Googling the Deep Web

Andrea Calì[1], Davide Martinenghi[2], and Riccardo Torlone[3]

[1]Birkbeck, University of London, UK, email: `andrea@dcs.bbk.ac.uk`
[2]Politecnico di Milano, Italy, email: `davide.martinenghi@polimi.it`
[3]Università Roma Tre, Italy, email: `torlone@dia.uniroma3.it`

Abstract. The Deep Web is constituted by data that are accessible through Web pages, but not indexable by search engines as they are returned in dynamic pages. In this paper we propose a conceptual framework for answering keyword queries on Deep Web sources represented as relational tables with so-called access limitations. We formalize the notion of optimal answer and characterize queries for which an answer can be found.

Keywords: Keyword Query, Access Pattern, Deep Web.

1 Introduction

It is well known that the data available on the Web through forms and in general in dynamic pages constitutes a corpus of data that is way larger than the Web data that are indexed by search engines. The former are commonly referred to as *Deep Web*, that is, data "hidden" in local databases whose content can only be accessed through queries. This happens for instance when we search for a flight on the Web site of an airline company – the data make sense only for a short time, after which they become obsolete due to the frequent changes of prices and availability. This immediately poses an interesting challenge, i.e., how to automatically retrieve relevant information from the Deep Web – a problem that has been investigated in recent years (see, e.g., [2, 4, 15] for discussion).

Usually, a data source in the Deep Web is modeled at the logical level by a relational table in which some columns, called *input* attributes, represent fields of a form that need to be filled in so as to retrieve data from the source, while all the others, called *output* attributes, represent values that are returned to the user. Consider for instance the following relations in which the i superscript denotes the input attributes.

$r_1 =$

$Dept^i$	Emp	
IT	John	t_{11}
AI	Mike	t_{12}

$r_2 =$

Emp^i	$Proj$	
John	P1	t_{21}
Ann	P2	t_{22}
Mike	P2	t_{23}

$r_3 =$

$Proj^i$	Emp	$Role$	
P1	John	DBA	t_{31}
P1	Ann	Analyst	t_{32}

Relation r_1 represents a form that, given a department, returns all the employees working in it; relation r_2 a form that, given an employee, returns all the projects he/she works on; and relation r_3 a form that, given a project, returns the employees working in it along with their role. These access modalities are commonly referred to as *access limitations*, in that data can only be queried according to given patterns.

Different approaches have been proposed in the literature for querying databases with access limitations: conjunctive queries [3], natural language [13], and SQL-like statements [11]. In this paper, we address the novel problem of accessing the Deep Web by just providing a set of keywords, in the same way in which we usually search for information on the Web with a search engine.

Consider for instance the case in which the user only provides the keywords "DBA" and "IT" for querying the portion of the Deep Web represented by the relations above. Intuitively, he/she is searching for employees with the DBA role in the IT department. Given the access limitations, this query can be concretely answered by first accessing relation r_1 using the keyword *IT*, which allows us to extract the tuple t_{11}. Then, using the value *John* in t_{11}, we can extract the tuple t_{21} from relation r_2. Finally, using the value *P1* in t_{21}, we can extract the tuples t_{31} and t_{32} from relation r_3. Now, since t_{31} contains *DBA*, it turns out that the set of tuples $\{t_{11}, t_{21}, t_{31}\}$ is a possible answer to the input query in that the set is connected (every two tuples in it share a constant) and contains the given keywords. However, the tuple t_{21} is somehow redundant and can be safely eliminated from the solution, since the set $\{t_{11}, t_{31}\}$ is also connected and contains the keywords. This example shows that, in this context, the keyword query answering problem can be involved and tricky, even in simple situations.

