HAL
open science

Flexible Querying using Disjunctive Concepts

Grégory Smits, Marie-Jeanne Lesot, Olivier Pivert, Ronald R Yager

To cite this version:

Grégory Smits, Marie-Jeanne Lesot, Olivier Pivert, Ronald R Yager. Flexible Querying using Disjunctive Concepts. International Conference on Flexible Query Answering Systems, Sep 2021, Bratislava, Slovakia. hal-03276700

HAL Id: hal-03276700

https://hal.science/hal-03276700v1

Submitted on 2 Jul 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Flexible Querying using Disjunctive Concepts

Grégory Smits[1], Marie-Jeanne Lesot[2], Olivier Pivert[1], and Ronald R. Yager[3,4]

[1]University of Rennes – IRISA, UMR 6074, Lannion, France
{gregory.smits,olivier.pivert}@irisa.fr
[2]Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
marie-jeanne.lesot@lip6.fr
[3]Faculty of Science, King Abdulaziz University, Jeddah, Saudi Arabia
[4]Machine Intelligence Institute, Iona College, New Rochelle, NY
yager@panix.com

Abstract. A DB querying system is said to be flexible if it adapts to the end user expectations and expertise. This paper introduces a novel strategy to fuzzy querying that reduces the gap between complex search conditions end users have in mind and formal queries understood by the underlying DB system. In the Flexible Querying By Example paradigm, the proposed strategy, called DCQ standing for Disjunctive Concept Querying, extends a flexible querying system with subjective disjunctive concepts: it proposes two stored procedures that can be embedded in any relational database management system to build a formal query from a few user-given examples that represent the diversity of what the user is looking for. The first procedure infers the membership function of the implicit imprecise concept underlying the provided examples, with the specificity of allowing for complex disjunctive concepts: it is able to both capture properties shared by most of the selected representative tuples as well as specific properties possessed by only one specific representative tuple. The second procedure allows to exploit the resulting fuzzy concept in a query.

Keywords: Fuzzy querying, disjunctive fuzzy concept, fuzzy measure, Choquet integral

1 Introduction

End users expect a lot from their stored data and from querying systems that make the link between users and data. As a first interaction with a querying system, users have *in fine* to express their information need using a formal query language: SQL in our case as relational data are considered. Faithfully translating concrete information needs into SQL queries is however a difficult task, and, to be considered as smart, a DataBase (DB) querying system should help users express what they are looking for. The Querying By Example (QBE) paradigm alleviates the user task, only requiring he/she provides a few input tuple examples or the evaluation of a few data instances. The difficulty is then to understand the underlying user need, which often corresponds to imprecise, complex, context-dependent and subjective concepts.

Table 1: Accomodation DB: excerpt of the *housing* relation extension
housing: {loan INT, surf INT, nbR INT, loc TEXT, outSurf INT}

	loan	*surf*	*nbR*	*loc*	*outSurf*
t_1	500	25	2	historical center	0
t_2	500	30	2	center	0
t_3	475	40	3	center	0
t_4	400	80	4	suburb	10
t_5	450	60	3	suburb	100
t_6	400	100	5	outskirts	500
t_7	800	25	3	historical center	0
t_8	500	20	1	center	10
...	...	...	...	...	...

Table 2: User given examples of *ideal housing*

	loan	*surf*	*nbR*	*loc*	*outSurf*
ih_1	400	30	2	center	0
ih_2	450	40	3	center	10
ih_3	400	80	4	suburb	10
ih_4	400	100	5	outskirts	500

As an illustrative example, consider a DB about accommodations for rent described by precise attribute values as surface, number of rooms, distance to center, loan, etc., and illustrated in Table 1, and a user looking for an *ideal housing* to start a professional life, for which he/she provides the instances given in Table 2. Several very different accommodations may satisfy this vague subjective concept of an ideal housing. The user may indeed be interested in a small flat without garden nor balcony if it is located close to the central and vibrant part of the city, or in a larger flat close to his/her office in the suburb or may even be willing to live in a small house with garden in the outskirts.

