General-Purpose Case-Based Planning: Methods and Systems

Ralph Bergmann[1], Héctor Muñoz-Avila[1], and Manuela Veloso[2]
[1]Dept. of Computer Science
University of Kaiserslautern
P.O. Box 3049, D-67653 Kaiserslautern, Germany
E-mail: {bergmann|munioz}@informatik.uni-kl.de

[2]School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213-3891, USA, E-mail: {mmv}@cs.cmu.edu

1 Introduction

Planning consists of the construction of some course of actions to achieve a specified set of goals, starting from an initial situation. For example, determining a plan of actions (often called operators) for transporting a certain set of goods from a source location to some destination provided that there are different means of transportation is a typical planning problem in the transportation domain. Many planning problems of practical interest also appear in engineering domains, e.g., the domain of process planning.

The classical generative planning process consists mainly of a search through the space of possible sets of operators to solve a given problem. In pure case-based planning instead, new problems are solved by reusing plans or portions of plans from previous cases. Since the space of possible plans is typically vast, it is extremely unlikely that a case-base contains a plan that can be reused without modification.

Given that classical generative planning may engage in a very large search effort and pure case-based planning may be encounter unsurmountable modification needs, several researchers have pursued a synergistic approach of generative and case-based planning. In a nutshell, the generative planner is used as the source of modification, and the case-based planner provides plans previously generated for similar situations. In this paper, we present three systems that integrate generative and case-based planning.

Pure case-based planners in general could be domain-specific or domain-independent. For example, Chef [Ham86] is a well-known domain-specific case-based planner. In this approach cases encode the domain knowledge. The adaptation mechanism uses domain knowledge for indicating, how previous plans can be transformed for solving new problems.

Our synergistic approaches can be viewed as domain-independent case-based planning. The domain knowledge is represented through the planning operators. Operators describe actions indicating how the state of the world can be changed. These changes are called the effects of the operator. For applying an operator certain conditions, called preconditions must be met. In the transportation domain for example, the action of moving a vehicle from one place x to another place y can be modeled through an operator. The precondition of the operator is that the truck is in the place x. The effect of applying the operator is that the truck is in y. Using a complete set of operators, a generative planner would theoretically be able to solve any consistently defined problem. However, in most practical situations the search space that a generative planner must traverse to find a solution is so vast that solutions cannot be found in reasonable time.

Domain-independent case-based planners accumulate and use planning cases to control the search. Under this perspective, cases encode knowledge on how and which operators where used for solving problems. In our synergistic systems, case-based and generative planning are integrated and work tightly together. Thereby the workload imposed on the generative planner depends on the amount of modification that is required to completely adapt a retrieved case.

2 Case-based Planning

We now describe a general framework for case-based planning based on the CBR process model by Aamodt and Plaza [AP94]. This will cover most work on case-based planning which is based on the integration of a generative problem solver with a case-based component. This includes the approaches developed at the CMU and the University of Kaiserslautern.

2.1 Retrieval and Organization of the Case-Base

Given a new problem, the goal of the retrieval phase is to select a case from the case-base whose problem description is most similar to the description of the new problem. In case-based planning, the similarity has a tight connection to the abilities of the reuse component, i.e. the retrieval should select adaptable cases [SK94]. More precisely, the main goal of similarity assessment is to predict the effort required for reusing the solution to solve the new problem. Therefore, a case should be considered very similar to the new problem, if only little effort is required for adapting the solution and less similiar if the adaptation is considered computationally very expensive. In case-based planning, the reusability of a

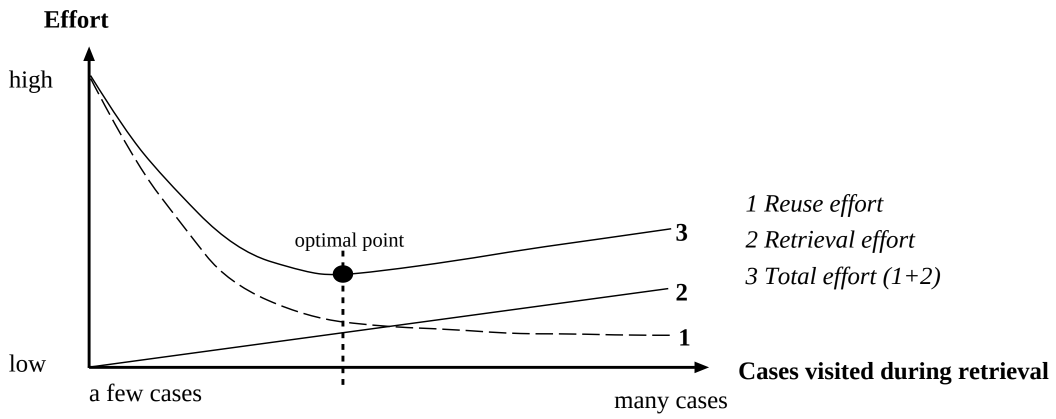


Figure 1: Trade-off between retrieval effort and reuse effort (adapted from [Vel94])

