Planning for Manufacturing Workpieces by Storing, Indexing and Replaying Planning Decisions

Héctor Muñoz-Avila and Frank Weberskirch
University of Kaiserslautern, Dept. of Computer Science
P.O. Box 3049, D-67653 Kaiserslautern, Germany
E-mail: {munioz|weberski}@informatik.uni-kl.de

Abstract

Planning for manufacturing workpieces is a complex task that requires the interaction of a domain-specific reasoner and a generic planning mechanism. In this paper we present an architecture for organizing the case base that is based on the information provided by a generic problem solver. A retrieval procedure is then presented that uses the information provided by the domain-specific reasoner in order to improve the accuracy of the cases retrieved. However, it is not realistic to suppose that the case retrieved will entirely fit into the new problem. We present a replay procedure to obtain a partial solution that replays not only the valid decisions taken for solving the case, but also justifications of rejected decisions made during the problem solving process. As a result, those completion alternatives of the partial solution are discarded that are already known to be invalid from the case.

Introduction

Planning in the domain of process planning has been the subject of wide studies in AI (Hayes 1987; Kambhampati *et al.* 1991; Gil 1991; Yang & Lu 1994; Nau, Gupta, & Regli 1995; Bergmann & Wilke 1995). Different approaches have been developed for solving problems in this domain ranging from using pure generative planning techniques (Gil 1991) to domain-specific planning (Nau, Gupta, & Regli 1995; Yang & Lu 1994). An interesting proposal combining both approaches has been described in (Kambhampati *et al.* 1991) where a generative planning system is integrated with specialized reasoners to solve complex tasks. We follow this approach with our case-based planning system CAPLAN/CBC. A domain-specific reasoner called the geometric reasoner is used for detecting geometrical interactions in rotary-symmetrical workpieces. These interactions help to speed up the planning and retrieval process. CAPLAN/CBC concentrates on guiding the base level planner on subgoaling and resolution of conflicts related to the use of the material and tools available. The base level planner is SNLP-like (McAllester & Rosenblitt 1991). The reason for this is that plans for machining rotary-symmetrical workpieces are non-serializable for which this class of planners is supposed to have a good performance (Barrett & Weld 1994).

CAPLAN/CBC uses derivational analogy (Veloso & Carbonell 1993) for reusing previous problem solving experiences. Under this approach, planning decisions taken in selected cases are replayed, when solving a new problem. Derivational analogy supposes the integration of the case acquisition, storage and reuse in an unified framework (Veloso 1994).

In the course of this paper we will present two novel techniques for reusing and storing previous problem solving experiences. The techniques are motivated by the following problems:

- Decisions taken when replaying a case may need to be *revised* when completing the solution for a new problem.
- A generic architecture for the case base is required that incorporates the information provided by the geometrical reasoner.

The first problem is a result of the high complexity of the problems involved in this domain. Therefore, it is not realistic to suppose an entirely adequate retrieval. This means that some of the decisions taken when replaying the case may need to be revised when completing the solution for a new problem. Highly accurate retrieval procedures reduce the number of decisions which have to be revised but they cannot avoid revisions entirely. Further, highly accurate retrieval usually diminishes the performance of the whole case-based solution process (Veloso 1994). To deal with this problem CAPLAN/CBC doesn't store only valid decisions in a case but also justifications of decisions that were rejected during the problem solving episode. This information will be used to avoid pursuing completion alternatives of the partial solution that are already known to be invalid from the case.

Little work has been done on indexing the case base in a general problem solving environment. Some systems suppose a preselection of cases (Kambhampati 1994), while others use domain specific techniques (e.g. early versions of this work (Muñoz-Avila, Paulokat, & Wess 1995)) for this purpose. Until now, only few attention has focussed on how to index a large amount of cases in a domain-independent environment (Veloso 1994). The architecture of the case base in CAPLAN/CBC represents the derivation order in the case. The retrieval procedure uses this information and the information provided by the geometrical reasoner to improve its accuracy without reducing the performance of the overall case-based solution process.

This paper is organized as follows. The next section outlines the domain of planning for manufacturing rotary-symmetrical workpieces. Then we present the replay algorithm, the architecture of the case base and the retrieval procedure. Empirical results obtained with CAPLAN/CBC show the advantages. Finally, we discuss related work and make concluding remarks.