In the rest of this paper, we formally investigate this problem in depth. We first propose, in Section 2, a precise semantics of (optimal) answer to a keyword query in the Deep Web. We then tackle, in Section 3, the problem of finding an answer to a keyword query by assuming that the domains of the keywords are known in advance. This allows us to perform static analysis to immediately discard irrelevant cases from our consideration. Section 4 ends the paper with some conclusions and future works.

Related work To the best of our knowledge, ours is the first comprehensive approach to the problem of querying the Deep Web using keywords. Other works addressed keyword search on relational data previously extracted from the Deep Web [**?**]. This paper reports results previously published in [**?**], the paper where we illustrate our techniques for keyword search in the Deep Web.

The problem of query processing in the Deep Web has been widely investigated in the last years, with different approaches and under different perspectives including: data crawling [16], integration of data sources [9], query plan optimization [3], and generic structured query models [11]. However, none of them has tackled the problem that we have addressed in this paper.

The idea of querying structured data using keywords emerged more than a decade ago [1] as a way to provide high-level access to data and free the user from the knowledge of query languages and data organization. Since then, a lot of work has been done in this field (see, e.g., [18] for a survey) but never in the context of the Deep Web. This problem has been investigated in the context of various data models: relational [12], semi-structured [14], XML [8], and RDF [17]. Within the relational model, the common assumption is that an answer to a keyword query is a graph of minimal size in which the nodes represent tuples, the edges represent foreign key references between them, and the keywords occur in some node of the graph [10]. Our definition of query answer follows this line but it is more general, since it is only based on the presence of common values between tuples, while not forcing the presence of foreign keys.

The various approaches to keyword query answering over relational databases are commonly classified into two categories: *schema-based* and *schema-free*. Schema-based approaches [1, 10] make use, in a preliminary phase, of the database schema to build SQL queries that are able to return the answer. Conversely, schema-free approaches [7, 12] rely on exploration techniques over a graph-based representation of the whole database. Since the search for an optimal answer consists in finding a minimal *Steiner tree* on the graph, which is known to be an NP-Complete problem [6], the various proposals rely on heuristics aimed at generating approximations of Steiner trees. Our approach makes use of the schema of the data sources but cannot be classified in any of the approaches above since, given the access limitations, it rather relies on building a minimal query plan of accesses to the data sources.

2 Preliminaries and problem definition

We model data sources as relations of a relational database and we assume that, albeit autonomous, they have "compatible" attributes. For this, we fix 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*). Therefore, in an abstract domain an object is uniquely represented by a value. The set of all values is denoted by $\mathcal{D} = \bigcup_{i=1}^{n} D_i$. For simplicity, we assume that all abstract domains are disjoint. We then say that a *(relation) schema* r, customarily indicated as $r(A_1, \ldots, A_k)$, is a set of attributes $\{A_1, \ldots, A_k\}$, each associated with an abstract domain $dom(A_i) \in \mathbf{D}$, $1 \leq i \leq k$. A *database schema* $\mathcal{S}$ is a set of *schemas* $\{r_1, \ldots, r_n\}$.

As usual, given a schema r, a *tuple* t over r is a function that associates a value $c \in dom(A)$ with each attribute $A \in r$, and a *relation instance* $r^{\mathcal{I}}$ of r is a set of tuples over r. For simplicity, we also write $dom(c)$ to indicate the domain of c. A (database) instance $\mathcal{I}$ of a database schema $\mathcal{S} = \{r_1, \ldots, r_n\}$ is a set of relation instances $\{r_1^{\mathcal{I}}, \ldots, r_n^{\mathcal{I}}\}$, where $r_i^{\mathcal{I}}$ denotes the relation instance of r_i in $\mathcal{I}$.

For the sake of simplicity, in the following we assign the same name to attributes of different schemas that are defined over the same abstract domain.

Definition 1 (Access pattern). *An* access pattern Π *for a schema* $r(A_1, \ldots, A_k)$ *is a mapping* $\Pi : \{A_1, \ldots, A_k\} \rightarrow M$*, where* $M = \{i, o\}$ *is called* access mode*, and* i *and* o *denote* input *and* output*, respectively;* A_i *is correspondingly called an* input *(resp.,* output*)* attribute *for* r *wrt* Π.

