Exploring Case-Based Building Design–CADRE

KEFENG HUA and BOI FALTINGS
Artificial Intelligence Laboratory(LIA)
Swiss Federal Institute of Technology(EPFL)
INR-Ecublens
CH-1015 Lausanne
Switzerland
e-mail: hua@lia.di.epfl.ch, faltings@lia.di.epfl.ch

Abstract.

Case-based design promises important advantages over rule-based design systems. However, the actual implementation of the paradigm poses many problems which put the advantages into question. In our work on CADRE, a case-based building design system, we have encountered seven fundamental problems which we think are common to most case-based design systems. We describe the problems and the ways we either solved or worked around them in the CADRE system. This leads us to conclusions about the general applicability of case-based reasoning to building design.

1. Introduction

Observations of human designers frequently point out their reuse of previous design cases as a source of design knowledge. In Artificial Intelligence, the technique of case-based reasoning(CBR) models this problem solving strategy, and case-based *design* (CBD) is therefore a promising technique for intelligent design systems. Cases are distinguished from other forms of knowledge or experience by the fact that they model specific, *ungeneralized* instances of artifacts or events.

Knowledge in a design system can be divided into *design knowledge* used to synthesize and *domain knowledge* to analyze designs. We define a *case-based design* system as one in which cases are used as a source of *design knowledge*. Applying

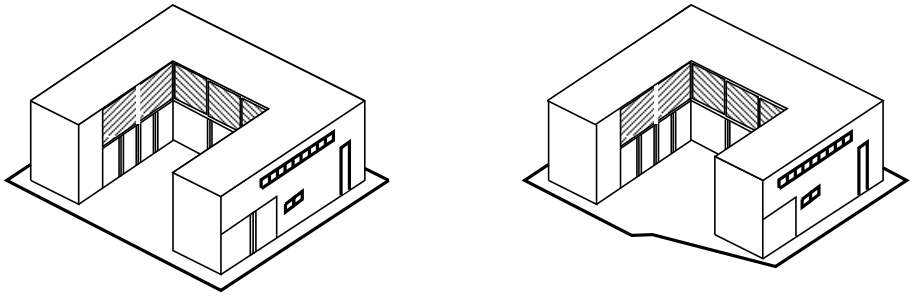
cases to *analysis* is addressed by much existing work in case-based reasoning and not the focus of this research.

Some experimental CBD systems include ARCHIE (Goel et al, 1991), CADSYN (Maher and Zhang, 1991) and CADET (Sycara et al, 1991), all of which use cases to generate new designs. Some CBD systems are hybrid systems: for example, the models of (Wang and Howard, 1991), (Rosenman, et al 1991) and (Golding et al, 1991) incorporate generalized design knowledge in a CBD system.

Case-based reasoning originated from ideas on human memory structure(Schank, 1982) and was thus motivated by considerations of cognitive psychology. However, applying case-based reasoning to design promises specific *practical* advantages:

- A case-based design system does not require a complete domain model, but can produce complete and complex designs even with a small knowledge base.
- Design starts from complete cases, implicitly achieving tradeoffs between several functions. This avoids the problem of multi-criteria optimization.
- Starting with complete cases reduces the complexity and thus increases the efficiency of problem solving.
- Using cases as the source of knowledge allows learning by simply storing new cases.

While routine design can be carried out by simple reuse of cases, novel designs can only be achieved through adaptation or combination of cases. The advantages of case-based design are consequences of the *hypothesis* that adaptation or combination of cases is easier than generation of a design from scratch. There are several reasons to believe in this hypothesis. One is that adaptation only has to focus on



Case **Adaptation of the case**

Fig. 1. *Example of case adaptation*

the differences between the old and new use of the case: we do not always have to redesign the details of the windows when a building is built in a different environment, for example. Another is that adaptation can better accommodate the fact that design specifications are often given by constraints (Sycara et al, 1992).