Domain Characteristics and Domain-specific Reasoning

The domain we are concerned with is process planning for rotary-symmetrical workpieces to be machined on a lathe. A planning problem in this domain is given by a geometrical description of a workpiece and of a stock. Figure 1 shows an example, "a long shaft". The description is built up from geometrical primitives like cylinders, cones and toroids that describe monotone areas of the outline, possibly augmented by features (threads, undercuts, surface conditions). For such a planning problem a sequence of processing operations is to be found that will machine the workpiece considering available resources (i.e. tools, machines) and technological constraints related to the use of these resources. The process begins with clamping the stock on a lathe machine that rotates it at a very high speed. In most cases, the outline of the workpiece cannot be machined in one step but repeated cutting operations are necessary to cut the difference between the raw material and the workpiece in thin horizontal or vertical layers.

For detecting the interactions between parts of the workpiece, a domain dependent system called the *geometrical reasoner* is used. It establishes constraints on the order for manufacturing certain parts of the workpiece. As each part of the workpiece constitutes a goal in the problem description, these constraints are interpreted as ordering constraints that must be met by any plan for manufacturing that workpiece. Further, these constraints are stated by the geometrical reasoner before the planning process begins, as they depend only on the geometry of the workpiece and not on available tools or on the clamping material.

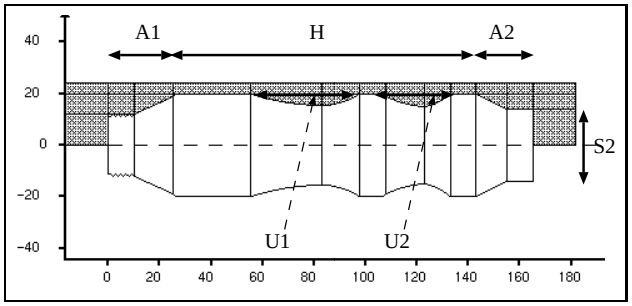


Figure 1: A rotary symmetrical workpiece.

For example, for the workpiece given in figure 1, the geometrical reasoner establishes that the processing area H must be manufactured before the processing areas $U1$ and $U2$. These constraints must be met by any partial-ordered plan for manufacturing that workpiece.

Figure 2 shows a plan fragment for the workpiece in figure 1. Solid lines represent the valid steps and orderings among them. There are steps for specifying a fixturing method, inserting cutting tools and for the cutting operations itself. Additionally, figure 2 shows that there might be alternatives for plan steps (dotted lines). E.g., for processing the second half of the area *U2*, using a certain cutting tool (a "right cutting tool") has been chosen (*STEP-6*) but there are two other alternatives. One is to clamp from the ascending area *A2* (*STEP-6'*), the other is to clamp from side *S2* (*STEP-6"*). This last alternative is labeled rejected to show that the planner already found out, that it cannot be used. The following section will refer to these alternatives again.

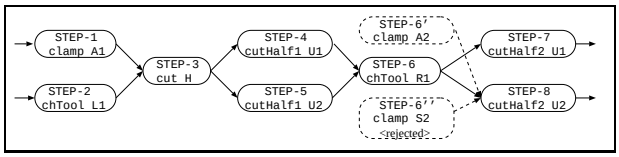


Figure 2: Outline of a manufacturing plan.

Complete Decision Replay

CAPLAN/CBC uses derivational replay (Veloso & Carbonell 1993) on an SNLP-like planner for reusing previous problem solving experiences (cases) similar to DerSNLP (Ihrig & Kambhampati 1994). In this approach derivation paths from cases are replayed when

solving a new problem. At the end of the replay process, a partially ordered plan is obtained that may contain unsolved goals. They are solved by the generative planning system then. This strategy of first replaying cases and then completing the solution by using a planning system is known as *eager replay*. It has been shown to be adequate for SNLP-planners (Ihrig & Kambhampati 1994) such as CAPLAN. However, we will extend this paradigm by replaying justifications of rejected decisions as well. In a way, our procedure *completely* replays previous problem solving experiences.

