Case-based Learning for Knowledge-based Design Support

Katy Börner[1] and **Dietmar Janetzko**[2]

Abstract. We present a general approach to combine methods of interactive knowledge acquisition with methods for machine learning. The approach has been developed in order to deliver knowledge required by support-systems for design-tasks. Learning rests upon a knowledge representation scheme for cases that distinguishes between knowledge needed for subgoaling and knowledge needed for design. We employ traces, i.e., protocols of the user's actions when tackling design-tasks as the initial input for incremental knowledge acquisition. This allows to learn task structures to be used for subgoaling and case-bases plus similarity relations applicable to particular case-bases.

1 INTRODUCTION

Integrating incremental learning into a knowledge-based systems seems to be a promising way to lessen the burden of knowledge elicitation to system development [9]. The goal of this paper is to point out how learning can be used in an interactive design-support system that uses CBR [8] as the main problem solving method. The next section briefly describes the domain we are dealing with. Section three sketches the computer–human–interaction and the role of traces of this interaction in learning. Section four presents the basic concepts used and the knowledge representation scheme employed. In section five the reasoning and learning procedures applied are presented. In the final part of the paper we discuss the approach introduced in this paper.

2 APPLICATION DOMAIN: INDUSTRIAL BUILDING DESIGN

The general application domain used to delineate our rationale of integrated knowledge acquisition and learning is industrial building design. Our focus is on installations in buildings with a complex infrastructure. Here, the main problem is how to layout subsystems for fresh and used air, electrical circuits, warm, cold, and used water, computer networks, phone cables, etc. Frequently, architects browse through old blueprints or drawings to solve new problems. Case-based reasoning seems to be the natural problem solving method that captures this procedure. In building design, CAD-like drawings correspond to snapshots of designed objects in problem solving.

[1] HTWK Leipzig, Department of Informatics, P.O. Box 66, 04251 Leipzig, FRG

[2] University of Freiburg, Department of Cognitive Science, 79098 Freiburg, FRG

In the sequel, we make use of a A4 [6], viz., a knowledge representation scheme developed to support computer-based building design. A4 allows to represent *objects* used in building design on a graphical level, which is the basis for man-machine interaction, and on a code level by an attribute-based representation, which is the basis for reasoning and learning.

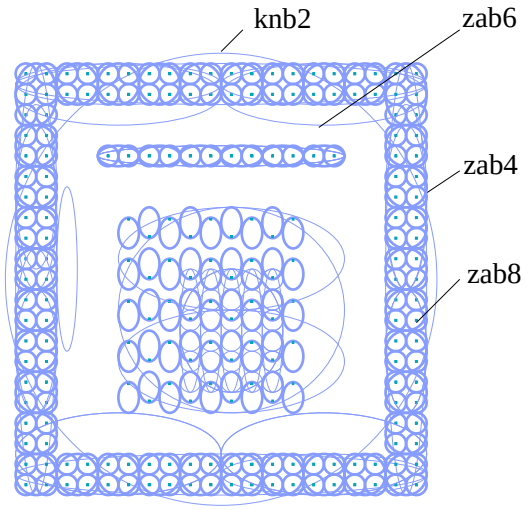

Figure 1. Snapshot of A4-objects constituting the fresh air supply net of some building. Type attributes of A4-objects, e.g., knb2, refer to states in design.

On the *graphical level*, A4-objects are represented by geometrical objects. Different states in the design of a building correspond to different configurations of objects represented by ellipses. Ellipses are a substitute for rectangles circumscribed by the ellipses. Using ellipses instead of rectangles is an unaccustomed but very useful trick: Ellipses overlap only in a few points. Thus, the readability of drawings becomes essentially improved. Fig. 1 represents some state of work during the design of the fresh air supply net of a building.

