Querying Structured Information Sources Over the Web

Sérgio L. S. Mergen and Carlos A. Heuser

Universidade Federal do Rio Grande do Sul, Av. Bento Gonalves, 9500
Porto Alegre - RS - Brasil
[mergen, heuser]@inf.ufrgs.br

Abstract. To provide access to distributed and heterogeneous sources, information integration systems have traditionally relied on the availability of a mediated schema, along with mappings between this schema and the schema of the source schemas. Queries posed to the mediated schema are reformulated in terms of the source schemas. On the Web, where sources are plentiful, autonomous and extremely volatile, a system based on the existence of a pre-defined mediated schema and mapping information presents several drawbacks. Notably, the cost of keeping the mappings up to date as new sources are found or existing sources change can be prohibitively high. In this paper, we propose a new querying mechanism for integrating a large number of sources that requires neither a mediated schema nor source mappings. In the absence of a mediated schema, the user formulates queries based on what she expects to find. These queries are rewritten using a best-effort approach: the rewriting component compares a user query against the source schemas and produces a set of rewritings based on the matches found. We demonstrate the feasibility of this approach by providing a query interface for integrating hundreds of (real) structured Web information sources. We also discuss experimental results which indicate that our query rewriting algorithm can be effective.

1 Introduction

From data produced by Web services and published from online databases to Web tables, the volume of structured data on the Web has grown considerably in the recent past. In contrast to unstructured (textual) documents, the presence of structure enables rich queries to be posed against these data. This creates new opportunities for mining, correlating and integrating information from an unprecedented number of disparate sources.

Existing approaches to data integration, however, are not effective for integrating, on-the-fly, the very large number of information sources that are available on the Web. Consider, for example, information mediators [9]. These systems define a global (integrated) schema, and a query over the global schema is translated into queries over the information sources based on pre-defined mappings between the global and source (local) schemas [20, 10, 14, 8]. Clearly, it would not be feasible to manually create and maintain such a system for thousands (or even hundreds) of sources. Because new sources are constantly added and existing sources modified (both content and structure) keeping track of sources as they change and updating mapping information and global schema can be prohibitively expensive. Besides, it is unlikely that a single integrated schema would be suitable for all users and information needs. Although solutions have been proposed to amortize the cost of integration, either through mass collaboration [15] or by using a peer to peer architecture [18], there is still a need to create and maintain mappings for data to be included in the integration system.

Contributions and Outline. In this paper we propose a new approach to integration designed to deal with the scale and dynamic nature of structured Web sources. Our goal is to provide users the ability to query and integrate a large number of structured sources available on the Web *on the fly.* There are two key distinguishing features in our approach:

- *It does not require a pre-defined global schema:* Instead of posing queries over a global schema, a user formulates queries based on her knowledge of the domain and on what she expects to find. The queries can be refined as the user explores and learns more about the information sources.
- *It automatically derives mappings:* Instead of requiring mappings to be pre-defined, a user query is rewritten into queries over the sources based on correspondences (automatically) identified between attributes in the query and attributes in sources.

Of course this approach cannot give guarantees of coverage (i.e., that all answers are retrieved) or even that the returned answers are 'correct'. Nonetheless, this best-effort integration can be useful for exploratory searches, and to help users better understand a domain and identify relevant sources as a prelude to a more structured (e.g., mediator-based) integration effort. The use of a best-effort approach represents a shift in paradigm for rewriting strategies which requires (1) new algorithms that are able to handle a large number of sources and to derive acceptable answers; (2) new mechanisms for ranking the answers; and (3) taking user feedback into account. In this paper we focus on the first two tasks.

As a proof of concept, we built a system that supports queries over hundreds of (real) structured Web information sources. We focused on data that is structured as relations and whose schema (i.e., attribute names) and contents can be extracted automatically. We describe this implementation and we also discuss preliminary experimental results that indicate the effectiveness of our query rewriting algorithm.