Henceforth, we denote input attributes with an 'i' superscript, e.g., A^i. Moreover, we assume that each relation has exactly one access pattern.

Definition 2 (Binding). *Let* $A'_1, \ldots, A'_\ell$ *be all the input attributes for* r *wrt* Π*; any tuple* $\mathscr{b} = \langle c_1, \ldots, c_\ell \rangle$ *such that* $c_i \in dom(A'_i)$ *for* $1 \leq i \leq \ell$ *is called a* binding *for* r *wrt* Π.

Definition 3 (Access). *An* access *is a pair $\langle \Pi, \mathscr{b} \rangle$, where Π is an access pattern for a schema r and $\mathscr{b}$ is a binding for r wrt Π. The* output *of such an access on an instance $\mathcal{I}$ is the set $\mathcal{T}$ of all tuples in the relation $r^{\mathcal{I}} \in \mathcal{I}$ over r that match the binding, i.e., such that $\mathcal{T} = \sigma_{A_1=c_1,\dots,A_\ell=c_\ell}(r)$.*

Intuitively, we can only access a relation if we can provide a binding for it, i.e., a value for every input attribute.

Definition 4 (Access path). *Given an instance $\mathcal{I}$ for a database schema $\mathcal{S}$, a set of access patterns $\mathbf{\Pi}$ for the relations in $\mathcal{S}$, and a set of values $\mathcal{C} \subseteq \mathcal{D}$, an* access path *on $\mathcal{I}$ (for $\mathcal{S}$, $\mathbf{\Pi}$ and $\mathcal{C}$) is a sequence $\xrightarrow{\mathscr{b}_1}_{r_1^{\mathcal{I}}} \mathcal{T}_1 \xrightarrow{\mathscr{b}_2}_{r_2^{\mathcal{I}}} \cdots \xrightarrow{\mathscr{b}_n}_{r_n^{\mathcal{I}}} \mathcal{T}_n$, where, for $1 \leq i \leq n$, (i) $\mathscr{b}_i$ is a binding for a relation $r_i \in \mathcal{S}$ wrt a pattern $\Pi_i \in \mathbf{\Pi}$ for r_i, (ii) $\mathcal{T}_i$ is the output of access $\langle \Pi_i, \mathscr{b}_i \rangle$ on $\mathcal{I}$, and (iii) each value in $\mathscr{b}_i$ either occurs in $\mathcal{T}_j$ with $j < i$ or is a value in $\mathcal{C}$.*

Definition 5 (Reachable portion). *A tuple t in $\mathcal{I}$ is said to be* reachable *given $\mathcal{C}$ if there exists an access path P (for $\mathcal{S}$, $\mathbf{\Pi}$ and $\mathcal{C}$) such that t is in the output of some access in P; the* reachable portion *$reach(\mathcal{I}, \mathbf{\Pi}, \mathcal{C})$ of $\mathcal{I}$ is the set of all reachable tuples in $\mathcal{I}$ given $\mathcal{C}$.*

In the following, we will write $\mathcal{S}^{\mathbf{\Pi}}$ to refer to schema $\mathcal{S}$ under access patterns $\mathbf{\Pi}$.

Example 1. Consider the following instance $\mathcal{I}$ of a schema $\mathcal{S}^{\mathbf{\Pi}} = \{r_1(A^i, B), r_2(B^i, A), r_3(A^i, B, C)\}$.

