

---

## Querying structured information sources on the Web

---

### Sergio Mergen\*

Instituto de Informática,  
Universidade Federal do Rio Grande do SUL (UFRGS),  
Porto Alegre 91501-970, Brazil  
E-mail: mergen@inf.ufrgs.br  
\*Corresponding author

### Juliana Freire

School of Computing,  
University of Utah,  
Salt Lake City 84112, USA  
E-mail: juliana@cs.utah.edu

### Carlos A. Heuser

Instituto de Informática,  
Universidade Federal do Rio Grande do SUL (UFRGS),  
Porto Alegre 91501-970, Brazil  
E-mail: heuser@inf.ufrgs.br

**Abstract:** To provide access to heterogeneous data distributed over the Web, we propose a solution that merges the expressiveness of information integration systems with the flexibility found in dataspaces-aware search engines. Our approach requires neither a mediated schema nor source mappings. In the absence of a mediated schema, the user formulates structured queries based on what she expects to find. 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 is both effective and scalable.

**Keywords:** search engines; dataspaces; information integration; query rewriting.

**Reference** to this paper should be made as follows: Mergen, S., Freire, J. and Heuser, C.A. (2010) 'Querying structured information sources on the Web', *Int. J. Metadata, Semantics and Ontologies*, Vol. 5, No. 3, pp.208–221.

**Biographical notes:** Sergio Mergen is a PhD student of the Instituto de Informática at the Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil. He achieved his MSc title at the same institute. He spent part of his PhD at the School of Computing (University of Utah) as part of the Web research group. His areas of interest include search engines and data integration applications. He currently works at Idealogic Software, doing research about ETL and BPML technologies.

Juliana Freire is an Associate Professor at the School of Computing at the University of Utah. Before, she was member of technical staff at the Database Systems Research Department at Bell Laboratories (Lucent Technologies) and an Assistant Professor at OGI/OHSU. An important theme in her work is the development of data management technology to address new problems introduced by emerging applications, including the Web and scientific applications. She is an active member of the database and Web research communities, having co-authored over 90 technical papers and holding 4 US patents. She is a recipient of an NSF CAREER and an IBM Faculty award. Her research has been funded by grants from the National Science Foundation, Department of Energy, National Institutes of Health, the University of Utah, Microsoft Research, Yahoo! and IBM.

Carlos A. Heuser is a Full Professor of the Instituto de Informática at the Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil. He received his Dr. in Computer Science from the University of Bonn, Germany, in 1986. He also holds an Electrical Engineer title from the Universidade Federal do Rio Grande do Sul in Brazil (1973) and a MSc in Computer Science from the same University (1976). He has published a book on database design (in Portuguese) and more than 90 conference and journal papers in the

areas of databases and software engineering. He has chaired latin American conferences and workshops, and has actively served in the program committees of international conferences and has been reviewer for journal papers. He is a member of the Brazilian Computer Society (SBC).

## 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. Additionally, there is a lot of distributed resources that speak about the same universe of discourse, and often present overlapping information about a specific object. *Dataspaces* have been proposed as a new paradigm for managing these structured Web sources (Franklin et al., 2005; Halevy et al., 2006). A dataspace can be seen as a set of structured and autonomous information sources that address a specific need and contain (implicit/explicit) connections among them.

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 (Agrawal et al., 2008).

Existing approaches to data integration, however, are very limited when it comes to dataspace. Consider, for example, information mediators (Garcia-Molina et al., 1997). 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 (Ullman, 2000; Halevy, 2000; McBrien and Poulouvasilis, 2003; Friedman et al., 1999).

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 amortise the cost of integration, either through mass collaboration (McCann et al., 2003) or by using a peer to peer architecture (Tatarinov et al., 2003), there is still a need to create and maintain mappings for data to be included in the integration system.

In an orthogonal direction, specialised search engines are also considered for the dataspace environment. As a regular search engine, the contents of the sources are stored into indexes (i.e., no mapping information is required). However, when indexing dataspace, not only the data, but the structure of the sources needs to be

preserved, such as metadata and connections between different sources. In this context, queries are expressed as keywords, and the keywords are identified in the indexes in order to obtain the data sources that can provide the answers. As a drawback, the querying expressiveness of keywords is limited, and cannot fully explore the structure of the sources. For example, it is hard to express a keyword query with a selection predicate (e.g., find movies released before 1950).

*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*, merging the query expressiveness of data integration systems to the flexibility of dataspace integration systems.

From this merging, three important features in our approach arise:

- *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.
- *Queries are structured.* Queries are composed by select-project-join predicates, which enables the user to pose meaningful queries, as opposed to the keyword based interfaces from traditional search engines.

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
- 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 that our query rewriting algorithm is both effective and scalable.

The remainder of this paper is organised 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 presents the system architecture in addition to some background information. Section 4 gives a detailed description of the proposed rewriting mechanism. In Section 5, we present experimental evaluation using hundreds of (real) Web sources. We evaluate both the scalability and result quality of different query rewriting algorithms. In Section 6, we discuss the related work. We conclude in Section 7, where we outline directions for future work.

## 2 Motivating example and solution overview

In order to access structured information meaningfully, it is extremely desirable that structured queries are supported. Currently, only data integration systems are able to express rich structured queries. However, these kind of system presents limitations regarding the mapping strategy underneath them. In what follows, we describe an example that illustrates these limitations. Although we restrict our discussion to Global-As-View (GAV) (Ullman, 2000) and Local-As-View (LAV) (Halevy, 2000), the same issues arise for other techniques, such as BAV (McBrien and Poulouvasilis, 2003) and GLAV (Friedman et al., 1999).

Consider we have five data sources ( $s_1, s_2, s_3, s_4, s_5$ ), as described in Figure 1, where each source is composed by a set of relations. Data sources from  $s_1$  to  $s_4$  contain information about movies, where  $s_1$  brings information about the movie *Casablanca*,  $s_2$  brings a list of sci-fi movies,  $s_3$  brings a list of famous movies ( $mv_4$ ) and awarded movies ( $mv_3$ ) and  $s_4$  brings movies from the forties. Oppositely, data source  $s_5$  does not list movies, but information about books from *Isaac Asimov*.