case is strongly determined by the particular solution contained in the case and not only on the problem description. The similarity assessment therefore determines the fragments of the problem description which are relevant for successfully reusing the plan. Basically, this can be achieved by computing a weakest precondition, based on the domain knowledge about the available operators that ensures that the plan can be successfully applied. The similarity will then be assessed based on the fragment of the conditions that are satisfied in the current problem to be solved.

Depending on the domain, computing the similarity assessment may become very expensive which is a major problem when the case-base has reached a considerable size. In this case, a trade-off between the goal to find the best case and the goal of minimizing the retrieval time appears. Figure 1 shows a typical behavior. As the number of cases visited during retrieval increases, the more time must be spent for retrieval (see curve 2) but the better cases resulting in a shorter adaptation time will be found (see curve 1). Upto a certain point (optimal point), the total case-based planning time (retrieval+reuse, see curve 3) decreases when more cases are visited during retrieval. However, beyond this point the total planning time can increases again if more cases are visited, because the possible gain through finding better cases does not outweight the effort of finding them.

2.2 Reusing Previous Solutions

In case-based reasoning, at least two different kinds of approaches to reuse can be distinguished: *transformation adaptation* and *generative adaptation* [CFS94]. Transformation adaptation methods usually consist of a set of domain dependent heuristics which directly modify the solution contained in the case, based on the difference between the problem descriptions in the case and of the current problem. While transformation adaption was also used in early case-based planning systems (e.g. in CHEF [Ham86]), most recent systems (including our own systems) are based on generative adaptation. For this kind of adaptation, the integration between a case-based approach and a generative problem solver is central. The retrieved solution is not modified directly, but is used to *guide* the generative problem solver to find a solution. The kind of guideance that a case provides differs from system to system and is particularly dependent on the type of generative problem solver that is used. However, a basic principle is the replay of *decisions* that where made during the process of solving the problem recorded in the case. When replaying the solution trace of a previous case, some (hopefully most) decisions can be reused, while the remaining decisions are taken by replaying another case or by using a generative planner. The result of this reuse phase is either a correct solution (w.r.t. the domain model) or the indication of a failure in case the problem could not be solved with allocated (time) resources.

2.3 Revision of Solutions

The goal of the revision phase is to validate the computed solution in the real world or in a simulation of it. Due to the correctness of the reuse-phase, the resulting solutions are known to be correct w.r.t the domain model. Consequently, the simulation of the solution cannot contribute to an additional validation. Therefore, the solution must be validated in the real world. However, no methodological support of this kind is provided by today's case-based planning systems.

2.4 Retaining new Cases

When a new case is obtained, it usually should be entered into the case base. In case-based planning, this phase requires much more effort than case-based approaches for analytic tasks in which the new case is simply stored "as it is". Case-based planning requires determining the "right" goals to index the cases as well as determining the set of decisions (the solution trace) that are taken by the generative problem solver. Roughly speaking, the goals used for indexing are those goals that are required for or affected by the taken decisions stored in the case.

3 Prodigy/Analogy

Prodigy/Analogy was the first system that achieved a complete synergy between generative planning and case-based planning [Vel94] and using for the first time a full automation of the complete CBR-cycle. Prodigy/Analogy is developed within the PRODIGY planning and learning architecture [CKM91]. The generative planner is a means-ends analysis backward-chaining nonlinear planner, performing state-space search. The integration is based on the derivational analogy method [Car86]. This is a *reconstructive* method by which *lines of* *reasoning* are transfered and adapted to a new problem as opposed to transformational methods that adapt directly final solutions. Prodigy/Analogy was and continues to be demonstrated in a variety of domains.

3.1 Retain: Generation of Planning Cases

A planning case to be stored consists of the successful solution trace augmented with justifications, i.e., the derivational trace. The base-level PRODIGY4.0 reasons about multiple goals and multiple alternative operators relevant to achieving the goals. This choice of operators amounts to multiple ways of trying to achieve the same goal. PRODIGY/ANALOGY provides a *language* to capture mainly three kinds of justifications for the decisions made during problem solving: links among choices capturing the goal dependencies, records of failed explored alternatives, and pointers to any external used guidance. We discovered in PRODIGY/ANALOGY that the key feature of this language is that it needs to be reinterpretable at planning replay time.

