Information Systems 55 (2016) 1–19

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Combining user and database perspective for solving keyword queries over relational databases

Sonia Bergamaschi [a], Francesco Guerra [a,*], Matteo Interlandi [b], Raquel Trillo-Lado [c], Yannis Velegrakis [d]

[a] DIEF – University of Modena and Reggio Emilia, Italy
[b] UCLA – University of California, Los Angeles, USA
[c] DIIS – University of Zaragoza, Spain
[d] DISI – University of Trento, Italy

ARTICLE INFO

Article history:
Received 15 June 2014
Received in revised form
18 July 2015
Accepted 21 July 2015
Recommended by: Martin Theobald
Available online 30 July 2015

Keywords:
Keyword search over relational databases
Hidden Markov Models
Dempster–Shafer Theory
Machine learning

ABSTRACT

Over the last decade, keyword search over relational data has attracted considerable attention. A possible approach to face this issue is to transform keyword queries into one or more SQL queries to be executed by the relational DBMS. Finding these queries is a challenging task since the information they represent may be modeled across different tables and attributes. This means that it is needed to identify not only the schema elements where the data of interest is stored, but also to find out how these elements are interconnected. All the approaches that have been proposed so far provide a monolithic solution. In this work, we, instead, divide the problem into three steps: the first one, driven by the user's point of view, takes into account what the user has in mind when formulating keyword queries, the second one, driven by the database perspective, considers how the data is represented in the database schema. Finally, the third step combines these two processes. We present the theory behind our approach, and its implementation into a system called QUEST (QUEry generator for STructured sources), which has been deeply tested to show the efficiency and effectiveness of our approach. Furthermore, we report on the outcomes of a number of experimental results that we have conducted.

1. Introduction

Keyword search has become the de-facto standard for searching on the web. Structured data sources contain a vast amount of information that is significant to be available for querying. Typically, query interfaces consist of web forms that allow predefined queries to be posed on their contents. Besides, web search engines index the content of these sources (the so-called hidden web) through the results of these web form queries, seen as free text. Apart from the fact that this restricts the kind of data that can be searched, the great deal of semantic information provided by the structure of the data, e.g., the schema, is basically lost. This gave rise to a special interest in supporting keyword search over structured databases [1] in ways that are as effective as those offered on text data and at the same time exploit as much as possible the structure of the data that databases provide.

Many approaches exploit full-text search functionalities natively implemented in the DBMS, such as the `contains` function in SQL server and the `match-against` function in MySQL, to discover the attributes of the database containing

* Corresponding author.
E-mail addresses: sonia.bergamaschi@unimore.it (S. Bergamaschi), francesco.guerra@unimore.it (F. Guerra), minterlandi@cs.ucla.edu (M. Interlandi), raqueltl@unizar.es (R. Trillo-Lado), velgias@disi.unitn.eu (Y. Velegrakis).

http://dx.doi.org/10.1016/j.is.2015.07.005
0306-4379/

the query keywords at run-time. Then, they construct the answer set by combining tuples containing different query keywords and selecting those combinations considered most likely to be what users were looking for [2–12]. All these approaches are typically heuristic-based, without a clear specification of the steps required to answer the keyword query. In this work, we advocate that there is a need for a more principled approach to keyword searching on structured data; in particular, we believe that keyword search on structured sources requires three fundamental steps. Existing works consist of either a monolithic end-to-end solution that provides no clear distinction of these three steps, or are focused on only some of them, considering some straightforward implementation of the remaining.

The three fundamental steps we consider are first to match the keywords to the database structures, then to discover ways these matched structures can be combined, and finally to select the best matches and combinations such that the identified database structures represent what the user had in mind to discover when formulating the keyword query. The first step is focused on trying to capture the meaning of the keywords in the query as they are understood by the user, and express them in terms of database terms, i.e., the metadata structures of the databases. In some sense, it provides the *user perspective* of the keyword query and it does so by providing a mapping of the keywords into database terms. This step is referred to as the *forward analysis step* since it starts from the keywords and moves towards the database. The second step tries to capture the meaning of the keywords as they can be understood from the point of view of the data engineers who designed that database organization, and express them in semantically coherent units of database structures containing the images of the keywords specified by the first step. So, in some sense, it provides the *database perspective* of the keyword query and it does so by providing the relationships among the images of the keywords. This task is referred to as the *backward analysis step* since it starts from the database structures and moves towards the query keywords through their images. The third step provides a ranking of the coherent units of database structures that the second step produced after selecting those that are more promising, i.e., those whose semantics more likely express what the user had in mind while was formulating the keyword query.

In our previous works we have studied different aspects of the keyword search problem over relational databases. The KEYMANTIC [13,14] system was focused on the first step. It provided a solution based on a bipartite graph matching model where user keywords were matched to database schema elements by using an extension of the Hungarian algorithm. KEYMANTIC is one of the first solutions that deals with the problem of querying structural databases through keywords when there is no prior access to the database content to build any indexes, thus, relying on semantic information of the database meta-data. This feature of KEYMANTIC makes it especially appropriate for keyword-based search on federated database systems and for exploring data sources in the hidden web. KEYRY [15,16] extended KEYMANTIC by providing a probabilistic framework, based on a HMM, to match keywords into database schema elements. Both works deal with the first step of the process described previously, i.e., the user perspective step. Our experience with these systems made clear that this was not enough for a complete solution. These systems were the motivation for the principled, holistic and unified framework presented in this work.

The main contributions of the current paper are the following: (i) we introduce a principled 3-step model for the keyword search problem over structured databases; (ii) we develop two different implementations of the first step, one that exploits heuristic rules and one that is based on machine learning techniques. Both aim at finding the appropriate Hidden Markov Model specification to generate the right mapping of the query keywords into database structures; (iii) we define an implementation of the second step based on Steiner Tree discovery which exploits a mutual information-based distance as edge weight and which works at the schema level instead of the instance level; (iv) we provide a probabilistic framework founded on the Dempster Shafer Theory that is able to combine the first two steps and modalities in a way that permits the system to promptly adapt to different working conditions by selecting the best combination among them; (v) we implement all the above in a system called QUEST (QUEry generator for STructured sources) [17] and provide the details of its implementation; and finally (vi) we perform an extensive set of experiments that offer a deep understanding of the whole process, its effectiveness and efficiency.

The remainder of the paper is as follows. First, the principled 3-step approach is introduced and our proposed framework is formally defined in Section 2. The implementation of each of the three steps in our developed QUEST prototype follows in Section 3. The relationship of our framework with the related works alongside our own previous works on the topic is explained in Section 4. Finally, the results of our extensive experimental evaluation are discussed in Section 5.

2. The three-step framework

As a data model for the structured database we assume the relational model, however the framework can be easily extended to other structured models as well.

We assume an infinite set $\mathcal{A}$ of attribute names, $\mathcal{R}$ of relation names, and $\mathcal{V}$ of value domains. A *tuple* is a finite set of attribute name–value pairs $\langle A_1{:}\,v_1, A_2{:}\,v_2, \ldots, A_n{:}\,v_n\rangle$ where $A_i \in \mathcal{A}$, $v_i \in V_i$ with $V_i \in \mathcal{V}$, for $i=1\ldots n$, and $A_i \neq A_j$ if $i \neq j$. The *schema of the tuple* is the $\langle A_1{:}\,V_1, A_2{:}\,V_2, \ldots, A_n{:}\,V_n\rangle$ and its *arity* is the number n. The domain V_i is referred to as the domain of the attribute A_i and will be denoted as $Dom(A_i)$, for $i=1\ldots n$. A *relation instance* is a finite set of tuples, all with the same schema. The *schema of the relation instance* is the common schema of its tuples and its *cardinality* the number of tuples it consists of. A *relation* is a pair $\langle R, I_R\rangle$, where $R \in \mathcal{R}$, referred to as the *relation name*, and I_R is a relation instance. The *schema* of a relation $\langle R, I_R\rangle$ is the schema of its relation instance, and will be denoted as $R(A_1{:}\,V_1, \ldots, A_n{:}\,V_n)$, where the $\langle A_1{:}\,V_1, \ldots, A_n{:}\,V_n\rangle$ is the schema of the relation instance I_R. In what follows, whenever there is no risk of confusion, the name R will be used to refer to the whole relation $\langle R, I_R\rangle$. Furthermore, the indication of the domains will be omitted leading to the simplified expression of the relation schema as $R(A_1, A_2, \ldots, A_n)$. Finally, the notation $|R|$ will denote the arity of the relation R and the $|I_R|$ the cardinality of its relation instance [18].

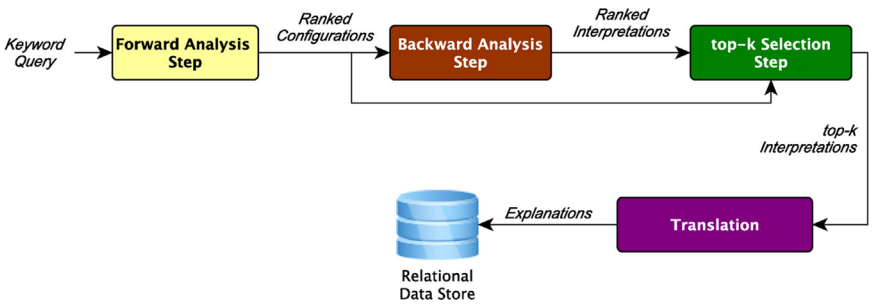

Fig. 1. Process steps for answering a keyword query over a relational database.

A *database* is a set of relations. The schema and instance of the database is the set of schemas and instances, respectively, of its relations. The *terminology* of a database is the set consisting of the names of its relations, their attributes and the domains of these attributes. In other words the terminology of a database D is the set $T_D = \{t | \exists R(A_1, A_2, \ldots, A_n) \in D \wedge \quad (t = R \vee t = A_i \vee t = \quad Dom(A_i)) \wedge \quad 1 \leq i \leq n\}$. The members of the terminology are referred to as *database terms*. The terminology of a database D has $2 * \Sigma_{i=1}^{|D|} arity(R_i) + |D|$ elements. Note that the model assumes that every attribute has its own domain. This, however, does not mean that the domains have to be disjoint. Two domains may even have the same content. So, such an assumption does not restrict the model, however, it offers a great flexibility in the materialization of the model in a real system.

A keyword query q is an ordered list of keywords $(k_1, k_2, \ldots, k_m)$. It is important to note that a keyword does not always mean a single word. A keyword may be a combination of words that together form a unit and should be treated as such, like for instance "United States" or "Barack Obama". Two or more words are considered as one keyword if they appear together as a value in one of the domains V defined earlier. For instance, the term *United States* is one of the values of the domain *Countries*, so when the two words are given together by the user, the tokenizer will consider them as one keyword.

2.1. Mapping keywords into database structures

The first step in answering a keyword query over a relational database is to understand the meaning of the keywords used in it. Each keyword describes some characteristic property of the elements of interest that the user has in mind. This property has to be expressed in terms of characteristics as they are modeled in the database through the database structures. Thus, understanding what each keyword means requires each keyword to be mapped into the database terminology so that it can be expressed with some database term. Such a mapping is referred to as a *configuration*.

Definition 2.1. A *configuration* c of a keyword query q on a database D is an injective function from q to the terminology T_D, i.e., $c|q \rightarrow T_D$. The *image* of a keyword k through the configuration c is a database term t for which $c(k) = t$.

The configuration has been defined as a function, because we have made the natural assumption first that each keyword is present in the query for a reason, i.e., no redundant or unjustified keywords, and second, that there is no intentional ambiguity, i.e., a keyword cannot serve more than one meaning in a query. For every meaning that the user has in mind, it is assumed that a different keyword has been selected and included in the query to represent it. The reason that the function is considered *injective*, on the other hand, is two-fold. First, we assume no over-specification, i.e., no two keywords are referring to the exact same thing. Second, we need to avoid the case of self-joins because this may lead to an infinite number of cases to be considered. Mapping of two keywords on the same term with a cyclic join path, may generate interpretations in which the path is considered arbitrary number of times, leading to an infinite number of possible interpretations. This choice is in line with similar mapping situations like schema mapping [19,20] where the chase technique used there generates infinite mappings if self joins or cycles exist.

