ELSEVIER

Knowledge-Based Systems 9 (1996) 83–91

Knowledge-Based SYSTEMS

Current status of case-retrieval in engineering domains: an analysis from the knowledge engineering perspective

Frank Maurer

Expert System Group, University of Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern, Germany

Abstract

The paper shows industrial practicians how current case-based reasoning technology can be used in engineering domains. It analyses knowledge needed to support case retrieval. It starts with descriptions of retrieval algorithms for several levels of expressiveness of the case representation language, and it discusses where the algorithmic complexity results from. To illustrate problems of current case-based reasoning approaches, it describes an application which supports the reuse of object oriented software.

Keywords: Case retrieval; Knowledge engineering

1. Introduction and overview

Designing artifacts like buildings, mechanical devices or software systems is known to be a complex and difficult task for well educated human experts. Trying to build knowledge-based systems for design tasks leads to huge problems in knowledge acquisition. The term 'knowledge acquisition bottleneck' was coined in the early 1980s [1], and it expresses the fact that building a knowledge base for a complex real-world task is not easy and slows down the practical application of expert system technology.

On the other hand case-based reasoning (CBR) is advertised as a possible solution to the knowledge acquisition problem. For example, Bergmann et al. and Cunningham et al. argue in this direction:

> "One of the aspired benefits of case-based reasoning is to reduce the need to acquire and explicitly represent general knowledge of the domain and thereby to overcome the knowledge acquisition bottleneck." [2]

> "One of the central advantages in using a case-based approach to develop knowledge-based systems (KBS) is that CBR systems can be developed without encoding a strong domain theory for the problem domain. In particular CBR systems can be developed without explicit encoding of problem solving knowledge." [3]

Reasons mentioned for this attractive property of case-based reasoning are as follows:

- Humans easily remember problem solving episodes. Therefore, it is easy to acquire examples (or cases) whereas it is difficult to get general problem solving knowledge.
- To solve new problems, humans often try to remember a similar old problem and adapt its solution to fit the new requirements. Therefore, CBR systems model the reasoning process of humans and are more understandable than classical expert systems. Therefore, the effort to maintain the knowledge base (or case base) is reduced.
- Modelling the domain is not needed or is at least easy because there is a natural representation for cases.

In this paper we will discuss the knowledge needs of CBR retrieval algorithms. Further, we will show that time spent on knowledge engineering

- improves the performance of simple CBR systems for classification tasks,
- is necessary for building CBR systems for design tasks.

The following section discusses four different levels of case representation languages. The representation used influences the complexity of the retrieval algorithms (see Section 3 and Section 4). In Section 5 derivational

SSDI 0950-7051(95)01021-1

analogy is analysed from the knowledge engineering point of view. Section 6 discusses the retrieval of designs and discusses as an example the knowledge needed for an application of the INRECA system in software engineering. We conclude with a summary.

2. Complexity of case representations

The basic idea of case-based reasoning is to use old experiences to solve new problems. The CBR principle is illustrated in Fig. 1. To solve a new problem, a CBR system searches for the best match of the new problem to the cases in the case base. Then it uses the solution of the retrieved case and adapts it to solve the new problem. Thus the new solution is not generated based on the new problem description and deep knowledge about the domain but from the solution of an old problem. In [4] a common structure of CBR systems is discussed and an overview of the task hierarchy and the inference structures of case-based reasoning is given.

The engineering of a CBR system therefore means that two components must be developed:

- The *retrieval component* is responsible for searching for a case in the case base with a solution which is a good start for the adaptation process.
- The *adaptation component* gets as input the old solution and adapts it to solve the new problem.

In this paper, we discuss algorithms for retrieval components which are often used in commercially available CBR tools. New research (e.g. [5]) leads towards an integration of adaptation knowledge into the retrieval process. These approaches are, although very promising, outside the scope of this paper, which concentrates on the current industrial status of case-based reasoning.

Case-based reasoning inherently incorporates an incremental learning mechanism. The new problem together with the adapted solution is a new case which can be stored in the case base and used for problem solving in the future. Usually, this incremental learning process improves the problem solving capabilities of CBR systems.

Looking at the advantages of CBR mentioned above, a question can be asked: why is not *every* knowledge-based system built with CBR technology? To avoid an overselling of the technology, in this paper we will analyse difficulties in building real-world CBR systems which can only be understood on a detailed and technical level. These relate to

- the availability of cases,
- the expressiveness of the case representation language,
- the complexity of the retrieval algorithm,
- the knowledge needed for the adaptation process.

