n, it minimizes the harmful interactions that refitting at n will have with the validations of other parts of R^a, thus preserving as much of the applicable parts of the old plan as possible and reducing the cost of refitting.
3. By preserving the *e*-preconditions of the task kernel of n, it utilizes the already existing parts of R^a that establish those validations and thereby reduces the cost of reduction of n (as the planner would not have to newly establish these preconditions for the new reduction below n).

Figure 6 shows the three-layered procedure that is used for ranking the schema instance choices for reducing a refit task. Notice that the definition of preservation of *p*-conditions in *step 2* is in terms of nonviolation. This is appropriate since the objective is to ensure that the *p*-condition validations are not violated by the reduction at n, rather than to establish them. Furthermore, notice that a *p*-condition is considered preserved only when the effects of the task reduction schema, in conjunction with the domain axioms, do not imply violation of the *p*-condition (see below). This allows the ranking process to anticipate and avoid interactions that would normally only be discovered at a later stage in the planning (see below). The best-ranked schema instances at the end of this three-layered ordering procedure are returned as the schema instances preferred by the task-kernel-based ordering.

These implicit levels of importance accorded to the various components of the task kernel reflect the effect of violation of those validations upon the overall refitting cost. In Section 5.3 we will discuss some refinements to task-kernel-based ordering which will allow us to further differentiate among validations on the same layer.

Example. Consider, once again, the example of reducing the refit task n_7 in the annotation-verified plan for 3BS→4BS1 problem shown in Fig. 5. The default selection strategy finds

Given. A refit task $n \in R^a$, and a set of schema instances $\{S_i\}$ capable of reducing n.

Step 1. Order the refitting choices (schema instances) according to the number of task kernel e-conditions they preserve. A schema instance S_i preserves an e-condition of the task kernel if its effects can supply that e-condition. Thus, the merit function used to order schema instances at this level is given by

$$Merit_1(S_i) = |\{v \mid v : \langle E, n_s, C, n_d\rangle \in e\text{-}conditions(K(n)) \wedge effects(S_i) \vdash C\}|$$

Step 2. In case of a tie, pick the set of schema instances that are ranked best by step 1 and rank them further according to the number of task kernel p-conditions they preserve. A task kernel p-condition is considered preserved by a schema instance S_i, if the effects of S_i do not violate that p-condition. Thus

$$Merit_2(S_i) = |\{v \mid v : \langle E, n_s, C, n_d\rangle \in p\text{-}conditions(K(n)) \wedge effects(S_i) \overset{d}{\nvdash} \neg E\}|$$

Step 3. In case of a tie, pick the set of schema instances that are ranked best by step 2 and further rank them according to the number of task kernel e-preconditions that they preserve. An e-precondition is considered preserved by a schema instance S_i, if S_i has an applicability condition that can be supported by the e-precondition. Thus,

$$Merit_3(S_i) = \left|\left\{v \mid \begin{pmatrix} v : \langle E, n_s, C, n_d\rangle \in e\text{-}preconditions(K(n)) \wedge \\ \exists C' \in applicability\text{-}conditions(S_i)\ s.t.\ E \vdash C' \end{pmatrix}\right\}\right|$$

FIGURE 6. Task-kernel-based ordering.

that this refit task can be reduced by the following three (blocks world) schema instances:

$\boldsymbol{A}$: *MakeClear-Table*(L, J)	**eff** : Clear(L), On(J,*Table*)
$\boldsymbol{B}$: *MakeClear-Block*(L, J, K)	**eff** : Clear(L), On(J,K), ¬Clear(K)
$\boldsymbol{C}$: *MakeClear-Block*(L, J, I)	**eff** : Clear (L), On(J ,I), ¬Clear(I)

where the schema *MakeClear-Block*(?X, ?Y, ?Z) clears ?X by putting ?Y (which is on top of ?X) on top of ?Z, and the schema *MakeClear-Table*(?X, ?Y) clears ?X by putting ?Y on *Table* (which is considered always clear).

When these schema instances are ordered using $K(n_7)$ given in the previous section, we find that the task kernel e-condition $\langle Clear(L), n_7, Clear(L), n_{15}\rangle$ is preserved by all three choices, as all of them have an effect $Clear(L)$. So, they all survive to the next layer ordering with respect to task kernel p-conditions. At this stage, choice B violates the first three p-conditions since

$$On(J, K) \overset{d}{\vdash} \neg Clear(K) \wedge \neg On(J, I) \wedge \neg On(K, J)$$

Choice A preserves $\langle Clear(K), n_1, Clear(K), n_8\rangle$, but violates one p-condition as

$$On(J, Table) \overset{d}{\vdash} \neg On(J, I)$$

The p-condition $\langle On(L, Table), n_1, On(L, ?x), n_{16}\rangle$ is preserved by both A and B since none of their effects negate $On(L, Table)$. Finally, choice $\boldsymbol{C}$ preserves all the p-conditions.

Thus, the task-kernel-based ordering clearly prefers choice ***C***, *Make-Clear*(L, J, I). This will be sent to the planner as the schema instance with which node n_7 should be reduced. Notice that this choice would in fact minimize the refitting cost as it causes no harmful interactions with the validations of R^a, and has the serendipitous effect of achieving the extra goal, $On(J, I)$.