On the *code level*, A4-objects are represented by values for fixed sets of (i) geometrical attributes that describe placement and extension $\{x, dx, y, dy\}$ and (ii) type attributes {*aspect, morphology, resolution, size*} representing the state to which some object adheres. The values of the type attributes form also fixed sets: *Aspect* relates to the subsystem of the building (e.g., a = used–air, k = climate, r = room, s = shaft, w =

way, z = fresh–air). *Morphology* denotes the general function of the area addressed in a state (e.g., a = linkage, e = development, n = usage, v = connection). The third attribute specifies the kind of *resolution* employed (e.g., b = area, h = bounding-box). *Size* relates to the part of the building that is envisaged (e.g., 2 = building, 4 = floor, 6 = room, 8 = areas within a room). For example *zab4* describes the state of objects representing the area of supply–air linkage that covers the floor. Other objects represented in Fig. 1 are of type *knb2, zab6*, and *zab8*. The knowledge representation scheme A4 provides the basis to learn task-structures, cases-bases, and similarity relations.

3 A RATIONALE FOR USER-SUPPORT

We will now give an outline of how the knowledge acquisition is integrated into human–system interaction. The input for procedures of learning is provided by traces. The traces we use are nothing else but protocols of the computer–human–interaction that capture knowledge about actions performed by the user in design. The knowledge extracted from traces feeds into a design-support system. Support of human designers tackling a complex task may be divided into two parts. First, providing guidance to select the next subgoal, i.e., the next state to be tackled in design. Second, generating concrete design solutions automatically. In both cases the user is in a position to accept, modify, or reject the proposal offered by the system.

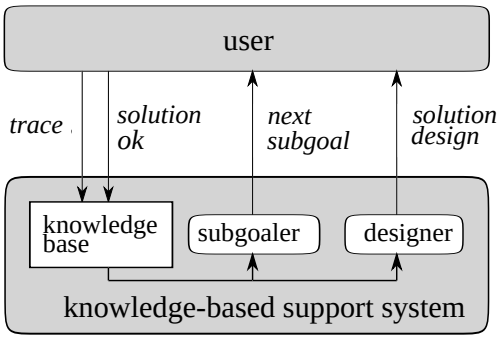

Figure 2. Human – system interaction

Figure 2 sketches the interaction of some human user and the knowledge-based system. During the work with DANCER [6], i.e., the knowledge-based CAD-system we employ traces are recorded that reflect a user's actions. Traces are used to elicit the knowledge provided by the user that is employed as an input for all learning procedures described in this paper. That is, knowledge acquisition proceeds by learning via analysis of the trace. Output of the learning procedures is a knowledge-base required by the system-modules subgoaler and designer. Given an actual situation the subgoaler suggests subsequent subgoals, and the designer presents concrete design solutions to an human user.

4 KNOWLEDGE REPRESENTATION

Now we take a closer look at the knowledge representation we use. We will give the representation of objects, states, and cases in an more accurate and domain independent way.

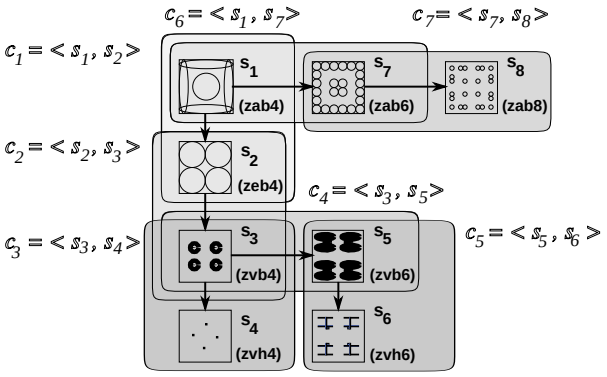

Figure 3. Object representation, state sequences, and cases in industrial building design

Illustrated in Fig. 3 are design states approached one after the other labeled $s_1, \ldots, s_8$. Each state corresponds to some arrangement of objects with identical types, e.g., (zab4), (zab6) etc. The cases $c_1, \ldots, c_7$ are subsequent states that are related to each other.

