Explanation-Based Retrieval in a Case-Based Learning Model

Gerhard Weber

Department of Psychology
University of Trier
D-5500 Trier, Germany
weber@uni-trier.dbp.de

Abstract

Retrieving previous similar cases from a memory of cases is central to case-based reasoning systems. In most systems, this retrieval is done by a detailed indexing mechanism. Thagard and Holyoak argue that indexing is the wrong way to retrieve analogues. They propose a retrieval model (ARCS) based on a competing constraint-satisfaction approach. In this paper, an explanation-based retrieval method (EBR) for retrieving analogues from a case-base with cases stored with respect to an interpretation of these cases as analyzed by a cognitive diagnostic component is described. The system is designed to the domain of problem solving in LISP. In a simulation study, it can be shown that the EBR-method performs equally well or even better than the ARCS-method.

Analogical Retrieval and Indexing

Both case-based reasoning (CBR) and finding analogies are concerned with retrieving previous cases and problem solutions. In CBR-systems, cases are usually retrieved by elaborated indexing mechanisms. Thagard & Holyoak (1989) argue in their contribution to the 1989 Case-Based Reasoning Workshop that indexing is the wrong way to retrieve analogues. They make their points on two different levels of argumentation. On a more general level, they claim

- that CBR when viewed as a cognitive model is based only on anecdotes,
- that CBR generally is concerned with analogies within a single domain only, and
- that their own model for analogical reasoning is a component of a more general cognitive architecture.

On a more detailed level they claim about indexing,

- that it is excessively serial instead of simultaneously probing into memory,
- that the retrieval process is not competitive,
- that indexing overemphasizes pragmatic features such as goals and prediction failures, whereas semantic features and structural similarities are underemphasized, and
- that indexing requires to much pre-processing.

In this paper, it will be shown how a CBR-system can be built in a way that most of these arguments can be rejected. Such a system retrieves analogues as good as the ARCS-program (Thagard et al. 1990) does, or even better.

As Thagard et al. (1990) point out, five aspects central to analogue retrieval in human memory are implemented in their ARCS-program. These are the constraints of (1) semantic similarity, (2) structural consistency, and (3) pragmatic centrality. Additionally, (4) a parallel algorithm has to be used for determining stored analogues fitting best these three constraints and (5) there must be a competition between potential candidates for source analogues.

In analogical retrieval (compared to mapping analogues) finding semantic similarities between elements and relations in the target analogue and in the source analogues in memory is essential (Holyoak & Koh 1987; Holyoak & Thagard 1990). But, structural and pragmatic constraints also contribute to the retrieval process. For most analogue retrieval models, it is a problem to consider all constraints simultaneously like it is done in ARCS.

In case-based reasoning systems, the retrieval of similar cases depends on indexes used to probe the case base. These indexes consist of relevant features of the input problem determined in a first analysis phase (Riesbeck & Schank 1989). For CBR-systems, it is critical how indexes can be found, how specific they are to identify the most similar cases, and how common they are, so that they apply to a wide range of possible cases.

To address these problems of indexing in CBR-systems we have built an 'Episodic Learner Model' (ELM), a case-based learning model in the domain of learning LISP. In this model, previous cases are stored such that information from stored cases can be used to analyze new cases. These interpreted input problems can be used directly to access similar previous cases. The ELM-model is used as a student model in an intelligent tutoring system for the programming language LISP (Weber 1988). One of the intelligent features to support tutoring is an analogical component. This component searches for similar cases to a given situation to give the tutorial component the opportunity to explain to the student his or her bugs and misconceptions related to previous errors and to solutions for these errors. One of the central tasks of this component is to retrieve analogous cases. As the ELM-LISP-Tutor is an on-line system, the retrieval process has to be fast enough to respond to the student within acceptable time. So, retrieving a candidate for an analogue with an acceptable effort in memory space and time is one further constraint for the retrieval process.

In the next section, the ELM-model is described to show how cases (LISP-programs students have completed to solve programming problems) are interpreted and stored. In the following section, the algorithm of the explanation-based retrieval method (EBR) is described. Finally, this method is compared to the ARCS-model by a simulation study.

ELM: A Case-Based Learning Model

The ELM student model is a type of user model containing knowledge about the user (student) in terms of a collection of episodes. In the sense of 'case-based learning', such episodes can be viewed as cases. To construct the student model, the student's code is analyzed related to the domain knowledge on the one side and to a task description on the other side. This cognitive diagnosis results in a derivation tree of concepts and rules the student might have used to solve the problem. These concepts and rules are instantiations of units from the knowledge base. The episodic student model is made up by these instantiations and later generalization based thereupon. To understand this form of episodic student model, a short description of the knowledge representation and the diagnostic process will be given.

Students have to program function definitions in a structured LISP-editor (Köhne & Weber 1987). So, the function code is at least syntactically correct. A series of a student's attempts to solve a programming problem from the exercises is shown in Tab. 1. The "cognitive analysis" of the program code works like an explanation-based generalization (EBG) method (Mitchell, Keller, & Kedar-Cabelli 1986; DeJong & Mooney 1986). It starts with a task description related to higher concepts, plans, and schemata in the knowledge base. For every concept, a set of rules is stated describing different ways to solve the goal given by the plan of the concept. These rules are comparable to the notation of implementation methods for goals in PROUST (Johnson 1986). There are "good", "suboptimal", and "buggy" rules explaining correct, correct but suboptimal, and incorrect solutions of the current goal or subgoal. The set of buggy rules is comparable to an error library in other ITSs (e.g. in Anderson's LISP-tutor (Anderson & Reiser 1985)). Applying a rule results in comparing the current plan description to the corresponding part of the student's code. In the plan description, further concepts may be addressed. The cognitive diagnosis is called recursively until a function name, a parameter, or a constant is matched.

The cognitive diagnosis results in a derivation tree (Weber 1989) built from all concepts and rules identified to explain the student's solution (Tab. 2). This derivation tree is an explanation structure in the sense of EBG and is the basis to build up the episodic learner model. Concepts and rules addressed in the derivation tree are the basis to create episodic frames. These frames are integrated into the knowledge base as instances of their concepts and rules. Slots in these instances refer to the context where they were used (especially the current task), to the type of transformations of concepts, to the observed rules, and to the argument bindings. The set of these instances (and further generalizations) constitutes the episodic student model.

In the next step, episodic frames are generalized. These generalizations represent the class of types of plans and corresponding data which will be used in further cognitive analyses to interpret the student's code. Generalizations refer to structural and semantic similarities of observed data for related concepts and rules. So, this is a form of similarity-based generalization within the framework of an EBG-method. If an episodic frame is the first instance under a concept of the knowledge base, this single case is generalized from structural and semantic aspects in the data so that this generalization will explain further matching data. In Fig. 1, a hierarchy of episodic instances and generalizations under the concept "THIRD-ELEM" is shown. The first generalization frame THIRD-ELEM.GEN-1 was obtained by a single-case generalization after diagnosing Solution 1 from Table 1. The frame THIRD-ELEM.GEN-2 was generalized after the student's second attempt to solve the problem with a different plan. The third generalization frame THIRD-ELEM.GEN-3 resembles the communalities of the second and third attempt to solve the problem. In Tab. 3 the generalized structures for DATUM-slot and for the PLAN-RULE-SEQUENCE-slot are shown.

With increasing knowledge about a particular student, hierarchies of generalizations and instances are built under the concepts and rules of the knowledge base. They constitute the episodic student model. This generalization mechanism is comparable to the single-case generalization in the "explanation-based learning" approach (Mitchell et al. 1986). One single event (or case) is interpreted considering the knowledge base and the student model and the result of this interpretation is integrated into the student model.

Problem:
"Define a function that returns the third element of a list."

Solution 1:
```
(DEFUN THIRD-EL (EXPR)
   (REST (REST (FIRST EXPR))))
```

Solution 2:
```
(DEFUN THIRD-EL (EXPR)
   (SECOND (FIRST (REST EXPR))))
```

Solution 3:
```
(DEFUN THIRD-EL (EXPR)
   (SECOND (REST EXPR)))
```

Table 1: A series of subsequent attempts to solve a programming problem from the exercises used in our LISP-course.

```
(THIRD-ELEM
 Third-Elem-With-Second-Rule
  (SECOND-ELEM-CALL
   Unary-Func-Rule
    (SECOND-ELEM-OPERATOR
     Correct-Coding-Rule)
    (REDUCE-LIST
     Unary-Func-Rule
      (REDUCE-LIST-OPERATOR
       Correct-Coding-Rule)
      (PARAMETER
       Correct-Param-Rule))))
```

Table 2: Explanation structure for the body of the function definition of solution 3 in Table 1.

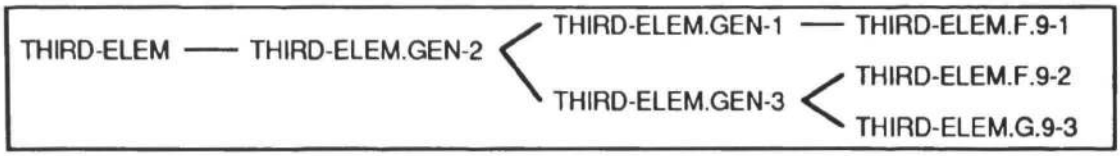

Figure 1: Hierarchy of episodic instances and generalizations under the concept THIRD-ELEM for the three solutions in Table 1.

?DATUM (SECOND (<SELECTOR-OPERATOR> ?ARG-273)) ?PLAN-RULE-SEQUENCE (THIRD-ELEM Third-Elem-With-Second-Rule !SEQ-273)

Table 3: Part of the slots of the generalization frame THIRD-ELEM.GEN-3 from Fig. 1.

This EBL-method implies that not only pragmatic aspects are stored in the case memory (as in most other case-based reasoning systems). The hierarchy of stored solutions for different subproblems resembles the structure of the solution. Stored and generalized data structures emphasize semantic similarities and may also be considered to observe superficial aspects of similarity.

In the context of the ELM-model in the ELM-LISP-tutor, two typical situations emerge where analogous cases are retrieved. On the one side, analogues are needed by the tutorial component to offer the student remindings and analogies to examples in the materials when an error or a suboptimal solution is detected by the diagnostic component in the student's program code. Such remindings can be used to explain to the student how the error is similar to a previous error and how the problem can be solved, or how the incorrect part of the code resulted from superficial similarities to previous solutions which were incorrectly mapped to the current problem. On the other side, if the student asks for help while developing the code for the current problem the tutorial system should be able to offer the student analogies to similar previous problems or examples. Here, the analogical component has to look for analogies for the solution of the next subproblem to be solved in the context of subgoals solved up to that point.

In both cases, existing program code is the basis to retrieve analogous cases from the case-base. In the next section, it is explained how an explanation-based retrieval method (EBR) is suited to find analogues in a case-memory of a case-based learning system.

The Explanation-Based Retrieval Method

On the basis of the explanation structure stored into distributed frames for the steps of the solution path, it is easy to retrieve analogical problem solutions for the current problem. There are two different situations where an analogue may be retrieved.

Similar solution situation. In this situation, the retrieval component is given a solution of a known problem and has to look in the case memory for the best analogues to the given solution. These analogues may help to interpret how the person solved the problem in the given way or why an observed error may have happened.

Similar problem situation. In this situation, the retrieval component is given a problem and possibly, a partial solution. By its knowledge about the problem, a solution can be generated considering the case memory and the partial solution. On the basis of the concepts and rules used to generate this solution, the case memory can be probed for cases most similar to this solution and analogues can be found for missing parts of the partial solution. As different subproblems have to be solved and gathered for the final solution there may be several different analogues which may match best the subproblems. This is a by-product of the distributed storage of problems in terms of solutions for the different subproblems. So, analogies to the subproblems may be detected in different previous cases which, therefore, may be retrieved as analogues.

Both situations differ with respect to the question whether the problem is already solved or not. In the first case, the already existing solution is used for analogy, in the latter case, a solution generated automatically is used for an interpretation and for retrieving analogues afterwards.

The algorithm of the EBR-method works in five steps as follows:

Step 1: Diagnose the solution of a problem which is either submitted by the problem solver or generated automatically by the system. This will result in an explanation structure of all concepts and rules which may be used to solve the problem.

Step 2: Store all elements of the explanation structure into episodic frames of the case memory and generalize if possible. If the solution was generated automatically by the system, store episodic frames and generalizations only temporarily.

Step 3: Scan all episodic frames of the new (target) episode and look for similar episodic frames in previous episodes. All those episodic frames subsumed under the same direct generalization frame are given the highest similarity weight. For other similar episodic frames subsumed under higher levels of abstraction or indexed by related rules assign lower similarity weights, accordingly.

Step 4: For each episode considered in Step 3, sum the similarity weights of the similar frames assigned in Step 3. For each episodic frame of the source episode only one similar frame of a previous episode may be considered.

Step 5: Sort the considered target episodes by their summed weights of episodic frames similar to the source episode computed in Step 4.

In the first place, this algorithm pays attention to semantic similarities of concepts, plans, and rules identified by the analysis of the program code. But, as structural

similarities play an important role in the generalization of episodic frames, structural consistencies are considered, too. Pragmatic aspects are not considered directly in the current version of the EBR-algorithm. But, it would be easy to give special pragmatic weights to observed buggy rules and to poorly solved concepts so that corresponding episodic frames from previous cases can dominate the retrieval of these episodes. So, tutorial goals could give pragmatic constraints to the retrieval process. An algorithm to retrieve analogues for special subgoals is still under development.

Using an explanation structure in the EBR-method to retrieve analogues cases is comparable to the construction of a hypothesis tree in the explanation-based indexing (EBI) approach (Barletta & Mark 1988). But, the EBR-method differs from the EBI-method in the distributed representation of episodic instances of concepts and rules.

A Simulation Study Comparing the EBR-Method with the ARCS-Model

The ARCS-model (Thagard et al. 1990) is supposed to be a general model for analogue retrieval. As the algorithm of this retrieval method is fully described in Thagard et al. (1990) and easy to compute, it can directly be compared to other methods in a simulation study. One of the basic assumptions of the ARCS-model is to represent the target and the potential source candidates in terms of the predicate calculus. Therefore, it is critical for this method how the description of a problem is decomposed into predicates.

For retrieving analogues of LISP-programs from a case-library of stored LISP-programs, the decomposition into predicates can be computed by an explicit algorithm. As LISP-programs are composed of hierarchically nested function calls, they can be decomposed into predicates as shown in Holyoak & Thagard (1989). Function calls are represented as predicates with n+1 arguments where n arguments stand for the arguments of the function and the additional argument for the result of the function call. Only special forms (e.g. schemata for defining functions with "defun" (special decomposition for variable list and function body) or schemata for case decisions with "cond") are handled specifically to reflect their semantics. An example for a decomposition of a simple function definition into a set of predicates for the ARCS-method is shown in Tab. 4A.

Identifying semantic similarities is the second critical point for the ARCS-method. As mentioned in several studies (e.g. Holyoak & Koh 1987; Ratterman & Gentner 1987; Ross 1987) semantic similarities are most important for retrieving analogues. In their ARCS-model, Thagard et al. (1990) use WordNet (Miller, Fellbaum, Kegl, & Miller 1988) to identify different kinds of semantic relations (e.g. synonyms, superordinates, coordinates, subordinates, holonyms, antonyms, etc.) and assign to them different weights determining their influence on the retrieval process.

Since analogs of LISP-programs are retrieved, the case of within-domain analogies is given. Therefore, most semantic similarities stem from the identity of function names. Further degrees of semantic relations can be obtained from hierarchically clustering functions by their meaning.

A) Decomposition of code into propositions:

(DEFUN (THIRD-EL Varlist-931 form-933) P-937)
(VARLIST (EXPR Varlist-931) P-932)
(SECOND (arg-934 form-933) P-936)
(REST (EXPR arg-934) P-935)

B) Decomposition of diagnosed code into propositions:

(DEFINE-PROCEDURE
  (Proc-Def-Rule arg-938) P-948)
(BODY-FORMS
  (Rule-For-Body arg-939 arg-938) P-947)
(THIRD-ELEM
  (Third-Elem-With-Second-Rule arg-940 arg-939) P-946)
(SECOND-ELEM-CALL
  (Unary-Func-Rule arg-941 arg-940) P-945)
(REDUCE-LIST
  (Unary-Func-Rule arg-942 arg-941) P-944)
(PARAMETER
  (Correct-Param-Rule arg-942) P-943)

Table 4: An example for a decomposition into propositions for A) LISP-code and B) the result of the explanation-based diagnosis for Solution 3 from Tab. 1.

For example, all predicates or all arithmetic functions are grouped into clusters of lower similarity, type predicates, arithmetic functions for performing addition, types of case-decisions, and so on, within these clusters are building smaller clusters with medium similarity, and synonyms (e.g. car and first) are linked on a high level of similarity.

One can argue that there is much preprocessing in the EBR-method diagnosing the LISP-code by the ELM-model to get the explanation structure. Also, the explanation structure may contain more information so that the EBR-method is able to retrieve better analogues than the ARCS-method based only on the pure LISP-code. As the explanation structure consists of a hierarchy of nested concepts and rules, this nested list can be decomposed into propositions, too. Different weights of similarity for concepts and rules were obtained from the hierarchy of concepts and rules in the knowledge base. A decomposition of diagnosed code into propositions is shown in Tab. 4B.

Our implementation of the ARCS-method was run with the same values for parameters as described in Holyoak & Thagard (1989) and Thagard et al. (1990) with decomposing LISP-programs and assigning weights for semantic similarities as described above.

Data for the simulation study were gathered from interaction protocols of 13 students computing function definitions for exercises of an introductory LISP-course. All function definitions were programmed in a structured LISP-editor (Köhne & Weber 1987), so they were at least syntactically correct. The range of programming problems comprised 5 lessons beginning with the first exercises defining LISP-functions to programming recursive functions. In these five lessons, 35 different problems could be solved by the students. All student's attempts to evaluate a complete function definition were considered as candidates for targets of analogies.

For each student, the sequence of solutions for different tasks within each lesson was matched step by step against all previous solutions within the same lesson and against the examples from the materials for these lessons. Retrieving analogues was restricted to analogies within one lesson for two reasons. First, examples and solutions within a lesson are thematically more similar than those between lessons. In simulations considering data from all lessons, only very few analogies were found between lessons. Second, the computational effort to retrieve analogues by the ARCS-method from a basis of more than 10 possible candidates increased dramatically, consuming too much time and space on a serial computer. All in all 406 cases were observed where analogue cases could be retrieved.

The same procedure to retrieve analogues for observed problem solutions step by step within each lesson was performed by the EBR-method. Time to retrieve an analogue by this method ranged from 0.05 seconds to 2 seconds for the range of problems described above. This is more than 200 times faster than the time needed to build up the network of hypotheses only. If we consider a minimum of 7 cycles to retrieve the best analogue, the ARCS-method takes more than 2000 times longer on a serial computer.

The results comparing both methods by the described procedure are shown in Tab. 5. In about 3 out of 4 cases, both methods (EBR and ARCS) retrieved the same analogue as their best result. This was computed by an algorithm comparing rank orders of retrieved episodes. As expected, the correspondence between both methods is slightly higher if the ARCS-method is run on a propositional decomposition of the explanation structure (77.3%) compared to a decomposition of the LISP-code (72.7%). When there was a discrepancy between the results of both methods two experienced LISP-programmers rated which of both methods retrieved the best analogue or whether both methods failed.

In less than 5% of all cases, the ARCS-method retrieved the better optimal analogue. With 4.7%, the ARCS-method worked slightly better with a propositional decomposition of the LISP-code (compared to 2.7% with interpreted code). In 1.7% of all cases, this resulted from errors or combinations of errors in the student's LISP-code which could not sufficiently be interpreted by the diagnostic component of the ELM-model. So, a retrieval process must be preferred which is based particularly on semantic similarities of predicates without knowledge about intentions and meaning of the program code and interpretations of errors. In most other cases, the ELM-method failed because the current generalization mechanism in the ELM-model is slightly too sensitive to structural differences in the student's LISP-code.

In about 6% of all cases, different analogues were retrieved as best results, but both analogues could be accepted equally well. In these cases, there was a very similar (and possibly buggy or incomplete) previous solution and additionally, a similar good example. In most cases, the correct similar example was preferred by the EBR-method whereas the similar, but possibly incorrect, previous solution was preferred by the ARCS-method.

		diagnosis	code
same		77.3	72.7
different	both methods equally well	6.4	5.9
	EBR better	11.6	15.0
	ARCS better	2.7	4.7
	both methods failed	2.0	1.7

Table 5: Percentage of cases showing the same or different best retrieved analogues by the EBR- and the ARCS-method for a total of 406 cases. In the "diagnosis" column, the ARCS-method was run with a propositional decomposition of the explanation structure from the cognitive diagnosis in the ELM-model, in the "code" column, the ARCS-method was run with a propositional decomposition of the LISP-code.

In more than 10% of all cases, the best analogue was retrieved by the EBR-method. In many of these cases, this could be attributed to two problems inherent to the constraint-satisfaction algorithm of the ARCS-method. In at least 8 cases (2%), the so-called "gang-effect" (McClelland & Rumelhart 1981) was observed. In these cases, very similar or identical solutions for a similar problem existed in the case base because the student solved the same problem twice with the same solution or recoded an example from the materials. These identical solutions grouped to a gang and supported each other while another better analogue was suppressed. In at least 15 cases (3.7%), a problem occured which we call the "additional-proposition-effect". This means that if two solutions differ in that one has an additional proposition, this solution was preferred though the other one was more similar. This must result from more supporting relations in the net of hypotheses in the case of the additional proposition.

In 2% of all cases both methods failed. That is, the retrieved cases could not be accepted as good analogues. In most cases, this resulted from the absence of appropriate examples in the case-base or from very buggy and uncommon solutions.

Conclusion

The ARCS-model (Thagard et al. 1990) is a very general and well suited program to model human retrieval. But, there are some questions about the efficiency of its algorithm. Thagard & Holyoak (1989) argue that indexing in case-based reasoning systems needs too much preprocessing and therefore, may not be appropriate. But, the ARCS-model needs very much time to build up the network of hypotheses. Even if the following process of settling the network is performed on a parallel computer and therefore, will be very fast, the network is temporarily built up only once for one retrieval process. For the next retrieval of another analogue a new network has to be constructed and for a little bit more complex problems than usually shown in examples, this is very much space and time consuming. For example, in the data of our students it is very typical that in the final lesson more than 40 previous

solutions and examples have to be considered as possible candidates for source analogues. Probing all candidates in parallel with the ARCS-model resulted in more than 3500 nodes and more than 200,000 symmetric links to be built up. Even on a fast 12 MIPS computer this requires more than 30 MByte of memory and more than 10 minutes to build up the network. If we extrapolate to situations in typical human retrieval situations where many more previous situations may be candidates for analogue retrieval because of overlap in their semantics, it cannot be imagined how such a problem could be handled by the ARCS-algorithm. An incremental model (Keane 1990) may be better suited to retrieve analogues from a large set of cases.

In this study, it could be demonstrated that the ARCS-methods has some problems if analogues have to be retrieved from a case base where possibly very similar previous cases exist. As shown for the "gang-effect" and the "additional-proposition-effect", the ARCS-method may fail if some special constellations of very similar cases exist. Maybe, this could not be observed in other studies with the ARCS-method because all cases in these examples differed in many aspects. But, in realistic, non-experimental situations, for example while learning to program, it is typical that many very similar cases are observed. In such a situation, the EBR-method seems to yield generally better results.

The EBR-method can be seen as an approximation to a competitive, parallel method of analogue retrieval as realized in the ARCS-model. As powerful parallel computers cannot be used in most cases, the EBR-method may be preferred in on-line situations (e.g. in tutorial systems). Additionally, information can be gathered about the mechanism how concepts and episodic structures are stored and retrieved from human memory.

Acknowledgments

This research was supported by the Deutsche Forschungsgemeinschaft under Grant We 498/12.

References

Anderson, J. R., and Reiser, B. J. (1985). The LISP tutor. *Byte* 10(4):159-175.

Barletta, R., and Mark, W. (1988). Explanation-Based Indexing of Cases. In J. L. Kolodner (Eds.), Proceedings of the DARPA-workshop on Case-Based Reasoning, 50-60. Los Altos, CA: Morgan Kaufmann Publishers.

DeJong, G., and Mooney, R. (1986). Explanation-based learning: an alternative view. *Machine Learning* 1:145-176.

Holyoak, K. J., and Koh, K. (1987). Surface and structural similarity in analogical transfer. *Memory & Cognition* 15:332-340.

Holyoak, K. J., and Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Science* 13:295-356.

Holyoak, K. J., and Thagard, P. (1990). A constraint-satisfaction approach to analogue retrieval and mapping. In K. J. Gilhooly, M. T. G. Keane, R. H. Logie, & G. Erdos (Eds.), *Lines of thinking* (Vol. 1, pp. 205-220). Chichester, England: Wiley.

Johnson, L. W. (1986). *Intention-based diagnosis of novice programming errors.* London: Pitman.

Keane, M. T. G. (1990). Incremental analogizing: theory and model. In K. J. Gilhooly, M. T. G. Keane, R. H. Logie, & G. Erdos (Eds.), *Lines of thinking* (Vol. 1, pp. 221-236). Chichester, England: Wiley.

Köhne, A., and Weber, G. (1987). STRUEDI: a LISP-structure editor for novice programmers. In H. J. Bullinger, & B. Schackel (Eds.), *Human-Computer Interaction INTERACT '87* (pp. 125-129). Amsterdam: North-Holland.

McClelland, J. L., and Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception: Part 1. An account of basic findings. *Psychological Review* 88:375-407.

Miller, G. A., Fellbaum, C., Kegl, J., and Miller, K. (1988). WORDNET: An electronic lexical reference system based on theories of lexical memory. *Revue Québécoise Linguistique* 17:181-213.

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

Ratterman, M., and Gentner, D. (1987). Analogy and similarity: determinants of accessibility and inferential soundness. In Proceedings of the Ninth Annual Conference of the Cognitive Science Society, 23-35. Hillsdale, NJ: Erlbaum.

Riesbeck, C. K., and Schank, R. C. (1989). *Inside case-based reasoning.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Ross, B. H. (1987). This is like that: the use of earlier problems and the separation of similarity effects. *Journal of Experimental Psychology: Learning, Memory, and Instruction* 13:629-639.

Thagard, P., and Holyoak, K. J. (1989). Why indexing is the wrong way to think about analog retrieval. In K. J. Hammond (Ed.), *Proceedings of the Second Workshop on Case-Based Reasoning* (pp. 36-40).

Thagard, P., Holyoak, K. J., Nelson, S., and Gochfeld, D. (1990). *Analog retrieval by constraint satisfaction* (CSL-Report No. 41). Princeton University.

Weber, G. (1988). Cognitive diagnosis and episodic modelling in an intelligent LISP-tutor. In Proceedings of Intelligent Tutoring Systems ITS-88, 207-214. Montreal, June 1-3.

Weber, G. (1989). Automatische kognitive Diagnose in einem Programmier-Tutor. In D. Metzing (Ed.), *Künstliche Intelligenz GWAI-89* (pp. 331-336). Berlin: Springer.