In order to explore if the hypothesis is really true, we have implemented a CAse-based building design system through Dimensionality REduction(CADRE) (Faltings, 1991; Hua et al, 1992). CADRE focuses on the *adaptation* of building designs into new environments. One example treated by CADRE is shown in Figure 1. It is a U-shaped building (the Felder House in Lugano, Switzerland) adapted to a slightly different site. CADRE modified both the dimensions and the topology of the case to obtain a solution that preserves the functionalities and tradeoffs in the case. While implementing CADRE, we encountered seven general problems associated with case-based design. In this paper, we discuss the problems and the ways in which we either solve or work around them in CADRE. This leads us to a general discussion on how the potential advantages of case-based design can best be achieved in an actual design system.

2. Cases for design

From the beginning of AI, cases have been treated as one of the most important sources of knowledge. For example, the checker player of Samuel(Samuel, 1959) used a library of some 53000 cases of positions as a basis for its play. Learning from examples is a basic technique of knowledge acquisition, and has also been applied to design cases. The main difference between learning from examples and case-based reasoning is that instead of being compiled into generalized descriptions during knowledge acquisition, cases are generalized with respect to a specific problem and during the problem solving process itself.

We distinguish *deep* and *shallow* design cases. A *deep* case is a design solution annotated with its design history. A *shallow* case is only the design solution itself. Because the design history is an important source of knowledge, deep cases can give better results than shallow ones. Modeling the design history requires a general design knowledge base which is sufficiently complete to generate the design and competing alternatives. The large amount of general knowledge required for representing deep cases is in opposition to one of the fundamental advantages of case-based reasoning, that of not requiring a complete knowledge base. In CADRE, we therefore focussed on *shallow* cases.

A *shallow* case can furnish the following knowledge to the design process:

- the actual structure needed to satisfy a set of requirements.
- the tradeoffs made between functional requirements.
- tacit considerations, such as the style of a building.

In CADRE, we represent the structure as a CAD model. The functional features

achieved by the structure are modelled in a symbolic vocabulary where each functional feature is mapped to a *constraint* on the CAD model. Case-based design is particularly advantageous because it can furnish knowledge about tradeoffs, which is otherwise hard to formalize. The mapping of functional features to constraints used in CADRE makes tradeoffs explicit in the ways that these constraints are satisfied or broken. Tacit considerations are by definition not captured explicitly and cannot be reasoned about.

The purpose of a case-based design system is to apply this knowledge to new situations. In the following, we describe the general problems that we have encountered in meeting this challenge in the CADRE system.

3. Problems for implementing case-based design

Case-based design is generally assumed to consist of three main processes: retrieval, adaptation and combination. In CADRE, we have so far implemented retrieval and adaptation processes. In this section, we review the problems that we have discovered in the course of this implementation, and the way we solved them in CADRE. We leave the discussion of the combination process, which we have not implemented yet, to the next section.

3.1. CASE RETRIEVAL

Superficial similarity problem

Case retrieval has been addressed as a central issue from the beginning of case-based reasoning(Schank, 1982; Kolodner, 1984). In case-based design, it is often *not* the

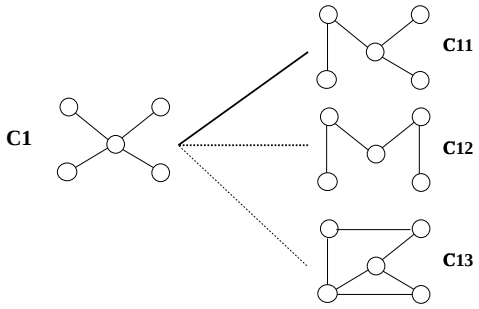


Fig. 2. *An example of superficial similarities.*

cases that are most similar to the new design context which generate the best designs. As pointed out by Kolodner(Kolodner, 1989), the best case to use is the one which contributes the most to the final design. However, how much a case can contribute only becomes clear once the design is finished. Predicting the contribution on the basis of superficial features is difficult. For example, in Figure 2, among C_{11}, C_{12} and C_{13}, topology C_{11} is structurally the one the most similar to C_1. However, in the new design context, symmetry can be a more important consideration. In this case C_{12} can contribute the most, in spite of the fact that the structural similarity metric would rank it as less promising. The choice of metric depends on the details of the design problem; any fixed metric can only determine *superficial* similarities. One potential solution is given by learning: adjust the similarity metric through analyzing failures of previous retrievals.

