Processing Keyword Queries under Access Limitations*

Andrea Calì[1,4], Thomas W. Lynch[5,1],
Davide Martinenghi[2], and Riccardo Torlone[3]

[1]Birkbeck, University of London, UK
andrea@dcs.bbk.ac.uk

[2]Politecnico di Milano, Italy
davide.martinenghi@polimi.it

[3]Università Roma Tre, Italy
torlone@dia.uniroma3.it

[4]Oxford-Man Inst. of Quantitative Finance
University of Oxford, UK

[5]Reasoning Technology Ltd, UK
thomas.lynch@reasoningtechnology.com

Abstract. The Deep Web is constituted by data accessible through web pages, but not readily indexable by search engines, as they are returned in dynamic pages. In this paper we propose a framework for accessing Deep Web sources, represented as relational tables with so-called access limitations, with keyword-based queries. We formalize the notion of optimal answer and investigate methods for query processing. We also outline the main ideas of our implementation of a prototype system for Deep Web keyword search.

1 Introduction

It is reasonabe to assume that a user might want or need to query relational data with keywords – for instance, if he is accessing data whose structure he ignores; in this setting, the user is also free from having to know the query language. This idea has been around for over a decade [3] Since then, a vast corpus of research has been carried out in this field from the point of view of applications (see e.g. [8] for a survey). The problem has been recently formalized in [2], where a formal approach is provided, which is independent of the organization of data in tables.

In this paper we propose an approach to keyword search on so-called Deep Web sources. The Deep Web is constituted by data that are accessible only if queried through a web page, usually by filling in an HTML form. We represent Deep Web sources as relational tables; the necessity of filling in the aforementioned form with values forces one to access the relations with suitable selections; these restrictions are referred to as *access limitations*. It is well known that answering relational queries under access limitations requires the execution of a

* A preliminary version of this paper was published in the 9th Alberto Mendelzon Workshop on Foundations of Data Management, 2015, http://ceur-ws.org/Vol-1378

recursive query plan [5, 4]. This is also true in our setting, where a recursive extraction of tuples (or facts) is performed, according to the access limitations, so as to search for the keywords in the data.

Our main contributions are the following.

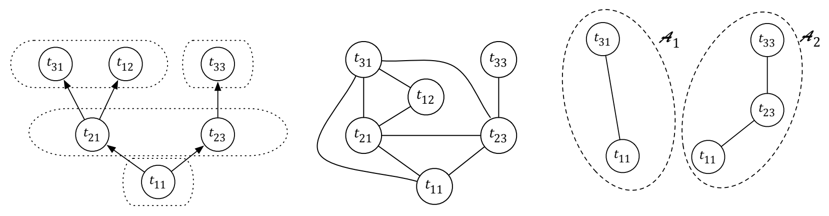
- We formally define the notion of keyword queries under access limitations, and the notion of answer (Section 2).
- We propose an algorithm for processing keyword queries under access limitations (Section 3).
- We discuss the computational complexity of our algorithm, showing that the problem of computing answers to keyword queries in our setting is tractable (Section 3).
- We outline the main features of a prototype system that processes keyword queries in the Deep Web (Section 4).

2 Problem definition

Basics. We model data sources as relations of a relational database and we assume that, albeit autonomous, they have "compatible" attributes. For this, we assume that the attributes of relations are defined over a set of *abstract domains* $\mathbf{D} = \{D_1, \ldots, D_m\}$, which, rather than denoting concrete value types (such as string or integer), represent data types at a higher level of abstraction (for instance, *car* or *country*). The set of all values is denoted by $\mathcal{D} = \bigcup_{i=1}^{n} D_i$.

In the following, we shall denote by $R(A_1, \ldots, A_k)$ a (relation) schema, by $dom(A) \in \mathbf{D}$ the domain of an attribute A, by r a relation over R, and by $\boldsymbol{r} = \{r_1, \ldots, r_n\}$ a (database) instance of a database schema $\mathbf{R} = \{R_1, \ldots, R_n\}$.