Automatic generation of the derivational planning episodes occurs by extending the base-level generative planner with the ability to examine its internal decision cycle, recording the justifications for each decision during its search process.

3.2 Indexing and Retrieval of cases

From the exploration of the search space and by following the subgoaling links in the derivational trace of the plan generated [Car86], the system identifies, for each goal, the set of *weakest preconditions* necessary to achieve that goal. We recursively create the so called *foot-print* of a goal conjunct of the problem by doing goal regression, i.e. projecting back its weakest preconditions into the literals in the initial state [Wal77]. Goal regression acts as an explanation of the successful path. The literals in the initial state are therefore *categorized* according to the goal conjunct that employed them in its solution.

The system automatically identifies the sets of interacting goals of a plan by partially ordering the totally ordered solution found. The connected components of the partially ordered plan determine the independent fragments of the case each corresponding to a set of interacting goals. Each case is multiply indexed by these different sets of interacting goals.

When a new problem is presented to the system, the retrieval procedure must match the new initial state and goal statement against the indices of the cases in the case library.

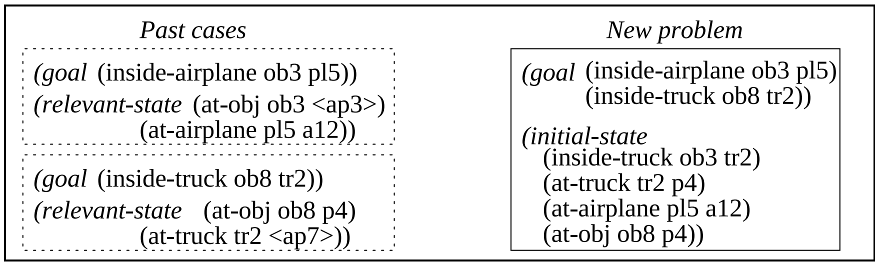
The retrieval algorithm focuses on retrieving past cases where the planner experienced equivalent goal interactions and has a reasonable match between initial states expecting therefore to achieve a large reduction in the new planning search effort.

3.3 Reuse: Replay of Multiple Planning Episodes

PRODIGY/ANALOGY can construct a new solution from a set of guiding cases as opposed to a single past case. Complex problems may be solved by resolving minor interactions among simpler past cases.

Consider the logistics transportation domain. In this domain packages are to be moved among different places, by trucks and airplanes. The example below is simple for the sake of a clear illustration of the replay procedure. Extensive empirical results on PRODIGY/ANALOGY have shown that the system scales well in problem complexity [Vel94].

Figure 2 shows a new problem and two past cases selected for replay. The cases are partially instantiated to match the new situation. Further instantiations occur while replaying.



```
(goal (inside-airplane ob3 pl5))
(relevant-state (at-obj ob3 <ap3>)
                (at-airplane pl5 a12))

(goal (inside-truck ob8 tr2))
(relevant-state (at-obj ob8 p4)
                (at-truck tr2 <ap7>))

(goal (inside-airplane ob3 pl5)
      (inside-truck ob8 tr2))
(initial-state
   (inside-truck ob3 tr2)
   (at-truck tr2 p4)
   (at-airplane pl5 a12)
   (at-obj ob8 p4))
```

Figure 2: Instantiated past cases cover the new goal and partially match the new initial state. Some of the case variables are not bound by the match of the goals and state.

Figure 3 shows the replay episode to generate a solution to the new problem. The new situation is shown at the right side of the figure and the two past guiding cases at the left.

The transfer occurs by interleaving the two guiding cases, performing any additional work needed to accomplish remaining subgoals, and skipping past work that does not need to be done. In particular, the case nodes `cn3'` through `cn5'` are not reused, as there is a truck already at the post office in the new problem. The nodes `n9-14` correspond to unguided additional planning done in the new episode.[1] At node `n7`, PRODIGY/ANALOGY prunes out an alternative operator, namely to load the truck at any airport, because of the recorded past failure at the guiding node `cn2'`. The recorded reason for that failure, namely a goal-loop with the `(inside-truck ob8 tr2)`, is validated in the new situation, as that goal is in the current set of open goals, at node `n6`. Note that the two cases are merged using a bias to postpone additional planning needed. Different merges are possible.

[1]Note that extra steps may be inserted at any point, interrupting and interleaving the past cases, and not just at the end of the cases.

