Efficient Retrieval of Abstract Cases for Case-Based Planning

Ralph Bergmann
University of Kaiserslautern
Centre for Learning Systems and Applications (LSA)
PO-Box 3049
D-67653 Kaiserslautern, Germany
bergmann@informatik.uni-kl.de

Abstract

Recently, the use of abstraction in *case-based reasoning* (CBR) is getting more and more popular. The basic idea is to supply a CBR system with cases at many different levels of abstraction. When a new problem must be solved, one (or several) 'appropriate' concrete or abstract case are retrieved from the case base and the solution that the case contains is reused to derive a solution for the current problem, e.g. by filling in the details that a retrieved case at some higher level of abstraction does not contain. A major problem that occurs when using this approach is, that for a given new problem, usually several cases, e.g., from different levels of abstraction could be reused to solve the new problem. Choosing a wrong abstract case can slow down the problem solving process or even prevents the problem from being solved.

This paper presents a new approach for selecting abstract cases from a case base within in the context of the case-based planning system Paris. Based on a general analysis of the efforts involved in abstraction-based CBR, the new retrieval technique is developed. Cases are organized in an *abstraction hierarchy* that is constructed during the retain phase. Abstract cases at higher levels of abstraction are located above abstract cases at lower levels. The leaf nodes of this hierarchy contain the concrete cases. Further, this abstraction hierarchy is pruned based on a cost model of the expected problem solving time in order to optimze the retrieval structure. Several experiments conducted in the domain of manufacturing planning shows clearly the advantage of the presented retrieval approach.

Introduction

In AI, the use of abstraction was originally inspired by human problem solving (cf. (Minsky, 1963)) and has already been successfully used in different fields such as theorem proving, model-based diagnosis or planning (Giunchiglia and Walsh, 1992). Recently, some researchers have started to investigate the use of abstraction in *case-based reasoning* (CBR). In case-based reasoning (Aamodt and Plaza, 1994; Kolodner, 1993) problems are solved on the basis of previous experience (called *cases*) which is stored in a *case base* in the form of problem-solution-pairs. The typical case-based reasoning problem solving cycle is as follows: A case that is similar to the current problem is *retrieved* from the case base. Then, the solution contained in this retrieved case is *reused* to solve the new problem, i.e., the solution is adapted in order to come to a solution of the current problem. Thereby, a new solution is obtained and presented to the user who can verify and possibly *revise* the solution. The revised case (or the experience gained during the case-based problem solving process) is then *retained* for future problem solving, e.g., the case can be stored in the case base.

The basic idea behind the use of abstraction in CBR is to supply a CBR system with cases at many different levels of abstraction. These cases are stored in a case base for being reused to solve new problems. When a new problem must be solved, one (or several) 'appropriate' concrete or abstract case have to be retrieved from the case base and the solution that the case contains is reused to derive a solution for the current problem, e.g. by filling in the details that a retrieved case at some higher level of abstraction does not contain. For these kind of approaches the terms *hierarchical case-based reasoning* (Smyth and Cunningham, 1992), *stratified case-based reasoning* (Branting and Aha, 1995), and *reasoning with abstract cases* (Bergmann and Wilke, 1995; Bergmann, 1996; Bergmann and Wilke, 1996) have been used so far.

A major problem that occurs when using this approach is, that for a given new problem, usually several cases, e.g., from different levels of abstraction could be reused to solve the new problem. However, since different cases describe previous solutions at a different level of detail or use different abstract views, they could be more or less suited for solving a new problem. Choosing a wrong abstract case can slow down the problem solving process or even prevents the problem from being solved. Therefore, abstract cases must be selected carefully. This paper presents an approach for selecting abstract cases from a case base. This work has been pursued in the context of the PARIS case-based planning system (Bergmann and Wilke, 1995; Bergmann, 1996). The next two sections briefly introduce the idea of case-based reasoning using abstraction and the concrete PARIS system. Section 4 describes a new approach for selecting appropriate abstract cases from a case base and section 5 presents an experimental evaluation of this approach. Finally, section 6 summarizes the results.