The remainder of this paper is organized as follows. In Section 2, we present a concrete example that illustrates the limitations of existing integration approaches that rely on pre-defined mappings. We also give a brief overview of our solution. Section 3 gives a detailed description of the proposed rewriting mechanism. In Section 4, we present experimental evaluation using hundreds of (real) Web sources. In Section 5, we discuss the related work. We conclude in Section 6, where we outline directions for future work.

Source Relations	
(s_1)	$MV_1(title, genre)$ $MV_2(title, year)$
(s_2)	$MV_3(title, award)$
(s_3)	$MV_4(title, year, award)$
(s_4)	$MV_5(title, genre, year)$
(s_5)	$BK_1(title, publisher, year)$

Fig. 1. Source Tables

2 Motivating Example and Solution Overview

In what follows, we present an example that illustrates the limitations of mapping-based query rewriting strategies. Although we restrict our discussion to global-as-view (GAV) [20] and local-as-view (LAV) [10], the same issues arise for other techniques, such as BAV [14] and GLAV [8].

Consider we have four data sources (s_1, s_2, s_3, s_4) containing information about the movies domain and one source (s_5) containing information about the books domain, as described in Figure 1. Also, each data source may contain more than one table.

In GAV, the tables of the global schema are defined in terms of the tables of the source schemas, or in other words, they are views over the source tables. Suppose that we initially have only the sources s_1, s_2 and s_3 (see Figure 1). A GAV view could be created over these relations combining information about movies, their titles and year of release:

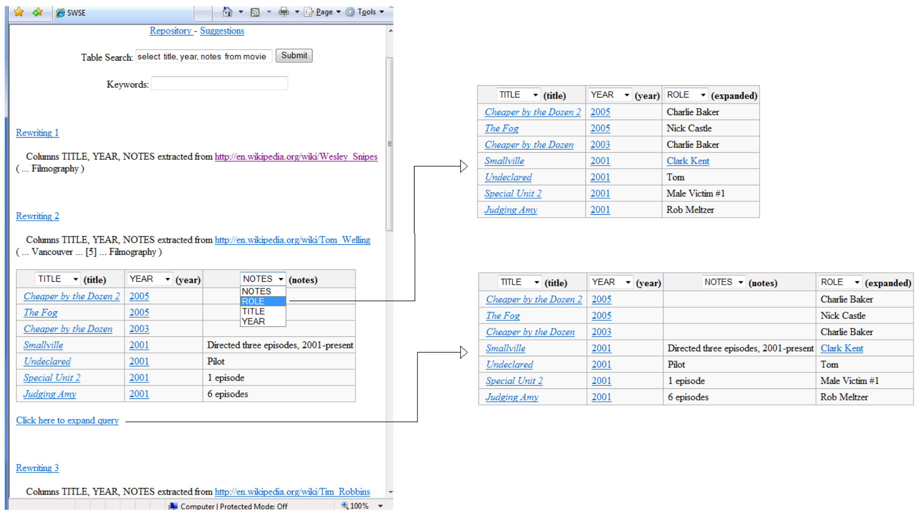
$all\_movies(title, year) : -MV_2(title, year)$
$all\_movies(title, year) : -MV_4(title, year, award)$

The benefits of this approach is that the rewriter algorithms are just a matter of unfolding [20]. However, adding new sources may require changes to the definition of the global schema. If a new source about movies is added (e.g., s_4), the definition of *all_movies* needs to be modified to include the new source:

$all\_movies(title, year) : -MV_5(title, genre, year)$

In contrast to GAV, in LAV the source tables are defined in terms of the tables of the global schema, or in other words, they are views over the global schema. LAV favors the extensibility of the system: adding a new source requires only the addition of a rule to define the source, the global schema needs no change. On the other hand, query rewriting is not as easy as in GAV, since it involves the process of discovering which sources have relevant data to the query [10].

