Exploring Schema Repositories with Schemr

Kuang Chen and Akshay Kannan
University of California, Berkeley
kuangc@cs.berkeley.edu,
akannan@cs.berkeley.edu

Jayant Madhavan and Alon Halevy
Google, Inc.
jayant@google.com,
halevy@google.com

ABSTRACT

Schemr is a search engine for users to search for and visualize schemas in a metadata repository. Users may search by keywords and by example, using schema fragments as query terms. Schemr uses a novel search algorithm, based on a combination of text search and schema matching techniques, coupled with a structurally-aware scoring metric. Schemr presents search results in a GUI that allows users to explore which elements match and how well they do. The GUI supports interactions, including panning, zooming, layout and drilling-in. This paper introduces Schemr as a new component of the information integration toolbox and discusses its benefits in several applications.

1. INTRODUCTION

All around the world, groups of small organizations want to share structured data with each other. For instance, consider the Nature Conservancy's[1] efforts in rallying small conservation organizations to contribute environmental monitoring data. As another example, consider a rural health system in sub-Saharan Africa, consisting of community health workers, low-resource health clinics, and district-, regional-, and national-level ministries of health. Unlike typical data integration scenarios, where the goal is to provide uniform access to multiple *existing* data sets, here organizations are more willing to share information right from the beginning. In particular, a database designer working on a new schema is likely to consult and explore existing schemata, given access to them. What is needed is a tool for schema search and visualization to guide the initial database design.

This paper describes Schemr, a tool for locating, exploring, and reusing relevant schemas (or schema fragments) in large schema collections. Schemr leverages the past experience of a collaborative community and the algorithmic techniques from existing information integration tools to lower the data sharing barrier-to-entry and nurture schema compatibility – simplifying the task of information integration during schema design. Beyond the example scenarios described above, a schema search tool is a valuable tool to navigate any dataspace of heterogeneous information [4].

[1]The Nature Conservancy http://www.nature.org

Schemr's search algorithm combines schema matching and text search techniques with a structurally-aware scoring metric. Designers can use keywords or existing schema fragments as search terms. Results are returned in an environment that allows users to visually explore and compare matching schemata.

Specifically, we make the following contributions:

- Schema search algorithm - Schemr's search algorithm combines techniques from document search and schema matching, and employs a holistic *tightness-of-fit* measure to find and rank schemas according to a query's semantic intent.
- Visualizations of results - Schemr visualizes search results in an interactive web application, allowing users to compare multiple results and drill-in to explore a schema with visually encoded similarity measures.
- Open source software component - Schemr is part of the OpenII open-source information integration framework [8], which any organization may use and extend for free.

A demonstration of Schemr was first presented at SIGMOD 2009 [2].

Example Scenario

We ground our motivation through discussion and observation of two such organizations, mentioned above: the Nature Conservancy and a large HIV/AIDS treatment program in Tanzania. We found that data management in these organizations takes place in an ad-hoc manner, with ad-hoc tools. Data administrators face a vicious cycle: they are overloaded with requests to manually curate data that should be produced by automated processes. Thus, having no time to tackle major system improvements, they create stopgap solutions. These data administrators say that they would gladly collaborate with others to share schemas and advice, but are hindered by the high-maintenance cost of the stopgap solutions. They need tools that provide an immediate productivity gain. For these organizations, sharing designs through schema search can provide this bootstrap path, which starts with better data modeling and leads to better integrated information.

In such a setting, the database administrator begins by designing a new table. She is unsure of the best way to model the table, and wants to search for related schemas and data examples. She opens Schemr in her web browser and has the option to specify either a simple keyword search and/or a partially designed schema fragment. In this case, let us suppose that she performs a search for existing data models by using the keywords `patient, height, gender, diagnosis`. Additionally, she specifies a partially designed schema to specify results that include elements semantically equivalent to ones she has already designed. A partially designed schema can be specified by uploading a *DDL* (Data Definition Language) or *XSD* (XML Schema Definition).

