Querying the Deep Web

Andrea Calì
Oxford-Man Institute and Computing Laboratory
University of Oxford, UK
andrea.cali@comlab.ox.ac.uk

Davide Martinenghi
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
davide.martinenghi@polimi.it

Abstract. Data stored outside Web pages and accessible from the Web, typically through HTML forms, consitute the so-called Deep Web. Such data are of great value, but difficult to query and search. We survey techniques to optimize query processing on the Deep Web, in a setting where data are represented in the relational model. We illustrate optimizations both at query plan generation time and at runtime, highlighting the role of integrity constraints. We discuss several prototype systems that address the query processing problem.

1. INTRODUCTION

The term *Deep Web* (sometimes also called *Hidden Web*) refers to the data content that is created dynamically as the result of a specific search on the Web. In this respect, such content resides outside Web pages, and is only accessible through interaction with the Web site – typically via HTML forms. It is believed that the size of the Deep Web is several orders of magnitude larger than that of the so-called *Surface Web*, i.e., the Web that is accessible and indexable by search engines.

Usually, data sources accessible through Web forms are modeled by relations that require certain fields to be selected – i.e., some fields in the form need to be filled in. These requirements are commonly referred to as *access limitations* in that access to data can only take place according to given patterns. Besides data accessible through Web forms, access limitations may also occur *(i)* in legacy systems where data scattered over several files are wrapped as relational tables, and *(ii)* in the context of Web services, where similar restrictions arise from the distinction between input parameters and output parameters.

In such contexts, computing the answer to a user query cannot be done as in a traditional database; instead, a query plan is needed that provides the best answer possible while complying with the access limitations. We illustrate the semantics of answers to queries over data sources under access limitations and present techniques for query answering in this context. We show different techniques to optimize query answering both at the time of query plan generation and at the time of execution of the query plan. We analyze the impact on query answering of integrity constraints of the kind that is usually found in database schemata, expressed on the source schemata. Finally, we present prototype systems that are aimed at querying the Deep Web, and show their achievements.

2. FRAMEWORK

There are several approaches to querying the Deep Web in the literature. Commonly, each data source accessible through a Web form is modeled as a relational table, whose corresponding (relational) predicate has an annotation that specifies a subset of the arguments (attributes) as *input* (or *must-select*, or *must-bind*, hence the term *binding pattern* or *access pattern* for the annotation), and the others as *output*. To access the relation, a query must necessarily make a selection on all the input attributes. Limitations in the way in which relations can be accessed were initially studied in the context of logic programs with (input or output) access modes [5], but a more thorough analysis of query processing issues involving relations with access patterns has emerged within the field of information integration [20, 7, 8, 22].

An important aspect of modeling the Deep Web is the notion of *abstract domain* of an attribute: to each attribute (argument of a relational predicate) an abstract domain is associated, that specifies the kind of values appearing in that attribute, in relationship with the system domain. For example, an abstract domain "plate_no" would indicate that the corresponding values are plate numbers; this is essential for optimizing query answering, and the choice of the abstract domain also raises design issues.

Access limitations make query processing significantly harder, since any traditional query plan that does not take binding patterns into account might attempt to access to a relation without providing all the necessary values for its input fields. To this end, different classifications of queries have emerged in this context with respect to their potential to retrieve the query answers. We provide below examples of the most relevant query classes for the simple language of *conjunctive queries* (i.e., select-project-join queries).

Consider the following relation schemata, where the access modes (I for input, O for output) are indicated as superscripts of the corresponding attribute names:

$r_1(\mathit{Title}^{\text{o}}, \mathit{City}^{\text{I}}, \mathit{Artist}^{\text{o}})$ that associates to a given city the song title and name of artist performing it in that city, and

$r_2(\mathit{Artist}^{\text{I}}, \mathit{Nation}^{\text{o}}, \mathit{City}^{\text{o}})$ that associates to a given artist name the nation and city of birth of the artist.

The query $q_f(A) \leftarrow r_2(A, \mathit{italy}, C), r_1(T, \mathit{ny}, A)$ requests the names of the Italian artists having performed in New York. Such a query cannot be executed according to a naive query plan that answers each subgoal in a left-to-right order, since a variable (A) occurs in an input attribute of r_2 but not in any previous subgoal (r_2 occurs in the leftmost subgoal) and therefore is not yet bound to a value.