Figure 7 shows the result of reducing the annotation-verified plan in Fig. 5 with the help of this refitting control strategy. The top part of the figure shows the hierarchical structure of the task reductions underlying the development of the plan (abstract tasks are shown on the left, with their reductions shown to the right). The bottom part shows the chronological partial ordering relations among the leaf nodes of the HTN. The black nodes correspond to the parts of the interpreted plan that were salvaged by the reuse process, while the white nodes represent the refit tasks added during the annotation-verification process and their subsequent reductions. In this example, the choice between ***B*** and ***C*** could have been made through a strategy of delayed binding of objects (e.g., the *merge objects* critic in NOAH (Sacerdoti 1977)). However, such a strategy would not have been able to deal with ***A***, which is an instance of a different schema. In general, controlling the choice among alternative schema instances involves more than delayed binding of objects, as there may be different schemata that can reduce the same refitting task, with significant differences among their effects and preconditions. Choices made with the help of task-kernel-based ordering will be able to effectively control refitting in such cases.[11]

The dominant cost of task-kernel-based ordering comes from the cost of checking if a given validation will be preserved or violated by the effects or applicability conditions of a schema instance S_i. In particular, this may involve deciding if the effects of S_i are consistent with the condition being supported by a given validation. While consistency analysis is undecidable in the general case, it can be made efficient by restricting the generality of the domain axioms (e.g., Drummond and Currie 1989).

5.3. Refinements to Task-Kernel-Based Ordering

The heuristic ordering procedure described above can be made more *informed* and *efficient* in a variety of ways by exploiting the hierarchical structure of the plan. Below we discuss some ways of doing this based on the PRIAR validation structure. The ordering procedure can be made more informed by allowing it to distinguish between the validations belonging to the same layer of the task kernel. It can be made more efficient by redefining the task kernel in such a way as to include only those validations that might potentially be violated by the schema reduction choices.

Improving the Informedness of the Ordering.

(1) Judging the Importance of Validations by "Level." Once again, by utilizing the level of the validations, the validations of the individual layers of the task kernel can be differentiated further, based on the difficulty of reestablishing them in the event they are not preserved by the schema instance chosen to reduce the refit task. When the validations are weighted in this fashion during the task-kernel-based ordering, the merit functions of the ordering procedure will then be computing the weighted sum of the validations preserved at each layer. For

[11] In fact, it can be argued that previous nonlinear planners avoided deliberating on the schema selection by relying on delayed commitment strategies and nondeterministic choice strategies. Control strategies such as *task-kernel-based* ordering help the planner to properly deliberate on schema selection.

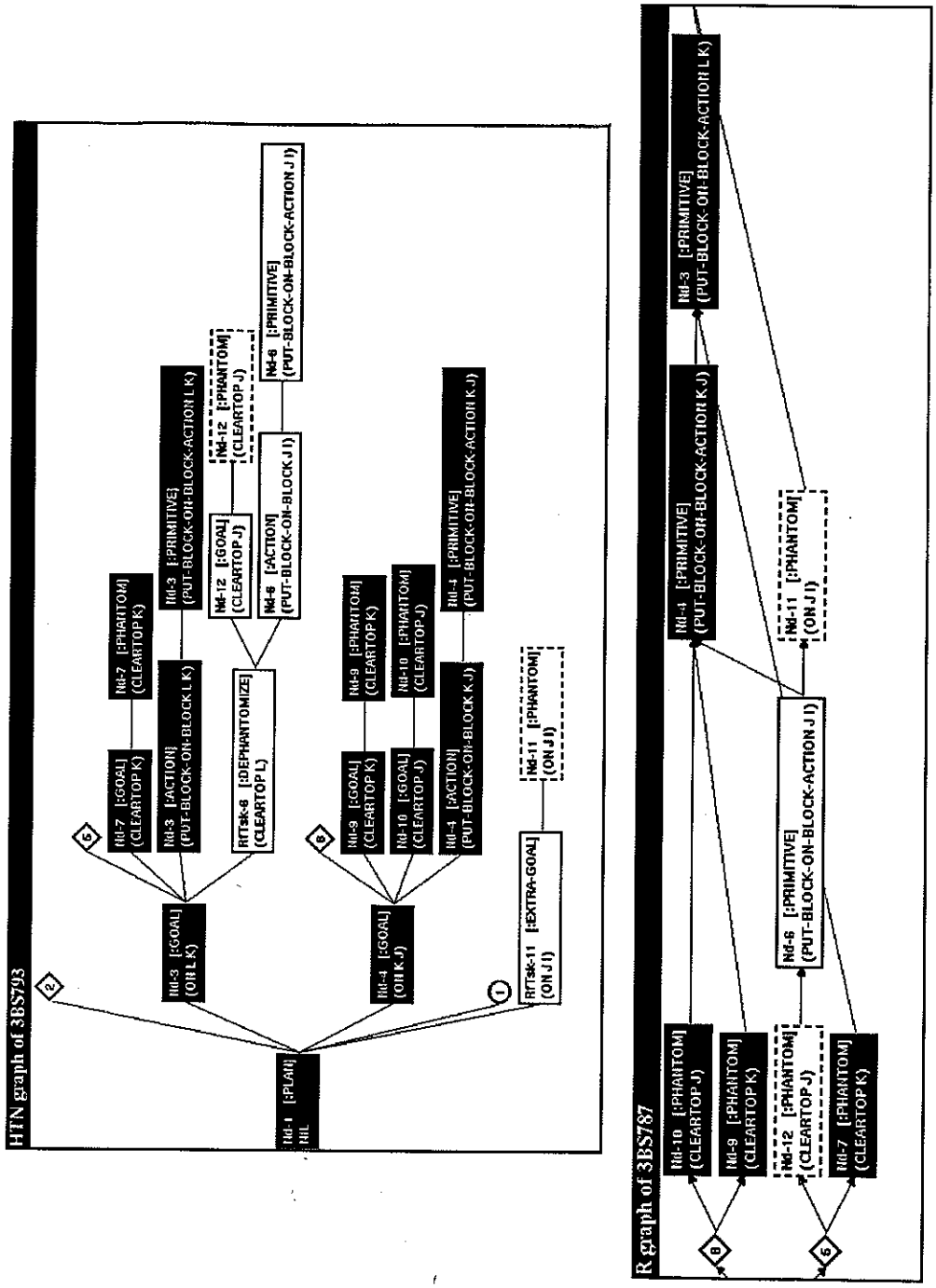


FIGURE 7. The plan produced by PRIAR for 3BS→4BS1.

example, the merit function for the second layer ordering will now be

$$Merit^{2'}(S_i) = \sum_{\Delta} level(v)$$

where

$$\Delta \equiv \{v \mid v : \langle E, n_s, C, n_d \rangle \in p\text{-}conditions(K(n)) \wedge effects(S_i) \overset{d}{\nvdash} \neg E\}$$

The levels of validations can be precomputed efficiently at annotation time in at most $O(d|\mathbf{V}|)$ for all the validations, if done at the time of annotation,[12] where d is the depth of the HTN (typically $d \ll N_P$).