For both LAV and GAV, changes to the global schema require changes to the mappings. In the example above, if users are interested in obtaining information about awards received by movies, or new sources are added that contain infor-



select title, year, notes from movie

Rewriting 1

Columns TITLE, YEAR, NOTES extracted from http://en.wikipedia.org/wiki/Wesley_Snipes (... Filmography)

Rewriting 2

Columns TITLE, YEAR, NOTES extracted from http://en.wikipedia.org/wiki/Tom_Welling (... Vancouver ... [5] ... Filmography)

TITLE (title)	YEAR (year)	NOTES (notes)
Cheaper by the Dozen 2	2005	
The Fog	2005	
Cheaper by the Dozen	2003	
Smallville	2001	Directed three episodes, 2001-present
Undeclared	2001	Pilot
Special Unit 2	2001	1 episode
Judging Amy	2001	6 episodes

NOTES
ROLE
TITLE
YEAR

Click here to expand query

Rewriting 3

Columns TITLE, YEAR, NOTES extracted from http://en.wikipedia.org/wiki/Tim_Robbins

TITLE (title)	YEAR (year)	ROLE (expanded)
Cheaper by the Dozen 2	2005	Charlie Baker
The Fog	2005	Nick Castle
Cheaper by the Dozen	2003	Charlie Baker
Smallville	2001	Clark Kent
Undeclared	2001	Tom
Special Unit 2	2001	Male Victim #1
Judging Amy	2001	Rob Meltzer

TITLE (title)	YEAR (year)	NOTES (notes)	ROLE (expanded)
Cheaper by the Dozen 2	2005		Charlie Baker
The Fog	2005		Nick Castle
Cheaper by the Dozen	2003		Charlie Baker
Smallville	2001	Directed three episodes, 2001-present	Clark Kent
Undeclared	2001	Pilot	Tom
Special Unit 2	2001	1 episode	Male Victim #1
Judging Amy	2001	6 episodes	Rob Meltzer

Fig. 2. Query interface of the Structured Web Search Engine. Users pose SQL queries and the system retrieves HTML tables that *best* match the queries.

mation about awarded actors, GAV rules must be modified or added to include this information.

Creating mappings and maintaining them as sources evolve and the global schema changes can be expensive and time consuming. This effort is required and justified for applications that are mostly static, cater to well-defined information needs, and integrate a relatively small number of sources. However, for exploring and integrating information sources on the Web, expecting the existence of a pre-defined global schema is not practical. To define a global schema, one first needs to know about which sources are available. And on the Web, there are too many of them. Besides, a single global schema is unlikely to be sufficient to fulfill the information needs of all users. Even if it is possible to model a very large global schema that contains constructs for comprehensive set of concepts, the effort spend on defining the mappings to the sources would become unfeasible, not to mention that Web sources are too volatile, which would require the mappings to be constantly updated.

Our Approach. Instead of requiring a global schema and mappings between this and the source schemas, we propose a search engine that allows users to explore Web information, discovering useful information sources and their relationships.

A screendump of the first prototype of our *Structured Web Search Engine* (SWSE) is shown in Figure 2. Similar to a traditional search engine, such as Google and Yahoo!, we use a Web crawler to locate structured information on the Web. For this prototype, we collected Web pages that contain HTML tables about movies (a more detailed description of how the information was collected is given in Section 4).

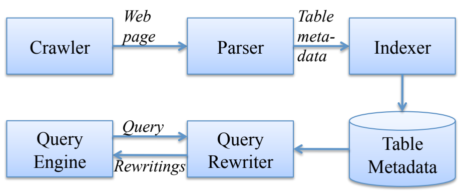


Fig. 3. High-level architecture of the Structured-Web Search Engine

Using the query interface, users can formulate structured queries on the fly. As shown in the figure, the query `select title, year, notes from movies` returns a list of rewritings that contain possible answers to the query. When the user clicks on the result, our search engine accesses the actual Web Pages and extracts the desired information from its HTML tables.[1] She can then manipulate the results by, for example, displaying all the attributes in the table, using the query expansion featured, as illustrated in Figure 2. This would help the user discover additional information that is available and related to her query. Additionally, the drop down list depicted in the Figure enables the user to refine the query by switching any of the returned columns by any other column that is available in the same table. Since the data is retrieved from the source, the result is always fresh.[2]

The search engine maintains an index which contains metadata for the tables (e.g., the attribute names). This information is used to derive the rewritings. The system can be set to periodically crawl the Web to update the index as well as to discover new information sources. The information will then be immediately available to the users.

