QUEST: A Keyword Search System for Relational Data based on Semantic and Machine Learning Techniques*

Sonia Bergamaschi
University of Modena and Reggio Emilia, Italy
sonia.bergamaschi@unimore.it

Francesco Guerra
University of Modena and Reggio Emilia, Italy
francesco.guerra@unimore.it

Matteo Interlandi
University of Modena and Reggio Emilia, Italy
matteo.interlandi@unimore.it

Raquel Trillo-Lado
University of Zaragoza, Spain
raqueltl@unizar.es

Yannis Velegrakis
University of Trento, Italy
velgias@disi.unitn.eu

ABSTRACT

We showcase QUEST (QUEry generator for STructured sources), a search engine for relational databases that combines semantic and machine learning techniques for transforming keyword queries into meaningful SQL queries. The search engine relies on two approaches: the *forward*, providing mappings of keywords into database terms (names of tables and attributes, and domains of attributes), and the *backward*, computing the paths joining the data structures identified in the forward step. The results provided by the two approaches are combined within a probabilistic framework based on the Dempster-Shafer Theory. We demonstrate QUEST capabilities, and we show how, thanks to the flexibility obtained by the probabilistic combination of different techniques, QUEST is able to compute high quality results even with few training data and/or with hidden data sources such as those found in the Deep Web.

1. INTRODUCTION

Languages for querying structured databases, e.g., SQL and SPARQL for relational and RDF sources, are typically oriented towards expert users who have to formulate queries specifying the data of interest, and also from where the data have to be retrieved, i.e., tables and/or attributes. This means that, in order to produce meaningful queries, users are required to have good knowledge of both the query language, and the data source structure and its contents. On the other hand, keyword queries are user-friendlier since they require neither knowledge of the query language nor of the way information has been modeled in the data repository.

The Information Retrieval community has already developed advanced techniques for keyword search over documents, but direct application of these solutions to relational data sources, where information is typically fragmented in multiple tables, is neither efficient nor effective. Full-text inverted indexes, typically used in IR approaches, can be useful to associate keywords with tuples in tables, but cannot say anything about how tables have to be joined to form a meaningful answer to the keyword query as a whole. As a matter of fact, it could happen that no meaningful join-path exists among the tuples containing the keywords in the query. Even if this issue can be partially addressed by applying IR techniques to the "universal relation", building and keeping up-to-date such a relation is not feasible in practice. Furthermore, IR techniques do not take into account the structural semantics conveyed by the source schemas, which could largely improve the quality of the results.

Existing approaches that support keyword search over relational databases can be classified as schema-based and graph-based [8]. Schema-based techniques exploit the schema information to issue SQL queries with the same meaning as the original keyword queries. These approaches aim to optimize some metrics coding the relevance of tuples and joining paths among them with respect to the input keyword query. On the other hand, graph-based techniques treat relational databases as graphs, where nodes are tuples and edges relationships between those tuples. Algorithms for solving keyword queries in this context are based on the computation of specific structures over the graphs (e.g., Steiner trees, rooted trees, radius Steiner graphs, etc.). In this setting, the main issues are related to the large size of the graphs induced by the database instance that makes the problem hardly tractable.

Most of the existing approaches rely on indexes and functions over the data values for selecting the most prominent tuples as results of queries. Only recently metadata-based approaches have been developed [1, 2]. By not relying on data analysis, these approaches are useful when there is not direct access to the database instance (as happens for sources that are behind data-intensive web applications as in the Deep Web) or when frequent updates make the process of building and updating indexes too expensive. These approaches exploit semantic techniques to guess the database portions relevant to the queries and the way the structures in these portions may be joined to form a meaningful answer.

In this demo, we showcase QUEST (QUEry generator for STructured sources) a novel keyword search system over relational databases implementing a three-step schema-based approach, as shown in the following in Figure 1.