In the three-block stacking plan example, the validation $\langle On(K, J), n_{15}, On(K, J), n_G \rangle$ is of higher level (level 0) than the validation $\langle Clear(K), n_1, Clear(K), n_8 \rangle$ (level 2). Thus, while ordering the schema instance choices for reduction n_7, a schema instance preserving only the former validation will be preferred over a schema instance preserving only the latter.

(2) Giving Importance to Filter Condition Validations. A heuristic that is useful to follow regarding the p-conditions of the task kernel is to give higher importance to the p-conditions of $K(n)$ supporting filter conditions. The rationale for this heuristic is that the violation of the former would necessitate *replace-reduction* refit tasks, thereby significantly increasing the amount of refitting required to completely reduce R^a. One way of using this heuristic is to split the second layer of the ordering into two sublayers. The first sublayer ranks the schema instances by the number of filter condition validations they preserve. The second sublayer then ranks the best choices (in the ranking of the first sublayer) by the rest of the p-conditions.

Improving the Efficiency of the Ordering. As mentioned earlier, the main cost of task-kernel-based ordering stems from the checks to ascertain whether the validations of the task kernel are preserved by the effects and applicability conditions of each of the schema instance choices. One way of reducing this cost is to reduce the size of the task kernel. This can be achieved by ensuring that the task kernel does not contain any irrelevant validations whose violation or preservation cannot reasonably be established from the effects and conditions of the refitting choices, or whose violation can be easily repaired by introducing additional ordering relations. In this section, we will discuss two ways of doing this.

(1) Eliminating Parallel p-conditions from Task Kernels. The ordering strategy presented in Section 5.2 assumes that the violation of any p-condition of the task kernel will necessitate costly backtracking or additional refitting. This is not always the case, since the violation of some of the p-conditions can be handled by the planner through the introduction of additional temporal orderings between tasks of the plan. If the cost of checking for violation of all p-conditions of $K(n)$ is a concern during the ordering (as would be the case, for example, when there is a significant amount of choice in the domain), then it would make sense to have the ordering consider only those p-conditions whose violation cannot possibly be resolved by imposing additional temporal orderings on the plan. In particular, a p-condition $v : \langle E, n_s, C, n_d \rangle$ of a node n cannot be reordered with respect to n only when $\Box(n_s < n < n_d)$. We can reduce the size of the task kernel by eliminating p-conditions that do not satisfy this.

[12] The levels of the validations can also be easily maintained dynamically during planning. In fact, the levels of validations can be used during interaction resolution to decide which of a set of conflicting validations should be preserved.

FIGURE 7. The plan produced by PRIAR for 3BS→4BS1.

(2) Eliminating Lower-Level Validations from Task Kernels. One way of improving the efficiency of the ordering is to reduce the size of the task kernels (which reduces the number of validations that need to be checked). In hierarchical planning, the schema instances often reduce a nonprimitive node to only the next lower level of abstraction. Thus, at the time of reduction of a refit task n, it is difficult to predict the complete subreduction below n. In general, without this information, it does not make sense to try to check for the preservation of all the e-conditions, p-conditions, and e-preconditions of n, since they may be pertaining to the nodes that are at a lower level of the abstraction. In other words, the task kernel validations need be at the same level of detail as the applicability conditions and effects of the schema instances that comprise the reduction choices. We can use this to provide an alternative formulation for the task kernel that has a lower match cost: *The e-preconditions, e-conditions, and p-conditions of n, which were originally introduced into R^o at more than one level of abstraction below that of n need not be included in the task kernel of n.* With this alternative formulation, the task kernel of the hypothetical *replace-reduction* refit task at n_3 would be

$$K'(n_3) = \left[\begin{array}{l} \textbf{e-conditions} : \langle On(L, K), n_3, On(L, K), n_G\rangle \\ \textbf{p-conditions} : \left\{\begin{array}{l}\langle On(K, J), n_{15}, On(K, J), n_G\rangle \\ \langle On(J, I), n_{10}, On(J, I), n_G\rangle\end{array}\right. \\ \textbf{e-preconditions} : \left\{\begin{array}{l}\langle Clear(K), n_I, Clear(K), n_3\rangle \\ \langle Clear(L), n_I, Clear(L), n_3\rangle \\ \langle Block(K), n_I, Block(K), n_3\rangle\end{array}\right. \end{array}\right]$$

In other words, the e-preconditions of $K'(n_3)$ will not include the lower-level validation $\langle On(L, Table), n_I, On(L, ?x), n_3\rangle$.

