Evaluating Logic-Based Scoring Functions on Uncertain Relational Data

Sebastian Lehrack and Sascha Saretz

Brandenburg University of Technology Cottbus,
Institute of Computer Science,
Postfach 10 13 44, D-03013 Cottbus, Germany,
{slehrack, ssaretz}@informatik.tu-cottbus.de

Abstract. Nowadays, for many retrieval scenarios a strict query evaluation just returning a Boolean truth value is not sufficient anymore. We often rather need the support of a gradual query fulfilment expressed by a score value out of the interval $[0, 1]$. *ProQua* is a new probabilistic database system which combines such information retrieval concepts with traditional database technologies. In contrast to other state-of-the-art probabilistic database systems ProQua facilitates logic-based similarity conditions *within* its SQL-like query language by a generic similarity operator. In this work we formalise logic-based scoring functions as the underlying concept of the supported similarity conditions and introduce respective evaluation techniques implemented by relational query plans. Additionally, we report on their experimental verification on a probabilistic TPC-H database.

Categories and Subject Descriptors: H.2.3 [**Database Management**]: Languages

Keywords: logic-based scoring functions, probabilistic databases, ProQua, similarity conditions

1. INTRODUCTION

A classical relational database system evaluates a query against a database tuple either to the truth value *true* on match or to the truth value *false* on mismatch. Nowadays, there are many retrieval scenarios where such a strict evaluation is not adequate anymore. It is known that Boolean truth values cannot satisfy user expectations about vague and uncertain conditions [Agrawal et al. 2003]. Thus, there is a need for incorporating the concepts of impreciseness and proximity into the a database query language.

Our new probabilistic database system *ProQua*[1,2] is designed as a combination of such information retrieval concepts and database technologies. Recently, leading database researchers emphasised this combination as an important and challenging research field in the last Claremont report[3] [Agrawal et al. 2008].

An established technique for handling vagueness is the usage of *similarity predicates* like '*age around 300*' or '*finding site is close to the temple of Artemis*' within a *logic-based query language*. This type of query language supports *complex* similarity conditions formulated by similarity predicates and the logical operators *AND* ($\wedge$), *OR* ($\vee$) and *NOT* ($\neg$). Data tuples fulfill similarity conditions to certain degrees which can be represented by *score values* out of the interval $[0, 1]$. By means of score values

[1]ProQua stands for <u>pro</u>babilistic and <u>qua</u>ntum logic-based database system.
[2]Online demo: http://dbis.informatik.tu-cottbus.de/ProQua/
[3]The *Database Research Self-Assessment Meeting* takes place every five years.

Sebastian Lehrack was supported by the German Research Foundation grants SCHM 1208/11-1 and SCHM 1208/11-2.
Copyright©2012 Permission to copy without fee all or part of the material printed in JIDM is granted provided that the copies are not made or distributed for commercial advantage, and that notice is given that copying is by permission of the Sociedade Brasileira de Computação.

Arte				
tid	aid	type	sond	age
t_1	art1	vase fragment	3	300
t_2	art2	spear head	10	500
t_3	art3	vase fragment	4	300

ArteExp					
tid	expert	rep	aid	culture	conf
t_4	Peter	B	art1	roman	0.3
t_5	Peter	B	art1	greek	0.4
t_6	Cathy	C	art1	roman	0.4
t_7	John	A	art2	egyptian	0.6

ArteMat					
tid	method	year	aid	culture	conf
t_8	XRF	1997	art1	roman	0.3
t_9	XRF	1997	art1	greek	0.3
t_{10}	ICS-MS	2008	art2	punic	0.8
t_{11}	XRF	2010	art2	egyptian	0.5

Fig. 1. Tables *Arte*, *ArteExp* and *ArteMat* of the running scenario

we are able to compute a ranking of all data objects giving the desired query result.

Besides similarity conditions known from information retrieval (IR) *probabilistic databases* have been established as a new type of database systems [Suciu et al. 2011]. In such a database a tuple belongs to a data table or a query answer with a specific occurrence probability expressing the uncertainty about the given data or the confidence in the answer, respectively. In other words, in a probabilistic database several possible database instances, also called *possible worlds*, are managed and queried simultaneously. Thereby, the “real world” is assumed to be unknown. To handle this uncertainty we define a probability measure over the set of all possible worlds.

To the best of our knowledge, ProQua is the first and only probabilistic database system which offers complex logic-based similarity conditions on uncertain relational data as an integrated query language concept. In this work we will formalise logic-based similarity conditions by *logic-based scoring functions*. In particular, we syntactically and semantically introduce logic-based scoring functions and develop evaluation techniques implemented by relational query plans. Thus, the main contributions of this article are:

(1) the formalisation of the syntax and the semantics of logic-based scoring functions (Sec. (3)),
(2) the definition of construction rules for logic-based scoring functions based on the ProQua query language QSQL2 (Sec. (4)),
(3) the development of evaluation techniques for logic-based scoring functions in form of relational query plans (Sec. (5)),
(4) the implementation of our methods into the probabilistic database system ProQua and
(5) the experimental verification against a probabilistic TPC-H database.

Running scenario: We demonstrate our basic ideas by means of a running example. It is motivated by the redevelopment of the CISAR project[4] [Henze et al. 2007] which is a web-based information system for archaeology and building history. ProQua technologies will be widely used in its successor project *OpenInfRA*.

[4]http://www.dainst.org/en/project/cisar/

```
select aid, type, culture
from     (  select aid, culture
            from ArteExp
            where rep ~ 'A'
           union
            select aid, culture
            from ArteMat
            where year ~ 2012
         ) origin
      inner join
         ( select *
           from Arte
           where age ~ 300
         ) prop
      on ( origin.aid = prop.aid )
where age ~ 300
```

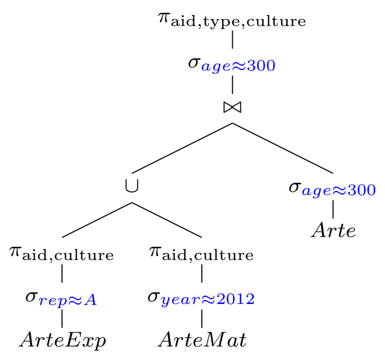

Fig. 2. Running scenario query as QSQL2 query (left) and abstract tree (right)

In our simplified example scenario we work with the deterministic table *Artefacts* (*Arte*) and the two probabilistic tables *Artefacts classified by Experts*[5] (*ArteExp*) and *Artefacts classified by Material* (*ArteMat*), see Fig. (1). In the table *Arte* we store information about several artefacts which were found during an archaeological excavation. Thereby, the sondage number (attribute *sond*) of an artefact describes its geographical area of finding .

Moreover, several experts gave an expertise about the originating culture (attribute *culture*) for each artefact, see table *ArteExp*. Each expert is also rated by her/his scientific reputation. We express this reputation by a mark from $\{A, .., E\}$ contained in the attribute *rep*. Thereby, the mark A represents the best possible rating and the mark E stands for the lowest possible rating. All expert estimations are additionally annotated by a confidence value (column *conf*) embodying the probability that the considered artefact belongs to the specified culture.

Besides these subjective expert valuations we also take more objective methods into account. These *archaeometrical methods* (e.g. XRF and ICS-MS[6]) rely on material analysis which were conducted in the year stored in the identically named attribute. In combination with the artefact finding site and the artefact age, the material composition gives us a valuable hint for the desired culture specification also quantified by a confidence value (column *conf*).

Based on the introduced data tables we run following query: *Determine all artefacts with their possible cultural origins whereby their ages should be around* 300 *years. Additionally, we require that the reputation of an expert is as high as possible and a considered material analysis is as new as possible.* To answer this query we use ProQua in conjunction with its query language QSQL2 [Lehrack et al. 2012], see Fig. (2). Please be aware that the similarity predicate $age \approx 300$ appears twice in the given query. This situation can easily occur when an automatic query generation took place or views were used.