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 relations  $mv_1$  and  $mv_2$  (see Figure 1). A GAV view could be created over these relations combining information about movies, their titles and year of release:

$$\begin{aligned} all\_movies(title, year) &: -mv_1(title, year) \\ all\_movies(title, year) &: -mv_2(title, year). \end{aligned}$$

**Figure 1** Data source

$(s_1)mv_1(title, yr, award)$	={"Casablanca", "1942", "Best Song"}
$(s_2)mv_2(title, year)$	={"I, Robot", "2004"}, {"Minority Report", "2002"}
$(s_3)mv_3(title, award\_won)$	={"Titanic", "Best Director"}, {"I, Robot", "Best Screenplay"}
$(s_3)mv_4(title, genre, year)$	={"Titanic", "romance", "1997"}, {"Casablanca", "Drama", "1942"}
$(s_4)mv_5(title, genre)$	={"Citizen Kane", "Mystery"}, {"Red River", "Western"}
$(s_5)bk_1(title, pub, year)$	={"Foundation", "Bantam", "2004"}, {"I, Robot", "Bantam", "1991"}

The benefit of this approach is that the rewriter algorithms are just a matter of unfolding (Ullman, 2000). However, adding new sources may require changes to the definition of the global schema. If a new relation about movies is added (e.g.,  $mv_4$ ), the definition of  $all\_movies$  needs to be modified to include the new source:

$$all\_movies(title, year) : -mv_4(title, year).$$

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 favours 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 (Halevy, 2000).

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

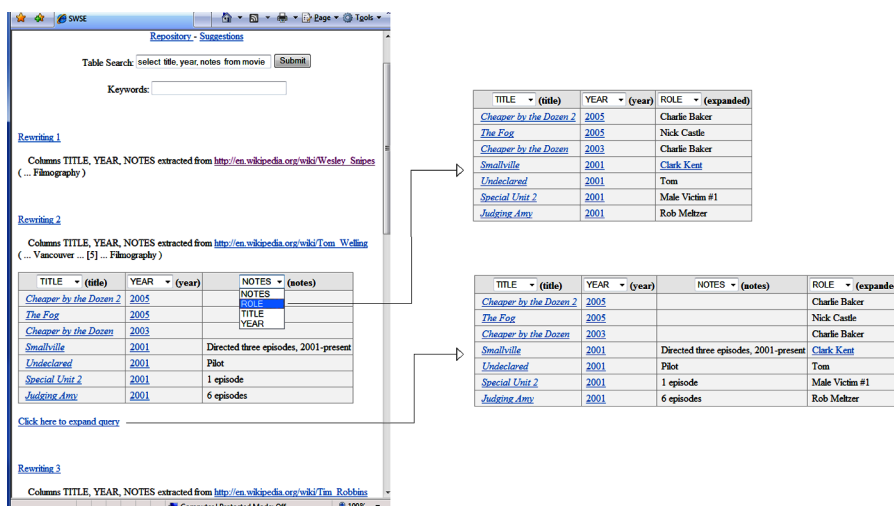
Creating mappings and maintaining them as sources evolve and the global schema changes can be expensive and time consuming. This effort is required and justified for applications that are mostly static, cater to well-defined information needs, and integrate a relatively small number of sources.

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 new query interface that allows users to

**Figure 2** Query interface of the Structured Web Search Engine. Users pose SQL queries and the system retrieves HTML tables that *best* match the queries (see online version for colours)



**Figure 3** A rewriting returned for the query `select actor, award from movie` and its corresponding table is shown on the left. An excerpt of the Wikipedia page from which this table was extracted is displayed on the right



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 5).

Using the query interface, users can formulate structured queries on the fly (in SQL). 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.<sup>1</sup>

She can then manipulate the results by, for example, displaying all the attributes in the table, using the query expansion feature, 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.

Now, if the user wants to explore information about actors and awards instead of movie and titles, she can issue a query like `select actor, award from movie`.

The first rewriting returned by this query is shown in Figure 3, together with an excerpt of the Web page from which the table in the result was extracted. Since the data is retrieved from the source, the result is always fresh.<sup>2</sup>

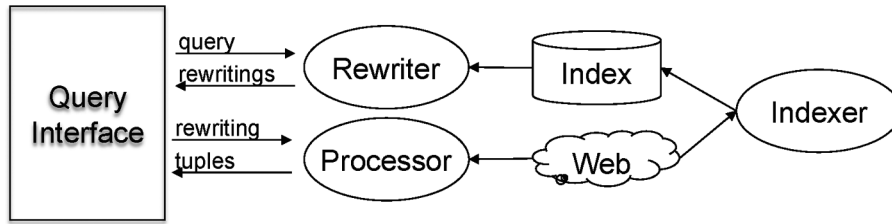
The search engine maintains an index which contains data/metadata for the tables, such as the attribute names and values from the tuples. This information is used to derive the rewritings. The system can be set to periodically crawl the Web to update the index as well as to discover new information sources. The information will then be immediately available to the users.

### 3 System architecture and background

The components of the SWSE are illustrated in Figure 4.

The *Indexer* stores data/metadata from the sources into a global repository. This component is composed by two sub-components: the *Crawler* and the *Parser*. 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 (Barbosa and Freire, 2007; Chakrabarti et al., 2002). The *Parser* is responsible for extracting tabular metadata (and records) from the Web pages retrieved by the *Crawler*. For our prototype, we implemented a parser that automatically extracts Web tables and metadata associated with them, e.g., the attribute names.

Figure 4 High-level architecture of the Structured-Web Search Engine



Given a structured query, the *Rewriter* must find all possible ways to answer this query using relations found on the Web. We call each possible answer a rewriting (Halevy, 2000). More specifically, a rewriting is a conjunctive query (where the subgoals refer to relations). The rewriter access the index in order to find the rewritings. Each rewriting can then be individually processed by the *Query Processor*, and its resulting tuples returned to the user. In this paper, we focus on the *Rewriter*, which is described in detail throughout the paper.

*Data model:* We treat sources as relations. Relations provide a natural model for a significant number of structured Web sources, in particular, HTML Tables. In fact, a recent study reports that there are over 144 million relations published as HTML tables in Google’s index (Cafarella et al., 2008).

By treating sources as relations, we are able to investigate optimisation techniques designed for relational databases. Furthermore, the relational algebra is able to express most of the common requests made by ordinary users.