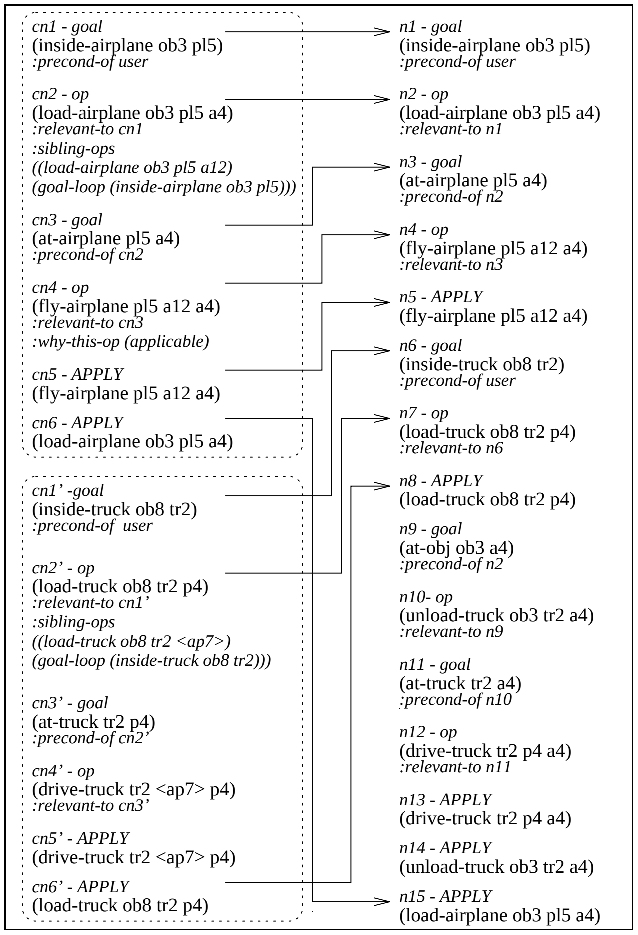


Figure 3: Derivational replay of multiple cases.

4 CAPLAN/CBC: Replanning in the Space of Plans

CAPLAN/CBC [MAPW95] is a generic case-based reasoning system that supports the integration of domain-specific reasoners in a modular way. This integration was inspired by practical needs in the domain of process planning. The overall architecture is built on top of CAPLAN [Web94], a plan-space nonlinear planner [MR91].

Intuitively, during the planning process, state-space planners transform states in contrast to plan-space planners that refine partial-ordered plans. The purpose is to obtain a plan which all goals are solved and that contains no conflicts. Conflicts may occur due to the partial-order between the plan steps. For example, in the domain of process planning different plan steps may require the same type of tool. A conflict takes place, when there are not enough tools of the required type available. For solving this conflict, some of the steps must be ordered to ensure that every tool is used one at the time. This kind of planners are also called least-commitment planners, as no order is introduced in the plan, unless it is necessary (i.e., for solving a conflict). Studies have been made for indicating, in which situations is better to search in the space of plans rather than in the space of states and vice versa.

4.1 Case-based problem-solving in CAPLAN/CBC

For solving a problem CAPLAN/CBC analyzes its description by using a domain-specific reasoner. In the domain of process planning it corresponds to a feature-based CAD reasoner. This reasoner is used for detecting geometrical interactions in rotatory symmetrical workpieces. The retrieval procedure uses this information to improve its accuracy without reducing the performance of the overall case-based solution process [MAH95]. The cases retrieved are then replayed in the actual situation. If the obtained plan is incomplete (i.e., there are unsolved goals), the generic planner is used to complete the plan. For the rest of this section we will concentrate on the replay mechanism of CAPLAN/CBC.

4.2 The replay phase in CAPLAN/CBC.

For solving a new problem CAPLAN/CBC reuses selected cases by following their derivation paths [Vel94] and replaying their decisions in the new situation. Decisions taken when replaying a case may need to be revised when completing the solution. Revising decisions implies backtracking which can be very expensive. For avoiding unnecessary backtracking steps, CAPLAN/CBC stores in the cases not only the valid decisions but also the justifications of decisions[2] that were redrawn later during the problem solving episode.

Process planning is concerned with the manufacturing of mechanical workpieces. Initialy a piece of raw material and the description of a workpiece are given. Descriptions of the cutting tools and clamping material available are also given. The problem is to remove layers of raw material in order to obtain the workpiece. Figure 4 shows an intermediate stage during this process. The grid area corresponds to the portion of raw material that still needs to be removed. The parts presented there correspond to the two sides of a workpiece, two ascending outlines and a so-called undercut. Undercuts always can be decomposed in two parts (labeled *u-cut1* and *u-cut2*). In this figure a left and a right cutting tools are also shown , labeled *A* and *B* respectively. For manufacturing the undercut the workpiece needs to be clamped from an ascending outline, for example *Ascend-1.* The left tool is used for removing the left part (i.e., *u-cut1*). For removing the right part (i.e., *u-cut2*) there are three possibilities: (1) to use the right tool, (2) to clamp the workpiece from the outline *Ascend-2* and use the same left tool again or (3) to clamp the workpiece from *Side2* and use the left tool. The last possibility requires that there is a perforation on *Side2.* Since in the example there is only a perforation on *Side 1*, it will be discarded if considered by the planner.

