A Fuzzy Approach to Case-Based Reasoning through Fuzzy Extension of SQL

Luigi Portinale, Stefania Montani
Dipartimento di Informatica
Università del Piemonte Orientale "Amedeo Avogadro"
Alessandria (Italy)

Riccardo Bellazzi
Dipartimento di Informatica e Sistemistica
Università di Pavia
Pavia (Italy)

TR-INF-2002-07-03-UNIPMN

Keywords: fuzzy retrieval, fuzzy SQL

Corresponding Author:
Luigi Portinale
Dipartimento di Informatica
Universita' del Piemonte Orientale "Amedeo Avogadro"
Spalto Marengo 33 - 15100 Alessandria (ITALY)
tel: +39 0131 287 451 fax: +39 0131 287 440
e-mail: portinal@unipmn.it

Abstract

The use of database technologies for implementing CBR techniques is attracting a lot of attention for several reasons. First, the possibility of using standard DBMS for storing and representing cases significantly reduces the effort needed to develop a CBR system; in fact, data of interest are usually already stored into relational databases and used for different purposes as well. Finally, the use of standard query languages, like SQL, may facilitate the introduction of a case-based system into the real-world, by putting retrieval on the same ground of normal database queries. Unfortunately, SQL is not able to deal with queries like those needed in a CBR system, so different approaches have been tried, in order to build retrieval engines able to exploit, at the lower level, standard SQL. In this paper, we concentrate on Fuzzy Case-Based Reasoning where case similarity is defined by means of fuzzy predicates, operators and standard fuzzy logic connectives, rather than through distance measures as in usual k-NN approaches. We present a proposal where case retrieval is implemented by using a straightforward fuzzy extension to standard SQL, where the boolean satisfiability condition for tuple selection is substituted with a fuzzy one. A case-based client/server architecture exploiting Fuzzy-SQL as a retrieval engine is then presented, together with some possible applications in e-commerce and medical domains.

1 Introduction

The construction of a case-based system involves a set of different phases, each one related to the different steps composing the CBR cycle [1]. In particular, in order to implement a particular case retrieval algorithm, suitable case structuring and case base organization have to be devised. Very often, the case memory requires specific data structures to be implemented like decision trees, k-d trees, E-MOPs, associative memories, etc... [10, 18]; this may be a further burden in the construction of a CBR system, especially if data concerning cases are already available in standard databases.

The use of database techniques for supporting the construction of case-based systems is recently attracting serious attention; the reasons are twofold:

1. if data of interest are already stored in a database, the database itself can be used as a case base;

2. part of the technological facilities of a DBMS may be exploited in the CBR cycle, in particular for case retrieval.

This is particularly true in e-commerce applications, where the *product database* represents the main data source, which records are easily interpretable as "cases" for a case-based recommendation system [14, 15, 20], as well as in the broad context of knowledge management, for the implementation of corporate-wide case-based systems [9, 16]. All these database driven proposals focus on the retrieval step of the CBR cycle[1], since similarity-based retrieval is the fundamental step that allows one to start with a set of relevant cases (e.g. the most relevant products in e-commerce), in order to apply any needed revision and/or refinement.

Case retrieval algorithms usually focus on implementing Nearest-Neighbor (NN) techniques, where local similarity metrics relative to individual features are combined in a weighted way to get a global similarity between a retrieved and a target case. A k-NN case retrieval algorithm will then return the k most similar cases to the target one. In [5], it is argued that the notion of *acceptance* may represent the needs of a flexible case retrieval methodology better than distance (or similarity). As for distance, local acceptance functions can be combined into global acceptance functions to determine whether a target case is acceptable (i.e. it is retrieved) with respect to a given query. In particular, very often local acceptance functions take the form of fuzzy distributions; in this way, the definition of a fuzzy linguistic term over the domain of a given attribute of a case can be exploited to characterize the acceptance of cases having similar (in the fuzzy sense) values for that particular attribute or feature.

The main goals and objectives of our work can be then summarized as follows:

- defining a case retrieval approach suitable for query cases expressible (either directly or indirectly) with fuzzy concepts;

- exploiting in a sound way fuzzy logic theory;

- exploiting as much as possible standard data representations for implementing cases (i.e. relational databases);

- exploiting as much as possible standard query tools and languages (like SQL or its extensions) for implementing retrieval.

[1]Actually, in [20] a revision of the classical CBR cycle presented in [1] is proposed, by specifically taking into account the e-commerce application; however no particular database technique is specifically proposed for implementing this new cycle.

In the present paper, we will present an approach where local acceptance relative to a feature can be actually expressed through fuzzy distributions on its domain, abstracting the actual values to linguistic terms. The peculiarity of the approach is that global acceptance is completely defined in fuzzy terms, by means of the usual combinations of local distributions through specific defined *norms* [11]. Moreover, an extended version of SQL, able to deal with fuzzy predicates and conditions, is introduced as a suitable way to directly query a case base stored on a relational DBMS; this approach is based on the **SQLf** language proposed in [3], extending standard SQL in order to deal with *fuzzy queries*.

The paper is organized as follows: section 2 compares the different approaches to retrieval, namely similarity-based and fuzzy-based; section 3 summarizes the standard approach to fuzzy querying on top of a relational DBMS by also showing how a fuzzy query with a suitable acceptability threshold can be translated in a boolean one; section 4 shows how the fuzzy querying framework can actually be exploited to implement fuzzy case retrieval and a client/server architecture able to generate a Fuzzy-SQL query from a case specification is presented in section 5; finally, in section 6 two different applications exploiting the presented approach are presented in e-commerce (travel suggestion) and medical domains (diabetic patient monitoring) respectively. Comparison with related works and some conclusions are then reported in section 7.

2 Similarity-Based Retrieval versus Fuzzy-Based retrieval

Standard case retrieval approaches are based on distance measures defined on case features (local distance) that are then combined in a suitable way, in order to get the actual distance between a query or target case and a retrieved case (global distance). The Nearest-Neighbor approach to case retrieval is actual based on such a use of distance functions and several retrieval algorithms proposed in the literature belong to this category. Case similarity is easily defined as a kind of *inverse* function of case distance (greater is the distance, smaller is the similarity).

However, similarity-based retrieval grounded on distance metrics is not the only way of considering retrieval; in fact, an often cited proposition is that *the main difference between database and case-based systems stands in the fact that database systems are only able to retrieve information that fulfills particular conditions in a precise way, while imprecise (fuzzy) matching is possible within CBR system.* This means that we can view retrieval as a way of accepting a set of cases, by means of some fuzzy measure of acceptance.

Despite their relative differences, fuzzy retrieval (based on acceptance) and nearest-neighbor retrieval (based on distance) have some common grounds. In [5], a correspondence between acceptability and distance has been pointed out; the way of considering the acceptability induced by a distance measure is through the so-called *characteristics similarity curves.* As an example (adapted from [5]) consider a local distance metric x_i on a given feature f_i. A very simple similarity metric induced by x_i can be $s(x_i) = 1 - |x_i|$ (see fig. 1.(a)). Let us consider now the *sum* (+) combination function for the global distance; fig. 1.(b) shows the characteristics similarity curves for a given value a and for a query case composed by two features q_1, q_2. They represent the acceptability regions induced by the global distance measure; every case (f_1, f_2) inside a particular region (rhombus) has an acceptability greater or equal to a with respect to the query case (q_1, q_2).

Acceptability functions can then be used instead of distance or similarity functions and local acceptability can be combined into global acceptability functions to determine whether the case can be retrieved, that is whether it is acceptable with respect to the query case and a particular acceptability threshold (see [5] for more details).