Similar observations have been made by researchers in other applications of case-based reasoning. Relevant discussions in the literature include in particular the distinction between “surface” and “deep” features and their relative merits in indexing cases (Hammond, 1989b; Birnbaum, 1989; Owens, 1989). Since indexing cases is not a problem which is specific to design, we have not attempted to develop

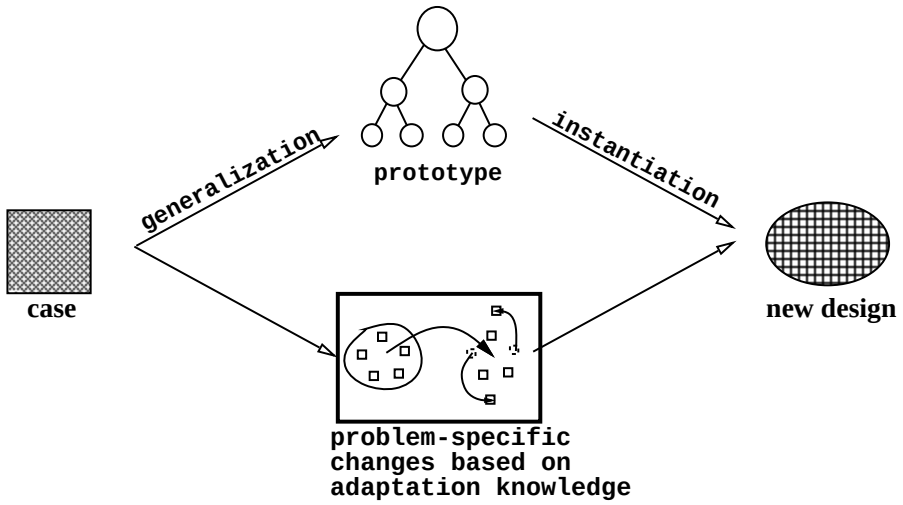


Fig. 3. *Two ways to provide adaptations: generalization to prototypes, or incremental changes by specific rules.*

a novel approach. In CADRE, we therefore let the user choose the case supported by a browsing mechanism based on a fixed index of functional features. In fact, this may also be the solution most preferred by the user, who would like to maintain control over this aspect of the design process.

3.2. CASE ADAPTATION

Adaptation knowledge problem

New design problems are rarely identical to those for which the case was designed, so *adaptation* is usually necessary to reuse a case in a new problem. However, *shallow* cases do not define by themselves how they can be adapted. Figure 3 illustrates the two ways in which adaptation knowledge can be formulated:

- by categorizing the case as instances of prototypical designs which can simply be reinstantiated. This brings up the question of why cases are necessary at all, as prototypes could fulfill the same function.
- by providing specific adaptation knowledge which modifies aspects of cases instead of regenerating or reinstantiating them. Such adaptation knowledge could be stored with the cases in the case library, be kept separately as generalized domain knowledge, or supplied by the user at runtime.

For case-based design systems, the solution of using specific adaptation knowledge offers several advantages: it is easier to formulate than generation knowledge (Sycara et al, 1992), and it need not be complete to obtain a useful design system. In CADRE, we store specific adaptation knowledge with each case, consisting of two parts:

- dimensional knowledge, expressed as constraints on the dimensions of the structure's CAD model. Constraints can be *definitions*, such as:

  $$area = width \cdot length$$

  or *restrictions*, such as:

  $$5.6 \cdot width_{column1} \cdot depth^2_{column1} > 0.25 \cdot load \cdot length^2_{beam1} \cdot length_{beam2}$$

  Constraints are used by general weak methods to adapt the dimensions of the case.
- topological knowledge, expressed by a set of topological change rules specific to the case. A topological change rule might propose to reduce the number of rooms when their dimensions fall below a prescribed minimum. The topological change rules generate a space of adaptations from which the user chooses

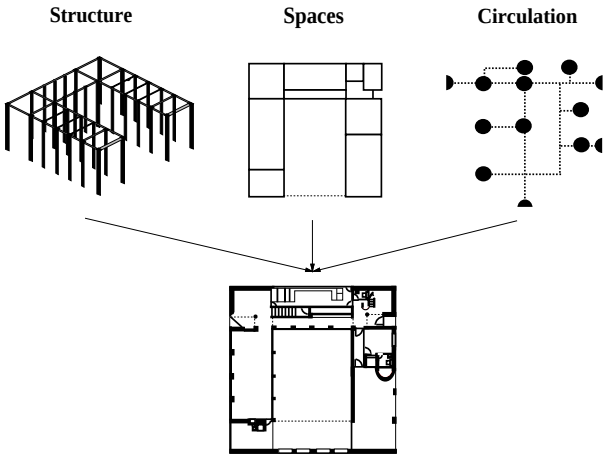


Fig. 4. *Design requires the integration of many different abstractions.*

the most appropriate one.