[2]For handling the justifications CAPLAN is built on the generic REDUX architecture [Pet91].

The left side of figure 5 outlines a plan for manufacturing the workpiece shown in figure 4, under the supposition that there is a left and a right cutting-tools available. Dashed boxes represent plan-steps and the arcs pointing downwards indicate the partial-order for performing them. This plan states that for removing *u-cut1*, the workpiece must be clamp from *Ascend-1* and the left tool must be used. After that, *u-cut2* is removed by using the right tool. This plan contains also the information that clamping from *Side2* failed because it does not have any perforation (node labeled *R*).

Suppose now that a new problem is given consisting of the same workpiece, but this time there is only a left tool available. For solving this problem the plan obtained with the two tools will be reused, as illustrated in figure 5. The horizontal arrows show the decisions of the case that are replayed in the new situation. Particularly the decisions concerning the manufacturing of *u-cut1* can be replayed in the new situation. However, the decision concerning the manufacturing of *u-cut2* cannot be replayed, since in the new situation there is no right tool available. As a result, a rejected decision is created (node labeled *S*). The rejection of the operator *clamping from Side-2* is replayed (node labeled *R'*), as in the new situation *Side-2* has still no perforation.

Once the replay of cases is finished, the remaining goals need to be solved by the generative planner. As stated before, the key issue is that even when the generative planner (CAPlan) needs to be used, performing unnecessary backtracking will be avoided. Particularly, for solving the goal corresponding to manufacturing *u-cut2*, CAPlan will not pursue to use the right tool, neither to clamp from *Side2*. Instead, it will select to clamp the workpiece from *Ascend.2* (arc labeled *P*), which is the right choice.

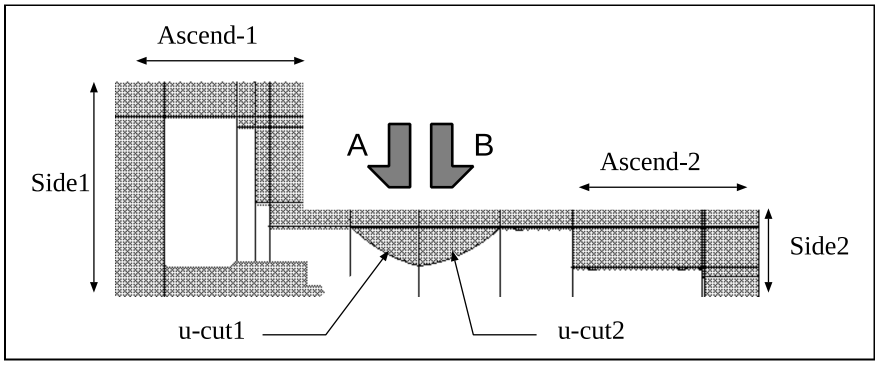


Figure 4: Half display of a workpiece and two cutting tools.

5 PARIS: Reuse at Different Levels of Abstraction

Paris (Plan Abstraction and Refinement in an Integrated System) [BW95] is a domain independent case-based planning system which allows the flexible reuse of planning cases