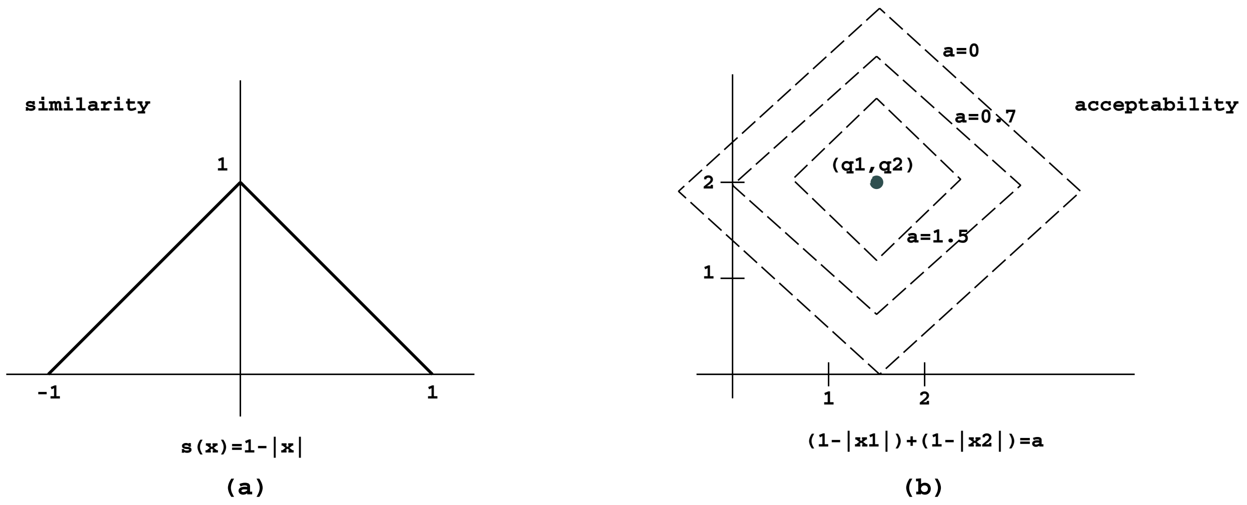


Figure 1: Characteristics similarity curves (adapted from [5]).

Fuzzy membership of a feature value with respect to a particular fuzzy set can then be viewed as local acceptance functions. Consider for example the feature *price* of a given product and consider the fuzzy term *average* price defined by the fuzzy distribution of fig. 2. As in the case of characteristics similarity curves, different acceptability thresholds give rise to different acceptability regions; for instance, retrieving cases concerning products having an average price with acceptability greater than 0.75 will result in retrieving cases concerning products with a price between \$300 and \$700. Notice that, in case of fuzzy retrieval, the main reference is a linguistic value (i.e. a fuzzy set), while in similarity-based retrieval the reference is a feature value, not defined on a fuzzy set. This essentially means that fuzzy retrieval is more general in use, by allowing the fuzzification of the queries, i.e. query cases can be expressed in terms of both crisp and fuzzy values. We will return on this aspect in section 4.

Finally, if fuzzy membership functions can model local acceptability, global acceptability is easily obtained by using fuzzy combination through suitable t-norms o t-conorms [11]; this means that the standard framework of fuzzy logic provides a sound way of aggregating local information (at the feature level) into a global measure (at the case level). Given the fact that fuzzy logic matching can be formally used to retrieve a set of cases given a particular query, the problem to solve is how to implement this kind of fuzzy retrieval. Our aim is to exploit as much as possible standard data representation and standard query tools. For this reason, we investigated the possibility of exploiting fuzzy extension to SQL for imprecise querying in relational databases. Next section will discuss how a relational database can be actually queried in a fuzzy way.

3 Fuzzy Querying in Databases

It is well-known that standard relational databases can only deal with precise information and standard query languages, like SQL, only support boolean queries. As already mentioned, this is what make CBR different from usual database technology where, if a specific condition is specified during information retrieval, only data exactly satisfying such a condition can be obtained, by loosing the possibility of

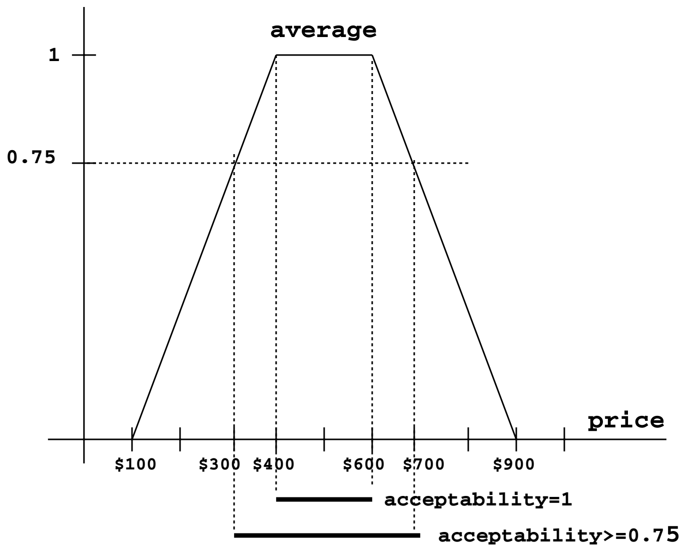


Figure 2: Fuzzy membership as local acceptability.

retrieving those items that, while not fulfilling the condition, exhibit only small differences with respect to those fulfilling it. Fuzzy logic provides a natural way of generalizing the strict satisfiability of boolean logic to a notion of satisfiability with different degrees; this is the reason why considerable efforts has been dedicated inside the database community toward the possibility of dealing with fuzzy information in a database.

There are at least two main approaches to the problem:

1. explicit representation of vague and imprecise information inside the database [4, 12];
2. vague and imprecise querying on a regular database [3, 8].

Even if the first class of approaches is more general that the second one, we will concentrate on the latter, since the interest in this paper is to show how flexible querying in a standard database can be used to implement case retrieval. In particular, in [3] standard SQL is adopted as the starting point for a set of extensions able to improve query capabilities from boolean to fuzzy ones. While the long term goal of the approach is to integrate such extensions directly at the DBMS level (directly providing the user with specific DBMS algorithms for flexible query processing), in the short term the implementation of the SQL extensions can be actually provided on top of a standard relational DBMS, by means of a suitable module able to transform a fuzzy query into a regular one through the so called *derivation principle* [2].

In the following, we concentrate on this methodology, by defining a set of fuzzy extensions to SQL that may be of interest for CBR and by showing how they can be processed, in order to transform them in standard queries; we will finally show how this can be actually implemented for case retrieval purposes.

The basic block of an SQL statement follows the form:

SELECT *field_list*

```
FROM table_list
WHERE condition
```

We are interested in simple SQL statements with no nesting (i.e. we consider the WHERE clause to be a reference to an actual condition and not to nested SQL statements); in our Fuzzy-SQL language, the condition can be a composite fuzzy formula involving both crisp and fuzzy predicates and operators.

3.1 Fuzzy Predicates

A set of linguistic values can be defined over the domains of the attributes of interest; three different types of fuzzy predicates are considered:

- *fuzzy predicates on continuous domain*: a set of linguistic values defined on a continuous domain and through a continuous distribution (e.g. a trapezoidal, triangular or a Gaussian-like distribution);
- *fuzzy predicates on discrete domains*: a set of linguistic values defined on a discrete domain, but with two different possible types of distribution: *continuous* for *scalar discrete* attributes (i.e. with ordered values) (e.g. a trapezoidal distribution for the linguistic value `young` defined over the discrete domain of integers of the attribute `age`) or *discrete* for *nominal* attributes (with unordered values) (e.g. a vector distribution for the linguistic term `bright` defined over the attribute `color`, associating to each color the membership function to the fuzzy set).

Furthermore, a set of fuzzy operators can be defined to relate fuzzy or crisp predicates; also in this case we have different types of operators:

- *continuous operators*: characterized by a continuous distribution function (e.g. the operator `near` over scalar attributes, characterized by a trapezoidal, triangular or Gaussian-like curve centered at 0 and working on the difference of the operands);
- *discrete operators*: defining a *similarity relation* characterized by a symmetric matrix of similarity degrees (e.g. the operator `compatible` over the attribute `job` defining to what degree a pair of different jobs are compatible).

By allowing fuzzy predicates and operators to form the condition of the `WHERE` clause, the result of the `SELECT` is actually a fuzzy relation, i.e. a set of tuples with associated the degree to which they satisfy the `WHERE` clause. Such a degree can be characterized as follows: let

`SELECT` A `FROM` R `WHERE` fc

be a query with fuzzy condition fc; the result will be a fuzzy relation Rf with membership function

$$\mu_{Rf}(a) = \sup_{(x \in R) \wedge (x.A = a)} \mu_{fc}(x)$$

The fuzzy distribution $\mu_{fc}(x)$ relative to fc must then be computed by taking into account the logical connectives involved and their fuzzy interpretation. It is well known that the general way to give a fuzzy interpretation to logical connectives is to associate negation with *complement to one*, conjunction with a suitable *t-norm* and disjunction with the corresponding *t-conorm* [11]. In the following, we will concentrate on the simplest t-norm and t-conorm, namely the min and max operators such that

$$\mu_{A \wedge B}(x) = \min(\mu_A(x), \mu_B(x))$$
$$\mu_{A \vee B}(x) = \max(\mu_A(x), \mu_B(x))$$

We will discuss the implementative advantages of this choice and what problems have to be addressed with other norms.