5.4. Discussion and Evaluation of Refitting Control Strategy

The task-kernel-based ordering is a cheap informed backtracking control strategy, where the heuristic attempts to predict the number of *interactions* that will eventually have to be resolved if a given refit task is reduced with the help of a particular schema instance. Using this, the planner can select schema instances that are likely to cause the least number of interactions[13] with the applicable parts of the retrieved plan. This is a very effective technique for controlling refitting as interaction detection and resolution increase the cost of nonlinear planning exponentially[14] and delocalize the refitting process by affecting the validations of the applicable parts of the plan. From a search reduction point of view, the refitting control strategy is best used as a *pruning* strategy. Unfortunately, however, using it as a pruning strategy can in general lead to loss of completeness (cf. Waldinger 1977). To preserve completeness, we use it as a nonpreemptive selection strategy, in that the unchosen reduction choices are not discarded but stored as choice points for later backtracking. This, however, makes it difficult to formally quantify the search reduction offered by the control strategy.

To understand the impact of the refitting control strategy on PRIAR's reuse performance,

[13] *Interaction* here signifies the amount of disturbance caused to the validation structure of the plan by a particular refitting choice. Resolving such interaction may necessitate reordering, re-reduction (backtracking), from-scratch achievement, or pruning.

[14] If the number of interactions caused by a refitting choice s is given by $I(s)$, and the average number of ways of resolving an interaction is k, then it can be shown (see Yang 1989) that the worst-case search space for resolving all the conflicts has a size of $O(k^{I(s)})$.

TABLE 2. Sample Statistics Regarding Effectiveness of PRIAR Refitting Control.

$R^o \rightarrow P^n$	P^n from Scratch	Reuse R^o without Refit Cntl.	(%)	Reuse R^o with Refit Cntl.	(%)
3BS→4BS1	[**4.0***s*, 12*n*, 5*i*]	[**3.3***s*, 7*n*, 9*i*]	**16**	[**2.4***s*, 4*n*, 1*i*]	**39**
3BS→5BS1	[**8.4***s*, 16*n*, 8*i*]	[**7.1***s*, 11*n*, 12*i*]	**15**	[**4.3***s*, 8*n*, 3*i*]	**49**
4BS→5BS1	[**8.4***s*, 16*n*, 8*i*]	[**5.3***s*, 8*n*, 11*i*]	**36**	[**3.2***s*, 5*n*, 2*i*]	**62**
5BS→7BS1	[**38.6***s*, 24*n*, 13*i*]	[**15.4***s*, 15*n*, 28*i*]	**60**	[**11.1***s*, 12*n*, 19*i*]	**71**
4BS→8BS1	[**79.3***s*, 28*n*, 14*i*]	[**24.9***s*, 22*n*, 26*i*]	**68**	[**15.4***s*, 19*n*, 17*i*]	**80**
7BS→9BS1	[**184.6***s*, 32*n*, 17*i*]	[**18.6***s*, 14*n*, 16*i*]	**89**	[**11.4***s*, 11*n*, 6*i*]	**93**
8BS→10BS1	[**401.5***s*, 36*n*, 19*i*]	[**22.0***s*, 15*n*, 16*i*]	**94**	[**14.5***s*, 12*n*, 7*i*]	**96**
7BS→10BS1	[**401.5***s*, 36*n*, 19*i*]	[**22.4***s*, 19*n*, 17*i*]	**94**	[**23.4***s*, 19*n*, 17*i*]	**94**
4BS→6BS1	[**17.7***s*, 20*n*, 10*i*]	[**11.6***s*, 14*n*, 28*i*]	**34**	[**11.6***s*, 14*n*, 28*i*]	**34**
5BS→12BS1	[**1758.6***s*, 44*n*, 23*i*]	[**73.7***s*, 35*n*, 35*i*]	**95**	[**51.8***s*, 32*n*, 26*i*]	**97**

we performed an *ablation study* by measuring the reuse performance with and without the refitting control over a variety of blocks world modification problems.[15] Table 2 shows some representative results from these experiments. It lists the planning times, the number of task reductions, and the number of detected interactions when a problem P^n is solved by PRIAR from scratch, by reusing a given plan R^o without refitting control, and by reusing R^o and controlling the refitting by the task-kernel-based ordering. It shows that the refitting control leads to significant improvements in the performance. There is a reduction in planning times, in the number of detected interactions, and in the number of task reductions during refitting.

Figure 8 compares the variation of PRIAR's performance when a particular blocks world problem, 7BS1, is solved by reusing existing plans *with* and *without* refitting control strategy. We noticed from these experiments that refitting control improves reuse performance whenever there is a choice to reduce refit tasks (by reducing the number of interactions that need to be resolved).

A word of explanation is in order regarding the statistics shown in Table 2. When the planner is run without the refitting control strategy, if it is allowed to choose randomly among the reduction choices, it often gets into extensive backtracking and fails to find a plan within a reasonable time limit. To limit backtracking in the runs without refitting control, we employed a domain-dependent heuristic and always clear a block by putting the block on top of it on the table (rather than transfer it to the top of another block). Further, in the blocks world, the only refit tasks for which there exists any choice of task reduction schemata are of the type $Achieve[Clear(?x)]$. Thus, when reusing R^o to solve P^n, if the HTN after the annotation verification, R^a, does not contain any refit tasks of this type,[16] we cannot expect any improvement from PRIAR's refitting control strategy. The last two entries of the table reflect this. We expect that in domains where there is more choice for reducing individual refit tasks, the effect of the refitting control strategy will be more significant. This conjecture is supported by our experience with manufacturing planning domain (see Section 6), where there is a significant amount of task-reduction choice. More interestingly, as we shall see in

[15] PRIAR has also been tested in a manufacturing planning domain where the objective is to construct a partially ordered sequence of machining operations for manufacturing simple machinable mechanical parts; see Section 6 for details.

[16] Whether or not R^a contains refit tasks of a particular form depends on the relation between R^o and P^n as well as on the particular mapping, α, between R^o and P^n that is chosen by the retrieval procedure.