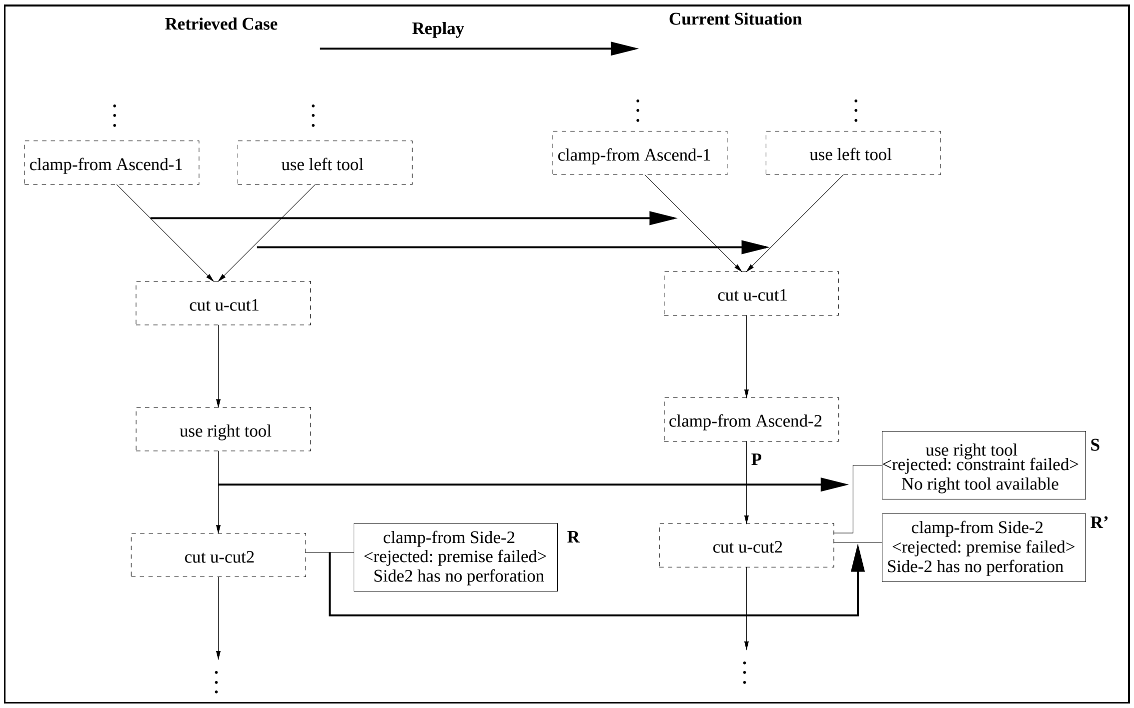


Figure 5: Example of replay in CAPLAN/CBC.

by abstraction and refinement. This approach is mainly inspired by the observation that reuse of plans must not be restricted to a single description level. In domains with a high variation in the problems, the reuse of past solutions must be achieved at various levels of abstraction.

In PARIS, planning cases given at the concrete level are abstracted to several levels of abstraction which leads to a set of *abstract cases* that are stored in the case-base. Case abstraction is done automatically in the retain phase of the CBR-process model. When a new problem must be solved, an abstract case is retrieved whose problem description matches the current problem exactly at an abstract level. In the subsequent reuse phase, the abstract solution is refined, i.e., the details that are not contained in the abstract case are added to achieve a full solution of the problem. This refinement is done by a generative planner that performs a forward directed state space search. We now explain how abstract cases are constructed and reused in this case-based reasoning process.

5.1 Case Abstraction

Case abstraction means reducing the level of detail contained in the problem description and in the solution of a case, i.e., an abstract case contains less operators and less states than the concrete case. Furthermore, abstract operators and states are described using

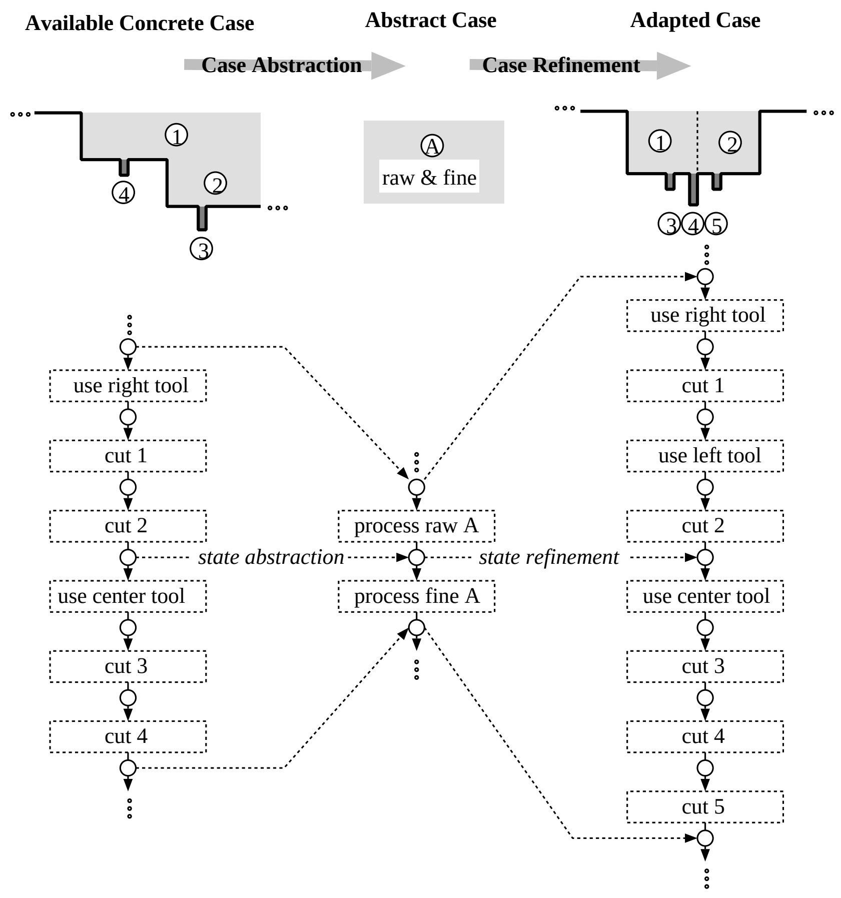


Figure 6: Example of generating and refining abstract cases.