- **Objects** are represented by attribute values for a fixed set of attributes. This set of attributes is divided into the set of attributes representing geometrical object properties and the set of attributes representing the state to which an object belongs, i.e., its type. The former represents concrete geometrical attribute values of objects. The latter refers to peculiarities of states with its different case-bases and similarity relations.

- **Tasks:** Types are also used to represent the *tasks* that have to be accomplished. This means that the objects of a certain type are the realizations of the task of this type. If we collect all types that are employed during the design of a building we have the full list of tasks that have to be accomplished. Whenever we describe the string-like entry of a A4-object on a syntactical level we use the notion of *type*. Whenever we refer to problem solving activities like design, planning, or testing we use the concept of *task*

- **States:** We define a state s as a set of objects with identical type. That is, states may be represented by their type and the set of geometrical attribute values of all their objects. There exists some semi-ordering over states which represents the required object types needed to design a certain type of objects. These dependencies are represented in a task–structure [7]. For means of reasoning preconditions and resulting states are related to each other by cases. Depending on the stage in problem solving a state can either take the role of a problem or of a solution.

• **Cases** c are defined as pairs of *problem* and *solution*. While the problem is defined to represent the needed requirements, the solution represents the objects designed that fulfil these requirements. There is an important difference between the type and geometrical description of the objects constituting the solution. The former represents the subgoaling solution, i.e., the next subgoal, the latter represents the concrete design solution.

These formalizations provide the basis to introduce procedures of learning taken to realize incremental knowledge acquisition in design.

5 KNOWLEDGE ACQUISITION AND LEARNING

The general purpose of the learning procedures introduced in this section is to enable incremental knowledge acquisition. That means, the output of each learning procedure provides the input for the subsequent learning procedure. The initial input is provided by a trace, i.e., a list of the user's actions recorded during the work with the system. On this basis we

- preprocess the trace of user actions into a noise-free *trace*,
- identify and extract the states tackled one after each other together with their dependencies leading to tasks which altogether form the *task–structure*,
- collect task-dependent *case-bases*, and
- derive *similarity relations* over case problems.

We will now turn to an examination of the procedures to extract noise free *traces* and to learn *task–structures*, state-dependent *case-bases*, and the *similarity relations* over cases applicable to a particular case-base.

5.1 Preprocessing of the Trace

A trace is made up of user actions recorded during the design of some building. Traces represent sequences of states traversed successively. They provide information about the `initialisation, selection, resizing, deselection` of objects to be designed. Moreover, attribute-value assignments are recorded, which are required when designing buildings by using the A4 representation scheme. These are, e.g., set aspect (`setA`), set z-dimension (`setZ`), set morphology (`setM`), set time (`setTtag`) etc.

Traces are a valuable source of knowledge to be used as an input for learning procedures. However, they cover unnecessary or even wrong actions that have to be eliminated. As a consequence, traces have to be preprocessed in order to obtain noise-free knowledge suitable for further knowledge acquisition and learning. An example of a trace that is used as an input for preprocessing is given in Fig. 4. During the early design of a building named `school` there are two new objects created, viz., object number `1` and `2`. To both objects attribute values for hight, time, aspect, morphology, region, and solution are assigned. In the following, the objects are resized; this is reflected by new assignments of values for x, dx, y, dy. Finally, the object under consideration is deselected. During the creation of object 1 and 2 the object 1 was selected and immediately deselected reflecting a user mistake. Information relevant for the subsequent extraction of the task–structure

```
school 1 init
school 1 setZ 0
school 1 setTtag 9.10826e-149
school 1 setA gebaeude
school 1 setM nutzungs
school 1 setR bereich
school 1 setS 2
school 1 resize 100 480 50 480
school 1 deselect
school 1 select
school 1 deselect
school 2 init
school 2 setZ 0
school 2 setTtag 9.108387e-149
school 2 setA klima
school 2 setM nutzungs
school 2 setR bereich
school 2 setS 2
school 2 resize 100 480 50 480
school 2 deselect
...
```

Figure 4. Example trace

are state type (e.g., gnb2) and values for geometrical dimensions x, dx, y, dy of the created objects. Filters are used to extract these data. In Fig. 4 these data are marked boldly. The filter output of the example trace given in Fig. 4 would be:

```
[gnb2:1] 100 480 50 480
[knb2:1] 100 480 50 480
```

That is, object 1 is the only object of type `gnb2` that is created. Object 2 is the first object of type `knb2`. Given such a noise-free trace we are able to extract the task–structure.

5.2 Learning the Task–Structure

The overall design task may be divided into sequences of subtasks to be tackled. There are tasks which have to be accomplished before others can be addressed, i.e., the creation of some objects provides the knowledge required for the design of other objects. A task–structure captures the knowledge used in subgoaling, which refers to dividing an overall task into a sequence of subtasks or subgoals.

In building design, however, it is difficult to acquire preconditions and effects of certain design steps. Noise free traces provide the input to learn the task–structures, i.e., complex structures corresponding to the alternatives and peculiarities when designing buildings.

Figure 5 provides an example for the task–structure of the design of fresh-air and used-air supply nets. The example output described in the last subsection, i.e., the state transition between `gnb2` and `knb2`, is the input to the procedure of learning the task–structure. The output of the procedure is the task–structure that starts with the design of the building usage region (`gnb2`) at the root node and continues with the design of the climate usage region (`knb2`). This step is necessary to design the rooms (`rnb4`), ways (`wnb4`), and shafts usage

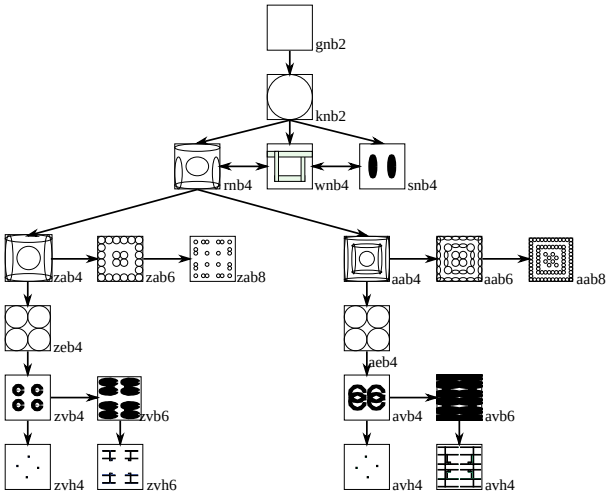

Figure 5. Task–structure

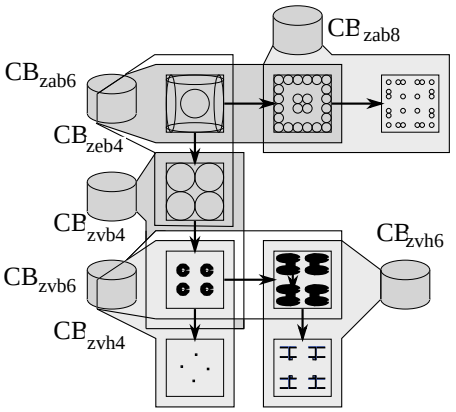

Figure 6. Task dependent case-bases

areas (snb4) in the sequel. Double arrows between the latter three state-types denote dependencies between them during design. Given all these design decisions we are now able to start the design of the fresh-air and used-air supply nets.

That is, the task–structure provides some semi–ordering over the set of object types. At the same time it expresses requirements to design specific objects. Parallel or alternative ways in design may also be represented. Determination of object types, i.e., goals to be accomplished becomes possible. Aiming at a task–oriented user support, the task–structure even provides guidance to partition complex realization of design-tasks into cases as we will see next.

5.3 Learning Case-Bases

Once a task-structure is specified, a kind of template sheet is introduced into the domain that may be employed for discerning and acquiring cases. Projecting such a template sheet or grid onto a domain is nothing more than an attempt at making up for natural units that are ambiguous, hidden, or even missing. This is an indispensable requirement for using case-based reasoning in domains that have no units suitable to be referred to as problem or solution.

We assume that different tasks in design require different design strategies. For example, while return air accesses will be connected by using the shortest path, connections for fresh air accesses take curved tracks to achieve noise reduction.

Case-bases are formed in a way that preconditions necessary to design a certain object type are represented as case problems, and the state to be designed is represented as their solution. Assuming that each case solution contains states of one type (e.g., zab6), case-bases are indexed by this type (e.g., CB_{zab6}, see Fig. 6). Note, that in the example given in Fig. 6 each problem description also contains only objects of one type. This has not necessarily to be the case since the specification of a problem may very well relate to quite a number of different tasks.

5.4 Learning Similarity Relations

Finally, we need to derive similarity relations over case-sets to be used for retrieval of prior cases in problem solving. Two-stage similarity assessment [5] is widely accepted as an efficient way to case-selection in CBR. While *attribute-based similarity assessment* is used for fast preselection to determine a set of candidate cases, *structural similarity assessment* allows for structural comparisons as a basis for adaptation. We will now describe how the corresponding similarity relations are learned.

Attribute-based similarity assessment proceeds by comparing weighted attribute values. Given some similarity measure (e.g., Hamming distance) we learn about the relevance of certain attributes for reasoning. That is, input are cases grouped to one case-base or case-set (e.g., cases with identical solutions). Output are weights characterizing the relevance of attributes for this classification.

Structural similarity assessment uses structural case representations (e.g., terms-based or graph-based ones) to determine similarity. Using the underlying term algebra (or graph algebra) and the concept of *antiunification* [10] we are able to learn inductively about proper substitutions. Applying these substitutions to some set of cases we derive their *least general anti-unifier* also called *most specific generalization* [11] which may be seen as a kind of prototype of this case set. This has been discussed in detail elsewhere [2].

As an example for using prototypes in similarity assessment imagine a prototypical room. This room, which may not exist, is representative for all rooms you have ever seen. Remembering concrete rooms (cases) permits to apply the proper modification rules which lead to this prototype. Entering an unseen space, you will be easily able to recognize if it is a room. You simply apply proper modifications and compare the result with the corresponding room prototype ("generate and match"). Knowing about prior concrete rooms permits to derive more knowledge about different types, forms, usages, etc. of rooms.

The set of modification rules (e.g., proper substitutions, deletions of objects) of a case-base together with the prototype of this case-base will be used to determine structural similarity between this case-base and an actual problem. Struc-

tural similarity assessment proceeds by selecting rules out of the rule-set and applying them to the actual problem. If this application yields the prototype of the case-base the actual problem is said to be structurally similar to this case set. Adaptation proceeds by transferring operational solution descriptions using the concept of *derivational analogy* [3, 4]. Note, that the application of identical operations to different terms (problems) results in different solutions.

6 DISCUSSION

System design that rests upon an immense effort for manual knowledge elicitation will hardly be successful. There is simply no time to feed in all knowledge required. System architectures that enable knowledge elicitation and learning thereby increasing user support during the system usage may provide a way out. This can only be achieved by combining learning and knowledge elicitation techniques such that knowledge elicitation is used to establish a framework for a knowledge-based system the knowledge-base of which is built or enlarged by using learning techniques. Developing such frameworks are a real challenge to enable useful and adaptable computer-aided support in design. We delineated an approach to incorporating learning into design-support systems. Differing from other approaches that pursue a similar goal [9] our emphasis is on case-based reasoning as the major problem solving method. Consequently, the learning procedures sketched deliver knowledge required by case- based reasoning. In our approach, incremental learning is accomplished by using a trace of the user's design activities, i.e., by recording states traversed by the user. The system incrementally learns state transitions, i.e., preferred sequences of states, cases, and similarity relations. The knowledge learned allows for a support of subgoaling, similarity assessment, and adaptation in building design. Each time the system lacks knowledge to solve a certain problem this problem is allocated to the user. But this is not simply a surrender of the system since each task allocation to the user triggers a well-defined learning-task of the system with the input, the learning-goal, and the background knowledge specified. By this close linkage of knowledge acquisition, problem solving, and learning we are able to adapt the system to the user and the pecularities of the domain chosen.

The algorithms underlying the preprocessing of the trace and the extraction of the task–structure have already been implemented and tested. The determination of similarity relations over cases and their usage for similarity assessment and adaptation constitute the backbone of a program called *SynTerm* (for **Syn**thesis by using **Term**-based knowledge representations). This has been discussed in detail elsewhere [1].

ACKNOWLEDGEMENTS

This research was supported by the German Ministry for Research and Technology (BMFT) within the joint project FABEL under contract no. 413-4001-01IW104. Project partners in FABEL are German National Research Center of Computer Science (GMD), Sankt Augustin, BSR Consulting GmbH, München, Technical University of Dresden, HTWK Leipzig, University of Freiburg, and University of Karlsruhe.

REFERENCES

[1] Katy Börner, 'Structural similarity as guidance in case-based design', in *Topics in Case-Based Reasoning Selected Papers from the First European Workshop on Case-Based Reasoning (EWCBR-93)*, eds., Stefan Wess, Klaus-Dieter Althoff, and Michael M. Richter, volume 837 of *Lecture Notes in Artificial Intelligence*, 197–208, Springer, (1994).

[2] Katy Börner, 'Towards formalizations in case-based reasoning for synthesis', in *AAAI-94 Workshop on Case-Based Reasoning*, pp. 177–181, (1994). also FABEL Report 22.

[3] Jaime G. Carbonell, 'Derivational analogy : A theory of reconstructive problem solving and expertise acquisition', in *Machine Learning: An Artificial Intelligence Approach*, eds., Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, volume 2, Morgan Kaufmann, Palo Alto, CA, (1986).

[4] Jaime G. Carbonell, Craig A. Knoblock, and Steven Minton, ' PRODIGY: An integrated architecture for planning and learning', in *Architectures for Intelligence*, ed., Kurt VanLehn, Erlbaum, Hillsdale, NJ, (1990). Also Technical Report CMU-CS-89-189.

[5] Dedre Gentner and Kenneth D. Forbus, 'MAC/FAC: A model of similarity-based retrieval', in *Proceedings of the Cognitive Science Conference*, pp. 504–509, (1991).

[6] Ludger Hovestadt, 'A4 – digital building – extensive computer support for the design, construction, and management of buildings', in *CAAD Futures '93, Proceedings of the Fifth International Conference on Computer-Aided Architectural Design Futures*, pp. 405–422. North-Holland, (1993).

[7] Dietmar Janetzko, Katy Börner, Oliver Jäschke, and Gerhard Strube, 'Task-oriented knowledge acquisition for design support systems', in *First European Conference on Cognitive Science in Industry*, (1994).

[8] Janet L. Kolodner, *Case-based reasoning*, Morgan Kaufmann, 1993.

[9] Katarina Morik, 'Balanced cooperative modelling', in *Machine Learning. Vol. IV*, eds., R. Michalski and G. Tecuci, pp. 295–317. Morgan Kaufmann, (1994).

[10] Steven Muggleton, *Inductive logic programming*, Academic Press, 1992.

[11] Gordon D. Plotkin, 'A note on inductive generalization', in *Machine Intelligence 5*, eds., B. Meltzer and D. Michie, 153–163, American Elsevier, (1970).