Bulletin of the Technical Committee on

Data Engineering

September 2011 Vol. 34 No. 3

IEEE Computer Society

Letters

Special Issue on Towards Quality Data with Fusion and Cleaning

Conference and Journal Notices

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TC on Data Engineering web page is http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

TCDE Activities

I wanted to give a very brief update on where the effort to change the governance of the Technical Activities Committee (TAC) stands. The governance subcommittee (chaired by Hilarie Orman from the TC on Security and Privacy) has produced a specific recommendation as to how to change the Computer Society rules to permit TC chairs to elect the chair of the TAC. As one of the subcommittee members, I am strongly supportive of the proposed change. The issue is presently being debated via email by the TC Chairs represented on the TAC. I hope that I can report again on this in the December issue, and tell you that the change in governance structure has progressed, perhaps even that it has been adopted.

The Current Issue

It never ceases to surprise me how much effort needs to go into "data processing" before the data can be usefully exploited or stored within a database system. Raw data is frequently very "dirty", containing typos or outright misinformation. This is a real burden on the users of the data. Further, to make textual data, e.g. from web sources, useful in a database system requires the introduction of "schema". And that requires information extraction technology. In industrial practice, this is all very resource intensive and costly. What is clearly needed are tools, by which I mean computer systems or applications, that are targetted at improving the quality and usefulness of the data. Such tools do exist, but much much more is needed.

The current issue directly addresses this subject, being on "Data Quality with Fusion and Cleaning". Wang-Chiew Tan and Xin Luna Dong, the issue editors work in this field themselves, know the state of the art, and knows those who are working in the area. The current issue thus contains a good cross section of the ongoing work, both from industry and academia. The diversity of the papers and their approaches reflects that no one approach will be the miracle solution. Rather, the field needs to make progress on multiple fronts to advance the state of the art. I want to thank Luna and Wang-Chiew, who have done a fine job in selecting the authors and in overseeing the issue. This is an area that cries out for more work. The current issue is a great place for starting to educate yourself about this area.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editors

The quality of data has always been important to businesses and intelligence, as policies and business decisions are often made based on analysis performed on data. This issue contains articles that explore different aspects of data quality that frequently arise in the context of multiple (Web) data sources providing overlapping, complementary, and sometimes contradictory information about the same concept or real world entity. It is therefore important to provide techniques for understanding the truthfulness and trustworthiness of data sources to the extent possible, and reconcile their differences in order to create a clean integrated view of the underlying data sources. Techniques for entity resolution, mapping, fusion, and data cleaning are important "ingredients" towards achieving such a clean unified view.

The first three articles in this issue describe different approaches for understanding and improving data quality in terms of truthfulness, trustworthiness, and respectively, eliminating (unnecessary) nulls. In the article *Truthfulness Analysis of Fact Statements Using the Web*, Li, Meng, and Yu describe an approach for determining the truthfulness of fact statements obtained from the Web. Whether a fact statement is considered true depends in part on the degree of truth of alternatives to the fact statement. In the next article *Corroborating Information from Web Sources*, Marian and Wu describe how one can exploit corroboration (i.e., similar answers provided by different data sources) to provide a measure of trust in the answers obtained from different sources. In the article *Eliminating NULLs with Subsumption and Complementation*, Bleiholder, Herschel, and Naumann describe how data quality can be improved by data fusion operators, subsumption and complementation, that minimizes the number of nulls in data. In addition, the authors described a set of rewrite rules based on these two operators and other database primitives that can be used to optimize the process of eliminating nulls.

The next two articles describe how constraints can be used to understand the quality of data and how different constraints that have been used for improving data quality can be captured uniformly in one language. In the article *Efficient and Effective Analysis of Data Quality using Pattern Tableaux*, Golab, Korn, and Srivastava describe the *Data Auditor* system, which can efficiently compute patterns of subsets of the underlying data that satisfy or violate a given constraint. The patterns derived are useful for understanding the semantics of the data. Fan and Geerts, in the article titled *A Uniform Dependency Language for Improving Data Quality*, describe how different dependency formalisms that have been used in the context of studying data quality can all be captured under their language of Quality Improving Dependencies (QID). They describe how a modification of the chase process can be used to enforce QIDs and thus, provide a uniform framework for reasoning and understanding data cleaning mechanisms behind different existing formalisms.

The rest of the articles in this issue describe generic frameworks towards data cleaning, entity resolution, and data integration in general. In the article *Towards a Domain Independent Platform for Data Cleaning*, Arasu *et al.* describe how various generic data cleaning primitives can be consolidated into a tool for facilitating the process of data cleaning. The primitives consist of various generic operators for determining textual similarity, clustering, and parsing. While these primitives are generic, the platform allows these operators to be customized to specific domains. Whang and Garcia-Molina described their efforts in a generic framework for entity resolution. They extended the core entity-resolution primitives for scalable iterative blocking, joint resolution, and incremental resolution in the article *Developments in Generic Entity Resolution*. They have also described how different existing quality measures could lead to different evaluations of the entity-resolution results and hence, proposed a single but configurable measure. Finally, the article *Extracting, Linking and Integrating Data from Public Sources: A Financial Case Study* showcases a real-world example of integrating various data sources from the financial domain. The integration process requires applications of techniques from mapping, entity resolution, and data fusion. As Burdick *et al.* described, the integrated result can be analyzed to provide valuable insights that is difficult to obtain otherwise. From this experience, the authors plan to develop and extend capabilities towards a generic, high-level infrastructure for large-scale data integration.

Xin Luna Dong and Wang-Chiew Tan
AT&T Labs-Research, IBM Research-Almaden and UC Santa Cruz

Truthfulness Analysis of Fact Statements Using the Web

Xian Li[1], Weiyi Meng[1] Clement Yu[2]
[1]Department of Computer Science, Binghamton University, USA
{xianli, meng}@cs.binghamton.edu
[2]Department of Computer Science, University of Illinois at Chicago, USA
cyu@cs.uic.edu

Abstract

Web users increasingly treat the Web as a very large knowledge base and use it to acquire all kinds of knowledge. In this article, we introduce a method that leverages the information on the Web to determine the truthfulness of any statement that attempts to state a fact. This method accomplishes the goal by checking whether there is an alternative statement that is more likely to be truthful. As a result, this method can find an alternative truthful statement when the given statement is not truthful. Our experimental results show that the proposed method is effective. In this article, we also provide some insights on how to extend this work and how to leverage this technique to estimate the trustworthiness of web pages and websites.

1 Introduction

The Web is being used by millions of users to acquire knowledge. Indeed, people can use the Web to find lots of useful knowledge such as the longest river in the world and the religion of President Barack Obama. But due to the existence of significant amount of untruthful information on the Web for various reasons such as typos, obsolete information and rumors, it is not always easy to determine the correct fact. The search result in Figure 1 is an example of a rumor-spreading article that attempts to portrait Barack Obama as a Muslim. It is important to develop useful tools that can help web users differentiate truthful and untruthful information.

In this article, we first introduce a method [1] for determining the truthfulness of any fact statement with a specified doubt unit and then discuss how to extend this work to more general situations. In this article, a *fact statement* is defined as a statement that attempts to state a fact, not an opinion. For example, "United States has 51 states" is a fact statement although the stated fact is not correct. In contrast, "Stinky Tofu is the worst dish" expresses an opinion.

We developed a system called T-verifier [4] for verifying the truthfulness of any fact statement. A user submits a fact statement, whose truthfulness is uncertain to the user, to T-verifier and indicates which part of the statement he/she is not certain about. Such a statement is called a *doubtful (fact) statement* and the uncertain part is called the *doubt unit* of the statement. For example, a doubtful statement may be "United States has [51] states", where [ ] indicates the doubt unit. T-verifier determines the truthfulness of a doubtful statement DS in two phases: (1) *Alternative statements* that are on the same topic as DS are generated from the *search result records* (SRRs) retrieved from a search engine using a query derived from DS. (2) The alternative statements are ranked based on features extracted from newly retrieved SRRs using each alternative statement as a query and

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1]This method was first reported in [4].

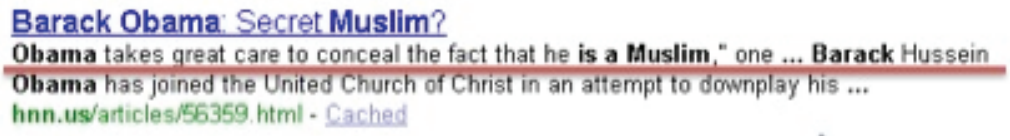


Figure 1: Results of "Barack Obama is Muslim" (Yahoo.com on 12/20/2009)

the top-ranked alternative statement is recognized as the truthful statement. In this way, T-verifier not only tries to determine the truthfulness of the doubtful statement but also tries to provide a relevant correct statement if the doubtful statement is not considered to be truthful.

The rest of the paper is organized as follows. In Section 2, we introduce a method to generate relevant alternative statements from a given doubtful statement. In Section 3, we discuss how to rank the alternative statements so as to determine the truthful statement. In Section 4, we report some experimental results. In Section 5, we briefly review some related works. In Section 6, we discuss some future extensions to the current work. Finally, a brief summary is given in Section 7.

2 Alternative Statements Generation

Let DS denote a given doubtful statement with a specified doubt unit DU. A doubt unit can be a single word, a phrase, a name entity, a number or a date, etc. Alternative statements to DS are very important to the truthfulness judgment, because they provide different versions of the fact and the truthful one is probably one of them. Different alternative statements should be different on the doubt unit. We call the term(s) in place of the doubt unit *alternative unit* (*alter-unit* for short). A relevant alternative statement should have the following properties:

- **Same Topic**. It should cover the same topic as the doubtful statement.
- **Term sense/type closeness**. Each alter-unit should have close word sense with the DU, in terms of both data type and semantic meaning.

We turn the problem of finding alternative statements into the problem of looking for relevant alter-units. To find relevant alter-units, we partition DS into two parts: *doubt unit* and *topic units*. The topic units contain content words in DS except the doubt unit. The topic units provide the appropriate context for finding relevant alter-units. We use the topic units to form a query (called *topic query*) and submit it to a search engine (T-verifier currently uses Yahoo!) to retrieve a set of SRRs. Each SRR consists of a title and a snippet. Let D be the set of N SRRs collected. Every term (including phrase) in these SRRs is considered a candidate alter-unit and we rank all candidates using seven features to select the alter-units from the top ranked candidates.

Our observation shows that there are two types of co-occurrence information with alter-units that can be used. First, relevant alter-units often co-occur with the topic units as the topic units provide the appropriate context for an alter-unit to be relevant. Second, relevant alter-units often co-occur with the doubt unit. This is because when people have doubt about a fact, they often mention other possible answers to the fact. Among the following seven features used in T-verifier, the first four capture the co-occurrence relationship between the topic units (i.e., the topic query Q) and a term T from SRRs (i.e., a candidate alter-unit) and the last three capture the relevance between a term T and the given doubt unit DU.

- **Result coverage (RC)**: It is the percentage of SRRs in D that contain T. If T appears in a higher percentage of the SRRs, it is more likely to be a good alter-unit. RC measures how frequently T co-occurs with the topic units.
- **Result Query Relevance (RQR)**: It is a measure of the relevance of the SRRs with respect to Q that contain T. If T appears in more relevant SRRs, T is more likely to be a good alter-unit. An SRR that contains more topic units is considered to be more relevant to Q.

- **SRR ranking (Rrank)**: It is a measure reflecting the ranks of the SRRs that contain T. Intuitively, if T appears in higher ranked SRRs, it is more likely to be a good alter-unit.

- **Term distance (TD)**: Intuitively, terms that appear closer to the topic units are more likely to be related to the topic units. To capture the proximity information between terms, we consider the size of the smallest window of consecutive words in each SRR (title and snippet are considered separately) that covers all the topic units contained in the SRR as well as T. The TD measure considers the length of the minimum window as well as the number of topic units covered in this window. When none of the topic units appears in the same sentence with T, we consider its TD for this SRR to be 0.

- **Data type matching (DM)**: Each valid alter-unit is required to have the same data type as the doubt unit in T-verifier. Several data types are supported in T-verifier, including date, time, telephone number, email address, person name, place name (e.g., name of state, city and attractions), and number. All others are considered as strings.

- **Sense closeness (SC)**: Intuitively, terms that have a closely related meaning as the doubt unit are good candidate alter-units. T-verifier utilizes WordNet to capture the sense closeness between two terms. Three special relations are considered in T-verifier (direct hypernym/hyponym, instance hyponym, and sibling) and they are treated differently when sense closeness is calculated [4].