Reasoning with Abstract Cases

While cases are usually represented and reused on a single level, abstraction techniques enable a CBR system to reason with cases at several levels of abstractions. Firstly, this requires the introduction of several distinct levels of abstraction.

Each level of abstraction allows the representation of problems, solutions, and cases as well as the representation of general knowledge that might be required in addition to the cases. Usually, levels of abstraction are ordered (totally or partially) through an abstraction-relation, i.e., one level is called *more abstract* than another level. A more abstract level is characterized through a reduced level of detail in the representation, i.e., it usually consists of less features, relations, constraints, operators, etc. Moreover, abstract levels model the world in a less precise way, but still capture certain, important properties. In traditional hierarchical problem solving (e.g., ABSTRIPS (Sacerdoti, 1974)), abstraction levels are constructed by simply dropping certain features of the more concrete representation levels. However, it has been shown that this view of abstraction is too restrictive and representation dependent (Bergmann and Wilke, 1995; Holte et al., 1995) to make full use of the abstraction idea. In general, different levels of abstraction require different representation languages, one for each level. Abstract properties can then be expressed in completely different terms than concrete properties.

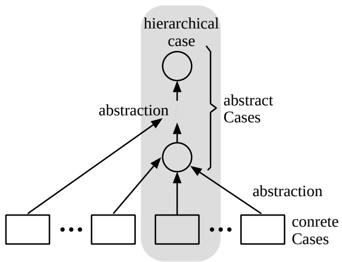

Figure 1: Different kinds of cases

Based on the level of abstraction, we can distinguish between two kinds of cases: *concrete cases* and *abstract cases*. A *concrete case* is a case located at the lowest available level of abstraction. An *abstract case* is a case represented at a higher level of abstraction. The case-base usually stores abstract and concrete cases. If several abstraction levels are given (e.g., a hierarchy of abstraction spaces), one concrete case can be abstracted to several abstract cases, one at each higher level of abstraction. Such an abstract case contains less detailed information than a concrete case. On the other hand several concrete cases usually correspond to a single abstract case (see Fig. 1). These concrete cases share the same abstract description; they only differ in the details.

During case-based reasoning, a case at some level of abstraction is *retrieved* (see Fig. 2). Then, the solution of the retrieved case is possibly *adapted*, i.e., the solution is modified but the level of abstraction is not changed. As a result, an abstract solution to the new problem is obtained. Then, this adapted abstract solution is *refined* to a concrete solution to the problem. During this refinement process (which can be either done by a generative hierarchical problem solver or in a case-based manner, see (Bergmann and Wilke, 1996)) the details that the abstract case does not contain are added. The refined solution is then presented to the user. For her/him it is transparent, whether the solution presented by the system stems directly from a matching concrete cases or whether the solution is obtained through the refinement of an abstract case.

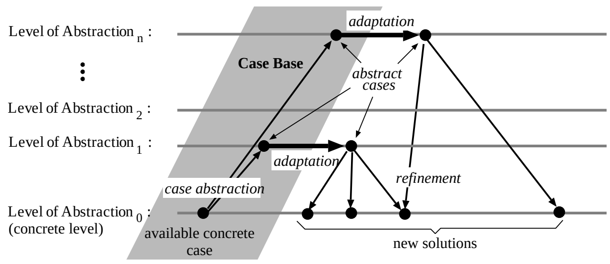

Figure 2: Adaptation of abstract cases

In (Bergmann and Wilke, 1996) we have shown that in general abstraction can support the CBR process

- by reducing the complexity of the case representation,
- by reducing the size of the case base,
- by increasing the flexibility of solution reuse, and
- by providing a means for solution adaptation.