3.2 Deriving Standard SQL from Fuzzy-SQL

In order to process a query using a standard DBMS, we have to devise a way of translating the Fuzzy-SQL statement into a standard one. We have noticed that the result of a fuzzy query is always a fuzzy relation, while the result of a standard query is a boolean relation; this means that, if we are able to translate a fuzzy query into a standard one, the resulting boolean relation has to satisfy certain criteria related to the original fuzzy query. The most simple way is to require the fuzzy query to return a boolean relation R_b whose tuples are extracted from the fuzzy relation R_f, by considering a suitable threshold on the fuzzy distribution of R_f. We consider, as in [3], the following syntax

`SELECT` (λ) A `FROM` R `WHERE` fc

whose meaning is that a set of tuples with attribute set A, from relation set R, satisfying the condition fc with degree $\mu \geq \lambda$ is returned (in fuzzy terms, the λ-cut of the fuzzy relation R_f resulting from the query is returned).

The idea is then to transform the fuzzy query into an SQL query returning a superset of the λ-cut of the result relation. The interesting point is that, if we restrict attention to the kind of queries previously discussed (which are suitable to model standard case retrieval) and if we adopt min and max operator as norms, then it is possible to derive from a Fuzzy-SQL query, an SQL query returning exactly the λ-cut required [2].

This can be easily verified as follows: let P be a fuzzy predicate; we write $P \geq \lambda$ to indicate that P is satisfied with degree greater or equal than λ. Let $DNC(P, \geq, \lambda)$ be the derived necessary condition for $P \geq \lambda$, i.e. a boolean condition such that $P \geq \lambda \rightarrow DNC(P, \geq, \lambda)$.
It is trivial to verify that

$AND(P_1, \ldots, P_n) \geq \lambda \leftrightarrow \min(P_1, \ldots, P_n) \geq \lambda \leftrightarrow DNC(P_1, \geq, \lambda) \ldots, DNC(P_n, \geq, \lambda)$

$OR(P_1, \ldots, P_n) \geq \lambda \leftrightarrow \max(P_1, \ldots, P_n) \geq \lambda \leftrightarrow DNC(P_1, \geq, \lambda) \ldots, DNC(P_n, \geq, \lambda)$

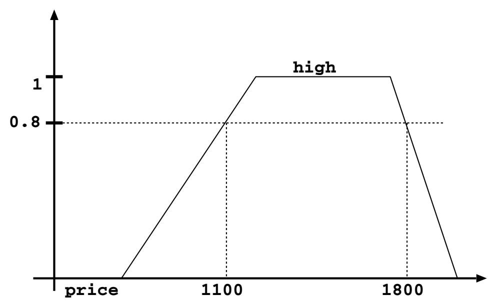
This implies that, using min and max operators, each derived necessary condition is also a sufficient one and thus the obtained boolean condition can be used to return exactly the required λ-cut[2]. Each DNC can then be obtained from the fuzzy distribution associated to the involved predicate.

Example. Consider a relation `product` containing the attribute `price` over which the linguistic term `high` is defined. Figure 3 shows the fuzzy distribution of `high` as well as the distribution of a fuzzy operator $\gg$ (much greater than). The condition $(\texttt{price} = \texttt{high} \wedge \texttt{price} \gg 1000) \geq 0.8$ will hold iff $\min(\texttt{price} = \texttt{high}, \texttt{price} \gg 1000) \geq 0.8$, iff $(\texttt{price} = \texttt{high}) \geq 0.8 \wedge (\texttt{price} \gg 1000) \geq 0.8$ iff $(1100 \leq \texttt{price} \leq 1800) \wedge (\texttt{price} - 1000) \geq 120$. The latter condition can be easily translated in a standard WHERE clause of SQL.

Unfortunately, there are situations (for instance when norms different that min, max are used) where the exact derivation of the λ-cut is not possible; in these cases a superset of the actual λ-cut will be returned. There are two possible solutions to that:

1. to implement an external filter receiving in input the result set of the query, producing in output the actual λ-cut, by filtering out those tuples having degree less than λ;

[2]Similar results hold for $(P, \leq, \lambda)$, to be used when negation is involved.



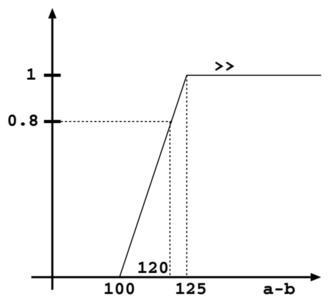


Figure 3: Fuzzy Distributions

2. if the DBMS support the inclusion of external functions in the query language, to use this facility to compute, during query processing, the actual degree of the tuple, by accepting only those belonging to the λ-cut.

In the next section, we will show how these strategies may be implemented in a system able to create a Fuzzy-SQL query from a target case, in order to retrieve a set of similar cases.

4 Implementing Fuzzy Case Retrieval

In order to make effective a fuzzy retrieval approach on top of a relational database we have defined the following framework for case retrieval implementation:

- cases are defined as collections of $\langle feature, value\rangle$ pairs and then represented as tuples of relations; it may be the case that that the relevant information for a case is scattered in different relations of a relational schema, however it is always possible to define a suitable view in order to reconstruct relevant case information in a single (possibly virtual) table or relation[3];
- as a consequence of the previous point, case features are represented by standard relational attributes;
- fuzzy terms and fuzzy operators are defined on the case base scheme (the database scheme of the case base) through a suitable meta-database containing all the fuzzy knowledge;

[3]Notice that a case is normally defined by a description, a solution and a possible outcome; the $\langle feature, value\rangle$ description is usually sufficient to represent all this part of a case.

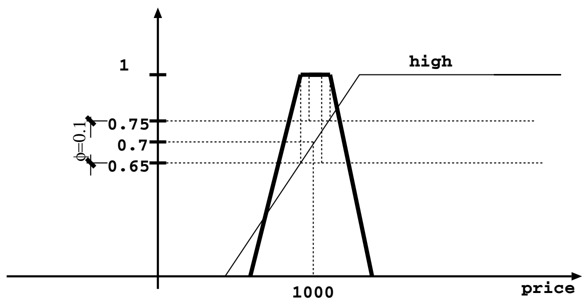


Figure 4: Fuzzification Example

- a target or query case is defined by specifying values for a set of features in a suitable way (see below);
- retrieval takes place by generating a Fuzzy-SQL query on the case base with threshold λ, returning the set of cases (tuples) within the λ limit of acceptability (i.e. the λ-cut of the resulting relation).

By considering the above framework, we notice that there is no fuzzy information represented in the case base. Cases are stored in the case base (the object database) by using, for each feature, only values from the corresponding attribute domain; on the other hand, query cases may also be specified by using linguistic (fuzzy) abstractions on such values. This means that, for each feature (attribute) to be specified in a query case, three different possibilities can be used:

1. specification of a linguistic value (if the corresponding domain allows for it);
2. specification of a crisp value, to be used *as is*;
3. specification of a crisp value *to be fuzzified.*

In the last case, fuzzification may take place in different ways; for instance, in case of a scalar attribute A whose input value is a, the fuzzy expression A `near` a can be used instead of $A = a$[4].

However, more sophisticated fuzzification strategies can also be defined, by constructing a fuzzy distribution from those defined over the domain of the attribute and from the input value. An example is provided in figure 4 where a fuzzification threshold $\phi = 0.1$ is specified on the fuzzy distributions for the linguistic term `high` of the attribute `price` with input value 1000; the result is the fuzzification of `price`= 1000 in terms of the fuzzy distribution shown as a bold line in figure 4; the slopes of the distribution can be determined in dependence on the original fuzzy distribution, by setting a *spread factor* similar to that used in the HCV algorithm [21]. Fuzzification may be more complex in case the input value belongs to more than one fuzzy set (i.e. if more linguistic terms are applicable to the input); in that case the fuzzification process must take into account a suitable norm to combine the results from each fuzzy set.

In addition to the specification of a crisp or linguistic value, the target case can also specify additional conditions involving specific (possibly fuzzy) operators (e.g. the user can specify that it requires cases

[4]A more general solution could be to provide specific operators of nearness $\mathtt{near}_{\mathtt{a}}$ for each individual scalar attribute A.

with `price=high` and `price` $\gg 1000$). Finally, a global retrieval (acceptability) threshold λ has to be specified. A Fuzzy-SQL query can then be generated as follows:

- one or more tables $(R_1, \ldots R_k)$ whose rows contain the cases of interest for retrieval are selected;
- a fuzzy condition fc is created by taking the AND of all the expressions (crisp, fuzzy and fuzzified) specified in the target case;
- a query of the form `SELECT` (λ) `* FROM` $R_1, \ldots R_k$ `WHERE` fc
  is generated and executed.

Retrieved cases will then be obtained as the tuples of the result table obtained from the query.