*Query language:* In our approach, users express requests in the form of conjunctive queries (Although in our implementation rewritings are expressed as select-project-join SQL, for clarity, here we show them using Datalogue). For instance, consider the conjunctive query  $Q_1$ :

$$(Q_1) \text{ q}(t,g,d,a):- \text{ movie}(t,g,d,y), \text{ award}(t,a), \\ y > 1900, y < 2000$$

where  $t = \text{title}$ ,  $g = \text{genre}$ ,  $d = \text{director}$ ,  $y = \text{year}$  and  $a = \text{awards}$ . This query is asking for movies in the 20th century that received an award.

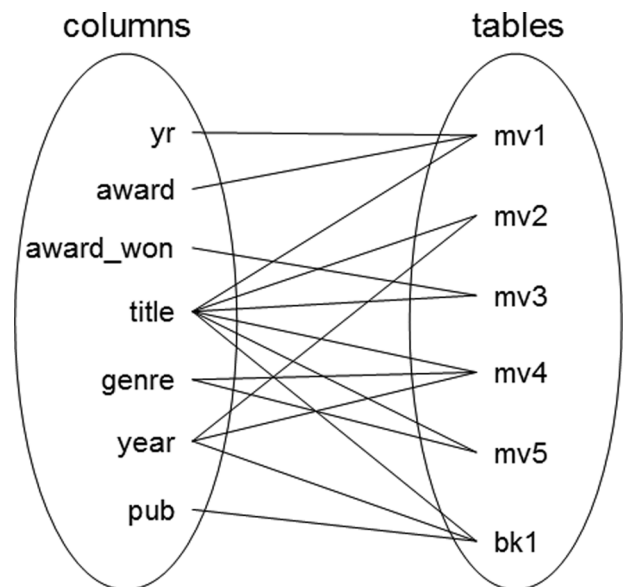
The *subgoals* in a query refer to tables (e.g.,  $\text{movie}(t,g,d,y)$ ) and the *variables* in a subgoal to the table attributes. For the data sources, we use the terms *source tables* (or tables) and *source columns* (or columns) to refer to the source relations and their attributes – in our prototype, HTML tables and their columns. Note that we assume that the structured sources supported by SWSE can be represented in the relational model.

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 selection and join predicates allow powerful queries to be asked. With the selection operator, the search can focus in the most relevant information (e.g., using predicates like “year between 1900 and 2000”). With the join operator, we have the power to combine overlapping information from multiple sources. For instance, relations  $mv_1$  and  $mv_4$  contain complementary information about the movie ‘Casablanca’.

*Index:* The indexed metadata is available through the structure described in Figure 5. The index is a bipartite graph with undirected edges connecting tables with columns. The undirected edges makes it possible to navigate through tables that share attributes in common. Index navigation starts either at the column level or at the table level (a sorted list of columns/tables is kept separately for fast access). Our rewriting mechanism uses the index in order to speed up the identification of the relevant sources of information for a given query.

Figure 5 Indexing relations from Figure 1. The index is a bipartite graph connecting tables and columns



*Rewritings:* The concepts of query containment and equivalence enable us to compare between a user query and its rewritings (Chekuri and Rajaraman, 2000). We say that a query  $Q_1$  is contained in  $Q_2$ , 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 (Abiteboul and Duschka, 1998).

In this context, 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 (Duschka and Genesereth, 1997). 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:

- 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.
- 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.
- 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 is incorrect. Based on this information, the system could run the same rewriting, but switching the incorrect column with the correct one.

## 4 Deriving query rewritings

Given a query, such as  $Q_1$ , the query rewriting process consists of two phases. In the first phase (Section 4.1), matches between subgoals of the query and the source tables are identified. In the second phase, the matches are combined in order to produce a set of rewritings (Section 4.2.1).

### 4.1 Computing table matches

A *Table Match* is 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.

Table 1 brings table matches for the subgoals of  $Q_1$ . For example, the source table  $mv_4(\text{title}, \text{genre}, \text{year})$  (Figure 1) is a possible table match for the subgoal  $\text{movie}(\text{title}, \text{genre}, \text{director}, \text{year})$ . The similarity between a subgoal and a source table is given by a score, which is detailed in Section 4.1.2.

**Table 1** Table matches for query  $Q_1$

User query subgoals	Source table	Score
movie( $t, g, d, y$ )	$mv_4(t, g, y)$	0.86
	$mv_2(t, y)$	0.7
	$mv_1(t, y, -)$	0.7
	$bk_1(t, -, y)$	0.7
award( $t, a$ )	$mv_1(t, -, a)$	1.0
	$mv_3(t, a)$	0.99

#### 4.1.1 Match constraints

The match derivation process is ruled by constraints that apply to both columns/table matches. These constraints are presented below:

*Local cardinality constraint:* Column and table matches are constrained to a 1-1 local bidirectional match. That is, within a table match, each column can match at most one variable (and each variable can match at most one column). Additionally, the columns of a table can only match variables of the same subgoal (in this case, the table and the subgoal have a 1-1 local match).

*Global cardinality constraint:* There can be n-n global table matches, where a table or subgoal participates in different local 1-1 matches. For instance, consider the table matches in Table 1. There are two local table matches for subgoal *award*, one involving the source table  $mv_3$  and the other involving the source table  $mv_1$ . Likewise, there are two local table matches for source table  $mv_1$ , one involving subgoal *movie* and the other involving subgoal *award*.

*Required variables:* We distinguish between required and optional variables in that required variables always need to be matched to a column. A match between a table and a subgoal is only accepted if all required variables of the subgoal are matched. The effect of this rule is that the produced rewritings will always cover the required variables, whereas the optional variables may not be covered. In our implementation 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.

#### 4.1.2 Matching process

Table matches are computed from the bottom up. First the column matches are found. Based on this

information, the table matches are defined. Below we describe how these two separate steps are performed.

*Finding column matches.* The column match is computed as the name similarity between each variable and the list of indexed columns.