PARIS: Using abstraction in case-based planning

Now, we briefly describe a concrete case-based reasoning system, called PARIS[1] that uses abstraction for case-based planning. A detailed description of the system can be found in (Bergmann and Wilke, 1995; Bergmann, 1996; Bergmann and Wilke, 1996). PARIS was designed as a generic (i.e., domain independent) case-based planning system but with a particular area of application domains in mind: manufacturing planning in mechanical engineering. Here, a plan is a sequence of manufacturing steps that must be performed in order to produce a particular mechanical workpiece. Planning in this domain can be viewed as classical STRIPS (Fikes and Nilsson, 1971) planning: a (manufacturing) *operator* transforms a certain *state* (current workpiece) into a successor state (workpiece after the manufacturing step). The planning task is to find a sequence of operators which transform a mold (initial state) into the desired workpiece (goal state). Since finding such a plan is known to be a NP-complete problem, several case-based approaches have been developed already that allow to make use of additional knowledge (in the form of previous cases) during planning (Bergmann et al., 1998).

The task of a case-based planning system in this domain is to produce a manufacturing plan (solution) for a new workpiece (problem) by reusing previous manufacturing cases. We have identified, a set of CBR specific requirements that are important in this domain (Bergmann, 1996):

- ability to cope with vast space of solution plans,
- construction of correct solutions,
- flexible reuse due to large spectrum of target problems,
- processing of highly complex cases, and
- only concrete planning cases available (e.g. in archives of a company).

Abstract Planning Cases

In PARIS,

- *abstract planning cases* are *generated automatically*,
- *stored* together with the concrete cases in the case-base,
- *used for indexing* during retrieval,
- and they are *adapted* and *refined automatically* during the reuse-phase.

Different levels of abstraction are realized by different *planning domains*, each of which consists of its own set of operators and its own representation of states (i.e., workpiece descriptions in our domain). Abstract operators and states are described using more abstract terms than concrete operators. While a concrete planning case consists of a sequence of operators from the concrete level, an abstract planning case consists of a sequence of operators form the abstract level. Each abstract operator that occurs in an abstract case stands for a sub-sequence of concrete operators of the corresponding concrete case. In (Bergmann and Wilke, 1995) a comprehensive formal model of case abstraction is explained in detail.

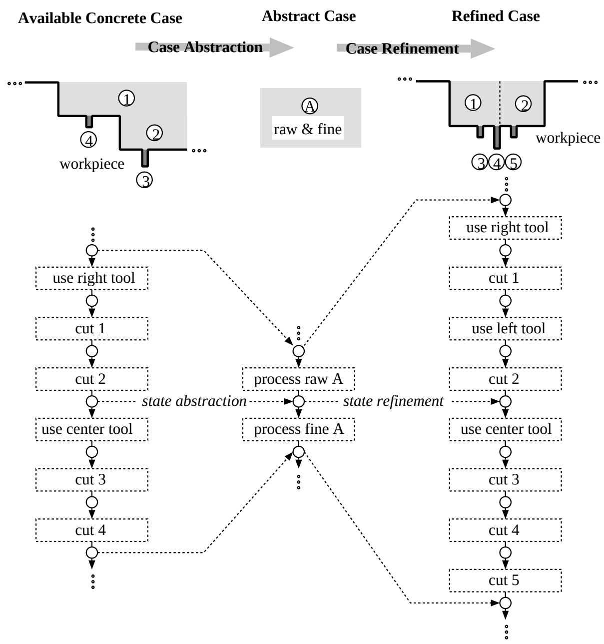

Figure 3: Example of generating and refining abstract cases.

Example

Figure 3 presents an example of the relationship between a concrete case and an abstract case in the manufacturing domain. Here, the concrete planning domain contains operators and predicates to describe the detailed contour of workpieces and individual manufacturing operations (e.g., cutting a certain area) that must be performed. The abstract domain abstracts from the detailed contour and represents larger units, called complex processing areas, together with the sta-

[1] PARIS stands for **p**lan **a**bstraction *and* **r**efinement *in an* **i**ntegrated **s**ystem.

tus of their processing (e.g. not processed, roughly processed, or completed). The left side of Figure 3 shows a section of a concrete case, depicting how a step-like contour with two grooves is manufactured by a subplan 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 (step-like contour) and fine (grooves) elements. The corresponding abstract plan contains 2 abstract steps: processing in a raw manner and processing in a fine manner.