A keyword query of m keywords, has $|T_D|!/(|T_D| - m)!$ possible configurations but not all of them are equally likely to represent the meaning that the user had in mind while formulating the keyword query. Determining the possible configurations and ranking them from the most likely to the less likely constitutes the first (*forward analysis*) step of the process of answering the keyword query (see Fig. 1).

The forward analysis step needs to take into consideration not only the actual keywords used in the query, but also their relative positions and try to guess their meaning by exploiting this information. There are studies that show that queries generated by users are typically having related keywords close to each other [21] and that the order follows some logical sequence. The forward analysis step is considered to provide the *user perspective* because it tries to model what the user had in mind with the individual keywords.

Example 2.1. Consider a university database with information on personnel, projects and research activities, a fragment of which is illustrated in Fig. 2. Assume that a user poses the query `Vokram IT`. Both `Vokram` and `IT` could represent any database structure, however, it is more likely for the former to be the name of a person. Similarly, the latter may be the name of a university, but most likely it is the name of a country. Thus, the configuration

A: {"`Vokram`" $\rightarrow$ *Dom*(*People.Name*), "`IT`" $\rightarrow$ *Dom*(*University.Country*)}

is more likely to be the one the user was thinking compared to the

B: {"`Vokram`" $\rightarrow$ *Dom*(*People.Name*), "`IT`" $\rightarrow$ *Dom*(*University.Name*)}

PEOPLE

Id	Name	Phone	Country	Email
p1	Vokram	4631234	USA	vkm@aaa.bb
p2	Reniets	6987654	IT	rts@bbb.cc
p3	Refahs D.	1937842	SP	ds@ccc.dd

AFFILIATED

id_prs	id_dpt	Year
p1	x123	2009
p2	cs34	2012
p3	cs34	2010

DEPARTMENT

id	Name	Address	University	Director
x123	CS	25 Blicker	SU	p122
cs34	EE	15 Tribeca	UM	p54
ee67	ME	5 West Ocean	UTN	p432

UNIVERSITY

Name	City	Country
MIT	Cambridge	USA
UR	Rome	IT
UTN	Trento	IT
SU	Stanford	USA

MEMBER OF

Person	Projct	Date
p1	Rx1	5/4/2012
p2	Rx1	9/3/2012

PROJECT

id	Name	Year	Topic
Rx1	Search it!	2011	DB&IR
Rt1	Analyze it!	2012	DB&ML

PARTICIPATION

Project	University
Rx1	UR
Rx1	UTN
Rt1	UM

Fig. 2. A fragment of a database schema with its data.

Obviously, in case of adoption of full-text indexes the number of possible configurations is reduced to the ones allowed by the database instance. In the case in Fig. 2, "Vokram" can be only the name of a person. "IT" can be a value associated to the country attribute or to the People or University tables.

2.2. Combining the identified database structures

The configurations are semantically ambiguous. They may describe the meaning of the keywords in database terms, but they do not explain how the terms are connected to form a coherent semantic unit that gives a semantic meaning to the whole keyword query. This connection has to be based on the way the images of the query keywords (as they are expressed through the configurations) are connected in the database.

There are typically two main ways database terms are connected. One is the structure, i.e., the way the data administrator has chosen to model the data in the repository. For instance, two attributes are placed in the same relation when the data designer believes that they describe two different properties of the concept that the relation is about, and consequently they should be connected. The other way is the use of schema constraints, in particular referential constraints like key/foreign key relationships. These relationships describe ways in which structures in different relations can be associated by forming join paths. We refer to the ways that the database terms that serve as images of the query keywords can be associate as *interpretations* because they do not simply indicate what each keyword represents, but they also provide an interpretation of the whole keyword query in terms of the database structures and semantic constraints.

To more formally define interpretations we introduce the notion of the *database graph*.

Definition 2.2. A *database graph* $G = \langle V, E \rangle$ is a graph structure where each node represents a database term (i.e., a relation, an attribute or a domain of an attribute). There is an edge between each attribute of a relation and its domain, and between each attribute and the table (relation) it belongs. Furthermore, there is an edge between the nodes that model the domains of two attributes that have an inclusion dependency (e.g., a key/foreign key relationship).

For simplicity, we assume no primary or foreign key is spanning multiple attributes. Nevertheless, this is not limiting QUEST since a surrogate key can always be created if a multi-attribute primary/foreign key exists.

Example 2.2. Fig. 3 illustrates the database graph of the database instance of Fig. 2. Nodes of the form *Dom*(*A*) represent the domain of attribute *A* and nodes in bold represent relations. The dotted lines represent referential constraints and the solid lines attribute-relation or attribute-domain relationships.

Using the database graph, the interpretations can be defined as sub-graphs of it.

Definition 2.3. Given a configuration *c* of a keyword query *q* on a database $\mathcal{D}$, an *interpretation* $\langle V^S, E^S \rangle$ is the connected component sub-graph of *G* composed by every node modeling the image $c(k)$, of a keyword *k* in the query *q*, and every other node or edge that is part of a path connecting two such image nodes, in a way that between any two image nodes there is one and only one path.

Note that since there may be multiple join paths connecting two attributes in the database, it is natural that given a configuration there may be more than one interpretation. Each interpretation describes some different semantics for the keyword query that the user provided.

Example 2.3. Consider again the configuration

A: {"`Vokram`" → *Dom*(*People.Name*), "`IT`" → *Dom*(*University.Country*)}

of Example 2.1. Looking at the database graph of Fig. 3 different interpretations can be found. One may be that `Vokram` is the director of a department of an Italian university, specified by a subgraph that connects the nodes[1] *Dom*(*People.Name*) and

[1] Although in the graph we use only the expression *Dom*(*A*) as a label on the node modeling the domain of the attribute A of a relation R, in order to avoid confusion in the text we instead write *Dom*(*R.A*).

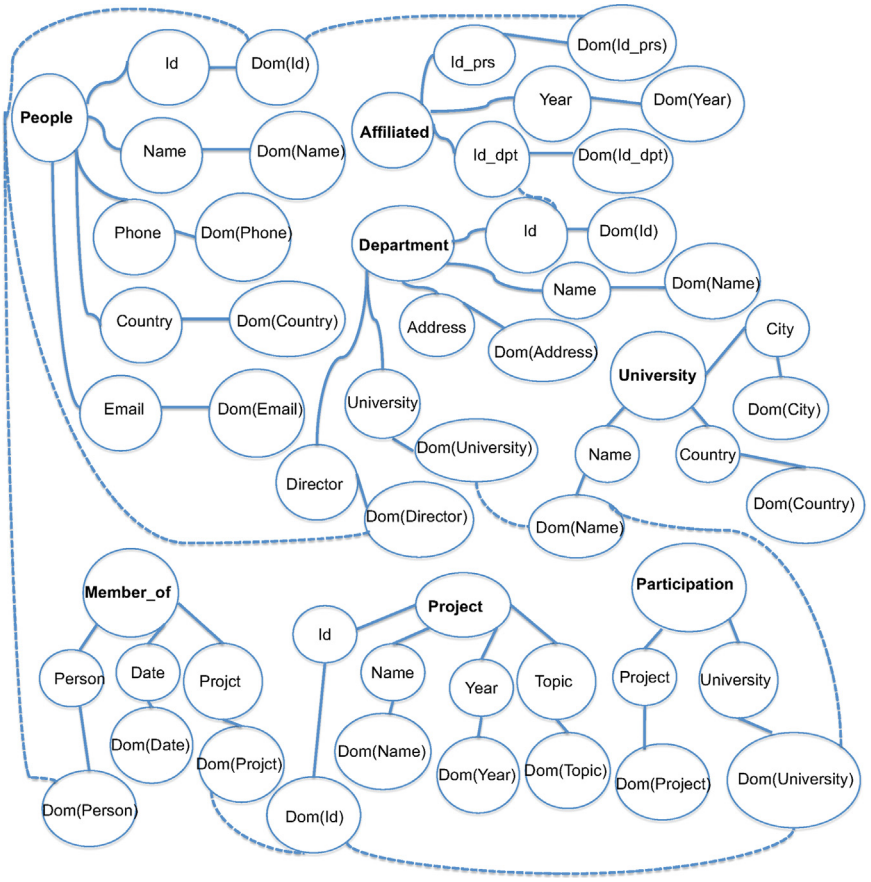

Fig. 3. The database graph corresponding to Fig. 2.

Dom(*University*.*Country*) through the DEPARTMENT relation. The specific interpretation is illustrated at the top of Fig. 4. Let it be referred to as interpretation [A.1]. Another interpretation, let it be referred to as [A.2] is the one that assumes that `Vokram` is the name of a person that is a member of a project in which an Italian university participates. This interpretation connects the nodes *Dom*(*People*. *Name*) and *Dom*(*University*.*Country*) through the relations MEMBEROF, PROJECT and PARTICIPATION as the middle graph in Fig. 4 illustrates.

For what concerns the configuration

B: {"`Vokram`" → *Dom*(*People*.*Name*), "`IT`" → *Dom*(*University*.*Name*)}

one possible interpretation is that `Vokram` is the director of a department of a university called `IT`, illustrated under the name [B.1] in Fig. 4.

Since this second step of providing some semantic interpretation of the configuration is driven by the way the data is structured in the database, it is referred to as the *backward analysis step*. Intuitively, it provides the "*database perspective*" on what keywords that the user used in the query may mean.

Of course not all the interpretations are equally likely to represent what the user had in mind, so they should be ranked according to the likelihood that they do so. In order to estimate this likelihood, different methods can be used. For instance, a ranking can favor interpretations with the shorter paths between nodes, or with paths passing through a specific central node as happens in star schemas of warehouses.

Moreover, there is a second dimension to take into account in ranking the interpretations: the existence in the database instance of tuples corresponding to the selected interpretation. Not all the interpretations have the same information power and some of them can produce empty results once they are translated in SQL and executed in the DBMS. Estimating if an interpretation could provide an empty result is useful to optimize the process and make it more responsive towards the user's needs.

Example 2.4. One of the heuristic rules commonly adopted for ranking the interpretations is based on the number of edges involved. Interpretations with more edges can include extra edges which are no justified by any term in the user query and relate things that are semantically far. Among the two interpretations [A.1] and [A.2] mentioned in Example 2.3, [A.2] has more edges. Although it is possible for [A.2] to actually represent the semantics that the user is looking for with the provided keyword query, it is less likely since [A.2] involves elements that are semantically further from the point of view of the database designer.