Other examples of such complex concepts are for instance: *atypical holidays*, *safe investment*, *promising student*, *good employee*, etc. Such complex searches are generally disjunctive by nature as very different tuples may satisfy a given concept, for various reasons. The key issue is then to infer automatically the definition of the concept underlying the provided representative examples and to retrieve the data instances satisfying it.

To the best of our knowledge, this issue of extending a flexible querying system with subjective disjunctive concepts using a QBE strategy has not been studied so far.

This paper introduces a cooperative querying strategy, called DCQ that stands for Disjunctive Concept Querying, to help users define fuzzy queries to retrieve from a relational DB tuples that best satisfy such complex concepts of a disjunctive nature. More precisely, we devise a Fuzzy Querying By Examples (FQBE) strategy that eases the concept definition step as it only requires the user to give a few representative examples of what he/she could be interested

in. From these examples, a membership function describing the associated imprecise underlying concept is automatically inferred exploiting the CHOCOLATE method [9]: a fuzzy measure is first built to quantify the extent to which combinations of attribute values match the underlying concept, as well as a measure quantifying the relevance of each attribute value individually. These two quantities are then aggregated using a Choquet integral, to determine the degree to which a point satisfies the concept. The paper proposes two stored procedures that can be embedded into any Relational DataBase Management System (RDBMS, PostgreSQL in our case). These two procedures, dedicated respectively to the definition of complex disjunctive fuzzy concepts and their use to fuzzy querying, can then serve as technical tools to develop enhanced DB querying or recommendation systems that better integrate the end user in data to knowledge translation processes.

The paper is structured as follows. Section 2 discusses related works and provides a reminder about the CHOCOLATE method the paper proposition exploits. Section 3 describes the proposed DCQ method and Section 4 a Proof-Of-Concept implementation of this strategy within a RDBMS.

2 Related Works

This section briefly reminds the main existing approaches to address the task of answering Query by Examples in data bases. It then describes some general works in the AI community regarding the definition and automatic learning of concepts based on some examples. It finally presents in more details the CHOCOLATE concept modeling approach on which the proposed DCQ strategy relies.

2.1 Flexible Querying and FQBE

Query by Example is a paradigm (initially introduced by M. Zloof [13] in the 70's) that makes it possible to interact with a data base and acquire results based on: i) one or several input tuples provided by the user or ii) the positive or negative evaluation of prototypical examples reflecting the content of the database.

To the best of our knowledge, three types of fuzzy QBE approaches can be found in the literature. The first one, by De Calmès et al. [3], presented by the authors as a case-based reasoning approach, looks for those items in the database that are similar to at least one example (wrt. all the attributes) and dissimilar to all counter-examples (wrt. at least one attribute). The goal of this approach is not to infer an interpretable fuzzy query expressing the preferences of the user but just to retrieve items based on a measure that combines similarity (with positive examples) and dissimilarity (with counter-examples).

In [12], Zadrozny et al. present a clustering-based approach that provides items iteratively until the user is satisfied. The new items to be assessed are selected with a k-NN algorithm. Positive and negative evaluations are used to find association rules determining whether some fuzzy predicates (from a list of

predefined linguistic terms) are relevant to the user. Then for each attribute, at best one linguistic term is selected to express the user preferences for this attribute, only if the support, confidence and lift for the association rule that found it are "high enough." As can be seen, this approach resorts to an iterative evaluation of examples until the user is satisfied, without ever evaluating a fuzzy query.

In [6], the authors propose an approach that infers a fuzzy query from user-assessed prototypical examples, based on an algorithm determining the fuzzy predicates that best represent the positive examples (and at the same time discards the negative ones). As in [12], they consider a fuzzy vocabulary for each attribute domain in the database. This vocabulary makes it possible to formulate, in a linguistic way, descriptions of the attribute values shared by positive examples that are not shared by counter-examples (defined as *characterizations*).

All these approaches aim to infer simple *conjunctive* fuzzy queries, sometimes not even explicitly. The DCQ technique we propose in the present paper makes it possible to build a fuzzy query involving complex fuzzy concepts that cannot be easily expressed by a conjunctive/disjunctive combination of atomic fuzzy terms. Moreover, it does not require the user to provide counter-examples but only a limited number of positive examples.

