Towards Top-k Query Answering in Deductive Databases

Umberto Straccia

Abstract— **In this paper we address a novel issue for deductive databases with huge data repositories, namely the problem of evaluating ranked top-k queries. The problem occurs whenever we allow queries such as "find cheap hotels close to the conference location" in which fuzzy predicates like cheap and close occur. We show how to compute efficiently the top-k answers of conjunctive queries with fuzzy predicates.**

I. INTRODUCTION

Since the introduction of relational databases in the Seventies [4], the close relationship between database theory and formal logic, in particular finite model theory, has been understood and exploited. In database theory, databases are often identified with finite relational structures. A query associates to each input database over a given schema (vocabulary) a result consisting of one or more output relations. Queries are formulated in query languages.

Deductive databases (cf. [1], [2], [12], [10], [11]) are relational databases whose query language and (usually) storage structure are designed around a logical model of data, usually Logic Programming [9]. It is well known that the deductive database field has close links with the logic programming community, and much of the development of deductive database systems has centered around languages based on Horn clauses. Relations are naturally thought of as the "value" of a logical predicate, and relational languages such as SQL are syntactic sugarings of a limited form of logical expression. Deductive database systems can be seen, thus, as an advanced form of relational systems. Deductive databases not only store explicit information in the manner of a relational database, but they also store rules that enable inferences based on the stored data to be made. Together with techniques developed for relational databases, this basis in logic means that deductive databases are capable of handling large amounts of information as well as performing reasoning based on that information.

There are many application areas for deductive database technology. One area is that of decision support systems. In particular, the exploitation of an organization's resources requires not only sufficient information about the current and future status of the resources themselves, but also a way of reasoning effectively about plans for the future.

Another fruitful application area is that of expert systems. There are many computing applications in which there are large amounts of information, from which the important facts may be distilled by a simple yet tedious analysis. For example, medical analysis and monitoring can generate a large amount of data, and an error can have disastrous consequences. A tool to carefully monitor a patient's condition or to retrieve relevant cases during diagnosis reduces the risk of error in such circumstances.

Planning systems are another application area. For example, a student planning a course of study at a university, or a passenger planning a round-the-world trip often need to consider a large body of information, as well as the ability to explore alternatives and hypotheses. A deductive database is able to advise students about pre-requisites and regulations on the choice of subjects, or a traveller of the financial implications of a given change in itinerary.

To date, deductive database systems have been the subject of extensive research, and several prototype deductive database systems.

In this paper we address a novel issue for deductive systems with huge data repositories, namely the problem of *evaluating ranked top-k queries*. So far, an answer to a query is a set of tuples that satisfy a query. Each tuple may or may not satisfy the predicates in the query. However, very often the information need of a user involves so-called *fuzzy predicates* [14]. For instance, a user may have the following information need:

> "Find *cheap* hotels *near* to the conference location."

Here, $cheap$ and $near$ are fuzzy predicates. Unlike the classical case, tuples satisfy now these predicates to a score (usually in $[0, 1]$). In the former case the score may depend, e.g., on the price, while in the latter case it may depend e.g. on the distance between the hotel location and the conference location.

Therefore, a major problem we have to face with in such cases is that now an answer is a set of tuples *ranked* according to their *score*. This poses a new challenge in case we have to deal with a huge amount of instances. Indeed, virtually every tuple may satisfies a query with a non-zero score and, thus, has to be ranked. Of course, computing all these scores, rank them and then select the top-k ones is not feasible in practice, as we may deal with millions of tuples.

For the sake of illustrative purposes, we address this problem for *Datalog* (cf. [1], [2], [12]. Datalog is a very powerful, well-studied declarative language based on Horn clause logic. Datalog adapts the paradigm of Logic Programming [9] to the database setting. We extend Datalog by allowing fuzzy predicates to appear in the queries (we call the language Datalog^{topk}) and propose methods to compute efficiently the top-k ranked answers, making the approach appealing for real world scenarios.

U. Straccia is with ISTI - CNR, Via G. Moruzzi, 1, 56124 Pisa, Italy straccia@isti.cnr.it

We proceed as follows. As next we recall Datalog, then present Datalog^{topk}. In Section IV we show how to compute efficiently the top-k answer set of conjunctive queries.

II. BASICS OF DATALOG

A Datalog *rule* is a Horn clause of the form

$$A \leftarrow A_1, \ldots, A_n \ ,$$

where A is the *head* of the rule, and $A_1, \ldots, A_n$ is the *body* of the rule. A and all A_i are atoms. An *atom* is of the form $P(t_1, \ldots, t_n)$, where P is an n-ary predicate symbol and all t_j are terms. A *term* is either a *variable* or a *constant*. For instance,

$$\begin{array}{lcll} \texttt{TeachesTo}(x,y) & \leftarrow & \texttt{Professor}(x), \texttt{Student}(y) & (1) \\ \texttt{HasTutor}(x,y) & \leftarrow & \texttt{Student}(x), \texttt{Professor}(y) & \end{array}$$

are two rules. The former dictatates that a professor teaches to a student, while the latter says that a student has a professor as tutor. A *fact* is a Datalog rule with empty body. For instance,

$$\begin{array}{ll} \texttt{Student}(\texttt{john}) & \leftarrow \\ \texttt{Professor}(\texttt{mary}) & \leftarrow \end{array} \quad (2)$$

are two facts. The former says that john is a Student, while the latter says that mary is a Professor. An *extensional database* (EDB) is a set of Datalog facts. We further require that no variable occurs in facts (the fact is ground, see below). For instance, the facts in Equation (2) form an extensional database. An *intentional database* (IDB) is a set of Datalog rules without facts. For instance, the rules in Equation (1) form an intentional database.

A Datalog program $\mathcal{P} = \langle \mathcal{P}_I, \mathcal{P}_E \rangle$ is given by an extensional database $\mathcal{P}_E$ and an intentional database $\mathcal{P}_I$ in which the predicates occurring in the extensional database do not occur in the head of rules of the intentional database. Essentially, we do not allow that the fact predicates occurring in $\mathcal{P}_E$ can be redefined by $\mathcal{P}_I$. For instance, the facts and rules in Equations (1) and (2) are a Datalog program $\mathcal{P}_0$.

Of course, not all deductive database systems restrict programs to be Datalog programs. Datalog programs are somewhat restrictive; for example, the append program is not a Datalog program, as it requires the use of function symbols. However, Datalog is sufficiently expressive to illustrate the salient issues related to top-k query answering.