The architecture of our search engine consists of five components, shown in Figure 3. The *Crawler* is responsible for finding relevant structured Web sources. Because structured information is sparsely distributed on the Web, we use a focused crawler for this task [2, 3]. The *Parser* is responsible for extracting tabular medatada (and records) from the retrieved Web pages. For our prototype, we implemented a parser that automatically extracts Web tables and metadata associated with them, e.g., the attribute names. The *Indexer* stores the extracted metadata into a global repository. When a user poses a query to the query engine, the *Rewriter* uses the information in the metadata repository to derive rewritings for the user query. The *Query Engine* then executes the rewritings and displays the results to the user. In this paper, we focus on the Rewriter, which is described in the following section.

[1] As we describe later, the name of the table `movies` is not used in the rewriting, only the attribute names are considered.
[2] For efficiency, results could also be cached.

3 Deriving Query Rewritings

Before we discuss our approach to query rewriting, in what follows we introduce the nomenclature, some background on query rewriting techniques and the assumptions underlying the query rewriter. Note that we assume that the structured sources supported by SWSE can be represented in the relational model. In addition, users express requests in the form of conjunctive queries.

Consider the conjunctive query Q_1:

(Q_1) q(t,g,d,a):- movie(t,g,d,y), award(t,a), y = 2005

where t = title, g = genre, d = director, y = year and a = award. This query is asking for movies that received an award. We will use this query as the running example for the remainder of the paper.

The *subgoals* in a query refer to tables (e.g., movie(t,g,d,y)) and the *variables* in a subgoal to the table attributes. For the data sources, we use the terms *source tables* (or tables) and *source columns* (or columns) to refer to the source relations and their attributes—in our prototype, HTML tables and their columns.

In a query, variables that appear in multiple subgoals are called *shared variables* (these correspond to join conditions); and variables that appear in a selection condition are called *selection variables.* In the query above, t is a shared variable and y is a selection variable.

The query rewriting process consists of two steps. First, the *Matcher* (Section 3.1) identifies matchings between variables mentioned in the query and the source columns in the metadata repository. Then, the *Combiner* uses the matching information to produce a set of rewritings (Section 3.2).

Background. The concepts of query containment and equivalence enable us to compare between a user query and its rewritings [5]. We say that a query Q_1 is contained in the Q_2, denoted by $Q_1 \subseteq Q_2$, if the answers to Q_1 are a subset of the answers to Q_2 for any database instance. The query Q_1 is equivalent to Q_2, denoted as $Q_1 \equiv Q_2$, if and only if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$. In such cases, the answers of Q_1 are equal the answers of Q_2.

Considering that the tables we are interested in reaching are distributed over the Web and belong to autonomous sites, our work falls into the *Open World Assumption* (OWA) category, where the extension of each source is incomplete with respect to the whole set of source [1]. Under this assumption, a rewriting can never be equivalent, even if the containment conditions stated above are satisfied. Thus, there may be many possible ways to answer a user query, each of them using a different set of sources and yielding different answers as the result. We call each individual way to answer a query a *local rewriting.* Under the OWA, the goal is to find a maximally contained rewriting, that is, one that returns the maximal set of answers from the sources [7]. For a maximally contained rewriting, it is necessary to perform a union over all computed local rewritings.

Even though our work fits into the open world environment, instead of building a maximally contained rewriting, we present each local rewriting to the user. The reasons behind this choice are threefold:

1. Having each local rewriting listed separately makes it easier to relate the resulting tuples to their respective sources. Knowing the location from where each tuple is retrieved can help the users identify which tables are better sources of information.

2. Since our approach relies on a best-effort matching mechanism, it is not possible to ensure that all rewritings will lead to relevant answers. Having the rewritings presented to the user individually, it is possible to associate answers that the user understood as irrelevant to a specific rewriting and ignore all answers that come from this rewriting.

3. Users can provide feedback on the individual rewritings. For example, a user can tell the system that the answer for a specific column in a rewriting are incorrect. Based on this information, the system could run the same rewriting, but switching the incorrect column with the correct one.