$r_1 =$

A^i	B	
a_1	b_1	t_{11}
a_2	b_3	t_{12}

$r_2 =$

B^i	A	
b_1	a_2	t_{21}
b_2	a_2	t_{22}
b_1	a_4	t_{23}

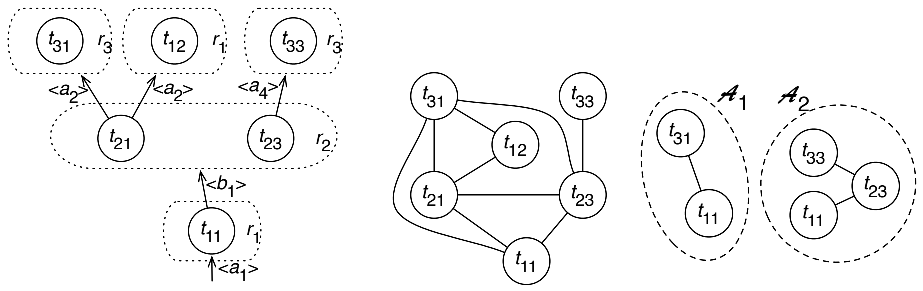
$r_3 =$

A^i	B	C	
a_2	b_1	c_1	t_{31}
a_3	b_2	c_2	t_{32}
a_4	b_4	c_1	t_{33}

Then, for instance, $\{t_{11}\}$ is the output of the access with binding $\langle a_1 \rangle$ wrt $r_1(A^i, B)$, and $\xrightarrow{\langle a_1 \rangle}_{r_1^{\mathcal{I}}} \{t_{11}\} \xrightarrow{\langle b_1 \rangle}_{r_2^{\mathcal{I}}} \{t_{21}, t_{23}\}$, is an access path for $\mathcal{S}$, $\mathbf{\Pi}$ and $\mathcal{C} = \{a_1\}$, since, given $\mathcal{C}$, we can extract t_{11} from r_1 and, given $\{b_1\}$ from t_{11}, we can extract t_{21} and t_{23} from r_2. The reachable portion of $\mathcal{I}$, given $\mathcal{C}$, is $reach(\mathcal{I}, \mathbf{\Pi}, \mathcal{C}) = \{t_{11}, t_{12}, t_{21}, t_{23}, t_{31}, t_{33}\}$, while $\{t_{22}, t_{32}\} \cap reach(\mathcal{I}, \mathbf{\Pi}, \mathcal{C}) = \emptyset$. Figure 1a shows the reachable portion $\mathcal{I}'$ of $\mathcal{I}$ given $\mathcal{C}$ along with the access paths used to extract it, with dotted lines enclosing outputs of accesses. ■

The definition of answer to a keyword query in our setting requires the preliminary notion of join graph.

Definition 6 (Join graph). *Given a set $\mathcal{T}$ of tuples, the* join graph *of $\mathcal{T}$ is a node-labeled undirected graph $\langle N, E \rangle$ constructed as follows: (i) the nodes N are labeled with tuples of $\mathcal{T}$, with a one-to-one correspondence between tuples of $\mathcal{T}$ and nodes of N; and (ii) there is an arc between two nodes n_1 and n_2 whenever the tuples labeling n_1 and n_2 have at least one value in common.*



(a) Reachable portion $\mathcal{I}'$, given $\{a_1\}$. (b) Join graph of $\mathcal{I}'$. (c) Two answers to KQ q.

Fig. 1: Illustration of Examples 1, 2, and 3.

Example 2. Consider instance $\mathcal{I}$ of Example 1 and the reachable portion $\mathcal{I}'$ of $\mathcal{I}$ given $\{a_1\}$, shown in Figure 1a. The join graph of $\mathcal{I}'$ is shown in Figure 1b. ■

A *keyword query* (KQ) is a non-empty set of values in $\mathcal{D}$ called *keywords*.