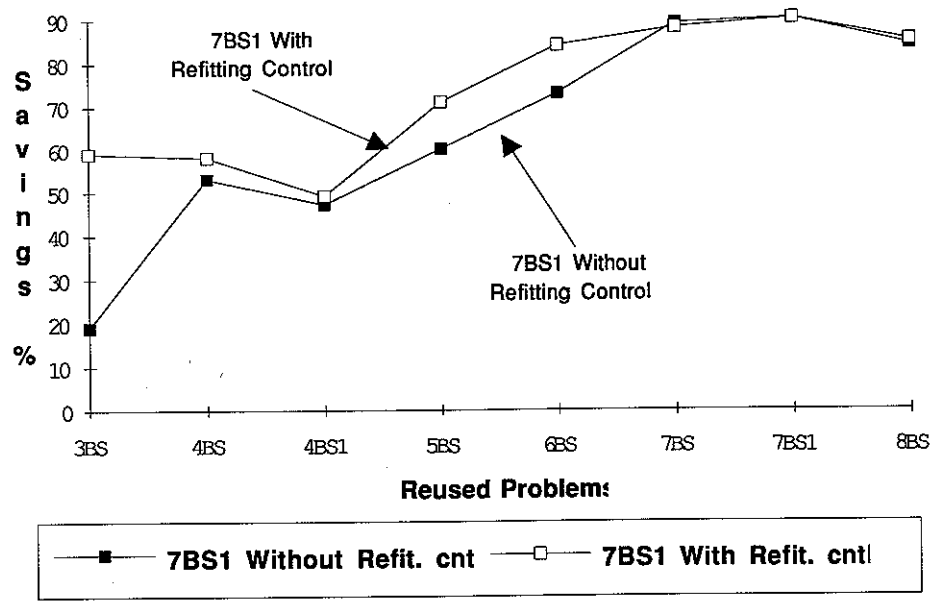


FIGURE 8. Effect of PRIAR refitting control strategy on a blocks world problem.

Section 6, the minconflict refitting control strategy is of added utility when the planner is part of a larger environment and the revisions to the plan affect other modules in the environment.

Because of its "status quo" nature, the minimal conflicts heuristic biases the search to find a solution that is nearest to the "seed guess." This may not necessarily be the best solution according to some global "objective function." In other words, it is a *satisficing* control strategy rather than an *optimizing* one (cf. Simon and Kadane 1975). Because of the status quo nature of the heuristic, we think this will be more suitable for controlling refitting than for planning from scratch. This is also in consonance with recent work on application of minconflict heuristics to constraint satisfaction problems (Minton *et al.* 1990; Zweben *et al.* 1990), which shows that the efficacy of these heuristics depends to a large extent on the nearness of the initial seed guess to the final solution.

Finally, as a minimum-conflict heuristic (Section 2), there are two important aspects of the task-kernel-based refitting control strategy that merit attention. These involve the way conflicts are computed, and the way they are classified.

Computing Conflicts. Given an expressive enough representation language, finding conflicts (interactions) and checking for consistency in a partially ordered plan is equivalent to theorem proving over the effects of partially ordered events. Most traditional planners avoid this complexity by employing the "STRIPS assumption" (Waldinger 1977). The STRIPS assumption shortcuts the interaction detection problem by assuming that the only conflicts to be considered are those arising from the explicit negations between represented conditions and effects of the actions in a plan. While this reduces the cost of interaction detection during planning, it often delays the detection of many interactions. Thus, using this method to compute the number of interactions is not going to give an accurate estimate of the conflicts generated by a choice. For example, most blocks world planners that use STRIPS assump-

tion coupled with the TWEAK truth criterion (Chapman 1987) fail to detect that the effect $On(A, B)$ of an action is inconsistent with a protected condition $On(D, B)$ (even though $On(A, B) \not\vdash^{d} \neg On(D, B)$). These planners eventually produce correct plans by ensuring that the missed conflicts are detected at a more detailed level (e.g., the conflict between $On(A, B)$ and $On(D, B)$ will be detected when it is discovered that $Clear(B)$ is a precondition of the operator for achieving $On(D, B)$, and it is being denied by the action that achieves $On(A, B)$.)

As we have seen, PRIAR refitting control strategy deals with this trade-off by using "domain axioms" for computing the minimum-conflicts heuristic during schema selection, while retaining the STRIPS assumption during planning. The underlying rationale is that there is some computational advantage to be gained in making choices by anticipating the conflicts that the planner will realize only later on (as long as the consistency analysis itself is not too costly).

Classifying interactions. Typical realizations of a minimum conflict heuristics in constraint satisfaction problems (e.g., Minton *et al.* 1990; Zweben *et al.* 1990) tend to consider all types of conflicts to be of uniform cost. However, such a model does not adapt well to controlling refitting. This is because the cost of interaction resolution varies widely depending on the type of the interaction—certain interactions can be resolved by simple parameter rebinding, while others might necessitate extensive backtracking. Thus, some understanding of the "cost of resolution" seems essential for proper application of the minimum conflicts heuristic to planning. One way of doing this is to acquire and utilize some domain-dependent information about the significance of various interactions. Another approach, employed by PRIAR, is to use the validation-structure-based representation of plan dependency structure to characterize the interactions. PRIAR's refitting control strategy estimates the relative cost of interactions along domain-independent dimensions such as the *type* of the failing validation, and the *level* of the validation.

6. EVALUATION IN MANUFACTURING PLANNING DOMAIN

In previous sections, we characterized the utility of the retrieval and refitting techniques by discussing their effect in improving the efficiency of plan reuse in the blocks world. As we mentioned however, the techniques themselves are domain independent and are of broad applicability. Indeed, the PRIAR plan reuse framework has been applied to a manufacturing planning domain to improve efficiency of planning. Although a comprehensive discussion of our work in the manufacturing planning domain is beyond the scope of the paper (see Kambhampati *et al.* 1993), in this section we will briefly discuss the effect of refitting and retrieval techniques on planning in that domain.