ProQua: Next, we briefly elucidate the main idea behind ProQua by sketching the interrelations among the core concepts of the ProQua system, see Fig. (3). The starting point is a QSQL2 query as given in Fig. (2). In the first step we map the different syntactical components of a QSQL2 query to (i) a logic-based scoring function ψ and (ii) an algebra query Q applied on probabilistic data. The first part of this mapping will be described in Sec. (4). Both query types are grounded in their own semantical model. On the one hand a logic-based scoring function is interpreted by a probabilistic view of a vector space retrieval model [Lehrack and Schmitt 2011; Schmitt 2008], see Sec. (3). On the other hand, we apply the well-known possible-world-semantics for handling an algebra query on probabilistic data. Respecting these semantical models the outcome of a logic-based scoring function is determined as result of a normalised similarity domain calculus query (see Sec. (5)). Practically, a

[5]Please note that the columns *tid* and *conf* do not belong to the actual data tables.
[6]XRF and ICP-MS stand for the *x-ray fluorescence* and the *inductively coupled plasma mass spectrometry* method.

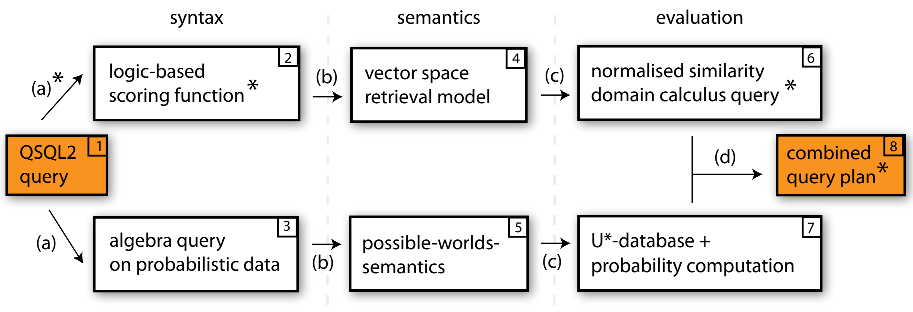
syntax
semantics
evaluation
(a)*
logic-based scoring function *
2
(b)
vector space retrieval model
4
(c)
normalised similarity domain calculus query *
6
QSQL2 query
1
(d)
combined query plan *
8
(a)
algebra query on probabilistic data
3
(b)
possible-worlds-semantics
5
(c)
U*-database + probability computation
7

Fig. 3. ProQua model: (a) syntax mapping, (b) query semantics, (c) evaluation techniques and (d) ranking semantics

logic-based scoring function ψ is evaluated by a relational query plan qp(ψ) which will be developed in Sec. (5). In contrast, the evaluation of an algebra query Q is performed by a query plan qp(Q). It relies on a novel representation system for probabilistic database called *U*-database*. By employing the ranking semantics of *expected scores* (see [Li et al. 2011; Ilyas and Soliman 2011]) we finally generate a combined query plan which produces the desired query result (see Sec. (5.4)).

The main contributions of this work are marked by a star ($*$) in Fig. (3) and will be presented in Sec. (3), (4) and (5). In the next section we will define four query classes based on the expressiveness of a query language. Conducted experiments will be presented in Sec. (6). Finally, we will finish our work by a discussion of related approaches in Sec. (7) and by our conclusions in Sec. (8). Due to the limited space, we defer all proofs and a comprehensive description of all queries investigated by our experiments (see Sec. (6)) to an extended version of this article [Lehrack 2012b].

2. QUERY TYPES

In the following section we present a classification of different query types which help us to clarify and compare our query language QSQL2 against existing approaches (see Sec. (7)). For setting up our classification we identify two independent criteria, namely (i) the expressiveness of the supported conditions and (ii) the nature of the underlying relational data basis. More concretely, we indicate (i) the capability of incorporating the concepts of impreciseness and vagueness in terms of similarity conditions (i.e. classical Boolean conditions (BC) vs. similarity conditions (SC)) and (ii) the capability of expressing different possible database states derived from tuple and attribute uncertainty (i.e. classical certain data bases (CD) vs. uncertain data bases (UD)). By applying these criteria orthogonally we obtain the following four query classes. They all are exemplified by a characteristic query taken from our running scenario.

(1) **Boolean conditions on certain data (BConCD)**: The class BConCD contains queries formed by Boolean conditions (i.e. the result is given by either the truth value true or false) on deterministic relational data. These queries are processed by traditional relational query languages as SQL [Date 1997]. Referring to our scenario a typical query of BConCD is formulated as *"Determine all artefacts which have a sondage number between 7 and 12 or an age between 250 and 350 years"*. This kind of query produces a homogeneous result set of artefacts matching the defined condition.

(2) **Similarity conditions on certain data (SConCD)**: The class SConCD stands for queries which support vagueness and impreciseness by similarity conditions. The result of such a query is given by a *list of tuples* ordered by score values from the interval $[0, 1]$. These score values express the degree of query fulfilment. For instance, a SConCD-query is given by *"Determine all artefacts whereby their sondage numbers should be around 10* or *their ages should be around 300 years"*. We will develop the query plan qp(ψ) for evaluating SConCD-query in Sec. (5).

(3) **Boolean conditions on uncertain data (BConUD)**: The queries of the class BConUD are typical for probabilistic databases. As an example query we give *"Determine all artefacts which*

are probably created by roman people". Once again the resulting tuples are ranked. But in this case occurrence probabilities are used for the final ranking instead of score values.

(4) **Similarity conditions on uncertain data (`SConUD`)**: If we combine both criteria, then we achieve a query class with an expanded expressiveness. A `SConUD`-query is formulated as *"Determine all artefacts which are probable created by romans and have an age around 300 years"*. The final ranking is now determined by a given ranking semantics, e.g. expected scores or expected ranks [Li et al. 2011]. If we presume the expected scores semantics, then we can employ a relational query plan as presented in Sec. (5.4) to compute the final result of a `SConUD`-query.

3. SYNTAX AND SEMANTICS OF LOGIC-BASED SCORING FUNCTIONS

In the following, we specify the syntax and the underlying semantics of a logic-based scoring function. Thus, we give the theoretical foundation for our practical evaluation techniques outlined in the next sections. At first, we define the syntax of a logic-based scoring function as a first-order logical formula extended by similarity predicates.

Definition 3.1 *Logic-based scoring function (syntax).* Let $R_1, \dots, R_k$ be relations over the attributes $A_1, \dots, A_r$. Then, the syntax of a logic-based scoring function $\psi : \mathrm{Dom}_\psi \rightarrow [0,1]$ is defined by the grammar

$$\psi ::= A\theta c \mid A_1\theta A_2 \mid R(A_1, \dots, A_l) \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi \mid \exists A : \psi \mid \forall A : \psi$$

whereby $R(A_1, \dots, A_l)$ represents a relational predicate, c stands for a constant and θ forms a Boolean or a similarity predicate (i.e. $\theta \in \{=, <, >, \approx, \dots\}$). We also call ψ a *condition* applied on a tuple t from the domain $\mathrm{Dom}_\psi = \mathrm{Dom}(A_1) \times \dots \times \mathrm{Dom}(A_r)$. If ψ involves similarity predicates as $A \approx c$ or $A_1 \approx A_2$, then we denote ψ as *similarity condition*.

The outcome of a logic-based scoring function is defined by an underlying semantical model, e.g. Boolean logic[7], fuzzy logic or probability theory. Roughly speaking, a semantical model has to give the interpretations of all Boolean and similarity predicates ($A\theta c, A_1\theta A_2$) and the evaluation rules for the logical operators/quantifiers ($\wedge, \vee, \neg, \exists A, \forall A$).

3.1 Geometric retrieval model as underlying semantics

In order to set a semantical model for a logic-based scoring function we utilise a probabilistic interpretation of a geometric retrieval model which is based on the squared cosine similarity measure [Schmitt 2008]. To be more precise, the core idea of the underlying retrieval model is employing a mathematical framework, also known from quantum mechanics and quantum logic, for database query processing. In this article we just lay out the main principles by means of a brief informal survey. For a mathematical description we refer to [Schmitt 2008], [Schmitt et al. 2009] and [Lehrack and Schmitt 2011].

Following, we explain the general correspondences between query processing concepts and the geometric retrieval model as given in Fig. (4). In principle, the retrieval model relies on the theory of vector spaces and determines the evaluation of a single tuple against a given similarity condition. Our short discourse starts by considering a vector space being the domain of all tuples. The direction of a tuple vector of length one encodes all attribute values of the queried tuple. Moreover, a complex similarity condition itself is embodied by a specific vector subspace of the domain vector space denoted as *condition space*.