Definition 7 (Answer to a KQ). *An answer to a KQ q against a database instance $\mathcal{I}$ over a schema $\mathcal{S}^{\mathbf{\Pi}}$ is a set of tuples $\mathcal{A}$ in reach$(\mathcal{I}, \mathbf{\Pi}, q)$ such that: (i) each keyword $k \in q$ occurs in at least one tuple t in $\mathcal{A}$; (ii) the join graph of $\mathcal{A}$ is connected; (iii) no proper subset $\mathcal{A}' \subset \mathcal{A}$ satisfies both Conditions (i) and (ii) above.*

It is straightforward to see that there could be several answers to a KQ; below we give a widely accepted criterion for ranking such answers [18].

Definition 8. *Let $\mathcal{A}_1, \mathcal{A}_2$ be two answers to a KQ q on an instance $\mathcal{I}$. We say that $\mathcal{A}_1$* is better than *$\mathcal{A}_2$ if $|\mathcal{A}_1| \leq |\mathcal{A}_2|$. The* optimal *answers are those of minimum size.*

Example 3. Consider a KQ $q = \{a_1, c_1\}$ over the instance $\mathcal{I}$ of Example 1. Figure 1a shows two possible answers: $\mathcal{A}_1 = \{t_{11}, t_{31}\}$ and $\mathcal{A}_2 = \{t_{11}, t_{23}, t_{33}\}$. $\mathcal{A}_1$ is better than $\mathcal{A}_2$ and is the optimal answer to q. ■

3 Detecting non-answerable queries

For convenience of notation, we sometimes write c: D to denote value c and indicate that $dom(c) = D$. In addition, in our examples, the name of an attribute will also indicate its abstract domain.

3.1 Compatible queries

In order to focus on meaningful queries, we semantically characterize queries for which an answer might be found.

Definition 9 (Compatibility). *A KQ q is said to be* compatible *with a schema $\mathcal{S}$ if there exist a set of access patterns $\mathbf{\Pi}$ and an instance $\mathcal{I}$ over $\mathcal{S}^{\mathbf{\Pi}}$ such that there is an answer to q against $\mathcal{I}$.*

Example 4. The KQ $q_1 = \{a\colon A, c\colon C\}$ is not compatible with schema $\mathcal{S}_1 = \{r_1(A,B), r_2(C,D)\}$, since no set of tuples from $\mathcal{S}$ containing all the keywords in q_1 can ever be connected, independently of the access patterns for $\mathcal{S}_1$. Conversely, q_1 is compatible with $\mathcal{S}_2 = \{r_1(A,B), r_3(B,C)\}$, as witnessed by a possible answer $\{r_1(a,b), r_3(b,c)\}$ and patterns $\mathbf{\Pi}$ such that $\mathcal{S}_2^{\mathbf{\Pi}} = \{r_1(A^i,B), r_3(B,C)\}$.

Similarly, KQ $q_2 = \{a\colon A, a'\colon A\}$ is compatible with a schema $\mathcal{S}_3 = \{r_1(A,B)\}$, as witnessed by a possible answer $\{r_1(a,b), r_1(a',b)\}$ and patterns $\mathbf{\Pi}$ such that $\mathcal{S}_3^{\mathbf{\Pi}} = \{r_1(A^i,B)\}$. However, q_2 is not compatible with a schema $\mathcal{S}_4 = \{r_4(A)\}$, since a unary relation, alone, can never connect two keywords. ■

Checking compatibility of a KQ with a schema essentially amounts to checking reachability on a graph. The main idea is that in order for an answer to ever be possible, we must find an instance that exhibits a witness (i.e., a set of tuples) satisfying all the conditions of Definition 7.

3.2 Answerable queries

A stricter requirement than compatibility is given by the notion of answerability.