Upon executing this query, the designer is presented with several relevant schemata to explore in further depth. The results can come from a variety of sources: reference schemata within the organization, shared schemata from partnering organizations, or public sources.

Internally, Schemr parses the input schema and creates a *query graph* (Figure 1) out of it, on which the similarity functions are computed. Schemr returns a ranked list of n results, presented in a tabular format, including columns for name, score, matches, entities, attributes, and description. The user can interact with the results by clicking on a particular entry to visualize its schema elements, or ask for the next n schemas. On drill-in to a particular schema, Schemr creates a detailed graph structure with visual encodings of similarity. Figure 2 shows an example of Schemr's visualizations.

2. SCHEMR OVERVIEW

In this section, we first describe Schemr's search algorithm, and then describe our implementation.

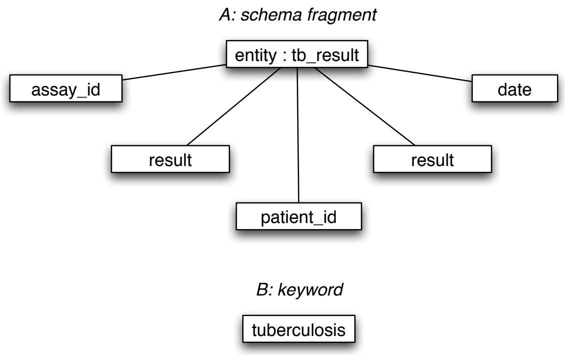


Figure 1: An example query graph consisting of both (A) a schema fragment and (B) a keyword