more abstract terms which typically requires also a reduced number of predicates. In general, a change of the complete representation language between abstraction levels may be required to achieve meaningful und useful abstractions in a domain. Therefore, PARIS assumes that in addition to the concrete planning domain, an *abstract planning domain* is contained as part of the general knowledge. It contains a set of *abstract operators* together with a set of rules that describe different ways of abstracting concrete states. For example in the domain of planning rotary symmetric workpieces the concrete domain contains opertors and predicates to describe the detailed contour of workpieces and individual cut operations that must be performed. The abstract domain abstracts from the detailed contour and represents larger units, called complex processing areas, together with the status of their processing.

Figure 6 presents an example of the relationship between a concrete case and an abstract case. The left side shows a section of a concrete case, depicting how a step-like contour with two grooves is manufactured by a sub-plan consisting of 6 steps. The abstract case, shown in the middle of this figure, abstracts from the detailed contour and just represents a complex processing area named A that includes raw and fine elements. The corresponding abstract plan contains 2 abstract steps: processing in a raw manner and processing in a fine manner. The arrows between the concrete and the abstract case show how concrete and abstract states correspond. Each abstract state is derived from one of the existing concrete states (state abstraction). However, not all concrete states are abstracted. Some concrete states are skipped because they are considered a detail. As a byproduct of this state abstraction, a sequence of concrete operators is abstracted to a single abstract operator. Note that the above explained kind of case abstraction is performed by an automatic procedure in Paris as part of the retain-phase of the CBR-cycle.

5.2 Retrieval and Reuse of Abstract Cases

When a new problem must be solved, an abstract case is retrieved which exactly matches the current problem at the abstract level. If several matching abstract cases are contained in the case-base, the retrieval procedure (using an abstraction hierarchy for indexing) selects the case that is located at the lowest level of abstraction. The motivation for this is that for more concrete cases, less work has to be done during refinement to achieve a complete solution. This corresponds to a similarity measure that ranks two cases more similar the lower the level of abstraction is, on which the problems are identical [BPW94].

During the reuse phase, the abstract solution contained in the retrieved case must be refined to become a fully detailed complete solution. The right side of Figure 6 shows an example of such a refinement. While the contour of the two workpieces differs drastically at the concrete level, the abstract case matches exactly because the 5 atomic contour elements in the new problem can be abstracted to a complex processing area with raw and fine elements. The abstract opertors of the abstract case are then used to guide the state-based planner to find a refined solution to the problem. Each abstract state is used as a sub-goal in the planning process. In the portion of the case shown in Figure 6, the abstract operator *process raw A* is refined to a sequence of four concrete steps which manufacture area 1 and 2. The next abstract operator is refined to a four-step sequence which manufactures the grooves 3, 4, and 5.

Note that PARIS allows the *reuse of problem decompositions* at different levels of abstraction. Abstract plans decompose the original problem into a set of much smaller subproblems. If these problems are small enough and mostly independent from each other, the underlying planner is able to solve them without stepping into the intractability problem.

6 Related work

Besides the systems discussed so far, several other case-based planning systems have been developed in USA and Europe.

PRIAR [KH92] reuses plans in a generative nonlinear hierarchical planner. Following the derivational analogy philosophy, PRIAR uses the *validation structure* of a plan, which represents the dependencies among the plan steps, for retieval and reuse of planning cases, but additionally employs domain independent strategies for solution adaptation. However, PRIAR does not address the problem of how to organize a case base efficiently.

The Deja-Vu system [SK94] uses a hierarchical case-based reasoning approach which is similar to PARIS in that abstract cases are used for problem decomposition. However, Deja-vu does not include a generative planner but uses multiple cases and transformational adaptation for refinement.

The MRL system [Koe94] is a domain-independent case-based planner. It reuses plans based on a deductive planning approach. This enable to compare problem descriptions on the semantic level instead of pure matching. In contraposition to the systems presented here, MRL requires the domain to be represented in formal logic. Further, for indexing the cases a formalization in terminological logic is also required.

Similar to the systems presented in this paper, the PLAKON system [GC93] is a domain-independent configuration system that combines generic and case-based approaches [Pfi93]. The domain is represented through a concept hierarchy. Cases correspond to instances of the concept hierarchy. The case-based component obtains a partial solution by combining multiple parts of cases provided that they do not overlap. The generative configuration system is then used to complete the partial solution.