2.2 Concept Modeling

The issue of concept modelling, at the core of the QBE paradigm, has been widely studied from a general perspective in artificial intelligence and data management, for instance in the framework of Formal Concept Analysis [11], as well as its fuzzy extensions [2], or as fuzzy prototypes [8,5]. In both cases, concepts are based on a conjunctive view, imposing that all members of the concept have common attribute values. Only a few works during the 70's and the 80's consider disjunctive concepts, both in the cognitive science [1] and the AI fields [4,7]. Whereas the conjunctive normal form seems to be the most appropriate way to model knowledge taxonomies, it is also clear that more sophisticated aggregation operators have to be defined to infer complex search conditions from a tiny set of user-provided prototypical examples that roughly illustrate what he/she is looking for.

This corresponds to the underlying principle of the CHOCOLATE approach [9] that allows to relax this classical constraint, as detailed in the next subsection.

Another axis of discussion regarding concept modelling and learning relates to the relations between concepts, depending whether they are defined in opposition one to another or independently. The first case is considered when building prototypes or performing a classification task, it requires the availability of both examples and counter-examples. It is also applied in most QBE approaches, as discussed in the previous section. Dealing with each concept individually, as is the case for FCA or CHOCOLATE, means that only representatives of the current concept are required. CHOCOLATE in particular alleviates the requirements on the user, as it considers that only very few representative examples are available.

2.3 The CHOCOLATE Approach

The CHOCOLATE method [9] is a procedure to infer a membership function to describe a concept C from a user-provided set $\mathcal{E}_C$ of representative examples: $\mathcal{E}_C$ constitutes a partial extent for the underlying and implicit concept C. This section summarises CHOCOLATE three steps (see [9] for a detailed description).

Throughout the paper, the following notations are used: R is a universal relation or the result of a join query, whose schema is $R : \{A_1, A_2, \ldots, A_m\}$, each attribute, categorical or numerical, $A_i \in R$ is associated with a definition domain D_i. Each tuple $t \in R$ ($t \in A_1 \times A_2 \times \ldots A_m$) is defined by the values it takes on the m attributes, $t.A_i$ denotes the value taken by t on attribut A_i. A *property* is defined as a couple made of an attribute and a value, denoted (A_i, p). A tuple t is said to possess a property (A_i, p) if $t.A_i = p$ and a set of properties s if $\forall (A_i, p) \in s$, $t.A_i = p$. Reciprocally, the properties of t are all the couples $(A_i, t.A_i)$ for $i = 1 \ldots m$.

Relevance of individual attribute values The importance of a property (A_i, p) wrt. a partial concept extension $\mathcal{E}_C$ serves to determine whether p is representative of the values taken by A_i in $\mathcal{E}_C$: the more p is shared, for attribute A_i, by representative elements of the concept, the more important it is. It is thus measured as

$$\delta_i(p) = \frac{|\{x \in \mathcal{E}_C / x.A_i = p\}|}{|\mathcal{E}_C|}. \tag{1}$$

The strict comparison imposed by the binary matching condition $x.A_i = p$ can be replaced by a relaxed comparison, considering the condition $sim_i\,(p,\, x.A_i) \geq \eta_i$, where sim_i is an appropriate similarity measure on the domain of attribute A_i and η_i a similarity threshold.

Representativity of attribute value sets Once the properties have been individually evaluated, the importance of *sets* of such properties is quantified. Whereas an individual value is considered important if it frequently appears in the partial concept extent, the importance of a subset of values depends on its size and whether it appears at least once in the partial concept extent. The fuzzy measure μ thus serves to quantify the extent to which the subset of attribute values matches one of the representative data points in $\mathcal{E}_C$. The μ score is maximal if the assessed set of properties exactly corresponds to one of the representative data points in $\mathcal{E}_C$. Denoting $s = \{(A_i, p_i), i = 1 \ldots |s|\}$ such a set of properties, the binary approach defines μ as:

$$\mu(s) = \max_{x \in \mathcal{E}_C} \frac{1}{m} |\{(A_i, p_i) \in s / x.A_i = p_i\}|, \tag{2}$$

where m denotes the number of attributes. As for the δ_i functions, the strict comparison $t.A_i = p_i$ can be relaxed as $sim_i\,(p,\, x.A_i) \geq \eta_i$.