Often it is claimed that cases are available in companies, and only these need to be used to solve the problem. When our group developed the first CBR system for diagnosing CNC machining centres eight years ago, the company claimed that they had a set of failure reports which could be used for case-based reasoning. We were very happy, and thought that our CBR system could easily be developed by transcribing the textual case description into the formal language of our tool. When we received the failure reports, we discovered the following:

- The failure reports contained a lot of customer information, the error code of the CNC control, and a description of the machine part which was replaced.
- For one error code, several different replacements were described. This meant that the error code was only a hint to be used in obtaining the solution, and it was *not sufficient* for a case-based classification process.

To solve these problems, we had to talk with the engineers and service technicians of the customer and define a set of attributes which were sufficient for finding the fault in the CNC machining centre. We had to do knowledge engineering and knowledge modelling. Therefore, we were focusing on the same problem as in developing conventional expert systems. [6] discusses analogous problems with the unavailability of cases and the effort of domain modelling in an architectural domain.

Modelling a domain for CBR brings up the question of an appropriate case representation language, a useful similarity measure, and the adaptation knowledge needed. The answers to these questions determine the effort for knowledge modelling and the efficiency of the resulting CBR system. Now we will discuss the influence of the expressiveness of the case representation on the efficiency of the retrieval algorithm.

The ESPRIT project INRECA [7], in which our group has been involved for three years, has developed CASUEL, a common case representation language. CASUEL focuses on representing cases for classification and diagnostic tasks. Therefore, the language support of background and adaptation knowledge is restricted to (simple) rules. For further details see [8]. In INRECA, we distinguish four levels of case representations.

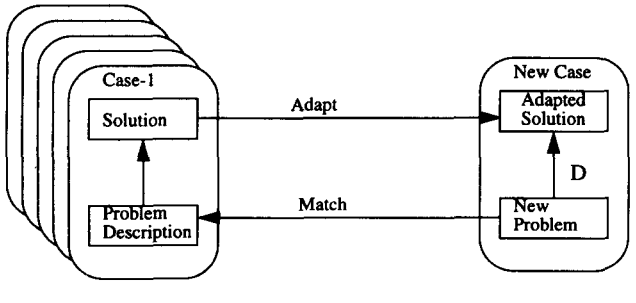

Fig. 1. CBR cycle.

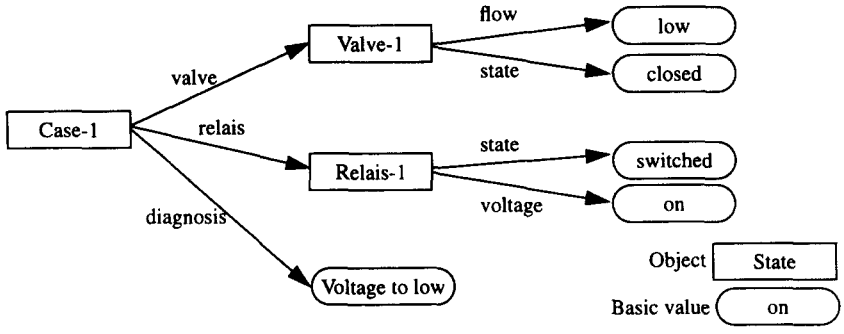

Fig. 2. Object tree in which every arc denotes an attribute of the object at its start.

2.1. Representation 1: feature vector with single-value slots

A case is described by a list of attribute value pairs. Every attribute value pair represents a feature which is useful for differentiating between cases. Often one attribute is marked as the target of the classification.

Example

Case-1
Tool_Changer = front
IO_Status_IN_37 = logical_1
Errorcode = I48
Contacts_21S5 = ok
Voltage_21S5 = 0 Volt
Diagnosis = Error_in_End_Switch_21S5**target**

2.2. Representation 2: feature vector with multivalue slots

A multivalue slot may contain several values. For example, German traffic lights may be RED and YELLOW at the same time (this means 'be ready to start'). Hence, the attribute value pairs may be

Traffic_Light = {RED, YELLOW}

The semantics of multivalue slots is not unambiguous. Therefore, in CASUEL, it is possible to define whether the slot has a CONJUNCTIVE or DISJUNCTIVE semantics.