5 A Client/Server Architecture for Fuzzy CBR

Following the approach outlined in the previous sections, we have implemented a client/server architecture, able to process Fuzzy-SQL queries on top of a relational DBMS; since emphasis is given to case retrieval, the system provides a way of generating the relevant fuzzy query from a target case specification and can be used as a basis for the construction of specific applications where fuzzy case retrieval is required. The architecture of the system is shown in Figure 5. A case base is implemented using a standard relational database where cases are stored in a suitable set of tables; in addition to the *object database*, a *meta-database* storing all fuzzy information and knowledge is also required. At the current stage, all the meta-data (definitions of fuzzy predicates and operators) are stored in a relational format as well, in such a way that standard queries can be used to process this kind of information (we have chosen POSTGRES as DBMS for the current implementation).

The relational schema of the meta-database consists of the following tables:

$\texttt{DOMAINS(}\underline{\texttt{tab,field}}\texttt{,type);}$
$\texttt{FUZZY\_PRD(}\underline{\texttt{tab,field,lval}}\texttt{,order\_no,distribution,a,b,c,d);}$
$\texttt{FUZZY\_DD(}\underline{\texttt{tab,field,attrval,lval}}\texttt{,degree);}$
$\texttt{FUZZY\_OP(}\underline{\texttt{symbol}}\texttt{,operand\_t,description,cardinality,based\_on,distribution,a,b,c,d);}$
$\texttt{SIM\_TAB(}\underline{\texttt{op\_name,tab,field,aval1,aval2}}\texttt{,degree);}$

Underlined fields represent primary keys of the tables. The `DOMAINS` table is meant to represent the type of the fuzzy domain defined on the pair `table,field`; the supported types are: 'x' for no fuzzy domain, 'a' for continuous distribution on discrete domain, 'c' for continuous distribution on continuous domain, 'd' for discrete distribution on discrete domains.

Table `FUZZY_PRD` represents the definition of continuous fuzzy distribution on either a continuous or discrete domain; in particular, each tuple defines a fuzzy predicate with linguistic value `lval` on the attribute `field` of table `tab` with a continuous fuzzy distribution having 4 real reference points: `a` the last point with membership degree 0, `b` the first point with degree 1, `c` the last point with degree 1 and `d` the first point with degree 0 again. The attribute `distribution` of the table is intended as a code for the type of distribution; for instance, a trapezoidal distribution will be identified by a code different than the code for a triangular distribution. In particular, the difference between a trapezoidal and a triangular distribution is that in the latter case $\texttt{a} < \texttt{b} = \texttt{c} < \texttt{d}$, while in the former $\texttt{a} < \texttt{b} < \texttt{c} < \texttt{d}$. Other possible distributions are for example the *crisp then decreasing* distribution ($\texttt{a} = \texttt{b} = -\infty < \texttt{c}$

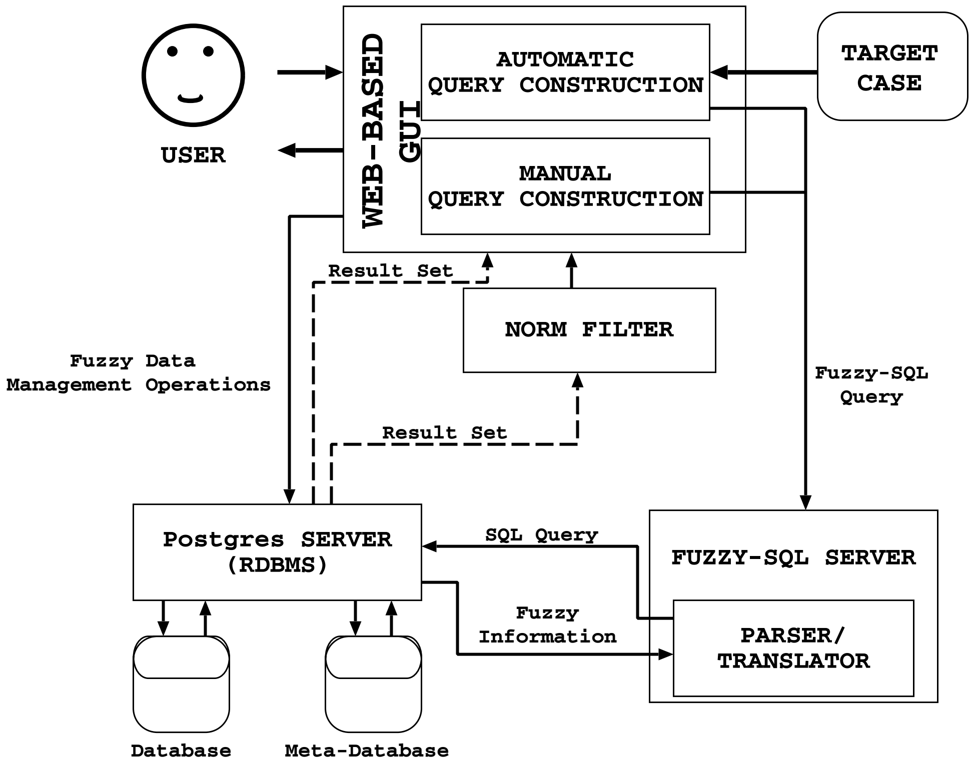


Figure 5: System Architecture

$< \mathtt{d}$), the *increasing then crisp* distribution ($\mathtt{a} < \mathtt{b} < \mathtt{c} = \mathtt{d} = +\infty$)s, etc ... The attribute `order_no` simply defines an order on the linguistic values that are defined on the same fields on a table.

Table `FUZZY_OP` is very similar, since it allows the definition of a continuous distribution corresponding to a fuzzy operator; specific information about the operator cardinality (`cardinality`), the operand types (`operand_t`) and the function on which the operator is based (`based_on`) are also stored. The attribute `symbol` stores the symbol used for the operand in the Fuzzy SQL queries.

Table `FUZZY_DD` deals with discrete distributions; the fuzzy value `lval` defined on the table `tab` and attribute `field` is defined such that the value `attrval` of `field` has a membership degree in the fuzzy set equal to `degree`.

Finally, table `SIM_TAB` is used to store information for similarity relations; a similarity called `op_name` is defined on table `tab` and attribute `field` such that the pair of attribute values `aval1` and `aval2` has similarity equal to `degree`.

Concerning the rest of the architecture, the actual core is a Fuzzy-SQL Server written in Java. It is devoted to the processing of the fuzzy query, by means of a Parser/Translator, implemented using Lex and Yacc; the syntax of the fuzzy query is checked and a standard SQL query is then generated, by deriving the DNCs using the fuzzy knowledge in the meta-database. The DB server and the Fuzzy-SQL server do not need to run on the same host and communicate via JDBC driver.

Finally, a Web-Based Graphical User Interface has been implemented as a Java applet, exploiting standard browser capabilities. The GUI fulfills the following requirements:

- automatic Fuzzy-SQL query generation from a target case template;
- manual Fuzzy-SQL query construction;
- meta-data management operations (definition of fuzzy knowledge) and database administration.

The GUI applet communicates directly with the DB server for database administration operation (as well as for standard querying) via JDBC, while a socket-based communication is exploited for the connection with the Fuzzy-SQL server.

The only constraint, in the current version, is that the Fuzzy-SQL Server must run on the same machine where the WWW server is running, because of the use of Java applets in the implementation of the GUI; no constraint is put on the location of the DBMS Server. The client browser, the DBMS and the Fuzzy-SQL Server can in principle be on different machines. Finally, it is worth noting that in figure 5, a specific module called Norm Filter is shown; this may be needed in case *norms* different than min and max are adopted, in order to reduce the result set to the desired λ-cut. As mentioned before, this filter would not be necessary if the Fuzzy-SQL Server includes in the translation process external function calls able to verify, during query processing, the degree of satisfiability of the tuples; this capability must be supported by the DBMS and the Fuzzy-SQL Server has to use the meta-database for function implementation. The different data flow of the result set in the different situations (use of the filter or not) are shown in figure 5 with dashed lines. At the current stage, the use of external functions, even if supported by POSTGRES, has not been implemented yet.