The first step is to determine how the keywords in the query can correspond to the structural elements of the database. This type of correspondences are referred to as *configurations* [1]. Of course, each correspondence comes with some degree of uncertainty that is typically expressed with a weight. We refer to this task as the *forward* task, since its starting point is constituted by the keywords

*This work was partially supported by the Action COST IC1302, TRISE Big Data Project and CICYT project TIN2010-21387-CO2.

Proceedings of the VLDB Endowment, Vol. 6, No. 12
Copyright 2013 VLDB Endowment 2150-8097/13/10... $ 10.00.

in the query as they were provided by the user. The forward step provides a user perspective since it requires an analysis that tries to unlock the intentions that the user had in mind when formulating the query. We implement this step by means of a Hidden Markov Model (HMM), which has the advantage of providing a solid and effective probabilistic framework. We use the term *a-priori* mode to refer to the modality of computation of the configurations which exploits a set of heuristic rules in order to choose the relevant elements without any involvement of the user. However, one can instead exploit machine learning techniques to properly train the HMM. We refer to this mode as the *feedback-based* mode.

The second step of the process is to identify the structure of the queries that can be generated from a given configuration. In particular, this step requires all the database elements discovered during the first task to be combined together to form a join-path. Each join-path is a materialization of certain semantics that likely represents the semantics that the user had in mind when formulating the keyword query, but not explicitly stated in the keywords due to its vague nature. We refer to these join-paths as *interpretations*. Hence, while the forward approach can be seen as a translation of a user-provided query into a set of elements described in terms of the database vocabulary, the second step can be depicted as a backward translation from the database vocabulary to a "structured" version of the initial query. For the above reason, we refer to the latter as the *backward* task. Steiner Tree discovery is a typical technique for implementing the task of finding a path joining a predefined set of elements. In contrast to other works, we use an extension of a previous algorithm [3] that works at the schema level instead of the instance level, and that has in place a mechanism for efficiently discarding Steiner Trees that are sub-trees of others that have been previously computed. Note that the starting point of the backward task is the database structure, since it specifies which path is or is not possible among the database elements identified by the configurations. Intuitively, in such a way the backward step is incorporating in the process the perspective of the database designer. However, this is not enough since we want to consider only join-paths actually existing in the database instance. QUEST implements a *mutual information*-based metrics to select the most informative join-paths, i.e., the ones which are likely to contain tuples in the database. As a consequence, the backward step introduces in the process a database instance perspective, mirror of the actual tuples stored in the database.

The third step is to decide which combination of keyword mappings into data structures and which paths connecting these data structures are the most likely to lead to structured queries representing the semantics that the user had in mind when formulating the keyword query. We refer to these combinations as *explanations*, since they provide the results of a keyword query in terms of data and its semantic interpretations. For the combination, we adopt a probabilistic framework based on the Dempster Shafer Theory (DST), for merging the scores associated with the configurations generated from the first step and the interpretations generated by the second. The DST combiner allows users to specify a confidence parameter denoting the importance they pay to the forward or backward approach. The same probabilistic framework, in addition, is employed in the forward approach for combining the configurations discovered by the a-priori and feedback-based operating modes.

Finally, every technique implemented in QUEST relies on a function that, given a keyword and the database attributes, ranks the attribute values on the basis of their importance. QUEST is conceived as a tool working on top of a traditional DBMS, however, it does not rely on a specific implementation of that function: a wrapper has been implemented for cases where this function is not available (because the DBMS does not support it or there is not a full access to the extension of the source). The wrapper exploits regular expressions, schema annotations, database metadata and external ontologies to guess the attributes that can be associated with each keyword. This makes QUEST able to query owned databases or hidden data sources such as the Deep Web, a feature which is not provided by any other existing approach.

Summarizing, the main contributions we showcase in this demonstration are: (i) the combination of two approaches, one taking into account the "user" perspective, and other considering the "database" perspective; (ii) the combination of two operating modes for the forward approach, one exploiting semantics and heuristic rules and the other based on machine learning techniques; (iii) the flexibility of the system to adapt to the different working conditions; and (iv) the ability to query full accessible databases and databases which provide a reduced access (via endpoints, web-services or forms).