7 On to real applications

Several case-based planning systems (incuding the ones reported here) already address important problems that occur in real planning applications. Experimental investigations have shown a drastic speedup (factor 1000 and more) of problem solving through case-based planning compared to pure generative planning approaches in many different domains. However, several important questions are still open such as

- knowledge engineering for case-based planning,
- user interaction particularly during the reuse phase,
- maintaining the quality of plans (e.g., the cost and resource usage during plan execution) during reuse, and finally
- the integration into an industrial environment.

We are quite optimistic that these problems can be solved in the future and propose to address them, by continuing the application oriented research strategy followed so far. Then, we can expect fielded applications of case-based planning in the near future.

References

[AP94] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundation issues, methodological variations and system approaches. *AI-Communications*, 7(1):pp 39–59, March 1994.

[BPW94] R. Bergmann, G. Pews, and W. Wilke. Explanation-based similarity: A unifying approach for integrating domain knowledge into case-based reasoning. In M.M. Richter, S. Wess, K.D. Althoff, and F. Maurer, editors, *Topics in Case-Based Reasoning*, volume 837 of *Lecture Notes on Artificial Intelligence*, pages 182–196. Springer, 1994.

[BW95] R. Bergmann and W. Wilke. Building and refining abstract planning cases by change of representation language. *Journal of Artificial Intelligence Research*, 3:53–118, 1995.

[Car86] Jaime Carbonell. Derivational analogy : A theory of reconstructive problem solving and expertise acquisition. *Machine Learning*, 2, 1986.

[CFS94] Padraig Cunningham, Donal Finn, and Sean Slattery. Knowledge engineering requirements in derivational analogy. In Stefan Wess, Klaus-Dieter Althoff, and Michael M. Richter, editors, *Topics in Case-Based Reasoning*, volume 1, pages 234–245, 1994.

[CKM91] J.G. Carbonell, C.A. Knoblock, and S. Minton. Prodigy: An integrated architecture for planning and learning. In K. VanLehn, editor, *Architectures for Intelligence*, pages 241–278. Lawrence Erlbaum Associates, Publishers, 1991.

[GC93] Andreas Günter and Roman Cunis. PLAKON - ergebnisse einer entwicklung. *KI*, (1):51–56, 1993.

[Ham86] Kristian Hammond. Chef: a model of case-based planning. In *Procceedings of American Asociation of Artificial Intelligence, AAAI-86*, 1986.

[KH92] S. Kambhampati and J.A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258, 1992.

[Koe94] J. Koehler. Flexible plan reuse in a formal framework. In *Current Trends in AI Planning*, pages 171–184. IOS Press, Amsterdam, Washington, Tokio, 1994.

[MAH95] H. Muñoz-Avila and J. Hüllen. Retrieving relevant cases by using goal dependencies. In *Case-Based Reasoning Research and Development, Proceedings of the 1st International Conference (ICCBR-95)*, number 1010 in Lecture Notes in Artificial Intelligence. Springer Verlag, 1995.

[MAPW95] H. Muñoz-Avila, J. Paulokat, and S. Wess. Retrieving relevant cases by using goal dependencies. In M. Keane, J.P. Halton, and M. Manago, editors, *Advances in Case-Based Reasoning. Selected Papers of the 2nd European Workshop (EWCBR-94)*, number 984 in Lecture Notes in Artificial Intelligence, 1995.

[MR91] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of AAAI-91*, pages 634–639, 1991.

[Pet91] Ch. Petrie. *Planning and Replanning with Reason Maintenance.* PhD thesis, University of Texas at Austin, CS Dept., 1991. (MCC TR EID-385-91).

[Pfi93] Kai Pfitzner. *Fallbasierte Konfigurieren technischer Systeme.* PhD thesis, Universität Hamburg, 1993.

[SK94] B. Smyth and M.T. Keane. Retrieving adaptable cases. In Stefan Wess, Klaus-Dieter Althoff, and Michael M. Richter, editors, *Topics in Case-Based Reasoning*, 1994.

[Vel94] M. Veloso. *Planning and learning by analogical reasoning.* Number 886 in Lecture Notes in Artificial Intelligence. Springer Verlag, 1994.

[Wal77] R. Waldinger. Achieving several goals simultaneously. In *Machine Intelligence*, volume 8. Ellis Horwood Limited, 1977.

[Web94] F. Weberskirch. Realisierung eines nichtlinearen Planungssystems zur Unterstützung der Arbeitsplanerstellung bei der computerintegrierten Fertigung (CIM). Diplomarbeit, Universität Kaiserslautern, 1994.