Acquisition of Abstract Cases

Engineering departments which develop manufacturing plans manually, usually record them (e.g., in a database) for documentation purposes. These plans contain all details necessary for manufacturing the workpiece; they represent concrete cases only. Because manual abstraction of such cases seems to be a tremendous effort, abstract cases are generated automatically from a given concrete case. For this purpose, a domain-independent case abstraction algorithm has been developed. Given a concrete and an abstract planning domain, this algorithm computes abstract cases from a given concrete case.

Refinement of abstract cases

In PARIS an abstract solution contained in an abstract case is refined automatically to a concrete level solution. The right side of Figure 3 shows an example of such a refinement. Please note that the contour of the two workpieces differs drastically at the concrete level. However, 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. During refinement, the abstract operators of the abstract case are used to guide the generative planner to find a refined solution to the problem. Therefore, each abstract state is used as a kind of subgoal. The planner starts with the concrete initial state from the new problem description and searches for a sequences of concrete operators leading to a concrete state that can be abstracted to the first abstract state in the abstract case. The resulting operator sequence is a refinement of the first abstract operator. All remaining abstract operators are then sequentially refined in the same way. In the portion of the case shown in Figure 3, 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.

We can seen that the abstract case decomposes the original problem into a set of much smaller subproblems. Due to this decomposition, the effort for problem solving is drastically reduced compared to a pure from scratch problem solver.

Adaptation of Abstract Cases

PARIS also performs solution adaptation. For that purpose, a case (either abstract or concrete) is *generalized* into a generalized case (similar to a schema or script). Such a generalized case does not only describe a single problem and a single solution but a *problem class* together with a *solution class*. Such classes are realized by introducing *variables* into the initial and goal state as well as into the plan. Additionally, a generalized case contains a *set of constraints* that restricts the instantiation of these variables. PARIS includes an algorithm for automatically generalizing concrete or abstract cases into schemas (Bergmann, 1996) by applying explanation-based generalization (Mitchell et al., 1986). Adaptation with generalized cases is done by finding an instantiation of the variables such that instantiated generalized case matches the target problem to be solved. In PARIS, matching (similarity assessment) and adaptation is done by a constraint satisfaction problem solver. The effort for solving this constraint satisfaction task can be very high: in the worst case it is exponential in the number of constraints and the size of the problem class. Typically, the representations at a higher level of abstraction are less complex than representations at lower levels. Consequently, generalized cases at higher levels of abstraction contain less constraints and the problem class is composed of a small number of prepositions. Therefore, adaptation of abstract cases requires less effort than adaptation of concrete cases.

Selecting Abstract Cases During Retrieval

Within the framework of using abstraction in case-based reasoning, a major problem is to retrieve appropriate cases from the case base. Selecting the wrong case can slow down problem solving or can even prevent a problem from being solved. Therefore, we now focus on how to retrieve appropriate cases.

General Considerations

To enable an efficient overall case-based problem solving two major factors must be considered during retrieval. Since the overall case-base problem solving effort is the sum of the retrieval effort and the reuse effort,

- a case (either abstract or concrete) must be retrieved that can be reused to solve the new problem by spending as little effort as possible for reuse and

- the effort for the retrieval step itself should be as low as possible.

For case-based reasoning systems that make use of abstraction, the effort for reuse can be again divided into the effort for adapting an abstract solution and the effort for refining the adapted abstract solution to a concrete solution (see also Fig. 2). If we take a closer look at these efforts, we observe that

- the effort for adaptation increases, the lower the level of abstraction of the case is. The reason for this is that cases at lower levels include more details in the representation all of which must possibly be modified consistently to come to a solution for the new problem.
- On the other hand, the effort for refinement decreases the lower the level of abstraction of the case is. The reason for this is that in more concrete cases less details are missing that must be generated during the refinement.