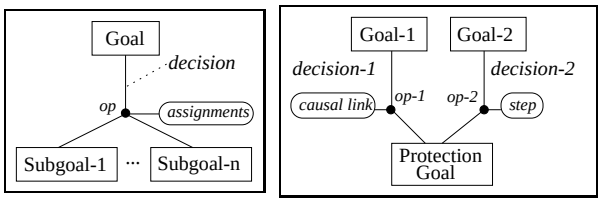
Preliminaries

CAPLAN (Weberskirch 1995) is the SNLP-like (McAllester & Rosenblitt 1991; Barrett & Weld 1994) base level planner of CAPLAN/CBC for generative planning. It uses a partially ordered plan representation (S, O, B) consisting of plan steps S (each of which is associated with an operator schema of the domain), ordering constraints O among the steps and variable binding constraints B.

At the beginning of the planning process, the set of open conditions is constituted by the goals of the problem. For the domain of process planning, the set of open conditions corresponds to the different parts of the workpiece. The planning process proceeds by making decisions about the *establishment of open conditions* and the *resolution of threats*. The former is achieved by adding a causal link between a new or an existing step and the step that consumes the condition, the latter is achieved by adding ordering or binding constraints that remove the threat. For example, for the plan shown in figure 2, the ordering link between *STEP-3* and *STEP-4* is the result of establishing an open precondition, as processing the undercut *U1* has a precondition that area H already has been processed. On the other hand, the link between *STEP-4* and *STEP-7* is the result of resolving a threat, as for each half of the undercut a different type of tool is needed, thus causing a conflict between these steps (not shown in fig. 2). The planning process terminates if no open preconditions are left and all threats to existing causal links are resolved. Both, precondition establishment and threat resolution, are *choice points* of the planning algorithm, where a decision has to be made.[1] Making wrong decisions causes backtracking and decreases the performance of the planning system. Supporting CAPLAN in this decision-making process is the purpose of the case-based control component CBC.

Basically, a case in CAPLAN/CBC consists of a structure that encapsulates the decisions made at the choice points during a problem solving episode. Additionally, cases contain information concerning justifications of rejected decisions. Roughly, justifications are reasons for rejecting previously made decisions at this choice point. For example, the plan in figure 2 shows two alternatives for plan step *STEP-6*. One of them, *STEP-6"*, is labeled *rejected* and the justification for the rejection can be expressed as follows: "the establishment *there is a hole in s2* has failed" (not shown in figure 2). The rationale behind storing justifications in cases is that decisions taken during the replay procedure may have to be rejected in order to complete the plan. As a result, reconstructing rejection justifications from the case avoids pursuing alternatives, which are known to be invalid from the cases.

For representing knowledge about plans and contingencies that occur during planning, CAPLAN is built on the generic REDUX architecture (Petrie 1991). Key concepts of REDUX are goals, constraints, and contingencies. Planning proceeds by applying operators to goals, what may result in subgoals and in assignments (fig. 3.a). Applying an operator is called a *decision* and



(a) Decision (b) Protection goal

Figure 3: Elements of the subgoal graph

represents a backtracking point, as different operators might be applicable to a goal.

Assignments originally are thought to assign values to variables, more generally, they stand for modifications made in the plan (addition of steps/orderings/constraints). So, the mapping of SNLP concepts to REDUX concepts is straightforward. There are two types of goals: (1) goals to establish open conditions (precondition goal) and (2) goals to resolve threats (protection goal). Each refinement method of SNLP (precondition establishment or threat resolution) is represented as a class of operators that can be chosen to be applied to such a goal.

Goals and subgoals build the *subgoal graph*. It represents basic dependencies between goals and subgoals as well as between subgoals and decisions. Originally, REDUX makes the assumption that each goal can have only one parent goal, so the subgoal graph is in fact a tree. However this is not adequate for SNLP and has

[1] Whether to establish preconditions or to resolve threats can be seen as another choice point with the difference that it is no backtracking point and unimportant for replay.

ReplaySubgoalGraph(P, G, N, C)
For each goal $g \in G$:
- compute $conflictSet(g)$
- get goal g_c of case C that matches g
- get chosen decision d_c of g_c from case
- IF $\exists op \in conflictSet(g)$ with $matches(op, d_c)$ THEN
  · apply operator op to goal g (modifies plan P)
  · get subgoals of goal g
  · match subgoals of g and subgoals of g_c
  · let SG be the set of matching subgoals
  · ReplaySubgoalGraph(P, SG, N, C)
  · ReplayDelayedGoals(P, N, C)
- ELSE
  add goal g to the set N of not linked goals