Definition 10 (Answerability). *A KQ q is* answerable *against a schema $\mathcal{S}^{\mathbf{\Pi}}$ if there is an instance $\mathcal{I}$ over $\mathcal{S}^{\mathbf{\Pi}}$ such that there is an answer to q against $\mathcal{I}$.*

Example 5. Consider KQ $q = \{a\colon A, c\colon C\}$ and schema $\mathcal{S}_1^{\mathbf{\Pi}} = \{r(A^i,B), s(B,C,D^i)\}$. Although q is compatible with $\mathcal{S}_1$, it is not answerable against $\mathcal{S}_1^{\mathbf{\Pi}}$, since no tuple from $\mathcal{S}_1$ can be extracted under $\mathbf{\Pi}$ (no values for domain D are available). Conversely, q is answerable against $\mathcal{S}_2^{\mathbf{\Pi}} = \{r(A^i,B), s(B^i,C,D)\}$, since an answer like $\{r(a,b), s(b,c,d)\}$ could be extracted by first accessing r with binding $\langle a\rangle$, thus extracting value b, and then s with binding $\langle b\rangle$. ■

In order to check answerability, we need to check that all the required relations can be accessed according to the access patterns. To this end, we first refer to a schema enriched with unary relations representing the keywords in the KQ. [1]

Definition 11 (Expanded schema). *Let q be a KQ over a schema $\mathcal{S}^{\mathbf{\Pi}}$. The* expanded schema *$\mathcal{S}_q^{\mathbf{\Pi}}$ of $\mathcal{S}^{\mathbf{\Pi}}$ wrt. q is defined as $\mathcal{S}_q^{\mathbf{\Pi}} = \mathcal{S}^{\mathbf{\Pi}} \cup \{r_c(C) | c \in q\}$, where r_c is a new unary relation, not occurring in $\mathcal{S}^{\mathbf{\Pi}}$, whose only attribute C is an output attribute with abstract domain $dom(C) = dom(c)$.*

Then, we use the notion of dependency graph (d-graph) to denote output-input dependencies between relation arguments, indicating that a relation under access patterns needs values from other relations.

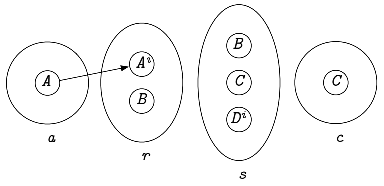
[1] If other values are known besides the keywords, this knowledge may be represented by means of appropriate unary relations with output mode in the schema.

Definition 12 (d-graph). *Let q be a KQ over a schema $\mathcal{S}^{\mathbf{\Pi}}$. The* d-graph $G_q^{\mathcal{S}^{\mathbf{\Pi}}}$ *is a directed graph $\langle \mathcal{N}, \mathcal{E} \rangle$ defined as follows. For each attribute A of each relation in the expanded schema $\mathcal{S}_q^{\mathbf{\Pi}}$, there is a node in $\mathcal{N}$ labeled with A's access mode and abstract domain. There is an arc $u \frown v$ in $\mathcal{E}$ whenever: (i) u and v have the same abstract domain; (ii) u is an output node; and (iii) v is an input node.*

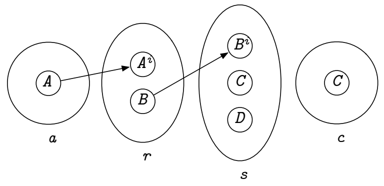
Some relations are made invisible by the access patterns and can be discarded.

Definition 13 (Visibility). *An input node $v_n \in \mathcal{N}$ in a d-graph $\langle \mathcal{N}, \mathcal{E} \rangle$ is* visible *if there is a sequence of arcs $u_1 \frown v_1, \ldots, u_n \frown v_n$ in $\mathcal{E}$ such that (i) u_1's relation has no input attributes, and (ii) v_i's and u_{i+1}'s relation are the same, for $1 \leq i \leq n-1$. A relation is visible if all of its input nodes are.*

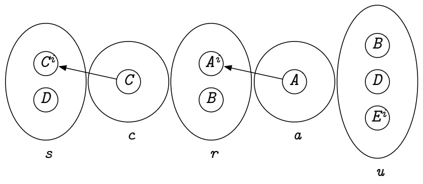
Example 5 (cont.). Consider the KQ and the schemas from Example 5. The d-graphs $G_q^{\mathcal{S}_1^{\mathbf{\Pi}}}$ and $G_q^{\mathcal{S}_2^{\mathbf{\Pi}}}$ are shown in Figures 2a and Figure 2b, respectively. ■



(a) D-graph $G_q^{\mathcal{S}_1^{\mathbf{\Pi}}}$ from Example 5; s is not visible.



