Exploiting Fuzzy-SQL in Case-Based Reasoning

Luigi Portinale and **Andrea Verrua**
Dipartimento di Scienze e Tecnologie Avanzate (DISTA)
Universita' del Piemonte Orientale "Amedeo Avogadro"
C.so Borsalino 54 - 15100 Alessandria (ITALY)
e-mail: portinal@mfn.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 present a proposal where case retrieval is implemented by using a fuzzy version of SQL. In the presented approach a fuzzy notion of similarity is adopted and an extended version of SQL, dealing with fuzzy predicates, is introduced. A case-based system prototype exploiting Fuzzy-SQL as a retrieval engine is then presented.

Introduction

The use of database techniques for supporting the construction of case-based systems is attracting serious attention; the reasons are twofold: *i*) if data of interest are already stored in a database, the database itself can be used as a case base; *ii*) 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, whose records are easily interpretable as "cases" for a case-based recommendation system (Schumacher & Bergmann 2000; Schumacher & Roth-Berghofer 1999; Wilke, Lenz, & Wess 1998), as well as in the broad context of knowledge management, for the implementation of corporate-wide case-based systems (Kitano & Shimazu 1996; Shimazu, Kitano, & Shibata 1993). 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 single features are combined in a weighted way to get a global similarity between a retrieved and a target case. In (Burkhard 1998), 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.

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 completed defined in fuzzy terms, by means of the usual combinations of local distributions through specific defined *norms* (Liu & Lee 1996). 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 (Bosc & Pivert 1995), extending standard SQL in order to deal with *fuzzy queries*. A system prototype able to generate a Fuzzy-SQL query from a case specification and working on top of a standard relational DBMS has been implemented; its main features will be presented in the paper.

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. 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

[1] Actually, in (Wilke, Lenz, & Wess 1998) a revision of the classical CBR cycle presented in (Aamodt & Plaza 1994) is proposed, by specifically taking into account the e-commerce application; however no particular database technique is specifically proposed for implementing this new cycle.

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: *i*) explicit representation of vague and imprecise information inside the database (Buckles & Petry 1982; Medina, Pons, & Vila 1994); *ii*) vague and imprecise querying on a regular database (Bosc & Pivert 1995; Kacprzyck & Ziolowski 1986). 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 (Bosc & Pivert 1995) 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 in a regular one (*derivation principle*).

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 finally show how this can be actually implemented for case retrieval purposes. 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.

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 or a gaussian-like possibility 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 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 to 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* (Liu & Lee 1996). 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.

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 (Bosc & Pivert 1995), 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, max as norms, then it is possible to derive from a Fuzzy-SQL query, an SQL query returning exactly the λ-cut required. 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 be $DNC(P, \geq, \lambda)$ the derived

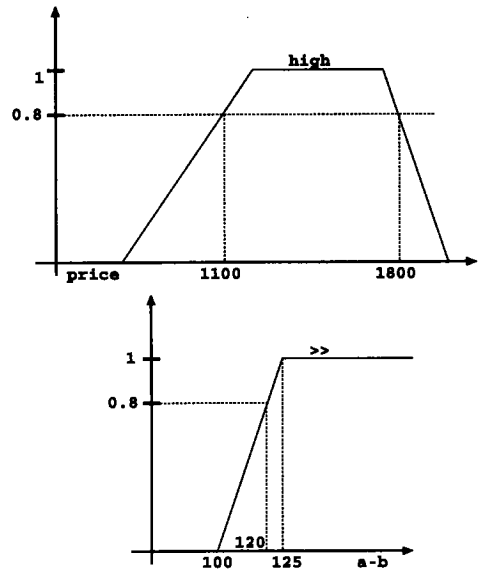

Figure 1: Fuzzy Distributions

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

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

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

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 1 shows the fuzzy distribution of `high` as well as the distribution of a fuzzy operator >> (much higher than). The condition $(\texttt{price} = \texttt{high} \wedge \texttt{price} >> 1000) \geq 0.8$ will hold iff $\min(\texttt{price} = \texttt{high}, \texttt{price} >> 1000) \geq 0.8$, iff $(\texttt{price} = \texttt{high}) \geq 0.8 \wedge (\texttt{price} >> 1000) \geq 0.8$ 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. 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.

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

An Architecture for Fuzzy Case Retrieval

Following the approach outlined in the previous sections, we have implemented a system 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. The architecture of the system is shown in Figure 2. 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). A Fuzzy-SQL Server 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.

Finally, a Web-Based Graphical User Interface has been implemented in Java, exploiting standard browsers' 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.

While the user can always require a suitable retrieval by constructing a fuzzy query by hand, the system also provides a more case-based interface able to construct a fuzzy query from a target case specification. We restricted our attention to flat representation of cases through $\langle feature, value \rangle$ pairs. However, while cases are stored in the database by using, for each attribute, only values from the corresponding domain, target 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 target case, three different possibilities can be used:

1. specification of a linguistic value;
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