The task addressed in manufacturing planning domain is that of generating and maintaining process plans for machined parts. The planner is part of a prototype environment for supporting concurrent design, called NEXT-CUT (Cutkosky and Tenenbaum 1990). NEXT-CUT can be seen as a CAD-tool for mechanical design, which aims to provide rapid feedback to the designer about the manufacturability of the evolving design. To support this goal, NEXT-CUT performs planning and analysis step by step as the designer constructs and modifies the design. The main objective of the process planner in the NEXT-CUT environment is thus to provide feedback regarding the manufacturability of the evolving part[17]. The input to

[17] Those familiar with process planning literature may note that this is somewhat different from the objectives of production-

the planner consists of the description of a part in terms of features, dimensions, tolerances, and corresponding geometric models. The process plan includes a sequence of "setups" (particular orientations in which the workpiece should be restrained using fixturing devices such as *vises* and *strap-clamps*), the set of machining operations (such as *drilling*, *milling*, *boring*) that should be carried out during each setup, and the tools (such as *0.25in-dia-twist-drill*) to be used during each machining operation. The PRIAR modification framework was used to automate the generation and incremental revision of machining plans, while separate domain dependent modules deal with the geometric and fixture planning portions of the overall process plan.[18]

Since we are modifying the current plan to accommodate user-initiated changes in design, the problem of retrieval is short-circuited. However, the efficiency of refitting was still important, and the refitting control techniques described in this paper prove to be particularly useful. In contrast to the blocks world, there is a larger choice for task reduction in process planning. For example, a task such as *make-hole* can be achieved in a variety of ways, including *drilling, milling, honing, boring*, etc. Each of these choices interacts in different ways with the commitments in the existing plan, making refitting control strategy quite valuable.

Apart from improving the refitting efficiency of the machining planner, these techniques also allowed a more efficient interaction between the machining planner and the other modules in the NEXT-CUT environment. To understand this, note that whenever the machining plan is modified to accommodate the user-initiated changes, these modifications in turn necessitate changes to the fixturing plan (which in this domain turns out to be a computation intensive operation). Figure 9 shows a plan revision scenario from the process planning domain which illustrates this point.[19] In this case, after initial process planning for the part shown in Window I is completed, the designer makes some changes to its specification—including addition of a new hole, change of diameter of one of the holes (shown in bold outline), and the positional tolerance of another. The response of the machining planner (implemented by PRIAR) to these changes is shown in Window III. Once again, the black nodes represent parts of the plan from the previous iteration, and the white nodes represent the results of refitting to accommodate the user-initiated changes. Windows V and VI show the fixturing plan corresponding to the new machining plan, with the black nodes representing the parts of the fixture plan that are salvaged from the original plan, while the white ones represent the results of new analysis. Notice that the only fixturing setup that is completely new is the one corresponding to *hole-5* shown in Window VI.

The machining planner's ability to accommodate the changes in the specification in a conservative fashion thus leads to a reduction in the amount of refixturing needed to accommodate the user-specified changes.[20] The process planning domain also brings out a new motivation for conservative refitting: It is required not only for ensuring internal efficiency of planning (as was discussed in Section 5), but also for *containing* the ripple effects of changes in the plan on the analyses of other modules.

level process planning. In particular, while plan optimality is very important in production-level process planning, in NEXT-CUT, which is concerned with prototyping, speed of response regarding feasibility of accommodating user-initiated changes is considered more important than optimality of the process plan.

[18]See Kambhampati *et al.* (1993) for details on how the interactions among the various modules are managed.

[19]There are several additional complexities of both plan generation and reuse (brought out to a large extent by the need for cooperation between the planner and the other modules), which are not reflected in the discussion here. For a more comprehensive treatment, the reader is referred to Kambhampati *et al.* (1993).

[20]Note once again that refitting based on such status-quo heuristics may not necessarily guarantee the optimality of the resultant process plan.