Moreover, when evaluated against the database, not all the interpretations correspond to instances which are really available. For instance, this is the case of [A.1] (i.e., the

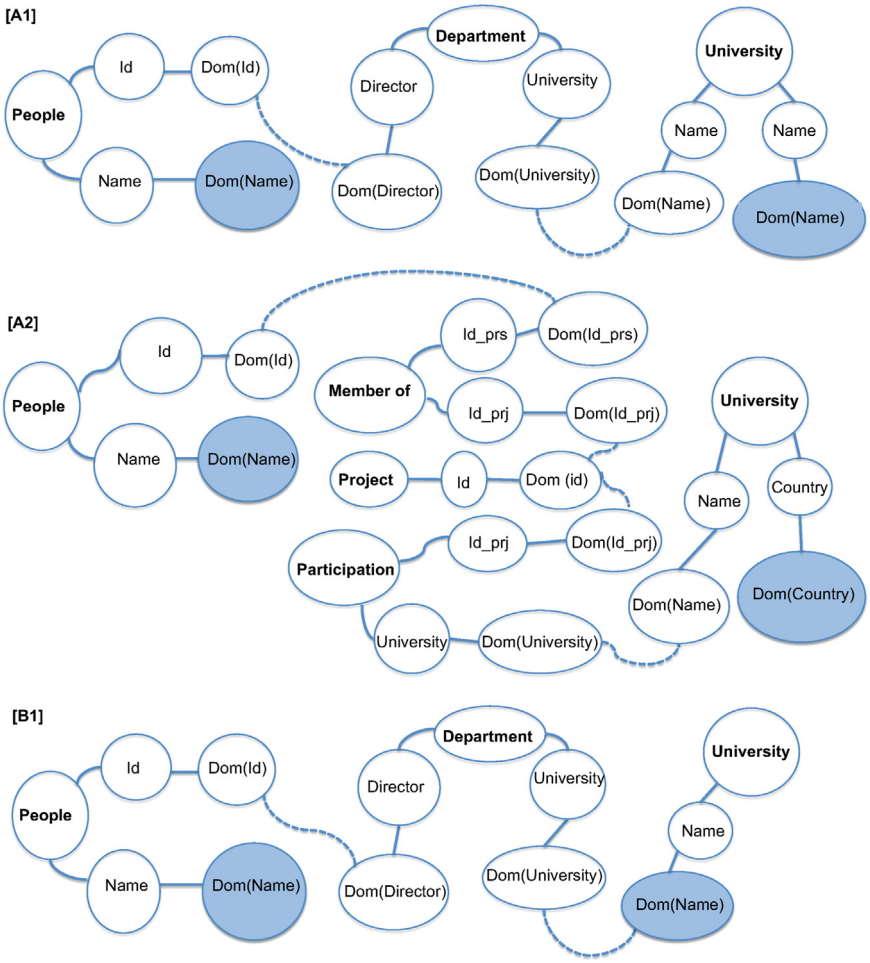

Fig. 4. Interpretation examples of the configurations A and B of Example 2.1.

person "Vokram" is not the Director of a Department of a University from "IT").

2.3. Producing the explanations

Since the data is stored in a relational data store, to retrieve the elements of interest it is needed to produce a number of SQL queries. We refer to these queries as *explanations* since they are actually describing a set of data to be retrieved as a response to the keyword query provided by the user, and in some sense "explain" what the query could actually have meant. How the explanations are generated is an issue of the specific implementation. However, what is important is that the final sql query respects the configuration, i.e., ensures that the images of the keywords as database terms are present in the query and these terms are associated in the way that the interpretation specifies.

Naturally, not all the explanations are equally likely to represent the intention that the user had in mind when formulating the query. The likelihood that an explanation is actually representing such an intention is based on the degree that both the configuration and the interpretation are believed to represent what the user had in mind. This means that to create a ranked list of the most promising candidate explanations, one needs first to create a ranked list of interpretations that takes into consideration not only the ranking of the interpretations produced by the backward analysis step, but also the ranking of the configurations from which they were derived, as produced by the forward analysis step. It may be the case for instance, that an interpretation ranks very high in the list produced by the second step, but the configuration from which the interpretation was derived is very low in the rank of configurations produced by the first step.

Example 2.5. Consider again the case of the keyword query `Vokram IT`, the configurations A and B mentioned in Example 2.1 and the three interpretations [A.1], [A.2] and [B.1] described in Example 2.4 and illustrated in Fig. 4. [A.1] is preferable compared to [A.2], according to the usual heuristic rule, even if it does not correspond to any tuple in the database as already mentioned. However, between [A.1] and [B.1] the decision is not that clear since the graphs of the two interpretations are both of the same size, as it can be seen in Fig. 4. It was mentioned in Example 2.1, however, that the configuration *A*, compared to *B*, is more likely to represent what the user had in mind, which makes an explanation derived by [A.1] more

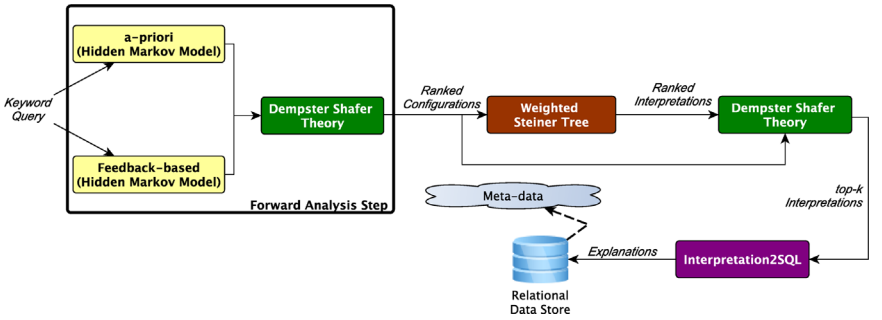

Fig. 5. The keyword search process in QUEST.

favorable than one derived by [B.1]. Whether an explanation derived by [B.1] is preferred over one derived by [A.2], depends on whether the mappings of the keywords into the database structures (i.e., the configuration) are considered more important than the way these structures are related to each other (i.e., the interpretation).

3. Framework implementation

We have materialized the previously described framework into a system called QUEST. The system can be used as an add-on that operates on top of a database system. Before operating, QUEST needs to know some meta-data information about the database. The meta-data information consists of the database terms alongside the referential constraints. This is done into a pre-processing step by accessing the database catalog tables. It also needs access to the full-text indexes over all the database attributes. Of course, there are cases in which such access is not possible. One such case is the one in which the data source is part of an integration system of independent sources. Typically these sources do not allow unrestricted full access to their content but only access to specific parts through a controlled interface. In these cases some partial information can be obtained from the user or by analyzing the interface that the database provides.

The overall framework implementation process is illustrated in Fig. 5. As it can be seen, for the forward analysis there are two different implementations that run on parallel and at the end their results are merged into one set of configurations. The configurations are provided to the backward step implementation, which takes one at a time and produces a set of possible interpretations. All the generated interpretations are then ranked according to their selection criteria and the ranked list is provided to the ranking module. The latter combines the ranked interpretations with the ranked configurations to produce a new ranked set of interpretations and select the top k. At the end, each one in the top-k is translated into a SQL query.

Algorithm 1. QUEST implementation of the keyword query answering.

Input: Keyword query: q
Number of top results to consider: k
Output: Set of SQL queries Q_{sql}
QUEST()
(1) $C_{apr} \leftarrow$ FW − HMMA−PRIORI(q, k)
(2) $C_{fdback} \leftarrow$ FW − HMMFEEDBACK(q, k)
(3) $TopC \leftarrow$ COMBINERDST(q, k, C_{apr}, C_{fdback}, $Conf_{apr}$, $Conf_{fdback}$)
(4) $I \leftarrow \emptyset$
(5) **foreach** $c \in TopC$
(6) $I \leftarrow I \cup$ BW − ST(q, c)
(7) $TopI \leftarrow$ RANK$_{ST}$(I)
(8) $TopI \leftarrow$ COMBINERDST(q, k, $TopC$, $TopI$, $Conf_{TopC}$, $Conf_{TopI}$)
(9) $Q_{sql} \leftarrow$ CONFIGURATIONS2SQL($TopI$)
(10) **return** Q_{sql}

3.1. Discovering configurations: the forward analysis implementation

A number of different techniques can be used to generate the possible mappings of the keywords to database terms, i.e., the configurations. If full access to the database content is available in advance and full-text search functions are supported, they can be exploited [2] to identify the appearance of the keywords in the content, and from there, the database structures these appearances belong. However, in many cases, this is not enough, since the same keywords can be found in several database relations, thus requiring the adoption of disambiguation techniques. Moreover, especially on the web, access to the data is not always available. Data sources expose typically a part of their schema that can be queried, either through wrappers or through web form interfaces. For these cases, semantic and heuristic techniques can be exploited. Existing works have shown to produce interesting results [13].

Since the process is mainly a guess on what the keywords in the query actually mean, each configuration proposed should be coming with some degree of confidence. In our own implementation, we use a First-Order Hidden Markov Model (HMM), similar to the one we presented in a previous work [15]. This approach has the advantage of using a solid and effective probabilistic framework and works as follows.

A HMM models a stochastic process that is not observable directly (it is hidden), but it can be observed indirectly through the observable symbols produced by another stochastic process. The model is composed of a finite number N of states. Assuming a time-discrete model, at each time step a new state is entered based on a *transition probability distribution*, and an observation is produced according to an *emission probability distribution* that depends on the current state, where both these distributions are time-independent. Moreover, the process starts from an initial state based on an *initial state probability distribution*. More formally, the First-Order HMM consists of: (i) a set os states $S = \{s_i\}$, $1 \le i \le N$; (ii) a set

of observation symbols $V=\{v_j\}$, $1\le j\le M$; (iii) a transition probability distribution $A=\{a_{ij}\}$, $1\le i\le N$, $1\le j\le N$ where

$$a_{ij}=P(q_{t+1}=s_j|q_t=s_i) \quad \text{and} \quad \sum_{0<j<N} a_{ij}=1;$$

(iv) an emission probability distribution $B=\{b_i(m)\}$, $1\le i\le N$, $1\le m\le M$ where

$$b_i(m)=P(o_t=v_m|q_t=s_i) \quad \text{and} \quad \sum_{0<m<M} b_i(m)=1$$

and (v) an initial state probability distribution $\Pi=\{\pi_i\}$, $1\le i\le N$ where

$$\pi_i=P(q_1=s_i) \quad \text{and} \quad \sum_{0<i<N} \pi_i=1$$

In our context, the keywords inserted by the user are the observable part of the process, while the correspondent database terms are the unknown variables that have to be inferred. For this reason, we model the keywords as observations and each term in the database vocabulary as a state.

The use of HMM requires the specification of its transition, emission and initial state probabilities. The challenging question is how these probabilities can be computed. We have developed two operating modes for the forward analysis step, each one has its own method for the computation of these distributions. These modes are based on techniques we have previously developed and implemented [15,16] for computing the transition probability and the initial state distributions. The first mode, is referred to as the "feedback-based". In that mode the system learns these probabilities by means of an Expectation-Maximization (E-M) on-line training algorithm [22] applied to a dataset composed of previous searches validated by the user. The second mode is referred to as the "a priori" mode. In that mode, the transition probabilities are defined by exploiting semantics collected from the data source metadata independently of the feedback from the users. They are computed by using heuristic rules that take into account the semantic relationships that exist among the database terms. The goal of these rules is to foster the transition between database terms belonging to the same table and belonging to tables connected through foreign keys. This means that in the A matrix the weights are established so that the values associated to:

- database terms belonging to the same table assume the highest values;
- database terms belonging to tables connected via foreign keys assume intermediate values;
- database terms belonging to table not directly connected assume the lowest values.

In all the above cases, the values are further differentiated on the basis of which kinds of database terms they are representing. In particular, transitions between states representing schema elements and their respective domains are assigned higher values than those assigned to the transitions between states associated to schema elements, states associated to attribute domains and finally, states associated to schema elements and domain attributes of not related attributes. The choice of the actual values are not important as long as they satisfy the above rules. In all the cases, the values have to be normalized to add up to one.

The initial state probability is computed by means of an adaptation of the HITS algorithm [23] that applied to a HMM retrieves the states with high "authority" scores, i.e., the ones that contain valuable information concerning the user query, thus associated with a higher initial state probability. The above two techniques (details of which can be found in some previous work of ours [15,16]) are extended and adapted in the QUEST environment. In particular, for both operating modes, the emission probabilities are computed on-the-fly, by analyzing the keyword query with the full-text index-based search function usually available in today DBMSs. The emission probability of a specific state is approximated by means of the scores returned by the full-text search function applied to all the keywords with respect to the considered state, and normalized to add to one. We use the calculated similarity as an estimate for the conditional probability $P(q_t=s_i|o_t=v_m)$ then, using the Bayes theorem, we calculate the emission probability $P(o_t=v_m|q_t=s_i)$. If we do not have a complete access to the database instance, we can adopt different measures to estimate the emission probability, by means of regular expressions and the "Semantic Distance" as we proposed in [14]. Note that the model is independent of the similarity measure adopted. Other more complex measures that take into account external knowledge sources (i.e., public ontologies and thesauri) can be applied without modifying the model.