Aggregation The final step combines the evaluations of atomic properties and set of properties. CHOCOLATE uses the Choquet integral to perform this aggregation of the δ_i and μ evaluations. It especially takes into account, when comparing a set of properties wrt. representative data points (using the μ function), if these evaluated properties are individually specific to one data point or shared by many. This makes it possible to differentiate between a set of properties possessed by only a single representative data point and another set of properties of the same size but shared by several representative data points.

The candidate sets of properties are defined as the set of the most promising ones, according to their individual δ values: let H_j denote the subset of the j properties that best match the representative data points from $\mathcal{E}_C$, i.e. the j properties with maximal δ values. Let $\kappa_j(x)$ denote the jth value among the δ_i, i.e. the matching degree of the jth most representative property possessed by x wrt. concept C. CHOCOLATE defines the membership degree of a data point x to concept C based on the set of representative examples $\mathcal{E}_C$ as

$$\mathcal{S}_C(x) = \sum_{j=1}^{m} (\mu(H_j) - \mu(H_{j-1}))\, \kappa_j(x). \tag{3}$$

3 Proposed DCQ Method for Querying by Example Disjunctive Concepts

This section explains how to build a fuzzy QBE approach based on the CHOCOLATE strategy , so as to extend a classical RDBMS with disjunctive fuzzy concept querying capabilities. An implementation of DCQ in a RDBMS (PostgreSQL in our case) is then presented, describing in turn the two stored procedures: the first one infers the membership function of a concept from a set of user-selected examples, implementing the CHOCOLATE strategy. The second one can be integrated into a selection statement to retrieve the tuples that best satisfy the concept.

3.1 Underlying Principles of DCQ

DCQ initiates the definition of a concept with the few user-given representative tuples. Identifying these representative tuples directly from the concerned relation extension is not conceivable in terms of system usability. DCQ is thus based on a user interface where some selected tuples are suggested to the user who just has to select those of interest. An illustration of such a graphical interface is provided Figure 3 in the Proof-of-Concept section. From the whole relation extension, choosing which tuples to suggest is a research topic in itself that is outside the scope of this paper.

Once a few tuples of interest have been selected by the user, he/she can then ask the system to infer a membership function that best covers the properties they possess. The user only has to name his/her concept, for instance *idealHousing* for the example mentioned in the introduction. A query is then executed on

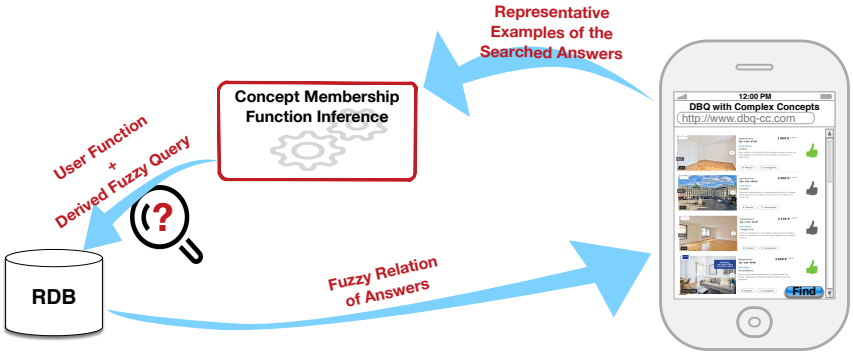

Fig. 1: Fuzzy QBE system handling complex search concepts

the DBMS to retrieve the tuples that best satisfy the concept. Tuples are then returned to the user in a decreasing order of their satisfaction degree wrt. the search concept. Figure 1 sketches the components of the query architecture.

3.2 Concept Definition Inference

Once the few representative tuples of the fuzzy concept to define have been selected by the user, the procedure that builds it membership function can be triggered. This procedure has the following prototype:

```
CREATE PROCEDURE infer_concept(label TEXT, rel_name TEXT, rep_ex HSTORE []);
```

where the parameters are respectively the label given to the concept to infer, the name of the considered relation, that can even be a view to the result of a join query and an array of dictionaries where each dictionary represents a user-selected representative example.