From the semantics point of view [9], the *Herbrand universe* $H_\mathcal{P}$ of $\mathcal{P}$ is the set of constants appearing in $\mathcal{P}$. If there is no constant symbol in $\mathcal{P}$ then consider $H_\mathcal{P} = \{a\}$, where a is an arbitrary chosen constant. For instance, for $\mathcal{P}_0$, $H_{\mathcal{P}_0} = \{\texttt{john}, \texttt{mary}\}$. The *Herbrand base* $B_\mathcal{P}$ of $\mathcal{P}$ is the set of ground instantiations of atoms appearing in $\mathcal{P}$ (ground instantiations are obtained by replacing all variable symbols with constants of the Herbrand universe). For instance,

$$\begin{array}{lll} B_{\mathcal{P}_0} = \{ & \texttt{Student}(\texttt{john}), \texttt{Student}(\texttt{mary}), & \\ & \texttt{Professor}(\texttt{john}), \texttt{Professor}(\texttt{mary}), & \\ & \texttt{TeachesTo}(\texttt{john}, \texttt{john}), \texttt{TeachesTo}(\texttt{john}, \texttt{mary}), & \\ & \texttt{TeachesTo}(\texttt{mary}, \texttt{mary}), \texttt{TeachesTo}(\texttt{mary}, \texttt{john}), & \\ & \texttt{HasTutor}(\texttt{john}, \texttt{john}), \texttt{HasTutor}(\texttt{john}, \texttt{mary}), & \\ & \texttt{HasTutor}(\texttt{mary}, \texttt{mary}), \texttt{HasTutor}(\texttt{mary}, \texttt{john})\} \ . & \end{array}$$

A *Herbrand interpretation* of a $\mathcal{P}$ is any subset $I \subseteq B_\mathcal{P}$ of its Herbrand base. Intuitively, the atoms in I are true, and all others are false. That is, I satisfies a ground atom A iff $A \in I$. For instance,

$$\begin{array}{lll} I_0 & = & \{\texttt{Student}(\texttt{john}), \texttt{Professor}(\texttt{mary}), \\ & & \texttt{TeachesTo}(\texttt{mary}, \texttt{john}), \texttt{HasTutor}(\texttt{john}, \texttt{mary})\} \end{array}$$

is an interpretation asserting that john is a Student, mary is a Professor who both teaches to john and is his tutor.

Let $\mathcal{P}^*$ be the set of ground rule instantiations obtained from $\mathcal{P}$. Note that $\mathcal{P}^*$ is always finite. For instance, for $\mathcal{P}_0$, $\mathcal{P}_0^*$ is

$$\begin{array}{lcl} \texttt{TeachesTo}(\texttt{john}, \texttt{john}) & \leftarrow & \texttt{Professor}(\texttt{john}), \texttt{Student}(\texttt{john}) \\ \texttt{TeachesTo}(\texttt{john}, \texttt{mary}) & \leftarrow & \texttt{Professor}(\texttt{john}), \texttt{Student}(\texttt{mary}) \\ \texttt{TeachesTo}(\texttt{mary}, \texttt{mary}) & \leftarrow & \texttt{Professor}(\texttt{mary}), \texttt{Student}(\texttt{mary}) \\ \texttt{TeachesTo}(\texttt{mary}, \texttt{john}) & \leftarrow & \texttt{Professor}(\texttt{mary}), \texttt{Student}(\texttt{john}) \\ \texttt{HasTutor}(\texttt{john}, \texttt{john}) & \leftarrow & \texttt{Student}(\texttt{john}), \texttt{Professor}(\texttt{john}) \\ \texttt{HasTutor}(\texttt{john}, \texttt{mary}) & \leftarrow & \texttt{Student}(\texttt{john}), \texttt{Professor}(\texttt{mary}) \\ \texttt{HasTutor}(\texttt{mary}, \texttt{mary}) & \leftarrow & \texttt{Student}(\texttt{mary}), \texttt{Professor}(\texttt{mary}) \\ \texttt{HasTutor}(\texttt{mary}, \texttt{john}) & \leftarrow & \texttt{Student}(\texttt{mary}), \texttt{Professor}(\texttt{john}) \\ \texttt{Student}(\texttt{john}) & \leftarrow & \\ \texttt{Professor}(\texttt{mary}) & \leftarrow & \end{array}$$

An interpretation I *satisfies* (is a *model* of) $\mathcal{P}$ iff I *satisfies* (is a *model* of) $\mathcal{P}^*$. $\mathcal{I}$ *satisfies* (is a *model* of) $\mathcal{P}^*$ iff I *satisfies* (is a *model* of) all rules in $\mathcal{P}^*$. I *satisfies* (is a *model* of) a rule $A \leftarrow A_1, \ldots, A_n$ occurring in $\mathcal{P}^*$ iff $A \in I$ whenever $\{A_1, \ldots, A_n\} \subseteq I$. For instance, it is easily verified that I_0 is a model of $\mathcal{P}_0^*$ and, thus, of $\mathcal{P}_0$.

It is well known that each $\mathcal{P}$ has an unique *minimal* model $M_\mathcal{P}$, which coincides with the intersection of all models of $\mathcal{P}$. For instance, I_0 is the minimal model of $\mathcal{P}_0$.

$M_\mathcal{P}$ is also the least fixed-point of the immediate consequence operator $T_\mathcal{P} \colon 2^{\mathcal{B}_\mathcal{P}} \rightarrow 2^{\mathcal{B}_\mathcal{P}}$ defined as

$$\begin{array}{l} T_\mathcal{P}(I) = \{A \in B_\mathcal{P} \mid \mathcal{P}^* \text{ contains a rule } A \leftarrow A_1, \ldots, A_n \\ \qquad \text{such that } \{A_1, \ldots, A_n\} \subseteq I \text{ holds}\} \ . \end{array}$$

Intuitively, it yields all atoms that can be derived by a single application of some rule in $\mathcal{P}$ given the atoms in I . Since $T_\mathcal{P}$ is monotone, by the Knaster-Tarski Theorem it has a least fixed-point, denoted by $T_\mathcal{P}^\infty$; since, moreover, $T_\mathcal{P}$ is also continuous, by Kleene's Theorem $T_\mathcal{P}^\infty$ is the limit of the sequence

$$\begin{array}{lcl} T_\mathcal{P}^0 & = & \emptyset \\ T_\mathcal{P}^{i+1} & = & T_\mathcal{P}(T_\mathcal{P}^i), i \geq 0 \ . \end{array}$$

Then $M_\mathcal{P} = T_\mathcal{P}^\infty$.