2. BACKGROUND

QUEST is theoretically founded on two basic frameworks, the Hidden Markov Model and the Weighted Steiner Tree discovery, which are combined according to the Dempster-Shafer theory.

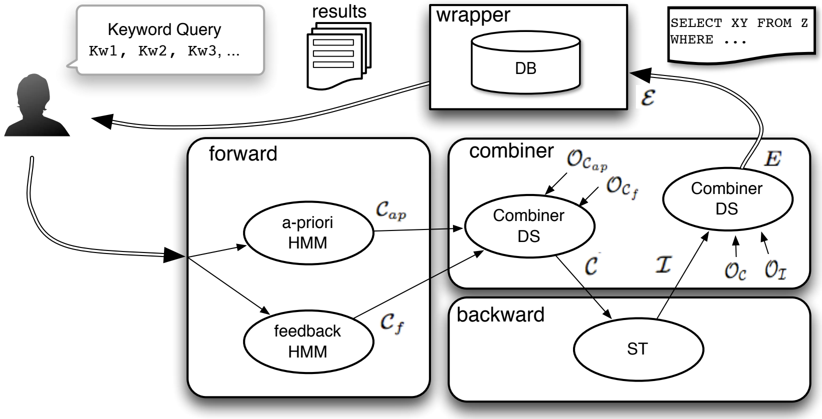
Hidden Markov Model (HMM): A HMM models a stochastic process that is not directly observable, but, however, can be indirectly observed through the observable symbols produced by another stochastic process. Assuming a time-discrete model, the process starts from an initial state based on an initial state probability distribution. Then, at each time step, a new state is entered based on a *transition probability distribution*, and an observation is produced according to a state-depended *emission probability distribution*. The list Viterbi Algorithm [5] applied to a HMM computes a list of top-k state sequences that have the highest probability of generating a particular sequence of observations given an input.

Weighted Steiner Tree (ST): Given a graph with weighted edges, a Steiner Tree is the minimum-weight tree connecting a designated set of vertices, called *terminals*. The tree may include non-terminals, called *Steiner points*. Even if finding the optimal Steiner tree is NP-complete in general, several approaches build weighted graphs from database instances and adopt Steiner trees as a technique for finding the tuples answering a given keyword query [8].

The Dempster-Shafer theory (DS) [6]: This probabilistic model allows to combine evidences coming from different sources under uncertainty conditions. The foundation of the theory is a probability mass function $m(X)$ that represents the belief committed to an evidence X. Given a set of base elements of interest Θ, called the *frame of discernment*, the elements of the power set of Θ with positive mass function values constitute the body of evidence. In addition, for each source, we associate a further mass to the universe denoting the *degree of uncertainty* specified for that particular source. The Dempster's rule of combination allows the aggregation of two independent bodies of evidence with the respective degree of uncertainty into one body of evidence.

3. THE FRAMEWORK

From an architectural point of view, QUEST has been designed as an add-on to existing databases, allowing users to express keyword query not only on owned databases, but also on virtually integrated data sources and on-line databases available on the Deep Web. The functional architecture of the system is depicted in Figure 1, where we show how the three steps in which we divide the process for solving a keyword query are implemented by