It would be desirable to formulate at least part of the adaptation knowledge as a general knowledge base, but so far we have not addressed this problem. Several researchers have proposed solutions to the problem of adaptation and providing the required knowledge in domains such as planning (CHEF, (Hammond, 1989a)) or explanation (SWALE, (Kass et al. 1986)). However, the adaptation methods which have been proposed apply only to context-free symbolic structures and are not powerful enough to provide meaningful adaptations of building designs.

Inadmissible generalization problem

A design satisfies functions in many different abstractions. For example, a building provides a set of spaces to use, a structure which makes it stand up, and a circulation pattern which allows people to get around in it (Figure 4). Adaptations are usually formulated with respect to one abstraction, and may clobber functions in another:

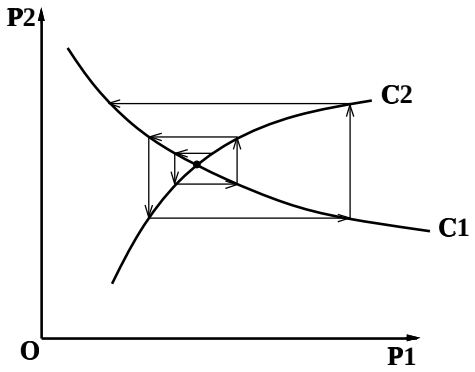


Fig. 5. *P1/C1 and P2/C2 represent parameters and constraints in two different but interacting abstractions: P1 has to fall onto constraint curve C1 and P2 onto C2. When each abstraction is treated in isolation, the process may diverge as indicated by the arrows.*

for example, modifying the spaces may make the structure unstable or create a bad circulation pattern. Such modifications amount to an *inadmissible generalization* of those aspects of the case which are not part of the abstraction being considered. This problem of integrating different abstractions occurs particularly in design.

Inadmissible generalization can be avoided by mapping the various abstractions into a *single* model in which adaptations are formulated. In CADRE, we ground all abstractions in a single geometric model of the design. All constraints defined by the abstractions are mapped into the geometric model, and modifications can be formulated in this model to be consistent with all abstractions.

Complexity of consistent modification problem

Modifications of a case are usually generated based on considerations of one abstraction at a time. For example, structure elements may be rearranged to satisfy stability requirements, or spaces may be modified according to functional considerations. Even when the different abstractions are mapped to a common model,

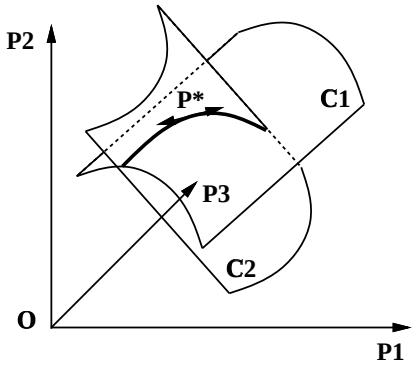


Fig. 6. *The two constraint surfaces leave only one effective degree of freedom among the three parameters. By dimensionality reduction, an explicit representation in terms of a single parameter, P*, along the intersection of the constraint surfaces is obtained.*

ensuring consistency during adaptation may be a difficult problem.

Figure 5 illustrates a case of two interacting abstractions, represented by the parameter/constraint combinations P1/C1 and P2/C2. The most natural way to formulate adaptation knowledge is a blackboard system with rules that treat each abstraction in isolation. In the example of Figure 5, this would mean that violations of C1 are corrected by adjusting parameter P1, and violations of C2 by adjusting P2. As indicated by the arrows in Figure 5, this process can actually *diverge* away from the actual solution. Ensuring the convergence of a such a modification process is a recognized problem in systems such as IBDE (Schmitt, 1990; Fenves, 1989) which attempt to integrate different abstractions using a blackboard approach.