Once the HMM parameters have been defined, the resulting HMM is used to compute the top-k configurations by applying the *List Viterbi* algorithm [16]. The algorithm returns the ordered list of the K state sequences $\hat{Q}_l^k=(q_1^k,q_2^k,\ldots,q_T^k)$, $1\le k\le K$ (i.e., the database terms) which have the highest probability of generating the sequence $O_l=(o_1,o_2,\ldots,o_T)$ (i.e., the user keywords).

Example 3.1. Let us consider a fragment of the database proposed in Fig. 2, composed only of the tables People and Department. The HMM (which has been constructed in advanced as explained previously) contains a state for each database term, i.e., a state for each attribute and attribute domain in the tables. If we assume to have access to full-text indexes, given a set of keywords, we know the database terms that more likely represent them. This is the emission probability distribution B. The transition probability matrix A, built according to one of the two techniques described above, expresses the probability that a keyword is assigned to a specific state, i.e., representing the respective database term for that state, given the assignment of the preceding keyword in the query. Now, given for example the keyword query `Country IT`, assuming that the first term is associated to the schema element Country of the table People, the probability to have the second term "IT" associated to the domain of the attribute "Country" is higher than any other assignment, if in matrix A the value associated to the transition between the states representing the database terms Country and domain of Country is higher than any other value associated to transitions from the "Country" state. The List Viterbi algorithm takes into account the transition and the emission probability distributions for returning a list of

possible state sequences, which are likely to represent the database terms that can be associated to the user keywords. In the specific case, the possible database terms associated to "Country" and "IT".

3.2. Discovering interpretations: the backward analysis implementation

Finding how the images of the keywords to the database terms, as they are specified by the configurations, are connected to each other dictates the way the data has been modeled in the database. For a relational system, this would translate into an exploration of the join paths among their respective tables and selecting those that most likely represents what the user that formulated the query had in mind. To do so it is useful to see the database as a *database graph* (see Definition 2.2).

Selecting the best join path is a challenging issue. Between two terms, the most natural selection would be the join path with the shortest length based on the idea that the closer two terms are, the higher the likelihood that they are semantically related. When there are more than two terms, the way they can be connected is more complex and the semantics of the shortest path is not so clear. For this reason, the idea of the Steiner Tree [24] is often adopted: given an edge-weighted graph $G=\langle V,E\rangle$ and a subset $V_q \subseteq V$, find the top-k minimum cost trees containing at least all the elements of V_q. For our case, the nodes V_q are the database terms that serve as images of the query keywords through the configuration. The cost of a Steiner Tree $t: \langle V_t, E_t\rangle$ is the sum of the weights of its edges, i.e.,

$$cost(t) = \sum_{e \in E_t} weight(e) \tag{1}$$

where $weight(e)$ is the weight of the edge $e \in E_t$

Since the Steiner Tree discovery process is about a weighted graph, QUEST employs $weight(e)=1$ for "attribute-domain of the attribute" edges and "relation-attribute of the relation" edges, and a mutual-information-based distance for computing the rest of weights on the edges (previously used in the context of databases summarization [6]). This assures to select the most informative join-paths, i.e., those that are more likely to contain tuples that can join. A similar measure has been adopted in the context of database summarization [25]. Consider two attributes A_1 and A_2 that appear in the same Steiner Tree, A_1 is a primary key, and A_2 is a foreign key referencing A_1. If $I(A_1,A_2)$ and $H(A_1,A_2)$ are respectively the *mutual* information and the *entropy*, as defined elsewhere [25], the weight of the edge between them can be defined as the distance function

$$D(A_1,A_2) = 1 - \frac{I(A_1,A_2)}{H(A_1,A_2)} \tag{2}$$

To define and apply the distance function, the works by Yang et al. [25] are considered to define a *joint distribution*[2] between two variables, X_{A_1} and X_{A_2}, that represent the attributes A_1 and A_2, respectively. In particular, if A_1 and A_2 belong to different relations, the joint distribution between X_{A_1} and X_{A_2} is defined by taking into account the full outer join on the attributes A_1 and A_2 as in this way, the distance function is aware of pairs of the types (*value*, *null*) and (*null*, *value*). If A_1 and A_2 belong to the same relation, then the joint distribution is the projection of the relation on $\{X_{A_1}$ and $X_{A_2}\}$.

Once this distribution gets computed and the distance function mentioned above calculated, its value is divided among the two edges that connect the two attributes through their common relation and become their respective weights. In practice, in order to further optimize the execution time, we are actually using a summary graph [25] instead of the whole Steiner Tree. The summary graphs are similar to the graph previously described, but their edges are meta-edges corresponding to paths of the original graph (in our case the Steiner Tree) so we do not really have to do this division of the weights. The fundamental concept, however, remains the same even in the case of the summary graphs. Given the set of top-k configurations, the set of interpretations that the backward analysis step is returning is the union of all the top-k summary graphs generated for each of the top-k configurations.

In QUEST we have extended a Steiner Tree discovery algorithm [24] by making it to work at the schema level instead of the instance level, and implemented a mechanism to efficiently discard trees whose computation does not provide any additional results, like for instance, those that are super-trees of other Steiner Trees already computed. More specifically, Steiner Trees are computed using an ad-hoc extension of the DPBF algorithm [24]: we introduce a tree-similarity function to discard, among the computed partial results, those that are subgraphs of others. Assume that we denote by $S(t)$ the Steiner nodes (i.e., non-terminals nodes), and with $T(t)$ the terminal nodes of tree t, the similarity function $simil(t,t')$ returns true if $S(t) \subseteq S(t')$ and $T(t) = T(t')$. The motivation is that the join-path associated to the Steiner Tree, which is supergraph of another tree, does not introduce any new information concerning how to map the keyword query into the graph and therefore can be discarded.

Working on graphs modeling the meta-data instead of the actual tuples in the instance [4,24,26], offers QUEST a number of advantages. In particular, (i) the graph is typically smaller and hence the technique is in general more scalable, (ii) it is less subject to changes than a graph over the database instance, (iii) it has uniform semantics for edges, i.e., primary/foreign key join, and (iv) it can be computed even in cases where the database instance is not directly accessible. These advantages allow QUEST to deal with "critical" real scenarios where the database size gives rise to graphs with millions of vertices and edges, for which the problem of finding Steiner Trees is becoming intractable.

Using graphs that model database schemas requires to address an additional important issue: the case that the obtained Steiner Tree does not provide any direct result (i.e., no actual tuple is returned), but only the specification of a join-path that could result in an empty set of tuples. This happens because the configurations discovered in the forward step map keywords into database terms in isolation. Therefore, there are no guarantees for a configuration to

[2] *Joint distribution* is a brief form of *joint probability distribution function* defined as a probability distribution that gives the probability that each pair (X_{A_1}, X_{A_2}) falls in a particular range or in a discrete set of values specified for those variables.

correspond to a tuple actually existing in the database instance. However, the use of the mutual-information-based approach that we have described previously minimizes the chances that the top-k interpretations that are selected based on the weights have this problem.

3.3. Ranking combination implementation

There are two points in the framework implementation in which different ranked results alongside their confidence score need to be combined into one, and the confidence scores of the elements in the new ranking list needs to also be computed. The first point is in the forward analysis step in which the results of the two different implementations of the step need to be combined into one. The second is at the point in which the ranked list of the interpretations computed by the backward analysis step need to be updated by taking into consideration the scores of the configurations that produced these interpretations in the forward analysis step. By updated, it is not meant simply changing the order but also computed the new scores for the elements in the list. When this is done, the top-k elements can then be selected. Although there are popular algorithms for merging multiple ranking lists into one [27], the need to also accurately compute the confidence probabilistic score of the elements in the final list poses some additional challenges.

We believe that a promising approach for this purpose is a probabilistic framework and for that we have selected the Dempster Shafer Theory (DST). The DST [28] is a generalization of the Bayesian theory of subjective probability that allows evidences coming from different independent sources, under uncertainty conditions, to be combined. Given a set of elements in which we are interested, called *universe* and denoted by U (the so-called frame of discernment), the idea of DST is to:

1. Obtain a *belief function* (a. k. a. *mass function*) $m: \mathcal{P}(U) \rightarrow [0, 1]$ for each source providing some type of information. Each belief function m must satisfy the following properties: (i) $m(\varnothing) = 0$, (ii) $\sum_{E \in \mathcal{P}(U)} m(E) = 1$. Moreover, $m(U)$ represents the *degree of uncertainty* specified for that particular source.
2. Combine the different degrees of belief by means of Dempster's rule. This rule establishes that given two mass functions m_1, m_2 representing source 1 and 2 respectively, then the aggregation mass functions m_{12} is computed by the following equations:

$$m_{12}(E) = \frac{\sum_{E' \cap E'' = E} m_1(E') m_2(E'')}{1 - K} \quad \text{if } E \neq \varnothing \tag{3}$$

$$m_{12}(\varnothing) = 0 \tag{4}$$

$$K = \sum_{E' \cap E'' = \varnothing} m_1(E') m_2(E'') \tag{5}$$

where K represents the mass associated with *conflicting evidences*, i.e., when the intersection is empty.

There have been different proposals to aggregate evidences by considering other methods to hand conflicts [28]. We have selected the Dempster's rule because it is well-known and extensively used in practice. Nevertheless, the approach can be easily adapted to use other rules.

For the case of combining the results of the a priori and the feedback based implementations of the forward analysis step, as *universe* it is considered the union of the two respective result sets. Furthermore, the scores of the elements in the two respective lists are used to approximate a probability distribution function. For doing so they have to first be normalized and add up to 1. This probability distribution function is what considered as the mass function that the DST requires. Finally, there is a need for deciding which ranking to trust more and which not. For this, two additional parameters are used that describe the confidence that is put on each of the two lists. These are the $Conf_{apr}$ and $Conf_{fdback}$, respectively.

The specific values of the latter two parameters may change as the system is operating, making QUEST a tool easily adaptable and reacting to changes in its working context. For example, when QUEST is used to query a new database, little feedback is available. Thus, the feedback-based mode can become less reliable than what will be after some long time querying the data source. Consequently, the parameter $Conf_{apr}$ is increased in order to allow the system to achieve a better overall performance. However, as the amount of the available feedback increases, the parameter $Conf_{fdback}$ is incremented. This same parameter will on the other hand decrease when "negative" feedback is obtained and the system will be reconfigured automatically according to this decrease.

For the case of combining the results of the interpretations with the configurations of the forward analysis step, the approach is similar. The scores that have been assigned to the configurations and interpretations from the forward and backward analysis step, respectively, are considered observation values (i.e., evidences) and are used to compute the probability distribution function.

One issue to be taken care here is that the two lists have heterogeneous contents (one has configurations, the other interpretations) and some configurations may be orphan, i.e., a configuration for which the interpretations has not made it to the top-k list that the backward analysis step has produced. For this, before performing the combination, this orphan configurations are discarded. Other than that, in this second case one can have two confidence parameters, $Conf_{fw}$ and $Conf_{bw}$ that specify the importance that is to be placed on the configuration (forward analysis) or the interpretation (backward analysis), respectively. Giving higher value to the first will lead towards results covering more disparate tables or attributes in the database, even if loosely connected (i.e., through very long paths). On the other hand, higher confidence value on the backward analysis will lead to strongly related database terms, i.e., more coherent structures be returned. The exact values to set is application specific. Although they can get any values, a good practice would be to assign values that add up to 1. In that way, knowing the value of one of the parameters, which will be between 0 and 1, the other can be computed.

Example 3.2. Applying the forward and the backward analysis implementations of QUEST on the running example keyword query `Vokram IT` leads to the scores indicated in Table 1. Each line corresponds to one of the three

interpretations [A.1], [A.2] and [B.1] of Example 2.3. The first column indicates the scores for the respective configurations, i.e., the A for the [A.1] and [A.2], and the B for the [B.1], as they are resulting from the forward analysis. The second column are the scores of the three interpretations resulting from the backward analysis. The next three columns indicate the scores of each interpretation depending on the relationship of the value of the confidence parameters. The number that is shown in parenthesis simply indicates the ranked position of the respective number among the three. The goal here is not to explain in details how these numbers were created, but simply to illustrate the effect that the values of the confidence parameters $Conf_{fw}$ and $Conf_{bw}$ can have to the final list that the system will return.