(b) D-graph $G_q^{\mathcal{S}_2^{\mathbf{\Pi}}}$ from Example 5; all relations are visible.



(c) D-graph $G_q^{\mathcal{S}^{\mathbf{\Pi}}}$ from Example 6; u is not visible.

Fig. 2: D-graphs from the Examples.

Answerability of a KQ q is checked by means of compatibility with a schema in which all non-visible relations have been eliminated.

Example 6. Consider a KQ $q = \{a\colon A, c\colon C\}$ and a schema $\mathcal{S}^{\mathbf{\Pi}} = \{r(A^i, B), s(C^i, D), u(B, D, E^i)\}$. Relation u is not visible in $G_q^{\mathcal{S}^{\mathbf{\Pi}}}$ (Figure 2c). Then, q is not answerable in $\mathcal{S}^{\mathbf{\Pi}}$, since q is not compatible with schema $\{r(A, B), s(C, D)\}$ (i.e., $\mathcal{S}$ without relation u). ■

4 Conclusions and future work

We have defined the problem of keyword search in the Deep Web. We are currently working on minimizing the number of accesses to the data sources.

We believe that several interesting issues can be studied in the framework defined in this paper. We plan, e.g., to leverage known values (besides the keywords) and ontologies to speed up the search for an optimal answer as well as to consider the case in which nodes and arcs of the join graph are weighted to model source availability and proximity, respectively.

References

1. Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.
2. Meghyn Bienvenu et al. Dealing with the deep web and all its quirks. In *Proc. of VLDS*, pages 21–24, 2012.
3. Andrea Calì and Davide Martinenghi. Querying data under access limitations. In *ICDE*, pages 50–59, 2008.
4. Andrea Calì and Davide Martinenghi. Querying the deep web. In *EDBT*, pages 724–727, 2010.
5. Andrea Calì, Davide Martinenghi, and Riccardo Torlone. Keyword search in the deep web. In *Proc. of the 9th AMW*, 2015.
6. M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing Steiner minimal trees. *SIAM J. on Applied Mathematics*, 32(4):835–859, 1977.
7. Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD*, pages 927–940, 2008.
8. Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. Xrank: ranked keyword search over xml documents. In *SIGMOD*, pages 16–27, 2003.
9. Bin He, Zhen Zhang, and Kevin Chen-Chuan Chang. Metaquerier: querying structured web sources on-the-fly. In *Proc. of SIGMOD*, pages 927–929, 2005.
10. Vagelis Hristidis and Yannis Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.
11. Hasan M. Jamil and Hosagrahar V. Jagadish. A structured query model for the deep relational web. In *CIKM*, pages 1679–1682, 2015.
12. Benny Kimelfeld and Yehoshua Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS*, pages 173–182, 2006.
13. Jens Lehmann et al. Deqa: Deep web extraction for question answering. In *ISWC'12*, pages 131–147. Springer-Verlag, 2012.
14. Guoliang Li et al. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008.
15. Jayant Madhavan, Loredana Afanasiev, Lyublena Antova, and Alon Y. Halevy. Harnessing the deep web: Present and future. In *CIDR*, 2009.
16. Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *VLDB*, pages 129–138, 2001.
17. Thanh Tran et al. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE*, pages 405–416, 2009.
18. Jeffrey Xu Yu, Lu Qin, and Lijun Chang. Keyword search in relational databases: A survey. *IEEE Data Eng. Bull.*, 33(1):67–78, 2010.