The evaluation result is then defined by the minimal angle between tuple vector and condition space. The squared cosine of such an angle is a value out of the interval $[0,1]$ and can therefore be interpreted

[7]We recall that Boolean logic does not support similarity predicates.

query processing	vs.	**vector space retrieval model**
value domain	$\leftrightarrow$	vector space
tuple to be queried	$\leftrightarrow$	tuple vector
condition	$\leftrightarrow$	vector subspace
score value	$\leftrightarrow$	squared cosine of the angle between tuple vector and vector subspace

Fig. 4. Correspondences between query processing and the geometric retrieval model

as a similarity measure. That means, if the tuple vector belongs to the condition space (i.e. an angle of 0°), the condition outcome equals to a complete match (i.e. a score value of 1). Contrarily, a right angle of 90° between tuple vector and condition space describes a complete mismatch (i.e. a score value of 0).

In [Lehrack and Schmitt 2011] we developed a probabilistic view of this geometric retrieval model. It provides the required interpretation and evaluation rules for a logic-based scoring function as specified in Def. (3.1). Precisely, we proved that we can interpret atomic similarity conditions as independent probability measures combined by a product probability space. In consequence, we easily apply standard probability aggregation rules, if all involved subconditions represent independent events.

Definition 3.2 *Logic-based scoring function (semantics)*. Let ψ be a logic-based scoring function. Then, the score value $\psi(t)$ for a tuple $t \in \mathrm{Dom}_\psi$ is recursively determined by

$$\begin{aligned}
\psi \equiv A\theta c \ &: \ \psi(t) := \mathrm{psf}_\theta(t[A], c), && (1)\\
\psi \equiv A_1\theta A_2 \ &: \ \psi(t) := \mathrm{psf}_\theta(t[A_1], t[A_2]), && (2)\\
\psi \equiv R(A_1,\dots,A_l) \ &: \ \psi(t) := 1 \ \text{ if } t[A_1,\dots,A_l] \in R, \ \ 0 \ \text{ otherwise} && (3)\\
\psi \equiv \psi_1 \wedge \psi_2 \ &: \ \psi(t) := \psi_1(t) * \psi_2(t), && (4)\\
\psi \equiv \psi_1 \vee \psi_2 \ &: \ \psi(t) := \psi_1(t) + \psi_2(t) - (\psi_1 \wedge \psi_2)(t), && (5)\\
\psi \equiv \neg\psi_1 \ &: \ \psi(t) := (1 - \psi_1(t)), && (6)\\
\psi \equiv \exists A : \psi_1 \ &: \ \psi(t) := \max_{\substack{\hat{t}\in \mathrm{Dom}_\psi, \hat{t}[\mathcal{B}]=t[\mathcal{B}],\\ \mathcal{B}=\mathrm{attr}(t)\setminus\{A\}}} \{\psi_1(\hat{t})\} && (7)\\
\psi \equiv \forall A : \psi_1 \ &: \ \psi(t) := \neg(\exists A : \neg\psi_1(t)) && (8)
\end{aligned}$$

whereby the auxiliary function $\mathrm{attr}(t)$ returns the attributes of t and $\mathrm{psf}_\theta(\cdot,\cdot)$ represents a predicate scoring function for evaluating the predicate θ. We require that an attribute is queried by at most one constant in all similarity predicates, i.e. $((A \approx c_1$ and $A \approx c_2$ are involved in $\psi) \Rightarrow c_1 = c_2)$ and $((A_1 \approx A_2$ and $A_1 \approx A_3$ are involved in $\psi) \Rightarrow A_2 = A_3)$.

The last requirement of Def. (3.2) assures the independence of the involved predicates, see [Lehrack and Schmitt 2011]. Please note that the predicate scoring functions $\mathrm{psf}_\theta(\cdot,\cdot)$ are not pre-defined. In general, any predicate scoring function returning similarity values which can be computed by the scalar product of normalised vectors is supported. That is, the similarity values of $\mathrm{psf}_\theta(\cdot,\cdot)$ must form a semi-positive definite correlation matrix. For instance, we can adapt the Levenshtein distance function for strings and equi-distance functions for bounded numeric domains. We give the logic-based scoring function ψ_e derived from our running scenario as an example in the next section.

4. CONSTRUCTING LOGIC-BASED SCORING FUNCTIONS

In this section we discuss the construction of a logic-based scoring function ψ based on a given QSQL2 query. Thereby, we restrict our considerations to QSQL2 syntax components embodying the core functionality of QSQL2. The presented components are equivalent to the standard relational

rule	QSQL2 query components	scoring function
R1	`R`	$\psi := R(A_1, \dots, A_m)$
R2	`select B1,..,Bm` `from E1` `where sc`	$\psi := \exists\, C_1, \dots, C_r : (\psi_{E_1} \wedge \texttt{sc})$, whereby $\text{attr}(\texttt{E1}) = \{B_1, \dots, B_m, C_1, \dots, C_r\}$
R3	`select B1,..,Bm` `from ( E1 inner join E2` `on jc ) where sc`	$\psi := \exists\, C_1, \dots, C_r : ((\psi_{E_1} \wedge \psi_{E_2} \wedge \texttt{jc}) \wedge \texttt{sc})$, whereby $\text{attr}(\texttt{E1}) \cup \text{attr}(\texttt{E2}) = \{B_1, \dots, B_m, C_1, \dots, C_r\}$
R4	`E1 intersect E2`	$\psi := \psi_{E_1} \wedge (\psi_{E_2})_{<\texttt{B}_{E_2,1},\dots,\texttt{B}_{E_2,m} \mid \texttt{B}_{E_1,1},\dots,\texttt{B}_{E_1,m}>}$
R5	`E1 union E2`	$\psi := \psi_{E_1} \vee (\psi_{E_2})_{<\texttt{B}_{E_2,1},\dots,\texttt{B}_{E_2,m} \mid \texttt{B}_{E_1,1},\dots,\texttt{B}_{E_1,m}>}$

Fig. 5. Syntax mapping rules for constructing a logic-based scoring function: `R` represents a relation with the attributes $A_1, \dots, A_m$, `E` stands for an arbitrary QSQL2 expression, `B` symbols an attribute, `sc` embodies a similarity condition and `jc` describes a join condition.

operators[8]: projection, selection, join, intersection and union. In order to generate a logic-based scoring function we recursively apply the syntax mapping rules as given in Fig. (5):

(R1) A relation R used in a QSQL2 query corresponds to a relation predicate $R(A_1, \dots, A_l)$ in ψ whereby $A_1, \dots, A_m$ are *unique* variables derived from the attributes of R. To achieve unique variable labels we distinguish the attributes of a relation occurring more than once in the considered query by an index. For example, if we consider a self-join $R(A_1, A_2) \bowtie R(A_1, A_2)$, then we build the two relation predicates $R(A_{1,1}, A_{2,1})$ and $R(A_{1,2}, A_{2,2})$.

(R2) A typical select-from-where block (SFW block) consists of three parts: (1) a list of *output attributes* `B1`, …, `Bm` given in the `select`-clause, (2) a (recursive) *data basis* `E1` formulated in the `from`-clause and (3) a complex *similarity condition* `sc` declared in the `where`-clause:
  (1) **output attributes:** We convert the list of output attributes `B1`, …, `Bm` into an existential quantifier using exactly the attributes which are *not* occurring in the output list, i.e. if `B1`, …, `Bm`, `C1`, …, `Cr` are the attributes of the data basis `E1`, then we choose the attributes `C1`, …, `Cr` for building the existential quantifier. This mapping is motivated by the correspondences between the algebra operator projection and the existential quantifier of the relational domain calculus [Maier 1983].
  (2) **data basis:** The (recursive) data basis is given as an arbitrary QSQL2 expression `E1`. Its corresponding function ψ_{E_1} is also integrated recursively.
  (3) **similarity condition:** The similarity condition `sc` is conjunctively connected to the data basis. As result, we collect all similarity conditions which can be distributed over different `where`-clauses of several SFW blocks.