ReconstructJustifications(P, C)
- get set D of reconstructed decisions of case C
- for each justification j_c of a $d_c \in D$:
  IF $\exists dependentDecisions(j_c)$ THEN
  · create justification j matching j_c
  · add j to decision d

ReplayDelayedGoals(P, N, C)
For each goal $g \in N$:
- compute $conflictSet(g)$
- get goal g_c of case C that matches g
- get chosen decision d_c of g_c from case
- IF $\exists op \in conflictSet(g)$ with $matches(op, d_c)$ THEN
  apply operator op to goal g (modifies P)

Figure 4: The replay algorithm – reconstruction of a subgoal graph and rejection justifications

been modified in CAPLAN for the following reason: protection goals represent threats and must depend on two other goals (fig. 3.b), first, the goal with an operator that added the threatened causal link, second, the goal with an operator that added the threatening step. This extension of the dependency structure is important for automatically identifying threats and threat resolutions that become invalid after the rejection of a decision.

Basically, REDUX represents validity and local optimality of decisions and dependencies among them (cf. (Petrie 1992)). In the case of rejecting a decision, this structure is traversed to reject dependent decisions also (Weberskirch 1995). We omit further details, as they are beyond the purpose of this paper.

Decision Replay in CAPLAN/CBC

A case in CAPLAN/CBC consists of the following elements: (i) the subgoal graph built with precondition and protection goals, (ii) for each goal the chosen operator (decision) and the rejected decisions (i.e., previously selected but later rejected operators for solving this goal) are saved, (iii) for each operator (decision), all rejection justifications are saved.

Replaying a case consists of two phases: (1) reconstruction of the subgoal graph of a case (i.e., subgoals and selected operators recursively as complete as possible) with respect to the current problem, (2) reconstruction of justifications for the rejection of operators.

After a case C has been chosen to be used in a new problem situation (see next section), the goals of the problem are mapped to corresponding top level goals of the case C. Let G be the set of matching goals of the problem. The algorithm sketched in figure 4 replays the decisions of the case C for the goals G with respect to a current plan P. N is an initially empty list of goals that could not be matched immediately and so are delayed (see below).

Phase 1 (**ReplaySubgoalGraph**), the main part of algorithm, reconstructs the part of the subgoal graph for each goal that matches a goal of the case. The computed conflict set of a goal is always a set of operators that can be applied to the goal. If the matching operator for a decision of the case is not yet in the conflict set of the goal, the replay of this decision is delayed by adding this goal to the list N of goals that have to be delayed. This can for example happen with a phantom operator[2] that uses a step that has not yet been replayed in the current plan. It will be delayed until the needed step is added by the replay procedure, or it is skipped if the step is not created during the replay. Otherwise, the same decision as in the case is replayed foe the goal, and **ReplaySubgoalGraph** is called recursively for subgoals that match subgoals in the case. After the recursive call, the algorithm makes a new attempt at replaying the goals which had been delayed before (**ReplayDelayedGoals**).

Phase 2 of the replay (**ReconstructJustifications**), starts after the reconstruction of the subgoal graph is finished. First, all decisions of the case that were reconstructed are collected. Then, for each rejection justification of a decision the algorithm checks if a reconstruction of the justification is possible. This is the case if all decisions that are referred to in the justification exist in the actual situation. Then an equivalent justification referring to elements of the current plan is added for this decision. Otherwise, the justification can be skipped as it depends on the existence of a part of the case that has not been reconstructed.

For illustration purposes, suppose that a new problem is given, consisting of manufacturing the workpiece shown in figure 1. Suppose additionally that the plan

[2] A *phantom operator* adds a causal link but not a new step (simple establishment).

shown in figure 2 is to be replayed, but that no tool of type right cutting tool is available. All the steps except *STEP-6* can be replayed in the new problem. Additionally, the justification in *STEP-6"* can also be replayed, as we are assuming that the workpiece is the same. Thus, there is no hole in *S2*. During the completion process, *STEP-6'* is the only remaining alternative that can be selected at this choice point. Notice however, that if the justification at *STEP-6"* would not have been stored, this step might have been selected during the completion process, resulting in backtracking. The reconstructed justification will definitely avoid backtracking here.

Automatic Indexing of Cases

As stated before, the geometrical reasoner computes ordering constraints for the manufacturing process of certain parts of a workpiece. This computation takes place before the planning process begins. CAPlan/CbC takes advantage of this information by explicitly representing the ordering constraints of the goals in the index structure of its case base.