2.3. Representation 3: object structures with single-value slot

If a case representation language allows a complex object to be stored as the value of a slot, the third level of expressiveness is reached. This representation allows a case to be viewed as an object tree (see Fig. 2).

First, object structures with single value slots can be transformed into flat attribute value representations without changing the information by defining one attribute for every path through the object tree. This means that this representation does not improve the inference capabilities of the CBR system. Nevertheless, it may help the human user to understand the case representation. Second, if it is possible to store one object (e.g. Valve-1) in two different slots, the object structure is a graph. Section 6 shows that graph representations for cases are natural in several engineering domains. Third, every arc has a unique name within the case. Therefore every object within a case can be identified unambiguously allowing efficient case retrieval.

2.4. Representation 4: object structures with multivalue slots

The highest complexity of case representations allows several complex objects to be stored in one slot. For example, the attribute all_valves contains two valves {Valve-1, Valve-2} which are equivalent for the CBR system (see Fig. 3). The problem resulting from this equivalency are discussed in the next section.

3. Case retrieval in INRECA

Because of the intended application domains, the INRECA project sees case-based reasoning basically as case retrieval using given similarity metrics[1], and applying the solution of the most similar case to the new problem without adaptation[2]. In this view, a simple retrieval algorithm searches linearly through all the cases in the case base:

```
algorithm CBR (new_problem);
Begin
  best_case := first case in case base;
  best_similarity := sim(new_problem, best_case);
  FORALL cases c in the case base DO
  BEGIN
    act_sim := sim(new_problem, c);
    if act_sim > best_similarity, then best_case := c;
      best_similarity := act_sim;
  END
  RETURN solution(best_case);
END
```

[1] Alternatively, a system can use dissimilarities (distance measures).

[2] The INRECA system also supports case adaptation based on adaptation rules. A deep explanation of these rules can be found in [8].

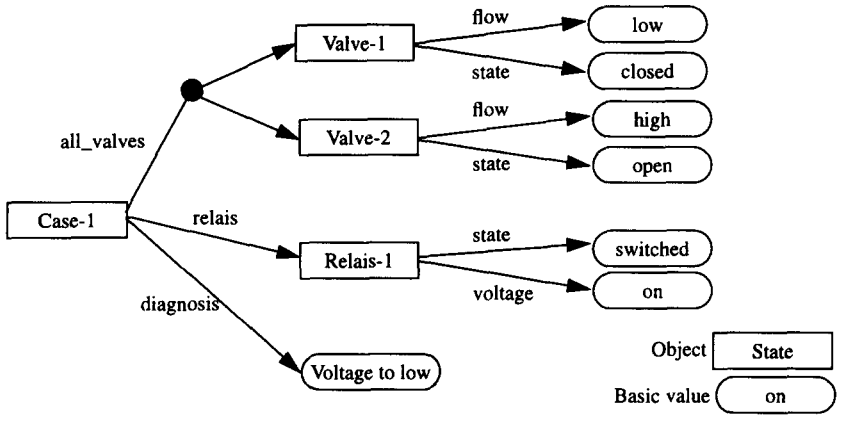

Fig. 3. Object tree with two valves in the slot all_valves.

There are several problems with this simple CBR algorithm which may result in inefficiencies and long response times. First, the algorithm has a linear time complexity with the number of cases in the case base. This is no problem with a small number of cases which can be held in main memory. However, in a lot of real world problems the case base may contain several tens of thousands or 100 000 cases which are stored on hard disks. Fetching these to compute their similarity to the new problem slows down the reasoning process dramatically. To overcome this problem, the INRECA project extended the so-called k-d trees, and developed an efficient indexing mechanism [9,10], which guarantees that the most similar case is retrieved, although on average only a small number of cases are accessed.

k-d trees can handle representations 1–3. The k-d trees of the INRECA system require unique names for every object in a case, e.g. multivalue slots containing objects are not supported. This problem results from the computation of similarities in INRECA (and most other commercial CBR tools).