Access limitations. An *access pattern* Π for a schema $R(A_1, \ldots, A_k)$ is a mapping sending each attribute A_i into an *access mode*, which can be either input or output; A_i is correspondingly called an *input* (resp., *output*) *attribute* for R wrt. Π. For ease of notation, we shall mark input attributes with an 'i' superscript to distinguish them from the output ones. Let $A'_1, \ldots, A'_l$ be all the input attributes for R wrt. Π; any tuple $\langle c_1, \ldots, c_l \rangle$ such that $c_i \in dom(A'_i)$ for $1 \leq i \leq l$ is called a *binding* for R wrt. Π. An *access* α consists of an access pattern Π for a schema R and a binding for R wrt. Π; the *output* of such an access α on an instance $\boldsymbol{r}$ is the set $\mathcal{T} = \sigma_{A_1=c_1,\ldots,A_l=c_l}(r)$. Intuitively, we can only access a relation if we can provide a value for every input attribute. Given an instance $\boldsymbol{r}$ for a database schema $\mathbf{R}$, a set of access patterns $\boldsymbol{\Pi}$ for the relations in $\mathbf{R}$, and a set of values $\mathcal{C} \subseteq \mathcal{D}$, an *access path* (for $\mathbf{R}$, $\boldsymbol{\Pi}$ and $\mathcal{C}$) is a sequence of accesses $\alpha_1, \ldots, \alpha_n$ on $\boldsymbol{r}$ such that each value in the binding of α_i, $1 \leq i \leq n$, either occurs in the output of an access α_j with $j < i$ or is a value in $\mathcal{C}$. A tuple t in $\boldsymbol{r}$ is said to be *reachable* if there exists an access path P such that t is in the output of some access in P; the *reachable portion* $reach(\boldsymbol{r}, \boldsymbol{\Pi}, \mathcal{C})$ of $\boldsymbol{r}$ is the set of all reachable tuples in $\boldsymbol{r}$ given the values in $\mathcal{C}$.



(a) Access paths (b) Join graph (c) Answers

Fig. 1. Example 1: Reachable portion, corresponding join graph, and answers.

Keyword queries. A *keyword query* is a set of values in $\mathcal{D}$ called *keywords.*

Example 1 Consider a query $q = \{k_1, k_2\}$, a schema (with access patterns $\boldsymbol{\Pi}$) $\mathbf{R} = \{R_1(A_1^i, A_2), R_2(A_2^i, A_1), R_3(A_1^i, A_2, A_3)\}$, and an instance $\boldsymbol{r}$ such that

$$r_1 = \begin{array}{|cc|l}\hline A_1^i & A_2 & \\ \hline k_1 & c_1 & t_{11} \\ c_2 & c_3 & t_{12} \\ \hline\end{array} \quad r_2 = \begin{array}{|cc|l}\hline A_2^i & A_1 & \\ \hline c_1 & c_2 & t_{21} \\ c_4 & c_2 & t_{22} \\ c_1 & c_6 & t_{23} \\ \hline\end{array} \quad r_3 = \begin{array}{|ccc|l}\hline A_1^i & A_2 & A_3 & \\ \hline c_2 & c_1 & k_2 & t_{31} \\ c_5 & c_4 & k_2 & t_{32} \\ c_6 & c_7 & k_2 & t_{33} \\ \hline\end{array}$$

Figure 1(a) shows the reachable portion of $\boldsymbol{r}$ given the values in q along with the access paths used to extract it, with dotted lines enclosing outputs of accesses. ■

Given a set $\mathcal{T}$ of tuples, the *join graph* of $\mathcal{T}$ is a node-labelled undirected graph $\mathcal{T} = \langle N, E\rangle$ constructed as follows: *(i)* the nodes N are labelled with tuples of $\mathcal{T}$, and *(ii)* there is an arc between two nodes n_1 and n_2 if the tuples labelling n_1 and n_2 have at least one value in common.

Example 1 (cont.) The join graph of $\mathit{reach}(\boldsymbol{r}, \boldsymbol{\Pi}, q)$ is shown in Figure 1(b). ■

Definition 1 (Answer). *An answer to a keyword query q against a database instance $\boldsymbol{r}$ over a schema $\boldsymbol{R}$ with access patterns $\boldsymbol{\Pi}$ is a set of tuples $\mathcal{A}$ in $\mathit{reach}(\boldsymbol{r}, \boldsymbol{\Pi}, q)$ such that: (1) each $c \in q$ occurs in at least one tuple t in $\mathcal{A}$; (2) the join graph of $\mathcal{A}$ is connected; (3) for every subset $\mathcal{A}' \subseteq \mathcal{A}$ such that $\mathcal{A}'$ enjoys Condition 1 above, the join graph of $\mathcal{A}'$ is not connected.*

It is straightforward to see that there could be several answers to a keyword query; below we give a widely accepted criterium for ranking such answers [8].