User Query Subgoals	Source table
movie(t,g,d,y)	$MV_5(t, g, y)$
	$MV_2(t, y)$
	$MV_4(t, y, -)$
	$BK_1(t, -, y)$
award(t, a)	$MV_3(t, a)$
	$MV_4(t, -, a)$

(a) Table matches for query Q_1

Variable	Column	Score
title (t)	title	1.0
genre (g)	genre	1.0
year (y)	year	1.0
award (y)	award	1.0
director (d)	-	-

(b) Column matches for Q_1

Fig. 4. Matching examples

3.1 Computing Table Matches

The *Matcher* is responsible for computing *table matches*, i.e., a match between a user query subgoal and a source table. Within a table match there is a finer grained level of matching, which we term *column match.* A column match is a match between a variable of the user query and a column of the source table. Figure 4(a) brings table matches for the subgoals of Q_1. For example, the source table `MV`$_5$`(title, genre, year)` (Figure 1) is a possible table match for the subgoal `movie(title,genre,director, year)`.

The variables of a query can be either required or optional. The difference is that a required variable always need to be matched to a source column. We consider shared and selection variables as required—all other variables are optional. Other alternatives are possible, for example, the user could define which variables are optional and which are required.

Next we discuss some details about how the table matches are computed. The whole process is divided in two steps: the Column Matching and the Table Matching.

Column Matching. This step involves finding column matches for every variable of the query: we compute the similarity between each variable and a list of the repository columns. A repository column is a column that appears in the definition of at least one table (in our case: `title`, `genre`, `year`, `award` and `publisher`).

The column match for a variable will be the one whose similarity is greater. If more than one column has the highest score, we arbitrarily select one of them. If the incorrect column is selected, the user can change the selection using the answer refinement mechanism mentioned in Section 2.

The name similarity is computed using a normalized string similarity technique that gives a high score for cases where the strings have substrings in common, even if they appear in different positions (i.e. "movieTitle" and "titleOfMovie")[3].

Figure 4(b) shows the column matches for query Q_1. Note that there is no column match for variable `director`. Since this is not a required variable, the matching process can proceed.

Table Matching. In this step we use the column matches for computing the similarity between the subgoals of a query and the source tables. The similarity between a subgoal S and a source table T is computed using Equation 1. Let $|S|$ be the number of variables of S and $\{w_1, w_2, ..., w_n\}$ be the list of matching scores between S and T, for every variable of S.

$$sim(S,T) = \frac{\sum_{j=1}^{n} w_j}{\sqrt[2]{\sum_{j=1}^{n}(w_j)^2 \times |S|}} \tag{1}$$

The equation returns a normalized score between zero and one. We are not using the length of the table ($|T|$) as part of the normalization factor to prevent the size of the table from affecting the score.

3.2 Combining Table Matches

In this section we describe an algorithm that generates rewritings using the table matches computed by the Matcher. The algorithm, named *M-Bucket*, works similarly to the *Bucket* algorithm[13] and the MiniCon Algorithm[16], used to generate rewritings from LAV mappings.

Generally speaking, the goal of the M-bucket algorithm is to fill buckets with source tables and generate rewritings combining one entry from each bucket. The steps of our algorithm are described below:

[3] The code can be downloaded from http://www.cs.utah.edu/˜juliana/downloads/Carla.java

1. For each subgoal S in the query, create a *m-bucket* B
2. Add an entry in B for every source table from which tuples of S can be possibly retrieved, i.e, the source tables that are matched with S.
3. Rank the entries of each bucket using their table match score.
4. Generate rewritings as conjunctive queries by combining one entry from each *m-bucket*.
5. Bind the selection variables predicates of the user query to its respective variables of the rewritings.

Our algorithm differs from the previous ones in three notable ways:

- **It does not rely on pre-defined mappings.** Since there is no mapping previously defined, the M-Bucket algorithm needs to compute rewritings based on assumptions on how the subgoals of the query and the source tables correspond to each other.