We use a normalised string similarity function that gives a high score for cases when the strings have substrings in common, even if they appear in different positions (i.e., ‘movieTitle’ and ‘titleOfMovie’).<sup>3</sup>

Table 2 shows the column matches for query  $Q_1$ . In the example, the similarity function was parameterised to consider substrings of length = 1 and to prioritise the occurrence of overlapping. Using this configuration, the string score is the sum of the common characters divided by the length of the smaller string. As a result, the function is able to find the correspondence between the variable `year` and the column `yr`, for example.

**Table 2** Column matches for  $Q_1$

Variable	Column	Score
title (t)	title	1.0
genre (g)	genre	1.0
year (y)	year	1.0
year (y)	yr	1.0
awards (y)	award	1.0
awards (y)	award_won	0.83
director (d)	–	–

A variable can have more than one column match, as long as the match score satisfies a threshold (the threshold is explained in Section 4.1.3). Note that there is no column match for variable `director`. If this was a required variable, the process could be interrupted, since no valid table match could be found.

*Finding table matches:* 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  $\{w_1, w_2, \dots, w_n\}$  be the list of column match scores between  $S$  and  $T$ , for every variable of  $S$ .

$$\text{sim}(S, T) = \frac{\sum_{j=1}^n w_j}{\sqrt{\sum_{j=1}^n (w_j)^2 \times |S|}}. \quad (1)$$

The equation returns a normalised score between zero and one. We do not use the length of the table ( $|T|$ ) as part of the normalisation factor to prevent the size of the table from affecting the score.

Table 1 shows the tables matches between the subgoals of  $Q_1$  and the source tables of Figure 1. Note that the score is directly related to the column matches. The more column matches between a subgoal and a source table, the higher the table match score.

#### 4.1.3 Tuning table and column matches

Below we present some parameters that can be used to tune the matching process.

*Threshold parameter.* The threshold ( $P_{\text{THRESHOLD}}$ ) is a value (ranging from 0 to 1) that is applied to both column and table matching. In the column matching, a variable remains unmatched if it has no column match that satisfies  $P_{\text{THRESHOLD}}$ . If an unmatched variable is required, the process is interrupted. In the table matching, a table  $T$  matches a subgoal  $S$  only if  $\text{sim}(S, T)$  is greater than  $P_{\text{THRESHOLD}}$ .

*Limit parameter.* The limit ( $P_{\text{LIMIT}}$ ) is a positive integer value that can be used as a way to stop computing table matches whenever the number of table matches reaches  $P_{\text{LIMIT}}$ . As the experiments show in Section 5, using a limit improves performance, but usually at the price of a poor quality in the rewritings.

*Relaxed mode and restricted mode.* In the Relaxed Mode, not every variable needs to have a matching column. In the Restricted Mode, every variable needs to have a matching column, even those that are not required. For example, under Restricted Mode, no table match can be found for query  $Q_1$ , since the variable `director` has no valid match.

#### 4.1.4 Table search

Given the column matches (such as the ones in Table 2), there is the need for optimised ways to search for the tables that match the subgoals, considering that the source tables can be accessed through a bipartite graph index.

We have implemented two table searching strategies (Table Scan and Table Intersection), along with some variations. Each strategy presents strengths and weaknesses, both with respect to table selectivity (the number of the tables found) and execution time, as described below.

*Table Scan.* This strategy involves scanning all source tables, one at a time. Once in a source table, the column matches are identified. The execution time of the scan is very sensitive to the number of tables in the repository.

*Pivot Table Scan.* This variation of the Table Scan Strategy narrows the table search by scanning only those tables that contain the column that was matched to a specific variable, that we call pivot variable.

The best candidates for pivot are the required variables, since only tables that provide a column match for the required variables can become a table match.

If a subgoal has more than one required variable, we use as pivot the one whose selectivity is higher (the variable whose matched column can be found in a lesser number of tables). For instance, in query  $Q_1$ , both `title` and `year` are required variables, but we choose `year` as the pivot because its source selectivity is higher.

If the subgoal has no required variables, the pivot becomes the variable that has the best source selectivity, so the search will be restricted to a fewer number of tables.

*Table intersection.* For each matched variable  $v$  in a subgoal, a list is created that contains all tables that match  $v$ . A merge-join is then used to compute the intersection among these lists of tables. A table match is found only for the tables that are part of the intersected list. The intersection is computed pair-wise, and as an optimisation, smaller lists of tables are considered first.

This strategy works as a boolean query with an AND operator, since a table becomes a table match only if it contains column matches for every matched variable. Considering query  $Q_1$ , subgoal  $movie(t, g, d, y)$  can have only one table match ( $MV_5(t, g, y)$ ), since this is the only table that contains column matches for all matched variables ( $t, g, y$ ).

*Incremental table intersection.* This strategy is a variation of Table Intersection that explores the Limit parameter. The idea is to compute the intersection incrementally. In the first iteration, the table intersection is computed only for a subset of tables in the first list of tables. In the subsequent iterations, other subsets are processed. The processing stops when the number of tables in the resulting intersected list is greater than  $P_{LIMIT}$ . As the size of each subset, we use  $P_{LIMIT} \times P_{INCREMENT}$ , where  $P_{INCREMENT}$  is a positive integer value.

#### 4.2 Combining table matches

In this section we describe an algorithm that generates rewritings using the computed table matches. The algorithm is named *M-Bucket*, as a reference to the use of buckets. A bucket is a concept that helps illustrating the way in which the generation of rewritings is conducted. The steps of the algorithm are described below:

- 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 Remove source tables in  $B$  if none of its tuples satisfies the selection predicates for variables of  $S$
- 4 Rank the entries of each bucket using their table match score
- 5 Generate rewritings as conjunctive queries by combining one entry from each *m-bucket*
- 6 Bind the selection variables predicates of the user query to its respective variables of the rewritings.

The general idea of the algorithm is to fill buckets with source tables and generate rewritings combining one entry from each bucket. The concept of bucket was already employed in past work, for the problem of generating rewritings from LAV mappings. The Bucket Algorithm, described in Levy et al. (1996), was proposed as a rewriting algorithm for the Information Manifold

system. Later, a better solution was proposed, which uses a more generalised notion of bucket, called MiniCon (Pottinger and Levy, 2000; Arvelo et al., 2006).

It is important to remark that it is not possible to compare any of these to the algorithm presented in this paper, basically because they were meant for different purposes, and consequently, it is unfeasible to measure them according to the same criteria.