In INRECA the similarity of two cases is defined as the weighted sum of the similarities of their attributes:

$$\mathrm{sim}_{\mathrm{case}}(c_1, c_2) = \frac{1}{n} \times \sum_{i=1}^{n} w(\mathrm{attribute}_i) \times \mathrm{sim}_i(c_1(\mathrm{attribute}_i), c_2(\mathrm{attribute}_i))$$

where n is the number of attributes and the weight on the ith attribute is

$$w(\mathrm{attribute}_i) \in [0, 1]$$

and the similarity of the values of the ith attribute of case c_1 and c_2 is

$$\mathrm{sim}_i(c_1(\mathrm{attribute}_i), c_2(\mathrm{attribute}_i)) \in [0, 1]$$

sim_i is easily computed if $c_1(\mathrm{attribute}_i)$ and $c_2(\mathrm{attribute}_i)$ are single values. However, what happens if $\mathrm{attribute}_i$ is a multivalue slot, as in Fig. 4? Then the system has to compute the similarity for every possible binding of values of $\mathrm{attribute}_i$ in Case-1 and Case-2. The *crossmatching* of attribute values of the two cases in Fig. 4 results in six computations of similarities for the first slot. In general, the number of similarity computations needed for a slot is the number of values in the first case times the number of values in the second case. Remembering that representation 4 defined above allows slots which store several objects which may also have slots containing several objects, and so on, one can see that the time needed to compute the similarity of two cases may explode.

The analysis of the properties of the INRECA retrieval algorithm shows that thinking about the model of the domain knowledge may improve the efficiency of the case retrieval:

- If two objects must definitely be matched to each other, storing them in a unique slot reduces search via crossmatching. For example, CNC machining centres contain several valves. A specific valve has a specific function for the machine and is therefore *not* equivalent to another valve. Fig. 5 shows two representations of a part of the CNC domain. Although they seem to be equivalent one leads to fast case retrieval whereas the other is inefficient.
- Avoid multivalue slots so that k-d tree indexing can reduce the number of cases accessed. For example, in one INRECA application (error detection of aeroplane engines), a case contained ten flags which represented features of the engine. The first model developed contained an attribute which stored a list of symbols representing the visible features. This model was natural and wanted by the service engineers. The

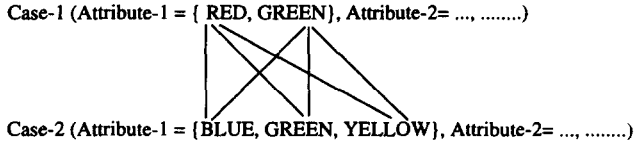

Fig. 4. Two cases with multivalue slots.

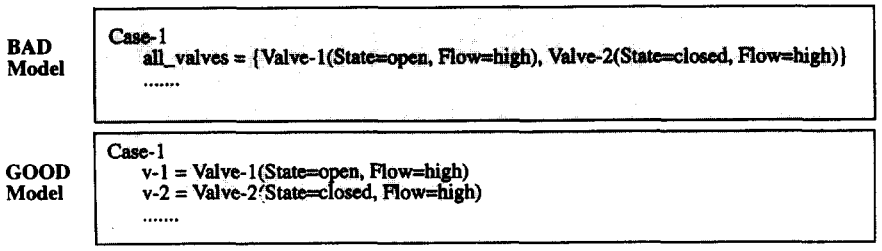

Fig. 5. Two case representations.

second model contained ten Boolean attributes, one for every feature. Then these attributes were handled by the k-d tree. (The transformation described is limited to features for which a unique identifier can be defined).

- Try to define an ordering for every enumeration type used which carries semantics of the domain. k-d trees use the ordering information to generate the indexing structure. If the ordering does not reflect semantics in the real world, then the retrieval algorithm must start to search in the k-d tree losing the advantage of indexing the case base.

4. Retrieval with graph matching

In many real world applications, multivalue slots cannot always be avoided. Also, there is often no clear target of the classification but the system has to retrieve the most similar structure in the case base. The following retrieval procedures for graph structures handle these complex domains:

- *Subgraph matching:* One similarity measure for graph structures proposed in the literature is subgraph isomorphy: two graphs are similar if one is a part of the other. [11] describes an efficient algorithm for the average case. On the basis of analogous ideas, [12] sketches a weighted subgraph matching algorithm. Both approaches use extensions of the RETE algorithm known from production systems to improve the matching speed for practical applications.
- *Hillclimbing:* The subgraph matching approaches mentioned above guarantee to find the best match of one graph to another. An alternative approach is to use hillclimbing to reduce the number of similarity computations in the crossmatching phase. The underlying idea is to search for the best match for the first object, then the second, and so on, without ever changing the previous bindings. The hillclimbing algorithm may result in a suboptimal binding of objects but it is faster than testing every permutation as in the correct procedure. Section 6 shows encouraging results in software reuse of integrating a hillclimbing algorithm into the INRECA system.