(R3) We already explained the handling of the outer SFW block of rule (R3) in the last rule (R2). Thus, we now only give the mapping of the inner join components. Both recursive functions ψ_{E_1} and ψ_{E_2} are conjunctively connected to the respective join condition `jc`, because all three parts must be fulfilled to generate a joined resulting tuple.

(R4) An intersection operation is directly mapped to the corresponding logical operator $\wedge$. Thereby, the operation $(\psi_{E_2})_{<\texttt{B}_{E_2,1},\dots,\texttt{B}_{E_2,m} \mid \texttt{B}_{E_1,1},\dots,\texttt{B}_{E_1,m}>}$ represents a string substitution, i.e. we replace the output attributes $\texttt{B}_{E_2,1}, \dots, \texttt{B}_{E_2,m}$ of $\texttt{E}_2$ in ψ_{E_2} by the output attributes $\texttt{B}_{E_1,1}, \dots, \texttt{B}_{E_1,m}$ of $\texttt{E}_1$.

(R5) We transform a union operation into the respective logical operator $\vee$ and apply the already discussed string substitution again.

For exemplifying our syntax mapping we give the scoring function ψ_e based on our running example query (see Fig. (2)):

[8]As the approaches [Re and Suciu 2008], [Widom 2008] and [Koch 2008] ProQua is currently focused on queries without the algebra operator *difference*. Please be aware that we still can use negations within a `where`-clause.

```
input:  logic-based scoring function
output:  normalised logic-based scoring function

(1)  transform $\psi$ into disjunctive normal form $\hat{c}_1 \vee \ldots \vee \hat{c}_m$ where $\hat{c}_i$ are
     conjunctions of literals
(2)  simplify $\psi$ by applying idempotence and invertibility rules
(3)  if there is an overlap on an attribute queried by similarity predicates
     between some conjunctions $\hat{c}_i$ then
  (3a) let $l$ be a literal of an attribute common to at least two conjunctions
  (3b) replace each conjunction $\hat{c}_i$ in $\psi$ with $(l \wedge \hat{c}_i) \vee (\neg l \wedge \hat{c}_i)$
  (3c) simplify $\psi$ by applying idempotence, invertibility and absorption, and
       obtain $\psi = (l \wedge \hat{c}_1) \vee \ldots \vee (l \wedge \hat{c}_{m_1}) \vee (\neg l \wedge \hat{c}_{m_1+1}) \vee \ldots \vee (\neg l \wedge \hat{c}_{m_2})$
  (3d) replace $\psi$ with $(l \wedge \psi_1) \vee (\neg l \wedge \psi_2)$ where
       $\psi_1 = \hat{c}_1 \vee \ldots \vee \hat{c}_{m_1}, \psi_2 = \hat{c}_{m_1+1} \vee \ldots \vee \hat{c}_{m_2}$
  (3e) continue with step (3) for $\psi_1$ and $\psi_2$
(4)  transform innermost disjunctions to conjunctions and negations by
     applying de-Morgan-law
```

Fig. 6. Syntactical normalisation algorithm norm(·)

$$\psi_e := \exists\ sond, age : [(\exists\ expert : (ArteExp(expert, rep, aid, culture) \wedge rep \approx A) \vee \exists\ method : ArteMat(method, year, aid, culture) \wedge year \approx 2012) \wedge (Arte(aid, type, sond, age) \wedge (age \approx 300))] \wedge age \approx 300$$

Please note that the join condition *(origin.aid=prop.aid)* (see Fig. (2)) is implicitly expressed by the attribute *aid* occurring in all relation predicates.

5. EVALUATION OF A LOGIC-BASED SCORING FUNCTION

In the following we explain the evaluation of a logic-based scoring function ψ as specified in Def. (3.1) and Def. (3.2). We presume that ψ is always derived from a QSQL2 query. In the remainder of this work we use the term *logic-based scoring function* according to this subclass. Conceptionally, we perform three steps to determine the outcome of a logic-based scoring function ψ:

(1) generating the syntax of ψ from a QSQL2 query,
(2) transforming ψ into a syntactical normal form by the normalisation algorithm norm(·) and
(3) evaluating the normalised scoring function norm(ψ) by a relational query plan qp(ψ).

The generation of a logic-based scoring function based on a given QSQL2 query is already described in Sec. (4). In the next subsections we will present the remaining points (2) and (3).

5.1 Syntactical normalisation

For starters, we motivate our syntactical normalisation by investigating the following simplified similarity condition

$$\psi_{sp}{}^{9} \equiv ((rep \approx A \vee year \approx 2012) \wedge age \approx 300) \wedge age \approx 300$$

which is taken from ψ_e, see Sec. (4). For the sake of simplicity, we temporarily neglect quantifiers and relation predicates in ψ_{sp}. Let us assume that we *directly* evaluate ψ_{sp} by the evaluation rules of Def. (3.2), i.e. *without* a preceded normalisation. In this case, we would obtain

$$((rep \approx A \vee year \approx 2012)^{10} * \text{psf}_{\approx,age}(t[age], 300)) * \text{psf}_{\approx,age}(t[age], 300)$$

[9]The index *sp* stands for simplified query.
[10]For clarity reasons, we have not applied the evaluation rule for $\vee$.

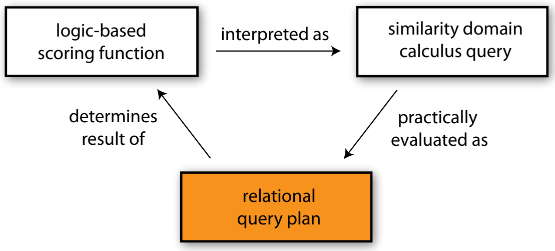

Fig. 7. Basic interrelations between logic-based scoring functions, domain calculus queries and relational query plans

as resulting score $\psi_{sp}(t)$. The resulting score for the subcondition $(rep \approx A \vee year \approx 2012)$ is multiplied with $\text{psf}_{\approx,age}(t[age], 300)$ *twice*. This double multiplication violates the *logical law of idempotence*[11]. Intuitively, this law expresses that the two conditions

$$((rep \approx A \vee year \approx 2012) \wedge age \approx 300) \wedge age \approx 300 \text{ and } (rep \approx A \vee year \approx 2012) \wedge age \approx 300$$

produce equal resulting scores, because we query the *same* predicate $age \approx 300$ twice in the first condition. In contrast, the *direct* evaluation of the first variant "blindly" computes the result without considering the underlying semantics. In this case, it ignores the fact that two subconditions are formed by the same predicate.

To avoid such incorrect evaluations we propose a *syntactical normalisation* of ψ. This guarantees a syntactical form where an evaluation by Def. (3.2) obeys all logical laws of a Boolean algebra, e.g. associativity, distributivity, contradiction and idempotence. We conduct such a transformation by employing the normalisation algorithm $\text{norm}(\cdot)$ [Schmitt 2008] given in Fig. (6). It basically relies on the well-known Shannon expansion. By applying norm on ψ_{sp} we obtain $\text{norm}(\psi_{sp}) \equiv (rep \approx A \vee year \approx 2012) \wedge age \approx 300$ which can be correctly evaluated by the rules of Def. (3.2).

5.2 Domain calculus queries

Our main idea for building a relational query plan $\text{qp}(\psi)$ basically relies on two principles. At first, we interpret a logic-based scoring function as a *domain calculus query* [Maier 1983] extended by similarity predicates. Secondly, we transform the result definition of such a *descriptive* query into a *procedural* query plan. The core concepts of our approach are interrelated in Fig. (7).