We further say that $\mathcal{P}$ *entails* a ground atom A (denoted $\mathcal{P} \models A$) iff $A \in M_\mathcal{P}$. For instance, $\mathcal{P}_0 \models \texttt{TeachesTo}(\texttt{mary}, \texttt{john})$.

Given a Datalog program $\mathcal{P} = \langle \mathcal{P}_I, \mathcal{P}_E \rangle$, a Datalog *query predicate* q is a designated n-ary predicate symbol appearing in the head of rules in $\mathcal{P}_I$. We require that all these rules involving q have the same head $q(\mathbf{x})$, where $\mathbf{x}$ is an n-tuple of variables. For instance, with respect to $\mathcal{P}_0$

$$q(x) \leftarrow \texttt{TeachesTo}(x, y), \texttt{HasTutor}(y, z) \quad (3)$$

is a query asking for professors that teach to students, which have a tutor.

The *answer set* of q with respect to $\mathcal{P}$ is the set of tuples $\mathbf{c}$, such that $q(\mathbf{c})$ is entailed by $\mathcal{P}$, i.e.

$$ans(\mathcal{P}, q) = \{\mathbf{c} \mid \mathcal{P} \models q(\mathbf{c})\} \ .$$

For instance, given $\mathcal{P}_0$ and the query above,

$$ans(\mathcal{P}_0, q) = \{\langle \texttt{mary} \rangle\} \ .$$

Finally, we note that we may check whether $\mathcal{P} \models q(\mathbf{c})$ by computing $\mathcal{P}^*$ and then use the $T_\mathcal{P}$ operator to compute $M_\mathcal{P}$ and verify whether $q(\mathbf{c}) \in M_\mathcal{P}$. However, in practice, generating $\mathcal{P}^*$ is often cumbersome, since, it is in general exponential in the size of $\mathcal{P}$. Moreover, it is not always necessary to compute $M_\mathcal{P}$ in order to determine whether $\mathcal{P} \models A$ for some particular atom A.

For these reasons, completely different strategies of deriving atoms from a logic program have been developed, such as *SLD-resolution* (cf. [9]). Roughly, SLD-resolution can be described as follows. A *goal* is a conjunction of atoms, and a *substitution* is a function θ that maps variables $x_1, \ldots, x_n$ to terms $t_1, \ldots, t_n$. The result of simultaneous replacement of variables x_i by terms t_i in an expression E is denoted by $E\theta$. For a given goal G and a program $\mathcal{P}$, SLD-resolution tries to find a substitution θ such that $G\theta$ logically follows from $\mathcal{P}$. The initial goal is repeatedly transformed until the empty goal is obtained. Each transformation step is based on the application of the resolution rule to a selected atom B_i from the goal

$$\leftarrow B_1, \ldots, , B_{i-1}, B_i, B_{i+1}, \ldots, B_m$$

and a rule

$$A \leftarrow A_1, \ldots, A_n$$

from $\mathcal{P}$. SLD-resolution tries to unify B_i with the head A, that is, to find a substitution θ such that $A\theta = B_i\theta$. Such a substitution θ is called a unifier of A and B_i. If a unifier θ exists, a most general such θ (which is essentially unique) is chosen and the goal is transformed into the goal

$$\leftarrow (B_1, \ldots, B_{i-1}, A_1, \ldots, A_n, B_{i+1}, \ldots, B_m)\theta \ .$$

For a more precise account see [9]. For instance, given $\mathcal{P}_0$ and the query in Equation 3, Table I shows a sequence of SLD-resolution for the goal

$$\leftarrow \texttt{TeachesTo}(x, y), \texttt{HasTutor}(y, z)$$

ending with the empty goal and the substitution `mary` for x, proving that $\mathcal{P}_0 \models q(\texttt{mary})$.

III. DATALOG^{topk}

Datalog^{topk} is as Datalog except that we allow fuzzy predicates to occur in Datalog rule bodies of a designed Datalog query predicate q. Specifically, let $\mathcal{P}$ be a Datalog program and q a designed *query predicate* not occurring in $\mathcal{P}$. A Datalog^{topk} *conjunctive query* is of the form

$$q(\mathbf{x}, s) \leftarrow \exists \mathbf{y} \mathit{body}(\mathbf{x}, \mathbf{y}), s = f(p_1(\mathbf{z}_1), \ldots, p_n(\mathbf{z}_n))$$

where

1) $\mathbf{x}$ are the *distinguished variables*;
2) s is the *score variable*, taking values in $[0, 1]$;
3) $\mathbf{y}$ are so-called *non-distinguished variables* and are distinct from the variables in $\mathbf{x}$;
4) $\mathit{body}(\mathbf{x}, \mathbf{y})$ is a conjunction of Datalog atoms;
5) $\mathbf{z_i}$ are tuples of constants or variables in $\mathbf{x}$ or $\mathbf{y}$;
6) p_i is an n_i-ary *fuzzy predicate* assigning to each n_i-ary tuple $\mathbf{c}_i$ as *score* $p_i(\mathbf{c}_i) \in [0, 1]$;
7) f is a *scoring* function $f\colon [0, 1]^n \rightarrow [0, 1]$, which combines the scores of the n fuzzy predicates p_i into and overall *query score* to be assigned to the score variable s. We assume that f is *monotone*, i.e., for each $\mathbf{v}, \mathbf{v}' \in [0, 1]^n$ such that $\mathbf{v} \leq \mathbf{v}'$, $f(\mathbf{v}) \leq f(\mathbf{v}')$ holds, where $(v_1, \ldots, v_n) \leq (v'_1, \ldots, v'_n)$ iff $v_i \leq v'_i$ for all i;
8) We assume that the computational cost of f and all fuzzy predicates p_i is bounded by a constant.

We call $s = f(p_1(\mathbf{z}_1), \ldots, p_n(\mathbf{z}_n))$ a *scoring atom*. A *disjunctive query* $\mathbf{q}$ is a finite set of conjunctive queries in which all the rules have the same head.

A Datalog^{topk} program is a pair $\mathcal{P} = \langle \mathcal{P}', \mathbf{q} \rangle$, where $\mathcal{P}'$ is Datalog program, $\mathbf{q}$ is a disjunctive query and q does not occur in $\mathcal{P}$. Essentially, the difference between Datalog and Datalog^{topk} is that now scoring atoms may appearing the rule body of "query rules".