The solution we developed for CADRE is that of *dimensionality reduction*: rather than propagate changes locally, we solve the constraint equations to directly compute the subspace of feasible solutions, and then search for an adaptation in this subspace. Thus, for the example in Figure 5, we directly compute the intersection point of the two constraints and return this as a result. In general, a space of

n parameters with m equality constraints reduces to $(n - m)$ non-conflicting parameters. Thus, Figure 6 shows an example where three parameters and two constraints reduce to a single parameter P*. Any modification obtained by varying P* is *guaranteed* to be consistent with all abstractions. When constraints are non-linear, there may be several disjoint subspaces, and only the one corresponding to the values present in the case is retained. Dimensionality reduction was originally developed for image understanding (Saund, 1989).

One problem with applying dimensionality reduction in design is that it only applies to constraints expressed as equalities, while many of the constraints on a building are in fact inequalities. Among inequality constraints, we can distinguish *critical* inequalities which are just satisfied or even slightly violated by the design, and *non-critical* ones which are satisfied by a large margin. The subset of critical constraints is a characteristic feature of the design and does not change during adaptation unless the case is radically modified. All critical inequality constraints can thus be treated as *equalities* and incorporated in the dimensionality reduction. Non-critical inequalities are likely to remain non-critical during adaptation; if verification detects that some are not satisfied after adaptation, they are also declared critical and the dimensionality reduction is recomputed.

In practice, the results of dimensionality reduction are impressive, reducing the thousands of variable parameters in a building to a small set (for example, only 2 parameters in the problem of Figure 1). Adaptations in the subspace thus defined can be found by optimization methods or simply user interaction. The non-critical inequalities are reformulated in terms of the reduced set of parameters and constrain

the adaptation within the subspace.

In CADRE, we implemented dimensionality reduction for linear and nonlinear constraints using the implementation of Buchberger's simplification algorithm (Buchberger, 1985) in the REDUCE system. When the size of the problem is large, the calculation of dimensionality can last hours. However, this time could be reduced significantly by specialized constraint solving methods which only treat those forms of non-linearity which actually appear in constraints on building designs.

When adapting a case requires a significant amount of topological changes, it would be desirable to have an analog of dimensionality reduction for discrete structures, such as graphs. Despite a significant effort, we have not been able to define such an analog in a useful way. Topological changes thus might still suffer from cycling or diverging behavior.

Complexity of matching problem

Reusing a case in a new context means that it has to be *matched* to the new environment. For example, a building in a square site can be inserted in at least four different ways, in which some are better than others since they connect well with surrounding streets, buildings or the general orientation. The matching problem can be defined as the problem of finding homomorphic components between graphs showing the components and connections of the original and the new context, such as shown in Figure 7. Since the subgraph homomorphism problem is known to be NP-complete, this problem can cause unmanageable computational complexity.

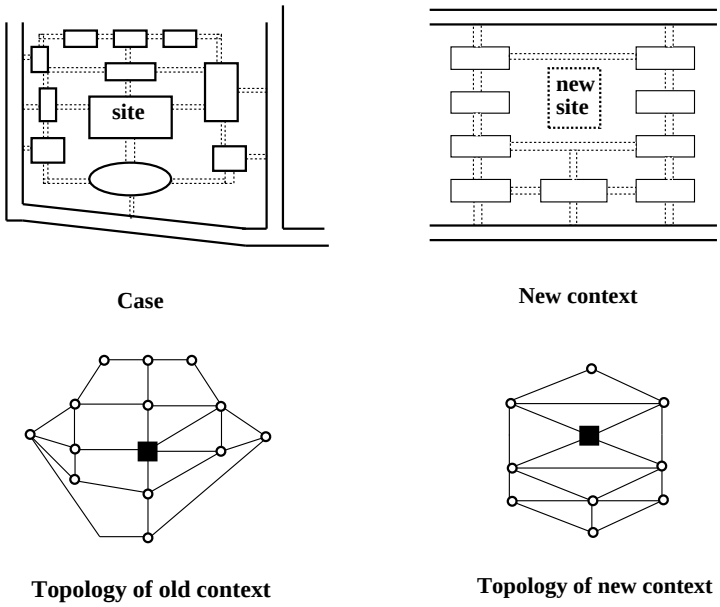


Fig. 7. *Finding all the different ways of matching the features of the case to those of the new problem can be very complex.*

In CADRE, the possible matches are restricted by representing the features in a hierarchy of detail, and matches found at less detailed levels constrain the possibilities at more detailed levels. At each level of abstraction, the user disambiguates by asserting additional constraints on the possible matches.