Conversely, the query $q_e(A) \leftarrow r_1(T, \mathit{ny}, A), r_2(A, \mathit{italy}, C)$, requesting the same thing as q_f, is *executable* "as is" from left to right (aka *well-moded*), since the constant *ny* occurs in the only input position of r_1, and the input position of r_2 is occupied by variable A, which already occurs in r_1, and therefore is bound to a value. In the left-to-right execution of q_e, first r_1 is accessed with the input value *ny*, with the effect of populating variables T (title) and A (artist name) with corresponding values, and then r_2 is accessed with the value(s) populating A, if any, thus returning values for the corresponding nation and city (C). After the call to r_2, the artist names associated with nation *italy* will be returned as results. For executable queries, it is always possible to retrieve the *complete answer*, i.e., the answer that would be found by a traditional query plan if no access limitations were present.

Note also that, although q_f is not executable, it admits a simple reordering of subgoals that transforms it into a query (q_e, in this case) that is executable in a left-to-right order (namely, by swapping the two subgoals). Queries with this property are said to be *feasible*. Feasible queries, too, always admit retrieving the complete answer.

The query $q_s(A) \leftarrow r_2(A, \mathit{italy}, C), r_1(T, \mathit{ny}, A), r_1(T, C', A)$ is as q_f but has an additional subgoal that further requires artists A to have performed the same song T also in a city C' (possibly, but not necessarily, different from *ny*). Such a query is neither executable nor feasible, since in any reordering of the subgoals, $r_1(T, C', A)$ will have variable C' in an input position, and C' does not occur elsewhere in the query (and thus cannot be bound to a value). However, the new subgoal $r_1(T, C', A)$ is redundant, since it does not pose any new requirement to the query. More formally, it can be shown that query q_s is equivalent to query q_f (and to q_e, too). A query, such as q_s, equivalent to a feasible query is said to be *stable*. Stable queries, too, always allow retrieving the complete answer. Stability is tightly related with the traditional query containment problem and, in general, can also be cast as an instance of the problem of answering queries using views [10]. Note that some authors [15, 18, 6] call *feasible* the stable queries, and *orderable* the feasible queries.

As discussed, a query plan for the Deep Web must comply with the access limitations by feeding all input fields of the involved relations with appropriate values of the corresponding abstract domain, i.e., either values that are initially known (such as the constant values occurring in the query) or values that have already been extracted during the query answering process. In some cases, this makes it impossible to always retrieve the complete answer: the set of answer tuples that can be disclosed for the query (i.e., the so-called *maximal answer* or *reachable certain answer*) is generally only a subset of the complete answer. Moreover, in general, an execution via a recursive query plan is needed, even for non-recursive classes of queries, such as conjunctive queries. Consider, e.g., the query $q_a(A) \leftarrow r_2(A, \mathit{italy}, \mathit{modena})$, requesting the names of artists born in Modena, Italy. This query has one single subgoal with a variable (A) occurring in an input position, so no executable reordering exists. For query q_a, the complete answer cannot, in general, be found. However, it is possible to adopt a *recursive* extraction strategy that makes use of all the constants known from the query and of the abstract domains associated with the relation attributes in order to find the maximal answer. Recursion comes from the fact that output values from one relation can be used in input fields of another relation, and there might be cyclic input-output dependencies among the relations in the schema. In this particular example, one may consider that the attribute *City* in r_1 has the same domain as *City* in r_2, i.e., the values used for the former can also be used for the latter, and vice versa; similarly for *Artist*. One could then exploit the only known value for the domain *City* (*modena*) and use it to access r_1, which indeed requires a city name as input. Although this access is unrelated with the query q_a, where r_1 is not even mentioned, it can provide values for artist names and song titles as output; in turn, these artist names can be used to access r_2 and retrieve values for nations and cities; the new city names can be used to access r_1 again, and so on. This possibly lengthy discovery process consists in disclosing all the content of the relations that can be extracted with the given initial knowledge; at the end of the process, when no new values can be discovered, the original query q_a can be evaluated over the data retrieved so far. The set of answer tuples obtained in this way is the maximal answer.

The expensive process required by the evaluation of the maximal answer raises several interesting optimization problems that will be discussed in the next section.

3. QUERY OPTIMIZATION IN THE DEEP WEB

Here, we illustrate the basic optimization techniques that can be applied to the evaluation of the maximal answer. The problem of minimizing the accesses to compute the maximal answer has been addressed for different classes of queries, both at query definition time (Section 3.1) and at run-time, and possibly with integrity constraints in the schema (Section 3.2).

Query optimization may also be obtained by query minimization, which is commonly done via rewriting based on query containment. However, the presence of access limitations makes query containment intrinsically harder. We will give hints at techniques for solving the problem in Section 3.3.

3.1 Static optimization

The techniques for generating a naive query plan computing the maximal answer are well-established by now. However, in a web environment, it is crucial to *minimize* the accesses to remote sources, which are potentially very slow. Notice that a recursive query plan that finds the maximal answer to a conjunctive query over web sources will in general also need to query sources that do not occur in the query, since they may provide useful data for overcoming access limitations.