First of all, LAV rewriting strategies known beforehand which source tables are mapped to the query subgoals, whereas the M-Bucket algorithm does not rely on pre-defined mappings. Since there is no mapping previously defined, our algorithm computes rewritings based on assumptions on how the subgoals of the query and the source tables correspond to each other.

Also, in LAV rewriting strategies, the containment check is part of the problematic. In other words, given a rewriting, it may be the case that the answers provided by the rewriting do not match the answers expected by the query. Some algorithms, like the Bucket Algorithm, need to run a containment check to make sure that the answers of a rewriting are valid. However, this 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.

Finally, as we rely on some inference mechanism to compute the rewritings, and considering there can be many different rewritings for the same query, we need a ranking mechanism so the more relevant rewritings appears first. This ranking is, at least, not as important in architectures based on pre-defined mappings.

##### 4.2.1 Results achieved: discussion

If we populate the buckets with the table matches provided by Table 1, the M-Bucket Algorithm would generate six rewritings for query  $Q_1$  (presented in Table 3). Next, we analyse the achieved results and discuss our approach, in the light of its strengths, weaknesses and overall properties.

**Table 3** Rewritings for the user query asking for awarded movies  $Q_1$

$Q_1(t, g, d, a)$ :-	$movie(t, g, y)$ ,	$award(t, a)$ ,	$y > 1900, y < 2000$
Rewriting 1:	$mv_4(t, g, y)$ ,	$mv_1(t, -, a)$ ,	$y > 1900, y < 2000$
Rewriting 2:	$mv_4(t, g, y)$ ,	$mv_3(t, a)$ ,	$y > 1900, y < 2000$
Rewriting 3:	$mv_1(t, y, -)$ ,	$mv_1(t, -, a)$ ,	$y > 1900, y < 2000$
<del>Rewriting 4:</del>	<del><math>mv_1(t, y, -)</math>,</del>	<del><math>mv_3(t, a)</math>,</del>	<del><math>y &gt; 1900, y &lt; 2000</math></del>
<del>Rewriting 5:</del>	<del><math>bk_1(t, -, y)</math>,</del>	<del><math>mv_1(t, -, a)</math>,</del>	<del><math>y &gt; 1900, y &lt; 2000</math></del>
Rewriting 6:	$bk_1(t, -, y)$ ,	$mv_3(t, a)$ ,	$y > 1900, y < 2000$

*Ranking.* As the general rule, the entries of the buckets are ranked based on their table match scores. When the entries are combined in the fifth step of the algorithm, entries with a better ranking position are processed first. Therefore, the order of the rewritings will be directly related to the order of the sorted entries.

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. Another interesting possibility is to define the ranking based on the rewriting as a whole instead of the tables in isolation. For instance, Rewriting 2 could get a better rank than Rewriting 1 if rewritings whose tables came from the same source were deemed more relevant.

*Pruning.* As Table 1 shows, four source tables match the *movie* subgoal, including table  $mv_2$ . However, the entry for this table was removed in the third step of the algorithm, because the range of the selection variable *year* is not satisfied by the table. Consequently, none of the generated rewritings extract data from this source.

This step reduces the generation of rewritings that return empty result sets. However, empty rewritings can still be generated, even if the entries of a bucket satisfy the filter predicates. This situation occurs when the join condition of the rewriting fails. For example, in Rewriting 4, table  $mv_1$  publishes a movie that respect the year selection range (*Casablanca*), but this movie is not published by table  $mv_3$ .

Rewritings with empty result sets are removed before they are ever presented to the user. The strike in Table 3 indicates the rewritings that would be discarded. The removal is based on tuple information stored in a separate index, not detailed in this paper.

*Incompleteness in the rewritings:* It is not possible to find information about *director* in the source tables. Still, the rewritings are able to find answers regarding other columns of the query. These incompleteness does not invalidate the answers, since the missing information is not needed for selecting the tuples (no filter or join condition depends on this column). On the contrary, incomplete rewritings increase the number of answers returned to the user.

Observe that some of the computed rewritings are more incomplete than others. For instance, Rewriting 3 does not cover the optional variable *genre*, while Rewriting 1 does. More complete rewritings are displayed first because their table match score tends to be higher.

*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  $T$ . Rewriting 3 is an example where two subgoals refer to the same table ( $mv_1$ ).

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), which is the case of Rewriting 3. After the folding, the body of this rewriting becomes  $mv_1(\tau, \mathbf{y}, \mathbf{a})$ .

Note that the duplicate subgoals folding does not minimise the query. Query minimisation techniques ensure that the folded query is equivalent to the unfolded one. In this scenario, we do not have sufficient

information to determine equivalence: there is neither global schema nor detailed information about the sources (e.g., the functional dependencies among them).

We can, however, ensure that the folded version of the rewriting is at least contained in the unfolded version. That is, using the folded version, the user will get at least a subset of the answers she is looking for. Additionally, the folding helps to simplify a query and optimise execution time, since the number of joins is reduced.

*Limitations of the matching:* In the core of our approach, we use a string similarity function to find tables that are related to the query subgoals. As expected, this can lead to false positives (in the case of homonyms) or false negatives (in the case of synonyms). For instance, Rewriting 6 reveals information about books instead of movies. This false positive is produced because the source table  $bk_1$  contains columns that match the query variables *title* and *year*.

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, such as ‘*I, Robot*’ in Figure 1), 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.

## 5 Experiments

Below, we describe an experimental evaluation we carried out to assess both the performance of the proposed rewriting techniques as well as of the quality of the derived rewritings.

### 5.1 Scalability of query rewriting

*Data set.* For measuring the time performance of our rewriting approach, we have used data sets containing computer-generated tables. Table 4 shows the columns used to generate the tables along with the probability of each column being part of a table. The underlined columns (*title*, *film*, *movie*) have a disjoint probability of being part of the same table (their individual probabilities sum up to 100%). The distribution of columns presented in Table 4 was derived from a collection of 993 tables that we have collected from the Web (see Section 5.2 for details).