Example 1: Suppose we have information about housings and conferences. We may represent the scenario in Datalog^{topk} as follows. The intentional database is

$$\texttt{Housing}(x) \leftarrow \texttt{Hotel}(x)$$
$$\texttt{Housing}(x) \leftarrow \texttt{Hostel}(x)$$
$$\texttt{Housing}(x) \leftarrow \texttt{Flat}(x)$$

dictating that a hotel, a hostel or a flat can be a housing. The tables below are the extensional database, where we do not report the hotel, hostel and flat tables.

HasHLoc	
ID	HasLoc
h1	hl1
h2	hl2
⋮	⋮

HasCLoc	
ID	HasLoc
c1	cl1
c2	cl2
⋮	⋮

HasHPrice	
ID	Price
h1	150
h2	200
⋮	⋮

Distance		
Loc1	Loc2	Distance
hl1	cl1	300
hl1	cl2	500
⋮	⋮	

Assume we have a fuzzy predicate `close` measuring the closeness degree between housing and conference locations, depending on the distance and fuzzy predicate `cheap`, which given the price determines how "cheap" a housing is. We may ask to find cheap housings close to a conference location, i.e. rank the housings according to their degree of closeness and cheapness. Then we may express our information need using the conjunctive query (`c1` is our conference location)

$$q(h, s) \leftarrow \texttt{HasHLoc}(h, hl), \texttt{HasHPrice}(h, p), \texttt{HasCLoc}(\texttt{c1}, cl), \texttt{Distance}(hl, cl, d), s = \texttt{cheap}(p) \cdot \texttt{close}(d) \ .$$

where the fuzzy predicates `cheap` and `close` are defined as

$$\texttt{close}(\texttt{d}) = \max(0, 1 - \tfrac{d}{1000})$$
$$\texttt{cheap}(\texttt{price}) = \max(0, 1 - \tfrac{\texttt{price}}{300})$$

Note that the scoring function is the product $f(\texttt{cheap}(p), \texttt{close}(d)) = \texttt{cheap}(p) \cdot \texttt{close}(d)$, which is monotone in its arguments `cheap` and `close`. Then we want to retrieve the top-k answers, according to the score s.

Please note that it is not feasible to compute all scores first and then rank them (there may be a huge amount of housings and conference locations).

We expect that the housing identified with `h1` is retrieved with score $s = \texttt{cheap}(150) \cdot \texttt{close}(300) = 0.5 \cdot 0.7 = 0.35$.

Note also that if we would like to find housings, which are either cheap or close to the conference location, then we may use the disjunctive query:

$$q(h, s) \leftarrow \texttt{HasHPrice}(h, p), s = \texttt{cheap}(p)$$
$$q(h, s) \leftarrow \texttt{HasHLoc}(h, hl), \texttt{HasCLoc}(\texttt{c1}, cl), \texttt{Distance}(hl, cl, d), s = \texttt{close}(d)$$

TABLE I

A SLD-REFUTATION.

Goal	θ
$\leftarrow \texttt{TeachesTo}(x, y), \texttt{HasTutor}(y, z)$	$\emptyset$
$\leftarrow \texttt{Professor}(x_1), \texttt{Student}(y_1), \texttt{HasTutor}(y_1, z)$	$\{x/x_1, y/y_1\}$
$\leftarrow \texttt{Professor}(x_1), \texttt{Student}(y_2), \texttt{Professor}(x_2)$	$\{x/x_1, y/y_2, z/x_2\}$
$\leftarrow \texttt{Student}(y_2), \texttt{Professor}(x_2)$	$\{x/\texttt{mary}, y/y_2, z/x_2, x_1/\texttt{mary}\}$
$\leftarrow \texttt{Professor}(x_2)$	$\{x/\texttt{mary}, y/\texttt{john}, z/x_2, x_1/\texttt{mary}, y_2/\texttt{john}\}$
$\leftarrow \square$	$\{x/\texttt{mary}, y/\texttt{john}, z/\texttt{mary}, x_1/\texttt{mary}, y_2/\texttt{john}, x_2/\texttt{mary}\}$

We point out that the above disjunctive query is different from the conjunctive query

$$q(h, s) \leftarrow \texttt{HasHLoc}(h, hl), \texttt{HasHPrice}(h, p), \texttt{HasCLoc}(\texttt{c1}, cl), \texttt{Distance}(hl, cl, d), s = \max(\texttt{cheap}(p), \texttt{close}(d))$$

as in the former we may find housings, which are close to the conference location, though the price is unknown. □

Form a semantics point of view, we have to take into account the additional scoring atom $s = f(p_1(\mathbf{z}_1), \ldots, p_n(\mathbf{z}_n))$.

Informally a conjunctive query

$$q(\mathbf{x}, s) \leftarrow \exists \mathbf{y} body(\mathbf{x}, \mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1), \ldots, p_n(\mathbf{z}_n))$$

is interpreted in an interpretation $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\langle \mathbf{c}, v \rangle$, such that when we substitute the variables $\mathbf{x}$ and s with the constants $\mathbf{c} \in \Delta \times \ldots \times \Delta$ and the score value $v \in [0, 1]$, the formula $\exists \mathbf{y} body(\mathbf{x}, \mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1), \ldots, p_n(\mathbf{z}_n))$ evaluates to true in $\mathcal{I}$.

Due to the existential quantification $\exists \mathbf{y}$, for a fixed $\mathbf{c}$, there may be many substitutions $\mathbf{c}'$ for $\mathbf{y}$ and, thus, we may have many possible scores for the tuple $\mathbf{c}$. Among all these scores for $\mathbf{c}$, we select the highest one, i.e. the sup.

In case the query is a disjunctive query $\mathbf{q}$, for each tuple $\mathbf{c}$ there may be a score v_i computed by each conjunctive query $q_i \in \mathbf{q}$. In that case, the overall score for $\mathbf{c}$ is the maximum among the scores v_i.

Specifically, we assume that the score combination function f and the fuzzy predicates p_i have a given *fixed arithmetic interpretation*.