Classical domain calculus query: The starting point for the following discourse is a classical domain calculus query given in the form of

$$\{t[\mathcal{A}] \mid t \in \text{Dom}, cond(t) \equiv \text{true}\}.$$

In this query Dom stands for an (infinite) set of tuples defining the domain of the query and $cond(t)$ represents a Boolean condition for selecting relevant tuples. Before the set of all relevant tuples build the final result, all relevant tuples are projected onto a set of output attributes $\mathcal{A}$. As an example we consider following query derived from our running scenario:

$$\{t[aid, type] \mid t \in \text{Dom}, (Arte(aid, type, sond, age) \wedge age > 350)(t) \equiv \text{true}\}.$$

This query asks for all stored artefacts being older than 350 years. The underlying domain is defined by the cartesian product of all involved attribute domains, i.e. $\text{Dom} := \text{Dom}(aid) \times \text{Dom}(type) \times \text{Dom}(sond) \times \text{Dom}(age)$. We illustrate the result definition of the current example query in Fig. (8). All domain tuples of Dom are sketched in the left columns of Fig. (8). They are built by all possible combinations of all possible attribute values. The evaluation of the relation predicate $Arte(\ldots)$ and the result of the Boolean predicate $age > 350$ are shown in the middle columns. The remaining

[11]Idempotence law: $\psi \wedge \psi \equiv \psi$ and $\psi \vee \psi \equiv \psi$

aid	**type**	sond	age	$Arte(\ldots)$	$age > 350$	$cond(t)$
⋮						
art1	vase fragment	3	299	false	false	false
art1	vase fragment	3	300	true	false	false
art1	vase fragment	3	301	false	false	false
art2	spear head	10	499	false	true	false
art2	**spear head**	10	500	**true**	**true**	**true**
art2	spear head	10	501	false	true	false
art3	vase fragment	4	299	false	false	false
art3	vase fragment	4	300	true	false	false
art3	vase fragment	4	301	false	false	false
⋮						

Fig. 8. Result definition for the example of a classical domain calculus query: domain tuples (left columns), predicate evaluations (middle columns) and the evaluation of the combined condition (right column).

column $cond(t)$ contains the result of the combined condition $Arte(\ldots) \wedge age > 350$. Generally, we determine the result of domain calculus query by iterating all domain tuples and testing them against the selection condition, i.e. in our case against $Arte(\ldots) \wedge age > 350$. Subsequently, we take all tuples evaluated to true and project them on their data values for the attributes *aid* and *type*. Finally, we achieve $\{(\text{art2}, \text{spear head})\}$ as resulting tuple set for our example query.

Similarity domain calculus query: Next we enhance classical domain calculus queries by incorporating similarity conditions. For this purpose, we work with following basic structure

$$\{t[\mathcal{A}] \mid t \in \text{Dom}, \psi(t) > 0\}.$$

Obviously, the selection condition $cond(t) \equiv \text{true}$ is replaced by the condition $\psi(t) > 0$ involving a logic-based scoring function. As an example we consider the query

$$\{t[aid, culture] \mid t \in \text{Dom}_{\psi_{sb}}, \psi_{sb}(t) > 0\}$$

whereby ψ_{sb}[12,13] is given by:

$$\begin{gathered}\psi_{sb} \equiv \exists\ expert, rep : (ArteExp(expert, rep, aid, culture) \wedge rep \approx \text{A}) \vee \\ \exists\ method, year : (ArteMat(method, year, aid, culture) \wedge year \approx 2012).\end{gathered}$$

The function ψ_{sb} is a subcondition of ψ_e. Analogously to the former classical query, we demonstrate the result definition of ψ_e by sketching the respective domain tuples and the determined predicate evaluations in Fig. (9). Since the condition ψ_{sb} is already given in the required normal form, we can directly apply the evaluation rules of Def. (3.2) for computing $\psi_{sb}(t)$. Please note that the application of both existential quantifiers always choose the maximal score value generated by a group of tuples which share the same attribute values for the *unbounded*[14] attributes of ψ, see rule (7) of Def. (3.2). By iterating over all domain tuples we obtain the result set

$$\{(\text{art1}, \text{roman}, 0.923), (\text{art1}, \text{greek}, 0.923), (\text{art2}, \text{eqyptian}, 1), (\text{art2}, \text{punic}, 0.918)\}$$

when we add the final score value as third output attribute.

After describing how we can theoretically determine the result we translate the demonstrated procedure into a relational query plan $\text{qp}(\psi)$.

[12]The index *sb* stands for s̲ub̲condition.
[13]The function ψ_{sb} is inferred from the QSQL2 query `(select aid, culture from ArteExp where rep ≈ A) union (select aid, culture from ArteMat where year ≈ 2012)` which is a subquery of our running example. For the construction of ψ_{sb} also see the rules of Fig. (5).
[14]An attribute is called unbounded in ψ when it is not involved in any existential/universal quantifier of ψ.

aid	**culture**	expert	rep	method	year	$AE(\ldots)$	$rep \approx$A	$AM(\ldots)$	$year \approx 2012$	$\psi_{sb}(t)$
⋮										
art1	roman	Peter	B	XRF	1996	1	0.75	0	0.673	0.75
art1	**roman**	Peter	B	XRF	1997	**1**	**0.75**	**1**	**0.694**	**0.923**
art1	roman	Peter	B	XRF	1998	1	0.75	0	0.714	0.75
art1	roman	Cathy	C	XRF	1996	1	0.5	0	0.673	0.5
art1	roman	Cathy	C	XRF	1997	1	0.5	1	0.694	0.847
art1	roman	Cathy	C	XRF	1998	1	0.5	0	0.714	0.5
art1	greek	Peter	B	XRF	1996	1	0.75	0	0.673	0.75
art1	**greek**	Peter	B	XRF	1997	**1**	**0.75**	**1**	**0.694**	**0.923**
art1	greek	Peter	B	XRF	1998	1	0.75	0	0.714	0.75
art2	**egyptian**	John	A	XRF	2010	**1**	**0.959**	**1**	**1**	**1**
art2	**punic**	Frank	A	ICS-MS	2008	**0**	**1**	**1**	**0.918**	**0.918**
art3	roman	Frank	A	ICS-MS	2012	0	1	0	1	0
⋮										

Fig. 9. Result definition for the similarity domain calculus query using ψ_{sb}: domain tuples (left columns), predicate evaluations (middle columns) and the evaluation of the combined similarity condition (right column). The abbreviated predicate names $AE(\ldots)$ and $AM(\ldots)$ stand for $ArteExp(\ldots)$ and $ArteMat(\ldots)$

5.3 Relational query plan

Similar to SQL, the domain calculus is a *descriptive* query language. The result of such a query is determined by specifying properties which all resulting tuples have in common. The practical and algorithmic generation of the result is thereby purposely left open. For instance, we used an iteration over an infinite domain to achieve the result of ψ in the previous subsection. Obviously, such an enumeration is not possible in practise.

Consequently, we have to develop a relational query plan which can be practically performed on a relational database system. For this purpose, we split the result definition of a similarity domain calculus query into three consecutive subtasks:

(1) generating the domain Dom_ψ by a relational algebra query Q_{Dom_ψ},

(2) computing the score value $\psi(t)$ for all tuples of Q_{Dom_ψ} and

(3) projecting all tuples with $\psi(t) > 0$ onto the given output attributes $\mathcal{A}$.

(1) Generating Dom_ψ by Q_{Dom_ψ}: Since our logic-based scoring functions are constructed by QSQL2 queries, we can show that their domains Dom_ψ are always finite and can be computed by a relational algebra query Q_{Dom_ψ}. Thereby, the structure of Q_{Dom_ψ} is derived from the logical combination of the relation predicates involved in ψ, e.g. $(ArteExp(\ldots) \vee ArteMat(\ldots)) \wedge Arte(\ldots)$ in ψ_e. Concretely, we map a conjunctive combination $R_1(\ldots) \wedge R_2(\ldots)$ to a natural join $R_1 \bowtie R_2$ and a disjunction $R_1(\ldots) \vee R_2(\ldots)$ to a <u>f</u>ull <u>o</u>uter join $R_1 \bowtie^{fo} R_2$. These both rules lead to the property: $\forall t \in \mathrm{Dom}_\psi : \psi(t) > 0 \Rightarrow t \in Q_{\mathrm{Dom}_\psi}$. That is, by processing Q_{Dom_ψ} we certainly obtain each domain tuple which might have a final score value greater than 0. For ψ_e we get the query $Q_{\mathrm{Dom}_\psi} \equiv (ArteExp \bowtie^{fo} ArteMat) \bowtie Arte$ producing all relevant tuples.