Algorithm 1 QUEST Search Process

```
Input: a keyword query $q$;
       maximum number of results $k$
       a set of uncertainty degree $\mathcal{O}_{\mathcal{C}_{ap}}$, $\mathcal{O}_{\mathcal{C}_f}$, $\mathcal{O}_\mathcal{C}$, $\mathcal{O}_\mathcal{I}$
Output: the top-k explanations $\mathcal{E}$

$\mathcal{C}_{ap} \leftarrow \emptyset$, $\mathcal{C}_f \leftarrow \emptyset$, $\mathcal{C} \leftarrow \emptyset$,
$\mathcal{I} \leftarrow \emptyset$, $E \leftarrow \emptyset$, $\mathcal{E} \leftarrow \emptyset$;

$\mathcal{C}_{ap} \leftarrow HMM\_a\_priori(q, k)$ |
$\mathcal{C}_f \leftarrow HMM\_feedback(q, k)$;
$\mathcal{C} \leftarrow CombinerDST(\mathcal{C}_{ap}, \mathcal{C}_f, \mathcal{O}_{\mathcal{C}_{ap}}, \mathcal{O}_{\mathcal{C}_f})$;
$\mathcal{I} \leftarrow ST(q, \mathcal{C}, k)$;
$E \leftarrow CombinerDST(\mathcal{C}, \mathcal{I}, \mathcal{O}_\mathcal{C}, \mathcal{O}_\mathcal{I})$;
$\mathcal{E} \leftarrow QueryBuilder(E)$;

Function: CombinerDST()
Input: a set of evidences $\mathcal{X}_1$, $\mathcal{X}_2$
       a set of uncertainty degree $\mathcal{O}_1$, $\mathcal{O}_2$
Output: the combined evidences E
$\mathcal{W}_1 \leftarrow \emptyset$, $\mathcal{W}_2 \leftarrow \emptyset$;
for $i := 1$ to 2 do
  for $j := 1$ to $\mathcal{X}_i.size()$ do
    $\mathcal{W}_i \leftarrow addEvidence(\mathcal{X}_{i,j})$
  end for
  $setUncertainty(\mathcal{W}_i, \mathcal{O}_i)$;
  $normalize(\mathcal{W}_i)$;
end for
$E \leftarrow apply\_DS\_rule\_of\_combination(\mathcal{W}_1, \mathcal{W}_2)$;
```

Figure 1: The keyword search process in QUEST.

means of three main software modules: the *forward*, the *backward* and the *combiner*. On top of this, a *w*rapper module manages the interactions with the data source.

Interacting with databases: the wrapper. In the setup phase, QUEST requires to know the database schema, which can be extracted from the metadata stored in the source catalogues, and to have full-text indexes instantiated over all the database attributes. If this is not possible, the user is supported in the definition of a schema enriched with the specification, for each attribute, of metadata such as data-type, and regular expression of admissible values. At run-time, the wrapper is in charge of executing the SQL queries generated by QUEST and computing the results.

Discovering Configurations: the forward module. The *forward module* implements the method described in [2] for discovering the top-k configurations associated with the user keyword queries. The process is modeled by means of a *HMM* that contains a state for each database element, i.e., there is a state for each table, attribute and attribute domain. If we consider the keyword query as a sequence of observations, the application of the list Viterbi algorithm to the HMM computes the top-k sequences of states (each one with an associated confidence value) with the highest probability of generating that sequence of observations. Therefore, by modeling in this way the search process, the emission probability distribution describes the likelihood for a keyword to be "generated" by a specific state, while the transition probability distribution describes the likelihood for two keywords to be associated with adjacent states. The emission probabilities are computed for each keyword and for each database attribute by applying the search function over full text indexes provided by the DBMS. We consider the value returned by this function as a probability, so we need to normalize it to add to 1 by means of a coefficient (different for each attribute) computed in the setup phase. In sources where the application of full-text indexes is not possible, similarity measures, domain compatibilities and semantic matchings are used to establish the admissible domains for each keyword in a query.

The probability distributions are usually computed by means of training algorithms. In the "feedback-based" operating mode, QUEST applies an Expectation-Maximization (E-M) [4] on-line training algorithm to a dataset composed of previous searches validated by the user. In addition, we implemented the "a-priori" operating mode that defines the parameter values by exploiting semantics collected from the data source metadata independently of the user feedback [2]. In this case, the transition probabilities are computed by using heuristic rules that take into account the semantic relationships that exist among the database terms (aggregation, generalization and inclusion relationships). 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.