The first important optimization in this context consists in determining which sources are *relevant* to a given query, so that non-relevant sources can be excluded, thus saving query execution time. The problem of determining relevant sources at query definition time (static optimization) has only been solved for limited classes of conjunctive queries in the case of relations with exactly one access pattern [14, 4], while a general solution is still unavailable.

After excluding non-relevant relations from a query plan, care must also be taken to avoid all accesses (to relevant relations) that are unnecessary for the computation of the maximal answer. Initial attempts are shown in [4] for determining, at query definition time, a *minimal* set of accesses to sources, according to a notion of minimality that arises naturally in the context of web sources.

3.2 Run-time optimization and impact of integrity constraints

The execution of a query plan under access limitations can be further improved at run time, by making use of the data being extracted from the web sources. In particular, when integrity constraints are enforced locally on the web sources, some of the accesses to sources planned by a query plan may turn out to be irrelevant w.r.t. the extracted data, i.e., immaterial for the computation of new answer tuples. In such cases, the accesses can be avoided. Relevant notions of access minimization for run-time optimization are discussed in [2], where results are available in the case of schemata with particular kinds of integrity constraints, namely functional dependencies and simple full-width inclusion dependencies (a restricted kind of inclusion dependencies that involve all the attributes of a relation). The latter kind of dependencies is particularly interesting in this context, since it can be used to assert that different relations are equivalent, and thus capture the notion of relations with multiple access patterns. Techniques for dynamic optimization for minimization of accesses under more expressive classes of integrity constraints are yet to be explored.

Stability, feasibility, and executability are well understood by now [13, 21], and several results are available that also cover the cases of schemata with integrity constraints and relations with possibly more than one access pattern [15, 18, 6]. In particular, access patterns can be encoded into integrity constraints of a suitable form, which can then be processed together with the other constraints.

3.3 Query containment

The problem of checking query containment is highly relevant in query optimization and can also be defined in the case of relations with access patterns [17], with a semantics that differs from the traditional one. The decision problem $q_1 \subseteq_{ap} q_2$ for queries q_1 and q_2 under access patterns asks whether, for every database instance D, the maximal answer to q_1 in D is always a subset of the maximal answer to q_2 in D. Although several algorithms have been proposed to address this problem, the currently available complexity bounds (obtained by a chase-based technique called *crayfish chase* [3]) are not known to be tight. Furthermore, query containment in the presence of access patterns with the above semantics has only been studied for conjunctive queries over relations with exactly one access pattern and no integrity constraints in the schema.

4. APPLICATIONS

We now discuss applications of query answering techniques for the Deep Web.

4.1 Deep Web and information integration

As mentioned above, querying a set of Deep Web sources is a way of integrating the information stored into them. When integrating heterogeneous data sources, a so-called *mediated schema*, usually with expressive database constraints, serves as a representation of the application domain or, in other words, as a *domain ontology*. The data relative to the mediated schema reside at the sources, and in *virtual* integration the mediated schema is not populated. The specification of the semantic relationship between the individual sources and the mediated schema is expressed by means of *views* [10]. Techniques for answering queries using views in the presence of access limitations are presented in [7, 6].

4.2 Systems

We briefly discuss systems that tackle the problem of querying the Deep Web at different levels.

(1) Google search engine's approach to the Deep Web is based on surfacing [16], i.e., precomputing queries to Deep Web sources through Web forms, collecting the results, and indexing them as if they were static pages. This approach, which is opposite to virtual integration, is suitable for a search engine due to its domain-independence and efficiency. It follows roughly the same ideas and extensional strategy as the HiWe system [19]. Technical challenges arise in surfacing, such as scaling w.r.t. the number of involved domains, and considering the semantics of content exposed by the precomputing phase. Another recent system developed at Google is *Octopus* [1], which combines search with integration, together with data extraction and cleaning. Octopus, in particular, enables the user to specify the semantics of Web data, and to define new data sets from those retrieved from the Web.

(2) The systems *MetaQuerier* [11] and *WISE-integrator* [12] adopt the virtual integration approach to the Deep Web, under several aspects. They tackle the schema matching problem, focusing on the specification of semantic relationships between different Deep Web sources. Both systems aim at scaling virtual integration to a large number of sources.

(3) Toorjah [4] is a scientific tool developed at Oxford University, specifically tailored towards query optimization in querying Deep Web sources. Its state-of-the-art query planning techniques are able to minimize the number of individual accesses to sources during the execution of query plans.

Toorjah also sports a mechanism for finely controlling the flow of data tuples from the sources, so as to achieve parallel and faster query execution.

(4) It is also worth mentioning SeCo[1], a large ongoing European project started in 2008 and coordinated by Politecnico di Milano, aiming at the development of tools and technologies for answering multi-domain queries on the Web that state-of-art search engines are unable to address.