(2) Computing $\psi(t)$ and (3) projection on $\mathcal{A}$: The primary goal for the subtasks (2) and (3) is a transformation of ψ into a syntactical form representing the basis for the desired query plan. The inferred query plan will consist of standard relation operators which are easily implementable by standard SQL. For this purpose, we have to prepare ψ by (i) a normalisation (see above), (ii) a transformation of all relation predicates and (iii) a shifting of all existential quantifiers. As already demonstrated, we need to normalise ψ before we can employ the evaluation rules of Def. (3.2). Additionally, we convert each relation predicate $R_i(\ldots)$ of ψ into a standard binary predicate of the form $R_i = 1$. This is necessary, because we intent to evaluate all predicates directly on attributes of Q_{Dom_ψ}. Thus, we map a relation predicate $R_i(\ldots)$ to an artificial predicate $R_i = 1$ embodying

tid	**aid**	**culture**	expert	rep	method	year	sond	age	AE	AM	A	$\psi_{qf}(t)$
(t_4, t_8, t_1)	**art1**	**roman**	Peter	B	XRF	1997	3	300	1	1	1	**0.923**
(t_5, t_9, t_1)	**art1**	**greek**	Peter	B	XRF	1997	3	300	1	1	1	**0.923**
(t_6, t_8, t_1)	art1	roman	Cathy	C	XRF	1997	3	300	1	1	1	0.847
(t_7, t_{11}, t_2)	**art2**	**egyptian**	John	A	XRF	2010	10	500	1	1	1	**0.933**
(t_{10}, t_2)	**art2**	**punic**	null	null	ICS-MS	2008	10	500	null	1	1	**0.857**

Fig. 10. This table shows the tuple set generated by Dom_{ψ_e} and the score values computed by $\psi_{e,qf}$ (the abbreviated column names AE, AM and A stand for ArteExp, ArteMat and Arte).

the connection between the data values and their source relation. Simultaneously, we extend each relation R_i involved in Q_{Dom_ψ} by an artificial attribute labelled by the respective relation name R_i. It contains the value 1 for all tuples. We denote the extended version of R_i as R'_i. For example, we get following transformed formula for the already normalised formula ψ_e:

$$\psi'_e := \exists\ sond, age : [(\exists\ expert : (ArteExp = 1 \wedge rep \approx A) \vee \exists\ method : (ArteMat = 1 \wedge year \approx 2012)) \wedge (Arte = 1 \wedge age \approx 300)].$$

The condition ψ'_e is then evaluated on the tuples generated by $(ArteExp' \bowtie^{fo} ArteMat') \bowtie Arte'$.

In the next step we transform ψ' into prenex normal form where all existential quantifiers are located at the beginning of ψ', i.e. $\psi' \rightsquigarrow \psi'' := \exists\ A_1, \ldots, A_l : \psi_{qf}$[15] whereby $\exists\ A_1, \ldots, A_l : \psi_{qf}$ is equivalent to ψ' and the subcondition ψ_{qf} does not contain any existential/universal quantifiers. Such an equivalent transformation is always possible, because our underlying geometric retrieval model forms a Boolean algebra [Schmitt 2008]. When we reconsider the last example condition, then we achieve:

$$\psi''_e := \exists\ sond, age, expert, method : [((ArteExp = 1 \wedge rep \approx A) \vee (ArteMat = 1 \wedge year \approx 2012)) \wedge (Arte = 1 \wedge age \approx 300)]$$

As result of this shifting, we can now evaluate all existential quantifiers of $\exists A_1, \ldots, A_l : \psi_{qf}$ by a *single* maximum operation applied on the tuples of Q_{Dom_ψ} (see rule (7) of Def. (3.2)), i.e.

$$\psi''(t) := \max_{\substack{\hat{t} \in Q_{\text{Dom}_\psi}, \hat{t}[\mathcal{B}] = t[\mathcal{B}], \\ \mathcal{B} = \text{attr}(t) \setminus \{A_1, \ldots, A_l\}}} \{\psi_{qf}(\hat{t})\}.$$

Thereby, the attribute set $\mathcal{B}$ is equal to all unbounded attributes of ψ, i.e. to all output attributes $\mathcal{A}$. In combination with the final projection on $\mathcal{A}$ we can therefore compute the final score values by a grouping operation as known from SQL [Date 1997]:

$$\text{qp}(\psi) := \gamma_{\mathcal{A};\max(\text{norm}(\psi_{qf})(t))}(Q_{\text{Dom}_\psi}),$$

whereby (i) the operator γ stands for a grouping operation, (ii) the attribute set $\mathcal{A}$ represents the grouping attributes of γ, (iii) the operation $\max(\text{norm}(\psi_{qf})(t))$ defines the applied aggregation function of γ and (iv) the algebra query Q_{Dom_ψ} computes the underlying finite domain. As an example we give the query plan $\text{qp}(\psi_e)$ for our running example:

$$\text{qp}(\psi_e) := \gamma_{(\text{aid},\text{type},\text{culture});\max(\text{norm}(\psi_{e,qf}(t)))}((ArteExp' \bowtie^{fo} ArteMat') \bowtie Arte'),$$

with $\text{norm}(\psi_{e,qf}(t)) \equiv ((ArteExp = 1 \wedge rep \approx A) \vee (ArteMat = 1 \wedge year \approx 2012)) \wedge (Arte = 1 \wedge age \approx 300)$. The domain tuple set determined by Q_{Dom_ψ} and the respective score values of $\psi_{e,qf}(t)$ are listed in Fig. (10). The final result returned by the last grouping operation is already known from the corresponding similarity calculus query as

$$\{(\text{art1}, \text{roman}, 0.923), (\text{art1}, \text{greek}, 0.923), (\text{art2}, \text{eqyptian}, 1), (\text{art2}, \text{punic}, 0.918)\}.$$

[15]The index *qf* stands for quantifier free.

Next we formalise our introduced ideas by the following definition. Specifically, it determines qp(ψ) by recursively constructing (i) the domain query Q_{Dom_ψ}, (ii) the quantifier free condition ψ_{qf} and (iii) the output attribute set $\mathcal{A}$. Thereby, we directly infer Q_{Dom_ψ}, ψ_{qf} and $\mathcal{A}$ from the given QSQL2 query instead of explicitly building ψ before.

Definition 5.1 *Constructing* Q_{Dom_ψ}*,* ψ_{qf} *and* $\mathcal{A}$ *for* qp(ψ). If a QSQL2 query is given, then we recursively build Q_{Dom_ψ}, ψ_{qf} and $\mathcal{A}$ with following rules:

rule	QSQL2 query	query plan components
R1	`R`	$Q_{\mathrm{Dom}_\psi} := R'$ $\psi_{qf} := (R = 1)$ $\mathcal{A} := \mathrm{attr}(R)$

R2	`select B1,..,Bm`
	`from E1`
	`where sc`
		$Q_{\mathrm{Dom}_\psi} := \sigma_{\mathrm{bool}(\mathtt{sc})}(Q_{E_1,\mathrm{Dom}_\psi})$
		$\psi_{qf} := \psi_{E_1,qf} \wedge \mathtt{sc}$
		$\mathcal{A} := \{B_1, \ldots, B_m\}$

R3	`select B1,..,Bm`
	`from ( E1 inner join E2`
	`on jc ) where sc`
		$Q_{\mathrm{Dom}_\psi} := \sigma_{\mathrm{bool}(\mathtt{sc})}(Q_{E_1,\mathrm{Dom}_\psi} \bowtie_{\mathtt{jc}} Q_{E_2,\mathrm{Dom}_\psi})$
		$\psi_{qf} := \psi_{E_1,qf} \wedge \psi_{E_2,qf} \wedge \mathtt{sc}$
		$\mathcal{A} := \{B_1, \ldots, B_m\}$

R4	`E1 intersect E2`	$Q_{\mathrm{Dom}_\psi} := Q_{E_1,\mathrm{Dom}_\psi} \bowtie \beta_{(\mathcal{A}_{E_1} \leftarrow \mathcal{A}_{E_2})}(Q_{E_2,\mathrm{Dom}_\psi})$ $\psi_{qf} := \psi_{E_1,qf} \wedge (\psi_{E_2,qf})_{<\mathcal{A}_{E_2}\mid\mathcal{A}_{E_1}>}$ $\mathcal{A} := \mathcal{A}_{E_1}$
R5	`E1 union E2`	$Q_{\mathrm{Dom}_\psi} := Q_{E_1,\mathrm{Dom}_\psi} \bowtie^{\text{full outer}} \beta_{(\mathcal{A}_{E_1} \leftarrow \mathcal{A}_{E_2})}(Q_{E_2,\mathrm{Dom}_\psi})$ $\psi_{qf} := \psi_{E_1,qf} \vee (\psi_{E_2,qf})_{<\mathcal{A}_{E_2}\mid\mathcal{A}_{E_1}>}$ $\mathcal{A} := \mathcal{A}_{E_1}$

The auxiliary function bool($\mathtt{sc}$) returns a Boolean condition such that the implication $\mathtt{sc}(t) > 0 \Rightarrow \mathrm{bool}(\mathtt{sc})(t) = \mathrm{true}$ always holds.