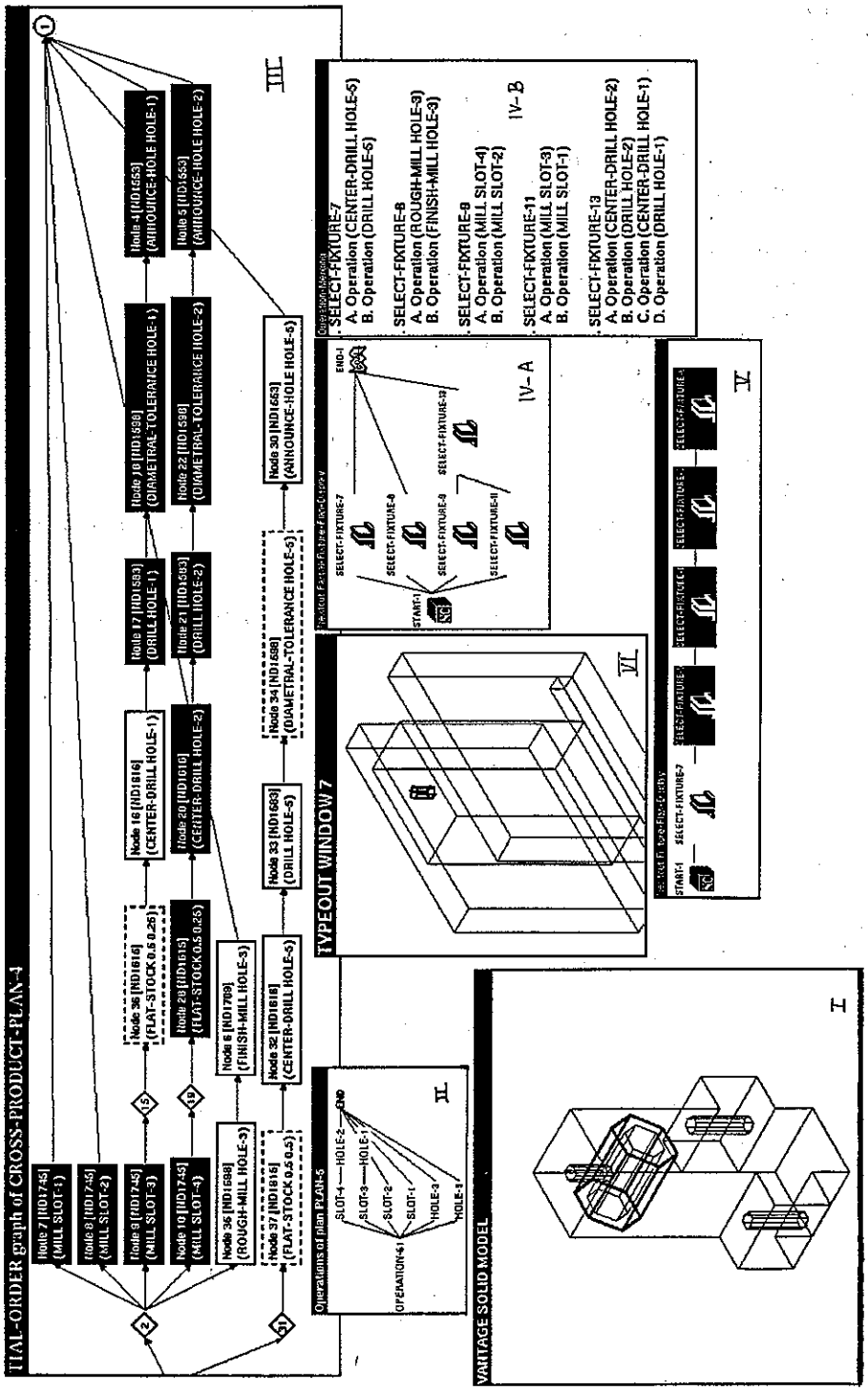


FIGURE 9. Example of incremental plan reuse in process planning domain.

7. CONCLUSION

In this paper, we have described two domain-independent heuristic techniques, which use a representation of plan causal dependency structure, called the *validation structure*, to control retrieval and refitting during plan reuse. We have discussed the implementation and evaluation of these techniques within the PRIAR plan modification framework. The evaluation consisted of testing in blocks world, as well as an application to manufacturing planning domain.

Our main contribution to the mapping and retrieval problem is an efficient domain independent computational measure of the degree of similarity of plans that takes surface *and* structure of solution into account. In particular, we utilize 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 (mis)match between the initial and goal state specifications of the stored plan and the new problem situation. The validation structure of the plan guides this matching process by providing an effective means for measuring the dependencies between the goal and initial states of the stored plan as well as the degree of importance of various features of the initial state of the stored plan. 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.

For controlling refitting, we presented an informed backtracking strategy based on a minimum conflicts heuristic. The strategy orders the refit task reduction choices based on the amount of disturbance they can cause to the validation structure of the plan being reused. In contrast to the traditional minimum-conflict type heuristics, which are only concerned with minimizing the number of conflicts, our strategy uses the plan validation structure to weight the inconsistencies in terms of the estimated difficulty of repairing the inconsistency. It also employs a consistency check based on the domain axioms to get a more realistic estimate of the conflicts caused by a particular modification choice. We have discussed several ways of making the strategy more informed by exploiting the hierarchical nature of the underlying planner and analyzed the complexity and the utility of the strategy.

The research reported in this paper demonstrates that in the absence of any other domain-specific knowledge, the causal dependency structure of plans can be exploited to effectively control retrieval and refitting during plan reuse. A promising direction for further study involves complementing these domain-independent control strategies with domain specific control information acquired through adaptive and speedup learning strategies. Our current work (Kambhampati 1992; Kambhampati and Chen 1993) is aimed at understanding the issues involved in facilitating such an integration.

ACKNOWLEDGMENTS

The advice and influence of Jim Hendler during the formative stages of this work, as well as the helpful feedback from Larry Davis, Lindley Darden, Mark Drummond, Amy Lansky, Jack Mostow, Austin Tate, Andrew Philpot, and the reviewers of this journal is gratefully acknowledged. Support for this research has been provided in part by the DARPA and the U.S. Army Engineer Topographic Laboratories under contract DACA76-88-C-0008 (to the University of Maryland Center for Automation Research), the Office of Naval Research under contract N00014-88-K-0620 (to Stanford University Center for Design Research), National Science Foundation Grant IRI-9210997 (to Arizona State University), the ARPA/Rome Lab-

oratory planning initiative under grant F30602-93-C-0039 (to University of Maryland and Arizona State University), and the Washington, DC, Chapter of A.C.M. through the "1988 Samuel N. Alexander A.C.M. Doctoral Fellowship."

REFERENCES

CARBONELL, J. G. 1983. Derivational analogy and its role in problem solving. Proceedings of National Conference on Artificial Intelligence, Washington DC: 64–69.

CARBONELL, J. G. 1986. Derivational analogy: a theory of reconstructive problem solving and expertise acquisition. *In* Machine Learning: an artificial intelligence approach, vol. 2. *Edited by* R. Michalski, J. Carbonell, and T. M. Mitchell. Morgan Kaufmann, Palo Alto, CA.

CHAPMAN, D. 1987. Planning for conjunctive goals. Artificial Intelligence, **32**:333–377.

CHARNIAK, E., and D. MCDERMOTT. 1984. Managing plans of actions. *In* Introduction to artificial intelligence. Addison-Wesley, Reading, MA, pp. 485–554.

CUTKOSKY, M. R., and J. M. TENENBAUM. 1990. A methodology and computational framework for concurrent product and process design. Mechanism and Machine Theory, **23**:5.