3.4. Generating explanations: the translation

One approach for generating the explanations from an interpretation is to consider all the tables for which there is an attribute or attribute domain database term in the interpretation, or the table itself appears as a database term in the interpretation. All these tables form part of the where clause. Furthermore, for every association between the database terms included in the configuration, the join conditions among their respective tables are added in the where clause. The challenging question is what to be placed in the select clause, since in contrast to SQL or other structured form of queries, keyword queries do not specify neither the objects to be retrieved nor the form or the attributes they should have. In the absence of such information, QUEST is returning a complete picture of the involved structures, i.e., the set of all the attributes related to the relation database terms involved in the interpretation. In other words, for QUEST an explanation, i.e., the generated final SQL query, is defined as follows:

Definition 3.1. An *explanation* e of an interpretation $\langle V^S, E^S \rangle$ derived by a configuration c of a keyword query q is an SQL query in the form

select a_1, a_2, …, a_n
from R_1 join R_2 on J_1 join …join R_o on J_p
where $A'_1 = k_1$ and $A'_2 = k_2$ and …and $A'_m = k_m$

formulated by adding: (i) a condition $A = k$ in the where clause for every node in V^S that corresponds to an attribute domain $Dom(A)$ which is the image $c(k)$ of the keyword k; (ii) a join condition $J: A = A''$ in the from clause for every edge between two nodes $Dom(A)$ and $Dom(A'')$; and (iii) a reference to the attribute a of relation R in the select clause if a node representing R is in V^S.

The above definition is implemented in the `Interpretation2SQL` module in Fig. 5 illustration.

Keyword queries are very vague. They are flat lists with no explicit relationships between the keywords, thus many different interpretations are possible [29]. The whole framework presented here is based on the hidden assumption that the semantics the user had in mind when formulating the query are expressible as an SPJ query. However, SPJ queries form a large class of queries that can fulfill the requirements of the majority or real life applications, something that has already been recognized [30]. Almost all the keyword query answering techniques on structured databases follow a similar assumption [2–12]. Clearly, there may be applications that require more complex queries that cannot be covered by our approach, e.g., self-joins. These are focused on specific applications, and can be handled on a case-by-case basis. For instance, the self-joins can be implemented by considering multiple copies of the same database term modeling the table for which a self join may be applied. As an example, the graph of Fig. 3 could have more than one nodes representing the table `Person`, and its included attributes to allow self-joins on that table to be considered as explanations produced by the system.

4. Related work

Over the last decade, a great amount of approaches to allow users to access structured data by means of keyword queries has been proposed. These proposals can be classified into two main categories [1]: *schema-based* (a.k.a. relation-based) and *graph-based* (a.k.a. tuple-based).

The *schema-based* approaches model the database to be queried as a graph where nodes represent relations and attributes, and edges represent key/foreign-key or membership relationships. In this kind of systems, a keyword query is typically evaluated in two steps. Firstly, SQL queries are generated to describe the intended meaning of the user query in database terms. Moreover, queries are ranked and evaluated based on their relevance (semantic proximity) to the assumed user query semantics. Secondly, the most relevant SQL queries are executed to retrieve tuples from the database. The main goal here is to optimize the algorithms used to generate the SQL queries and to select the right metrics for the evaluation of the tuples retrieved by these queries. Examples of systems following the schema-based approach include DISCOVER [2], DBXplorer [3], SPARK [5], and SQAK [6].

The *graph-based* approaches model relational databases as graphs where nodes are tuples and an edge between two tuples denotes that they are connected by a key foreign-key constraint. Thus, connected tuples can be extracted directly

Table 1
Ranking scores for the running example results.

Result	Scores of the approaches		Explanation score (rank)		
	Forward (f)	Backward (b)	$Conf_{fw} = Conf_{bw}$	$Conf_{fw} < Conf_{bw}$	$Conf_{fw} > Conf_{bw}$
[A.1]	−5.080 (1)	−0.174 (1)	−4.809 (1)	−11.217 (1)	−2.402 (1)
[A.2]	−5.080 (1)	−0.492 (3)	−4.959 (3)	−12.048 (3)	−2.780 (2)
[B.1]	−9.123 (2)	−0.178 (2)	−4.820 (2)	−11.321 (2)	−4.596 (3)

from the data-graph. BANKS [4], BLINKS [31], PRECIS [9], DPBF [24] and STAR [32] are examples of graph-based systems. Their main aim is to optimize the computation of specific structures over the graphs (e.g., Steiner Trees, rooted trees, etc.) to find the most relevant top-*k* connected tuples. Their challenge is to handle the large and complex graphs induced by the database instance, as it could make the problem hardly tractable. Furthermore, different interpretations (with different structures) that arise due to inherent keyword ambiguities appear all mixed up in the result sets.

Our QUEST system is a schema-based approach as it performs the matching of keyword queries into database structures by means of the forward step, and after that, it exploits information from the schema and the instances of the database to generate possible interpretations (called "candidate networks" in some previous schema-based approaches), i.e., portions of the database schema graph that includes the keywords in the user query seen as a bag of words. Nevertheless, by using the HMM, these words are seen as a sequence, considering their relative positions in the query and the inter-dependencies between the matchings of the keywords to the various database structures. QUEST also makes use of techniques based on Steiner Trees, which are generally exploited by graph-based approaches. In [1,24], Steiner Trees are applied to the instance graph for identifying the results of a query. However, in the backward step, QUEST builds a graph of the database schema instead, and uses Steiner Trees to identify the tables and join paths needed for the SQL query formulation.

SQL query SUGGestion (SQLSUGG) [33] is currently the most similar system to QUEST. It implements a schema-based approach that also uses a probabilistic model based on entropy to suggest SQL queries efficiently to the users by means of a friendly interface. The main differences between SQLSUGG and QUEST are the following ones: (1) in QUEST, firstly users' keywords are matched to elements of the database, and, after that, the different elements of a set of matches (i.e., a configuration) are related among them by exploiting schema-information (i.e., different interpretations are generated). On the other hand, in SQLSUGG firstly templates are selected (i.e., a fragment of the database that connect different elements of the database) and after that keywords are matched to elements in the different fragments; (2) in SQLSUGG templates to generate the SQL queries are generated offline (i.e., they are predefined) while in QUEST are generated on-the-fly at run time; and (3) SQLSUGG considers aggregate functions and QUEST does not consider them. Nevertheless, the techniques used by SQLSUGG to consider that kind of operators can be easily incorporated to QUEST. Another interesting work is DBease [34], a system that allows users to choose between three different options to perform their search: (1) using a recommendation system that suggests keywords from the database instance while the users are typing their queries, (2) using multiple input boxes to formulate queries instead of only one (the different boxes are related to different tables of the database underlying), or (3) using SQLSUGG (previously described). Nevertheless, neither DBease nor SQLSUGG consider queries previously formulated by users or the order of the keywords in the query to suggest SQL queries. In this way, they only exploit the information obtained from the database without taking into account the users' point of view (users' perspective).

Most of the existing approaches rely on indexes and functions over the data values to select the most prominent tuples as results of queries. Only recently metadata-based approaches have been developed [35]. These approaches are useful when there is no direct access to the database instance (as it happens for sources that are behind data-intensive web applications in the deep web) or when frequent updates make the process of building and updating indexes too expensive. QUEST can also work in this scenario, by "simulating" and approximating the results of full-text index search (unavailable in these conditions) with semantic matchings, similarity measures and data type compatibilities using the techniques we have previously developed [13].

Finally, although keyword search is the most well-known approach that facilitates query answering, there are others that also try to help the user in various ways. These include the Query-by-Example [36], the Exemplar Queries [29], and Query Relaxation [37,38].

5. Experimental evaluation

5.1. Experimental setup

The data sources. We employed two databases frequently used in the literature for experimental evaluations: *Mondial*[3] and a relational implementation of *DBLP*.[4] Even if the databases contain a comparable number of database terms (227 and 237 terms, respectively), they differ in size and number of connections among the data structures. DBLP has a simple structure where the tables can be joined in the majority of the cases by a unique path. Conversely, the Mondial structure is complex and tables are often joined by multiple paths. Concerning the instances, the size of Mondial is more than two orders of magnitude smaller than DBLP. By way of example, "People" and "inproceedings", describing authors and papers, are definitely two of the largest DBLP tables and have both a cardinality close to one million tuples. Moreover, papers are linked to the respective authors through the table "author_inproceedings" which counts around four million tuples. The tables in Mondial are smaller: only one table, "city" contains three thousand instances, and the other tables include around (or less) five hundred tuples. Table 2 summarizes the main characteristics of the evaluation dataset.[5]

These features make the selected databases at opposite levels in an evaluation system which compares small size vs. big size databases and flat databases vs. databases with complex data structures. Consequently, we expect that in the computation of configurations (i.e., the matching of user keywords into database terms) QUEST performs better in Mondial than in DBLP, due to the database size. On the contrary, we expect that in the computation of the interpretations QUEST performs better in DBLP, since

[3] http://www.dbis.informatik.uni-goettingen.de/Mondial/
[4] http://dblp.uni-trier.de/
[5] The data about the Mondial database are taken from [39].

Mondial has a greater number of possible paths (i.e., FK–PK relations) connecting the tables.

The query datasets. The construction of a query training dataset is a critical task since synthetic keyword queries do not resemble the actual distribution of information needs, and, on the other hand, self-authored queries have a strong potential for bias [40,41]. In addition, it is unfeasible to have a sufficient number of queries formulated by independent third parties with a description of their intended meaning. For this reason, we adopted a mixed approach where a small dataset of queries provided by independent users are exploited for creating a large dataset. In particular, we asked a set of users (25 person) to provide keyword queries for these databases alongside a natural language description of what they were looking for. We obtained a total of 69 and 144 queries for the Mondial and DBLP database, respectively. The numbers of keywords in the queries range from 1 to 5 (3.2 in average). A database expert translated each description into an *explanation* (i.e., an SQL query), obtaining a reference to evaluate the results returned by QUEST.[6] Then, the explanations have been exploited by a software application as a template for the generation of new keyword queries. We adopted a 10-fold cross validation evaluation approach, where each fold is composed of 10 000 keyword queries (9000 queries are used as training dataset, and 1000 as test dataset). We performed the experiments for each fold, and we computed the result as the average of the outcomes obtained in each fold. We also adopted subsets of the training datasets (i.e., including the first 0, 250, 500, and 1000 keyword queries of each fold) for evaluating the performance of the system in cold-start scenarios.

The system setup. The experiments were performed on a Linux Ubuntu virtual machine created in Oracle VM VirtualBox. This machine was setup with 3 processors and 5 GB of RAM. QUEST modules were implemented in C++ and in Java. MySQL was our DBMS.

The experiments. QUEST has been evaluated by taking into account five perspectives. Firstly, the three components of the approach have been evaluated separately: in Section 5.2 the forward step, in Section 5.3 the backward step, and in Section 5.4 the combination. Then, in Section 5.5 we evaluated how QUEST performs on sources where no direct access to the instance is available. Finally, in Section 5.6 a benchmark [39] is used for comparing QUEST with other systems.

The evaluation. The experiments aimed to evaluate the *effectiveness* and the *efficiency* of the approach. In the literature, the effectiveness of a keyword search engine is usually evaluated in terms of recall and precision of the results retrieved. Nevertheless, except for the comparison with the benchmark, we decided to evaluate the effectiveness by measuring the *accuracy of the results* considering the top-*k* results retrieved (the percentage of correct results in *k* results retrieved). For example, if the accuracy of the top-1 results is 90%, this means that in 90% of queries evaluated the first output matches the expected result of the user. Moreover, we applied the accuracy measure to the configurations/interpretations/explanations because QUEST actually does not retrieve directly the data, but it generates a set of keyword queries to be executed by the database management system holding the data.