6 Application Examples

6.1 Travel Suggestion

A first application we have tested concerns the construction of a fuzzy retrieval system for an on-line travel agency. We have been inspired on that by several examples that may be found on the Web and

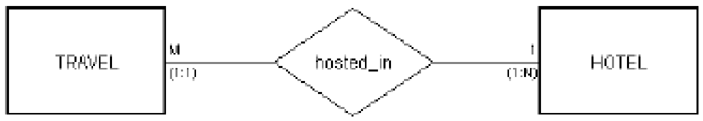


Figure 6: ER model for the travel database

Table.Field	Fuzzy Values	Distribution/Domain Type
hotel.stars	low_cat, med_cat, luxury	continuous/discrete
travel.duration	short, standard, long	continuous/discrete
travel.number_of_persons	small, medium, large	continuous/discrete
travel.price	very_cheap, cheap, medium, expensive, very_expensive	continuous/discrete

Table 1: Fuzzy predicates for the travel domain.

by the fact that a specific database containing travel information is available from the AI-CBR web site (`www.ai-cbr.org`). The object database is very simple and the conceptual data schema is reported in figure 6. The corresponding relational schema is as follows:

```
HOTEL(id,name,stars);
TRAVEL(journeycode,holidaytype,price number_of_persons,region,transportation,duration,
   season,accommodation,hotel);
```

A foreign key constraint is defined on table `TRAVEL` as

```
   foreign key(hotel) references HOTEL(id)
```

Field names can be considered self-explaining; the domain of each attribute in both tables is actually a discrete domain (either an `integer` or a `varchar` SQL type), so both discrete and continuous fuzzy distributions can be defined (i.e. `DOMAINS.type` in the meta-database can be either 'a' or 'd'). A set of fuzzy predicates has been defined on this schema as shown in table 1. The plotting of the predicate *luxury* for `hotel.stars` as produced by the system in shown as an example in figure 7. In addition to these predicates, we have also defined a similarity relation *sim* on the attribute `holidaytype` of table `travel`. The admissible value for such an attribute are: *Active, Bathing, Education, Language, Recreation, Skiing, Wandering, City*; for instance an *Active* type of holiday has a 0 similarity degree with a *Bathing* type of holiday, while a *City* type has a 0.7 similarity degree with an *Education* type. Finally, some fuzzy operators have been defined as well; for example the operator *much less than* for the attribute `travel.price` (`<<p`) and the operator *near* for `travel.number_of_persons` (`~nop`), both binary and defined on the difference of the operands. The fuzzy membership functions of such operands are shown in figure 8.

6.2 Example Queries and Retrieval

The object database we have used for our examples contains 1024 cases, each one corresponding to a different travel; actually this means that the view obtained as

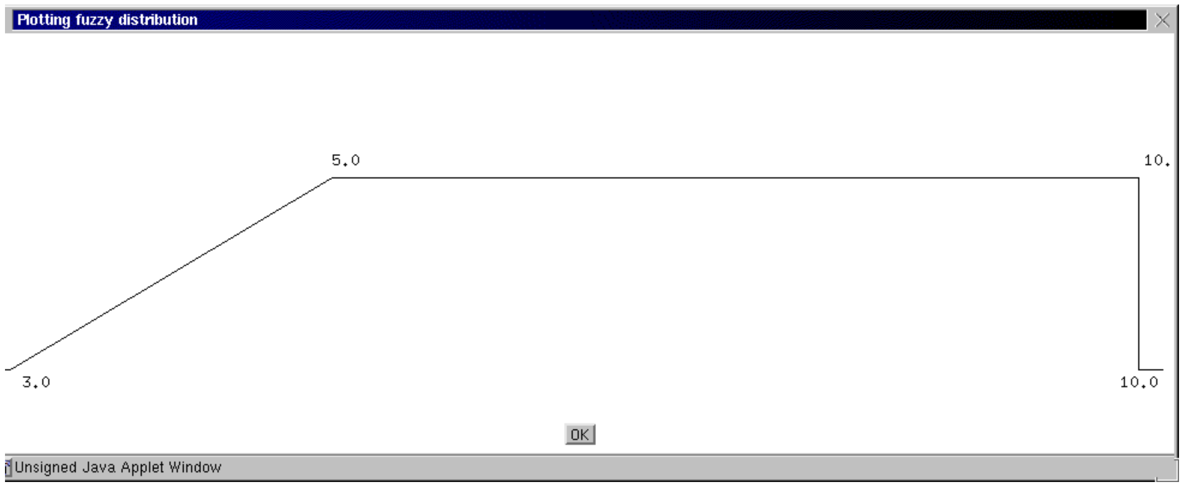


Figure 7: Plotting of the fuzzy membership of hotel.stars=luxury

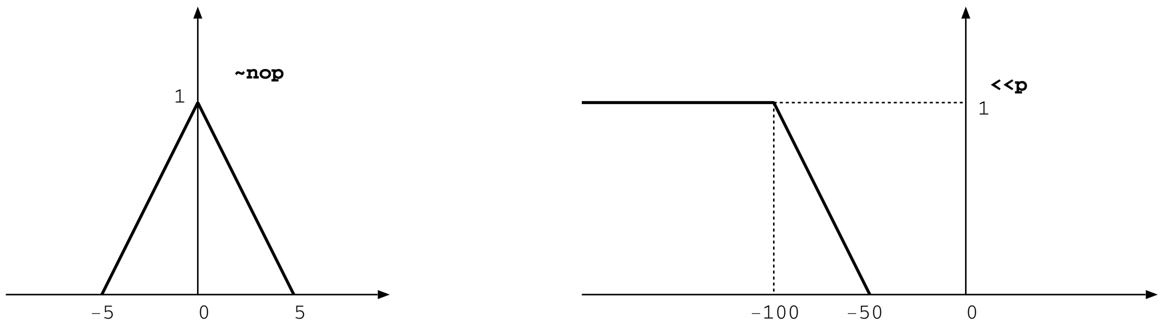
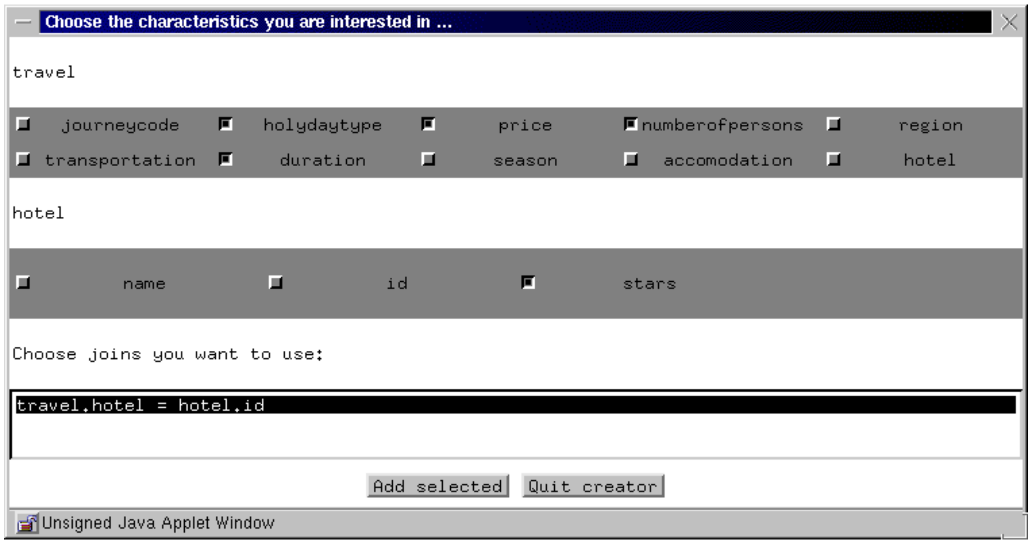


Figure 8: Membership function for operands `<<p` and `nop`.