The following theorem proves the correctness of the developed query plan qp(ψ).

THEOREM 5.2. *Let* ψ *be a logic-based scoring function which is derived from a QSQL2 query. If* Q_{Dom_ψ}*,* ψ_{qf} *and* $\mathcal{A}$ *are constructed as specified in Def. (5.1), then*

$$\{t[\mathrm{unb}(\psi)] \bullet (\psi(t)) \mid t \in \mathrm{Dom}_\psi, \psi(t) > 0\} \text{ equals } \sigma_{\mathrm{score}>0}(\gamma_{\mathcal{A};\max(\mathrm{norm}(\psi_{qf})(t)) \text{ as score}}(Q_{\mathrm{Dom}_\psi}))$$

whereby $\mathrm{unb}(\psi)$ *returns all unbounded variables of* ψ*. The complexity of* $\gamma_{\mathcal{A};\max(\mathrm{norm}(\psi_{qf})(t))}(Q_{\mathrm{Dom}_\psi})$ *is in* $\mathcal{O}(n^k * l + 2^r)$ *whereby* n *describes the size of the database,* k *stands for the number of all involved relations in* Q*,* l *represents the effort for evaluating the normalised* $\psi_{qf}(t)$ *and* 2^r *gives the cost of the normalisation, if* r *gives the formula length of* ψ_{qf}*). For a detailed proof see [Lehrack 2012b].*

5.4 Combined query plans

So far, we developed a relational query plan evaluating queries from class SConCD. When we extend our focus to SConUD-queries we need to apply specific ranking semantics. ProQua exploits *expected scores* [Li et al. 2011], i.e. the resulting score value of a tuple is weighted by the occurrence probability of the considered tuple: $\mathrm{escore}(t) := \psi(t) * \mathrm{Pr}_Q(t)$. The value $\mathrm{Pr}_Q(t)$ describes the occurrence probability of t.

Since qp(ψ) and the query plan for occurrence probabilities qp(Q) generate equivalent tuple sets (augmented by $\psi(t)$ and $\mathrm{Pr}_Q(t)$, respectively), we can compute the final expected scores by a single natural join:

$$\pi_{\mathcal{A},\psi(t)*\mathrm{Pr}_Q(t) \text{ as escore}}(\gamma_{\mathcal{A};\max(\mathrm{norm}(\psi_{qf})(t)) \text{ as } \psi(t)}(Q_{\mathrm{Dom}_\psi}) \bowtie \pi_{\mathcal{A},\mathrm{Pr}_Q(t)}(\mathrm{qp}(Q))).$$

We want to point out that we are able to optimise this single join by *merging* both query plans. A detailed discussion about merging techniques would require a comprehensive introduction of the query plan qp(Q) which is not addressed in this work.

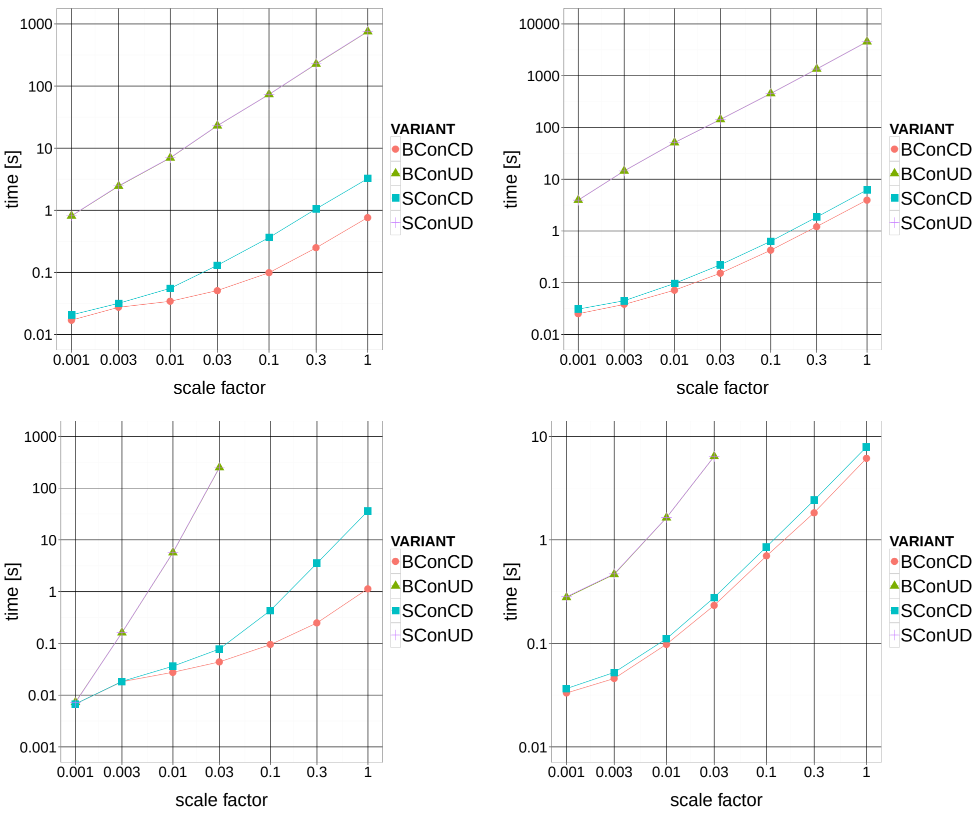

Fig. 11. TPC-H experiments measured in seconds: adapted versions of Q2 (top-left), Q3 (top-right), Q7 (bottom-left) and Q19 (bottom-right).

6. EXPERIMENTS

In order to compare the computation times of queries taken from all four query classes `BConCD`, `SConCD`, `BConUD` and `SConUD` we carried out multiple experiments on a probabilistic version of the TPC-H database (version 2.14.3)[16]. Thereby, the experiments were conducted on a 2xAMD octo-core Opteron 6134 (2.3GHz)/64bit/64GiB RAM machine running CentOS 5.8 (Linux)/Oracle 11.2.

To create a probabilistic variant of the database we augmented each tuple by a Boolean random variable with a uniform distribution. In detail, we investigated the TPC-H queries Q2, Q3, Q7 and Q19 as variants of all four classes, i.e. we removed all aggregations and added logic-based similarity conditions, respectively. A more detailed description of the investigated queries can be found in the extended version of this article [Lehrack 2012b].

In Fig. (11) the computation times of the queries Q2, Q3, Q7 and Q19 as `BConCD`, `SConCD`, `BConUD` and `SConUD` variants are depicted. Please note that the `BConUD` and `SConUD` variants of Q7 and Q19 reached the time limit very soon at scale factor 0.1. Not surprisingly, we achieve the variant ordering `SConUD` $>$ `BConUD` $>>$ `SConCD` $>$ `BConCD` for all queries, when we compare the corresponding computation times for a given combination of query and scale factor. This reflects the complexity classes of the query variants, i.e. $\#\mathcal{P}$ for `SConUD` and `BConUD` (see [Dalvi and Suciu 2007]) vs. $\mathcal{P}$ in data size for `SConCD` and `BConCD` (see Theorem (5.2)). Additionally, we calculated the additional costs for computing score values instead of Boolean truth values (i.e. the increasing rates from a `BConCD` variant to a corresponding `SConCD` variant) for our example queries as Q2 : $15.9\% - 323.6\%$, Q3 : $17.8\% - 56.8\%$, Q7 : $0.2\% - 373.3\%$ and Q19 : $10.3\% - 31.9\%$ ranging over the given scale factors

[16]http://www.tpc.org/tpch/

between 0.001 and 1. Summarising, we conclude that the increasing rates strongly depend on the given query structure. The broad range of rates of growth also point out that there is a promising potential for optimisation techniques which will be approached in future works.