We do not discuss tools that address related issues such as wrapping and annotation of unstructured content, which are vital for querying the Deep Web, but not central w.r.t. the topics here covered. Since we consider sources represented (or wrapped) in relational format, we just mention the *Lixto* system [9], which is capable of wrapping (semi)structured content from HTML pages by means of a semi-automated, user-guided specification.

5. DISCUSSION AND OPEN ISSUES

Several issues related with the above mentioned techniques for query processing under access limitations are still open. First, we point out that a solution to the minimization problem for the case of sources allowing *multiple* access patterns is still not available. State-of-art solutions only account for data sources with a single pattern. Second, static techniques for access minimization are not known when integrity constraints are imposed on the schema. The literature only considers dynamic solutions that, based on the data being extracted and the constraints in the schema, avoid some of the accesses that can be deemed as irrelevant for computing the query answer. Third, a tight complexity bound for the conjunctive query containment problem in the presence of access limitations is not known. In particular, the best available lower bound is the same as for conjunctive query containment without access limitations. Last, query containment under access limitations together with integrity constraints in the schema is not yet known to be decidable, although different formal tools known from the literature might be combined to attack this problem.

Acknowledgments. Andrea Calì was supported by the EPSRC project "Schema Mappings and Automated Services for Data Integration and Exchange" (EP/E010865/1). Davide Martinenghi acknowledges support from the "Search Computing" (SeCo) project, funded by the ERC under the 2008 Call for "IDEAS Advanced Grants".

6. REFERENCES

[1] Michael J. Cafarella, Alon Y. Halevy, and Nodira Khoussainova. Data integration for the relational web. *PVLDB*, 2(1):1090–1101, 2009.

[2] Andrea Calì, Diego Calvanese, and Davide Martinenghi. Dynamic query optimization under access limitations and dependencies. *Journal of Universal Computer Science*, 15(21):33–62, 2009.

[3] Andrea Calì and Davide Martinenghi. Conjunctive query containment under access limitations. In *Proc. of ER*, pages 326–340, 2008.

[4] Andrea Calì and Davide Martinenghi. Querying data under access limitations. In *Proc. of ICDE*, pages 50–59, 2008.

[5] Piotr Dembinski and Jan Maluszynski. And-parallelism with intelligent backtracking for annotated logic programs. In *Proc. of Symp. on Logic Programming*, pages 29–38, 1985.

[6] Alin Deutsch, Bertram Ludäscher, and Alan Nash. Rewriting queries using views with access patterns under integrity constraints. *Theoretical Computer Science*, 371(3):200–226, 2007.

[7] Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. In *Proc. of IJCAI*, pages 778–784, 1997.

[8] Daniela Florescu, Alon Y. Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *Proc of SIGMOD*, pages 311–322, 1999.

[9] Georg Gottlob, Christoph Koch, Robert Baumgartner, Marcus Herzog, and Sergio Flesca. The Lixto data extraction project – Back and forth between theory and practice. In *Proc. of PODS*, pages 1–12, 2004.

[10] Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[11] 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.

[12] Hai He, Weiyi Meng, Clement T. Yu, and Zonghuan Wu. Wise-integrator: A system for extracting and integrating complex web search interfaces of the deep web. In *Proc. of VLDB*, pages 1314–1317, 2005.

[13] Chen Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB Journal*, 12(3):211–227, 2003.

[14] Chen Li and Edward Chang. Answering queries with useful bindings. *ACM TODS*, 26(3):313–343, 2001.

[15] Bertram Ludäscher and Alan Nash. Processing union of conjunctive queries with negation under limited access patterns. In *Proc. of EDBT*, pages 422–440, 2004.

[16] Jayant Madhavan, Loredana Afanasiev, Lyublena Antova, and Alon Y. Halevy. Harnessing the deep web: Present and future. In *Proc. of CIDR*, 2009.

[17] Todd D. Millstein, Alon Y. Halevy, and Marc Friedman. Query containment for data integration systems. *JCSS*, 66(1):20–39, 2003.

[18] Alan Nash and Bertram Ludäscher. Processing first-order queries under limited access patterns. In *Proc. of PODS*, pages 307–318, 2004.

[19] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proc. of VLDB*, pages 129–138, 2001.

[20] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of PODS*, pages 105–112, 1995.

[21] Guizhen Yang, Michael Kifer, and Vinay K. Chaudhri. Efficiently ordering subgoals with access constraints. In *Proc. of PODS*, pages 22–22, 2006.

[22] Ramana Yerneni, Chen Li, Hector Garcia-Molina, and Jeffrey D. Ullman. Computing capabilities of mediators. In *Proc. of SIGMOD*, pages 443–454, 1999.

[1] `http://www.search-computing.com`