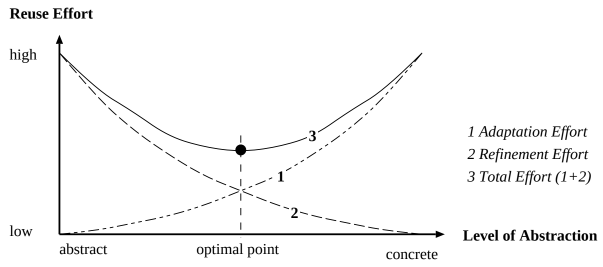

Figure 4: Reuse effort depending on level of abstraction.

Figure 4 shows these efforts depending on the level of abstraction. From these very general considerations it becomes immediately clear that the best case for reuse is usually located at some middle level of abstraction; neither the most abstract nor the most concrete case should be reused.

Besides the selection of an abstract case that can be reused efficiently, the retrieval process itself must be efficient as well. A trade-off between the objective to find the best case and the objective of minimizing the retrieval time exists as depicted in Figure 5 (adapted from (Veloso, 1992)). As the number of cases visited during retrieval increases, more time must be spent for retrieval (see curve 2) but better cases resulting in a shorter adaptation time will be found (see curve 1). Up to 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 increases again if more cases are visited, because the possible gain through finding better cases does not outweigh the effort of finding them.

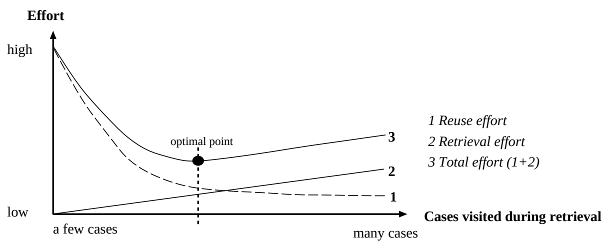

Figure 5: Trade-off between retrieval effort and reuse effort.

Hierarchical organization of the case base

The enable an efficient retrieval, the organization of the case base plays an important role. In Paris, abstract cases located at different levels of abstraction are used as hierarchical indexes to those concrete (or abstract) cases that contain the same kind of information but at a more detailed level. For this purpose, an *abstraction hierarchy* is constructed during the retain phase, in which abstract cases at higher levels of abstraction are located above abstract cases at lower levels. The leaf nodes of this hierarchy contain concrete cases (see Fig. 6).

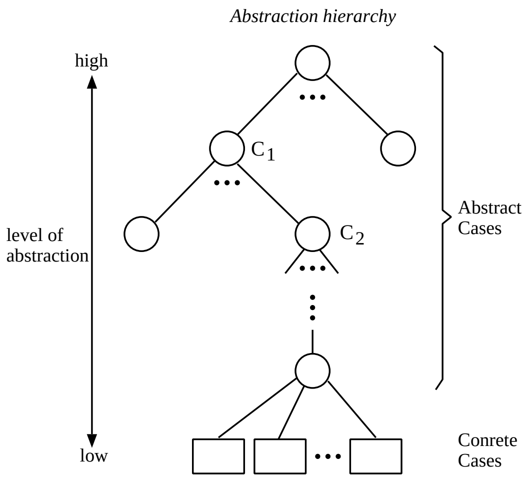

Figure 6: Abstraction hierarchy for indexing cases.

During retrieval, this hierarchy is traversed top-down. The abstract case at the node of the hierarchy is examined in order to find out whether the abstract solution it contains can be adapted to become an abstract solution to the current problem. If this is the case, the successor nodes of the hierarchy are examined in a predefined order. The search through this hierarchy proceeds at the next level with the first successor node that contains an abstract case that is adaptable for the current problem.

As explained so far, this retrieval strategy has two major shortcomings:

- It usually ends up with an applicable case at a lowest level possible level of abstraction. Hence, we do not hit the optimal point in Fig. 4, but we are too much on the right side of curve 3.

- It has to examine many cases to decide whether they are adaptable, particularly also case at lower levels of abstraction. Hence, we do not hit the optimal point in Fig 5, but again we are too much on the right side of curve 3.

As a consequence of these two shortcomings, this approach spends a lot of effort for retrieval and selects a case that requires high adaptation effort.