4. Case combination

An important means to obtain novel designs is by combination of several cases (Faltings, 1991; Hinrichs, 1991). We can distinguish two forms of combination: *crossing* and *composition*. In *crossing*, we combine properties of cases. In *composition*, we assemble pieces of structure taken from the cases. When considering the problem of case combination within CADRE, we discovered the following two problems, for which we have not worked out any solutions so far:

Fig. 8. *Crossing a building with a banana.*

Property assimilation problem

The main problem associated with case *crossing* is that properties of a *contributing* case must be *assimilated* into an *accepting* case. For example, crossing a banana as a contributing case with a rectangular building as an accepting case (Figure 8) will normally mean that only the curved shape of the banana should be assimilated into the building, but not its size or colour. This assimilation may affect every element in the originally rectangular building. The property assimilation problem is the problem of defining the property which is to be carried over, and the precise way in which assimilating it will affect the elements of accepting case. We have not found a solution to this problem yet, but it seems that the ideas for integration of multiple abstractions of design in CADRE may be useful for property assimilation as well. Functional features could then be assimilated into other cases by imposing the corresponding constraints into their models.

Incompatible abstraction problem

Composition means combining pieces of structure taken from different cases. This requires first of all a decomposition of the cases into pieces which can be recombined independently of one another. The incompatible abstraction problem is the problem of finding decompositions of a case that make sense in all abstractions simultaneously. For example, connecting two dead-end hallways by eliminating a

wall between them may be simple to represent in the geometrical abstraction, but in the abstraction of circulation patterns it implies profound changes which are by no means easy to model.

A first precondition for solving this problem is that all abstractions are mapped to a uniform and compatible representation on which decompositions can be defined. CADRE already uses such a uniform representation, but we have not yet developed the necessary decomposition methods for cases necessary for solving this problem.

5. Overview of the CADRE system

Computationally, the processes in CADRE can be divided into two layers: a symbolic layer and a numerical layer. They correspond to the topological and dimensional models of the case. CADRE focuses on case adaptation, and lets the user select the case to be used. Adaptation is conducted in the following steps:

1. Evaluation of the existing case in the original and new environments in order to find discrepancies. Insertion of the case into the new design context so that a maximum coincidence is achieved, subject to constraints posted by the user. In the example shown earlier in Figure 1, insertion is governed by the orientation with respect to sun exposure.
2. If there are dimensional discrepancies, identify the violated constraints and the parameters which are involved in them. Complete the set of applicable parameters and constraints with all those which are related to the original ones through cycles in the constraint network. This defines the complete *base set* of parameters and constraints related to the discrepancies. In the example of

Figure 1, one wing of the case does not completely fit into the new site, and the base set contains all parameters and constraints of this wing.

3. Apply *dimensionality reduction* to the base set of parameters and constraints to define an adaptation parameterization which is guaranteed to avoid conflicts. In the example, this results in two parameters, the width and length of the wing.
4. Modify the dimensions using the parameters resulting from dimensionality reduction. The user controls the process by asserting additional constraints or manually identifying suitable values whenever this is required to avoid ambiguities.
5. Check the validity of the adaptation by verifying inequality constraints in the base set that were not critical and thus not treated by the dimensionality reduction. In the example of Figure 1, purely dimensional changes would make the sizes of rooms fall below building code limits, and in fact this constraint turns out to contradict the floor space requirements.
6. If there is no solution at the dimensional level, trigger topological transformation rules to relax the constraint set. If there is a feasible transformation, apply it and go back to step 2, otherwise the case is not suitable. In the example, topological transformations rearrange spaces within the offending wing so that a solution becomes feasible (right of Figure 1).

Tests on several real examples and discussions with practicing engineers and architects lead us to believe that this procedure supports their activities.