- **It does not have to check for containment.** Containment check only makes sense when there is mapping information. In our case, we not only lack this information, but we expect that the rewritings may bring incorrect answers.

- **It uses a ranking.** As we rely on some inference mechanism to compute the rewritings, and considering there can be many different rewritings for the same query, we use a ranking mechanism so the more relevant rewritings appear first.

The entries of a bucket are ranked according to the similarity score between the subgoal and the source tables, as presented in Section 3.1. Other ranking criteria are possible, including the number of records of the matched tables and the relevance of the Web source (e.g., the pagerank) from where the matched table was extracted.

If we populate the buckets with the tables matches provided by Figure 4(a), the M-Bucket Algorithm would generate eight rewritings for query Q_1. Figure 5 shows such rewritings.

Query: q(t,g,a) :-	movie(t,g,y),	award(t,a),	y=2005
Rewriting 1:	MV_5(t,g, y),	MV_3(t,a),	y=2005
Rewriting 2:	MV_5(t,g, y),	MV_4(t,_, a),	y=2005
Rewriting 3:	MV_2(t,y),	MV_3(t,a),	y=2005
Rewriting 4:	MV_2(t, y),	MV_4(t,_,a),	y=2005
Rewriting 5:	MV_4(t,y, _),	MV_3(t,a),	y=2005
Rewriting 6:	MV_4(t,y, _),	MV_4(t,_, a),	y=2005
Rewriting 7:	BK_1(t,_, y),	MV_3(t,a),	y=2005
Rewriting 8:	BK_1(t,_, y),	MV_4(t,_, a),	y=2005

Fig. 5. Rewritings for the User Query Q_1 Asking for Awarded Movies movies

Note that rewritings 7 and 8 include information about books instead of movies. These rewritings are derived because the source table $\mathtt{BK}_1$ contains columns `title` and `year` which match the query variables. Although it is conceivable that a user may want to integrate information about books and movies (e.g., finding movies and books that have the same title), in some cases, these rewritings are not desirable. Tuning the rewriting component to better match a user's preference is a problem we plan to study in future work.

Removing Duplicate Subgoals There is a possibility that the M-Bucket algorithm generates rewritings with duplicate subgoals, that is, subgoals that refer to the same source table T. In some cases, such rewritings can be folded by removing the duplicate occurrences of T.

Query folding can be applied whenever the duplicate subgoals share variables, and these shared variables appear in the same position in their respective subgoals (i.e., the duplicated tables are self-joined by the same column). In such cases, the duplicate subgoals can be removed. In our running example, rewriting 6 contain such duplicate subgoals. After the folding, the body of rewriting 6 becomes $\mathtt{MV}_4\mathtt{(t,y,a)}$.

4 Experiments

Below, we describe a preliminary experimental evaluation we carried out to assess the quality of the derived rewritings.

The prototype. Our prototype is available to use at `http://cumbuco.cs.utah.edu:8080/servlets-examples/servlet/Mesa`. The prototype is currently indexing tables on the movie domain extracted from wikipedia. The tables were collected using the ACHE focused crawler [2].

ACHE was configured to retrieve pages that have at least one HTML table whose header has at least 3 columns and where either `title`, `film` or `movie` appear in the header. The purpose of this condition was twofold: i) detect tables that are used for data tabulation purposes instead of formatting purposes and ii) discover relevant sources of data for the movie domain.

A Wikipedia page that brings a list of the 100 best American movies [4] was used as the seed for the crawl. We stopped crawling after 400 web pages were retrieved. From this collection of pages, we extracted 993 HTML tables that satisfy our condition. We manually checked a sample of 10% of the collection and we verified that 98% of the extracted tables are indeed related to movies.

Currently the prototype accepts select-project-join queries in SQL. Wildcards and the `LIKE` operator are not implemented. As a next step we intend to make the query interface more friendly by providing a simpler query language.