Now, let $\theta^{\mathbf{cc}'v}_{\mathbf{xy}s} = \{\mathbf{x}/\mathbf{c}, \mathbf{y}/\mathbf{c}', s/v\}$ be a substitution of the variables $\mathbf{x}, \mathbf{y}$ and s with the tuples $\mathbf{c}, \mathbf{c}'$ and score value $v \in [0, 1]$. Let $\psi(\mathbf{x}, \mathbf{y}, s)$ be $body(\mathbf{x}, \mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1), \ldots, p_n(\mathbf{z}_n))$. With $\psi(\mathbf{x}, \mathbf{y}, s)\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ we denote the ground formula obtained by applying the substitution $\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ to $\psi(\mathbf{x}, \mathbf{y}, s)$.

We say that an interpretation $\mathcal{I}$ is a *model* of $\psi(\mathbf{x}, \mathbf{y}, s)\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ iff $\psi(\mathbf{x}, \mathbf{y}, s)\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ evaluates to true in $\mathcal{I}$, i.e. all ground atoms and the grounded scoring atom occurring in $\psi(\mathbf{x}, \mathbf{y}, s)\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ are true. We will write $\mathcal{I} \models \psi(\mathbf{x}, \mathbf{y}, s)\theta^{\mathbf{cc}'v}_{\mathbf{xy}s}$ in this case.

Then, the interpretation $\mathbf{q}^{\mathcal{I}}$ of a disjunctive query $\mathbf{q} = \{q_1, \ldots, q_n\}$ in $\mathcal{I}$ is

$$\mathbf{q}^{\mathcal{I}} = \{\langle \mathbf{c}, v \rangle \in \Delta \times \ldots \times \Delta \times [0, 1] \mid v = \max(v_1, \ldots, v_n), v_i = \sup_{\mathbf{c}' \in \Delta \times \ldots \times \Delta} \{v' \mid \mathcal{I} \models \psi_i(\mathbf{x}, \mathbf{y}, s)\theta^{\mathbf{cc}'v'}_{\mathbf{xy}s}\}\} , \quad (4)$$

where each conjunctive query $q_i \in \mathbf{q}$ is of the form $q(\mathbf{x}, s) \leftarrow \exists \mathbf{y} \psi_i(\mathbf{x}, \mathbf{y}, s)$, $\sup \emptyset$ is undefined, and $\max(v_1, \ldots, v_n)$ is undefined iff all its arguments are undefined.

Note that some tuples $\mathbf{c}$ may not have a score in $\mathcal{I}$ and, thus, $\langle \mathbf{c}, v \rangle \notin \mathbf{q}^{\mathcal{I}}$ for no $v \in [0, 1]$. Alternatively we may define $\sup \emptyset = 0$ and, thus, all tuples $\mathbf{c}$ have a score in $\mathcal{I}$, i.e. $\langle \mathbf{c}, v \rangle \in \mathbf{q}^{\mathcal{I}}$ for some $v \in [0, 1]$. We use the former formulation to distinguish the case where a tuple $\mathbf{c}$ is retrieved, though the score is 0, from the tuples which do not satisfy the query and, thus, are not retrieved.

Finally, for all $\mathbf{c} \in \Delta \times \ldots \times \Delta$ and for all $v \in [0, 1]$, we say that $\mathcal{I}$ is a *model* of $q(\mathbf{c}, v)$ iff $\langle \mathbf{c}, v \rangle \in \mathbf{q}^{\mathcal{I}}$. Also, it is not difficult to verify that, as for Datalog, Datalog^{topk} programs have an unique minimal model $M_{\mathcal{P}}$, which can be obtained as least fixed-point of the $T_{\mathcal{P}}$ operator. We say that a Datalog^{topk} program $\mathcal{P} = \langle \mathcal{P}', \mathbf{q} \rangle$ *entails* $q(\mathbf{c}, v)$, written $\mathcal{P} \models q(\mathbf{c}, v)$, iff $M_{\mathcal{P}} \models q(\mathbf{c}, v)$.

The basic inference services that concerns us is the top-k retrieval problem, where this latter is defined as:

Top-k retrieval: Given a Datalog^{topk} program $\mathcal{P}$, retrieve the top-k ranked tuples $\langle \mathbf{c}, v \rangle$ that instantiate the disjunctive query $\mathbf{q}$ and rank them in decreasing order w.r.t. the score v, i.e. find the top-k ranked tuples of the answer set of $\mathbf{q}$, denoted

$$ans_k(\mathcal{P}, \mathbf{q}) = \texttt{Top}_k\{\langle \mathbf{c}, v \rangle \mid \mathcal{P} \models q(\mathbf{c}, v)\} .$$

For the sake of illustrative purposes, we consider the following abstract example.

Example 2: Suppose the intentional database contains the following rules r_1, r_2 and r_3:

$$\begin{array}{lrcl} r_1: & R(x, y) & \leftarrow & A(x) \\ r_2: & A(x) & \leftarrow & P(y, x) \\ r_3: & P(x, y) & \leftarrow & B(x) . \end{array}$$

We also assume that the extensional database of assertions is stored in the two tables below

P	
0	s
3	t
4	q
6	q

B
1
2
5
7

C
1
3
2
4

Assume our disjunctive query is $\mathbf{q} = \{q', q''\}$ where q' is

$$q(x, s) \leftarrow P(x, y), R(y, z), s = f(\texttt{p}(x)) ,$$

q'' is

$$q(x, s) \leftarrow C(x), s = f(\texttt{r}(x)) ,$$

the scoring function f is the identity $f(z) = z$ (f is monotone, of course), the fuzzy predicate p is $\texttt{p}(x) =$

$\max(0, 1 - x/10)$, and the fuzzy predicate $\mathtt{r}$ is $\mathtt{r}(x) = \max(0, 1 - x/5)$.