Optimized Retrieval

In order to cope with these problems, we have to modify the approach in a way that cases that require a high adaptation effort are not selected by the retrieval and are not even be examined during retrieval. This can be efficiently realized through a pruning of the abstraction hierarchy, i.e., deleting some branches of the tree. If a certain branch of the tree is removed (together with the respective concrete and possibly abstract cases) the abstract cases that remain accessible can still cover the set of target problems previously covered by the deleted case. However, not all details are present any more. During reuse they must therefore be reconstructed by the generative planner. Consequently, pruning of the abstraction hierarchy has two contrary effects on the overall problem solving time:

- Since the detailed parts of the solution are not available any more, the refinement effort increases, but the adaptation effort decreases.

- Since the number of cases that must be inspected during retrieval is reduced, the retrieval effort is also reduced.

Now, the question arises which parts of the hierarchy should be pruned? This determines how far we move the points in Fig 4 and 5 to the left side. We aim, of course, getting close to the optimal point.

Cost Model

The PARIS system makes use of an elaborated cost model for determining the expected cost/benefit (retrieval effort + reuse effort) of pruning certain parts of the abstraction hierarchy (and the case base). This cost model is based on an estimation of the expected value of the case-based problem solving time.

$$T_{EW}(\mathcal{P}) = \sum_{p \in \mathcal{P}} Pr(p) \cdot T_L(p).$$

Here, $\mathcal{P}$ is the set of all problems that possibly must be solved, $Pr(p)$ is the probability that the problem p must be solved, and $T_L(p)$ is the overall case-based problem solving time for solving p given a particularly pruned abstraction hierarchy. The idea is to compute an estimation of this expected problem solving time for different pruned abstraction hierarchies and to select the hierarchy with lowest expected value. In order to compute such an estimation, the assumption must be made that the distribution of the cases from the cases base is equal to the distribution of the problems to be solved in the future. Given this assumption, the probabilities $Pr(p)$ can be determined directly from the case base.

In order to compute an estimation for $T_L(p)$ for a given problem and an abstraction hierarchy, for each node K in the hierarchy an estimation is required for the following values:

- $T_a(K,p)$: The time required for adapting the abstract case at node K to the problem p. We assume that the time for determining whether an abstract case is adaptable is the same than the time for doing the actual adaptation. This is at least the case for the adaptation approach used in PARIS.

- $T_v(K,p)$: The time required for refining the adapted abstract solution from the case at node K to a solution of p.

- $P_j(K,p)$: Let K have the not-pruned successor nodes $K_1, \ldots, K_r$. Then $P_j(K,p)$ for $j = 1..r$ is the probability that the abstract case at the node K_j is adaptable for p and the abstract cases at the nodes $K_1, \ldots, K_{j-1}$ are not adaptable for p given that the abstract case at node K is adaptable for p.

Now, it can be shown that the expected value for the case-based problem solving time can be computed recursively as follows:

$$T_{EW}(K,p) = \begin{cases} T_v(K,p) & K \text{ is leaf node in the hierarchy,} \\ \\ \sum_{i=1}^{r} T_{EW}(K_i,p) \cdot P_i(K,p) + & K \text{ has of the not-pruned successors} \\ \sum_{i=1}^{r} T_a(K_i,p) \cdot (1 - \sum_{j=1}^{i-1} P_j(K,p)) + & K_1, \ldots, K_r \\ T_v(K,p) \cdot (1 - \sum_{i=1}^{r} P_i(K,p)) & \end{cases}$$

In order to compute this expected value for a pruned hierarchy, estimations for $T_a(K,p)$, $T_v(K,p)$, $P_j(K,p)$ are necessary. Estimations for $T_a(K,p)$ and $T_v(K,p)$ can be obtained by measuring the average adaptation time and average the refinement time when using the abstract case at node K during problem solving. Estimations for $P_j(K,p)$ can be obtained from the information which previous problem from the case base can be solved by using which abstract case from the hierarchy.