User study. We have asked two students to formulate SQL queries asking for information about the movie domain. We then evaluated the top-k precision

[4] The list was created by the American Film Institute `http://en.wikipedia.org/wiki/100_Years...100_Movies`

top-k	Query							
	1	2	3	4	5	6	7	8
1	100	100	100	100	75	75	66	66
5	100	100	100	90	75	75	66	53
10	100	100	95	75	75	70	66	43
20	100	100	85	62	75	60	66	38

Fig. 6. Precision measured for top-k rewritings

for every query, having k=1,5,10,20. Given a rewriting, precision was measured based on the number of columns that were correctly retrieved. For instance, if the query is asking for five columns, and the first rewriting brings four correctly identified columns, precision in the top-1 is 80%. If the second rewriting brings two correct columns, precision in the top-2 falls to 60% (not 40%, since we take into account the previous rewritings).

Each user formulated four queries. The complete list of queries is described below. As we can see, the queries vary by the number of variables and subgoals.

Query 1: select t1.title,t2.year, t1.actors,t1.genre from t1, t2
where t1.title=t2.title
Query 2: select title, year from movie
Query 3: select title, director, genre, year from movie
Query 4: select title,year, actors,genre from movies where genre='drama'
Query 5: select t1.title,t2.director, t1.genre,t1.year from t1, t2
where t1.title=t2.title
Query 6: select name, genre, year, direction from movie
Query 7: select title, year, rate from t
Query 8: select name, movie, role from movies

Figure 6 brings the precision achieved for each one of the queries. Note that for some queries, precision remains on 100% even when k = 20. In the worst case, precision fall to 38% when k = 20. One of the reason for this decrease is that the system is reaching a small subset of the Web, and most of the indexed tables cannot answer for a specific information need. For instance, no source table contains the column `name`, which explains the lower precision in queries 6 and 8.

In the case of queries 3 and 4, we could get a better coverage by joining complementary source tables. However, our current implementation only performs joins between different source tables if the user query explicitly specify the join in the query.

In some other cases, it would be possible to increase precision by using a better matching technique. For instance, every source table provides either `title`, `movie` or `film` for representing the title of a movie. If we use this information during the matching, the quality of the rewritings would be improved, specially for query 8.

Precision could also be improved if additional information were collected from the Web pages other than the tables. Our Table Parser module only extracts information that appear inside HTML tables. For instance: in our collection, there are no tables that contain both the name of an artist and its role in different movies. However, we have noticed that the column `role` usually appear in Web Pages of a particular artist. Thus, even though the name of the artist is not directly represented in the table, it could be found in the surroundings, or sometimes even in the URL itself.

We have also tested the system against some queries of our own, and the preliminary results showed promising. In particular, the ability to join data from different source tables allowed us to discover some interesting information. For example, by issuing the query `select d.director from d,c where d.director=c.cast`, it was possible to discover that `Burt Reynolds` (amongst others) has not only worked acting, but also directing movies.

5 Related Work

Current approaches to data integration rely on pre-defined mappings between a global schema and the underlying information sources [20, 10, 14, 8]. This architecture, however, is not suitable for integration tasks on the Web, where there is a very large number of information sources and these sources are highly volatile. In an attempt to improve scalability, recent approaches have been proposed to amortize the cost of integration, such as for example, peer-to-peer systems [18] and community-based information integration systems [15]. In the former, users provide mappings between peers, whereas in the latter, users collaborate in the creation of mappings between the sources and the global schema. In all of these approaches, mappings are used to support query rewriting, i.e., the reformulation of a query posed against the global schema into queries that conform to the local schemas [4, 12].

We take a different approach, and instead of requiring mappings to be defined, based on source metadata, we derive mappings automatically in response to user queries. Similar to the universal relation [19], queries are expressed in terms of attributes the user is interested in, the system then determines how the relevant source relations and how they can be composed to cover the query. Davulcu et al. [6] used an extension of universal relation, the structured universal relation (SUR), as the interface to query dynamic Web content from multiple sources (e.g., content published by Web services and online databases). SUR was developed as the infrastructure to support *webbases* that are designed for well-defined domains (e.g., cars, jobs, houses) by experts in those domains. They assume that an expert will define compatibility constraints among information sources, as well as concept hierarchy which relates the different attributes of the universal relation. Clearly, these assumptions do not hold if queries must be supported over sources that can be added on the fly.