All retrieval procedures mentioned in Section 3 and Section 4 basically follow the transformational analogy paradigm: they search for old cases based on features of the problem description and *transform* the old solution to solve the new problem. The knowledge needed for the transformation process ranges from none, or a very small amount, for case-based classification, as for example in INRECA, to a huge amount of background domain knowledge, as in [13].

A completely different approach uses the traces of old problem solving behaviour to perform case-based reasoning.

5. Derivational approaches: reusing problem solving traces

Transformational approaches have an inherent problem: solutions do not show the underlying reasons for a special result. They show *what* is, and not *why* it is. The missing justifications for the reasoning steps lead to a lot of trouble in the adaptation process. Not knowing why a given part of the solution is there, special adaptation knowledge is required by the system to transform the solution to fit to the new problem; additional knowledge acquisition effort is needed.

Derivational approaches [14,15] tackle this problem. The justifications for reasoning steps were available when the old solution was generated. Storing these in the case allows them to be taken into account when adapting the old solution to the new problem. Therefore, instead of reusing old solutions, a derivational CBR system reuses the inference processes on which the old solution is based.

This paper only gives a very brief overview of derivational approaches, because up to now there are no commercial CBR systems which support them.

Derivational approaches typically search cases using deep domain features [16,17] and replay the stored reasoning traces in the new context. Normally, derivation-based CBR systems contain a component which is able to generate the solution for a new problem from scratch using knowledge stored in a knowledge base. Developing the knowledge base requires the same knowledge engineering as KBSs which reason from deep domain knowledge. Cases are used as heuristics which guide the search. They allow better solutions to be found in a shorter time.

Using cases, the effort to acquire search control knowledge is reduced.

Derivational approaches are typically used for synthetic tasks where the solution itself has a complex inner structure: it is a plan, a drawing, or a system configuration. The goal in these applications is normally to find a *good* solution. Hence, optimality is the core problem. Usually, in these domains, two solutions can be compared and ranked: more or fewer optimal solutions for a given problem exist. Because every case can be adapted to solve the new problem[3], the problem is to find a case which is a *useful* starting point for the adaptation process. This analysis also shows that the similarity of two cases is in fact an a posteriori criterion and its value can only be computed after the reuse has happened. Therefore, for building applications, one has to search for an a priori criterion which gives a good approximation of the adaptation effort. Then, this can be used in a similarity-based retrieval procedure (as described in Section 3 and Section 4).

To reuse old problem solving traces, the system has to manage dependencies between sets of information [18,19] so that the underlying reasons for a decision can be found. In this sense, design rationale [20] can be used to support the reuse of designs and design processes.

6. Reuse in engineering domains

In the previous sections we described approaches to case retrievel. Now we will analyse how these can be applied in engineering domains.

Case-based reasoning has dealt with engineering domains since its early phases. In [21] a case-based problem solver for landscape design is described. [22] illustrates how a functional model of a device can be used to adapt its design. Case-based reasoning for autoclave layout design is described in [23,24]. Storing design knowledge in cases for later reuse was discussed earlier in [25].

In engineering domains, graphical representations of the artifacts to be constructed are often used. In mechanical engineering or architecture the artifact is developed with the aid of (feature-based) CAD systems. The drawings can be mapped onto typed graphs where the nodes represent the objects of the domain and link annotated relations between these objects. Types are used to model the semantics of objects. For example, we are developing a system which supports city planners in developing urban land-use plans which determine what kind of buildings are allowed in a special part of the city and how much of the ground an be used for building. Every part of the plan has a semantics which is defined in the relevant German laws. We analysed the domain and defined classes (in the object-oriented sense) carrying these semantics. Therefore, internally, the plan is represented by a set of linked objects with meaning in the domain instead of lines and points.

In software engineering CASE tools are used to build models of the domain. Entity relationship diagrams, dataflow diagrams or state-transition diagrams are naturally mapped onto graph structures. If these graphs are stored in a case base of old designs, they can be retrieved using the algorithms described in Section 4.

In addition to transformational approaches, derivational analogy will have its place in design reuse. Efforts are being made to model design processes to capture the reasons for every decision. Quality management standards (ISO 9000–9004) require the documentation of every step in design and production processes. Stored in (formalized) case bases, documentation containing the reasons for decisions can be used for adapting the retrieved solution to the new problem. These developments strongly encourage the use of derivational approaches in building real world applications. Therefore, the improvement and extension of derivational approaches for complex domains is a research topic for the future.