Based on the model, a hill-climbing optimization algorithm computes a pruned abstraction hierarchy and thereby the related fragment of the case base that leads to the lowest estimation of the expected overall cost for solving a new problem.

Experimental Results

We now present the results of an experimental study which shows the benefits of the developed retrieval approach. This study was done using the fully implemented PARIS system in the domain of manufacturing planning for rotary symmetric workpieces on a lathe (see (Bergmann and Wilke, 1995) for details of the domain).

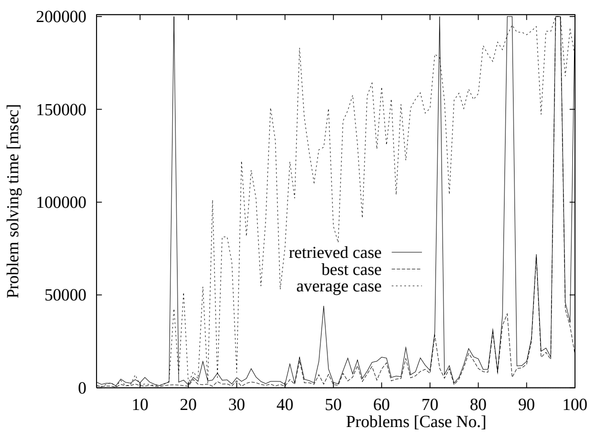

Figure 7: Problem solving time for 100 problems.

For the experiments, 100 concrete cases were generated randomly. From these concrete cases a set of

Table 1: Cases in the Case Base

Selected Case	average problem solving time
Best case	12 sec.
Average case	109 sec.
retrieved case	13 sec.

111 cases at four levels of abstraction could be generated. The retrieval mechanism constructed an abstraction hierarchy. Due to the pruning of the hierarchy, 85 cases became inaccessible and the retrieval is restricted to the remaining 26 cases.

In order to assess the efficiency of the retrieval, we determined the case-based problem solving time (retrieval+adaptation+refinement) for solving 100 problems. A time limit of 200 seconds was imposed. Problems that could not be solved within this time limit remained unsolved and a value of 200 seconds was noted as problem solving time for this problem. Fig. 7 shows the problem solving time for each of the 100 problems. The problems are ordered according to the length of the solution plan, i.e., problems with a higher number require a longer solution plan. As a reference, we also determined for each problem the problem solving time when using the best possible cases and the average problems solving time over the set of all possible applicable cases. The respective curves are also plotted in Fig. 7. Table 1 summarizes the average problem solving time over the problems that could be solved for the three conditions. Additionally, the maximally conservative sign text as described in (Etzioni and Etzioni, 1994) was performed to determine whether the performance gain when selecting the retrieved case over selecting the average case was significant. As expected, it was significant with a p-value $p < 0.000001$.

First, this experiment shows clearly that accurate retrieval is crucial for the performance of the system since the difference between retrieving the best and retrieving the worst case is very big. Second, the curve which shows the problem solving time for the retrieved case indicates only slightly higher values for most cases than the curve representing the best case.

Please note that the problem solving time measured

Table 2: Comparison of different retrieval techniques.

Retrieval approach	solved problems (percentage)	average time (solved probs.)	average time (all probs.)
linear	6 %	157 sec.	197 sec.
hierarchical, not pruned	41 %	26 sec.	127 sec.
hierarchical, pruned	93 %	13 sec.	24 sec.

for the retrieved case includes also the retrieval time, while the problem solving time for the best case does not. A detailed examination of the retrieval has shown that in 57% of the problems the retrieval selects the best possible case.

In a second experiment, we compared the retrieval results with two other variants:

- first, a linear retrieval in which cases are stored in arbitrary order in a list and the first adaptable case is selected and

- second, using the retrieval approach presented in section but without pruning of the abstraction hierarchy.

Fig. 8 shows the resulting curves. Tab 2 summarizes for every condition the percentage of solved problems, the average problem solving time over the solved problems, and the average problem solving time over all problems (for unsolved problems the time limit of 200 seconds was used). Additionally, the maximally conservative sign text was performed to determine whether the performance gain of the proposed retrieval approach with pruned hierarchies was significant over the linear retrieval and over the hierarchical retrieval without pruning. It turned out to be significant in both cases with a p-value $p < 0.000001$.