Choose the characteristics you are interested in ...

travel

☐ journeycode	☑ holydaytype	☑ price	☑ numberofpersons	☐ region
☐ transportation	☑ duration	☐ season	☐ accomodation	☐ hotel

hotel

☐ name	☐ id	☑ stars

Choose joins you want to use:

travel.hotel = hotel.id

Figure 9: Template for Travel Features Selection

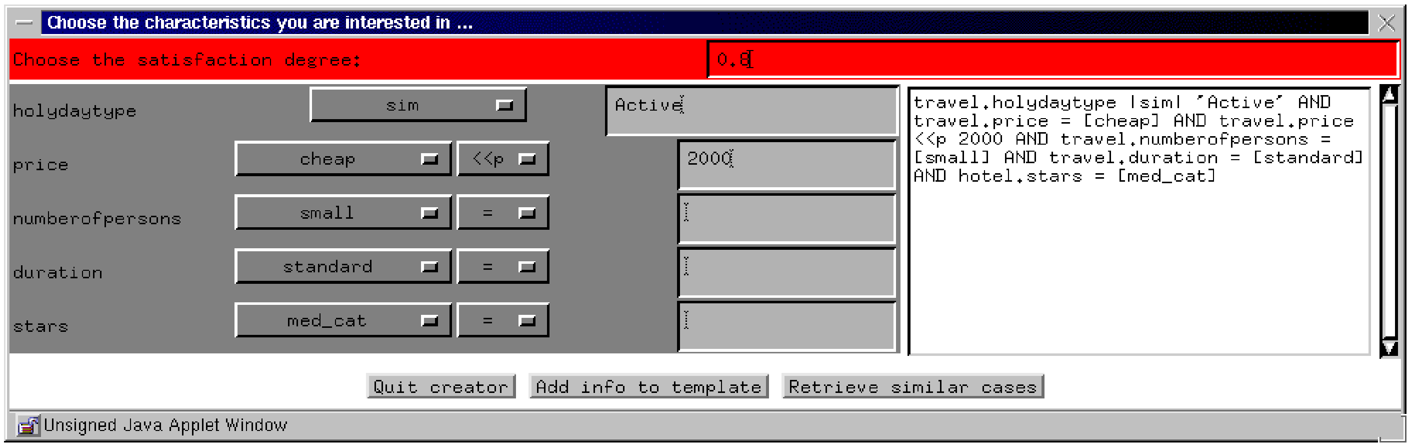
```
SELECT * FROM travel,hotel WHERE travel.hotel=hotel.id
```

contains 1024 tuples.
A target or query case is constructed by the user through a suitable template, where the feature he/she is interested in are selected and instantiated (in a mixed crisp and fuzzy way). The user can be either an end-user interested in finding a suitable travel form him/her (as in a B2C e-commerce application) or a travel agent interested in knowing the proposal of a tour operator that can be adopted for his/her agency (as in a B2B e-commerce application). Suppose for example that a travel agent is interested in analyzing cases corresponding to a type of holiday *similar to Active*, with a *cheap* price (and in fact much less than 2000 DM), with a *small* number of people, with a *standard* duration and which have been hosted in a *medium category* hotel. He/She can select the interesting features by a dialog window as shown in figure 9. A second dialog window is used to instantiate each feature as needed (see fig. 10). At this stage, the user has also to specify how strict are the conditions he/she requires for the retrieval, by specifying a number between 0 and 1, with the meaning that, closer is the number to 1, stricter is the retrieval condition. The specified number is used by the fuzzy query constructor module as the λ threshold for the query, while the user specified retrieval conditions will be used in the WHERE clause of the generated query.

In this example, if a threshold of 0.8 is specified (as in fig. 10), the following query is generated[5]:

```
SELECT (0.8) *
FROM travel, hotel
WHERE travel.hotel = hotel.id AND
travel.holydaytype |sim| 'Active' AND
travel.price = [cheap] AND travel.price <<p 2000 AND
```

[5]Special syntactical notation has been introduced in our Fuzzy-SQL language to identify fuzzy predicates (e.g. square brackets) or similarity relations (e.g. '|' symbol).


Choose the characteristics you are interested in ...
Quit creator
Add info to template
Retrieve similar cases
Unsigned Java Applet Window

```
Choose the satisfaction degree:  0.8

holydaytype        sim              Active
price              cheap    <<p     2000
numberofpersons    small    =
duration           standard =
stars              med_cat  =
```

```
travel.holydaytype |sim| 'Active' AND
travel.price = [cheap] AND travel.price
<<p 2000 AND travel.numberofpersons =
[small] AND travel.duration = [standard]
AND hotel.stars = [med_cat]
```

Figure 10: Query Construction Template

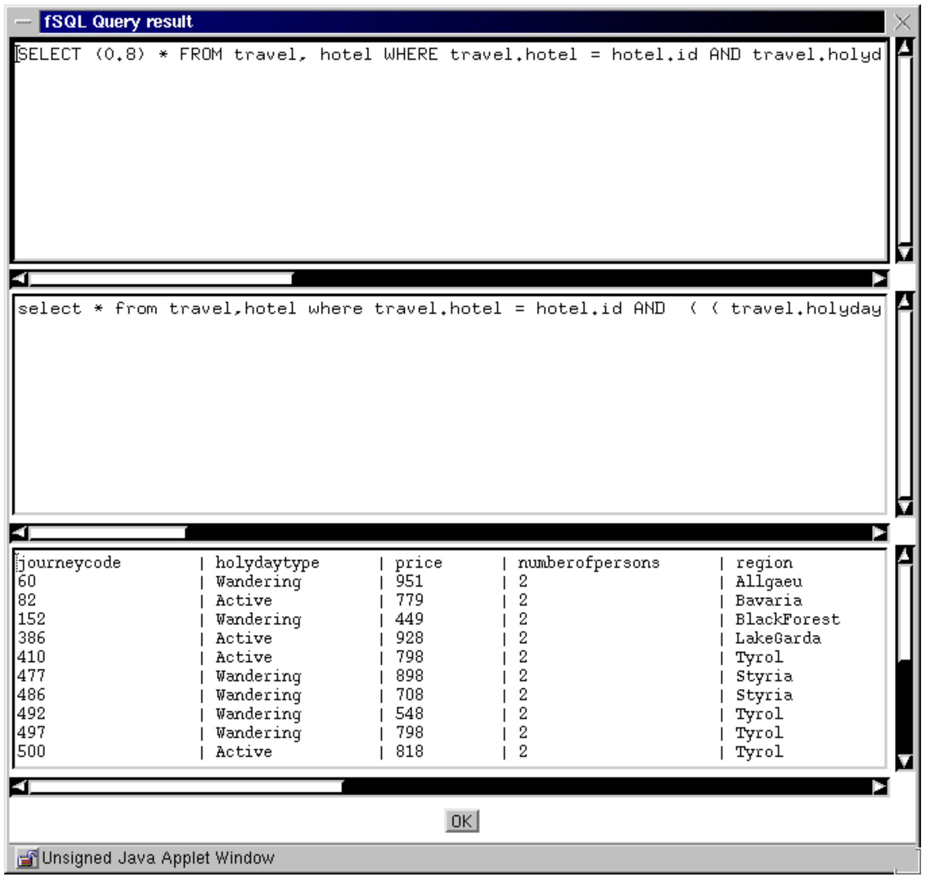
```
travel.numberofpersons = [small] AND
travel.duration = [standard] AND
hotel.stars = [med_cat]
```

corresponding to the following standard SQL query:

```
SELECT *
FROM travel,hotel
WHERE travel.hotel = hotel.id AND
      ((travel.holydaytype = 'Active') OR
       (travel.holydaytype = 'Skiing') OR
       (travel.holydaytype = 'Wandering')) AND
      (travel.price Between 420 And 1040) AND
      ((travel.price-2000) Between -100000 And -90) AND
      (travel.numberofpersons Between 0 And 2) AND
      (travel.duration Between 7 And 15) AND
      (hotel.stars = 3)
```

This results in a retrieval of 19 cases out of 1024 such that 6 cases are travels of type *Active*, 11 of type *Wandering* and 2 of type *Skiing*. Figure 11 shows the results of the retrieval provided by the system: the upper canvas reports the automatically generated Fuzzy SQL query, the middle one the translated standard SQL query and the lower one the set of retrieved cases.
If a more strict threshold is used (e.g. 0.9), the corresponding SQL query is:

```
SELECT *
FROM travel,hotel
WHERE travel.hotel = hotel.id AND
      (travel.holydaytype = 'Active')) AND
      (travel.price Between 435 And 1020) AND
```



```
SELECT (0.8) * FROM travel, hotel WHERE travel.hotel = hotel.id AND travel.holyd
```

```
select * from travel,hotel where travel.hotel = hotel.id AND  ( ( travel.holyday
```

```
journeycode          | holydaytype       | price       | numberofpersons      | region
60                   | Wandering         | 951         | 2                    | Allgaeu
82                   | Active            | 779         | 2                    | Bavaria
152                  | Wandering         | 449         | 2                    | BlackForest
386                  | Active            | 928         | 2                    | LakeGarda
410                  | Active            | 798         | 2                    | Tyrol
477                  | Wandering         | 898         | 2                    | Styria
486                  | Wandering         | 708         | 2                    | Styria
492                  | Wandering         | 548         | 2                    | Tyrol
497                  | Wandering         | 798         | 2                    | Tyrol
500                  | Active            | 818         | 2                    | Tyrol
```

Figure 11: Example of Retrieval for travel database

```
((travel.price-2000) Between -100000 And -95) AND
(travel.numberofpersons Between 0 And 2) AND
(travel.duration Between 7 And 15) AND
(hotel.stars = 3)
```

restricting the previous set of 19 cases to just 5 cases. As we can notice from the above query, these 5 cases only concern *Active* travels; one of the previous *Active* cases is not retrieved, because the restriction in the threshold also changes the price limits[6]. The travel agent can then analyze the result of retrieval and possibly to furtherly restrict the retrieval either by increasing the threshold or by changing some of the retrieval conditions.