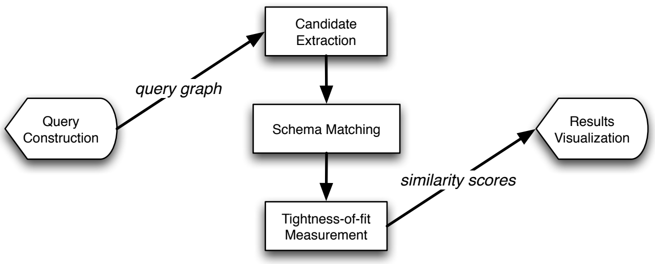


Figure 3: Schema search algorithm data flow

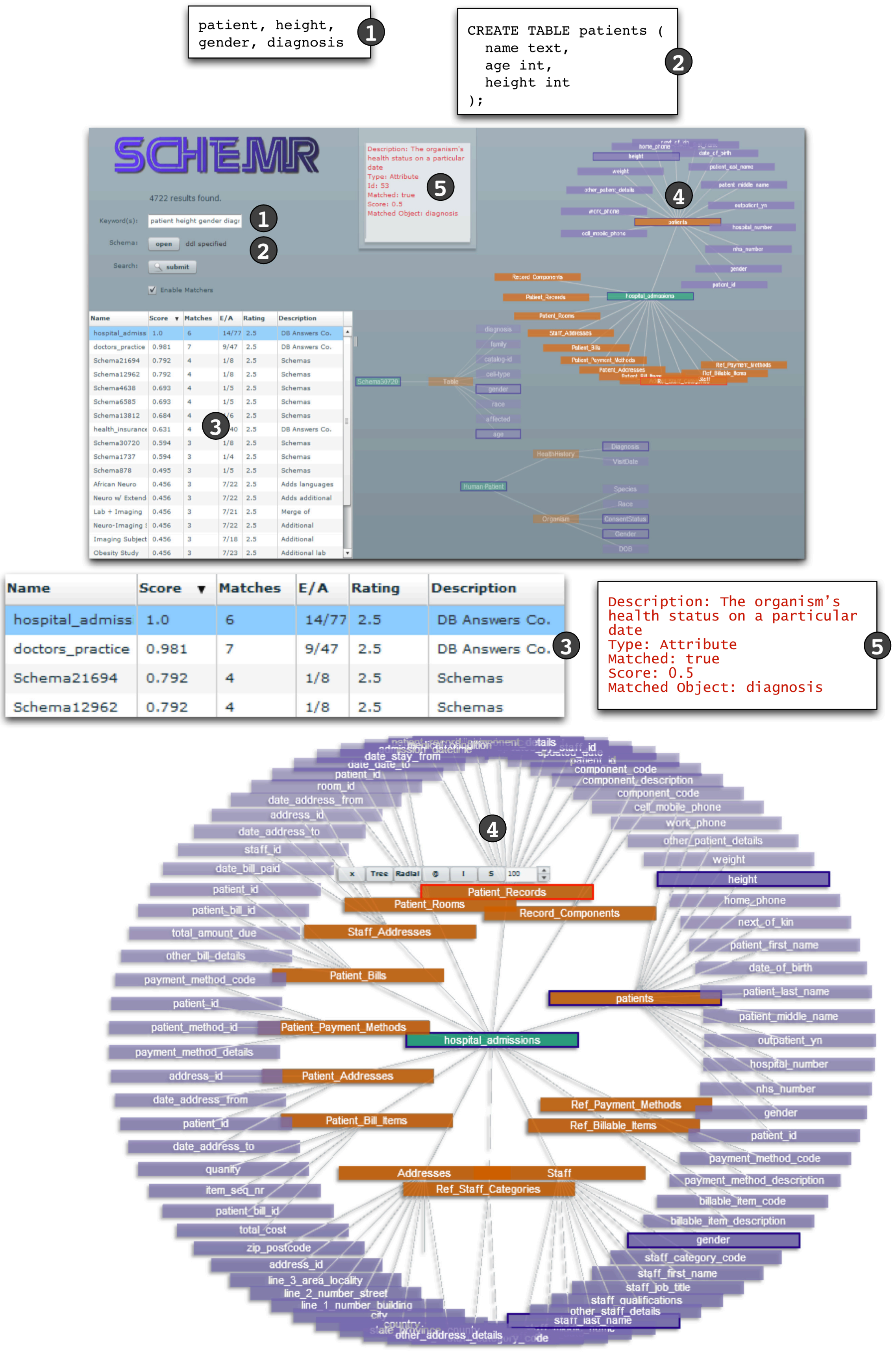
Algorithm

Schemr's search algorithm (Figure 3) consists of three phases. Prior to executing a search, the query parser creates a query-graph from the keyword terms and schema fragments given by user input. In the first phase, *Candidate Extraction*, Schemr flattens the query-graph into a list of keywords and quickly retrieves the top candidate schemas from a scalable document index. In the second phase, *Schema Matching*, Schemr evaluates the top candidate schemas with an ensemble of schema matchers [3, 6], scoring the semantic similarity between candidate schema and the query-graph elements. In the third phase, Schemr computes a final score by weighing similarity scores with a *Tightness-of-fit Measurement*.

Candidate Extraction

The input query-graph Q is a forest of trees consisting of schema fragments and keywords, as shown in Figure 1. The example illustrates that Q can represent several graphs, where each keyword is represented as a graph of one item. The query-graph abstraction can capture multiple query formats, including relational and XML.

The system contains a document index of the schema corpus, which we build offline. Each schema



```
patient, height,
gender, diagnosis
```

```
CREATE TABLE patients (
  name text,
  age int,
  height int
);
```

Name	Score ▼	Matches	E/A	Rating	Description
hospital_admiss	1.0	6	14/77	2.5	DB Answers Co.
doctors_practice	0.981	7	9/47	2.5	DB Answers Co.
Schema21694	0.792	4	1/8	2.5	Schemas
Schema12962	0.792	4	1/8	2.5	Schemas
Schema4638	0.693	4	1/5	2.5	Schemas
Schema6585	0.693	4	1/5	2.5	Schemas
Schema13812	0.684	4	1/6	2.5	Schemas
health_insurance	0.631	4	[illegible]	2.5	DB Answers Co.
Schema30720	0.594	3	1/8	2.5	Schemas
Schema1737	0.594	3	1/4	2.5	Schemas
Schema878	0.495	3	1/5	2.5	Schemas
African Neuro	0.456	3	7/22	2.5	Adds languages
Neuro w/ Extend	0.456	3	7/22	2.5	Adds additional
Lab + Imaging	0.456	3	7/21	2.5	Merge of
Neuro-Imaging S	0.456	3	7/22	2.5	Additional
Imaging Subject	0.456	3	7/18	2.5	Additional
Obesity Study	0.456	3	7/23	2.5	Additional lab

```
Description: The organism's
health status on a particular
date
Type: Attribute
Id: 53
Matched: true
Score: 0.5
Matched Object: diagnosis
```

Figure 2: Search results for a keyword + schema fragment query. (1) Search keywords (2) DDL schema fragment specified as part of query (3) Tabular view of search results allows sorting and comparison (4) Schema visualizations allow side-by-side schema comparison. Node color corresponds to schema element types (e.g. entity or attribute). Visualization types include hierarchical tree-view and radial view (shown). Nodes can be collapsed and expanded to allow drill-in on particular schema elements in greater detail.

in the index is represented as a document, for which we store a title, a summary, an ID, and a flattened representation of each element in the schema. Our inverted index stores a term dictionary of frequency data, proximity data, and normalization factors, providing a fast and scalable filter for relevant candidate schemas.

When searching the index online, we first create a list of keywords by flattening the query graph Q and performing keyword matching on the document index. We use a variant of standard TF/IDF to obtain an initial coarse-grain matching. To preserve recall, the candidate extraction algorithm need not match all search terms; rather, match scores are computed independently for each search term and summed to produce a coarse-grain score for returning the top n candidate results. A coordination factor, defined as the number of terms matched divided by the number of terms in the query, is multiplied into the coarse-grain score in order to reward results which match the most terms in the original query.

Schema Matching

The top candidate schemas are evaluated against the query-graph and ranked using an ensemble of fine-grained matchers. We summarize two matchers we found to be most useful, but note that other matchers may be used as well.

A *name matcher* normalizes terms and computes n-gram overlap between query terms and terms in the indexed schemas. Each schema element in the query is parsed into a set of all possible n-grams, ranging in length from one character to the length of the word. Each n-gram is then ranked against each element of the candidate schemas to compute a final match score. We found this matcher to be particularly helpful for properly ranking schemas containing abbreviated terms, alternate grammatical forms, and delimiter characters not in the original query.

A *context matcher* builds a set of terms from neighboring elements, and tries to capture matches when neighboring-element sets are similar to each other [6].

Each matcher produces a similarity matrix between query graph elements and schema elements. Each (query element, schema element) pair has a corresponding value which describes the match quality – a value between 0 and 1, For every candidate schema, the similarity matrices of the different matchers are combined into a single matrix containing *total similarity scores*. We combine the scores from each matcher with a weighting scheme, which is initially uniform. As Schemr is utilized in practice, we can record search histories to create a training set of search-term to schema-fragment matches. With such a training set, we may then determine an appropriate weighting scheme. For instance, Madhavan *et al* use a "meta-learner" to compute a logistic regression over a training set of schemas [5].

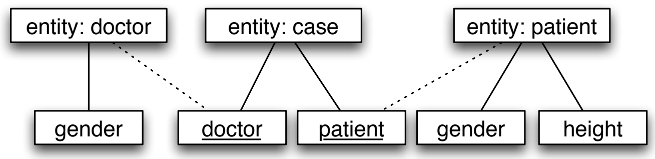


Figure 4: An example schema showing only *matched* schema elements

Tightness-of-fit Measurement

Schemr's task, in this phase, diverges from the traditional aim of schema matching: rather than generating mappings between elements, we use the similarity matrix of total similarity scores to create an overall score that captures the semantic intent of schema search. Our principle here is to measure the tightness-of-fit by minimizing the distance between relevant elements in a result schema.

We begin by selecting the maximum value of each schema element's entry in the matrix as the final match score for that element. Next, we apply penalties to the scores of the schema elements based on a relative distance measure and take the average of the scores to arrive at a final score for the entire schema.

The intuition behind our distance measure is as follows. For elements e_i and $e_j \in E$:

- If they are in the same entity, no penalty.
- If they are in the same entity neighborhood (transitive closure on foreign key), then a small penalty applies.
- If they are in unrelated entities, then a larger penalty applies.

There can be many configurations by which a set of query-graph elements match a set of result schema elements. Each such configuration consists of penalties on elements computed with respect to a particular anchor entity. Given an anchor entity, the scores of elements in other entities are penalized by their distances to the anchor and averaged. This calculation is repeated for all possible anchor entities, and the maximum of all calculations is selected as the final match score for the schema.

Continuing with our original health clinic example, consider the following simplified candidate schema

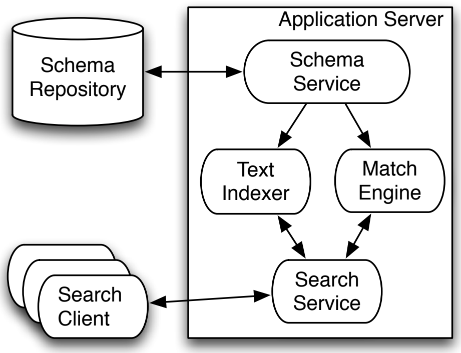


Figure 5: Schemr system architecture diagram

of matched elements in Figure 4. First, `case` is selected as an initial anchor entity. No penalty is applied to the scores of the `case.doctor, case.patient` schema elements, because they reside in the same entity as the anchor, whereas a small transitive closure penalty is applied to the scores of `patient.height, patient.gender, doctor.gender`. Finally, the penalized scores of the schema elements are averaged to produce a score for the `case` anchor. Next, `patient` is selected as an anchor entity. No penalty is applied to `patient.height, patient.gender`, the small transitive closure penalty is applied to the elements in the `case` entity, and a larger penalty is applied to the elements in the unrelated `doctor` entity. Finally, this calculation is repeated with `doctor` as the final anchor entity, and the maximum value of the three anchored calculations is returned as the final match score of the schema.

For a set of similarity scores S, each choice of anchor element A results in penalties P. A tightness-of-fit score t can be computed by $t = \sum(S \cdot P)$. We are interested in the configuration which maximizes the tightness-of-fit score:

$$t_{max} = \arg\max_{A} \sum(S \cdot P_A).$$

We use this total score to rank the final search results.

Architecture and Implementation

Schemr's architecture (Figure 5) features a web-based GUI for entering search terms and graphically reviewing search results. The GUI processes a set of search terms and delivers them as a request to the *Search Service*.

On the Schemr server, we use the open-source schema repository Yggdrasil [8]. At scheduled intervals, an offline Lucene [11] *Text Indexer* flattens schemas from the *Schema Repository* to construct or update the document index.

When a request is received by the server, the query is initially flattened into a collection of keywords and used to filter candidate schemas from the document index. These candidate schemas are next passed to the *Match Engine*, where fine-grained matchers are used to compute a final relevance score for ranking the candidate schemas. This list of candidate schemas, along with their corresponding score, is finally sent as an XML response to the client.

When the user clicks on a search result to view the visualization, another request containing the schema ID is sent to the server. The server performs a lookup of this ID in the schema repository and returns a graphical representation of the schema to the client as a GraphML[10] response, which is parsed and displayed on the front-end.

Visualizations

Schemr visualizes result schemas in an interactive GUI, supporting panning, zooming, auto-layout, and drilling-in. Our client is implemented using Adobe Flex and the Flare visualization toolkit. Using Flash ensures cross-browser compatibility without any additional browser-handling code. All search requests and visualizations are dynamically retrieved using asynchronous HTTP requests.

Schemr's user interface features two panels (Figure 2). The left-side search panel allows users to supply a query in the form of a keyword search or a DDL/XSD schema fragment and lists ranked search results in a tabular format. The right-side results panel provides a workspace for users to explore graph visualizations of schemas. In graph visualizations, element nodes are encoded by color, and multiple graphs can freely be compared side-by-side and explored in further depth. Clicking on a graph node displays detailed information about the schema element in a toolbox, and double-clicking on a graph node re-centers the layout of the graph such that the new node is in the center. We allow for multiple graph layouts, including a hierarchical tree layout and a radial layout. To ensure Schemr scales to very large schemas, we cap the displayed graph depth to 3. To drill in on a particular branch at a greater depth, users can simply double click on a node to view its descendants in further detail. To ensure Schemr scales to very large schemas, we plan to employ schema visualization and summarization techniques, such as those proposed in [7, 9].

Applications

Schemr's search capabilities have been tested on a repository of over 30,000 public schemas, both relational and semi-structured, small and large, spanning many domains. These schemas came a collection of 10 million HTML tables [1], and were filtered by removing schemas containing non-alphabetical characters, schemas that only appeared once on the web, and trivial schemas with three or less elements.

We plan to make Schemr available as a part of an open-source information integration framework, OpenII [8]. As a module of OpenII, other framework components enable new schema search applications and scenarios, magnifying Schemr's benefit. For example, integrating Schemr with schema import and export functionality gives users motivation to build metadata repositories. As well, integrating Schemr's search functionality with a codebook that contains data types like units, date/time, and geographic location, would encourage a deeper standardization of data types alongside schema search results. With an OpenII community of users searching the repository, collaboration functionality that provides usage statistics and comments on schemas would improve schema search results. Finally, integrating Schemr with a schema editor would allow for a new model development process, in which search results are iteratively used to augment a schema. In this process, we can also capture implicit semantic mappings between schema elements, information on schema re-use, and the provenance of new schema entities.

3. SUMMARY

Schemr demonstrates an effective approach to schema search and visualization. It uses a novel combination of document based filtering, schema matching, semantics, and structure-aware scoring to efficiently search and visualize large schema repositories. Schemr can be internally deployed as a standalone tool for organizations to search and share schemas, facilitating the schema design process and paving the way for information integration. Additionally, Schemr will play a role as a module of the OpenII framework, serving to improve the accessibility and benefit of many information integration applications.

Schemr can also be deployed as a publicly available web service. To facilitate finding quality schemas in a large public repository, we plan to incorporate collaborative functionality such as mechanisms for users to leave ratings and comments on schemas. Through these comments, users can suggest improvements or additions that can be made to schemas. Ultimately, we hope that this will evolve into a general repository for storing multi-purpose schemas to meet the community's needs. In a sense, we are hoping to democratize development of standards and consequently improve the quality of schemas in the data ecosystem.

Acknowledgments

We are grateful to Peter Mork, Arnie Rosenthal, Len Seligman and Chris Wolf who provided invaluable advice and the Yggdrasil schema repository. We would like to thank Kristin Barker, Harr Chen, Tyson Condie, Joe Hellerstein, Neal Lesh, Jamie Lockwood, Tapan Parikh and Sanjay Unni.

4. REFERENCES

[1] M. Cafarella, A. Halevy, D. Wang, E. Wu, and Y. Zhang. Webtables: Exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[2] K. Chen, J. Madhavan, and A. Halevy. Exploring schema repositories with schemr. In *Proceedings SIGMOD*, pages 1095–1098. ACM, 2009.

[3] A. Doan, P. Domingos, and A. Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning*, 50(3), 2003.

[4] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *ACM Sigmod Record*, 34(4):27–33, 2005.

[5] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *Proceedings of ICDE*, pages 57–68. IEEE, 2005.

[6] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4), 2001.

[7] G. G. Robertson, M. P. Czerwinski, and J. E. Churchill. Visualization of mappings between schemas. In *Proceedings SIGCHI conference on Human factors in computing systems*, 2005.

[8] L. Seligman, P. Mork, A. Halevy, K. Smith, M. Carey, K. Chen, D. Burdick, C. Wolf, J. Madhavan, and A. Kannan. Openii: An open source information integration toolkit. In *Proceedings of SIGMOD*, 2010.

[9] C. Yu and H. V. Jagadish. Schema summarization. In *Proceedings VLDB*, 2006.

[10] Graphml file format. `http://graphml.graphdrawing.org`.

[11] Lucene. `http://lucene.apache.org`.