To summarize, CADRE adopts the following solutions to the problems we have outlined earlier:

- superficial similarity: user interaction.
- adaptation knowledge: storing constraints and topological change rules with the case. We are experimenting with mechanisms to add these automatically from a knowledge base.
- inadmissible generalization: all abstractions are mapped to a single model.
- complexity of consistent modification: dimensionality reduction.
- complexity of matching: user interaction.

Since CADRE does not yet cover case combination, the incompatible representation and property assimilation problems are not addressed. Using these solutions, CADRE has completely achieved only one of the four potential advantages we hoped for: preserving implicit tradeoffs. Even though the adaptation knowledge stored with a case is much simpler than what would be required to generate solutions of the complexity handled by CADRE, adaptation cannot function without at least some domain knowledge. Furthermore, having to add constraints and topological modification rules to cases may make learning difficult; this is an issue we have yet to investigate. Finally, CADRE requires a significant amount of computation, although the fact that no existing automated design system can automatically generate solutions of similar complexity makes comparisons difficult. We can therefore only report partial success as far as the other three advantages are concerned.

On the other hand, in the course of implementing and using CADRE we have discovered another important advantage of case-based design over classical rule-based systems: that of integrating different abstractions, such as civil engineering and architectural considerations. This advantage is due to the fact that integration

can be guaranteed during much of the adaptation, and only few topological changes which might lose the integration are typically required. In fact, case-based design seems to be the only practical solution in domains such as building design where such different abstractions are important.

6. Conclusion: How far can the potential advantages of case-based design be achieved?

Case-based reasoning has been credited for its advantages in solving design problems. However, we have shown that case-based design poses fundamental problems, some of which may make it impossible to benefit from the promises of the paradigm. For example, there does not seem to be a way to guarantee the correctness of adaptations without using a complete domain model. Avoiding inadmissible generalization requires explicit integration of tradeoffs in several abstractions, so that it is questionable whether they can be implicit. An attempt to classify the problems by the advantages that they may invalidate is shown in Table I. The top line of the table shows the expected advantages of case-based design as described in the introduction. In the corresponding columns are the problems which are in conflict with the advantage. The degree to which the advantages of case-base design can be exploited depends on the solution to these problems.

Among the entries in Table I, we consider the opposition between the incomplete domain model and the adaptation knowledge problem as the most serious one, as it appears to be unsolvable by its definition. However, adaptation knowledge for cases can be simpler than generation knowledge for designs of comparable complexity,

Types of Advantages	Work with Incomplete Model	Implicitly Achieve Tradeoffs	Efficiency	Learning by storing cases
Opposing Problems	1. Superficial Similarity 2. Adaptation Knowledge 3. (Property Assimilation)	1. Inadmissible Generalization 2. (Incompatible Abstraction)	1. Complexity of Matching 2. Consistent Modification 3. (Property Assimilation)	1. Superficial Similarity 2. Adaptation Knowledge 3. (Incompatible Abstraction) 4. (Property Assimilation)

TABLE I
Classification of the fundamental problems that counter the advantages of case-based design. Entries in parenthesis refer to systems that would also use case combination.

as shown in the CADRE example. Depending on the domain, the other entries can be solved more or less by technical solutions, of which CADRE shows examples. In general, problems fall into three classes:

- those that cause computational intractability, and are solvable by smart computational mechanisms and by adding additional principles to narrow the focus. Examples are the complexity of matching and complexity of consistent modification problems.
- those that require additional domain knowledge, and are solvable by querying the user. Examples are the superficial similarity problem and the adaptation knowledge problem.
- those that are associated with different models and abstractions, and can be solved by mapping to a uniform representation. Examples are the inadmissible generalization problem, the property assimilation problem and the incompatible representation problem.

The applicability of case-based reasoning in a design system depends on the

performance expected of it, and the degree to which the domain permits solving the problems we described. The experience with CADRE has convinced us that the fundamental hypothesis of case-based design, that *adaptation is easier than generation*, is satisfied in building design. In the right domains, case-based reasoning is a promising tool for future intelligent design assistants.

7. Acknowledgements

The work described in this paper is funded by the Swiss National Science Foundation and was performed in collaboration with LIA, ICOM - EPF Lausanne and CAAD - ETH Zurich, Swiss Federal Institute of Technology. We thank Gerhard Schmitt, Ian Smith, Shen-Guan Shih and Simon Bailey for their collaboration and comments on this work and paper.