6.3 Diabetic Patient Monitoring

The second example we discuss is a medical application.The introduction of Hospital Information Systems into clinical practice has led to the memorization of a huge quantity of data, automatically collected at each hospital during the every day work. This *operative* knowledge stores individual expertise, organizational practices, and, obviously, patients' clinical data. Since CBR is known to be a very suitable methodology for managing knowledge of the operative type [17], building a case retrieval system for a medical application would be helpful for maintaining expert's know-how over time, and for storing patients' clinical histories so that, even in presence of changes in the staff, the quality of care would not decrease due to a lack of information. Moreover, the use of a CBR application is motivated by the nature of medical decision making itself. As a matter of fact, physicians normally find useful to recall past situations similar to the current one and to adapt past solutions to the present problem, but the process is often biased by the tendency of recalling only more recent cases. It is therefore important to enable the retrieval of all the operative information, also related to older examples. A CBR approach would be particularly useful for supporting chronic diseases treatment, were patients are followed for long periods of time (sometimes for their whole life), and several cases are collected for each of them. Starting from these observations, in previous years we developed a complex decision support system that included a CBR module, and which was applied and successfully tested in the domain of type 1 diabetes mellitus [13]. In that system, where case retrieval was realized resorting to NN techniques, cases were stored as table rows in a relational database. Therefore, it would be easy to add the fuzzy querying facility to the existent architecture. In the following, we present some tests we made in this direction. The diabetes domain was chosen for testing Fuzzy-SQL also because we think that this solution could provide some advantages from the clinical viewpoint as well.

6.3.1 The Application Domain

Diabetes mellitus, one of the major chronic diseases in the industrialized countries, is a pathology deriving from an insufficient secretion of insulin by the pancreatic beta-cells. In particular, patients affected by type 1 diabetes don't have any residual endogenous insulin production. Therefore, they need exogenous insulin injections to regulate blood glucose, and to reduce the risk of invalidating later life complications. Implementing the therapy is therefore a complex task, that requires frequent contacts with the physician through clinical visits. During the problem identification and therapy revision process, the physician normally relies not only on structured domain knowledge, that is largely available and well assessed, but also on previous experience (e.g. the information that a certain insulin

[6]The excluded case has a price of 434 DM and does not satisfy the new retrieval condition.

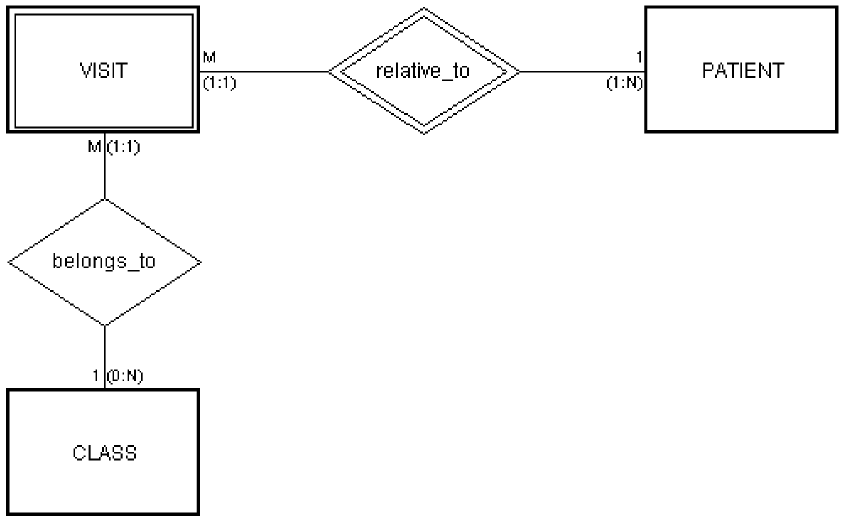


Figure 12: ER schema for DIABETES database

protocol has been applied in the past to that patient, or to patients with similar characteristics, leading to a positive/negative outcome): the use of case retrieval is therefore largely motivated and a periodical control visit can be naturally mapped to the concept of case.

6.3.2 The Database

Our case base is composed by about 150 visits (cases), belonging to 29 real pediatric patients. The conceptual ER schema of the database is reported in figure 12.

Cases are stored in the relational database corresponding to such a schema and can be logically thought as the tuples of the view corresponding to the join of the tables PATIENT, VISIT and CLASS. We avoid here the details of the relational schema, however it is worth noting that the relevant information for retrieval are contained in the tuples of VISIT; each visit is relative to a specific patient (whose general information is a tuple of PATIENT) that, at the visit time, belongs to one of the defined diagnostic classes stored in CLASS (see below).

Case features are the data collected during the visit, or abstractions made over these data. Some examples are: age, weight, height, HbA1c, insulin requirement, distance from onset. In particular HbA1c is the main indicator of the presence of hyperglycemia in the latest 60 days, insulin requirement represents daily insulin intake, and the distance from onset is the number of months since diabetes was diagnosed. The case solution is represented by the insulin therapy prescribed on the visit day, and the case outcome is summarized by the metabolic indicators (mainly HbA1c) collected during the following visit (that is another case in the table, belonging to the same patient). On the basis of medical knowledge, cases can be classified resorting to a set of prototypical classes [13], that describe the typical pathological or clinical course conditions in which pediatric patients may incur. Classes are mutually exclusive. In particular, for our tests, we considered some cases belonging to the three following classes: Anorexia, Bulimia and Clinical Remission. Anorexia and Bulimia are two nutritional disorders, that can

Table.Field	Fuzzy Values	Distribution/Domain Type
visit.age	infant, puberal, young, adult	continuous/discrete
visit.HbA1c	good_control, sufficient_control, poor_ control	continuous/continuous
visit.distance_from_ onset	short, medium, long	continuous/discrete

Table 2: Fuzzy predicates for the diabetes domain.

be often diagnosed in diabetic patients. Clinical Remission is a condition in which all newly diagnosed patient incur, generally until 24 months after diabetes onset; during this period, they normally have a good metabolic control (due to the great efficacy of the recent introduction of an exogenous insulin therapy), but they may suffer from hypoglycemia.

The diabetes meta-database stores fuzzy predicates and fuzzy operators helpful for querying the case base. Table 2 lists some of the predicates and the case feature over which they are defined. All these predicates are continuous, and the table reports if their domain is discrete or continuous as well. When more than one predicate is defined, they may be partially overlapping.

The following continuous operators were defined: ∼w (near) over the weight attribute, and ≫h (much greater) over the height attribute. The operator ∼w is defined by means of a trapezoidal distribution, where $a = -4$, $b = -2$, $c = 2$ and $d = 4$. On the other hand, the operator ≫h is defined through an *increasing then crisp* distribution, with $a = 0$, $b = 10$ and $c = d = +\infty$.

6.3.3 Example Queries

From medical knowledge, we know that pediatric patients belonging to the class Anorexia are usually under-weight, have a good metabolic control (i.e. values of HbA1c ≤ 7.0), are in the puberal age, and their distance from onset is at least 22 months. To reproduce this situation, we used the fuzzy predicates good control (over HbA1c), puberal (over age), and medium or long (over distance from onset). Moreover we limited our search to patients whose weight was $\sim w$ 52 Kg, and whose height was ≫h 150 cm. The query is reported below:

```
SELECT (lambda) * FROM visit
WHERE Hba1c=[good_control] AND
age=[puberal] AND
(distance_from_onset=[long] OR distance_from_onset=[medium]) AND
weight ~w 52 AND
height >>h 150
```

By choosing `lambda`=0.9 we were able to retrieve 9 cases, all belonging to the Anorexia class. The retrieval results matched our prior knowledge, confirming the capability of the Fuzzy-SQL system of correctly identifying the required tuples.

Similarly, patients belonging to the Clinical Remission class have a good metabolic control (i.e. values of HbA1c ≤ 7.0), are in the puberal age, and their distance from onset is less than 24 months. To reproduce this situation, we used the fuzzy predicates good control (over HbA1c), puberal (over age), and short or medium (over distance from onset). Moreover we limited our search to patients whose weight was ∼w 63 Kg, and whose height was ≫h 150 cm. The query is reported below:

```
SELECT (lambda) * FROM visit
```

```
WHERE Hba1c=[good_control] AND
age=[puberal] AND
(distance_from_onset=[long] OR distance_from_onset=[medium]) AND
weight ~w 63 AND
height >>h 150
```

Again the fuzzy query with a high lambda (0.9) retrieved 2 cases, both belonging to the Clinical Remission class. We then repeated the second query, by choosing a lower lambda (0.6). This time we retrieved 7 cases, 2 of which belonging to Anorexia, and 5 to Clinical Remission. No cases belonging to the Bulimia class were retrieved: weight has to be > 63 Kg for bulimic patients whose height is $\gg$h 150 cm.