Dependencies between goals

In the following sections we will only consider complete partial-ordered plans, unless stated otherwise. A partial-ordered plan (S, O, B) is said to be complete if all preconditions of steps in S are established and all threats to causal links are resolved (McAllester & Rosenblitt 1991). The following definition formalizes the notion of order among goals.

Definition 1 (Goal Dependencies) *Let g, h be two goals achieved by a complete partial-ordered plan (S, O, B). The goal h is said to depend on the goal g, $g <_{goal} h$, if there are two plan steps s_g, s_h in S such that:*

1. *The plan step s_g, achieving g, precedes the plan step s_h, achieving h.*
2. *There is no plan step $s \in S$, such that $s_g <_O s$[3] ($s_h <_O s$) and s clobbers g (respectively h).*

For example, for the plan shown in figure 2, processing *U1* (*STEP-4*) depends on processing *H* (*STEP-3*). Notice that the second condition in definition 1 is equivalent to stating that s_g (s_h) is the maximal step with respect to $<_O$ to achieve g (respectively h). In this case g (h) can not be clobbered, as we are assuming that (S, O, B) is complete. Further, there cannot be two maximal steps achieving g (h) that meet the second condition of definition 1, because the completeness of (S, O, B) ensures that all positive threats are resolved. Together with the fact that $<_O$ is transitive, this ensures that the dependency relation is transitive. Non-reflexivity follows from the fact that the directed graph $(S, <_O)$ is always acyclic and from the completeness of (S, O, B). The following lemma summarizes these properties:

Lemma 1 *$<_{goal}$ is an ordering relation.*

Given a plan (S, O, B), it is possible to form a directed acyclic graph $(G, <_{goal})$, where G is the set of goals achieved by (S, O, B). Figure 5 shows the corresponding goal graph for the plan shown in figure 2. Under this perspective, we can state that a set of goals

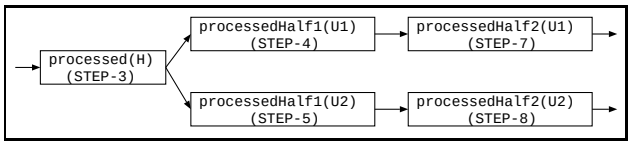


Figure 5: A dependency graph

interact (Veloso 1994) if they are in the same connected component of the goal graph. In particular, all the goals shown in figure 5 interact. This shows that the dependency order refines the concept of interacting goals. We will use this refinement to discriminate cases that have the same set of interacting goals, as shown in the following section.

Organization of the Case base

The case base in CAPlan/CbC is organized as a four level structure (fig. 6). At the top level, there are two access tables (shown partially in fig. 6). The access tables define entry points in the goal discrimination network. They discriminate cases based on the class representation[4] of sets of interacting goals. Basically, these tables define a hash function that identifies for each set of interacting goals a tree in the goal discrimination network.

At the second level there is a goal discrimination network (GDN). The goal discrimination network defines entry points into the partially-ordered initial-state discrimination network. Every tree in this network represents different orderings of the same set of interacting goals. This ordering is based on the dependency order as we will see in the next section.

[3] $<_O$ is the ordering relation represented by O.

[4] The *class representation* (Veloso 1994) of a set of predicates is obtained by replacing the arguments of each predicate with their types. For example the class representation of $\{processed(H)\}$ is $\{processed(CL - H)\}$. Comparing class representations is not expensive, as the arguments of the predicates are constants.

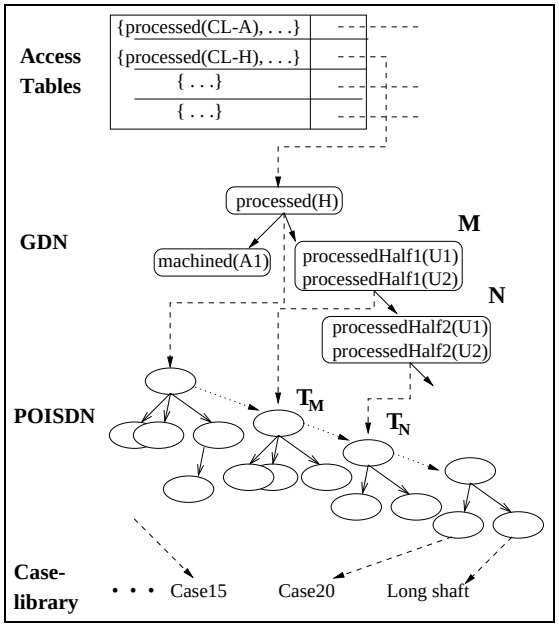


Figure 6: Architecture of the case base.