Finally, the system efficiency was measured by taking into account the time needed for generating a set of configurations/interpretations/explanations. We did not report the time required to actually evaluate the explanations – i.e., SQL queries running time – since our aim is to measure QUEST avoiding the performance of the DBMS to blur the results.

5.2. The forward analysis step

The goal of the forward analysis step is to compute configurations, i.e., mapping of user keywords into database terms. Two operating modes, the "a priori", based of the database analysis, and the "feedback-based", exploiting the user feedback, have been implemented and evaluated.

Effectiveness. We expect our application to be employed for querying unknown databases where little or no information about the queries previously submitted to the system is available for the training. In such scenario, the a priori operating mode is able to compute configurations of keywords based on the knowledge of the database schema and contents. Fig. 6 reports the cumulative accuracy achieved in our experiments. The result expected by the user is always in the top-10 results generated by QUEST in the Mondial database, slightly worse accurate levels are achieved against DBLP, as expected for the large size of this database. Let us recall that the a priori operating mode is based on a transition matrix where the values are computed by applying heuristic rules. As a reference, Fig. 6 shows the accuracy obtained with a transition matrix populated with uniform values (no heuristic rules are applied).

In the feedback-based operating mode, the implicit feedback provided by the users is used as a training supervised dataset, i.e., the user can in some way specify if and which answers satisfy their information needs. To simulate the performance of the system in this scenario, we conducted several experiments where QUEST has been trained with a dataset composed of two parts: one including supervised data, the other unsupervised data. In particular, the unsupervised dataset was built by assuming the system working correctly. Therefore, the unsupervised dataset contains for each instance a query and the best configuration computed by QUEST. By means of this setting, we try to mimic the users' behavior (i.e., we assume that the user feedback is not always available). The fraction of supervised/unsupervised data in the training dataset was different in each test-run.

Fig. 7 shows the results of our experiments. We label series in Fig. 7(a) and (c) (referring to the DBLP and Mondial database, respectively) with the format "###*S*_1/*X*" to indicate that we are using a training dataset of 9000 queries with a supervised initial training set of ### queries, and after there is a user feedback for 1 out of *X* queries (i.e., 1 out *X* queries in the dataset is supervised). Fig. 7(b) and (d) shows how accurate the performance is after smaller training datasets composed of 0,[7] 250, 500 and 1000 queries (in this

[6] Note that the extraction of the corresponding configuration and interpretation from an explanation is a straightforward process.

[7] In this case no query is provided as training set.

Table 2
Main characteristics of the evaluation dataset.

Dataset	Size (MBs)	Relations	$\|V\|$	$\|E\|$	$\|T\|$
Mondial	16	28	17	56	12
DBLP	>1800	15	>4000	>7500	>4000

$|V|$, number of nodes (tuples) in data graph (in thousands); $|E|$, number of edges (foreign keys) in data graph (in thousands); and $|T|$, number of unique terms (in thousands).

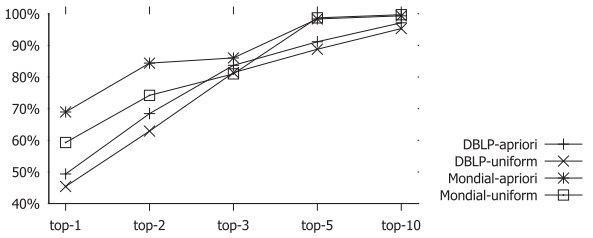

Fig. 6. Cumulative accuracy of the a priori forward analysis step.

case, the labels of the series have the format "$$$:", meaning that we are considered a dataset composed of just $$$ queries) are used. In both cases, the test set includes 1000 queries. We experimented these last scenarios with only a complete supervised and unsupervised approach.

The experiments show that with a large training dataset only a little amount of supervised queries is required for obtaining high performance. With smaller training datasets, the behavior of the system is polarized: only few supervised queries (more than 250) enable accurate results, around 100% even considering only top-1 answers. When the datasets are uniquely composed of unsupervised queries, the performance decreases. Note that (i) the accuracy is generally worse in the experiments against DBLP, due to the large size of the database; (ii) in the DBLP database, the accuracy slightly decreases when the set of unsupervised queries increases (see Fig. 7(b)). This is due to the "wrong answers" learned by the system. Finally, note that the level of accuracy obtained when no query is provided as training (i.e., the lines referred as unsupervised in Fig. 7(a) and (c), and the line referred as 0: unsuper in Fig. 7(b) and (d)) is different from what obtained in the a priori mode. This is due to the different probability distribution in matrix A. In the first case the distribution follows the heuristic rules described in Section 3.1. In the second case, the distribution is uniform: the E-M training algorithm did not perform any change to the initial probability distribution, since no training query has been provided.

To experiment the accuracy of the combination of the operating modes, we performed a number of test-runs where the results computed with the a priori and the feedback-based modes are combined with different "degrees of uncertainty". Fig. 7(e) reports the results of our experimentation for some settings of the Mondial database with the parameters maximizing the accuracy of the results. Similar results are obtained with DBLP. Note that in small size datasets, where the operating modes in isolation do not perform well, the combination of the approaches generates high accuracy results.

Efficiency. The forward analysis step has been evaluated considering two parameters: the number of keywords in a query, and the number of configurations to be computed as a result. In particular, we tested the system with queries composed of 1, 2, 3 and 5 keywords, and we computed the time required for retrieving top-1, top-10 and top-100 results. Fig. 7(f) reports the results of the experiments: the time performance differs in the two databases (solving queries in DBLP is up to 10 times slower than in Mondial) and it increases with the number of keywords. Since Mondial and DBLP have almost the same number of database terms, the experimental result seems to be only partially consistent with the computational complexity of the Viterbi algorithm, used for computing the configurations, which is $O(lkd^2)$, where l is the number of keywords, k the number of answers considered (top-k), and d the number of states. The discrepancy can be justified by the different size of the databases: DBLP is far larger than Mondial, and the time required to the full-text search function for retrieving the possible attributes (not estimated in the theoretical complexity calculus) largely differs between the databases. In addition, we would expect the efficiency to get worse with the increase of the results retrieved. In practice this is not the case since the number of possible configurations is in the majority of the cases less than 10. Finally, we do not take into account efficiency concerns related to the combination process performed by the DST, since it takes only a few milliseconds and therefore can be ignored.

5.3. The backward analysis step

The backward analysis step provides the interpretations of the configurations computed with the forward analysis step, i.e., it maps the configurations into paths over the database schema.

Effectiveness. For evaluating the effectiveness, we firstly tested the accuracy of the step in isolation, i.e., without the previous forward analysis step. In this way, we compute the highest level of accuracy that the backward analysis step can achieve. Then, we implemented and tested an algorithm computing the interpretations as the top-k shortest path connecting the terms in the configurations. This allowed us to compare the capability of the backward analysis step with another technique. The series with a 'BEST" suffix in Fig. 8 measure the accuracy of the backward analysis step implemented as described in Section 3.2; the series with a suffix "SP" show the shortest path trends. As expected, the accuracy is higher in DBLP, due to the flat structure of the schema, and the BEST algorithm performs better than the shortest path (especially for a database with complex schema as the Mondial database).

Efficiency. Fig. 9 shows the time required for building the interpretations. The time complexity of the DPBF algorithm extended in QUEST for computing the Steiner Trees is $O(3^l n^2 k + 2^l mnk)$, where l is the number of keywords, n the number of nodes (i.e., the attributes in the database), k the number of answer to be considered, and m the number of edges in the tree (i.e., the connections from attributes to primary keys in the same table, and from foreign keys to primary keys). The performances are consistent with the theoretic computation: they decrease with the increase of keywords, connections among schema elements (m), and the number of answers to be computed (k).

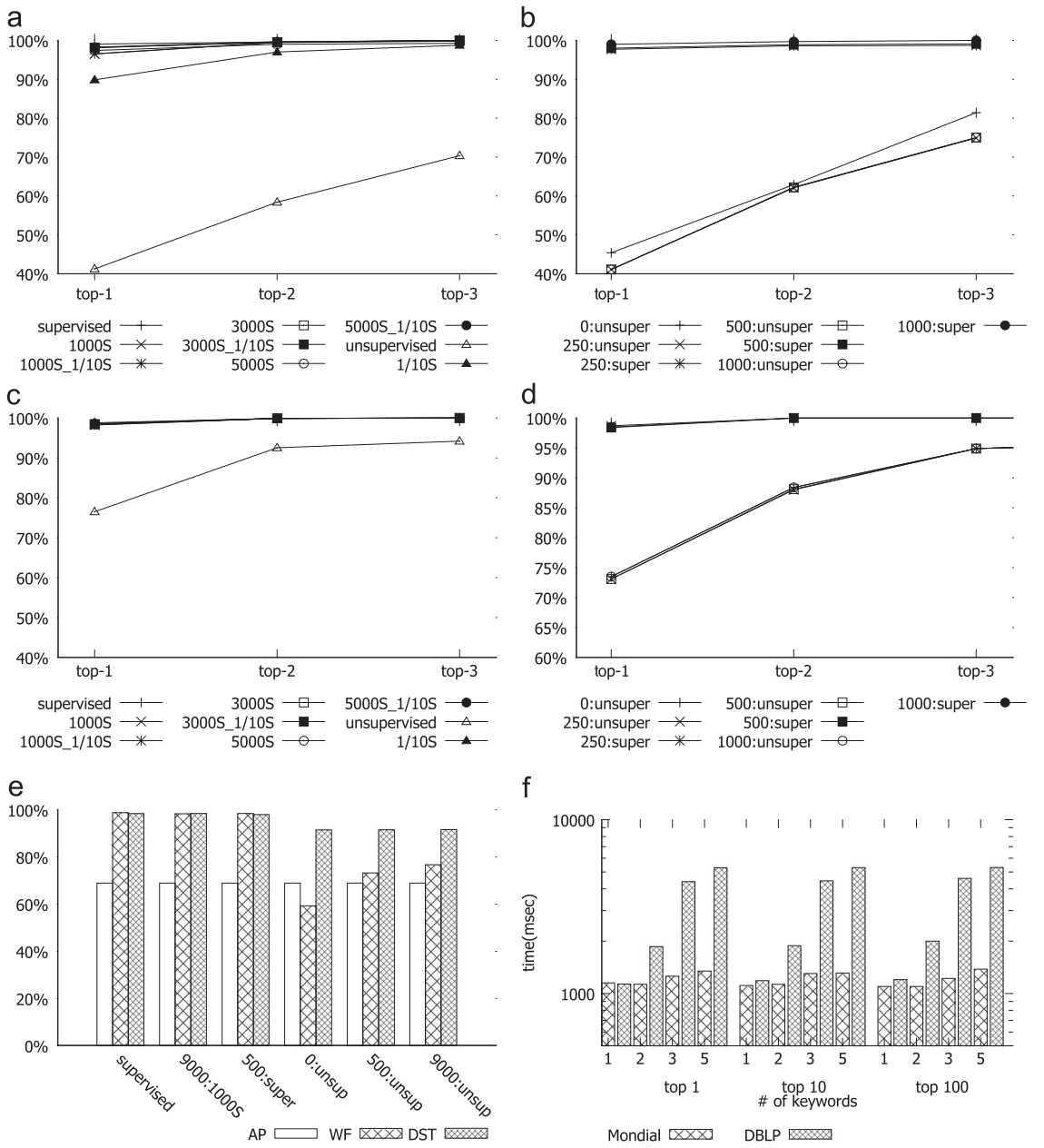

Fig. 7. Cumulative accuracy of the forward analysis step with feedback. (a) DBLP database – large datasets. (b) DBLP database – small datasets. (c) Mondial database – large datasets. (d) Mondial database – small datasets. (e) Application of the DST to Mondial. (f) Time for computing the configurations.