**Table 4** Column frequencies

<i>title</i>	= 75%	<i>notes</i>	= 65%	<i>director</i>	= 58%
<i>genre</i>	= 56%	<i>cast</i>	= 55%	<i>year</i>	= 30%
<i>role</i>	= 17%	<i>film</i>	= 17%	<i>category</i>	= 8%
<i>movie</i>	= 8%	<i>other_notes</i>	= 6%	<i>country</i>	= 1%

*Experimental set up.* We ran the experiments on a Pentium Core 2 Duo with a 2.67GHZ processor and 4GB of RAM. Each experiment was executed 100 times – we report the one that took the lowest time.

*Performance study.* We measured the cost in time for the four table search strategies presented in Section 4.1.4 (Table Scan, Pivot Table Scan, Table Intersection, Incremental Table Intersection). We have also measured how the strategies work under different configurations (with/without a limit and using Relaxed/Restricted Mode). As fixed parameters, we use  $P_{LIMIT} = 10$  (when applied) and  $P_{THRESHOLD} = 0.7$ . Also, we use  $P_{INCREMENT} = 50$  for the Incremental Table Intersection strategy.

In our experiments, we used different queries, varying the number of variables and subgoals. We report the results obtained for the following query:

$Q_e(\text{title}, \text{year}, \text{director}, \text{genre}) :-$   
 $\text{movie}(\text{title}, \text{year}, \text{director}, \text{genre}).$

The results for the other queries are not considered because they present a similar behaviour.

Figure 6 shows the time each strategy took to find the table matches for data sets of different sizes (from 10 to 100,000 tables). Note that log scale is used on both X and Y axes.

Considering the Table Scan strategy, the worst configurations are those that do not use a limit. They take longer to execute because they need to scan every single source table. The Pivot Table Scan is a bit faster than Table Scan, but it still takes exponential time to perform. Both Table Scan and Pivot Table Scan are faster in Restricted Mode, since they can stop checking the column matches of a table when they find out that one of the variables is not matched to the table.

Table Scan performs better when there is a limit, since it does not need to go through all the source tables – it stops after finding the first ten tables matches (whose score is greater than 0.7). Besides, Table Scan (with a limit) runs faster in Relaxed Mode, since it finds tables matches sooner (considering that in Relaxed Mode a table can match a subgoal even if not every variable of the subgoal matches a column of the table).

As for Table Intersection, it takes the same amount of time regardless of the set up used. The reason for this is that the table intersection needs to be performed in any case, whether in Relaxed/Restricted mode or With/Without a limit.

The Incremental Table Intersection is only measured when there is a limit, after all, it uses the limit value

as a key piece in its processing. We can see that there is a improvement when comparing it against the Table Intersection. In fact, the Incremental Table Intersection takes a relatively constant time to compute the table matches regardless of the size of the data set.

### 5.2 Answer quality

To measure the quality of the rewritings returned by different configurations of our search engine, we have indexed a collection of tables extracted from HTML pages.

*Data set.* We used the ACHE focused crawler (Barbosa and Freire, 2007) to collect tables in the movie domain. ACHE was configured to retrieve pages that have at least one HTML table whose header has at least three columns and where either `title`, `film` or `movie` appear in the header. The purpose of this condition was two fold:

- detect tables that are used for data tabulation purposes instead of formatting purposes
- discover relevant sources of data for the movie domain.

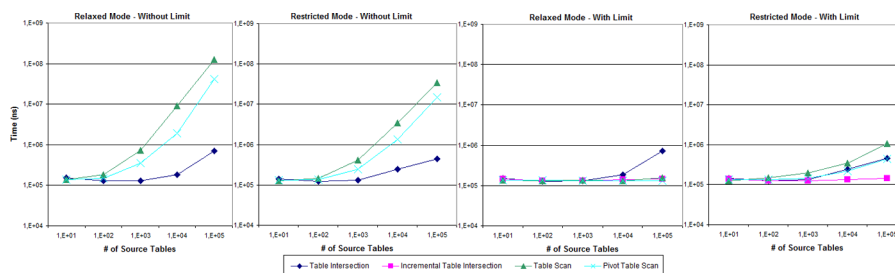
A Wikipedia page that brings a list of the 100 best American movies <sup>4</sup> was used as the seed for the crawl. We stopped crawling after 400 Web pages were retrieved. From this collection of pages, we extracted 993 HTML tables that satisfy our condition. We manually checked a sample of 10% of the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

*User study.* We have asked two students to formulate SQL queries asking for information about the movie domain. For each query, we ran different configurations – as explained below. We then evaluated the precision for the top  $k$  rewritings, for  $k = 1, 5, 10, 20$ .

The precision was measured based on the number of columns that were correctly retrieved. For instance, if the query asks for five columns, and the first rewriting returns four correctly matched 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 the previous rewritings into account).

Each student formulated five queries. From the resulting ten queries, we choose to report the results of four of them (the selected queries are good representations of the whole set).

**Figure 6** Execution time (see online version for colours)



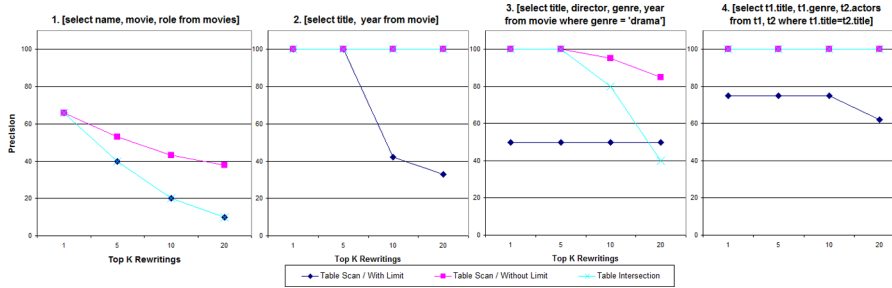
**Figure 7** Quality of the rewritings time (see online version for colours)

Figure 7 shows the results for three different configurations: Table Scan with limit, Table Scan without limit and Table Intersection. All three strategies run under the Relaxed Mode.

Configurations with Restricted mode are not discussed because they behave similarly to configurations that use the Table Intersection. Regarding the intersection-based strategies, there is hardly any difference if we use it with/without increments, with/without limit and in Restricted/Relaxed mode, so these variations are not considered. Also, we decided to show the Table Scan instead of Pivot Table Scan because the former brings at least the same answers as the latter, and their execution time is similar. As fixed parameters, we use  $P_{LIMIT} = 20$  (when applied) and  $P_{THRESHOLD} = 0.7$ .