The third level is constituted by a partially-ordered initial-state discrimination network (POISDN). It discriminate cases that have the same set of interacting goals and that meet the same order in the goal discrimination network. As illustrated in figure 6, every node N in the goal discrimination network points to a tree T_N in the POISDN. Every tree T_N in this network is constructed in such a way that any path from the root of T_N to a leaf contains the foot-print[5] of the goals in node N. Finally, the fourth level is constituted by the library of cases.

The Goal Discrimination Network (GDN)

The basic principle is to represent the dependency order through the order indicated by the arcs of every path within the goal discrimination network (GDN). This is not always possible, because not every graph (in our case dependency graphs) can be represented through a partial linearization (in our case paths). However, the paths in GDN are constructed in a way that discriminate cases by using the dependency order.

Before continuing, some notations are necessary. Let N be a node in GDN. Then $cases(N)$ denotes the set of cases that are reachable from the corresponding tree T_N in POISDN. $CASES(N)$ denotes the union of all $cases(M)$ such that there is a path from N to M in GDN, including N itself (i.e., the empty path). The modal operators "$\Box$" and "$\Diamond$" denote necessary and possible truth of an assertion. An assertion is necessarily true in a plan, if it is true in every ground linearization of the plan. Finally, we will refer to cases as plans, actually meaning the plan represented by a case. The properties of a GDN can be stated as follows:

Definition 2 (GDN-properties) *For all nodes M, N in the goal discrimination network, such that N is a successor of M, the following conditions are met:*

1. *For every goal g_M in M and every plan (S, O, B) in $CASES(N)$, there exists a goal g_N in N such that: $\Box(g_M <_{goal} g_N)$*
2. *There exists no goals g_M in M, g_N in N and no plan (S, O, B) in $CASES(N)$ such that: $\Diamond(g_N <_{goal} g_M)$*

For illustrative purposes consider the nodes labeled M and N in figure 6 and the plan in figure 2. The arc connecting M and N does not represent the dependency order. The reason for this is that $processedHalf1(U1) <_{goal} processedHalf2(U2)$ does not hold for that plan. However M and N satisfy the properties stated above. The first property can be illustrated by the fact that $processedHalf1(U1) <_{goal} processedHalf2(U1)$ holds in that plan. The second property can be illustrated by the fact that $processedHalf2(U1) <_{goal} processedHalf1(U2)$ does not hold in this plan. As we will see later, these properties, result in a major improvement for retrieving relevant cases and for speeding up the overall case-based solving process.

Retrieval

Roughly, *the GDN reflects the derivation order that the replay procedure will follow when using the underlying cases.* We will now show how the retrieval mechanism takes advantage of this fact to improve its accuracy.

A planning problem (Muñoz-Avila & Hüllen 1995; Weberskirch 1995) in CAPLAN/CBC can be seen as a triple $(I, G, <_G)$, where I is the initial state, G is the set of goals and $<_G$ is the ordering constraints for achieving the goals. As stated previously, these order constraints are established by the geometrical reasoner prior to the beginning of the planning process. Any plan (S, O, B) that solves $(I, G, <_G)$ must meet these constraints. This means that the plan steps achieving the goals in G, are ordered in a way that meets the order for achieving these goals, $<_G$. Thus, *the dependency order of the plan extends the ordering constraints of the problem (i.e. $<_G \subseteq <_{goal}$).*

For retrieving cases CAPLAN/CBC follows a top down strategy (Veloso 1994): it tries to cover the set

[5] Given a goal g achieved by a plan (S, O, B), it is possible to identify the connected component C in the directed acyclic graph $(S, <_O)$, where g is achieved. All the elements of the initial state contained in C, are referred to as the *foot-print* of g (Veloso 1994).

of goals G with as few cases as possible. Covering a set of goals $G1$ ($G1 \subseteq G$) with a plan (S, O, B) means that $G1$ matches a set of interacting goals G_{plan} in the plan, and that I matches the foot-print of G_{plan} with a predefined accuracy. CAPLAN/CBC imposes an additional condition:

Definition 3 (Order Consistency Condition) *For every pair of goals $g_1, g_2 \in G1$, if $g_1 <_G g_2$ holds, then $g_2 <_{goal} g_1$ does not hold.*

The rationale behind the order consistency condition is that ordering constraints represent decisions which have to be made during the replay phase or later on when completing the plan. Choosing a case which violates ordering constraints implies that a decision will be replayed, which has to be rejected later. This rejection process implies backtracking that is exactly what we are trying to reduce by replaying previous decisions.

Notice that the order consistency condition corresponds to the second GDN-property. As a result, during the early stage of the retrieval procedure, most of the cases in the case base may be discarded, as they violate the order consistency condition. We have conducted experiments which show that indeed the GDN contributes to improve the accuracy of the retrieval process and thus the performance of the overall case-based solving process.

Empirical results

To measure the effectiveness of the replay procedure presented in this paper, we performed the following experiment in the domain of process planning: Several problems were solved by the base-level planner CAPLAN and we stored the plans in the case base. While solving the problems we misdirected CAPLAN at key choice points to ensure that justifications for wrong decisions which result in backtracking at those key choice points are constructed. The created case base was then used to solve new planning problems (variations of the cases). For each problem a case was selected and replayed with and without reconstructing justifications of failed decisions. Table 1 summarizes the results. The second column shows the averages

Replay	**Complete**	**Normal**
Total Time (s)	25	65
# Inferences	37	129
# Decisions	33	72
# Valid Decisions	21	21

Table 1: Effectiveness of complete decisions replay.

of the measurements made during the completion process for all given problems when complete replay was used. The third column shows the measurements for normal replay, i.e. when the justifications were not reconstructed. We measured the completion time, the number of inference steps to find the solution, the total number of decisions made during completion and the number of valid decisions. The ratio between the number of interence steps and the number of valid decision shows the degree of backtracking needed to find the solution. The overhead caused by reconstructing the justifications was approximately 25% of the replay time, which was smaller than the completion time (not shown in table 1). However, table 1 shows the advantage of reconstructing justifications over replaying only valid decisions as in (Ihrig & Kambhampati 1994): completion time decreased more than 50% and significantly less inference steps were need to find a solution.

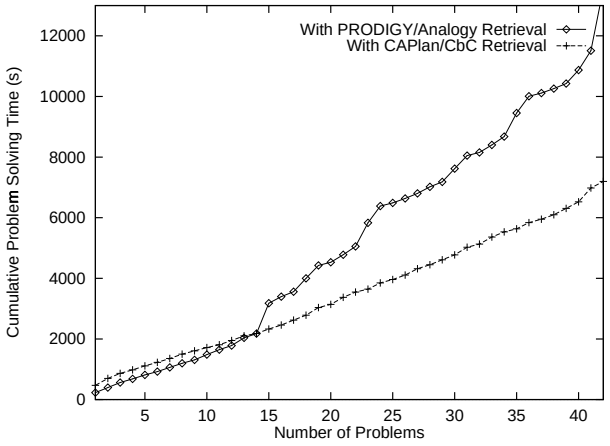


Figure 7: Evaluating the effectiveness of the case base

To measure the effectiveness of our retrieval approach we incorpored the architectures of the case bases as described in this paper and of PRODIGY/ANALOGY in our system (see (Muñoz-Avila & Hüllen 1995) for a detailed comparison). Both architectures were given the same sequence of problems $\{P_i\}$ of the domain of process planning. The problem P_i consists of a variation of P_{i-1} by addding a goal and/or changing and adding conditions in the initial state. Further, P_{i-1} covers the goals of P_i and their initial states matched with an accuracy of at least 80%. However, the order restriction $<_{P_i}$ of P_i is not necessarily consistent with the order $<_{P_{i-1}}$ of every solution of P_{i-1}. We measured the overall problem-solving time for each architecture of the case base. Figure 7 shows the results. The first twelve problems were given without considering the information provided by the ge-

ometrical reasoner. In this situation our architecture lowers the performance of the overall process. The reason is that the retrieval procedure will have to perform several traverses of the GND, repeatly doing the same matches. Under these circumstances the GND tends to disperse the cases rather than to classify them. The remaining problems where given including the ordering restrictions provided by the geometrical reasoner, which results in a significantly decrease of the problem-solving time.