Although graphs are natural representations in engineering domains, an analysis of the domain is needed to build efficient case retrieval systems. Spending even more effort on knowledge modelling allows simpler case representations and efficient indexing procedures to be used. Often it is possible to develop a set of features which allow a case to be characterized. An example from software engineering will illustrate this.

We carried out a case study on how to use INRECA technology to support the reuse of object oriented software. INRECA is one of the most advanced CBR tools commercially available. [26] compares several (older) case-based reasoning systems. Overcoming their problems was one of the main objectives in the INRECA project.

To evaluate our approach, we used two Smalltalk class libraries: the collection classes of the standard Smalltalk-80 system (e.g. lists, sets, bags, arrays, streams, etc.) and a library of user interfaces developed by our group. Every class in the library is one case. The goal of the retrieval was to find a class which could be adapted (via subclassing) to implement the required functionality. From the knowledge modelling perspective we distinguished functional (semantic) and syntactic features of cases. Fig. 6 shows an extended entity relationship (ER) diagram which describes the structure of a case from the collection class domain. The syntactic features, which are shown inside the tinted boxes, were generated automatically from the source code[4]. Normally, syntactical

[3] In the worst case, the retrieved information is useless and thrown away and the solution is generated from scratch.

[4] Exception: the ParameterTypes and the ElementType were acquired manually because Smalltalk-80 is an untyped language. For C++ class libraries these can be extracted from the source code.

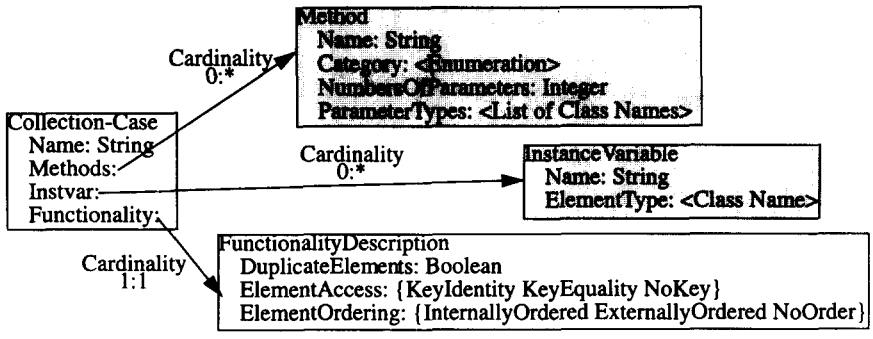

Fig. 6. Case structure in the Collection-Class domain.

features also express semantic properties because software programmers use sensible names.

Functional features were defined by Smalltalk experts and required a comprehensive domain analysis. For the collection classes, the functional features are (cf. Fig. 6) as follows:

- Are duplicates allowed in a collection? As an example, identical elements can be added to lists but not to sets.
- Do you want to access the elements of the collection via a key?
- How is the sequence of elements determined?

The ER diagram shows that we used a fourth level case representation (restricted to object trees) to build a CBR system which supported the reuse of object oriented software. To reduce the retrieval times, we used the hillclimbing algorithm.

To test the model, my colleague R. Bergmann carried out some small experiments. The results of the experiments can be seen as a hint as to how useful syntactic/functional features are. For an empirical proof, further experiments would have to be carried out, which were outside the scope of the case study. The case bases contained 30 cases of collections and 49 cases of user interfaces. The numbers of queries were 17 and 24, respectively. The first experiment measured the number of cases with the highest similarity. The results (see Table 1) show how discriminating the case retrieval is. A more detailed description of the case study and its experiments is given in [27].

The second experiment dealt with the retrieval accuracy. Table 2 shows the percentage of queries which retrieved at least one useful case within the set with maximum similarity.

To summarize the result of the experiments, using functional features allows the retrieval of useful cases whereas syntactic features reduce the number of retrieved cases. A reduced number of retrieved cases helps because the human user has fewer cases to inspect to decide which is the most useful for his/her problem. Using only functional features allows the restriction of the case representation to level 1, i.e. every class is described by a set of attribute–value pairs. This means that k-d trees can be used to improve the retrieval efficiency. Using only functional features may result in many retrieved cases with maximum similarity. To further reduce the number an additional domain analysis would be needed which led to new functional features.