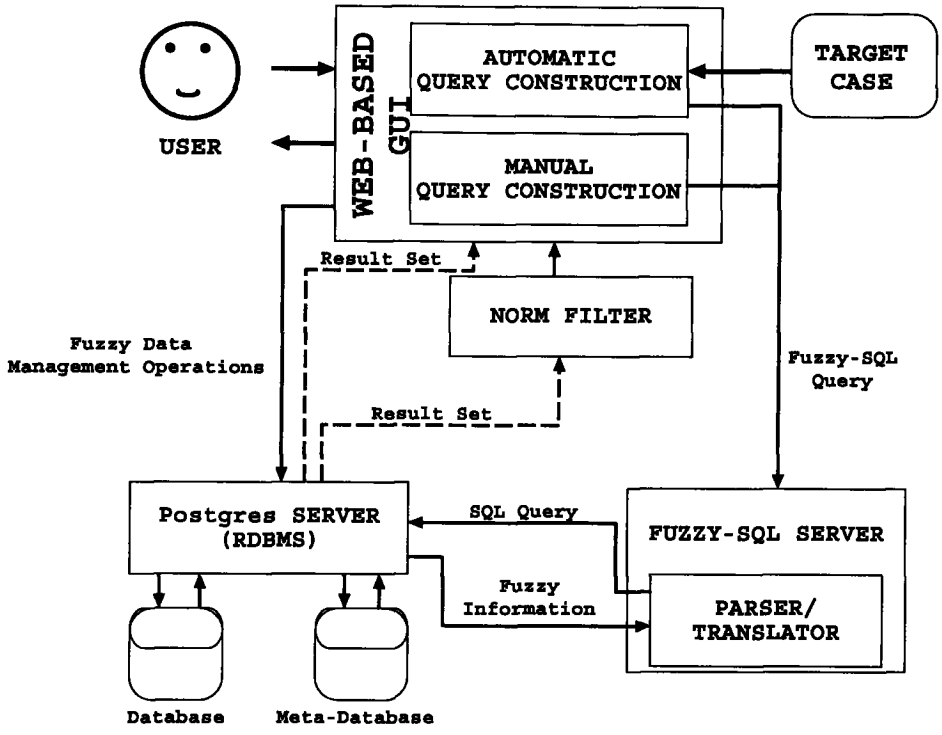

Figure 2: System Architecture

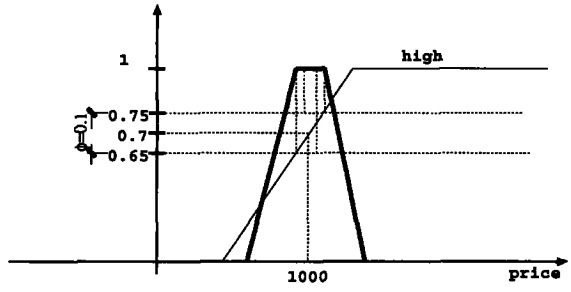

Figure 3: Fuzzification Example

instead of $A = a$[3]. 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 3 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 3; 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 (Wu 1999). 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 *t-conorm* (usually the max operator) to combine the results from each fuzzy set.

In addition to the specification of a crisp or linguistic value, in the target case template the user can also specify additional conditions involving specific (possibly fuzzy) operators (e.g. the user can specify that it requires cases with `price=high` and `price >>` 1000). Finally, a global retrieval threshold λ is specified. A Fuzzy-SQL query can then be generated as follows: *i*) one or more tables $(R_1, \ldots R_k)$ whose rows contain the cases of interest for retrieval are selected; *ii*) a fuzzy condition fc is created by taking the AND of all the expressions (crisp, fuzzy and fuzzified) specified in the case template; *iii*) a query of the form
`SELECT` (λ) * `FROM` $R_1, \ldots R_k$ `WHERE` fc
is generated. Figure 4 shows a screen shot of part of the interface for constructing the query (the example uses a slight variation of the `travel` database available from `http://www.ai-cbr.org`). The execution of this query will then return a set of cases (in the form of table tuples) having the required degree of similarity (i.e. at least λ) with the target case.

From the architectural point of view, the module of the systems have been designed to run on the Internet platform, so communication among module is realized by means of TCP/IP protocol. The only constraint, in the current version, is that the Fuzzy-SQL Server must run on the same ma-

[3] A more general solution could be to provide specific operators of nearness $\mathbf{near}_a$ for each individual scalar attribute A.

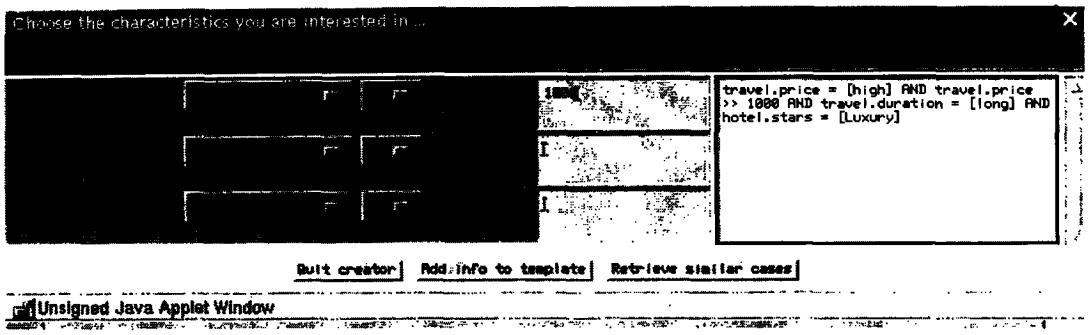

Figure 4: WHERE Clause Construction

chine 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. Since the user interface is realized via standard browser facilities, 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 2, 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 2 with dashed lines. At the current stage, the use of external functions, even if supported by POSTGRES, has not been implemented yet.

Conclusions

In the present paper a case retrieval system, exploiting a Fuzzy-SQL query language 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. 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

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

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

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

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

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

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

Liu, C., and Lee, C. G. 1996. *Neural Fuzzy Systems*. Prentice Hall.

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

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

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

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

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

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