Therefore, we can rewrite the query $\mathbf{q}$ as

$$\begin{array}{lcl} q(x,s) & \leftarrow & P(x,y), R(y,z), s = \max(0, 1 - x/10) \\ q(x,s) & \leftarrow & C(x), s = \max(0, 1 - x/5) \ . \end{array}$$

Now, it can be verified that

$$\begin{array}{lcl} \mathcal{P} & \models & q(3, 0.7) \\ \mathcal{P} & \models & q(2, 0.8) \ , \text{ and} \\ \mathcal{P} & \not\models & q(9, v) \ \text{ for any } v \in [0, 1] \ . \end{array}$$

In the former case, the minimal model of $\mathcal{P}$, $M_{\mathcal{P}}$, satisfies $P(3, \mathtt{t})$. But, $M_{\mathcal{P}}$ satisfies the intentional database, and, thus, using the second rule, $M_{\mathcal{P}}$ satisfies $A(\mathtt{t})$. By the first rule, there is some constant c in the Herbrand universe such that $M_{\mathcal{P}}$ satisfies $R(\mathtt{t}, c)$. As a consequence, using q', $M_{\mathcal{P}}$ satisfies

$$q(3, 0.7) \leftarrow P(3, \mathtt{t}), R(\mathtt{t}, c), 0.7 = \max(0, 1 - 3/10)$$

and, thus, the tuple $\langle 3, 0.7 \rangle$ evaluates the body of q' true in $\mathcal{M}_{\mathcal{P}}$. On the other hand, it can be shown that $\langle 3, 0.4 \rangle$ evaluates the body of q'' true in $M_{\mathcal{P}}$. Hence, by Equation 4, the maximal score for 3 is $0.7 = max(0.7, 0.4)$, i.e., $\mathcal{M}_{\mathcal{P}}$ is a model of $q(3, 0.7)$. The other cases can be shown similarly. In summary, it can be shown that the top-4 answer set of $\mathbf{q}$ is $ans_4(\mathcal{P}, \mathbf{q}) = [\langle 0, 1.0 \rangle, \langle 1, 0.9 \rangle, \langle 2, 0.8 \rangle, \langle 3, 0.7 \rangle]$. □

IV. TOP-k QUERY ANSWERING

We discuss now how to determine the top-k answers of a disjunctive query in a Datalog^{topk} program $\mathcal{P}$. In the following we assume that $\mathcal{P}$ is *non-recursive* (we deserve to recursive programs more attention in future work). To this end:

1) By considering the intentional database $\mathcal{P}_I$ of $\mathcal{P}$ only, the user query $\mathbf{q}$ is *reformulated* into a set of conjunctive queries $r(\mathbf{q}, \mathcal{P}_I)$. Informally, the basic idea is that the reformulation procedure resembles the top-down SLD-resolution procedure for logic programming.
2) The reformulated queries in $r(\mathbf{q}, \mathcal{P}_I)$ are *evaluated* over the extensional database $\mathcal{P}_E$ of $\mathcal{P}$ only (which is stored in a database), producing the requested top-k answer set $ans_k(\mathcal{P}, \mathbf{q})$.

So, we start by preparing our Datalog^{topk} program $\mathcal{P} = \langle \mathcal{P}', \mathbf{q} \rangle$ for effective management, where $\mathcal{P}' = \langle \mathcal{P}_I, \mathcal{P}_E \rangle$.

A. Extensional database storage

We first store the data of the extensional database $\mathcal{P}_E$ into a relational database. For each n-ary predicate P occurring in the extensional database $\mathcal{P}_E$, we define a relational table tab_P of arity n, such that $\langle \mathbf{c} \rangle \in tab_P$ iff $P(\mathbf{c}) \in \mathcal{P}_E$. We recall that all facts in $\mathcal{P}_E$ are ground. We denote with $\mathtt{DB}(\mathcal{P}_E)$ the relational database thus constructed.

B. Query reformulation

The query reformulation step is as follows. Consider a conjunctive query

$$q(\mathbf{x}, s) \leftarrow B_1, \ldots, B_{i-1}, B_i, B_{i+1}, \ldots, B_n, \sigma$$

where the atoms B_j are Datalog atoms and σ is the scoring atom. Assume that there is a rule r

$$A \leftarrow A_1, \ldots, A_n$$

which belongs to the intentional database $\mathcal{P}_I$ of $\mathcal{P}$, such that

1) the variables in r have been renominated with new variables;
2) B_i and A unify with the most general unifier θ.

The *resolvent* of r and B_i, denoted $resolve(r, B_i)$, is the conjunctive query

$$q(\mathbf{x}, s) \leftarrow (B_1, \ldots, B_{i-1}, A_1, \ldots, A_n, B_{i+1}, \ldots, B_n, \sigma)\theta \ .$$

We say that the rule r is *applicable* to B_i if B_i and A unify and in $resolve(r, B_i)$ all variables in $\sigma\theta$ occur in $(B_1, \ldots, B_{i-1}, A_1, \ldots, A_n, B_{i+1}, \ldots, B_n)\theta$ as well. Essentially, we do not allow that some variable in the scoring atom is left "unbound" and, thus, the score is undefined.

We are now ready to present the query reformulation algorithm. Given a disjunctive query $\mathbf{q}$ and an intentional database $\mathcal{P}_I$, the algorithm reformulates $\mathbf{q}$ in terms of a set of conjunctive queries $r(\mathbf{q}, \mathcal{P}_I)$, which then can be evaluated over $\mathtt{DB}(\mathcal{P}_E)$.

Algorithm 1 $\mathtt{QueryRef}(\mathbf{q}, \mathcal{P}_I)$

Input: Disjunctive query $\mathbf{q}$, Datalog intentional database $\mathcal{P}_I$.
Output: Set of reformulated conjunctive queries $r(\mathbf{q}, \mathcal{P}_I)$.

```
1: r(q, P_I) := q
2: repeat
3:     S = r(q, P_I)
4:     for all queries q ∈ S do
5:         for all Datalog atoms B_i ∈ q do
6:             if rule r ∈ P_I is applicable to B_i then
7:                 r(q, P_I) := r(q, P_I) ∪ {resolve(r, B_i)}
8: until S = r(q, P_I)
9: return r(q, P_I)
```

This concludes the query reformulation step.

Example 3: Consider Example 2. At step 1. $r(\mathbf{q}, \mathcal{P}_I)$ is initialized with $\{q', q''\}$, where q' is

$$q(x, s) \leftarrow P(x, y), R(y, z), s = f(\mathtt{p}(x)) \ ,$$

and q'' is

$$q(x, s) \leftarrow C(x), s = f(\mathtt{r}(x)) \ .$$

It is easily verified that no rule is applicable to any atom in q''. So we proceed with q'. Let σ be $s = \max(0, 1 - x/10)$.

Then at the first execution of step 7., the algorithm inserts query q_1,

$$q_1 : \quad q(x, s) \leftarrow P(x, y), A(y), \sigma$$

into $r(\mathbf{q}, \mathcal{P}_I)$ using rule r_1 applied to $R(y, z)$.

At the second execution of step 7., the algorithm inserts query q_2,

$$q_2 : \quad q(x, s) \leftarrow P(x, y), P(z, y), \sigma$$

using rule r_2 applied to $A(y)$.