- **Term local correlation (TLC)**: It measures the correlation between T and DU. As mentioned earlier, relevant alter-units are often mentioned together with the doubt unit. T-verifier uses the correlation coefficient formula in [9] to measure the correlation strength between T and DU.

T-verifier selects alter-units by combining the seven features described above in two steps. It first filters the candidate terms by data type matching (DM), i.e., only those terms that match the data type of the DU will be considered further. In the second step, it ranks each remaining candidate term using the other six features. The ranking score of T is a weighted sum of the scores based on these six features. The best values for these weights can be determined empirically. Note that if DU is one of the terms in the retrieved SRRs, it will participate in the ranking just like other terms in these SRRs.

As an example, for the doubtful statement "Barack Obama is a Muslim" with DU = "Muslim", the top-5 ranked alter-units found by the above method are "christian", "muslim", "president", "black", and "senator". Note that "president" is the alter-unit with the highest result coverage, however, it is not the top-ranked alter-unit when other features are considered (it has low sense closeness with "Muslim").

According to our experiment, the above alter-unit ranking method performs well on our test dataset. For each of the 50 sample statements tested, this method always ranks the truthful alter-unit among the top five results. Furthermore, in 31 cases, the truthful alter-unit is ranked at the top. Each of the top-5 ranked alter-units is used to generate one alternative statement by taking the place of the doubt unit in the doubtful statement. As a result, we only need to consider these five alternative statements in the statement truthfulness verification phase for each doubtful statement. Note that DU may or may not be one of the top-5 alter-units depending on whether it appears in the retrieved SRRs and what its ranking score is.

3 Alternative Statements Truthfulness Verification

In T-verifier, the verification phase is carried out by ranking the top five alternative statements generated in the previous phase and selecting the top-ranked statement as the truthful one. The phase described in Section 2 does provide a ranking of all the alter-units which can be regarded as the ranking of the corresponding alternative statements. Unfortunately this ranking is not sufficiently accurate for practical use as it ranks only 62% of the truthful alter-units at the top among the 50 doubtful statements we tested. Thus a better solution is needed. In this section, we introduce the truthfulness verification process for a given group of alternative statements as used

in T-verifier. This process has three steps: first, send every alternative statement as a query to a search engine and collect relevant SRRs; second, employ a number of basic rankers and use each of them to generate a ranking list of the alternative statements based on the newly collected SRRs; third, use a rank merging algorithm to merge the rank lists into a combined list.

Intuitively, verifying the truthfulness of a statement requires relevant evidence. For each alternative statement, T-verifier submits it as a query to a search engine to retrieve the top 200 SRRs and uses the information from these SRRs to rank the alternative statements. An implicit assumption underpinning this approach is that truthful information is usually more widespread than untruthful ones on the Web and tends to be consistent.

The Basic Rankers

Given a set of alternative statements and the retrieved SRRs, the basic rankers used in T-verifier to rank the alternative statements are introduced below.

- **Alter-Unit Ranker (AUR)**: In Section 2, we discussed a method for selecting alter-units from the SRRs retrieved by the topic query. The alter-units are ranked based on a number of features collected from SRRs. Since each alter-unit corresponds to an alternative statement, this ranking of the alter-units from this method can be considered as a ranking of the alternative statements, and we call it Alter-Unit Ranker (AUR).

- **Hits Ranker (HR)**: A seemingly reasonable method is to rank the alternative statements by the number of hits they retrieve from a search engine. Web users often use this method to do quick truthfulness verification. We call this method the Hits Ranker. One potential problem of this ranker is that it implicitly assumes that all the SRRs retrieved by an alternative statement support this alternative statement. However, it is possible that some of the SRRs are actually against it (i.e., saying it is not true). As a result, the number of hits may not be a reliable indicator for judging the truthfulness of an alternative statement.

- **Text Feature Rankers (TFR)**: Text feature rankers are a set of 4 rankers measuring the relevance between the alternative statements and each SRR. In Section 2, we discussed several text features used to measure the relevance between an SRR and the topic query. There are mainly four features involved: Result Coverage (RC), Result Query Relevance (RQR), SRR ranking (Rrank), and Term Distance (TD). In TFR, we reuse these features to rank the alternative statements. However, the rankings generated by TFR here are different from those by the alter-unit generation phase because these four features are now used against a different set of SRRs, which is retrieved by a different query (i.e., the full alternative statement under consideration).

- **Domain Authority Ranker (DAR)**: Some researchers have observed that web pages published by certain domains are more likely to be truthful, such as ".gov", ".edu", etc [2]. This is because websites in these domains claim responsibility for their published materials. Based on this observation, a higher weight is assigned to a domain that is considered to be more trustworthy. In T-verifier, the weight for each domain is learned from the SRRs in this domain that are retrieved by sample truthful and untruthful alternative statements [4]. The rank position given by DAR for an alternative statement is determined by the number of SRRs from each domain retrieved by the alternative statement and the weights of these domains.

Different basic rankers may produce different rankings of the alternative statements. Table 1 shows the ranks of five alternative statements corresponding to the alter-units listed in the first column by the seven basic rankers. Thus, a method is needed to combine the different rankings into a single overall ranking. Based on the evaluation result of several rank merging techniques (including a probabilistic method, a machine learning method, several variations of the Borda method and several variations of the Condorcet method (see [4] for more details)), a variation of the Borda method called Weighted Positional Borda is found to have the best performance and is used in T-verifier. This method is described below.

Table 1: Ranks of Alternative Statements by Basic Rankers

Alter-units	AUR	HR	RC	RQR	Rrank	TD	DAR
christian	1	3	1	1	1	1	2
muslim	2	2	2	3	2	2	1
president	3	1	3	2	3	5	3
black	4	4	4	4	4	3	5
senator	5	5	5	5	5	4	4

The basic Borda algorithm [1] works as follows. Suppose *n* candidates (they are the top ranked alternative statements in our case) are being ranked. If an alternative statement is ranked at the *i*-th position, it receives a score of $n-i+1$, $1 \leq i \leq n$. The combined score of an alternative statement is the sum of the scores it receives from all the basic rankers. Finally, the alternative statements are ranked in descending order of the combined scores. One weakness of this algorithm is that it treats all basic rankers equally, ignoring the fact that some basic rankers may be better than others.

One way to improve the basic Borda algorithm is to differentiate the importance of different basic rankers and assign a weight to each ranker, as was also done in [1]. In T-verifier, the weight for each basic ranker is obtained through training [4]. After the weight for a basic ranker BR is obtained, the ranking score an alternative statement receives from BR in the basic Borda algorithm is revised by multiplying the weight of BR. Specifically, if the weight of BR is w, then the alternative statement ranked at the *i*-th position by BR will receive a score of $(n-i+1)*w$. This version of the Borda algorithm is called the Weighted Borda algorithm [4].

Another weakness of the basic Borda algorithm (as well as that of the Weighted Borda algorithm) is the way it assigns a score to each ranking position. While it is reasonable to assign a larger score to a higher position (the top-ranked result has the highest position), there is little scientific justification on why each next lower position should get exactly one point less than the position above. In T-verifier, the score assigned to each position by a basic ranker is the estimated probability that an alternative statement assigned to this position by this basic ranker is truthful. This probability is called the *position probability* of the basic ranker and it can be estimated from training i.e., from ranking the alternative statements of a set of sample doubtful statements with known truthful answers [4]. This variation of the Borda algorithm is called the Positional Borda in [4].

Finally, the Weighted Positional Borda algorithm used in T-verifier is a combination of the Weighted Borda algorithm and the Positional Borda algorithm. For a basic ranker BR with weight w, the final score received by its *i*-th ranked alternative statement is $p_i * w$, where p_i is BRs position probability for the alternative statement ranked at position *i*. Once again, the combined score of each alternative statement is the sum of the final scores it receives from all the basic rankers. The alternative statement with the highest combined score is selected as the truthful statement for the given doubtful statement.

For the ranks by individual basic rankers shown in Table 1, all three Borda-based rank merging algorithms rank "Barack Obama is a Christian" at the top among the five alternative statements.

4 Experimental Evaluation

T-verifier was evaluated using 50 doubtful statements [2] transformed from 50 factoid questions from TREC-8 and TREC-9 Question Answering track [10]. Half of these statements are truthful while the other half are untruthful. The answer part in each statement is specified as the doubt unit. The proposed methods described in Sections 2 and 3 are used to determine the truthfulness of each statement and compare the result with the ground truth to compute the precision of the proposed methods.

To evaluate the alternative statement generation method described in Section 2, half of the 50 doubtful statements are randomly selected as training set and the remaining half as testing set. The method is performed on the top 200 SRRs for each statement. The results in Table 2 show that the method described in Section

[2]This dataset is available at http://cs.binghamton.edu/∼xianli/doubtful_statements.htm.

Table 2: Performance of Alternative Statements Generation

Training dataset		Testing dataset	
Total cases	25	Total cases	25
Truthful one as top 1st	17	Truthful one as top 1st	14
Truthful one as top 2nd	7	Truthful one as top 2nd	9
Truthful one as top 3rd	0	Truthful one as top 3rd	1
Truthful one as top 4th	1	Truthful one as top 4th	1
Truthful one as top 5th	0	Truthful one as top 5th	0

2 can always rank all truthful alternative statements among the top 4 candidates for our dataset. A 10-fold cross-validation yielded similar result [4].

Based on the result on alterative statements generation, it is sufficient to compare the top-5 alternative statements to determine which one is most likely to be truthful and consequently decide whether the doubtful statement is truthful. The precisions of the seven basic rankers introduced in Section 3 vary from 0.2 to 0.66 (AUR 0.62, HR 0.2, TFR(TD) 0.32, TFR(RC) 0.66, TFR(RQR) 0.6, TFR(Rrank) 0.62, and DAR 0.2) based on the 50 doubtful statements. Overall, the precisions of individual basic rankers are not very good. The hit ranker and the domain authority result ranker performed especially poorly for our dataset. Combining ranks from the basic rankers using the Weighted Positional Borda algorithm as employed by T-verifier yields a precision of 0.9 (or 0.94 if multiple correct answers are allowed for a doubtful statement) based on 10-fold cross-validation.

5 Related Works

The "Honto?Search" system [11, 12] is another system aimed at verifying the truthfulness of fact statements. The basic idea of this system is similar to that of T-verifier in that it looks for alternative statements via a search engine and finds most likely truthful one from them. The method is based on hits numbers, temporal information (i.e., when a page was published) [11] and sentimental factors [12].

Fact statement truthfulness verification discussed in this article is related to answer verification in question-answering (QA) systems. The method proposed in [5, 6] also uses co-occurrence information between the keywords in a question and each extracted candidate answer to rank different candidate answers. To the best of our knowledge, very few papers address the answer verification problem directly, although many systems incorporate answer verification into their candidate answer ranking component. The ranking component scores candidate answers based on certain features [8, 13], and considers the answer with the highest score as the most likely correct one [3].

T-verifier differs from the existing solutions in the following aspects. (1) T-verifier operates in two carefully designed phases (i.e., alternative statement generation and statement truthfulness verification) which explore different sets of features, including new features not used before such as the semantic closeness between the doubt unit and each alter-unit. None of the existing solutions used all of these features in a single solution and used them like in T-verifier. None of them has studied both phases as comprehensively as in T-verifier. In fact, most of them studied only one of these phases. (2) Our alternative statement generation method is able to rank the truthful alter-unit among the top 5 results in our experiment. To the best of our knowledge, similar results have not been reported before. (3) Based on seven basic rankers and the Weighted Positional Borda rank merging method, T-verifier is able to achieve a precision of about 90%.

Yahoo!Answers (http://answers.yahoo.com/) and answers.com (http://www.answers.com/) are two Web QA systems based on completely different technology. They ask users to post questions, provide answers to any question, and vote for existing answers. This type of systems will not be able to provide answers to questions that have not been posted and answered. Furthermore, for answers with few votes, the accuracy is likely to be low. We tested the 50 doubtful statements in our dataset on these two systems in May 2011. For answers.com, there

were no answer for 6 statements and for 4 additional statements, the answers were wrong. For Yahoo!Answers, there were no answer for 18 statements and for 12 additional statements, the answers were wrong.

It is worth noting that fact statements as studied in this article and questions in QA systems have significant differences. First, there is a specified doubt unit in each doubtful statement which enables certain features to be utilized such as the sense of the doubt unit. In contrast, no specific doubt unit is given in questions. Second, it is possible to expand the doubtful statements to allow multiple doubt units to be specified (see Section 6). Questions in QA systems, on the other hand, can target only one doubt at a time. Third, many fact statements cannot be easily converted to WH-questions used in QA systems. These differences lead to different solutions for the two types of systems (truthfulness verification and QA).

6 Future Directions

The work introduced in this article can be extended in a number of ways to either handle more general doubtful statements or to make the technique more useful to web users. Some ideas are introduced below.

Process doubtful statements that have multiple truthful results: For many doubtful statements, the truthful alter-unit is not unique. For example, since time-sensitive statements often have different truthful results at different times, they can be considered as a special case of doubtful statements with multiple truthful results. Many time-insensitive statements may also have multiple results. For example, doubtful statement "Barack Obama was born in [Kenya]" has multiple truthful units: "Honolulu", "Hawaii", "Honolulu, Hawaii", "the United States", etc. Challenging issues here include how to identify doubtful statements that have multiple truthful results and how to find all the truthful results accurately for such statements.

Allow multiple doubt units in a doubtful statement: Sometimes a user may specify multiple doubt units in a doubtful statement. For example, in "President Nixon visited [South Africa] in [1970]", two doubt units "South Africa" and "1970" are specified. An interesting issue is how to process such type of doubtful statements. Clearly processing each doubt unit independently will not ensure the generation of statements that are truthful on all doubt units. For this type of doubtful statements, we need to select the alter-units for different doubt units in synchronization but also deal with the possibility that multiple combinations of alter-units are truthful.

Process doubtful statements that have no specified doubt unit: Such statements may either be entered by users or extracted directly from web pages. The ability to determine the truthfulness of any fact statement without a pre-specified doubt unit can open the door for examining claims made in web pages automatically. A naive method is to treat each content word or phrase in such a statement as a doubt unit, one at a time, and process the corresponding doubtful statement using the technique introduced in this article. We can also generate doubtful statements by treating combinations of content words and/or phrases as doubt units. While this method may work, it is not at all efficient, especially for long fact statements as the number of content words/phrases and their combinations can be quite large. An interesting issue is how to automatically and efficiently identify worthy doubt unit(s) in any given fact statement using the data on the Web?

Determine the truthfulness of fact statements on the Web: It would be nice if the truthfulness of every fact statement on the Web can be verified (or corrected) as this can benefit all web users. The problem is that the number of fact statements that exist in web pages is huge and new fact statements will keep popping up at rapid speed, and the verification process described in this article is complex. As a result, it is not realistic to use the described method to process all fact statements independently. What we need is an automatic, effective and scalable solution. One idea is to focus only on interesting fact statements, rather than all fact statements. How to identify all interesting fact statements is an open research problem. One type of interesting fact statements is *controversial fact statements*. A fact statement FS is a controversial fact statement if there exists another fact statement FS* on the Web such that FS and FS* state contradictory facts. Another idea is to save all fact statements whose truthfulness or untruthfulness has been verified and then use these statements to determine the truthfulness of new fact statement whenever possible before trying the more expensive method.

Estimate the trustworthiness of web pages/websites: The ability to determine the truthfulness of fact statements opens up a new way to estimate the trustworthiness of web pages and websites. Specifically, given a web page or a website, we can estimate its trustworthiness by estimating the percentage of the fact statements in the web page or websites that are truthful. An interesting issue here is how to take into consideration the situations where different fact statements and different types of "errors" (e.g., typo vs. obsolete information) may contribute to trustworthiness differently? Another issue is how to combine different trust models into an integrated model? More specifically, there are other techniques to estimate the trustworthiness of web pages and websites (such as using PageRank [7]/TrustRank [2] and considering the domains, e.g., documents/sites in the .edu domain are often considered to be more trustworthy than those in the .com domain [2]). We need to figure out how to combine these existing trust models with the statement truthfulness based model.

7 Summary

In this article, we introduced a method to analyze the truthfulness of fact statements with a specified doubt unit using the data on the Web. This method consists of two main phases, with the first trying to identify the most likely relevant alternative statements and the second trying to rank them. The T-verifier system implemented based on the introduced method has an accuracy of about 90% according to our experimental evaluation. We also introduced several future research directions which can lead to a system capable of processing more general fact statements and to applications that can improve the chance for ordinary users to obtain more reliable and trustworthy information on the Web.

Acknowledgements

This work is supported in part by the following NSF grants: IIS-0842608, IIS-0842546 and CNS-0958501.

References

[1] J. A. Aslam, M. H. Montague: Models for Metasearch. ACM SIGIR Conference, pp.275-284, 2001.

[2] Z. Gyngyi, H. Garcia-Molina, J. Pedersen: Combating Web Spam with TrustRank. International Conference on Very Large Data Bases (VLDB), pp.576-587, 2004.

[3] C. T. Kwok, O. Etzioni, D. S. Weld: Scaling question answering to the web. In ACM Transactions on Information Systems, 19(3): 242-262, 2001.

[4] X. Li, W. Meng, C. Yu. T-verifier: Verifying Truthfulness of Fact Statements. IEEE ICDE Conference, pp.63-74, 2011.

[5] B. Magnini, M. Negri, R. Prevete, H. Tanev: Is It the Right Answer? Exploiting Web Redundancy for Answer Validation. ACL 2002: 425-432.

[6] B. Magnini, M. Negri, R. Prevete, H. Tanev: Mining Knowledge from Repeated Co-Occurrences: DIOGENE at TREC 2002. TREC 2002.

[7] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bring Order to the Web. Technical Report, Department of Computer Science, Stanford University, 1998.

[8] X. Qiu, B. Li, C. Shen, L. Wu, X. Huang, Y. Zhou: FDUQA on TREC2007 QA Track. Sixteenth Text REtrieval Conference (TREC), 2007.

[9] C. Silverstein, M. R. Henzinger, H. Marais, M. Moricz: Analysis of a Very Large Web Search Engine Query Log. SIGIR Forum 33(1): 6-12, 1999.

[10] http://trec.nist.gov/data/qamain.html

[11] Y. Yamamoto, T. Tezuka, A. Jatowt, K. Tanaka: Honto?Search: Estimating Trustworthiness of Web Information by Search Results Aggregation and Temporal Analysis. APWeb/WAIM 2007: 253-264.

[12] Y. Yamamoto, T. Tezuka, A. Jatowt, K. Tanaka: Supporting Judgment of Fact Trustworthiness Considering Temporal and Sentimental Aspects. In WISE 2008: 206-220.

[13] Z. Zheng: AnswerBus Question Answering System. HLT 2002: 399-404.

Corroborating Information from Web Sources

Amélie Marian, Minji Wu

Department of Computer Science, Rutgers University, New Brunswick, NJ, USA

{amelie,minji-wu}@cs.rutgers.edu

Abstract

Information available on the Internet is abundant but often inaccurate. Web sources have different degrees of trustworthiness based on their accuracy, freshness, origin or bias. Web users are then left with the daunting task of assessing the correctness of possibly conflicting answers to their queries. In this paper, we present techniques for corroborating information from different web sources. We discuss techniques that estimates the truthfulness of answers and the trustworthiness of the sources based on an underlying probabilistic model. We show how to apply data corroboration to a web setting where data sources can have multiple forms, all with various quality issues: individual web sites, search engine query results, user reviews, map and street view data, and social tags.

1 Introduction

Traditional query processing techniques assume correctness of the information provided by an underlying data source (e.g., a database, a document collection). For many data needs, accessing a single trusted source of information is no longer possible; the information needs to be gathered from several, often completely independent sources. This is particularly true when accessing Web sources, which provide a variety of information and viewpoints with different degree of trustworthiness based on the sources' origin or bias. When different sources provide conflicting or incomplete information, guaranteeing good query answer quality can be challenging. The most daunting problem when trying to answer a question seems not to be *where* to find an answer, but *which* answer to trust among the ones reported by different Web sources. This happens not only when no true answer exists, because of some opinion or context differences, but also when one or more true answers are expected. Such conflicting answers can arise from disagreement, outdated information, or simple errors.

Using corroborative evidence, in the form of similar answers from several sources, is an intuitive way to increase trust in the returned answers. The use of data corroboration can significantly improve query answer quality in a wide variety of scenarios [14, 4, 7, 13, 8].

In this paper we discuss different approaches to corroborate data (Section 2). We show how data corroboration can improve data quality in a web source scenario, where data sources can have multiple forms, all with various quality issues: individual web sites, search engine query results, user reviews, map and street view data, and social tags (Section 3). Finally, we discuss open issues in data corroboration (Section 4).

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

2 Background

The problem of identifying the best outcome from multiple sources of information is not a new one. Voting theory and multi-criteria decision making have been subjects of research for centuries. Data corroboration is a natural way of dealing with conflicting information, and has been used in various aspects of society: justice, journalism, gossip. The more (independent) sources corroborate a fact, the more likely the fact is true, assuming the sources are trustworthy. Unfortunately, many web sources are not trustworthy, because of erroneous, misleading, biased, or outdated information.

In this section we describe some of the recent advances in data corroboration and comment on some related work.

Web Sources. In [13], we developed corroboration scoring techniques for web data in order to save users the hassle of manually checking query-related web sites for corroborated answers. The existence of several sources providing the same information is then viewed as corroborating evidence, increasing the quality of the corresponding information. We score each answer by taking into account the number, relevance, and originality of the sources reporting the answer as well as the prominence of the answer within the sources. Our results show that corroboration-based methods provide significant improvements in answer quality compared to simple frequency-based approaches.

`TRUTHFINDER` [14] assigns confidence to web sources by recursively computing confidence in the sources based on the expected truth of the facts they report. Presence of similar facts in different sources is then seen as positive reinforcement of the expected truth of the fact.

Probabilistic Model. A probabilistic data model for corroboration was introduced in [7] to take into account the uncertainty associated with facts reported by the sources, the possibility of conflicting information coming from separate sources, as well as the limited coverage of the sources. The model sets the bases for a systematic study of trust-based corroboration, where trust values are assigned to sources of information and used to infer truth of facts reported by the source. In addition, the probabilistic model was used as the basis for corroboration algorithms based on fixpoint computation techniques that derive estimates of the truth value of facts reported by a set of sources, as well as estimates of the quality of the sources. The resulting corroboration techniques consistently outperformed existing voting-based strategies.

Several theoretical works have focused on estimating the probability of an event in the presence of conflicting information. Osherson and Vardi [11] study the problem of inconsistent outcomes when aggregating logic statements from multiple sources. Their goal is to produce a logically coherent result. Work in subjective logic and trust management [9] consider the issue of trust propagation from one source to another, in a model where the sources are not independent.

Statistics. The expectation-maximization (EM) algorithm [3] was presented in 1977 to find the maximum likelihood estimates of parameters in a model containing latent variables. It was implemented [2] to take into account observer (source) errors to assess performance of individual observers and derive an agreement in classification tasks. More recently, it was adapted to estimate worker quality in Mechanical Turk experiments [8], and the derived quality used to generate agreements. A limitation of the EM algorithm is scalability, which makes it difficult to apply to a web source scenario.

Question Answering. Question answering systems, such as [1, 10, 6] consider the frequency of an extracted answer as a measure of answer quality. However, these techniques rely mostly on redundancy of information and do not consider the trust associated with each extraction source to score extracted answers.

3 Case Study: Identifying Good Business Listings

We now rely on the web to get details about the location and contact information of local businesses. While the old-fashioned paper-based yellow pages did have inaccuracies: places gone out business, new stores not yet included, several names, preferably starting with the letter "A," —an early spamming technique— for the same business; overall the information available was of high quality as businesses had to pay for their listing to appear and a human would typically check the listing before it went to print. In contrast it is easy and cheap to add information to the web through a web page, or by registering listings directly on services such as Google Places. This gives the opportunity to make more up-to-date and comprehensive listings available to the users, but the sheer number of businesses makes it impossible to verify the accuracy of each of these listings manually.

We consider the use of data corroboration techniques in a restaurant listings scenario, in which our goal is to assess whether a reported listing ((restaurant,address) pair) is true (correct) or false (incorrect). We consider a wide variety of data sources, and harness the power of the web 2.0 to improve data quality. Data corroboration allows us to (1) assess the accuracy of web business listings, and (2) assess the quality of the sources from which the listings are derived to identify sources of spam and determine the usefulness of each source's contribution.

We corroborate information from traditional web sources, user reviews and web 2.0 data (e.g., user check-ins, restaurant reservation portals, mechanical turkers assessments of the visibility of the business at the listing's address on Google Streetview). By themselves, none of these sources offer definite information as to whether the listing is correct or not. Similarly, the fact that some of these sources may not have information for a business does not mean that the listing is inaccurate. By corroborating information from different sources and considering the number of sources sharing similar information we can provide an estimated truth for each of the listings in the sample.

3.1 Data Model

We consider a probabilistic data model similar to that described in [7]. Sources of data can be seen as views over the real world W. Views report beliefs that are positive or negative statements. Based on these beliefs, our task is to "guess" what the correct values for the real world W are.

Let $\mathcal{F}$ be a set $\{f_1 \dots f_n\}$ of *facts*. A *view* (over $\mathcal{F}$) is a (partial) mapping from $\mathcal{F}$ to the set $\{T, F\}$ (T stands for *true*, and F for *false*). We consider a set of views $\mathcal{V} = \{V_1 \dots V_m\}$ from which we try to estimate the real world W, defined as a total mapping from $\mathcal{F}$ to the set $\{T, F\}$. From a mathematical viewpoint, based on some probabilistic model, we want to estimate the most likely W given the views. (Note that views can also report no belief for a given fact, in which case we consider the fact unknown in the source ($V_i(f_j) =?$).

We implemented the COSINE data corroboration strategy [7].COSINE is a heuristic approach, based on the classic cosine similarity measure that is popular in Information Retrieval, that estimates the truth values of facts and the trustworthiness of views with values between -1 and 1, where -1 means false facts, systematically wrong views, 0 means indeterminate facts, views with random statements, and 1 means true facts, perfect views. The idea is then to compute, for each view V_i, given a set of truth values for facts, the similarity between the statements of V_i, viewed as a set of ± 1 statements on facts, and the predicted real world.

3.2 Experimental Settings

We have extracted a sample of New York City restaurant listings from three web sources *YellowPages, CitySearch* and *MenuPages*, a total of 1,000 distinct listings. We then cross checked each of the three sources to identify whether listings extracted from the other two sources appeared in them. For each 1,000 listing, if it exists in a source, then the source reports it as true; if it does not, then it is considered unknown by the source.

We also considered four sources of data:

Source coverage	Yelp	Foursquare	OpenTable	M. Turk	YellowPages	CitySearch	MenuPages
	0.52	0.42	0.05	0.91	0.65	0.70	0.41
Source overlap	Yelp	Foursquare	OpenTable	M. Turk	YellowPages	CitySearch	MenuPages
Yelp	1	0.58	0.09	0.49	0.43	0.44	0.55
Foursquare	0.58	1	0.11	0.40	0.35	0.40	0.51
OpenTable	0.09	0.11	1	0.05	0.05	0.07	0.10
M. Turk	0.49	0.40	0.05	1	0.61	0.64	0.40
YellowPages	0.43	0.35	0.05	0.61	1	0.46	0.26
CitySearch	0.44	0.40	0.07	0.64	0.46	1	0.37
MenuPages	0.55	0.51	0.10	0.40	0.26	0.37	1

Table 3: Source coverage and overlap

	Yelp	Foursquare	OpenTable	M. Turk	YellowPages	CitySearch	MenuPages
Source Trust	0.98	0.99	0.99	0.86	0.87	0.89	0.81

Table 4: Corroborated trust in the sources

- *Yelp*, a user reviewing web site. If a listing has a review in *Yelp*, then the source reports it as a true fact. We did a quick analysis of the text of the reviews to identify closed restaurants and report those as false. Listings for which there are no *Yelp* entry are considered unknown.
- *Foursquare*, a user check-in service. If a restaurant has associated entries, then it is reported true by *Foursquare*, otherwise it is reported unknown.
- *OpenTable*, a restaurant reservation service. If a restaurant is available for reservations in OpenTable, then it is considered true, if not it is unknown.
- *Mechanical Turk*, we considered a crowdsourcing approach where mechanical turkers were asked to manually check Google Streetview to see if the restaurant was located at the address. Unfortunately, it is well documented that *Mechanical Turk* data tend to be of low quality [8]. To prevent skew in the result due to spamming done by MT workers, we asked 5 workers to evaluate each listing and use a majority vote to assign true, false or unknown values.[1]

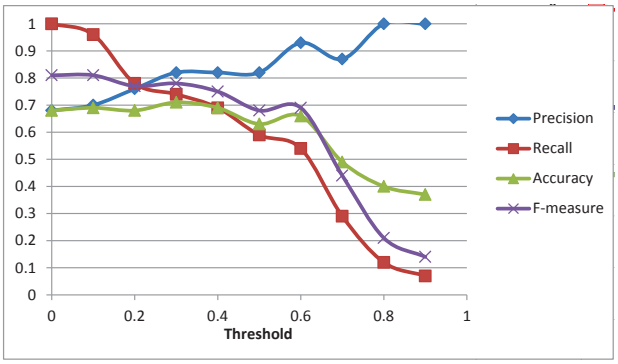
Table 3 reports the source coverage (the fraction of the sample that is contained in each source), as well as the source overlap (a measure of how much of the sample two sources have in common). All sources contain only a fraction of the domain. Considering several sources is then critical to have a large picture view of the data. *Mechanical Turk* has the largest coverage as it was specifically created by asking MT workers about each specific restaurant in the sample, nevertheless, 9% of the sample is not covered by this source as MT workers could not assess the presence of the restaurant at the specified address (missing Streetview, obstructed view, etc.). On the opposite, OpenTable only contains 5% of the sample as it is a commercial website that connects to real restaurant reservation systems.

3.3 Experimental Results

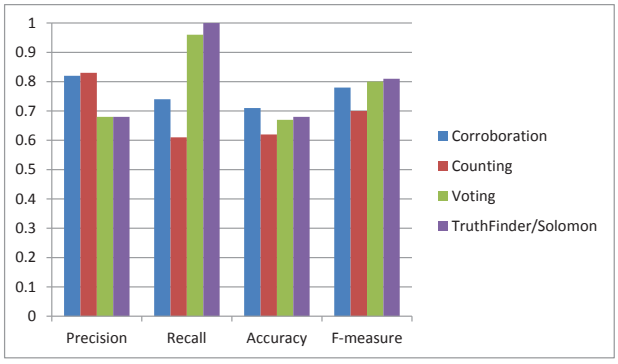
We corroborated our listing data using the COSINE algorithm presented in [7]. Table 4 shows the quality values for each source, as computed by the COSINE algorithm (trustworthiness value). Unsurprisingly, *OpenTable*,

[1]We integrated the EM-based algorithm from [8] to corroborate MT workers answers. Unfortunately, it did not perform as well as expected as our data set is highly skewed towards true listings, and our experiments had a majority of workers with few answers.

which connects to real restaurant reservation systems is of very high quality, but with low coverage (Table 3). The dynamic user-based sources *Foursquare* and *Yelp* also show high quality, whereas the directory sources *CitySearch*, *YellowPages* and *MenuPages* have lower quality, possibly because of stale results. *Mechanical Turk*'s quality is reasonable but not as good as hoped despite our postprocessing of its data through voting due to the low quality of individual workers.



(a) Precision, Recall, Accuracy and F-measure values when varying the corroboration threshold



(b) Comparison of the quality of the corroborated answers with that of standard voting techniques

Figure 1: Evaluation of the corroborated answers

We evaluate the quality of our corroborated answers using a "gold standard" test set of 100 restaurant for which we have identified the correct answer using a combination of *Mechanical Turk* experiments (with 10 workers) and manual evaluation. For each of the restaurant in the test set we compare its correct value with that reported by the corroborated approach.

We report on the Precision, Recall, Accuracy, computed as the fraction of correctly classified listings, and F-measure of the true facts (confirmed listings) varying the threshold truth value at which we classify listing as being true in Figure 1(a). Note that negative truth values are associated with negative facts (i.e., confidence that the restaurant is not at the listed address). Our sources mostly report positive facts, except for *Mechanical Turk* and *Yelp*, which reports a few restaurants as closed. As a result only eight listings have a final truth value lower than 0, and the Precision, Recall and Accuracy values for threshold values between -1 and 0 are equal to that reported at $threshold = 0$. As expected, the higher the threshold, the better the quality of the listings identified as true is (higher Precision), but more correct listings are classified as false (lower Recall). The highest Accuracy and F-measure values are found around a threshold of 0.3. (High F-measure values at threshold 0 and 0.1 are due to high Recall as almost all listing are classified as true.) In practice, this means that the results are more accurate when at least two or three sources report the listing as true with no dissenting view, as expected.

In Figure 1(b), we compare the corroborated results (at $threshold = 0.3$) with standard voting techniques: COUNTING , which assigns a listing as true if at least half the sources report it true, and VOTING , which assigns a true value to any listings that has more sources reporting it true than false. As expected VOTING has high Recall but low Precision, since any listing that is reported true by only one source is likely to be classified true. In contrast COUNTING has high Precision but low Recall since at least half of the sources have to report on a listing for it to be identified as true. We also compare our results with that of techniques that consider conflicting sources information to make truth assessments: `TRUTHFINDER` [14] and `SOLOMON` [4]. On our restaurant listing data set, both techniques report the same results. Interestingly, they return all listings as true; this mechanically results in perfect Recall and high F-measure values. We believe this is due to the fact that our restaurant listing sources report contain very few false values (only two sources, *Mechanical Turk* and *Yelp*, have false facts). `TRUTHFINDER` and `SOLOMON` will then not detect much conflicting information and will assign high trust levels to all sources, resulting in facts being considered true even if they are reported by only

one source. In contrast, COSINE will assign lower scores to facts that are reported by only a few sources. By varying COSINE 's threshold value, we can tune our corroboration to require more source evidence to report a listing as true. As a result, the corroborated COSINE approach has the best Accuracy (percentage of correctly classified listings) over the five techniques.

4 Open Issues

We showed that data corroboration can effectively be used to identify answers in the presence of multiple incomplete and possibly conflicting sources. This work opens the road to many interesting research challenges:

- **Multiple Answers:** So far, most of the work on data corroboration [7, 13, 14] assume that there is only one valid answer for each fact (e.g., a listing is correct or not, a historical figure has only one valid birth date). In many cases, facts can have multiple answers (e.g., a person can have multiple phone numbers). The presence of multiple answers raise several interesting challenges, such as how many answers to consider based on the expected distribution of answers? How to adapt trust in the sources when all possible answers may be correct? How to rank, and possibly aggregate similar answers?

- **Functional Dependencies:** Many query scenarios have underlying functional dependencies. For instance if an email address can only be associated with one person, any source reporting a mapping (john, js@gmail.com) is stating implicitly that all other mappings ($person, js@gmail.com) are false. A simple way to model this would be to add a false statement for the mapping to js@gmail.com for every $person. This is not practical and creates a blowup of the number of facts. In addition, more complex functional dependencies are not simple to address, e.g., each person can only have one father and one mother. Integrating complex functional dependencies in a data corroboration model such as the probabilistic model of [7] raises interesting theoretical modeling issues.

- **Uncertain Data:** In addition to the trustworthiness of the source, we can also include another source of approximation in the model: uncertainty of the source. Sources providing results of ranking queries, belief databases, probabilistic databases are example of sources that report fact with an associated degree of confidence. How to integrate the uncertainty of the data, as reported by the source is also an interesting avenue to extend the corroboration model.

- **Domains:** In many cases, sources are specialized and the quality of a source should be assessed with respect to specific domains. This could be used for source selection during corroboration focusing on sources that have high quality rather than considering all possible sources [12] .

- **Time:** One of the main reasons for errors in web sources is due to outdated information (e.g., a restaurant has closed). We could leverage information from sources with timestamp data (e.g., user checkins, user reviews) in the corroboration, giving more weight to the "freshest" sources [5].

- **Social Network Trust:** Users originating queries may have a personal bias that they wish the corroboration to take into account. This could be preference towards sources sharing the same political views, or more trust in our friends beliefs. Including bias in the truth assessment can be done in several ways: by using biased facts as a training set of a recursive corroboration algorithm, or by heavily weighting trusted sources. A study of the tuning and impact of such bias may lead to interesting insights on how information propagates.

- **Source Dependence:** Recent work has focused on finding interdependencies, due to copying, between the sources [5, 4]. Ignoring such dependencies in corroboration can result in assigning more weight to sources

that were heavily copied, regardless of their quality. Combining copying detection and corroboration is a promising direction for improving data quality.

References

[1] E. Brill, S. Dumais, and M. Banko. An analysis of the AskMSR question-answering system. In *Proc. of the ACL conference on Empirical methods in natural language processing (EMNLP'02)*, 2002.

[2] A. P. Dawid and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied Statistics*, 28(1):20–28, 1979.

[3] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[4] X. Dong, L. Berti-Equille, and D. Srivastava. Integrating conflicting data: The role of source dependence. In *Proc. VLDB*, Lyon, France, 2009.

[5] X. Dong, L. Berti-Equille, and D. Srivastava. Truth discovery and copying detection in a dynamic world. In *Proc. VLDB*, Lyon, France, 2009.

[6] D. Downey, O. Etzioni, and S. Soderland. A probabilistic model of redundancy in information extraction. In *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 2005.

[7] A. Galland, S. Abiteboul, A. Marian, and P. Senellart. Corroborating information from disagreeing views. In *Proc. of the Third ACM International Conference on Web Search and Data Mining (WSDM'10)*, 2010.

[8] P. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *Proc. of the Second Human Computation Workshop (KDD-HCOMP 2010)*, 2010.

[9] A. Jøsang, S. Marsh, and S. Pope. Exploring different types of trust propagation. In *Proc. Trust Management*, Pisa, Italy, May 2006.

[10] C. C. T. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. In *Proc. of the 10th International Conference on the World Wide Web (WWW'01)*, 2001.

[11] D. Osherson and M. Y. Vardi. Aggregating disparate estimates of chance. *Games and Economic Behavior*, 56(1):148–173, July 2006.

[12] A. D. Sarma, X. L. Dong, and A. Halevy. Data integration with dependent sources. In *Procòf the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, 2011.

[13] M. Wu and A. Marian. A framework for corroborating answers from multiple web sources. *Information Systems*, 36(2), 2011.

[14] X. Yin, J. Han, and P. S. Yu. Truth discovery with multiple conflicting information providers on the web. In *Proc. of the 13th ACM International Conference on Knowledge Discovery and Data Mining (KDD'07)*, 2007.

Eliminating NULLs with Subsumption and Complementation

Jens Bleiholder
Opitz Consulting Berlin GmbH, Berlin, Germany
jens.bleiholder@opitz-consulting.com

Melanie Herschel
University of Tübingen, Germany
melanie.herschel@uni-tuebingen.de

Felix Naumann
Hasso Plattner Institute, Potsdam, Germany
naumann@hpi.uni-potsdam.de

Abstract

*In a data integration process, an important step after schema matching and duplicate detection is data fusion. It is concerned with the combination or merging of different representations of one real-world object into a single, consistent representation. In order to solve potential data conflicts, many different conflict resolution strategies can be applied. In particular, some representations might contain missing values (*NULL*-values) where others provide a non-*NULL*-value. A common strategy to handle such* NULL*-values, is to replace them with the existing values from other representations. Thus, the conciseness of the representation is increased without losing information.*

Two examples for relational operators that implement such a strategy are minimum union *and* complement union *and their unary building blocks* subsumption *and* complementation*. In this paper, we define and motivate the use of these operators in data integration, consider them as database primitives, and show how to perform optimization of query plans in presence of subsumption and complementation with rule-based plan transformations.*

1 Data Fusion as Part of Data Integration

Data integration can be seen as a three-step process consisting of *schema matching*, *duplicate detection* and *data fusion*. Schema matching is concerned with the resolution of schematic conflicts, for instance through schema matching and schema mapping techniques. Next, duplicate detection is concerned with resolving conflicts at object level, in particular detecting two (or more!) representations of same real-world objects, called duplicates. For instance, considering two data sources describing persons, schema matching determines that the concatenation of the attributes *firstname* and *lastname* in Source 1 is semantically equivalent to the attribute *name* in Source 2. Duplicate detection then recognizes that the entry *John M. Smith* in Source 1 represents the same person as the entry *J. M. Smith* in Source 2.

This article focuses on the step that succeeds both schema matching and duplicate detection, namely data fusion. This final step combines different representations of the same real-world object (previously identified

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Operator	Definition
Subsumption	A tuple t_1 *subsumes* another tuple t_2 ($t_1 \sqsupset t_2$), if (1) t_1 and t_2 have the same schema, (2) t_2 contains more NULL values than t_1, and (3) t_2 coincides in all NON-NULL attribute values with t_1 [7]. The unary subsumption operator β removes subsumed tuples from a relation R, i.e., $\beta\ (R) = \{t \in R : \neg\exists t' \in R : t' \sqsupset t\}$.
Complementation	A tuple t_1 complements a tuple t_2 ($t_1 \gtrdot t_2$) if (1) t_1 and t_2 have the same schema, (2) values of corresponding attribute in t_1 and t_2 are either equal or one of them is NULL , (3) t_1 and t_2 are neither equal nor do they subsume one another, and (4) t_1 and t_2 have at least one attribute value combination in common. A complementing set CS of tuples of a relation is a subset of tuples from R's extension where for each pair t_i, t_j of tuples from CS it holds that $t_i \gtrdot t_j$. A complementing set S_1 is a *maximal complementing set* (MCS) if there exist no other complementing set S_2 s.t. $S_1 \subset S_2$. We define the unary complementation operator κ to denote the replacement of all existing maximal complementing sets in a relation R by the complement of the contained tuples.
Outer, minimum, and complement union	The *outer union* operator ($\uplus$) combines two relations $R_1 \uplus R_2$ with respective schemas S_1 and S_2 and extensions T_1 and T_2 by (i) constructing the result schema $S = S_1 \cup S_2$ and (ii) combining the two extensions to $T = T_1 \cup T_2$, where missing values are padded with $\perp$. The *minimum union* operator ($\oplus$) is defined as $A \oplus B = \beta\ (A \uplus B)$. The *complement union* operator ($\boxplus$) is defined as $A \boxplus B = \kappa\ (A \uplus B)$.

Table 5: Definitions for subsumption, complementation, outer union, minimum union, complement union [2]

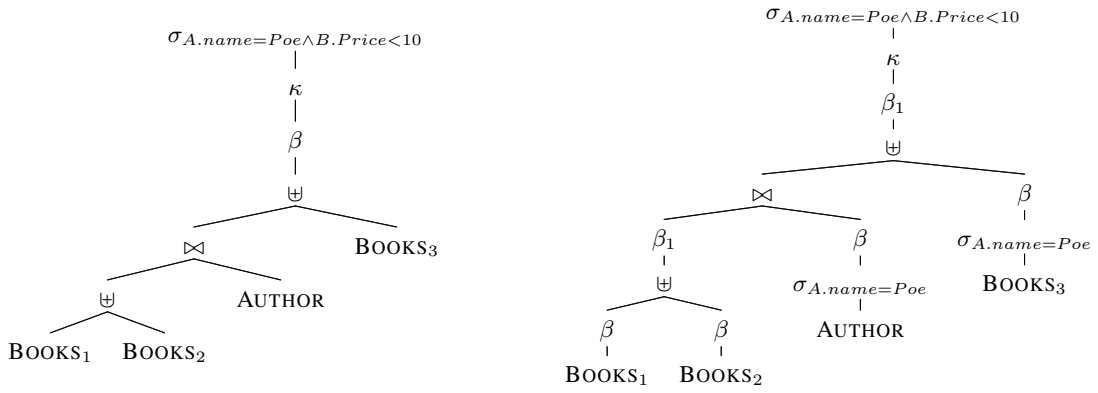
during duplicate detection) into one single consistent representation. To do so, fusion has to resolve any conflicts existing among the duplicates. The data fusion result is a potentially more concise and more complete representation of a real-world object than the representations obtained from the individual sources.

We distinguish two types of conflicts between attribute values, namely *uncertainties* and *contradictions*. We qualify conflicts as uncertainties when all duplicates agree in an attribute value and at least one does not specify any value, i.e., contains a NULL value for the given attribute. Contradictions, on the other hand, exist between duplicates if they each do provide a value, albeit different. For instance, consider the two (out of five) representations of the CD "Comme si de Rien n'Etait" by Carla Bruni shown in Fig. 1. We observe that data concerning the music label or price information are only provided in one representation, hence, we qualify this as an uncertainty among the two representations. Opposed to that, both provide an ASIN value, but the values differ. Clearly, these values contradict each other.

Figure 1: CD representations when searching for "Carla Bruni rien" on Amazon.com (January, 9th 2009)

There are many different strategies to handle conflicting data as surveyed in [1]. This paper focuses on resolving uncertainties using new techniques that follow the TAKE THE INFORMATION resolution strategy, which takes existing information and leaves aside NULL values [7, 8]. Our techniques stand out by being defined as database primitives. We previously defined relational operators and presented algorithms implementing them [2, 3] and repeat relevant definitions in Tab. 5.

The main contribution of this paper is a description of how these operators are used within relational query



(a) Plan A (b) Plan B

Figure 2: Two query plans for a sample data fusion process

planning. More specifically, we present transformation rules for *minimum union* ($\oplus$) and *complement union* ($\boxplus$) and their building blocks *outer union* ($\uplus$) *subsumption* (β) and *complementation* (κ) [2, 3] in Sec. 2. Further, we show cost and selectivity estimates for use within a cost-based optimizer in Sec. 3 and some experimental results in Sec. 4. Sec. 5 briefly discusses related work and concludes.

2 Transformation Rules for Data Fusion Queries

To illustrate transformation rules, consider the following bookstore scenario: one bookstore stores its fiction books in relation BOOKS$_1$ and its non-fiction books in relation BOOKS$_2$. As a book is either fiction or non-fiction, there is no overlap between the relations, however, duplicate entries in one relation are still possible. Authors are stored in an additional relation AUTHOR. A second bookstore stores its books (authors included) in relation BOOKS$_3$. When integrating data from both bookstores, assume we are interested in (i) removing subsumed tuples, (ii) combining complementing tuples, and (iii) only selecting books written by Poe and that cost less than 10 EUR. This task can be expressed by a *data fusion query* and visualized by a query plan.

Fig. 2 depicts two query plans that express the above task. Plan A first unifies fiction and non-fiction books using *outer union* and then combines these with their authors. Next, the books from the second store are added. Uncertainties are then handled by subsequently applying *subsumption* and *complementation*. Finally, the desired *selection* is applied. In contrast, Plan B pushes *selection*s as far down as possible and removes subsumed tuples early in the process. Note that we distinguish two types of subsumption operators: β_1 is used when the input relation can be partitioned such that each partition does not contain subsumed tuples, which is the case when data without subsuming tuples has been previously combined, whereas β does not make use of such knowledge.

We summarize the query rewriting rules we have shown to be correct for moving the *subsumption* and the *complementation* operators around in query plans in Tab. 6. *Minimum union* and *complement union* can be moved around in a tree by splitting the operators into their individual components (*subsumption*/*complementation* and *outer union*) and moving these components separately.

Combinations with Outer Union: As there may be subsuming tuples across sources, simply pushing *subsumption* through *outer union*s is not possible without leaving an additional *subsumption* operation on top of the *outer union* (Rules S1 and S2). As mentioned above, β_1 denotes the *subsumption* operator that only tests for subsumed tuples across subsumption-free partitions of its input. Using a similar technique for *complementation*, i.e., a corresponding κ_1, is not possible, because *complementation* and *outer union* are not exchangeable in general. This is due to the fact that tuple complementation is not a transitive relationship (Rules C1 and C2).

Combinations with Projection: For *subsumption*, potentially all attributes are needed to decide if one tuple subsumes another. So, in general, it is not possible to introduce arbitrary *projection*s below a *subsumption*

Combinations with *outer union*	
C1	$\kappa\ (\text{A} \uplus \text{B}) = \kappa\ (\text{A}) \uplus \kappa\ (\text{B})$, if there are no complementing tuples across sources, and
C2	$\kappa\ (\text{A} \uplus \text{B}) \neq \kappa\ (\text{A}) \uplus \kappa\ (\text{B})$, in all other cases
Combinations with *projection*	
C3	$\kappa\ (\text{A}) \neq \kappa\ (\pi\ (\text{A}))$, and
C4	$\pi\ (\kappa\ (\text{A})) \neq \kappa\ (\pi\ (\text{A}))$
Combinations with *selection*	
C5	$\kappa\ (\sigma_c\ (\text{A})) \neq \sigma_c\ (\kappa\ (\text{A}))$, if c is of the following form: x `IS NULL`, x `IS NOT NULL`, or x <op> y with x,y being attributes and <op> being one of $\{=, \neq\}$, and
C6	$\sigma_c\ (\kappa\ (\sigma_{c \vee cnull}\ (\text{A}))) = \sigma_c\ (\kappa\ (\text{A}))$, in all other cases, with *cnull* being the additional test for NULL values for all involved columns
Combinations with *cartesian product* and *join*	
C7	$\kappa\ (\text{A} \times \text{B}) = \kappa\ (\text{A}) \times \kappa\ (\text{B})$, if the base relations A and B do not contain subsumed tuples, and
C8	$\kappa\ (\text{A} \times \text{B}) = \kappa\ (\kappa\ (\text{A}) \times \kappa\ (\text{B}))$, in all other cases
Combinations with *grouping* and *aggregation*	
C9	$\gamma_{f(c)}\ (\kappa\ (\text{A})) = \gamma_{f(c)}\ (\text{A})$, for any column c and aggregation function $f \in \{\max, \min\}$
C10	$\kappa\ (\gamma_{f(c)}\ (\text{A})) = \gamma_{f(c)}\ (\text{A})$, for column c and any f
C11	$\gamma_{c,f(d)}\ (\kappa\ (\text{A})) = \gamma_{c,f(d)}\ (\text{A})$, for columns c, d with c as grouping column not containing NULL values and aggregation function $f \in \{\max, \min\}$
C12	$\kappa\ (\gamma_{c,f(d)}\ (\text{A})) = \gamma_{c,f(d)}\ (\text{A})$, for columns c, d with c as grouping column not containing NULL values and aggregation function $f \in \{\max, \min\}$
Other combinations	
C13	$\tau\ (\kappa\ (\text{A})) \neq \kappa\ (\tau\ (\text{A}))$
C14	$\delta\ (\kappa\ (\text{A})) = \kappa\ (\delta\ (\text{A}))$
C15	$\kappa\ (\kappa\ (\text{A})) = \kappa\ (\text{A})$
C16	$\kappa\ (\text{A}) = \text{A}$, if A has only one or two attributes

(a) Rules for *complementation*

Combinations with *outer union*	
S1	$\beta\ (\text{A} \uplus \text{B}) = \beta\ (\text{A}) \uplus \beta\ (\text{B})$, if there are no subsumed tuples across source relations, and
S2	$\beta\ (\text{A} \uplus \text{B}) = \beta_1\ (\beta\ (\text{A}) \uplus \beta\ (\text{B}))$, in all other cases.
Combinations with *projection*	
S3	$\beta\ (\text{A}) \neq \beta\ (\pi\ (\text{A}))$, and
S4	$\pi\ (\beta\ (\text{A})) \neq \beta\ (\pi\ (\text{A}))$
Combinations with *selection*	
S5	$\beta\ (\sigma_c\ (\text{A})) \neq \sigma_c\ (\beta\ (\text{A}))$, if c is of the following form: x `IS NULL`, with x being an attribute with NULL values, and
S6	$\beta\ (\sigma_c\ (\text{A})) = \sigma_c\ (\beta\ (\text{A}))$, in all other cases
Combinations with *cartesian product* and *join*	
S7	$\beta\ (\text{A} \times \text{B}) = \beta\ (\text{A}) \times \beta\ (\text{B})$
S8	$\beta\ (\text{A} \bowtie_c \text{B}) = \beta\ (\text{A}) \bowtie_c \beta\ (\text{B})$, whenever *selection* with condition c can be pushed down
Combinations with *grouping* and *aggregation*	
S9	$\gamma_{f(c)}\ (\beta\ (\text{A})) = \gamma_{f(c)}\ (\text{A})$, for any column c and aggregation function $f \in \{\max, \min\}$
S10	$\beta\ (\gamma_{f(c)}\ (\text{A})) = \gamma_{f(c)}\ (\text{A})$, for any column c and any f
S11	$\gamma_{c,f(d)}\ (\beta\ (\text{A})) = \gamma_{c,f(d)}\ (\text{A})$, for any different columns c, d with c being the grouping column and not containing NULL values and aggregation function $f \in \{\max, \min\}$
(S12	$\beta\ (\gamma_{c,f(d)}\ (\text{A})) = \gamma_{c,f(d)}\ (\text{A})$, for any different columns c, d with c being the grouping column and not containing NULL values and aggregation function $f \in \{\max, \min\}$
Other combinations	
S13	$\tau\ (\beta\ (\text{A})) \neq \beta\ (\tau\ (\text{A}))$
S14	$\delta\ (\beta\ (\text{A})) = \beta\ (\delta\ (\text{A}))$
S15	$\beta\ (\beta\ (\text{A})) = \beta\ (\text{A})$
S16	$\kappa\ (\beta\ (\text{A})) \neq \beta\ (\kappa\ (\text{A}))$

(b) Rules for *subsumption*

Table 6: Transformation rules for *subsumption* and *complementation* in combination with other operators.

operator without changing the final result (Rules S3 and S4). For *complementation* and *projection*, the same considerations apply as for *subsumption* and *projection* (Rules C3 and C4).

Combinations with Selection: The main goal is to push *selection*s toward the base relations to decrease input cardinality. In case the *selection* is applied to a column c without NULL values (e.g., a key), we can indeed push *selection* down through the operator (Rule S6). If column c allows NULL values, pushing *selection* through *subsumption* alters the result only if a tuple subsuming another tuple is removed from the input of *subsumption* by the *selection*. We distinguish several cases resulting in Rules S5 and S6. In contrast to *subsumption*, where only the subsuming tuple needs to be preserved, computing the complement requires all complementing tuples to be kept in the input of *complementation*. We consider the same cases as for *subsumption* and need to assure that either both or none of the complementing tuples pass the *selection*. This property can be achieved for some cases when pushing *selection* through *complementation* by adding an additional condition B `IS NULL` to the selection predicate. However, as *complementation* combines its input tuples into new tuples, the original *selection* has to be applied again after *complementation* (Rules C5 and C6). If the operator tree includes a *selection* with conjuncts or disjuncts of conditions, we can push it entirely through *subsumption* or *complementation* if there is no single condition that prohibits the pushdown. Then, we first need to split the conditions using the standard transformation rules for *selection* and then push them according to the rules above.

Combinations with Join: When exchanging *subsumption* and *cartesian product* we must ensure that when

applying *cartesian product*, (1) no additional subsuming tuples are introduced if there are none in the base relations, and (2) all subsuming tuple pairs in the base relations still exist after applying *cartesian product*. The former follows from the definition of tuple subsumption and the latter follows from the fact that by *cartesian product*, two subsuming tuples from one base relation are combined with the same tuples from the other base relation and therefore the two resulting tuples being subsumed in the result of the *cartesian product*. When exchanging β and $\bowtie_c$ in the query plan, we need to apply the rules involving *selection* from the previous paragraph (Rules S7 and S8). Considering *complementation*, the same two properties as for *subsumption* have to be ensured. However, already the first property is no longer satisfied: applying *cartesian product* may introduce complementing tuples, even if there are no complementing tuples in the base relation. More precisely, it can be shown that this is the case for base relations that contain subsumed tuples. We can fix this problem by introducing an additional κ operator on top that removes the newly introduced complementing tuples (Rules C7 and C8). Based on the above observation and the transformation rules for *selection* and *complementation*, a general rule for combining *join* and *complementation* cannot be devised.

Combinations with Grouping and Aggregation: In general, *subsumption* and *grouping* and *aggregation* are not exchangeable, because *subsumption* may remove tuples that are essential for the computation of the aggregate. However, there are certain cases in which we can remove the *subsumption* operator and leave only the *grouping/aggregation* (Rules S9-S13). If non-standard aggregation functions are allowed, the rules above extend in a way that only null-tolerant, duplicate-insensitive functions are allowed to be be used as function f, such as *min*, *max*, *shortest*, and *longest*. Similar rules (Rules C9-C12) hold for *complementation*.

Other Combinations. The *subsumption* and *complementation* operators are not order-preserving (Rules S13 and C13). When dealing with bags of tuples, *subsumption/complementation* and *distinct* do not interfere with each other (Rules S14 and C14). Additionally, two *subsumption* operators can be combined into one (Rule S15). Although they seem to handle two entirely different cases, *subsumption* and *complementation* are not exchangeable. Their order matters, as a tuple that complements another tuple (and therefore adds some additional information to it) may well be subsumed by another tuple, resulting in that additional information not being added (Rule S16). Finally, two *complementation* operators can be combined into one (Rule C15) and a relation needs to have at least three attributes for two tuples being able to complement each other (Rule C16).

Referring to the example in Fig. 2, Plan A can be transformed into Plan B by splitting the selection, applying rules C6, S6, pushing down selections, S1, S8, S1, and recombining the top selection again.

3 Cost Model and Selectivity Estimates for *Subsumption* and *Complementation*

To estimate runtimes of query plans we give a brief sketch of cost formulas for some of the implementations for computing *subsumption* and *complementation*. We consider the subsumption algorithms – presented in [2] – *Simple Subsumption* and *Null-Pattern-Based Subsumption* with special partitioning steps, which exhibited high efficiency in practice. For complementation we consider the *Simple Complement* algorithm and the *Partitioning Complement* algorithm, which were introduced in [3]. The cost formulas are based on the number of distinct tuple operations (e.g., tuple comparisons). A detailed cost model would require quite specific information about the distribution of NULL and other values among the attributes to determine partition sizes. As such information is difficult to acquire or costly to store and to keep up-to-date we restrict the cost model to only include the number of partitions and the size of the NULL partition as both can easily be deduced.

Cost formulas: Let n be the number of tuples in the relation. The cost formula for the *Simple Subsumption* algorithm is given as $\mathcal{C}_{SMPS} = \frac{1}{2}n^2$. The cost formula for the *Null-Pattern-Based Subsumption* algorithm is given as $\mathcal{C}_{NPBS} = n \log n$. We cover its partitioning variant in more detail; all algorithms are given in [2]. Assuming $k = |P_\perp|$ as the size of the NULL partition and p as the number of partitions P_i, then the average size of a partition P_i is $\frac{n-k}{p}$ (assuming uniform distribution). Furthermore, we assume an intermediate selectivity of 100% (no tuple subsumes the other), assume using the *Simple Subsumption* algorithm and that creating the

partitioning can be done at practically no cost while reading the relation into memory. Then, the cost formula for the *Partitioning(SMPS)* algorithm is: $\mathcal{C}_{\textit{Partitioning(SMPS)}} = \frac{1}{2}k^2 + \frac{1}{2}\frac{(n-k)^2}{p} + nk - k^2$. We consider two special cases: If there is no NULL partition, then $k = 0$ and the formula is simplified to $\mathcal{C}_{\textit{Partitioning(SMPS)}} = \frac{1}{p}\frac{1}{2}n^2$, thus showing the performance boost by partitioning. This effect has been verified in experiments. The second special case is to partition by a key, and additionally allowing for a NULL partition. Then, the number of partitions p is $p = n - k$, thus simplifying the formula to $\mathcal{C}_{\textit{Partitioning(SMPS)}} = -\frac{1}{2}k^2 - \frac{1}{2}k + \frac{1}{2}n + nk$, resulting in an asymptotic runtime of $\mathcal{O}(n)$ in the number of tuples and $\mathcal{O}(k^2)$ in the size of the NULL partition. A cost formula for *Partitioning(NPBS)* can be devised accordingly: $\mathcal{C}_{\textit{Partitioning(NPBS)}} = k \log k + (n-k) \log \left(\frac{n-k}{p}\right) + (n-k)k$.

For *complementation* we consider cost functions for the *Simple Complement* algorithm and the *Partitioning Complement* algorithm given in [3]. With m being the size of the largest set of tuples complementing each other (largest *maximal complementing set* MCS) and n being the number of tuples of the relation, the cost results as the sum of the costs for the two steps of the algorithm $\mathcal{C}_{\textit{Simple Complement}} = \frac{1}{2}n(n-1)2^m + n2^m$. As in the case for partitioning and *subsumption* we assume $k = |P_\perp|$ as the size (in tuples) of the NULL partition, n as the size of the relation and p as the number of partitions P_i, then – assuming an equal distribution of tuples among partitions – the average size of a normal partition P_i is $\frac{n-k}{p}$. In the algorithm, tuples from $|P_\perp|$ are added to each partition before building complements on the partitions, so each P_i' has a size of $\frac{n-k}{p} + k$. Furthermore we assume applying the *Simple Complement* algorithm to the partitions and that creating the partitioning can be done at practically no cost while reading the relation into memory. This leads to the general cost formula for the *Partitioning Complement* algorithm: $\mathcal{C}_{\textit{Partitioning Complement}} = 2^m p \left(\frac{1}{2}\left(\frac{n-k}{p} + k - 1\right)\left(\frac{n-k}{p} + k\right) + \left(\frac{n-k}{p} + k\right)\right) + pk$. We consider two special cases: If there is no NULL partition, then $k = 0$ and we simplify to $\mathcal{C}_{\textit{Partitioning Complement}} = \frac{1}{2}\frac{1}{p}n^2 2^m + \frac{1}{2}n2^m$. Considering the second special case, where we partition by a key, and additionally allow for a NULL partition, then the number of partitions p is $p = n - k$ and the cost formula results in: $\mathcal{C}_{\textit{Partitioning Complement}} = 2^m(n-k)(\frac{3}{2}k + \frac{1}{2}k^2 + 1) + nk - k^2$.

Selectivity estimation: Given a selectivity factor $\mathcal{S}$ for an operator, the output cardinality of an operation is computed by multiplying it with the input cardinality. In the case of subsumed and complementing tuples, selectivity estimation is accomplished by estimating how many tuples will be subsumed or complemented by computing probabilities of NULL values existing in tuples.

Consider the case of a simple relation with only one attribute. All tuples that are NULL in that single attribute are subsumed by others. The number of NULL values can easily be deduced from the relations histogram; the probability $\mathcal{P}(a_1 = \perp)$ that a tuple has a NULL value in attribute a_1 can be determined likewise. We present a simple model for estimating selectivity based on this limited information. However, estimating the correct number of subsumed tuples in general is a difficult task: Our experiences with real-world datasets show that (1) this number can significantly vary, and that (2) NULL values – which highly influence this number – are not always evenly distributed among tuples and columns.

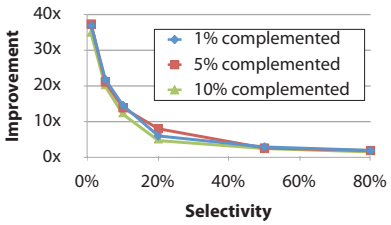
In the following we assume a) an equal distribution of values in an attribute and b) independence among attributes. We estimate the number of subsumed or complementing tuples by computing the number of tuples that contain NULL values, as this is the precondition for a tuple to be subsumed or complementing. However, this way, we might overestimate the number of actual subsumed or complementing tuples, because for a tuple being subsumed there must exist another tuple that coincides in all NON-NULL attributes.

Given a relation with two attributes a_1 and a_2, and the likelihood that $a_1 = \perp$ by $p_1^\perp$ and $a_2 = \perp$ by $p_2^\perp$ respectively. Then the likelihood $\mathcal{P}_2$ that a tuple contains at least one NULL values is $1 - (1 - p_1^\perp)(1 - p_2^\perp)$. Expanding this formula to the case of k attributes a_i results in the following formula for the likelihood that a tuple contains at least one NULL value: $\mathcal{P}_k = (\text{tuple contains at least one NULL value}) = 1 - \mathcal{P}(\text{all } a_i = \perp) = 1 - \mathcal{P}(\bigwedge_{i=0}^k a_i \neq \perp) = 1 - \prod_{i=1}^k (1 - p_i^\perp)$ The likelihood $p_i^\perp$ can be approximated by the count of NULL values in the attributes from the relations histogram. If all values in a_i are uniformly distributed, then $p_i^\perp = \frac{1}{|a_i|}$ with $|a_i|$ being the number of different attribute values in a_i. Thus, $\mathcal{P}_k^= = 1 - \prod_{i=1}^k (1 - \frac{1}{|a_i|})$.

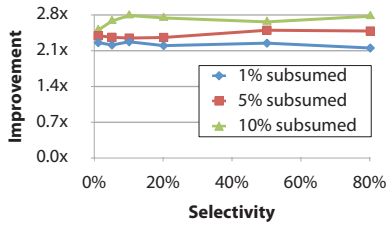
Selectivity estimation for *subsumption* and *complementation* then is determined using probability P_k: $\mathcal{S}_\beta = \mathcal{S}_\kappa = 1 - \mathcal{P}_k = \prod_{i=1}^{k}(1 - p_i^{\perp})$ for operators β and κ and $p_i^{\perp}$ being the likelihood of a NULL value present in column a_i. Additionally, in both cases the lower bound for selectivity is given by the maximum of distinct values over all attributes, as this marks the number of tuples that are left over at least: $\mathcal{S}_\beta \geq \frac{\max|a_i|}{|\mathrm{R}|}$ and $\mathcal{S}_\kappa \geq \frac{\max|a_i|}{|\mathrm{R}|}$.

4 Experimentation

We exemplarily evaluate transformation Rules C6 and S8, measuring the effect they have on runtime. More specifically, in both cases, we measure the runtime of a set of simple queries on artificial data where a *selection/join* follows a *complementation/subsumption* and compare their runtime to the transformed queries where *complementation/subsumption* follows *selection/join*. In both cases, we respectively varied the percentage of complemented and subsumed tuples (1%, 5%, and 10%). Also, we tested different selectivities between 1% and 80% for the *selection* and *join* operator, respectively. We report the performance gain of both cases in Fig. 3, on generated tables with six columns. Runtimes are median values over ten runs and the figure shows the values for tables of (a) one million and (b) 20.000 tuples. As implementations of *subsumption* and *complementation*, we used *Partitioning Complement* and the *Simple* algorithm [2, 3], respectively.



(a) Pushing *selection* below *complementation*



(b) Pushing *subsumption* below *join*

Figure 3: Runtime improvement on artificial data when applying Rules C6 and S8.

Regarding Fig. 3(a), we see that pushing *selection* below *complementation* pays off. A similar improvement (not shown here) has been observed for the combination *selection/subsumption* (Rule S6). Indeed, the performance gain ranges from around 1.4x times for a selectivity of 80% up to around 37x for a selectivity of 1%, indicating that the transformed query (*selection* pushed down) is always faster for the tested combinations of selectivity, size, and percentage of subsumed tuples. Experiments with other table sizes show similar results. The performance gain is higher for lower selectivities of the *selection*, as in these cases fewer tuples pass the *selection* and are input to the costly *complementation* operation. The difference in performance gain between tables with different percentages of complemented tuples is small, but consistently existent, except the outlier for 20% selectivity.

The results reported in Fig. 3(b) show an improvement of around 2.5x for all selectivities and the three percentages of subsumed tuples. Experiments with other table sizes show a comparable behavior. Improvement varies a bit but we see a tendency to a general and uniform performance gain. This is mainly due to the difference in total runtime of the two considered operators: whereas the *subsumption* operator is optimized for performance, the main part of the queries' total runtime is consumed by the implementation of the *join*[1]. Essentially, Fig. 3(b) shows the improvement caused by the reduced input cardinality of the *join*. The improvement is higher for tables with 10% subsumed tuples (than it is for tables with 5% or even 1% subsumed tuples), because more subsumed tuples are removed from the input to the *join*. Although the improvement is not as large as in the other experiment, it shows that applying the transformation rule pays off.

When operators can be eliminated from the tree (e.g., C9-C12 and S9-S12) performance gain is evident. Other rules, such as C3-C4 and S3-S4, do not need evaluation. Experimentation for the remaining rules is deferred to future work.

[1]We implemented our operators in the XXL framework [5] and use its general-purpose Nested-Loop-Join implementation.

5 Related Work and Conclusions

Related work on conflict resolution and data fusion is widely covered in [1]. The *minimum union* operator [7] is used in many applications, e.g., in query optimization for *outer join* queries [6, 9]. However, an efficient algorithm for the general subsumption task is still considered an open problem therein. Different to our work is the assumption that the base relations do not contain subsumed tuples and the use of *join* instead of *union* to combine tables before removing subsumed tuples. [9] proposes a rewriting for *subsumption*, using the data warehouse extensions provided by SQL. However, removing subsumed tuples using the proposed SQL rewriting depends on the existence of an ordering such that subsuming tuples are sorted next to each other. As subsumption establishes only a partial order, such an ordering does not always exist. [2] presents efficient algorithms to compute subsumption in the general case as well as more related work. Data fusion with the semantics of *complement union* has been first introduced in [2, 3], together with definitions, implementation details and a first draft of transformation rules. However, similar concepts have been previously explored. Replacing complementing tuples by their complement in a relation is equivalent to finding all maximal cliques in a graph that has been constructed by creating one node per tuple and an edge between nodes if the corresponding tuples complement one another [4].

To conclude, data fusion, despite its seemingly simple nature, is a complex and important task in the field of data integration. In this article we have addressed only cases in which a NON-NULL value competes with a NULL-value – deciding to choose the NON-NULL value for the fused record is natural. However, doing so in a consistent and efficient manner is not trivial. We have introduced the two concepts of subsumption and complementation as new algebraic operators, and have shown how to incorporate them into relational DBMS by providing corresponding transformation rules, a simple cost model and selectivity estimates. Future work lies in the direction of a refinement of the cost model.

Acknowledgments. This research was partly funded by the German Research Society (DFG grant no. NA 432) and was performed while the authors worked at the Hasso Plattner Institute.

References

[1] Jens Bleiholder and Felix Naumann. Data fusion. *ACM Computing Survey*, 41(1):1–41, 2008.

[2] Jens Bleiholder, Sascha Szott, Melanie Herschel, Frank Kaufer, and Felix Naumann. Subsumption and complementation as data fusion operators. In *International Conference on Extending Database Technology (EDBT)*, 2010.

[3] Jens Bleiholder, Sascha Szott, Melanie Herschel, and Felix Naumann. Complement union for data integration. In *International Workshop on New Trends in Information Integration (NTII)*, 2010.

[4] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[5] Jochen Van den Bercken, Björn Blohsfeld, Jens-Peter Dittrich, Jürgen Krämer, Tobias Schäfer, Martin Schneider, and Bernhard Seeger. XXL - a library approach to supporting efficient implementations of advanced database queries. In *International Conference on Very Large Databases (VLDB)*, 2001.

[6] César Galindo-Legaria and Arnon Rosenthal. Outerjoin simplification and reordering for query optimization. *ACM Transactions on Database Systems (TODS)*, 22(1):43–74, 1997.

[7] César A. Galindo-Legaria. Outerjoins as disjunctions. In *ACM International Conference on Management of Data (SIGMOD)*, 1994.

[8] Sergio Greco, Luigi Pontieri, and Ester Zumpano. Integrating and managing conflicting data. In *Revised Papers from the 4th International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 349–362. Springer-Verlag, 2001.

[9] Jun Rao, Hamid Pirahesh, and Calisto Zuzarte. Canonical abstraction for outerjoin optimization. In *ACM International Conference on Management of Data (SIGMOD)*, 2004.

Efficient and Effective Analysis of Data Quality using Pattern Tableaux

Lukasz Golab, Flip Korn and Divesh Srivastava
AT&T Labs - Research
180 Park Avenue, Florham Park NJ, 07932, USA
{lgolab, flip, divesh}@research.att.com

Abstract

Data Auditor is a system for analyzing data quality via exploring data semantics. Given a user-supplied constraint, such as a functional dependency or an inclusion dependency, the system computes pattern tableaux*, which are concise summaries of subsets of the data that satisfy (or fail) the constraint. The engine of Data Auditor is an efficient algorithm for finding these patterns, which defers expensive computation on patterns until needed during search, thereby pruning wasted effort. We demonstrate the utility of our approach on a variety of data as well as the performance gain from employing this algorithm.*

1 Introduction

Recently, a constraint-based approach has shown promise in detecting and correcting data quality problems; some specific examples of constraints employed include Conditional Functional Dependencies (CFDs) [7], Conditional Inclusion Dependencies (CINDs) [2], and Conditional Sequential Dependencies (CSDs) [10]. A key idea behind this approach is to use *conditioning*, that is, to allow different (but possibly overlapping) subsets of the data to satisfy a supplied constraint locally, rather than forcing the entire data to satisfy the constraint globally. This is especially useful when the data evolve over time or have been integrated from multiple sources and, thus, exhibit heterogeneity with locally shared characteristics.

We have developed a system—Data Auditor—that employs this approach as follows (see [9] for more details). The user first hypothesizes a constraint, either based on expertise and intuition, or one that is automatically derived (algorithms for doing so are outside the scope of this paper). What Data Auditor provides is a platform for testing such hypotheses, performing multidimensional analysis to discover and parsimoniously summarize which portions of the data fit (or fail) the given constraint. That is, users can "try out" a constraint to see if (and where) it holds or fails.

Discovering and summarizing which subsets of the data satisfy a supplied constraint is useful for understanding the semantics of data. This, in turn, is useful not only for making sense of analysis results, but also for detecting quality problems, which may be indicated by violations of these semantics. As a caveat, violations of a user-specified constraint do not always indicate data quality problems but could be naturally occuring (and perhaps interesting) phenomena in the data that were unknown to the user; it is up to the domain expert to make this inference.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

There are two crucial observations to make this approach effective on real world data. The first is that the notions of satisfaction and violation of a constraint should be robust to outliers. For example, a single spurious tuple should not cause a certain portion of the data to be eliminated from consideration. Therefore, a small number of exceptions are allowed before declaring failure of constraint satisfaction; this relaxation is quantified as the *confidence* of a constraint. The other observation is that the reported summary of semantics should not overfit the data. On the one hand, satisfaction of the constraint should be pervasive enough in the data to be considered a legitimate rule. At the same time, the constraint should not be applied to regions of the data having poor quality. Therefore, we do not require all the data to be captured in the summary but only a minimum fraction, denoted by the *support*. This flexibility enables the constraint to "snap" to the portion of best fit.