Formulating Interpretations: the backward module. The *backward module* adopts a Steiner Tree-based technique (ST) to select, for each configuration, the top-k paths joining the involved database schema elements. Nevertheless, in QUEST, conversely to other systems based on Steiner Trees, the tree structure is built over a graph representing the database schema instead of the database tuples. In particular, we model the relational schema as a weighted graph where there is a node for each attribute in the database and edges connecting (i) the node representing the primary key of a table with all the other attributes in the same table, and (ii) nodes associated with couples of primary-foreign keys. Using a Steiner Tree over such a database graph offers advantages with respect to traditional approaches: a graph over a schema (i) is typically smaller and hence the approach is more scalable, (ii) is less subject to changes due to updates than a graph over the database instance, (iii) has uniform semantics for edges, i.e., primary/foreign key join, and (iv) can be computed even in cases where the database instance is not directly accessible. These advantages are related to critical issues in real scenarios where the database size gives rise to graphs with millions of vertices and edges, thus making the problem of finding Steiner Trees intractable. Adopting graphs over database schemas requires to address a new issue: 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 approach map keywords into database terms in isolation. Therefore, we are not assured a configuration to correspond to a tuple actually existing in the database instance. To create Steiner Trees consistent with the database content and the user keywords, we use a *mutual information*-based distance for computing the weights of the edges (see [7] where a similar measure has been adopted for database summarization).

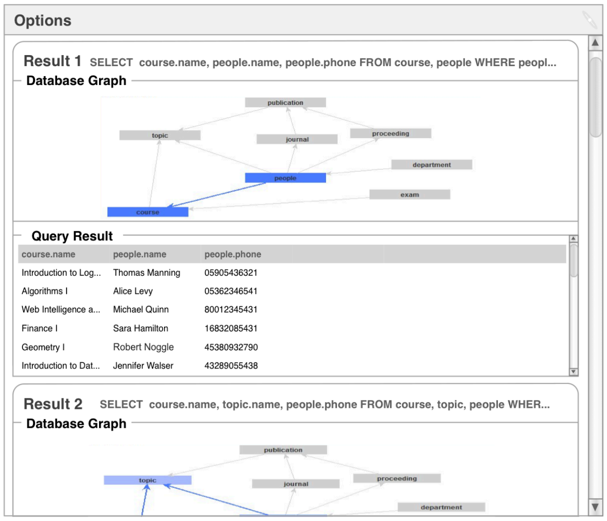
Combining partial results and providing Explanations. The *combiner* module based on the Dempster-Shafer's theory (DS) of evidence is used in two steps along the process of generation of explanations. Firstly, it is used for aggregating the results of the forward approach running under its two operating modes, i.e., a-priori and feedback-based. After that, it is employed to combine the results provided by the two different approaches – i.e., forward and backward – and to generate the explanations. In the former, for each operating mode, the union of the sets of top-k configurations is considered as the *universe*. Besides, a mass function is added by taking into account the configurations obtained, together with the parameters $\mathcal{O}_{\mathcal{C}_{ap}}$ and $\mathcal{O}_{\mathcal{C}_f}$, specifying the degree of uncertainty of the a-priori and feedback-based operating modes, respectively. For each operating mode, the scores associated with the configurations are used to approximate the probability that the correspondent configuration describes the intended meaning of the user query. For this reason, they are normalized to add to 1, as the Algorithm in Figure 1 shows. The specific values of the parameters $\mathcal{O}_{\mathcal{C}_{ap}}$ and $\mathcal{O}_{\mathcal{C}_f}$ change as the system performs, making QUEST a tool easily adaptable and reacting to changes in the working context. For example, when QUEST is used to query a new database, little feedback is available. Thus, the feedback-based mode is less reliable than after a long time queried data source. Consequently, the parameter $\mathcal{O}_{\mathcal{C}_{ap}}$ must be increased in order to obtain a better performance. On the other hand, as the amount of feedbacks increases, the related parameter $\mathcal{O}_{\mathcal{C}_f}$ must be incremented. Moreover, this same parameter should be decreased when "negative" feedbacks are obtained in order to re-configure the system accordingly.

In the latter case, the inputs of the combination function are the configurations resulting from the previous combination process and the interpretations computed by the backward module. Also in this case, users need to specify two parameters to indicate the uncertainty of the two approaches ($\mathcal{O}_{\mathcal{C}}$ and $\mathcal{O}_{\mathcal{I}}$ related to the forward and backward approach, respectively). Finally, the results of this module are the top-k explanations, i.e., the SQL queries which, executed, are the answers for the user keyword queries.