At the third execution of step 7., the algorithm inserts query q_3,

$$q_3 : \quad q(x, s) \leftarrow B(x), P(z, y), \sigma$$

into $r(\mathbf{q}, \mathcal{P}_I)$ using rule r_3 applied to $P(x, y)$.

At the fourth execution of step 7., the algorithm inserts query q_4,

$$q_4: \quad q(x,s) \leftarrow B(x), B(z), \sigma$$

into $r(\mathbf{q}, \mathcal{P}_I)$ using again rule r_3 applied to $P(z,y)$, and stops.

Note that we need not to consider all queries $q_i \in r(\mathbf{q}, \mathcal{P}_I)$. At first, it is easily verified that q_2 can be simplified to

$$q_2: \quad q(x,s) \leftarrow P(x,y), \sigma$$

because whenever $P(x,y)$ holds, also $\exists z.P(z,y)$ holds. Similarly, q_4 can be simplified to

$$q_4: \quad q(x,s) \leftarrow B(x), \sigma$$

because whenever $B(x)$ holds, also $\exists z.B(z)$ holds. These simplifications can easily obtained by checking whether there is an appropriate unifier among the atoms.

At second, it can easily be verified that now for each query q_i and all constants c, the scores of q_2 and q_4 are not lower than all the other queries. That is, we can restrict the evaluation of the set of reformulated queries to $\{q_2, q_4\}$ only. As a consequence, e.g. the top-4 answers to the original query are the tuples $\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.8\rangle$ and $\langle 3, 0.7\rangle$, which are the top-4 ranked tuples of the union of the answer sets of q_2 and q_4. □

C. Computing top-k answers

The main property of the query reformulation algorithm is as follows. It can be shown that

Proposition 1: Let $\mathcal{P} = \langle \mathcal{P}', \mathbf{q}\rangle$ be a Datalog^{topk} program such that $\mathcal{P}' = \langle \mathcal{P}_I, \mathcal{P}_E\rangle$. Then

$$ans_k(\mathcal{P}, \mathbf{q}) = \mathtt{Top}_k\{\langle \mathbf{c}, v\rangle \mid q_i \in r(\mathbf{q}, \mathcal{P}_I), \mathcal{P}_E \models q_i(\mathbf{c}, v)\} \ .$$

The above property dictates that in order to determine the top-k answers, we may reformulate the query $\mathbf{q}$ using the intentional database only, and then query the reformulated queries $q_i \in r(\mathbf{q}, \mathcal{P}_I)$ against the extensional database. From the union of these answer sets we can find the top-k answers.

In the following, we show how to find the top-k answers of the union of the answer sets of conjunctive queries $q_i \in r(\mathbf{q}, \mathcal{P}_I)$.

1) Naive solution: A naive solution to the top-k retrieval problem is as follows: we compute for all $q_i \in r(\mathbf{q}, \mathcal{P}_I)$ the whole answer set

$$ans(q_i, \mathcal{P}_E) = \{\langle \mathbf{c}, v\rangle \mid \mathcal{P}_E \models q_i(\mathbf{c}, v)\} \ ,$$

then we compute the union of these answer sets,

$$\bigcup_{q_i \in r(\mathbf{q}, \mathcal{P}_I)} ans(q_i, \mathcal{P}_E) \ ,$$

order it in descending order of the scores and then we take the top-k tuples.

We note that each conjunctive query $q_i \in r(\mathbf{q}, \mathcal{P}_I)$ can easily be transformed into an SQL query expressed over $\mathtt{DB}(\mathcal{P}_E)$. The transformation is conceptually simple.

A major drawback of this solution is the fact that there might be too many tuples with non-zero score and hence for any query $q_i \in r(\mathbf{q}, \mathcal{P}_I)$, all these scores should be computed and the tuples should be retrieved. This is in practice *not feasible*, as a there may be millions of tuples in the database.

2) Using top-k retrieval engines: A more effective solution consists in relying on existing top-k query answering algorithms for relational databases (see, e.g. [3], [5], [8]), which support efficient evaluations of ranking top-k queries in relational database systems. Though there is no work supporting top-k query answering of disjunctive queries, we can still profitably use top-k query answering methods for relational databases. Indeed, an immediate and much more efficient method to compute $ans_k(\mathcal{P}, \mathbf{q})$ is:

1) we compute for all $q_i \in r(\mathbf{q}, \mathcal{P}_I)$, the top-$k$ answers $ans_k(\mathcal{P}_E, q_i)$, using e.g. the system RankSQL [8] [1];
2) if both k and the number, $n_q = |r(\mathbf{q}, \mathcal{P}_I)|$, of reformulated queries is reasonable, then we may compute the union,

$$U(q, \mathcal{P}) = \bigcup_{q_i \in r(\mathbf{q}, \mathcal{P}_I)} ans_k(\mathcal{P}_E, q_i) \ ,$$

of these top-k answer sets, order it in descending order w.r.t. score and then we take the top-k tuples.

For small k and n_q this solution is already satisfactory. However, we can further improve this solution.

3) The Disjunctive Threshold Algorithm (DTA): As an alternative, we can avoid to compute the whole union $U(q, \mathcal{P})$, so further improving the answering procedure, by relying on a *disjunctive* variant of the so-called *Threshold Algorithm* (TA) [6], which we call *Disjunctive TA* (DTA). We recall that the TA has been developed to compute the top-k answers of a conjunctive query with monotone score combination function. In the following we show that we can use the same principles of the TA to compute the top-k answers of the union of conjunctive queries, i.e. a disjunctive query.

1) First, we compute for all $q_i \in r(\mathbf{q}, \mathcal{P}_I)$, the top-$k$ answers $ans_k(\mathcal{P}_E, q_i)$, using top-$k$ rank-based relational database engine. Now, let us assume that the tuples in the top-k answer set $ans_k(\mathcal{P}_E, q_i)$ are sorted in decreasing order with respect to the score.
2) Then we process each top-k answer set $ans_k(\mathcal{P}_E, q_i)$ ($q_i \in r(\mathbf{q}, \mathcal{P}_I)$) in parallel or alternating fashion, and top-down (i.e. the higher scored tuples in $ans_k(\mathcal{P}_E, q_i)$ are processed before the lower scored tuples in $ans_k(\mathcal{P}_E, q_i)$).
   a) For each tuple $\mathbf{c}$ seen, if its score is one of the k highest we have seen, then remember tuple $\mathbf{c}$ and its score $s(\mathbf{c})$ (ties are broken arbitrarily, so that only k tuples and their scores need to be remembered at any time).
   b) For each answer set $ans_k(\mathcal{P}_E, q_i)$, let s_i be the score of the last tuple seen in this set. Define the threshold value θ to be $\max(s_1, ..., s_{n_q})$. As soon as at least k tuples have been seen whose score is at least equal to θ, then halt (indeed, any successive retrieved tuple will have score $\leq \theta$).
   c) Let Y be the set containing the k tuples that have been seen with the highest scores. The output

[1] RankSQL will be available in the middle of this year. Personal communication.

is then the set $\{\langle \mathbf{c}, s(\mathbf{c})\rangle \mid \mathbf{c} \in Y\}$. This set is $ans_k(\mathcal{P}, \mathbf{q})$.