Example 1. The following example shows how to call the *infer_concept* procedure that builds the membership function of the concept *idealHousing* from the user-selected representative examples introduced in Table 2[1].

```
CALL infer_concept('idealHousing', 'housing_ads', {
    ' "loan"=>400, "surf"=>30, "nbR"=>2, "loc"=>"center", "outSurf"=>0 ',
    ' "loan"=>450, "surf"=>40, "nbR"=>3, "loc"=>"center", "outSurf"=>10 ',
    ' "loan"=>400, "surf"=>80, "nbR"=>4, "loc"=>"suburb", "outSurf"=>10 ',
    ' "loan"=>400, "surf"=>100, "nbR"=>5, "loc"=>"outskirts", "outSurf"=>500 '});
```

The procedure infer_concept creates a user function that implements the inferred membership function. This user function, that can then be integrated into the selection statement of a query, has the following prototype:

[1] The *hstore* module of Postgresql is used to store key/value pairs using the syntax "key"=>"value".

Input: $t, \mathcal{E}_C$ $\triangleright\ t \in R,\ \mathcal{E}_C \subset R$
Output: $\mu_C(t)$ $\triangleright\ \in [0,1]$
Function `concept_name()`:
  $\mu_C(t) \longleftarrow 0$; $ds \longleftarrow []$;
  foreach $(A_i, p_i) \in t$ **do**
    $ds[i] \longleftarrow \delta_i(p_i)$;
  end
  $sortDesc(ds)$;
  foreach $j \in 1..m$ **do**
    $H_j \longleftarrow ds[1:j]$;
    $H_{j-1} \longleftarrow ds[1:j-1]$;
    $\mu(H_j) \longleftarrow \max_{t' \in \mathcal{E}_C} \frac{1}{m} \times |\{(A_i, p_i) \in H_j, t'.A_i = p_i\}|$;
    $\mu(H_{j-1}) \longleftarrow \max_{t' \in \mathcal{E}_C} \frac{1}{m} \times |\{(A_i, p_i) \in H_{j-1}, t'.A_i = p_i\}|$;
    $\mu_C(t) \longleftarrow \mu_C(t) + (\mu(H_j) - \mu(H_{j-1})) \times ds[j]$;
  end
  return $\mu_C(t)$;
End Function

Algorithm 1: Pseudo-code of the proposed function to implement the CHOCOLATE strategy

CREATE FUNCTION *concept_name*() RETURNS DECIMAL ;

where *concept_name* is the name given to the concept by the user, *idealHousing* for the considered example.

The function *concept_name* implements the CHOCOLATE strategy [9], reminded in Section 2.3. The implementation of this function, that returns the satisfaction degree of a tuple to evaluate wrt. the considered search concept, is given in Alg. 1 in pseudo-code (its implementation in the pl/python language contains technical tricky aspects that do not help for its understanding).

3.3 Querying with Disjunctive Fuzzy Concepts

The user functions representing the inferred disjunctive fuzzy concepts can then be embedded into SQL queries in the following way:

```
SELECT *, get_mu() as mu
FROM rel_name
WHERE concept_name() > 0;
```

As underlined in [10], relying on user functions to implement fuzzy predicates and their direct use in selection statements may lead to a significant computation cost overhead. Query engines do not leverage existing indexes when a selection statement involves a user-function, and therefore perform a sequential search on the whole relation to identify tuples satisfying the predicate. A way to avoid this extra-cost is to perform a so-called *derivation* of the fuzzy query so as to translate it into a Boolean query whose execution can be natively optimised by the RDBMS query engine. A derived query aims at retrieving the tuples that may somewhat satisfy the initial fuzzy query. The membership degree within

the concerned fuzzy predicate is then computed only for these tuples. As shown in the prototype query given hereafter, the selection clause now contains the Boolean condition, whose execution can be optimized by the query engine, and the user function to compute the concept membership degree is involved in the projection statement only:

```
SELECT *, concept_name() as mu
FROM rel_name
WHERE (derivedConditionOn_A1) OR
    (derivedConditionOn_A2) OR
    ...
    (derivedConditionOn_Am);
```

where $\text{derivedConditionOn_}A_i$ is the derived Boolean condition regarding attribute A_i. The derived condition aims at selecting the tuples whose value on attribute A_i is shared with at least one of the representative examples.