References

Birnbaum, L. and Collins, G. 1989. Remindings and Engineering Design Themes: A Case Study in Indexing Vocabulary. Proceedings of Workshop on Case-based Reasoning. 47-51.

Buchberger, B. 1985. Groebner Bases: An Algorithmic Method In Polynomial Ideal Theory. In: (Bose, N. K. ed.) Progress, Directions and Open Problems in Multidimensional Systems Theory. Dordrecht: Reidel, 184-232.

Faltings, B. 1991. Case-Based Representation of Architectural Design Knowledge. Computational Intelligence **2**, North-Holland.

Fenves, S. J., Flemming, U, Hendrickson, C., Maher, M.L. and Schmitt, G. 1989. An Integrated Software Environment for Building Design and Construction. Symposium Proceedings

for CIFE. Stanford University.

Goel, A. K., Kolodner, J. L. 1991. Towards a Case-based Tool for Aiding Conceptual Design Problem Solving. Proceedings of Workshop on Case-based Reasoning. 109-120

Golding, A. R., Rosenbloom, P. S. 1991. Improving Rule-Based Systems Through Case-based Reasoning. AAAI, 22-27.

Hammond, K.J. 1989a. Case-based Planning. Boston: Academic Press.

Hammond, K.J. 1989b. On Functionality Motivated Vocabularies: An Apologia. Proceedings of Workshop on Case-based Reasoning. 52-56.

Hinrichs, T.R. 1991. Problem Solving in Open Worlds: A Case Study in Design. PhD thesis. Georgia Institute of Technology.

Hua, K., Smith, I., Faltings, B., Shi, S. and Schmitt, G. 1992. Adaptation of Spatial Design Cases. Second International Conference on Artificial Intelligence in Design. CMU, Pittsburgh, USA, 559-575.

Kass, L. and Leake, D. B. and Owens, C.C. 1986. SWALE: A Program that Explains. Explanations Patterns: Understanding Mechanically and Creatively. Hillsdale, NJ: Lawrence Erlbaum Associates, 232-254

Kolodner, J. L. 1984. Retrieval And Organizational Strategies in Conceptual Memory: A Computer Model. Hillsdale, NJ: Lawrence Earlbaum Associates.

Kolodner, J. L. 1989. Judging Which is the *Best* Case for a Case-based Inference. Proceedings of Workshop on Case-based Reasoning. 77-81.

Maher, M. L., Zhang, D. M. 1991. Case-based Reasoning in Design. Artificial Intelligence in Design. Butterworth Heinemann. 137-150.

Owens, C. 1989. Plan Transformations as Abstract Indices. Proceedings of Workshop on Case-based Reasoning. 62-65.

Rosenman, M. A., Gero, J. S., and Oxman, R. E. 1991. What's in a Case: the Use of Case Bases, Knowledge Bases and Databases in Design. Proceedings of CAAD Future 1991, Zurich, Switzerland. 263-278.

Samuel, A.L. 1959. Studies in Machine Learning Using the Game of Checkers. IBM J. Research and Development **3**, 210-229.

Saund, E. 1989. Dimensionality-Reduction Using Connectionist Networks. IEEE trans. PAMI **11**, 304-314.

Schank, R. C. 1982. Dynamic Memory: A Theory of Reminding and Learning in Computers and People. London: Cambridge University Press.

Schmitt, G. 1990. IBDE, VIKA, ARCHPLAN: Architectures for Design Knowledge Representation, Acquisition and Application. In H. Yoshikawa, T. Holden (Eds.): Intelligent CAD II, North Holland.

Sycara, K.P., Navinchandra, D. 1991. Influences: A Thematic Abstraction for Creative Use of Multiple Cases. Proceedings of Workshop on Case-based Reasoning. 133-144.

Sycara, K.P., Navinchandra, D., Narasimhan, S. 1992. Parametric Adaptation in Case-based Design. Workshop on Case-Based Design, CMU, Pittsburgh, USA.

Wang, J. and Howard, H.C. 1991. A Design-dependent Approach to Integrated Structural Design. Artificial Intelligence in Design. Heinemann. 151-170.