6 Discussion and Future Work

In this paper we present a new framework that supports ad-hoc queries over structured sources available on the Web without requiring pre-defined schemas or mappings. An important benefit from not having pre-defined mappings is that the cost of maintenance is reduced: new sources can be added to the system and queried without the overhead of creating new mappings (or updating a global schema). On the flip side, the framework cannot provide guarantees with respect to recall and precision. Instead, it uses a best effort approach to match user queries against the information in the sources.

This approach is not a substitute for the traditional integration approaches, which are needed for application that require precise answers. However, it provides the means whereby users can more easily explore structured information on the Web, learn about different sources, and how they can be connected. And these are important tasks and can help in the construction of (more rigid) integration systems.

Many new challenges arise when one considers mapping-free integration strategies. In this paper, we take a first step in exploring this direction. Even though we have made progress towards creating a usable querying system, there are many open problems and ample room for improvements.

Our approach to determine connections among information sources is very simple and based solely on the string similarity between the names of subgoal variables and the column names of sources (Section 3.1). This may lead to false positives (in the case of homonyms) or false negatives (in the case of synonyms). As a result, rewritings can be derived that do not contain any answers or that contain answers that do not make sense. Although users experienced with search engines are already used to ignoring irrelevant answers and refining queries, we believe that new mechanisms are needed to better guide them to formulate queries as well as adapt to their needs, for example, by taking into account user feedback on the quality of the rewritings. Besides leveraging user feedback to tune the query rewriting process, we also intend to investigate more sophisticated approaches to matching. For example, applying techniques for schema matching [11, 17], we could use both the source metadata and contents to determine the compatibility of the sources and prune rewritings that involve incompatible sources.

References

1. Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *ACM Ssymposium on Principles of database systems*, pages 254–263, 1998.
2. Luciano Barbosa and Juliana Freire. An adaptive crawler for locating hidden-web entry points. In *WWW*, pages 441–450, 2007.
3. Soumen Chakrabarti, Kunal Punera, and Mallela Subramanyam. Accelerated focused crawling through online relevance feedback. In *WWW*, pages 148–159, 2002.
4. Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.
5. Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000.

6. Hasan Davulcu, Juliana Freire, Michael Kifer, and I. V. Ramakrishnan. A layered architecture for querying dynamic web content. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 491–502, New York, NY, USA, 1999. ACM Press.
7. Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *ACM Symposium on Principles of database systems*, pages 109–116, 1997.
8. Marc Friedman, Alon Y. Levy, and Todd D. Millstein. Navigational plans for data integration. In *AAAI/IAAI*, pages 67–73, 1999.
9. Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey Ullman, Vasilis Vassalos, and Jennifer Widom. The tsimmis approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.
10. Alon Y. Halevy. Theory of answering queries using views. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(4):40–47, 2000.
11. Bin He and Kevin Chen-Chuan Chang. Statistical Schema Matching across Web Query Interfaces. In *ACM SIGMOD*, pages 217–228, 2003.
12. T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In C. Knoblock and A. Levy, editors, *Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, Stanford, California, 1995.
13. Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the Twenty-second International Conference on Very Large Databases*, pages 251–262, 1996.
14. P. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules, 2003.
15. Robert McCann, AnHai Doan, Vanitha Varadaran, Alexander Kramnik, and ChengXiang Zhai. Building data integration systems: A mass collaboration approach. In *WebDB*, pages 25–30, 2003.
16. Rachel Pottinger and Alon Y. Levy. A scalable algorithm for answering queries using views. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 484–495, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
17. Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal: The International Journal on Very Large Data Bases*, 10(4):334–350, 2001.
18. Igor Tatarinov, Zachary Ives, Jayant Madhavan, Alon Halevy, Dan Suciu, Nilesh Dalvi, Xin (Luna) Dong, Yana Kadiyska, Gerome Miklau, and Peter Mork. The piazza peer data management project. *SIGMOD Record*, 32(3):47–52, 2003.
19. Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II.* Computer Science Press, 1989.
20. Jeffrey D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.