Definition 2. *Let $\mathcal{A}_1, \mathcal{A}_2$ be two answers of a keyword query q on an instance $\boldsymbol{r}$ of size $|\mathcal{A}_1|$ and $|\mathcal{A}_2|$ respectively; we say that $\mathcal{A}_1$* is better than *$\mathcal{A}_2$, denoted $\mathcal{A}_1 \preceq \mathcal{A}_2$, if $|\mathcal{A}_1| \leq |\mathcal{A}_2|$. The* optimal *answers are those of minimum size.*

Example 1 (cont.) The sets $\mathcal{A}_1 = \{t_{11}, t_{31}\}$ and $\mathcal{A}_2 = \{t_{11}, t_{23}, t_{33}\}$ are answers to q; $\mathcal{A}_1$ is better than $\mathcal{A}_2$ and is the optimal answer to q. ■

3 Keyword-based answering in the Deep Web

We now present a vanilla algorithm to discuss the computational complexity of answering a keyword query q in the deep Web modeled as an instance $\boldsymbol{r}$ of a schema $\mathbf{R}$ with access patterns $\boldsymbol{\Pi}$. Example 1 shows that, in the worst case, we need to extract the whole reachable portion to obtain the tuples involved in an optimal answer. In fact, $\boldsymbol{s} = reach(\boldsymbol{r}, \boldsymbol{\Pi}, q)$ is actually a connected join graph, since every tuple in it is in some output of some access path starting from the values in the query (see for example Figure 1.a), but further paths may exist between tuples in $\boldsymbol{s}$ (see Figure 1.b). Therefore, query answering requires in general two main steps, described in Algorithm 1: *(i)* extract the reachable portion $\boldsymbol{s}$ of $\boldsymbol{r}$; *(ii)* if possible, remove tuples from $\boldsymbol{s}$ so that the obtained set satisfies the conditions of Definition 1, while minimizing its size.

Algorithm 1: Computing an optimal answer ($Answer(q, \boldsymbol{\Pi}, \boldsymbol{r})$)

```
Input: Keyword query $q$, access patterns $\boldsymbol{\Pi}$, instance $\boldsymbol{r}$ over $\boldsymbol{R}$
Output: Answer $\mathcal{A}$
 1. $\mathcal{A} := reachablePortion(\boldsymbol{r}, \boldsymbol{\Pi}, q)$; // see Algorithm 2
 2. if $\mathcal{A}$ does not contain all values in $q$ then return nil;
 3. else $prune(\mathcal{A}, q)$; // see Algorithm 3
 4. return $\mathcal{A}$;
```

A simple way of extracting the reachable portion, inspired by the procedure described in [4], is shown in Algorithm 2. This algorithm may be allowed to terminate early if the answer is not required to be optimal (flag ω set to $false$), and thus can stop as soon as the reachable portion contains all the keywords in the query. This is coherent with the *distinct root-based semantics* of keyword search in relational databases, which provides a tradeoff between quality of the result and efficiency of the method to evaluate it [8].

Algorithm 2: Reachable portion ($reachablePortion(\boldsymbol{r}, \boldsymbol{\Pi}, q)$)

```
Input: Instance $\boldsymbol{r}$ over $\boldsymbol{R}$, access patterns $\boldsymbol{\Pi}$, initial values $q$
Flag: boolean $\omega$ // if $\omega = true$ the answer is guaranteed to be optimal
Output: Reachable portion $RP$
 1. $RP := \emptyset$; $\mathcal{C} := \emptyset$;
 2. while an access can be made with a new binding $b$ for some $R \in \mathbf{R}$ wrt. $\boldsymbol{\Pi}$ using values in $\mathcal{C} \cup q$
 3.     $\mathcal{O}$ := output of access to $r$ over $R$ with binding $b$;
 4.     $RP := RP \cup \mathcal{O}$; // cumulating all the obtained tuples into $RP$
 5.     $\mathcal{C} := \mathcal{C} \cup \bigcup_{A \in R, t \in \mathcal{O}} \{t(A)\}$; // cumulating all the obtained values into $\mathcal{C}$
 6.     if $\mathcal{C} \supseteq q \wedge \neg\omega$ then break;
 7. return $RP$;
```

Basically, determining an optimal answer from the reachable portion corresponds to finding a Steiner tree of its join graph [8], i.e., a minimal-weight subtree of this graph involving a subset of its nodes. An efficient method for solving this problem in the context of keyword search over structured data is presented in [6], where a *q-fragment* can model our notion of answer. Yet, when optimality is not required, a simple technique (quadratic in the size of $\boldsymbol{r}$) to obtain an answer (steps 2–6 of Algorithm 3) consists in trying to remove any tuple from the set as long as it contains all the keywords and remains connected.