To obtain a derived condition, two aspects have to be taken into account, the nature of A_i (numerical vs. categorical), and the type of comparison used (binary strict one or similarity-based relaxed one).

When a strict comparison is used then the derived condition is of the form:

$(A_i \text{ IN } V)$

where $V = \{x.A_i, x \in \mathcal{E}_C\}$, C being the involved concept and $\mathcal{E}_C$ its partial extent. When A_i is numerical and a relaxed comparison is used, the derived condition is of the form:

$(A_i \text{ BETWEEN } v^- \text{ AND } v^+)$

where $v^- = \inf_{v \in D_i, t \in \mathcal{E}_C} \text{sim}_i(t.A_i, v) \geq \eta_i$
and $v^+ = \sup_{v \in D_i, t \in \mathcal{E}_C} \text{sim}_i(t.A_i, v) \geq \eta_i$.

Example 2. The user query looking for *idealHousing* is thus translated into the following fuzzy query:

```
SELECT *, get_mu() as mu
FROM housing
WHERE idealHousing() > 0;
```

whose derived version is:

```
SELECT *, idealHousing() as mu
FROM rel_name
WHERE (loan IN (400, 450) ) OR
    (surf BETWEEN 30 AND 100) OR
    (nbR IN (2, 3, 4, 5) ) OR
    (loc IN ('center', 'suburb', 'outskirts') );
```

Considering, for a sake of clarity, a strict equality comparison between values of the tuple to evaluate and those of the representative tuples used to infer the concept membership function, Table 3 gives a very short excerpt of the fuzzy relation returned by the previous query. Applied on a toy example, the extract of the fuzzy relation given in Table 3 shows the interesting behaviour of the CHOCOLATE approach to quantify the satisfaction of a tuple regarding a

Table 3: Excerpt of the returned fuzzy relation containing the retrieved *ideal-Housing* instances associated with their membership degrees (column *mu*)

loan	*surf*	*nbR*	*loc*	*outSurf*	*mu*
...	...	...	...	...	...
500	25	2	historical center	0	0.15
500	30	2	center	0	0.25
400	40	3	center	0	0.3
400	80	4	suburb	40	0.3
650	80	4	outskirts	1000	0.1
...	...	...	...	...	...

complex disjunctive concept. One can indeed remark that the second and third tuple match the concept (with respective score of 0.25 and 0.3) for different reasons. For the second tuple, it is due to the fact that it shares 4 properties with one of the representative tuples (Tab. 2) and one of the them (*loc* = *center*) is possessed by two representative examples of the concept. For the third evaluated tuple, despite the fact that it shares 3 properties (*surf*, *nbR* and *loc*) with one of the representative example, it also possesses a property (*loan* = *400*) that seems to be important as it is possessed by 3 of the 4 representative examples. The Choquet-based approach thus captures and aggregates the importance of properties taken individually and combined conjunctively.

4 Proof-Of-Concept of DCQ

A Proof-Of-Concept (POC) has been implemented[2] for a database containing 1,200 descriptions of accommodations to rent, each accommodation being described on 11 attributes. This POC aims at comparing the proposed strategy with two classical distance-based approaches.

4.1 Selection of the Suggested Tuples

The first question to address is to identify the tuples from the database to be suggested to the user. The idea is to cover as much as possible the diversity of the available data. It has been empirically decided to suggest up to 40 tuples. To do so a clustering algorithm based on affinity propagation has been employed to build 40 clusters whose centroids constitute the suggested examples. Figure 2 illustrates the use of this algorithm on two attributes (distance to center and loan) to show how centroids are identified.