The results show that Table Scan without limit performs at least as well as the other strategies for all cases. This makes sense since it traverses the whole repository looking for the best matches. On the other hand, Table Scan with limit is the worst configuration. The reason is that the matcher stops after finding  $P_{LIMIT}$  rewritings whose score is greater than  $P_{THRESHOLD}$ . However, in most cases, those rewritings do not cover all columns that were asked. For instance, in the query asking for `title` and `year`, only the seven first rewritings cover both columns.

The Table Intersection strategy (and others in the same group) performs better in queries 2 and 4. The reason is that there are enough tables that cover all columns of the queries. Take query 2, for instance. There are more than 20 tables that contain columns `title` and `year`.

On the other hand, precision drops in queries 1 and 3. The reason was that there are fewer than 20 tables that cover all columns of the queries. Take query 1, for instance. There are no source columns that are similar to `name`, so this column remains unmatched. As for the columns `movie` and `role`, there are only three source tables that cover them both, so only three rewritings are generated. The same thing happens to query 3. In this case, only eight source tables cover all matched columns of the query.

We can see that for some queries, no strategy is able to keep the precision at 100% for the top-20. In some cases, this happens simply because the repository is reaching a small subset of the Web, and most available tables cannot

answer for that information need. For instance, there is no source table that covers all columns of query 1 (the variable name cannot be matched). Furthermore, there are only eight source tables that cover all columns of query 3.

In the case of query 3, 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 if we apply a better matching technique. For instance, the source tables use either `title`, `movie` or `film` for representing the title of a movie. If we use this information during the matching, we could get a better precision, specially for query 1.

Precision could also be improved if additional information was 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.

Even if some information is not directly returned to the user, it is possible to take the initial answers as a starting point, and find out additional information by exploratory means, such as refining the query, checking the URL or going straight to the data source.

## 6 Related work

Current approaches to data integration rely on pre-defined mappings between a global schema and the underlying information sources (Ullman, 2000; Halevy, 2000; McBrien and Poulouvasilis, 2003; Friedman et al., 1999). This architecture, however, is not suitable for dataspace, where there is a very large number of information sources and these sources are highly autonomous and volatile.

In an attempt to improve scalability, recent approaches have been proposed to amortise the cost of integration, such as for example, peer-to-peer

systems (Tatarinov et al., 2003) and community-based information integration systems (McCann et al., 2003). 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 (Chawathe et al., 1994; Kirk et al., 1995).

Instead of a pre-defined global schema, some search engines use specialised indexes that store information about the sources, which can include both data and metadata (Dong and Halevy, 2007; Madhavan et al., 2007). Similar to our idea in nature, these approaches are more dataspace oriented, and are designed to handle the volatile nature of Web data. The main difference resides on the type of query supported: queries are expressed as plain keywords, instead of a more structured language (e.g., SQL). In this sense, a keyword can refer to either a value of a column or the name of the column itself. During the generation of answers, the keywords are identified in the indexes, and some correlation algorithm is used to find the relevant sources of information.

Keyword search has also been proposed in the light of relational databases (Hulgeri and Nakhe, 2002; Agrawal et al., 2002; Hristidis and Papakonstantinou, 2002). In BANKS, for example, information is viewed as a graph of tuples connected by foreign key relationships (Hulgeri and Nakhe, 2002). The keywords of the queries are matched against the tuples, and the subgraphs that contain all matched tuples are present as possible answers. Since these approaches focus on single relational databases, it is not clear if and how they could be extended to scale to a very large number of tables.

The benefits from the simpler keyword based approach is that they are easier to use. On the other hand, the expressiveness of the queries becomes limited. For instance, it is not possible to pose queries which contain predicate filters. Additionally, the ability to specify joins allows the user to discover associations between related sources that would be hard to find automatically (in a scenario where foreign key relationships are not known).

Madhavan et al. (2007) also discuss how the index of a dataspace can be improved in a pay as you go fashion, where user feedback plays an important rule to enrich the information of the indexes. The same ideas could be applied to our work as well. After all, rewritings are presented to the user individually. Hence, it would be possible for the user to mark an incorrect rewriting, and leave it up to the system to learn from that information.

## 7 Discussion and future work

In this paper we present a new framework that supports ad-hoc structured queries over dataspace 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 dataspace, 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 column names of the source tables (Section 4.1). 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 (He and Chang, 2003; Rahm and Bernstein, 2001), we could use both the source metadata and contents to determine the compatibility of the sources and prune rewritings that involve incompatible sources.

## References

- Abiteboul, S. and Duschka, O.M. (1998) 'Complexity of answering queries using materialised views', *ACM Symposium on Principles of Database Systems*, Seattle, Washington, USA, pp.254–263.
- Agrawal, R., Ailamaki, A., Bernstein, P.A., Brewer, E.A., Carey, M.J., Chaudhuri, S., Doan, A., Florescu, D., Franklin, M.J., Garcia-Molina, H., Gehrke, J., Gruenwald, L., Haas, L.M., Halevy, A.Y., Hellerstein, J.M., Ioannidis, Y.E., Korth, H.F., Kossmann, D., Madden, S., Magoulas, R., Ooi, B.C., O'Reilly, T., Ramakrishnan, R., Sarawagi, S., Stonebraker, M., Szalay, A.S. and Weikum, G. (2008) 'The claremont report on database research', *SIGMOD Rec.*, Vol. 37, No. 3, pp.9–19. ISSN 0163-5808.