5.4. Combining the steps

Effectiveness. To experiment the accuracy of the combination of the steps, we ran the backward analysis step with the configurations computed by the forward analysis step. Then, we computed the final scores of the explanations as the combination of the scores returned by the forward and the backward analysis step. Again, we conducted a number of test runs where different "degrees of uncertainty" have been fixed for both the steps to find the parameters maximizing the accuracy of the results. Fig. 10(a) and (b) for DBLP and Fig. 10(c) and (d) for Mondial) shows that there is no relevant difference considering a small or a large dataset.

In addition, we ran another experiment, depicted in Fig. 10(f), with two particular settings to evaluate if the forward and the backward analysis steps can provide accurate results in isolation. In the first setting (the series with the suffix "SP"), we computed the interpretations of the configurations retrieved with the forward analysis step by selecting the ones with the shortest path (similarly to what we did in the previous section). In the second setting (represented by the series with the postfix "NOF"), we ran the backward analysis step decoupled from the forward

analysis step, i.e., interpretations are computed for all the possible combinations of the attributes associated to the keywords, as found by the full-text search function. The experiment shows that the results achieved with these settings are worse than the ones computed with the QUEST technique (marked as "supervised").

Efficiency. The time for computing the combinations and generating the explanations is few milliseconds and can be ignored.

5.5. QUEST without full access to the database

In some scenarios, to suppose a full access to the database instance and the availability of full-text indexes could be infeasible. This is the case for databases behind data-intensive websites in the deep web (i.e., the database is not directly accessible, only its schema), data sources which are the result of a virtual data integration process (i.e., there is no data, only an integrated schema), and databases which are subject to frequent and unpredictable updates (due to the high costs for keeping the indexes updated). To experiment if QUEST is able to perform well in these circumstances, we considered a scenario where only schema descriptions are available. The application of techniques exploiting this knowledge for finding out which database terms can be possibly matched into the user keywords makes QUEST usable also in this scenario. In this experiment, we used regular expressions for defining for each attribute a syntactic description of its possible content. Other techniques, based for example on public ontologies and thesauri or on the "Semantic Distance" as we proposed in [14] can provide improved results. Fig. 11 shows the cumulative accuracy of the configurations obtained in some settings of the DBLP database. As expected, the accuracy is lower compared to the scenarios where the full-text indexes are available.

As described in Section 3.2, the computation of the interpretations in the backward analysis step relies on a distance based on mutual information. Such a distance is computed by analyzing the data actually stored in the source. If the data are not accessible, as in the current scenario, this measure cannot be computed. Despite of this fact, we performed an experiment where the results obtained from previous queries are used for populating a "reduced" version of the original database, having the same schema and containing only the tuples returned to the user as an answer. Hence we employed a training dataset of 250, 500, and 1000 query to accordingly populate the reduced database following the usual supervised/unsupervised settings. The results reported in Fig. 12 show that in unsupervised scenarios, even if a relative small training dataset is available, in the 80% of the cases, the interpretation matching the intended meaning of the user query is in the first 10 results provided by the system. In the supervised scenarios, this happens almost for the 100% of the queries.

Finally, we combined the interpretations obtained with the corresponding configurations. The results in Fig. 10(e) show that QUEST has a polarized behavior. Without supervised queries the accuracy is low and only 60% of the correct answers are in the top-10 results provided by the system. Nevertheless, with a few amount of supervised queries, the performance increases and reaches values greater than 85%.

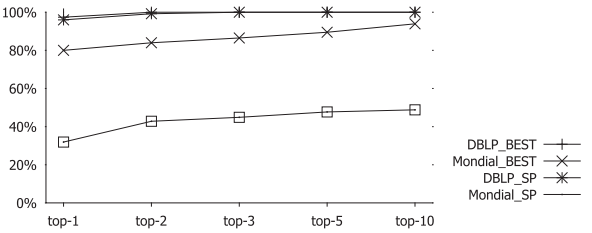

Fig. 8. Cumulative accuracy of the backward analysis step.

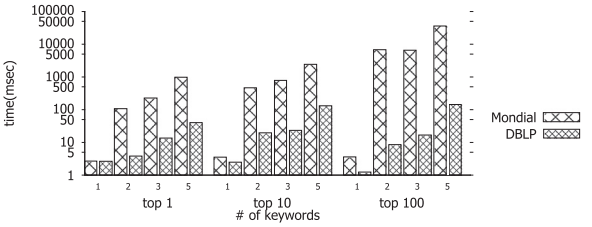

Fig. 9. Time required for computing the interpretations.

The performance remains practically unvaried if we consider supervised datasets of 250, 500 and 1000 queries. With a supervised set of 9000 queries (not shown in Fig. 10(e)), the results improve: the accuracy is 64.7% considering the top-1 result, and 93.7% considering the top-10.

5.6. Comparison with other approaches

The comparison of the performance obtained by keyword search approaches over relational databases is a complex task, mainly, due to the absence of a standard benchmark. The existing approaches have been evaluated against different databases with different query sets. This fact prevents their direct comparison based on their original experimental results. Moreover, in some cases, the evaluation framework adopted seems to be inadequate, mainly, due to the employment of a small number of self-authored queries [41], leading to biased results. Only recently, a benchmark [39] proposed some metrics and a query set to evaluate approaches against three data sources (Mondial, IMDB and Wikipedia). Even if the benchmark represents an important step towards a fair evaluation of keyword search approaches, the metrics adopted (precision and recall compared to a golden standard, and time needed for returning the results) cannot be suitable when applied to a schema-based keyword search system, such as QUEST, which transform keyword queries into SQL queries. The benchmark, in fact, computes the effectiveness of the approaches by analyzing the results (instances) retrieved with specific keyword queries whereas schema-based search approaches provide SQL queries as results [42]. Note that all the tuples resulting from the same SQL query have intrinsically the same score, and that the same result can be obtained by different queries.

Aware of the possible limitations, we evaluated the effectiveness of our approach against the benchmark in [39], to provide an "external" reference to the QUEST capabilities. The Mondial database has been selected as reference source, since it has a complex schema (more challenging than the other sources) and a non trivial size. Fig. 13 illustrates the result of

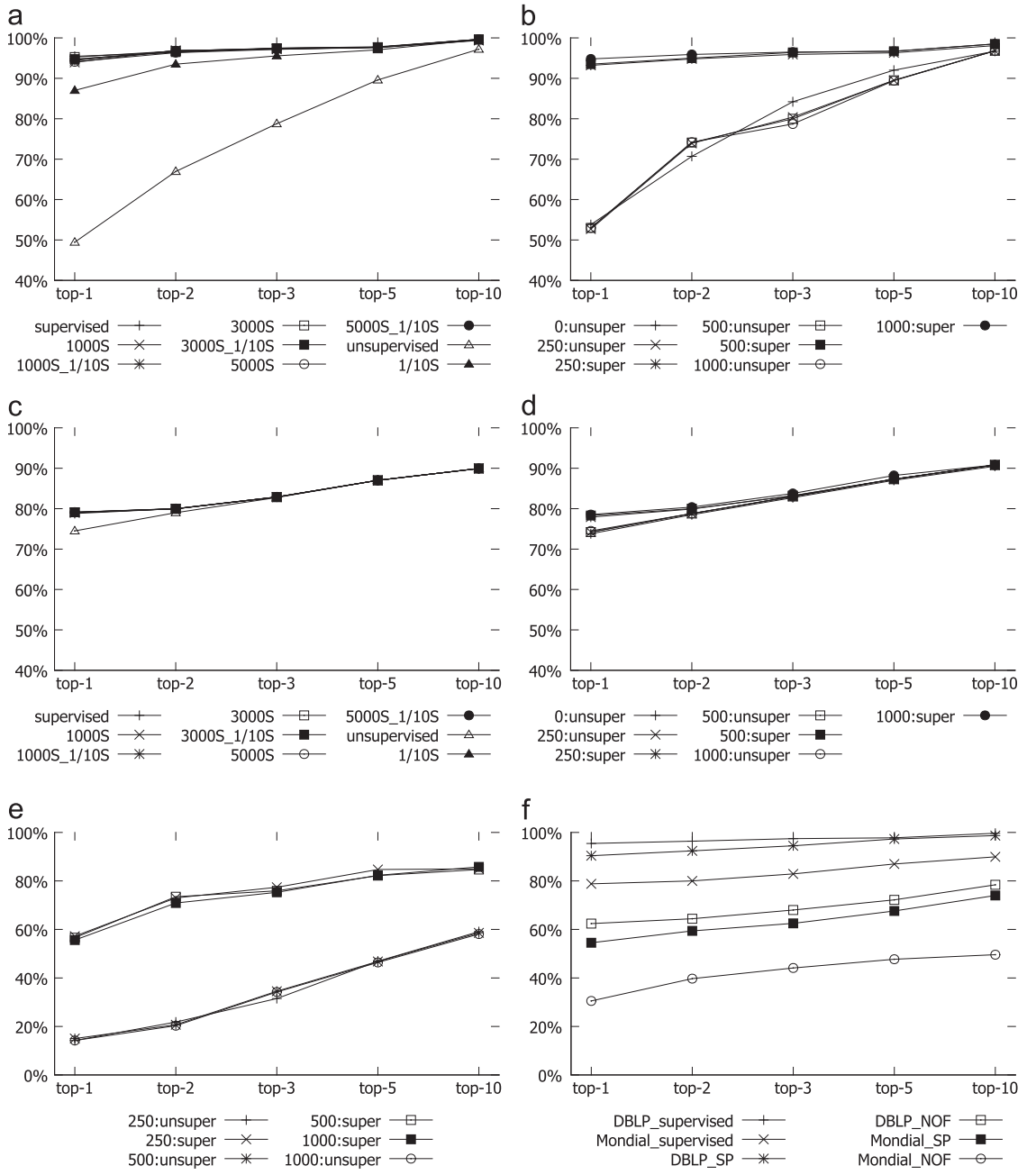

Fig. 10. Cumulative accuracy of the combined step. (a) DBLP database – large datasets. (b) DBLP database – small datasets. (c) Mondial database – large datasets. (d) Mondial database – small datasets. (e) DBLP database without full text indexes. (f) Simulation for the steps in isolation.

our experiments in which for the forward analysis we use the HMM with the a priori mode only. We limited the experimentation to 35 queries out of the 50 included in the benchmark. The 15 queries that were not considered[8] since their resolution is possible only with SQL queries with self-joins, feature currently not supported in QUEST. This choice in the design of our framework directly implies from the definition of a configuration as an injective function, and, consequently, two keywords in the same query cannot be mapped into the same database term. Nevertheless, this kind of query is not frequent (in our experiments with "real" users we have never found this kind of queries) and this constraint can be removed with the only effect to increase the number of possible configurations. To make our approach as simple as possible, we preferred to maintain the constraint in the paper and to limit the number of experiments. moreover, as highlighted in the benchmark, most of the approaches are not able to solve all the queries over all the databases. Fig. 13(a) shows that QUEST is able to find the solution for all the 35 queries.

[8] These are the queries 21–25 and 36–45.

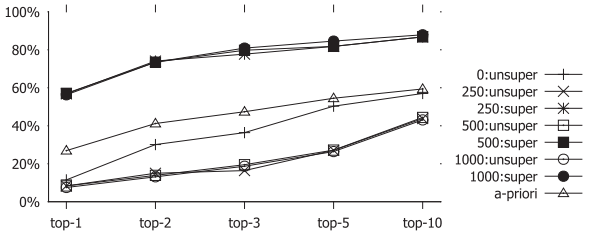

Fig. 11. Cumulative accuracy of the forward analysis step applied to the DBLP database without full text indexes.