Analysing commercially available case retrieval

Table 1
Cases with maximal similarity[a]

Case base	Only functional features	Only syntactical features	Functional and syntactical features
Collection classes	4.3 cases (14.1%)	3.9 cases (13.1%)	1.65 cases (5.5%)
User interface classes	4.2 cases (8.5%)	1.54 cases (3.1%)	1.46 cases (3.0%)

[a] The percentages are percentages of the case base.

Table 2
Percentage of queries which retrieved at least one useful case within the set with maximum similarity

Case base	Only functional features	Only syntactical features	Functional and syntactical features
Collection classes	100%	59%	71%
User interface classes	100%	88%	100%

techniques from a knowledge engineering perspective clearly leads to a number of research questions:

- Where is a methodology for knowledge modelling for case-based reasoning systems?
- How can background knowledge be used to support retrieval and adaptation?
- Given background knowledge (a domain theory) and a case representation, how can an indexing structure be generated which supports the retrieval of useful cases?
- How can derivational knowledge be used for efficient case retrieval and adaptation?

7. Summary

In this paper we have discussed the influence of the case representation on the computational complexity of case retrieval algorithms. We have shown that in design domains there are applications for every level of case representation. There is a clear tradeoff between the time spent on domain analysis and knowledge modelling and the time the CBR system needs to retrieve the most similar case. Defining functional features of designs is a huge knowledge acquisition effort whereas using graph matching procedures may lead to intractable retrieval times. Clearly, this shows that currently case-based reasoning is no silver bullet, and that it does not overcome the knowledge acquisition bottleneck in general, but only widens the neck of the bottle a little.

Acknowledgements

The work reported on in this paper was partially funded by ESPRIT project P-6322 INRECA. The partners of INRECA are AcknoSoft (Paris, France), IMS (Dublin, Ireland), tecInno (Kaiserslautern, Germany) and the University of Kaiserslautern, Germany.

References

[1] E. Feigenbaum and P. McCorduck, *The Fifth Generation*. Addison–Wesley, USA, 1983.

[2] R. Bergmann, G. Pews and W. Wilke, Explanation-based similarity: a unifying approach for integrating domain knowledge into a case-based reasoning for diagnosis and planning tasks, in S. Wess, K.-D. Althoff and M.M. Richter (eds.), *Topics in Case-Based Reasoning*, Springer-Verlag, Germany, 1994.

[3] P. Cunningham, D. Finn and S. Slattery, Knowledge engineering requirements in derivational analogy, in S. Wess, K.-D. Althoff and M.M. Richter (eds.), *Topics in Case-Based Reasoning*, Springer-Verlag, Germany, 1994.

[4] E. Armengol and E. Plaza, A knowledge level model of knowledge-based reasoning, in S. Wess, K.-D. Althoff and M.M. Richter (eds.), *Topics in Case-Based Reasoning*, Springer-Verlag, Germany, 1994.

[5] B. Smyth and M. Keane, Retrieving adaptable cases — the role of adaptation knowledge in case retrieval, in S. Wess, K.-D. Althoff and M.M. Richter (eds.), *Topics in Case-Based Reasoning*, Springer-Verlag, Germany, 1994.

[6] A. Voss, The need for knowledge acquisition in case-based reasoning — some experiences from an architectural domain, in A. Cohn (ed.), *Proc ECAI 94: 11th European Conference on Artificial Intelligence*, John Wiley, UK, 1994.

[7] M. Manago, K.-D. Althoff, E. Auriol, R. Traphöner, S. Wess, N. Conruyt and F. Maurer, Induction of reasoning for cases, in M. Richter, K.-D. Althoff, F. Maurer and S. Wess (eds.), Proc. *EWCBR-93: 1st European Workshop on Case-Based Reasoning*, University of Kaiserslautern, Germany, 1993.

[8] M. Manago, R. Bergmann, N. Conruyt, R. Traphöner, R. Pasley, J. Le Renard, F. Maurer, S. Wess, K.-D. Althoff and S. Garry, CASUEL: A common case representation language, *INRECA Deliverable D1* (Version 2.01), 1994.

[9] S. Wess, Case-based problem solving in knowledge-based systems for decision support and diagnosis, *Dissertation*, Universität Kaiserslautern, Germany, 1995 (forthcoming German title: Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik).