- Agrawal, S., Chaudhuri, S. and Das, G. (2002) 'Dbxplorer: a system for keyword-based search over relational databases', *International Conference on Data Engineering (ICDE)*, IEEE Computer Society, Washington DC, USA, pp.5–16.
- Arvelo, Y., Bonet, B. and Vidal, M.E. (2006) 'Compilation of query-rewriting problems into tractable fragments of propositional logic', *National Conference on Artificial Intelligence (AAAI)*, Boston, MA, pp.225–230.
- Barbosa, L. and Freire, J. (2007) 'An adaptive crawler for locating hidden-Web entry points', *Proceedings of the 16th International Conference on World Wide Web*, Banff, Alberta, Canada, pp.441–450.
- Cafarella, M.J., Halevy, A., Wang, D., Wu, E. and Zhang, Y. (2008) 'Webtables: exploring the power of tables on the Web', *VLDB Endowment*, Vol. 1, No. 1, pp.538–549.
- Chakrabarti, S., Punera, K. and Subramanyam, M. (2002) 'Accelerated focused crawling through online relevance feedback', *Proceedings of the 11th International Conference on World Wide Web*, Honolulu, Hawaii, USA, pp.148–159.
- Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J.D. and Widom, J. (1994) 'The TSIMMIS project: integration of heterogeneous information sources', *16th Meeting of the Information Processing Society of Japan*, Tokyo, Japan, pp.7–18, URL [citeseer.ist.psu.edu/chawathe94tsimmis.html](http://citeseer.ist.psu.edu/chawathe94tsimmis.html)
- Chekuri, C. and Rajaraman, A. (2000) 'Conjunctive query containment revisited', *Theoretical Computer Science*, Vol. 239, No. 2, pp.211–229, URL [citeseer.ist.psu.edu/chekuri98conjunctive.html](http://citeseer.ist.psu.edu/chekuri98conjunctive.html)
- Dong, X. and Halevy, A. (2007) 'Indexing dataspace', *ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, ACM, pp.43–54, ISBN 978-1-59593-686-8.
- Duschka, O.M. and Genesereth, M.R. (1997) 'Answering recursive queries using views', *ACM Symposium on Principles of Database Systems*, Tucson, Arizona, USA, pp.109–116.
- Franklin, M., Halevy, A. and Maier, D. (2005) 'From databases to dataspace: a new abstraction for information management', *SIGMOD Rec.*, Vol. 34, No. 4, pp.27–33, ISSN 0163-5808.
- Friedman, M., Levy, A.Y. and Millstein, T.D. (1999) 'Navigational plans for data integration', *AAAI/IAAI*, pp.67–73, URL [citeseer.ist.psu.edu/friedman99navigational.html](http://citeseer.ist.psu.edu/friedman99navigational.html)
- Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V. and Widom, J. (1997) 'The tsimmis approach to mediation: data models and languages', *Journal of Intelligent Information Systems*, Vol. 8, No. 2, pp.117–132.
- Halevy, A.Y. (2000) 'Theory of answering queries using views', *SIGMOD Record (ACM Special Interest Group on Management of Data)*, Vol. 29, No. 4, pp.40–47, URL [citeseer.ist.psu.edu/halevy00theory.html](http://citeseer.ist.psu.edu/halevy00theory.html)
- Halevy, A., Franklin, M. and Maier, D. (2006) 'Principles of dataspace systems', *PODS '06: Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, New York, NY, USA, ACM, pp.1–9, ISBN 1-59593-318-2.
- He, B. and Chang, K.C.-C. (2003) 'Statistical schema matching across Web query interfaces', *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, California, pp.217–228.
- Hristidis, V. and Papakonstantinou, Y. (2002) 'Discover: keyword search in relational databases', *International Conference on Very Large Data Bases (VLDB)*, Hong Kong, China, pp.670–681, VLDB Endowment.
- Hulgeri, A. and Nakhe, C. (2002) 'Keyword searching and browsing in databases using banks', *International Conference on Data Engineering (ICDE)*, San Jose, CA, pp.431–440.
- Kirk, T., Levy, A.Y., Sagiv, Y. and Srivastava, D. (1995) 'The information manifold', *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, Stanford, California, pp.85–91, URL [citeseer.ist.psu.edu/16242.html](http://citeseer.ist.psu.edu/16242.html)
- Levy, A.Y., Rajaraman, A. and Ordille, J.J. (1996) 'Querying heterogeneous information sources using source descriptions', *Proceedings of the Twenty-second International Conference on Very Large Databases*, pp.251–262, URL [citeseer.ist.psu.edu/levy96querying.html](http://citeseer.ist.psu.edu/levy96querying.html)
- Madhavan, J., Cohen, S., Dong, X.L., Halevy, A.Y., Jeffery, S.R., Ko, D. and Yu, C. (2007) 'Web-scale data integration: You can afford to pay as you go', *CIDR*, pp.342–350, [www.crdrrdb.org](http://www.crdrrdb.org), URL <http://dblp.uni-trier.de/rec/bibtex/conf/cidr/MadhavanCDHJKY07>
- McBrien, P. and Poulouvasilis, A. (2003) 'Data integration by bi-directional schema transformation rules', *Proceedings of the 19th International Conference on Data Engineering*, Bangalore, India, pp.227–238.
- McCann, R., Doan, A.H., Varadarani, V., Kramnik, A. and Zhai, C.X. (2003) 'Building data integration systems: a mass collaboration approach', *WebDB*, pp.25–30.
- Pottinger, R. and Levy, A.Y. (2000) 'A scalable algorithm for answering queries using views', *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp.484–495, ISBN 1-55860-715-3.
- Rahm, E. and Bernstein, P.A. (2001) 'A survey of approaches to automatic schema matching', *The VLDB Journal: The International Journal on Very Large Data Bases*, Vol. 10, No. 4, pp.334–350.
- Tatarinov, I., Ives, Z., Madhavan, J., Halevy, A., Suciu, D., Dalvi, N., Dong, X.(L.), Kadiyska, Y., Miklau, G. and Mork, P. (2003) 'The piazza peer data management project', *SIGMOD Record*, Vol. 32, No. 3, pp.47–52.
- Ullman, J.D. (2000) 'Information integration using logical views', *Theoretical Computer Science*, Vol. 239, No. 2, pp.189–210, URL [citeseer.ist.psu.edu/ullman97information.html](http://citeseer.ist.psu.edu/ullman97information.html)

## **Notes**

<sup>1</sup>As we describe later, the name of the table `movies` is not used in the rewriting, only the attribute names are considered.

<sup>2</sup>For efficiency, results could also be cached.

<sup>3</sup>The code can be downloaded from <http://www.cs.utah.edu/~juliana/downloads/Carla.java>

<sup>4</sup>The list was created by the American Film Institute [http://en.wikipedia.org/wiki/100\\_Years...100\\_Movies](http://en.wikipedia.org/wiki/100_Years...100_Movies)