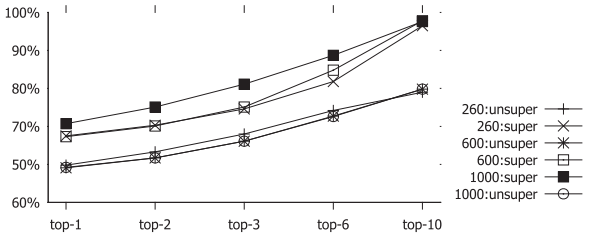

Fig. 12. Cumulative accuracy of the backward analysis step applied to the DBLP database without full text indexes.

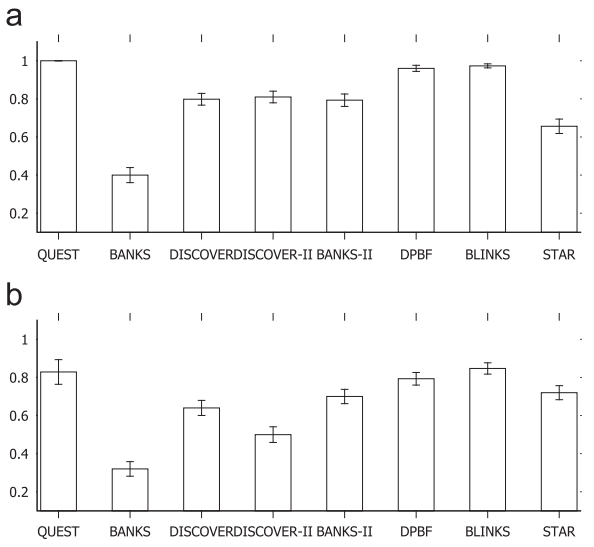

Fig. 13. Comparison with other approaches. (a) Recall. (b) Precision top 1.

Fig. 13(b) shows the precision if we only consider the first result provided by the search engines. QUEST obtains results with high precision degree, even if compared to the other approaches. The same happens if we consider the precision degree with respect to the top-10 results provided by the search engines. In this case, QUEST achieves a precision equal to 0.58 with a standard error 0.06. The precision of the other approaches is an average around 0.4 (only the STAR system obtains a final score similar to QUEST, close to 0.6).

6. Conclusion

We have presented QUEST, a framework for keyword search over relational databases which divides the process for solving keyword queries in three steps: forward, backward and the combination of the two. The forward step generates configurations, i.e., mappings keywords into database terms. Configurations are derived following a user perspective, i.e., taking into account how the query has been formulated by the user. The backward step formulates interpretations of the obtained configurations, i.e., paths joining the database structures involved in a configuration. These are computed following a database perspective, i.e., taking into account how the information is actually fragmented in a number of tables in the database. Configurations and respective interpretations are combined to form an answer to the keyword query and ranked by means of a probabilistic framework which allows users to specify a level of uncertainty.

QUEST is completely customizable and able to provide – as experimental results demonstrate – highly accurate results independently of the database size, structure complexity, direct access to the instance, and availability of full-text search functions.

Acknowledgment

The authors would like to acknowledge networking support by the ICT COST Action IC1302 KEYSTONE – Semantic keyword-based search on structured data sources (www.keystone-cost.eu).

References

[1] J.X. Yu, L. Qin, L. Chang, Keyword search in databases, in: Synthesis Lectures on Data Management, Morgan & Claypool Pub., San Rafael, California (USA), 2010.

[2] V. Hristidis, Y. Papakonstantinou, Discover: keyword search in relational databases, in: Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, August 20–23, Hong Kong, China, 2002, pp. 670–681.

[3] S. Agrawal, S. Chaudhuri, G. Das, Dbxplorer: a system for keyword-based search over relational databases, in: Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26–March 1, IEEE Computer Society, Washington, DC 20036-4928, 2002, pp. 5–16.

[4] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, S. Sudarshan, Banks: browsing and keyword searching in relational databases, in: Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, August 20–23, Hong Kong, China, 2002, pp. 1083–1086.

[5] Y. Luo, X. Lin, W. Wang, X. Zhou, Spark: top-*k* keyword query in relational databases, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12–14, ACM, New York, New York 10121, 2007, pp. 115–126.

[6] S. Tata, G.M. Lohman, SQAK: doing more with keywords, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10–12, ACM, New York, New York 10121, 2008, pp. 889–902.

[7] F. Liu, C.T. Yu, W. Meng, A. Chowdhury, Effective keyword search in relational databases, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27–29, 2006, pp. 563–574.

[8] L. Qin, J.X. Yu, L. Chang, Keyword search in databases: the power of rdbms, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29–July 2, ACM, New York, New York 10121, 2009, pp. 681–694.

[9] A. Simitsis, G. Koutrika, Y.E. Ioannidis, Précis: from unstructured keywords as queries to structured databases as answers, VLDB J. 17 (1) (2008) 117–149.

[10] T. Tran, H. Wang, S. Rudolph, P. Cimiano, Top-*k* exploration of query candidates for efficient keyword search on graph-shaped (rdf) data, in: Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29–April 2, Shanghai, China, IEEE Computer Society, Washington, DC 20036-4928, 2009, pp. 405–416.

[11] V.S. Uren, Y. Lei, E. Motta, Semsearch: Refining semantic search, in: The Semantic Web: Research and Applications, Proceedings of the 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1–5, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2008, pp. 874–878.
[12] Q. Zhou, C. Wang, M. Xiong, H. Wang, Y. Yu, Spark: adapting keyword query to semantic search, in: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11–15, 2007, pp. 694–707.
[13] S. Bergamaschi, E. Domnori, F. Guerra, R. Trillo-Lado, Y. Velegrakis, Keyword search over relational databases: a metadata approach, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12–16, ACM, New York, New York 10121, 2011, pp. 565–576.
[14] S. Bergamaschi, E. Domnori, F. Guerra, M. Orsini, R. Trillo-Lado, Y. Velegrakis, Keymantic: semantic keyword-based searching in data integration systems, Proc. VLDB Endow. 3 (2) (2010) 1637–1640.
[15] S. Bergamaschi, F. Guerra, S. Rota, Y. Velegrakis, A hidden Markov model approach to keyword-based search over relational databases, in: Proceedings of the 30th International Conference on Conceptual Modeling, ER 2011, Brussels, Belgium, October 31–November 3, Lecture Notes in Computer Science, vol. 6998, Springer, Berlin, Heidelberg, 2011, pp. 411–420.
[16] S. Rota, S. Bergamaschi, F. Guerra, The list viterbi training algorithm and its application to keyword search over databases, in: Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24–28, ACM, New York, New York 10121, 2011, pp. 1601–1606.
[17] S. Bergamaschi, F. Guerra, M. Interlandi, R. Trillo-Lado, Y. Velegrakis, QUEST: a keyword search system for relational data based on semantic and machine learning techniques, Proc. VLDB Endow. 6 (12) (2013) 1222–1225.
[18] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, Boston, USA, 1995.
[19] L. Popa, Y. Velegrakis, R.J. Miller, M.A. Hernandez, R. Fagin, Translating web data, in: Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, August 20–23, Hong Kong, China, 2002, pp. 598–609.
[20] R. Fagin, L.M. Haas, M.A. Hernández, R.J. Miller, L. Popa, Y. Velegrakis, Clio: schema mapping creation and data exchange, in: Conceptual Modeling: Foundations and Applications – Essays in Honor of John Mylopoulos, Lecture Notes in Computer Science, vol. 5600, Springer, Berlin, Heidelberg, 2009, pp. 198–236.
[21] R. Kumar, A. Tomkins, A characterization of online search behavior, IEEE Data Eng. Bull. 32 (2) (2009) 3–11.
[22] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, J. R. Stat. Soc.: Ser. B 39 (1977) 1–38.
[23] J.M. Kleinberg, Authoritative sources in a hyperlinked environment, J. ACM 46 (September (5)) (1999) 604–632.
[24] B. Ding, J.X. Yu, S. Wang, L. Qin, X. Zhang, X. Lin, Finding top-*k* min-cost connected trees in databases, in: Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15–20, IEEE, Washington, DC 20036-4928, 2007, pp. 836–845.
[25] X. Yang, C.M. Procopiuc, D. Srivastava, Summary graphs for relational database schemas, Proc. VLDB Endow. 4 (11) (2011) 899–910.
[26] K. Golenberg, B. Kimelfeld, Y. Sagiv, Keyword proximity search in complex data graphs, in: SIGMOD, New York, NY, USA, ACM, New York, New York 10121, 2008, pp. 927–940.
[27] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, E. Vee, Comparing and aggregating rankings with ties, in: Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 14–16, Paris, France, 2004, pp. 47–58.
[28] L. Liu, R.R. Yager, Classic works of the Dempster–Shafer theory of belief functions: an introduction, in: Classic Works of the Dempster–Shafer Theory of Belief Functions, Studies in Fuzziness and Soft Computing, Springer, Berlin, Heidelberg, 2008, vol. 219, pp. 1–34.
[29] D. Mottin, M. Lissandrini, Y. Velegrakis, T. Palpanas, Exemplar queries: give me an example of what you need, Proc. VLDB Endow. 7 (5) (2014) 365–376.
[30] A. Bonifati, Y. Velegrakis, Schema matching and mapping: from usage to evaluation, in: Proceedings of the 14th International Conference on Extending Database Technology, EDBT 2011, Uppsala, Sweden, March 21–24, 2011, pp. 527–529.
[31] H. He, H. Wang, J. Yang, P.S. Yu, Blinks: ranked keyword searches on graphs, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12–14, ACM, New York, New York 10121, 2007, pp. 305–316.
[32] G. Kasneci, M. Ramanath, M. Sozio, F.M. Suchanek, G. Weikum, Star: Steiner-tree approximation in relationship graphs, in: Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26–30, IEEE, New York, New York 10121, 2009, pp. 868–879.
[33] J. Fan, G. Li, L. Zhou, Interactive sql query suggestion: making databases user-friendly, in: Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11–16, Hannover, Germany, IEEE Computer Society, Washington, DC 20036-4928, 2011, pp. 351–362.
[34] G. Li, J. Fan, H. Wu, J. Wang, J. Feng, Dbease: making databases user-friendly and easily accessible, in: Online Proceedings of the 5th Biennial Conference on Innovative Data Systems Research, CIDR 2011, Asilomar, CA, USA, January 9–12, 2011, pp. 45–56.
[35] L. Blunschi, C. Jossen, D. Kossmann, M. Mori, K. Stockinger, Soda: generating sql for business users, Proc. VLDB Endow. 5 (10) (2012) 932–943.
[36] D. Braga, A. Campi, S. Ceri, *XQBE* (*XQ*uery *B*y *E*xample): a visual interface to the standard XML query language, ACM Trans. Database Syst. 30 (2) (2005) 398–443.
[37] D. Mottin, A. Marascu, S.B. Roy, G. Das, T. Palpanas, Y. Velegrakis, A probabilistic optimization framework for the empty-answer problem, Proc. VLDB Endow. 6 (14) (2013) 1762–1773.
[38] C. Mishra, N. Koudas, Interactive query refinement, in: M.L. Kersten, B. Novikov, J. Teubner, V. Polutin, S. Manegold (Eds.), Proceedings of the 12th International Conference on Extending Database Technology, EDBT 2009, Saint Petersburg, Russia, March 24–26, ACM International Conference Proceeding Series, vol. 360, ACM, New York, New York 10121, 2009, pp. 862–873.
[39] J. Coffman, A.C. Weaver, An empirical performance evaluation of relational keyword search techniques, IEEE Trans. Knowl. Data Eng. 26 (1) (2014) 30–42.
[40] J. Coffman, A.C. Weaver, A framework for evaluating database keyword search strategies, in: Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26–30, ACM, New York, New York 10121, 2010, pp. 729–738.
[41] W. Webber, Evaluating the effectiveness of keyword search, IEEE Data Eng. Bull. 33 (1) (2010) 55–60.
[42] S. Bergamaschi, N. Ferro, F. Guerra, G. Silvello, Keyword search and evaluation over relational databases: an outlook to the future, in: 7th International Workshop on Ranking in Databases, DBRank 2013, Riva del Garda, Italy, August 30, ACM, New York, New York 10121, 2013, p. 8.