7. RELATED WORK

Over the last years a remarkable amount of probabilistic database approaches as [Fuhr and Roelleke 1997; Dalvi and Suciu 2007; Koch 2008; Widom 2008] have been proposed. They all facilitate the processing of relational queries on uncertain relational data, i.e. queries from class `BConUD`.

An important comparison criterion for a query language is its expressiveness. Especially, the landmark papers [Fuhr and Roelleke 1997] and [Dalvi and Suciu 2007] already discussed the additional support of similarity predicates, i.e. queries of class `SConUD`, by modelling score values as probabilities. Fuhr and Rölleke [Fuhr and Roelleke 1997], for instance, suggested to integrate similarity predicates as built-in predicates. Concretely, they encoded a scoring function by a probabilistic relation containing all relevant domain value pairs and their respective similarity values. These *auxiliary scoring relations* were then used within an algebra query substituting a respective function call. Unfortunately, the class of valid queries constructed by this method is very limited. All queries which build complex tuple events involving more than one basic tuple event from the same auxiliary scoring relation are forbidden. This is rooted by the fact that basic tuple events of auxiliary scoring relation can be complexly correlated. Those correlations are not captured in the auxiliary scoring functions.

In contrast, Dalvi and Suciu proposed in [Dalvi and Suciu 2007] to determine the score values of all similarity predicates in advance by a pre-processing step. The calculated score values are then integrated into the queried probabilistic relations as occurrence probabilities. Inconveniently, the corresponding system MystiQ [Re and Suciu 2008] leaves this pre-processing step to the user, since its query language does not feature similarity predicates as an integrated language concept. Furthermore, this method is restricted to the set of conjunctive queries, because the applied join operations always aggregates probabilities conjunctively.

Further approaches as [Widom 2008; Koch 2008] support uncertainty on attribute level. In these approaches the evaluation of a single similarity predicate could be encoded in the queried uncertain attribute. But once again this approach is limited to conjunctive queries, because the probability for an entire tuple is technically combined by a conjunctive join operation.

To the best of our knowledge, ProQua [Lehrack and Schmitt 2011; Lehrack and Saretz 2012; Lehrack 2012a] is the first and only probabilistic database system which enables complex `SConCD`-queries and `SConUD`-queries by integrating a generic similarity operator into its query language QSQL2. In [Lehrack et al. 2012] we give a *non-technical* introduction of the capabilities of our query language QSQL2 *without* describing any technical evaluation techniques. In [Lehrack and Schmitt 2010] we presented the first version of QSQL as a query language which is exclusively applied on deterministic data, i.e. this version *only* supports queries from `BConCD` and `SConCD`.

Besides ProQua, fuzzy databases as [Galindo et al. 2006] also offer complex `SConCD`- and `SConUD`-queries employing fuzzy logic [Zadeh 1988]. However, fuzzy databases do not rely on probabilistic semantics. In [Schmitt et al. 2009] we presented a detailed comparison between fuzzy logic and quantum logic which can be interpreted probabilistically [Lehrack and Schmitt 2011]. Unfortunately, there is a significant set of fuzzy queries where the computed result does *not* meet user expectations adequately. Especially, the result of the *minimum* function, which is the *only* t-norm with the logic properties idempotence and distributivity, can depend only on one input parameter (dominance problem).

8. CONCLUSION AND OUTLOOK

In this article we introduced the concept of logic-based scoring functions formalising complex similarity conditions on uncertain relational data as a key feature of our probabilistic database system ProQua. In particular, we developed comprehensive techniques for evaluating logic-based scoring functions. Furthermore, we showed that we can easily combine the computations of score values and occurrence probabilities by a relational query plan when we presume expected scores as underlying ranking semantics. In future works we want to integrate optimisation rules for merged query plans and extend ProQua by further ranking semantics.

REFERENCES

Agrawal, R., Ailamaki, A., and Bernstein, Philip A., e. a. The claremont report on database research. *Journal of ACM Special Interest Group on Management of Data* vol. 37, pp. 9–19, September, 2008.

Agrawal, S., Chaudhuri, S., Das, G., and Gionis, A. Automated ranking of database query results. In *Proceedings of the Conference on Innovative Data Systems Research*. Asilomar, CA, USA, 2003.

Dalvi, N. and Suciu, D. Efficient query evaluation on probabilistic databases. *Journal of Very Large Data Bases* 16 (4): 523–544, October, 2007.

Date, C. *A Guide to the SQL Standard, 4th Edition*. Addison Wesley, 1997.

Fuhr, N. and Roelleke, T. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. IS* 15 (1): 32–66, 1997.

Galindo, J., Urrutia, A., and Piattini, M. *Fuzzy Databases: Modeling, Design and Implementation*. Idea Group Publishing, Hershey, USA, 2006.

Henze, F., Lehmann, H., and Langer, W. Cisar - a modular database system as a basis for analysis and documentation of spatial information. In *Proceedings of the 35th International Conference on Computer Application and Quantitative Methods in Archaeology*, L. K. H. I. Posluchny, A. (Ed.). Dr. Rudolf Habelt GmbH Verlag, Berlin, pp. 228–233, 2007.

Ilyas, I. F. and Soliman, M. A. *Probabilistic Ranking Techniques in Relational Databases*. Synthesis Lectures on DM. Morgan & Claypool, 2011.

Koch, C. *MayBMS: A System for Managing Large Uncertain and Probabilistic Databases*. Springer-Verlag, 2008.

Lehrack, S. Applying Weighted Queries on Probabilistic Databases. In *Proceedings of the International Conference on Information and Knowledge Management*. Maui, USA, 2012a.

Lehrack, S. Evaluating Logic-Based Scoring Functions on Uncertain Relational Data, 2012b. BTU Cottbus, Technical Report.

Lehrack, S. and Saretz, S. A Top-k Filter for Logic-Based Similarity Conditions on Probabilistic Databases. In *Proceedings of the Advances in Databases and Information Systems*. Poznan, Poland, pp. 268–281, 2012.

Lehrack, S., Saretz, S., and Schmitt, I. QSQL2: Query Language Support for Logic-Based Similarity Conditions on Probabilistic Databases. In *Proceedings of the Research Challenges in Information Science*. Valencia, Spain, pp. 1–12, 2012.

Lehrack, S. and Schmitt, I. Qsql: Incorporating logic-based retrieval conditions into sql. In *Proceedings of the Database Systems for Advanced Applications*. Tsukuba, Japan, pp. 429–443, 2010.

Lehrack, S. and Schmitt, I. A Probabilistic Interpretation for a Geometric Similarity Measure. In *Proceedings of the European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. Belfast, UK, pp. 749–760, 2011.

Li, J., Saha, B., and Deshpande, A. A unified approach to ranking in probabilistic databases. *Journal of Very Large Data Bases* 20 (2): 249–275, 2011.

Maier, D. *The Theory of Relational Databases*. Computer Science Press, 1983.

Re, C. and Suciu, D. Managing Probabilistic Data with MystiQ: The Can-Do, the Could-Do, and the Can't-Do. In *Proceedings of the Scalable Uncertainty Management*. Naples, Italy, pp. 5–18, 2008.

Schmitt, I. QQL: A DB&IR Query Language. *Journal of Very Large Data Bases* 17 (1): 39–56, 2008.

Schmitt, I., Nuernberger, A., and Lehrack, S. On the relation between fuzzy and quantum logic. In *Views on Fuzzy Sets and Systems from Different Perspectives*, R. Seising (Ed.). Studies in Fuzziness and Soft Computing, vol. 243. Springer, pp. 417–438, 2009.

Suciu, D., Olteanu, D., Ré, C., and Koch, C. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

Widom, J. Trio: A system for data, uncertainty, and lineage. In *Managing and Mining Uncertain Data*. Springer, pp. 113–148, 2008.

Zadeh, L. A. Fuzzy logic. *IEEE Computer* 21 (4): 83–93, 1988.