From this simple test, we are able to draw some conclusions. First, it seems that some features don't discriminate very well among certain classes; in the example some anorexic patients, and some in the Clinical Remission phase, can be retrieved by the same query. Only the choice of a high lambda can restrict the result to cases belonging to just one class. This may indicate that, along certain directions (i.e. the query features) the two classes are *similar*: to obtain a complete discrimination we have to introduce stricter conditions (e.g. imposing that the distance from diabetes onset is short, and not medium), or conditions on additional features. Querying the visit table is then a way of performing inter-class retrieval, i.e. to obtain cases from different classes, that, limiting the analysis to the features in the query, are similar to the target case. Intra-class retrieval (i.e. retrieval of cases belonging to a single class) can be obtained by the same query, simply by increasing the value of lambda. Identifying such *similarities* among the classes could be an added value from the clinical viewpoint: it would be possible to highlight relations among different prototypical conditions that are not clearly stated by medical knowledge, but that can be learned from real data.

Moreover, the Fuzzy-SQL approach could be used as the pre-processing step in a data mining application, for reducing the number of data to work with. The fuzzy query would retrieve cases that are similar from some viewpoints (again, along the directions of the attributes in the query): then classical data mining techniques could be applied to discover additional correlations among the other features of the reduced case set. As a matter of fact, an accurate pre-selection seems to be particularly critical when dealing with medical data.

Finally, we also envision the possibility of adopting Fuzzy-SQL as a mean for evaluating the quality of patient's care. For example, in the diabetes application, it would be possible to retrieve cases with an unsatisfactory outcome (where *unsatisfactory* is a fuzzy concept), and to review the therapy (i.e. the solution of the previous case in terms of time, belonging to the same patient) that led to that outcome. It would then be possible to verify if *similar* mistakes (e.g. non adherence to clinical guidelines) were made in the therapies under examination, or if wrong therapeutic choices were always taken by the same physician.

7 Conclusions and Related Works

In the present paper a case retrieval client/server architecture, exploiting a fuzzy extension to SQL has been presented. The system is able to automatically produce a fuzzy query from a target case template and can exploit the support of a standard relational DBMS for storing and retrieving cases. Unlike similar approaches, the emphasis is given to acceptance criteria defined through fuzzy distribution, instead that to distance functions as in NN approaches.

The main contributions of the present papers are:

1. the use of standard fuzzy logic, in order to define acceptability criteria for case retrieval, by taking advantage of the standard definition of λ-cut of a fuzzy set and of standard fuzzy logic connectives to realize the aggregation of local acceptability measures (defined at the feature level) into a global acceptability measure (at the case level);

2. the exploitation of database techniques, both for storing and retrieving cases in standard relational databases, bridging the gap between the need for flexible querying and the need for maintaining usual databases already present in application fields.

Both issues have been already faced in the past, but usually in different contexts. In [7], a difference is pointed out between case filtering (as preliminary to retrieval) and case selection; hierarchical fuzzy classification is proposed for the first step, while a particular pattern matching technique exploiting fuzzy information is used for case selection.

More emphasis to the problem of aggregating information is given in [19] where the use of fuzzy integrals is suggested as a suitable methodology for the synthetic evaluation of cases; a financial application is used to present the approach.

Fuzzy extension to standard k-NN techniques is presented in [6] as a useful way for dealing with the challenging problem of retrieving cases for whether prediction. The approach aims at retrieving the k most similar cases, in fuzzy terms of similarity, with respect to a target case description. However, the main goals are slightly different than ours, putting emphasis on classification and prediction via fuzzy weighting.

Concerning the problem of exploiting standard database techniques for case retrieval, particularly significant is the work presented in [14]. The use of a relational database and of standard SQL is exploited for implementing case retrieval in a commercial CBR toolbox called `orenge`; however, differently from our approach, no fuzzy extension to SQL is proposed, since the goal is to implement standard (crisp) k-NN via a query relaxation technique, iterated until the desired result is reached. Commonalities with our approach can be pointed out, since both approaches are implemented on top of a relational DBMS and suitable thresholds must be defined for case retrieval.

Our future work will concentrate on the possibility of creating more parametric queries, also supporting the different importance a given attribute may have in specific retrievals.

References

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations and system approaches. *AI Communications*, 7(1):39–59, 1994.

[2] P. Bosc and O. Pivert. Fuzzy queries in conventional databases. In L. Zadeh and J. Kacprzyk, editors, *Fuzzy Logic for the Management of Uncertainty*, pages 645–672. John Wiley, 1992.

[3] P. Bosc and O. Pivert. SQLf: a relational database language for fuzzy querying. *IEEE Transactions on Fuzzy Systems*, 3(1), 1995.

[4] B.P. Buckles and F.E. Petry. A fuzzy representation of data for relational databases. *Fuzzy Sets and Systems*, 7:213–226, 1982.

[5] H-D. Burkhard. Extending some concepts of CBR: foundations of case retrieval nets. In M. Lenz, B. Bartsch-Spoerl, H-D. Burkhard, and S. Wess, editors, *Case Based reasoning Technology: from Foundations to Applications*, pages 17–50. LNAI 1400, Springer, 1998.

[6] B.K. Hansen. *Whether prediction using CBR and fuzzy set theory.* Master Thesis, Dalhousie University, 2000. `http://www.cs.dal.ca/~ bjarne/thesis.pdf`.

[7] M. Jaczynski and B. Trousse. Fuzzy logic for the retrieval step of a case-based reasoner. In *2nd European Workshop on Case-Based Reasoning - EWCBR94*, pages 313–320, Chantilly, France, 1994.

[8] J. Kacprzyck and A. Ziolowski. Database queries with fuzzy linguistic quantifiers. *IEEE Transactions on Systems, Man and Cybernetics*, 16(3), 1986.

[9] H. Kitano and H Shimazu. The experience-sharing architecute: a case study in corporate-wide case-based software quality control. In D.B. Leake, editor, *Case Based Reasoning: Experiences, Lessons and Future Directions.* AAAI Press, 1996.

[10] J.L. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann, 1993.

[11] C.T. Liu and C.S. George Lee. *Neural Fuzzy Systems.* Prentice Hall, 1996.

[12] M. Medina, O. Pons, and M.A. Vila. GEFRED, a GEnerilized Fuzzy model for REletional Databases. *Information Sciences*, 76(1-2):87–109, 1994.

[13] S. Montani, R. Belazzi, L. Portinale, and M. Stefanelli. A multi modal reasoning methodology for managing IDDM patients. *International Journal of Medical Informatics*, (58-59):243–256, 2000.

[14] J. Schumacher and R. Bergmann. An efficient approach to similarity-based retrieval on top of relational databases. In E. Blanzieri and L. Portinale, editors, *Proc. 5th EWCBR*, pages 273–284, Trento, 2000. Lecture Notes in Artificial Intelligence 1898.

[15] M. Schumacher and T. Roth-Berghofer. Architectures for integration of CBR systems with databases for e-commerce. In *Proc. 7th German Workshop on CBR (GWCBR'99)*, 1999.

[16] H. Shimazu, H. Kitano, and A. Shibata. Retrieving cases from relational databases: another strike toward corporate-wide case-based systems. In *Proc. 13th Intern. Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 909–914, 1993.

[17] G. Simon and M. Grandbastein. Corporate knowledge: a case study in the detection of metallurgical flaws. In *Proceedings of ISMICK 95*, pages 43–52, Compiegne, France, 1995.

[18] I. Watson. *Applying Case-Based Reasoning: Techniques for Enterprise Systems.* Morgan Kaufmann Publ., 1997.

[19] R. Weber-Lee, R. Barcia, and S. Khator. Case-based reasoning for cash flow forecasting using fuzzy retrieval. In *Proc. First International Conference on Case-Based Reasoning - ICCBR95*, pages 510–519, Sesimbra, Portugal, 1995.

[20] W. Wilke, M. Lenz, and S. Wess. Intelligent sales support with CBR. In M. Lenz, B. Bartsch-Spoerl, H-D. Burkhard, and S. Wess, editors, *Case Based reasoning Technology: from Foundations to Applications*, pages 91–113. LNAI 1400, Springer, 1998.

[21] X. Wu. Fuzzy interpretation of discretized intervals. *IEEE Transactions on Fuzzy Systems*, 7(6):753–759, 1999.