The following example illustrates the DTA.

Example 4: Consider Example 3. Suppose we are interested in retrieving the top-3 answers of the disjunctive query $\mathbf{q} = \{q', q''\}$. We have seen that it suffices to find the top-3 answers of the union of the answers to q_2 and to q_4. Let us show how the DTA works. First, we submit q_2 and q_4 to a rank-based relational database engine, to compute the top-3 answers. It can be verified that

$$\begin{array}{rcl} ans_3(\mathcal{P}_E, q_2) & = & [\langle 0, 1.0\rangle, \langle 3, 0.7\rangle, \langle 4, 0.6\rangle] \\ ans_3(\mathcal{P}_E, q_4) & = & [\langle 1, 0.9\rangle, \langle 2, 0.8\rangle, \langle 5, 0.5\rangle] \ . \end{array}$$

The lists are in descending order w.r.t. the score from left to right. Now we process alternatively $ans_k(\mathcal{P}_E, q_2)$ then $ans_k(\mathcal{P}_E, q_4)$ in decreasing order of the score.

The table below summaries the execution of our DTA algorithm. The tuple column contains the current processed tuple, while the ranked list column contains the list of tuples processed so far.

Step	tuple	s_1	s_2	θ	ranked list
1	$\langle 0, 1.0\rangle$	1.0	-	1.0	$\langle 0, 1.0\rangle$
2	$\langle 1, 0.9\rangle$	1.0	0.9	1.0	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle$
3	$\langle 3, 0.7\rangle$	0.7	0.9	0.9	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 3, 0.7\rangle$
4	$\langle 2, 0.8\rangle$	0.8	0.7	0.8	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 3, 0.7\rangle, \langle 2, 0.8\rangle$

At step 4. we stop as the ranked list already contains three tuples above the threshold $\theta = 0.8$. So, the final output is

$$ans_k(\mathcal{P}_E, q_3) = [\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.8\rangle] \ .$$

Note that not all tuples have been processed. □

As computing the top-k answers of each query $q_i \in r(\mathbf{q}, \mathcal{P}_I)$ requires (sub) linear time w.r.t. the database size (using, e.g. [3]), it is easily verified that the disjunctive TA algorithm is (sub) linear in data complexity.

Proposition 2: Let $\mathcal{P} = \langle \mathcal{P}', \mathbf{q}\rangle$ be a Datalog^{topk} program such that $\mathcal{P}' = \langle \mathcal{P}_I, \mathcal{P}_E\rangle$. Then the DTA computes $ans_k(\mathcal{P}, \mathbf{q})$ in (sub) linear time w.r.t. the size of $\mathcal{P}_E$.

Furthermore, the above method has the non-negligible advantage to be based on existing technology for answering top-k queries over relation databases, improves significantly the naive solution to the top-k retrieval problem, and is rather easy to implement using current Prolog engines for the query reformulation step.

V. CONCLUSIONS

Deductive databases based on logic programming have a wide application area. We have presented Datalog^{topk} in which fuzzy predicates are allowed to appear in conjunctive queries allowing to express queries such as "find *cheap* hotels". Such queries are very common. To the best of our knowledge, this is the first time this problem has been addressed in deductive databases. A major distinction of Datalog^{topk} is that, an answer to a query is a set of tuples *ranked* according to their score. As a consequence, whenever we deal with a huge amount of tuples, the ranking of the answer set becomes the major problem that has to be addressed.

We have shown how to answer disjunctive queries efficiently over a huge set of instances. The main ingredients of our solution is a simple and effective query reformulation procedure, the use of existing top-k query answering technology over relational databases and the DTA algorithm. Indeed, a user query is reformulated into a set of conjunctive queries using the intentional database only and, then, the reformulated queries can be submitted to the top-k query answering engine over a relational database where the tuples have been stored. Finally, the DTA algorithm performs the final computation to retrieve the actual top-k results.

For future research, we will consider the following issues: (i) To extend top-k query answering to more expressive logic programming languages than Datalog^{topk}; (ii) to improve the DTA by using more sophisticated, but better performing TA-based algorithms such as [7]; and (iii) to improve the core top-k conjunctive query answering technology over relational databases towards the management of the disjunctive queries directly.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts, 1995.

[2] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer Verlag, 1990.

[3] Kevin Chen-Chuan Chang and Seung won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *SIGMOD Conference*, 2002.

[4] E.F. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Courant Computer Science Symposium 6: Database Systems*, volume 3, pages 65–98. Prentice Hall and IBM Research Report RJ 987, 1972.

[5] Ronald Fagin. Combining fuzzy information: an overview. *SIGMOD Rec.*, 31(2):109–118, 2002.

[6] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Symposium on Principles of Database Systems*, 2001.

[7] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kiesling. Optimizing multi-feature queries for image databases. In *The VLDB Journal*, pages 419–428, 2000.

[8] Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. RankSQL: query algebra and optimization for relational top-k queries. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 131–142, New York, NY, USA, 2005. ACM Press.

[9] John W. Lloyd. *Foundations of Logic Programming*. Springer, Heidelberg, RG, 1987.

[10] Raghu Ramakrishnan and Jeffrey D. Ullman. A survey of research on deductive database systems. *Journal of Logic Programming*, 23(2):125–149, 1993.

[11] Kotagiri Ramamohanarao and James Harland. An introduction to deductive database languages and systems. *VLDB Journal*, 3:107–122, 1994.

[12] J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1,2. Computer Science Press, Potomac, Maryland, 1989.

[13] David S. Warren. Memoing for logic programs. *Commun. ACM*, 35(3):93–111, 1992.

[14] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.