Algorithm 3: Pruning ($prune(\mathcal{T}, q)$)

```
Input:  Set of tuples $\mathcal{T}$, keyword query $q$
Flag:  boolean $\omega$ // if $\omega = true$ the answer is guaranteed to be optimal
Output:  Minimal set of tuples $\mathcal{T}$
 1. if $\omega$ then return a minimal subtree of the join graph of $\mathcal{T}$ that contains $q$;
 2. $\mathcal{T}' := \mathcal{T}$; $\mathcal{T}'' := \emptyset$;
 3. while $\mathcal{T}'' \neq \mathcal{T}'$
 4.     $\mathcal{T}'' := \mathcal{T}'$;
 5.     for each $t \in \mathcal{T}''$ if $\mathcal{T}' \setminus \{t\}$ is connected and $\mathcal{T}' \setminus \{t\} \supseteq q$ then $\mathcal{T}' := \mathcal{T}' \setminus \{t\}$;
 6. return $\mathcal{T}'$;
```

The extraction of the reachable portion of an instance $\boldsymbol{r}$ with access limitations can be implemented by a Datalog program over $\boldsymbol{r}$ [4], which can be evaluated in polynomial time in the size of the input [7]. In addition, in [6] it is shown that the optimal q-fragments of $\boldsymbol{r}$ can be enumerated in ranked-order with polynomial delay, i.e., the time for printing the next optimal answer is again polynomial in the size of $\boldsymbol{r}$. Hence, we can state the following preliminary result.

Theorem 1. *An optimal answer to a keyword query against a database instance with access limitations can be efficiently computed under data complexity.*

4 Implementation

We have implemented a prototype version of a system for processing keyword queries over a set of Deep Web sources. Given the nature of the search, we extract data on-the-fly and store them in main memory, conveniently represented in graph format. The keyword search algorithms are run on the extracted data in main memory rather than in a DBMS; the nature of keyword query processing in this setting suggests that relying on a relational DBMS does not provide any significant advantage.

The system is based on an underlying relational layer of Deep Web data called *Dataplex*[1]. The Dataplex framework was developed in Racket, a language of the Lisp/Scheme family. In Dataplex, the only field type is text (string), and tables are seen as containers with interface methods. The main idea behind Dataplex is that facts can have *lists* as arguments; facts belong to *shapes*, that are the equivalent of single-relation schemata. A shape is created with the call

[1] `http://thomaswlynch.com/liquid/liquid-doc/liquid/doc/liquid/index.html`

```
(dataplex:create-shape name length)
```

and a tuple can be placed into the created container by

```
(shape:insert name value_list).
```

Notice that in Dataplex data are stored, logically speaking, in two structures: one that holds the values, and the other that bridges references to these values; indices are used to improve performances in the accesses to data in Dataplex.

We are currently running experiments on keyword search by using the Dataplex framework, which we will publish soon. As it is natural to predict, the bottleneck in the efficiency of query processing is the extraction of data from the sources.

5 Discussion and future work

In this paper we provided preliminary insights on keyword search in the Deep Web. As future work on the problem, we plan to:

- devise optimization strategies for query answering; in particular, identify conditions under which an optimal answer can be derived without extracting the whole reachable instance;
- leverage known values (besides the keywords), modeled as relations with only one (output) attribute, to speed up the search for an optimal answer;
- study the problem in a scenario in which the domains of the keywords are known in advance: in this case schema-based techniques can be used;
- consider the case in which nodes and arcs of the join graph are weighed to model source availability and proximity, respectively.

Acknowledgments. Andrea Calì and Thomas Lynch acknowledge support by the EPSRC grant "Logic-based Integration and Querying of Unindexed Data" (EP/E010865/1).

References

1. A.V. Aho, C. Beeri, and J.D. Ullman. The theory of joins in r elational databases. ACM Trans. on Database Syst., 4(3):297314, 1979
2. R. Torlone. Towards a new Foundation for Keyword Search in Relational Databases. *AMW*, 2014.
3. V. Hristidis, Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. *VLDB*, pages 670–681, 2002.
4. A. Calì, D. Martinenghi. Querying Data under Access Limitations. In *ICDE*, pag. 50–59, 2008.
5. C. Li, E. Y. Chang. Answering queries with useful bindings. ACM Trans. Database Syst., 26(3), pages 313–343, 2001.
6. B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS*, pag. 173–182, 2006.
7. M. Vardi.The complexity of relational query languages. In *STOC*, pag. 137–146, 1982.
8. J. Xu Yu, L. Qin, and L. Chang. Search in Relational Databases: A Survey. *IEEE Data Eng. Bull.*, 33(1): 67–78, 2010.