Discussion

A significant difference with previous work done in case-based planning such as (Veloso 1994; Ihrig & Kambhampati 1995; Yang & Lu 1994) is that we extended the concept of problem description. In our approach problem descriptions not only include the initial state I and goals G, but also ordering constraints $<_G$ between goals. As explained before, the ordering constraints restrict the order between the plan steps of any solution of that problem. Thus, the extended problem description $(I, G, <_G)$ can be viewed as an intentional description of a collection of plan fragments $\{P'\}$. Under this perspective, the retrieval problem can be reestated as to find a plan P in the case-base, such that there is a plan fragment P' in the collection which can be extended into a complete solution by reusing P.

Conclusions

Planning for manufacturing mechanical workpieces is a complex task that cannot be performed by the generic case-based problem solver alone. We have shown how to use the information provided by the generic problem solver to organize a case base. We have also seen how the retrieval procedure takes advantage of the information provided by the geometric specialist. Even though we saw that this organization improves the accuracy of the retrieved cases, it is nevertheless unrealistic to suppose that backtracking can be completely avoided. We have presented a procedure that replays valid and rejected decisions stored in the cases. The former contributes to obtaining a partial solution. The latter help to discard completion alternatives of the partial solution that are already known to be invalid from the case.

References

Barrett, A., and Weld, D. 1994. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence* 67(1):71–112.

Bergmann, R., and Wilke, W. 1995. Building and refining abstract planning cases by change of representation language. *Journal of Artificial Intelligence Research* 3:53–118.

Gil, Y. 1991. A specification of manufacturing processes for planning. Technical Report CMU-CS-91-179, School of Computer Science, Carnegie Mellon University, Pittsburg.

Hayes, C. 1987. Using goal interactions to guide planning. In *Proceedings of AAAI-87*, 224–228.

Ihrig, L., and Kambhampati, S. 1994. Derivational replay for partial-order planning. In *Proceedings of AAAI-94*, 116–125.

Ihrig, L., and Kambhampati, S. 1995. Automatic storage and indexing of plan derivations based on replay failures. In *Proceedings of IJCAI-95*.

Kambhampati, S.; Cutkosky, M.; Tenenbaum, M.; and Lee, S. 1991. Combining specialized reasoners and general purpose planners: A case study. In *Proceedings of AAAI-91*, 199–205.

Kambhampati, S. 1994. Expoiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence* 10(2):213–244.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of AAAI-91*, 634–639.

Muñoz-Avila, H., and Hüllen, J. 1995. Retrieving relevant cases by using goal dependencies. In *Proceedings of the 1st International Conference on Case-Based Reasoning (ICCBR-95)*.

Muñoz-Avila, H.; Paulokat, J.; and Wess, S. 1995. Controlling non-linear hierarchical planning by case replay. In Keane, M.; Halton, J.; and Manago, M., eds., *Advances in Case-Based Reasoning. Selected Papers of the 2nd European Workshop (EWCBR-94)*, number 984 in Lecture Notes in Artificial Intelligence. Springer.

Nau, D.; Gupta, S.; and Regli, W. 1995. AI planning versus manufacturing-operation planning: A case study. In *Proceedings of IJCAI-95*.

Petrie, C. 1991. *Planning and Replanning with Reason Maintenance*. Ph.D. Dissertation, University of Texas at Austin, CS Dept.

Petrie, C. 1992. Constrained decision revision. In *Proceedings of AAAI-92*, 393–400.

Veloso, M., and Carbonell, J. 1993. Derivational analogy in prodigy: Automating case acquisition, storage, and utilization. *Machine Learning* 10.

Veloso, M. 1994. *Planning and learning by analogical reasoning*. Number 886 in Lecture Notes in Artificial Intelligence. Springer Verlag.

Weberskirch, F. 1995. Combining SNLP-like planning and dependency-maintenance. Technical Report LSA-95-10E, Centre for Learning Systems and Applications, University of Kaiserslautern, Germany.

Yang, H., and Lu, W. 1994. Case adaptation in a case-based process planning system. In Hammond, K., ed., *Proceedings of the 2nd International Conference on AI Planning Systems (AIPS-94)*. The AAAI Press.