4.2 Answers Retrieval

The user then peruses the suggested tuples to select those illustrating what he/she is looking for. Once five tuples have been selected (Fig. 3) to represent

[2] It is available at the url http://51.210.243.246:8081/fuzzy_queries/

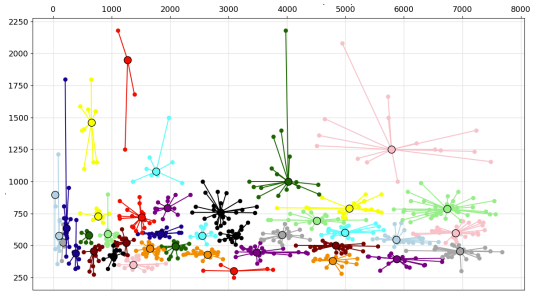

Fig. 2: Illustration of the clustering-based identification of the tuples to suggest

Fig. 3: QBE user interface showing the five tuples selected to initiate the search

his/her concept of ideal housing, the infer_concept function is called to infer the membership function of the search concept and to store it so that it can be latter reused.

A derived fuzzy query involving the user concept, as described in the previous section, is then submitted to retrieve the tuples that best satisfy the user requirements.

5 Conclusion and Perspectives

This paper introduces a technical solution to allow end users query relational data using complex search conditions materialized by fuzzy disjunctive concepts and their associated membership function. Functionalities have been first added to a classical RBMS so that it handles selection statements involving the satisfaction of such complex concepts. To ease the definition of complex search concepts, a query by example principle is envisaged. End users just have to select a few tuples that represent the diversity of what they are looking for. The approach then learns the important atomic properties as well as combinations of them possessed by at least one user-given representative example. A prototype has been developed to compare the proposed strategy to concept learning with two others based on the distance wrt.the closest or all the representative examples. Ongoing works aim at collecting and analysing the results of the ongoing experimentation with end users to have a subjective assessment of the proposed system on a real

use case. An implementation of the approach to movie recommendation is also in progress and may constitute an interesting solution to the cold-start problem.

References

1. Anisfeld, M.: Disjunctive concepts? The Journal of general psychology 78(2), 223–228 (1968)
2. Bělohlávek, R., Vychodil, V.: What is a fuzzy concept lattice? In: Proc. of the 3rd Int. Conf. on Concept Lattices and Their Applications, CLA05. pp. 34–45 (2005)
3. De Calmès, M., Dubois, D., Hullermeier, E., Prade, H., Sedes, F.: Flexibility and fuzzy case-based evaluation in querying: An illustration in an experimental setting. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 11(01), 43–66 (sep 2003)
4. Iba, G.A.: Learning disjunctive concepts from examples (1979)
5. Lesot, M.J., Rifqi, M., Bouchon-Meunier, B.: Fuzzy prototypes: From a cognitive view to a machine learning principle. In: Bustince, H., Herrera, F., Montero, J. (eds.) Fuzzy Sets and Their Extensions: Representation, Aggregation and Models, pp. 431–452. Springer (2008)
6. Moreau, A., Pivert, O., Smits, G.: Fuzzy query by example. In: Proc. of ACM Symposium on Applied Computing, SAC'18. pp. 688–695 (2018)
7. Murray, K.S.: Multiple convergence: An approach to disjunctive concept acquisition. In: Proc. of the Int. Joint Conf. on Artificial Intelligence, IJCAI. pp. 297–300 (1987)
8. Rifqi, M.: Constructing prototypes from large databases. In: Proc. of the Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'96). pp. 300–306 (1996)
9. Smits, G., Yager, R., Lesot, M.J., Pivert, O.: Concept membership modeling using a choquet integral. In: Proc. of the Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'20). pp. 359–372 (2020)
10. Smits, G., Pivert, O., Girault, T.: Reqflex: fuzzy queries for everyone. Proceedings of the VLDB Endowment 6(12), 1206–1209 (2013)
11. Wille, R.: Restructuring lattice theory: an approach based on hierarchies of concepts. In: Ordered sets, pp. 445–470. Springer (1982)
12. Zadrozny, S., Kacprzyk, J., Wysocki, M.: On a novice-user-focused approach to flexible querying: The case of initially unavailable explicit user preferences. Proc. of the 2010 10th Int. Conf. on Intelligent Systems Design and Applications, ISDA'10 pp. 696–701 (2010)
13. Zloof, M.M.: Query-by-example: A data base language. IBM Syst. J. 16(4), 324–343 (1977), https://doi.org/10.1147/sj.164.0324