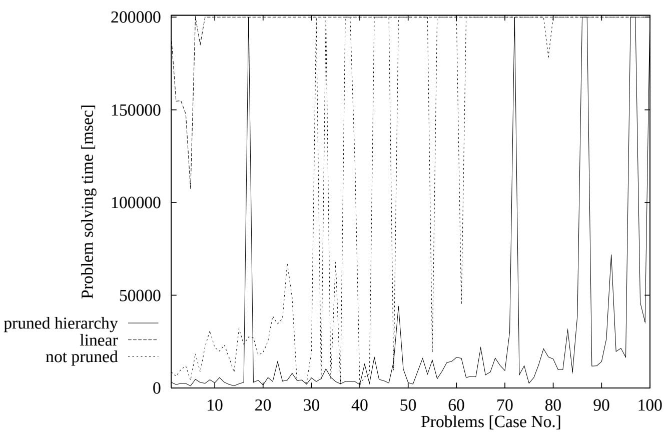

Figure 8: Comparison of different retrieval techniques.

This comparison makes clear that the pruning approach based on the developed cost model is in fact responsible for the good results. The naive linear retrieval is completely inappropriate; only 7% of the problems could be solved at all.

Summary

In this paper, we demonstrated how abstraction techniques can be successfully employed within case-based problem solving. We briefly presented the case-based planning system Paris as an example of an approach that integrates CBR with abstraction. Further, we argued that the selection of appropriate cases is very crucial for those systems, a claim that is further supported by the presented experiments. Therefore, we focused on a particular retrieval technique for selecting appropriate cases efficiently. Based on a general analysis of the efforts involved, the new retrieval technique is developed. Its main source of efficiency is a cost model of the expected problem solving time which allows to use an optimization approach for determining a good retrieval structure.

References

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59.

Bergmann, R. (1996). *Effizientes Problemlösen durch flexible Wiederverwendung von Fällen auf verschiedenen Abstraktionsebenen (in German)*. DISKI 138, Infix Verlag, Sankt Augustin (Germany).

Bergmann, R., Munoz-Avila, H., Veloso, M., and Melis, E. (1998). Case-based reasoning applied to planning tasks. In Lenz, M., Bartsch-Spoerl, B., Burkhard, H.-D., and Wess, S., editors, *Case-Based Reasonig Technology from Foundations to Applications*. Springer (in press).

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.

Bergmann, R. and Wilke, W. (1996). On the role of abstraction in case-based reasoning. In Smith, I. and Faltings, B., editors, *Advances in Case-Based Reasoning*, Lecture Notes in Artificial Intelligence, 1186, pages 28–43. Springer Verlag.

Branting, K. and Aha, D. (1995). Stratified case-based reasoning: Reusing hierarchical problem solving episodes. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 384–390.

Etzioni, O. and Etzioni, R. (1994). Statistical methods for analyzing speedup learning. *Machine Learning*, 14:333–347.

Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

Giunchiglia, F. and Walsh, T. (1992). A theory of abstraction. *Artificial Intelligence*, 57:323–389.

Holte, R. C., Mkadmi, T., Zimmer, R. M., and MacDonald, A. J. (1995). Speeding up problem solving by abstraction: A graph-oriented approach. Technical report, University of Ottawa, Ontario, Canada.

Kolodner, J. L. (1993). *Case-based reasoning*. Morgan Kaufmann.

Minsky, M. (1963). Steps toward artificial intelligence. In Feigenbaum, E., editor, *Computers and Thought*. McGraw-Hill, New York, NY.

Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80.

Sacerdoti, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135.

Smyth, B. and Cunningham, P. (1992). Deja vu: A hierarchical case-based reasoning system for software design. In Neumann, B., editor, *ECAI-92*, pages 587–589.

Veloso, M. M. (1992). *Learning by analogical reasoning in general problem solving*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA.