Detecting Discontinuities in Case-Bases

Hideo Shimazu and Yosuke Takashima
Information Technology Research Laboratories
NEC Corporation
4-1-1 Miyazaki, Miyamae, Kawasaki, 216 Japan
{shimazu, yosuke}@joke.cl.nec.co.jp

Abstract

This paper describes a discontinuity detection method for case-bases and data bases. A discontinuous case or data record is defined as a case or data record whose specific attribute values are very different from those of other records retrieved with identical or similar input specifications. Using the proposed method, when a user gives an input specification, he/she can retrieve not only exactly-matched cases, but also similar cases and discontinuous cases. The proposed method has three steps: (1) Retrieving case records with input specifications which are the same as or similar to a user's input specification (*Maybe Similar Case, MSC*), (2) Selecting a case record which most closely matches the user's input specification among MSCs (*Base Case, BC*), and (3) Detecting cases among MSCs whose output specifications are very different from those of BC. The proposed method has been implemented in the CARET case-based retrieval tool operating on commercial RDBMS. Because case-based reasoning systems rely on the underlying assumption that *similar input specifications retrieve similar case records*, discontinuity detection in case-bases is indispensable, and our proposed method is especially useful.

Introduction

This paper analyzes discontinuities in case-bases and the architecture of automatic discontinuity detection in case-bases and data bases. Specifically, we implement a discontinuity detection mechanism on an enhanced version of CARET, a case-based retrieval tool that operates on a relational data base management system (RDBMS). When a user gives an input specification, CARET automatically retrieves not only exactly-matched cases, but also similar cases and discontinuous cases.

Discontinuities in a case-base or a data base exist where exceptional records exist in a cluster. Some (one to all) of the attribute values of such exceptional records are very different from those of the majority of records retrieved with identical or similar input specifications. Figure 1 shows a simple and common example

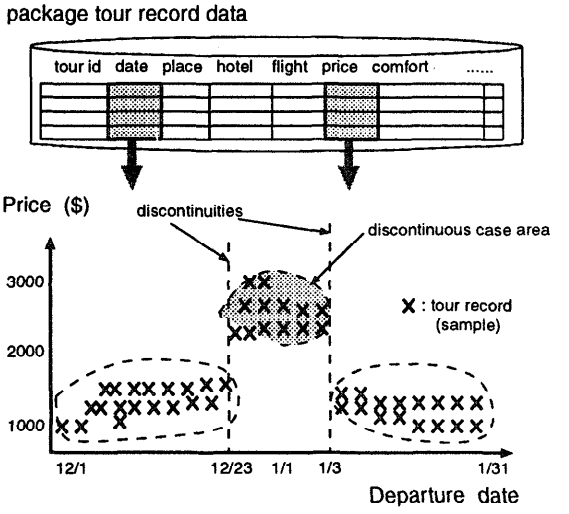

Figure 1: Discontinuity in package tour data records. Scattergram of departure date versus price

of discontinuities in a data base. The data base contains tour data records for a set of package tours from Japan to Hawaii in December and January. Here we plot the departure date of a tour versus the tour price in the record. The ordinary tour price is generally about \$1500. However, since package tours departing between December 23 and January 3 are \$1000 more expensive than ordinary ones, they are exceptional records from the viewpoint of price. There are discontinuities from Dec 23 to Jan 3. Such exceptional records often contain useful information. For example, a user planning to depart on Dec 23 would appreciate hearing that he can save money if he departs on Dec 22.

The motivation behind this research is that detecting discontinuities in a case-base is necessary for installing case-based reasoning (CBR) systems in actual applications. CBR is a problem solving method which retrieves similar precedent cases and adapts old solutions in them to solve new problems (Schank 1982;

Kolodner 1993). After many failures in expert system development projects, CBR has been widely used in actual applications, because cases are much easier than rules to formulate and maintain. CBR relies on the underlying assumption, *similar input specifications retrieve similar case records.* However, if there are discontinuous cases in a case-base, a CBR system may unfortunately retrieve such discontinuous cases as precedent cases and generate incorrect solutions by adapting them. Therefore, discontinuities in case-bases should be detected in CBR systems.

For example, a previously reported help desk system (Shimazu, Shibata and Nihei 1994) is a case-based retrieval system which stores previous customer inquiries. While the help desk operator answers a user's inquiry, it shows the operator previous cases similar and related to the inquiry. The following are two discontinuous cases found in the case-base of the help desk system:

Discontinuous case 1: Only a machine type produced during a specific period of time has a very high percentage of trouble claims regarding its switching equipment. Other than that, the machine type is the same as those produced in different time periods.

Discontinuous case 2: Only a specific version of a software product does not run with a specific peripheral I/O board. Other than that, the version is the same as other versions.

When a help desk operator retrieves previous cases while answering a customer's inquiry regarding these symptoms, these cases should not be retrieved if the produced time period is different (Discontinuous case 1) or the software version is different (Discontinuous case 2).

Cases are defined and stored by help desk operators after answering each customer's inquiry. Because they are stored in a case-base without special indices, discontinuities are naturally generated in the case-base. However, no CBR study yet appeared on discontinuity detection method in a case-base. Even commercial CBR tools, such as CBR Express (Inference 1993) and ReMind do not incorporate them. Thus, we had to develop our own mechanism, optimized for our CBR systems.

Design Decision

Analysis of Discontinuities

There are three factors which affect the existence of discontinuities in a case-base.

1. **Observed attributes in cases:**
   Whether a case is discontinuous or not depends on observed attributes in the case. Figure 2 shows two scattergrams generated from the same data base as in Figure 1. The new scattergram shows hotel names versus customers' subjective evaluations of the degree of comfort. Depending on observed attributes in case records, an identical record (specified in the scattergrams) becomes a discontinuous case in a scattergram, while it is a normal case in another scattergram. In general, the number of observed attributes is more than one, and each attribute is given a different weight of importance.

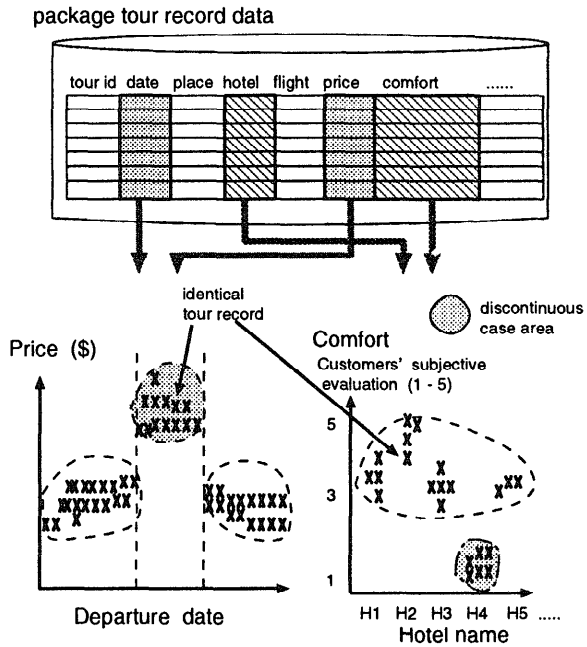

Figure 2: Two scattergrams generated from identical package tour data records

2. **Compared neighboring cases:**
   Whether a case is discontinuous or not depends on the difference from compared neighboring cases. For a person who can depart at any time in December, package tours departing after Dec. 23 seem exceptionally expensive. However, for a person who must depart on Jan. 1, since any package tours departing around Jan. 1 are as expensive as that on Jan. 1, he does not think a tour departing on Jan. 1 is exceptionally expensive.

3. **Hidden attributes:**
   Discontinuities often exist because of hidden attributes which are calculated from a set of existing attribute values in cases. For a person who travels alone, a $1,000 difference in the price attribute may not be so important. However, for a person who takes his family (for example, 5 persons), a $5,000 ($1,000 × 5) difference is a big issue. Here, a hidden attribute, *total price* is a salient attribute when detecting discontinuous cases.

Based on the analysis, a discontinuous case or data record can be defined as a case or data record whose output specifications are different from those of other records retrieved with the same or similar input specifications. Here, the output specifications are defined as

a set of weighted specific attributes including hidden attributes, and the input specifications are another set of weighted attributes.

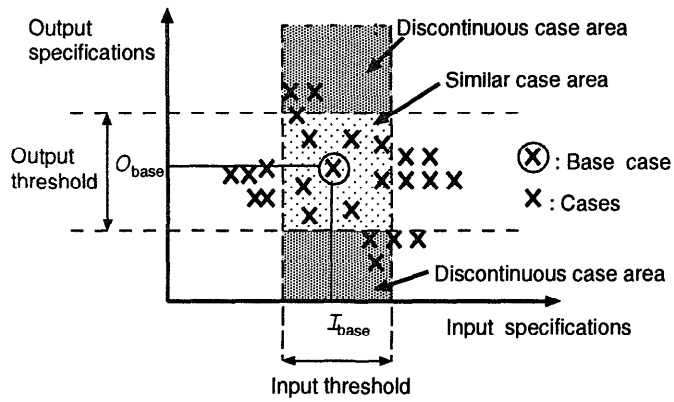

Figure 3: General scattergram visualizing discontinuous case area

Three Steps to Detect Discontinuities

Based on the above definition of discontinuities, there are three steps in detecting discontinuities.

Step 1: Retrieving cases with input specifications which are the same as or similar to a user's input specification (*Maybe Similar Case, MSC*)

Step 2: Selecting a case record which most closely matches the user's input specification among MSCs (*Base Case, BC*)

Step 3: Detecting discontinuous cases among MSCs whose output specifications are very different from that of BC

Figure 3 is a general scattergram indicating input specifications (X axis) versus output specifications (Y axis) of case records in a case-base. A user's input specification is indicated as I_{base}.

MSCs are retrieved with input specifications around I_{base} within the pre-defined *Input threshold*. Based on the assumption that *similar input specifications retrieve similar case records*, all MSCs must be similar. However, there is a possibility that discontinuous cases are included in the MSCs. Because the retrieved cases are either the *really* similar cases or discontinuous cases, they are located in either *similar case area* or *discontinuous case area* in Figure 3.

BC is a case which most closely matches the user's input specification among MSCs. In Figure 3, the circled point indicates BC. BC's output specification is indicated as O_{base}.

The output specification of each MSC is compared with that of O_{base}. If the difference is smaller than the pre-defined *Output threshold*, the case is regarded as a similar case (located in the *similar case area*). Otherwise, the case is regarded as a discontinuous case (located in the *discontinuous case area*).

Case Retrieval using the Nearest Neighbor

CARET is a case-based retrieval tool that operates on a commercial relational data base management system (RDBMS) (Shimazu, Kitano and Shibata 1993). It carries out similar-case retrieval using the nearest neighbor retrieval method. In the nearest neighbor retrieval method, similarity between a user's input query (Q) and a case (C) in the case-base ($S(Q,C)$) is defined as the weighted sum of similarities for individual attributes:

$$S(Q,C) = \frac{\sum_{i=1}^{n} W_i \times s(Q_i, C_i)}{\sum_{i=1}^{n} W_i} \quad (1)$$

where W_i is the i-th attribute weight, and $s(Q_i, C_i)$ is the similarity between the i-th attribute value for a query (Q) and that for a case (C) in the RDB.

Traditional implementations compute the similarity value for all records, and sort records based on their similarity. However, this is a time consuming task, as computing time increases linearly with the number of records in the case-base (C : *Cardinality* of the data base) and with the number of defined attributes (D : *Degree* of the data base). This results in time-complexity of $O(C \times D)$. This implementation strategy for RDBMS would be a foolish decision, as individual records must be retrieved to compute similarity. Thus, the total transaction number would be intolerable.

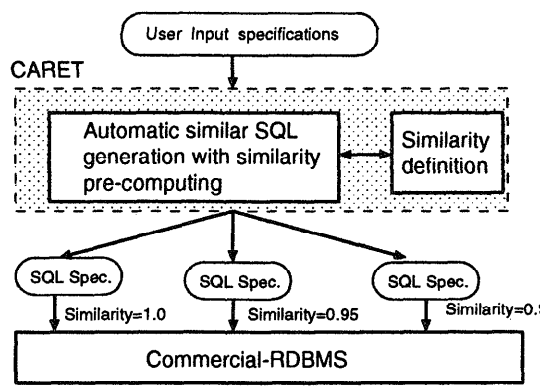

Figure 4: Automatic SQL generation

CARET carries out the nearest neighbor retrieval efficiently using the SQL data base language. It generates SQL specifications in varying degrees of similarity, and dispatches generated SQL specifications to RDBMS to retrieve cases from RDB. (Figure 4).

The CARET algorithm can be enhanced to retrieve not only similar cases but also discontinuous cases. The essence of this work, therefore, is to extend the CARET architecture to be able to retrieve not only

similar cases but also discontinuous cases. The following sections describe the architecture and report on its performance.

The Extended CARET System

Cases are represented as a flat record of n-ary relations, and stored in tables in commercial RDBMS. For each case attribute, its similarity measure and weight are defined. The similarities between values in individual attributes in case records are defined similarity by domain experts (Figure 5). In this example, the similarity between "C" and "C++" is 0.7. The input threshold and output threshold are also defined by domain experts.

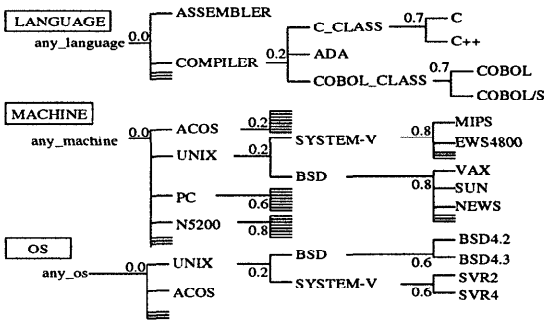

Figure 5: Attribute Hierarchy Example

Step 1: Retrieving MSCs

Step 1-(1): Creating Neighbor Value Sets A user gives attributes and their values as input specifications. For each user specified attribute, CARET refers to a similarity definition (Figure 5), to generate a set of values neighboring the user specified value. For example, if the user specifies "C++" as a value for the **Language** attribute, "C++" is an element in *the first-order neighbor value set (1st-NVS)*. "C" is an element in *the second-order neighbor value set (2nd-NVS)*. "ADA, COBOL and COBOL/S" are elements in *the third-order neighbor value set (3rd-NVS)*.

Step 1-(2): Enumerating Combinations of NVSs Next, all possible neighbor value combinations are created from the n-th order neighbor value sets. Figure 6 illustrates how such combinations are created. This example assumes that the user described "C++" as the value for the **Language** attribute and "BSD4.2" for the **OS** attribute as input specifications. All value combinations under attribute **Language** and **OS** are created. In this example, 9 combinations (3×3) are generated. Each combination of NVSs becomes the seed for SQL specifications.

Step 1-(3): Calculating Similarity Values for Each Combination For each combination, a similarity value is calculated using the similarity between the user specified values and those in each combination

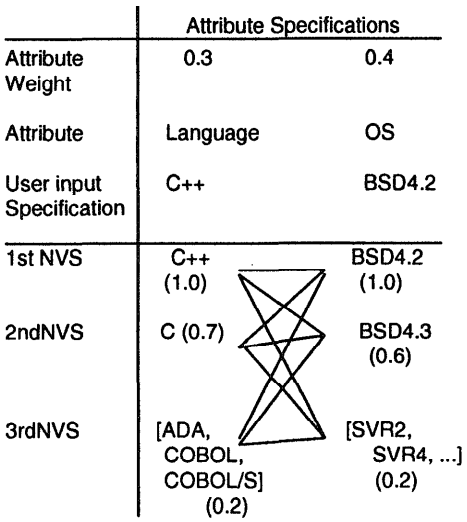

Figure 6: An Example showing Possible NVS combinations

created in the previous step. The calculation is similar to that for the weighted nearest neighbor, except that not all attributes are involved. Whether an attribute is part of the input attribute or not is shown in a mask matrix (M), which is a one-dimensional matrix whose size is equivalent to the case-base degree. The matrix element M_i will be 1, if the attribute i is part of the input specifications. Otherwise, M_i will be 0. The similarity between a input query(Q) and a NVS combination (F) is $S(Q, F)$, and is calculated as

$$S(Q,F) = \frac{\sum_{i=1}^{n} M_i \times W_i \times s(Q_i, F_i)}{\sum_{i=1}^{n} M_i \times W_i} \quad (2)$$

where F is an NVS combination, F_i is the i-th attribute for the combination, and W_i is the i-th attribute weight. For example, the similarity between a combination, (["C"],["BSD4.3"]) and the user's input specifications is 0.64 by the following calculation:

$$S(Q,F) = \frac{0.3 \times 0.7 + 0.4 \times 0.6}{0.3 + 0.4} = 0.64 \quad (3)$$

Step 1-(4): Generating SQL Specifications Only combinations whose similarity value is higher than the pre-defined input threshold are selected, and each selected combination is translated into a corresponding SQL specification. The only SQL expression type used here is the SELECT-FROM-WHERE type. For example, the NVS combination example shown above is translated into the following SQL expression:

```
SELECT * FROM case_table
WHERE language = 'C'  and OS = 'BSD4.3';
```

Then, CARET dispatches the SQL specifications from the highest similarity score. Since SQL does not

involve a similarity measure, the similarity values precomputed in this process are stored in CARET, and are referred to when the corresponding query results are returned.

Step 2: Selecting a Base Case

Because CARET dispatches SQL specifications from the highest similarity score, the first returned case becomes a *Base case*, because it matches the user's input specification exactly or most closely. [1] The output specification of the base case, O_{base} is extracted from the base case. All other retrieved cases are stored with their similarity values in CARET as potentially similar cases.

Step 3: Detecting Discontinuous Cases among MSCs

CARET calculates the similarity between the output specification of each retrieved case and that of the base case, O_{base}. Because the total number of the retrieved cases is much smaller than the total number of cases in the case-base, traditional implementation for the weighted nearest neighbor is carried out and is sufficiently effective here.

The calculation method is the same as that for traditional implementation, except that not all attributes are involved. Whether an attribute is part of the output specification or not is shown in a mask matrix (N), which is a one-dimensional matrix whose size is equivalent to the case-base degree. The matrix element N_i will be 1, if the attribute i is part of the output specification. Otherwise, N_i will be 0. The similarity between the output specification of a base case (B) and that of a retrieved case (C) is $S(B,C)$ and is calculated as

$$S(B,C) = \frac{\sum_{i=1}^{n} N_i \times W_i \times s(B_i, C_i)}{\sum_{i=1}^{N} N_i \times W_i} \quad (4)$$

where B_i is the i-th attribute for the base case, C_i is the i-th attribute for the case (C), and W_i is the i-th attribute weight.

For each retrieved case, its similarity value is compared with the output threshold. If the similarity value is higher than the output threshold, it is a similar case. Otherwise, it is regarded as a discontinuous case.

For example, suppose that an output attribute is `Work load (Man-Month)`, its O_{base} value is 10 Man-Month Work Load (MM), its similarity definition is as shown in Table 1, and the output threshold is 0.5. If there are cases among MSCs whose `Work-load` attribute value is less than 1 MM or more than 36 MM, they are regarded as discontinuous cases.

[1] If several base case candidates exist, there are selection methods, such as calculating average cases and asking the user to choose one.

Work Load	0-1	1-6	6-12	12-36	36-
0-1	1.0	0.7	0	0	0
1-6	-	1.0	0.7	0.2	0
6-12	-	-	1.0	0.7	0
12-36	-	-	-	1.0	0.7
36-	-	-	-	-	1.0

Table 1: Similarity Definition of Work-load (Man-Month)

Factors	Query-1	Query-2
Query length	2	3
Tree depth	6	8
Tree width	28	25
Generated SQL number	6	4
Retrieved cases	105	3
Similar cases	102	3
Discontinuous cases	3	0
Input threshold	0.8	0.8
Output threshold	0.8	0.8

Table 2: Query Characteristics

Empirical Results

The extended CARET performance has attained an acceptable speed. The experiment was evaluated by using a SUN Sparc Station 2, and by using ORACLE version 6 installed on SunOS version 4.1.2. Figure 7 shows the response times measured for typical user queries. The algorithm scalability was tested by increasing the number of cases in the case-base. The number of cases was increased to 1,500. The average response time for a query was about 2.0 seconds. The queries in Figure 7 are :

Query-1: `(LANGUAGE = C++) and (MACHINE = SUN)`

Query-2: `(TROUBLE-TIME = SYSTEM-GENERATION-TIME)` and `(TROUBLE-TYPE = VERSION-MISMATCHED)` and `(CHOSEN-METHOD = CHANGE-LIBRARY-VERSION)`

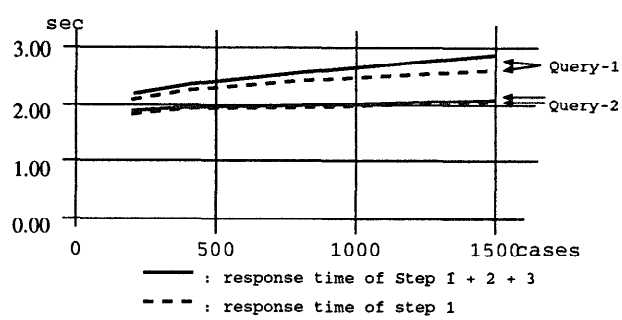

Figure 7: Case-base retrieval and discontinuity detection time

Characteristics for each query are shown in Table 2. Query length refers to the number of conjunctive

clauses in the **WHERE** clause. Tree depth shows the abstraction hierarchy depth for each attribute. Tree width is the number of terminal nodes of the abstraction hierarchy of each attribute. The generated SQL number is the number of SQL specifications generated and actually sent to RDBMS. Retrieved cases shows the number of cases retrieved by each query. Similar cases shows the number of similar cases retrieved. Discontinuous cases shows the number of discontinuous cases retrieved. These numbers are measured with a case-base containing 1500 cases. The input threshold and output threshold are pre-defined as fixed numbers. The speed of retrieving MSCs is affected by the query length, tree depth, tree width, generated SQL number, and retrieved cases. The speed of detecting discontinuous cases among MSCs is affected by the input threshold.

Based on a field test in our help desk operation, the capability of showing both similar cases and discontinuous cases was welcomed by help desk operators. They used the information to ignore discontinuous cases as useless cases. However, we found two major problems. First, discontinuous cases should indicate more information. For example, if many individual cases of the same type of problem inquiries are retrieved as discontinuous cases, the CBR system should generalize them and warn about the potential existence of a common problem not recognized yet by domain experts. Combining the proposed method with statistical approaches will be a subject for further research. Second, it is difficult to select appropriate attributes in cases for detecting exceptional cases. As we showed in this paper, whether a case is discontinuous or not depends on the observed attributes in the case. In the package tour example, if we had not observed the relation between the departure date attribute and the price attribute, we would not have known about the existence of such discontinuities in the data base. Similarly, if we had not noticed the production time period of a machine or the version of a software product, we would not have recognized exceptions like the cases in the help desk systems described above. Monitoring all combinations of attributes in a case-base is inefficient and wasteful because there are too many combinations. Therefore, determining which attributes should be monitored to detect important discontinuities is another subject for further research.

Related Work

Among intelligent data base researchers, Siklossy (Siklossy 1977) indicated the existence of discontinuities in data bases. Parsaye (Parsaye 1993) proposed a discontinuity detection mechanism using constraint rules, like "if Department = "Sales" then Salary > 30,000". System designers pre-define rules which check for the existence of exceptions likely to occur in the future. This mechanism detects and prevents many errors during data entry. However, it can not deal with discontinuities which have not been yet recognized by system designers, such as the discontinuities described in this paper.

Conclusion

This paper described a discontinuity detection method in case-bases and data bases. A discontinuous case or data record is defined as a case or data record whose specific attribute values are very different from those of other records retrieved with identical or similar input specifications.

Using the proposed method, when a user gives an input specification, he/she can retrieve not only exactly-matched cases, but also similar cases and discontinuous cases.

The propose method has three steps: (1) Retrieving case records with input specifications which are the same as or similar to a user's input specification (*Maybe Similar Case, MSC*), (2) Selecting a case record which most closely matches the user's input specification among MSCs (*Base Case, BC*), and (3) Detecting cases among MSCs whose output specifications are very different from that of BC.

The proposed method has been implemented in the CARET case-based retrieval tool operating on commercial RDBMS. Because case-based reasoning systems rely on the underlying assumption that *similar input specifications retrieve similar case records*, discontinuity detection in case-bases is indispensable, and our proposed method is especially useful for this. However, because discontinuous data records often contain useful information, discontinuity detection is also useful for ordinary data base applications as long as the above assumption is applicable to the data bases.

References

Inference Corporation, 1993. CBR Express and Case-Point: Product Introduction, Presentation slides for NDS Customers, Tokyo.

Kolodner, J., 1993. Case-Based Reasoning, Morgan Kaufmann.

Parsaye, K, 1993 Intelligent database tools and applications John Willey & Sons, Inc.

Schank, R. C., 1982. Dynamic memory: A theory of learning in computers and people, Cambridge Univ. Press.

Shimazu, H., Kitano, H., and Shibata, A., 1993. Retrieving Cases from Relational DataBase: Another Stride Towards Corporate-Wide Case-Based Systems, In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-93)

Shimazu, H., Shibata, A., and Nihei, K., 1994. Case-Based Retrieval Interface Adapted to Customer-Initiated Dialogues in Help Desk Operations, In Proceedings of the National Conference on Artificial Intelligence (AAAI-94)

Siklossy, L, Question-Asking Question-Answering, Report TR-71, Austin: University of Texas, Computer Science.