[10] S. Wess, K.-D. Althoff and G. Derwand, Using k-d trees to improve the retrieval step in case-based reasoning, in S. Wess, K.-D. Althoff and M.M. Richter (eds.), *Topics in Case-Based Reasoning*, Springer-Verlag, Germany, 1994.

[11] H. Bunke and B.T. Messmer, Similarity measures for structured representations, in S. Wess, K.-D. Althoff and M.M. Richter (eds.), *Topics in Case-Based Reasoning*, Springer-Verlag, Germany, 1994.

[12] F. Maurer, Similarity based retrieval of interpretation models, in M. Richter, K.-D. Althoff, F. Maurer and S. Wess (eds.), Proc. *EWCBR-93: 1st European Workshop on Case-Based Reasoning*, University of Kaiserslautern, Germany, 1993.

[13] K. Hua, I. Smith and B. Faltings, Integrated case-based building design, in S. Wess, K.-D. Althoff and M.M. Richter (eds.), *Topics in Case-Based Reasoning*, Springer-Verlag, Germany, 1994.

[14] J.G. Carbonell, Derivational analogy: a theory of reconstructive problem solving and expertise acquisition, in R.S. Michalski, J.G. Carbonell and T.M. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol. II, Morgan Kaufmann, 1986, pp. 371–392.

[15] M. Veloso and J. Carbonell, Derivational analogy in PRODIGY: automating case acquisition, storage, and utilization, in J. Kolodner (ed.), *Machine Learning*, 10 (1993) (special issue on case-based reasoning).

[16] H. Munoz, J. Paulokat and S. Wess, Controlling nonlinear hierarchical planning by case replay, in M. Keane, J.P. Haton and M. Manago (eds.), *Proc. EWCBR-94: Second European Case-Based Reasoning Workshop, Abbaye de Royaumont, France*, November 7–10 1994.

[17] M.M. Veloso, Prodigy/analogy: analogical reasoning in general problem solving, in S. Wess, K.-D. Althoff and M.M. Richter (eds.), *Topics in Case-Based Reasoning*, Springer-Verlag, Germany, 1994.

[18] F. Maurer and J. Paulokat, Operationalizing conceptual models based on a model of dependencies, in A. Cohn (ed.), *Proc. ECAI 94: 11th European Conference on Artificial Intelligence*, John Wiley, UK, 1994.

[19] C. Petrie, Planning and replanning with reason maintenance, *Dissertation*, University of Texas at Austin, USA, 1991.

[20] Working Notes of AAAI 92 Workshop on Design Rationale Capture and Use, San Jose, CA, USA, July 15, 1992.

[21] D. Navinchandra, Case-based reasoning in CYCLOPS, a design problem solver, in *Case-Based Reasoning: Proc. Workshop on Case-Based Reasoning, Clearwater, USA*, 1988.

[22] A. Goel and B. Chandrasekaran, Use of device models in

adaptation of design cases, in *Case-Based Reasoning: Proc. Workshop on Case-Based Reasoning, Pensacola Beach, USA*, 1989.

[23] R. Barletta and D. Hennessy, Case adaptation in autoclave layout design, in *Case-Based Reasoning: Proc. Workshop on Case-Based Reasoning, Pensacola Beach, USA*, 1989.

[24] W.S. Mark, Case-based reasoning for autoclave management, in *Case-Based Reasoning: Proc. Workshop on Case-Based Reasoning, Pensacola Beach, USA*, 1989.

[25] P. Alexander, G. Minden, C. Tsatsoulis and J. Holtzman, Storing design knowledge in cases, in *Case-Based Reasoning: Proc. Workshop on Case-Based Reasoning, Pensacola Beach, USA*, 1989.

[26] K.-D. Althoff, R. Barletta, M. Manago and E. Auriol, *A Review of Industrial Case-Based Reasoning Tools*, AI Intelligence, UK, 1995.

[27] R. Bergmann and U. Eisenecker, CBR to support the reuse of object-oriented software: a case study, in M. Richter and F. Maurer (eds.), *Proc. Expertensysteme 95: 3rd German Expert Systems Conf.*, Infix Verlag, Germany, 1995 (German title: Fallbasiertes Schließen zur Unterstützung der Wiederverwendung objektorientierter Software: Eine Fallstudie).

Bibliograhy

AAAI 94 Workshop on Case-Based Reasonin, Seattle, USA, 1994.