4. THE DEMONSTRATION

In this demonstration, we intend to show the use of QUEST against a number of real application scenarios, such as the IMDB database (www.imdb.com), the DBLP collection (dblp.uni-trier.de) and the Mondial database (www.dbis.informatik.uni-goettingen.de/Mondial). These databases provide a wide range of different scenarios to demonstrate QUEST: IMDB has a simple star schema but contains millions of instances, Mondial has few instances but a very complex schema where tables are connected through many paths, and DBLP contains many instances (e.g., 1 million people, 8 hundred thousand papers, and more than 2 million instances in the "is_author" relation) in a non-trivial schema. The demonstration will consist of two phases. In the first phase, we will run a number of chosen keyword queries against these sources, and demonstrate how the system handles ambiguous queries that generate multiple possible mappings, each one with multiple possible paths associated. During the second phase, the participants will be free to run their own queries and the system will display the different explanations (i.e., the possible SQL queries) along with the results obtained by querying the real databases. By browsing these explanations, as allowed by the system GUI shown in Figure 2, the users will be also able to get a taste of the parts of the sources that are related to their interest (as described in the keyword query).

There are five main messages we intend to communicate to the audience through this presentation. First, we will demonstrate that a schema-based approach for transforming keyword queries into SQL is really effective in querying large-size databases. Second, we will show that the different types of semantics implemented in the modules provide different results when applied to the same keyword query. For this purpose, we will compare and explain the partial results provided by each module separately to each query. Third, the analysis of the partial results will be also useful for demonstrating that Steiner trees are effective in computing answers to keyword queries even if applied to graphs representing database schemas. This is an original use of Steiner trees that other approaches have not adopted yet. Fourth, we will show how the partial results provided by each module can be effectively combined by exploiting the Dempster Shafer's theory. We will show how, setting different levels of uncertainty to each module and operating mode, we obtain different results and we can adapt the behaviour of the system to different scenarios. Fifth, we will show a new paradigm for visualizing query answers, by coupling the list of tuples with a graphical representation of the portion of the database involved by the query.



Options

Result 1 SELECT course.name, people.name, people.phone FROM course, people WHERE peopl...

Database Graph

Query Result

course.name	people.name	people.phone
Introduction to Log...	Thomas Manning	05905436321
Algorithms I	Alice Levy	05362346541
Web Intelligence a...	Michael Quinn	80012345431
Finance I	Sara Hamilton	16832085431
Geometry I	Robert Noggle	45380932790
Introduction to Dat...	Jennifer Walser	43289055438

Result 2 SELECT course.name, topic.name, people.phone FROM course, topic, people WHER...

Database Graph

Figure 2: The QUEST user interface.

REFERENCES

[1] S. Bergamaschi, E. Domnori, F. Guerra, R. T. Lado, and Y. Velegrakis. Keyword search over relational databases: a metadata approach. In *SIGMOD*, pages 565–576. ACM, 2011.

[2] S. Bergamaschi, F. Guerra, S. Rota, and Y. Velegrakis. A hidden markov model approach to keyword-based search over relational databases. In *ER*, LNCS 6998, pages 411–420. Springer, 2011.

[3] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *ICDE*, pages 836–845. IEEE, 2007.

[4] S. Rota, S. Bergamaschi, and F. Guerra. The list viterbi training algorithm and its application to keyword search over databases. In *CIKM*, pages 1601–1606, 2011.

[5] N. Seshadri and C.-E. Sundberg. List Viterbi decoding algorithms with applications. *Communications, IEEE Transactions on*, 42(234):313 – 323, feb/mar/apr 1994.

[6] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, New Jersey, 1976.

[7] X. Yang, C. M. Procopiuc, and D. Srivastava. Summary graphs for relational database schemas. *PVLDB*, 4(11):899–910, 2011.

[8] J. X. Yu, L. Qin, and L. Chang. *Keyword Search in Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Pub., 2010.