DEAN, T., and M. BODDY. 1988. Reasoning about partially ordered events. Artificial Intelligence, **36**:375–399.

DRUMMOND, M., and K. CURRIE. 1989. Goal ordering in partially ordered plans. Eleventh International Joint Conference on Artificial Intelligence: 960–965.

FELDMAN, R., and P. MORRIS. 1990. Admissible criteria for loop control in planning. Eighth National Conference on Artificial Intelligence (AAAI-90):1151–1157.

GENTNER, D. 1983. Structure-mapping: a theoretical framework for analogy. Cognitive Science, **7**, 155–170.

HAMMOND, K. J. 1990. Explaining and repairing plans that fail. Artificial Intelligence, **45**:173–228.

KAMBHAMPATI, S. 1989. Flexible reuse and modification in hierarchical planning: a validation structure based approach. CS-Technical Report-2334, CAR-Technical Report-469, Center for Automation Research, Department of Computer Science, University of Maryland, College Park, MD (Ph.D. dissertation).

KAMBHAMPATI, S. 1990. A classification of plan modification strategies based on their information requirements. AAAI Spring Symposium on Case-Based Reasoning.

KAMBHAMPATI, S. 1992. Utility tradeoffs in incremental modification and reuse of plans. AAAI Spring Symposium on Computational Considerations in Supporting Incremental Modification and Reuse.

KAMBHAMPATI, S., and J. CHEN. 1993. Relative utility of EBG based plan reuse in partial ordering vs. total ordering planning. Proceedings of 11th National Conference on Artificial Intelligence: 514–519.

KAMBHAMPATI, S., M. R. CUTKOSKY, J. M. TENENBAUM, and S. H. LEE. 1993. Integrating general purpose planners and specialized reasoners: case study of a hybrid planning architecture. IEEE Transactions on Systems, Man and Cybernetics (Special Issue on Planning, Scheduling and Control), **23**(6).

KAMBHAMPATI, S., and J. A. HENDLER. 1992. A validation structure based theory of plan modification and reuse. Artificial Intelligence Journal, **55**(2–3):193–258.

KAMBHAMPATI, S., and S. KEDAR. 1994. A unified framework for explanation-based generalization of partially ordered partially instantiated plans. Artificial Intelligence, **67**(2). A preliminary version appears in the proceedings of AAAI-91).

KOLODNER, J. L. 1983. Maintaining organization in a dynamic long-term memory. Cognitive Science, **7**:243–280.

MINTON, S., A. B. PHILIPS, P. LAIRD, and M. D. JOHNSTON. 1990. Solving large-scale constraint-satisfaction and scheduling problems using a heuristic repair method. Eighth National Conference on Artificial Intelligence (AAAI-90): 17–24.

MITCHELL, T. M., R. M. KELLER, and S. T. KEDAR-CABELLI. 1986. Explanation-based generalization: a unifying view. Machine Learning, **1**, 1.

MITTAL, S., and A. ARAYA. 1986. A knowledge-based framework for design. Proceedings of 5th National Conference on Artificial Intelligence: 856–865.

SACERDOTI, E. D. 1977. A structure for plans and behavior. Elsevier North-Holland, New York.

SELMAN, B., H. LEVESQUE, and D. MITCHELL. 1992. A new method for solving hard satisfiability problems. Proceedings of 10th National Conference on Artificial Intelligence (AAAI-92): 440–446.

SIMMONS, R. 1988. A theory of debugging plans and interpretations. Proceedings of 7th National Conference on Artificial Intelligence: 94–99.

SIMON, H., and J. B. KADANE. 1975. Optimal problem solving search: all-or-none solutions. Artificial Intelligence, **6**.

TATE, A. 1977. Generating project networks. Proceedings of 5th IJCAI: 888–893.

TURNER, R. M. 1987. Issues in the design of advisory systems: the consumer-advisor system. GIT-ICS-87/19, School of Information and Computer Science, Georgia Institute of Technology.

WALDINGER, R. 1977. Achieving several goals simultaneously. *In* Machine intelligence 8, *Edited by* E. B. D. Michie. Edinburgh University Press, pp. 94–136.

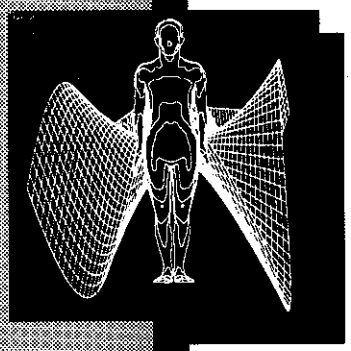
WILKINS, D. E. 1984. Domain-independent planning: representation and plan generation. Artificial Intelligence, **22**:269–301.

ZWEBEN, M., M. DEALE, and R. GARGAN. 1990. Anytime rescheduling. Proceedings of DARPA workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, CA.

LIST OF SYMBOLS

Symbol	Meaning
$>$	Successor of
$<$	Predecessor of
$\parallel$	Unordered
$\exists$	There exists
$\forall$	For all
$\vdash$	Directly follows from facts
$\overset{d}{\vdash}$	Follows from facts *and* domain axioms
$\Diamond$	Modal operator for possible truth
$\Box$	Modal operator for necessary truth
$\supset$	Logical implication
$\neg$	Logical negation
$\langle\,\rangle$	Tuple notation
$\cup$	Set union
$\in$	Element of
$\rightarrow$	Maps to
$\wedge$	Logical conjunction
$\vee$	Logical disjunction
$\mathbf{V}$	Set of validations of a plan
N_P	Plan length (or number of leaf nodes in an HTN)

Volume 10, Number 2 May 1994



Computational Intelligence

AN INTERNATIONAL JOURNAL

Contents

Published by Blackwell Publishers Cambridge MA & Oxford UK