The mechanism by which the (satisfaction or violation of a) constraint is concisely summarized is a special kind of relation called a *tableau*, which we define in the next section. Briefly, a tableau consists of patterns, each of which lists the attribute values that most (but not necessarily all) satisfying or violating tuples have in common. Effectively and efficiently finding such tableaux is the subject of this paper. Previous work assumed that the attributes used to construct subsets of the data on which to test the given constraint are shared with those specified in the constraint itself [11]; this assumption was made for the purpose of efficiency, as the so-called "on-demand" algorithm [11] for tableau discovery exploited this fact. In this paper, we decouple the conditioning attributes from those in the constraint and discuss a more general on-demand algorithm. As a result, more powerful tableaux analyses are possible and can be done efficiently.

The remainder of this paper is organized as follows. Section 2.1 gives examples of two constraints that are common in Data Auditor. Section 2.2 defines patterns and tableaux. Section 2.2 presents the tableau generation problem and summarizes the on-demand algorithm. Section 2.4 illustrates specification within Data Auditor using the two examples. Section 3 gives a few case studies to demonstrate the utility of generalized pattern tableaux and reports the performance improvements of the on-demand algorithm over straightforward computation. Section 4 discusses related work and Section 5 concludes the paper.

2 Overview of Pattern Tableau Discovery

2.1 Supported Constraints

The supported constraints assert some property between two sets of attributes, X (corresponding to the LHS) and Y (corresponding to the RHS), from either the same or different relations. These attribute sets can be used in a variety of constraints, such as Equality-Generating Dependencies (EGDs) [6] to impose consistency and Tuple-Generating Dependencies (TGDs) [6] to impose completeness. We can define views corresponding to an arbitrary conjunction of atomic formulas on the LHS (for EGDs and TGDs) and the RHS (for TGDs), in which case simple constraints can be defined over these view tables. The requirements for these constraints will become clear in Section 2.2.

A common EGD constraint, which we highlight in this paper, is the functional dependency. Let R be a relation, t_k be a tuple and $t_k[Z]$ be the value(s) of its attribute(s) Z. A functional dependency $X \rightarrow Y$ is said to hold when $\forall i, j$, if $t_i[X] = t_j[X]$ then $t_i[Y] = t_j[Y]$. $X \subseteq R$ is a set of attributes referred to as the *antecedent* and $Y \subseteq R$ is a set of attributes referred to as the *consequent*.

A common TGD constraint is the inclusion dependency $R_1.X \subseteq R_2.Y$, for two relation schemas, R_1 and R_2. It is said to hold when for all tuples r_1 in the domain of R_1, there exists a tuple r_2 in the domain of R_2 with $r_1[X] = r_2[Y]$.

2.2 Pattern Tableaux

Real data are heterogeneous. Hence, in addition to reporting the confidence (i.e., the extent to which the constraint holds) on the entire relation, it is useful to identify parts of the relation that satisfy the constraint (i.e.,

have high confidence) and parts that violate it (i.e., have low confidence).

Following [7], we use *pattern tableaux* to encode subsets of a relation. Consider a schema $R = A_1, A_2, \ldots, A_k$. A tableau consists of a set of *patterns* over the selected *conditioning attributes* $X_p \subseteq R$, where each attribute in X_p contains a *symbol*. A symbol is either a value in the corresponding attribute's domain or a special "wildcard" symbol '*'. Let $p_i[A_j]$ denote the symbol corresponding to the jth conditioning attribute of the ith pattern, and let $t[A_j]$ be the value of the jth attribute of a tuple t. A tuple t is said to *match* a pattern p_i if, for each A_j in X_p, either $p_i[A_j] =$ '*' or $t[A_j] = p_i[A_j]$. We denote this matching operator as $t[A_j] \asymp p_i[A_j]$. The confidence of a pattern p_i is defined as the confidence of a sub-relation containing only those tuples that match p_i; its support is defined as the relative size of said sub-relation as compared to the whole relation. Note that the pattern consisting of all '*' symbols matches the entire relation.

2.3 Tableau Generation

The input to Data Auditor is:

- a pair of schemas R_1 and R_2 (which optionally can refer to the same relation);
- a constraint involving two sets of attributes, $X \subseteq R_1$ and $Y \subseteq R_2$, encoded as User-Defined Functions (UDFs) for incrementally computing the confidence of this constraint over a set of tuples;
- a set of conditioning attributes $X_p \subseteq R_1$;
- confidence threshold $\hat{c}$ (which is either a lower- or upper-bound on the confidence of each pattern)
- minimum support threshold $\hat{s}$.

The output is a tableau T_p over attributes X_p, each of whose patterns has confidence of at least (or at most[1]) $\hat{c}$, all of whose patterns collectively cover at least $\hat{s}|dom(R_1)|$ tuples, where $dom(R_1)$ denotes the instance of relation R_1. Applying Occam's Razor, the goal is for $|T_p|$ to be as small as possible while satisfying these constraints. This was formulated as an instance of PARTIAL SET COVER in [11], for which a greedy set cover heuristic attempts to produce the smallest possible tableau (having the fewest patterns) matching the largest possible fraction of the relation. Thus, general patterns with wildcards are more likely to be included (provided they have the appropriate confidence) than specific patterns matching a small fraction of the data. Tableau construction for more general constraints extends the *on-demand* algorithm from [11], which was originally proposed to generate tableaux for CFDs where the conditioning attributes, X_p, and FD antecedent attributes, X, are the same (see [11] for a detailed explanation). The only requirement for the generalization in this paper is that the confidence of any pattern can be computed in a single scan after sorting the data lexicographically on $X \cup Y$.

Starting from the all-wildcards pattern, the algorithm traverses the different patterns in top-down fashion, along the way inserting into the tableau patterns that meet the required confidence threshold and match the most tuples that have not already been matched. For each pattern p encountered, its support set (the subset of the tuples from R_1 matching p) is scanned for computing confidence. Therefore, the key is to associate a list of each pattern p's support set in the order visited, so that the confidence of any other pattern contained within p can be generated by scanning this list, rather than re-scanning the tuples in R_1.

An essential part of this algorithm is the computation of confidence for a given pattern. The API for specifying the confidence function of any pattern, given the pattern's support set as input, involves supplying six user-defined functions: `declare`, `init`, `update`, `closeXY`, `closeX` and `output`. These functions are used, respectively, to declare relations and attribute sets used by the constraint; initialize counter variables used in confidence computation; maintain these counters incrementally for each tuple t of R_1; "close" a sub-aggregate

[1] As in [11], we refer to the former as a *hold tableau*, since it summarizes subsets on which the constraint holds, and the latter as a *fail tableau*, which detects subsets on which the constraint fails.

when a new value of $t[XY]$ is encountered; "close" a super-aggregate when a new value of $t[X]$ is encountered; and report the final confidence. The output is a confidence value betwen 0 and 1 for the given pattern.

The choice of conditioning attributes is crucial to obtaining concise and informative tableaux—some attributes may be irrelevant to the constraint at hand. Fortunately, in many cases, useful attributes are self-evident. In future work, we plan to study automatic selection of conditioning attributes in more detail, e.g., by exploiting attribute correlations.

2.4 Examples

Define $S(p, R)$ as the support set of pattern p over R, that is, the set of tuples from relation R matching p:

$$S(p, R) = \{t : t \in dom(R) \wedge t[X_p] \asymp p\}.$$

A Conditional Functional Dependency (CFD) ϕ on R is a pair $(R : X \rightarrow Y, T_p)$, where (1) $X \rightarrow Y$ is a standard FD, referred to as the *embedded FD*; and (2) T_p is a pattern tableau conditioned over attributes X_p, where for each row $t_p \in T_p$ and each attribute $A \in X_p$, $t_p[A] = a$, for some $a \in dom(A)$, or $t_p[A] =$ '*'. One way to define the confidence of a pattern p over R with respect to the embedded FD is to compute the size of the largest "consistent" subset of $S(p, R)$. Formally, given a CFD $\phi = (R : X \rightarrow Y, T_p)$, let $Q(p, R)$ denote the set of tuples from $S(p, R)$ that remains after removing the fewest tuples needed to eliminate all violations. Then the confidence of pattern p is $\frac{|Q(p,R)|}{|S(p,R)|}$.[2] Table 7(a) shows UDFs based on this definition of confidence for the CFD $R : \{A, B\} \rightarrow C$ conditioned on $\{A, E, F\}$. Note that we find $|Q(p, R)|$ by partitioning the support set on the antecedent attribute(s) X, and, for each distinct value of X, finding the most frequently occurring value of the consequent attribute(s) Y.

A Conditional Inclusion Dependency (CIND) ψ on R_1 and R_2 is a pair $(R_1.X \subseteq R_2.Y, T_p)$, where (1) $R_1.X \subseteq R_2.Y$ is a standard IND, referred to as the *embedded IND*; and (2) T_p is a pattern tableau conditioned over attributes $R_1.X_p$, where for each row $t_p \in T_p$ and each attribute $A \in R_1.X_p$, $t_p[A] = a$, for some $a \in dom(A)$, or $t_p[A] =$ '*'. Given a CIND $\psi = (R_1.X \subseteq R_2.Y, T_p)$, let $Q(p, R_1)$ denote the tuples from $S(p, R_1)$ having a match in $dom(R_2)$. Then one way to define the confidence of pattern p is $\frac{|Q(p,R_1)|}{|S(p,R_1)|}$. Table 7(b) shows UDFs based on this definition of confidence for the CIND $R_1.B \subseteq R_2.C$ conditioned on $\{A, E, F\}$.

3 Case Studies

Here we consider both CFDs and CINDs where the attributes participating in the dependencies are decoupled from the conditioning attributes used for the tableau. In addition to illustrating the utility of discovering tableaux based on these dependencies (for the purposes of data exploration and data cleaning), we demonstrate the performance improvement that comes from employing the on-demand algorithm.

We used 60K records of sales data from an online retailer containing attributes `type, itemid, title, price, vat, quantity, userid, street, city, region, country, zip`. The first six attributes describe the item being purchased, including the tax rate (vat) and the quantity purchased in each order. The remaining attributes describe the client (keyed by userid) who purchased these items. Using the FD `{title}` $\rightarrow$ `{price,vat}` and conditioning attributes `{type,region,country}`, we obtained the tableau in Table 8 with $\hat{c} = 0.99$ (only the first six rows are displayed, yielding 0.379 cumulative support).

Since VAT typically applies to the country level, one might expect each separate country to require a separate row in the tableau. Indeed, there was such a partitioning based on uniformity of VAT (many of the individual countries are not shown in Table 8). An even more interesting characteristic captured in this tableau is that the

[2]This is the definition of confidence used in [11]; note that other definitions of confidence (see [13]) can easily be written as UDFs in our framework.

```
declare()
  R1 :  A,B,C,D,E,F;
  R2 :  A,B,C,D,E,F;
  X : A,B;
  Y : C;
  Xp :  A,E,F;
init()
  ntuples = 0;
  freq = 0;
  maxfreq = 0;
  sumfreq = 0;
update()
  ntuples++;
  freq++;
closeXY()
  if (freq > maxfreq)
    maxfreq = freq;
    freq = 0;
close X()
  sumfreq += maxfreq;
  maxfreq = 0;
output()
  return sumfreq/ntuples;
```

(a) CFD confidence

```
declare()
  R1 :  A,B,E,F;
  R2 :  C;
  X : B;
  Y : C;
  Xp :  A,E,F;
init()
  ntuples = 0;
  sumfreq = 0;


update(Tuple t)
  ntuples++;
  if (t[X] in Y) sumfreq++;
closeXY()
  /* do nothing */


close X()
  /* do nothing */

output()
  return sumfreq/ntuples;
```

(b) CIND confidence

Table 7: UDFs for two different constraints

type	region	country	conf	cumSupp
music	*	GBR	0.99	0.111
book	*	GBR	0.99	0.221
*	TX	*	0.99	0.274
*	CA	*	0.99	0.318
*	NY	*	0.99	0.351
*	PA	*	0.99	0.379

Table 8: Hold Tableau from Retailer data ($\hat{c} = 0.99$)

same title may be shared by different forms of media (and have different prices across the different media), which is the reason why music and book items from Britain (GBR) were separated into separate patterns. Note the absence of the tableau row (dvd,*,GBR), indicating that the FD didn't hold for the only other media type (DVDs). After inspecting the data, this turned out to be due to a greater variety of prices for the same item among DVDs than among other media types. Since, unlike Britain, VATs are not uniform throughout the country but vary per region in the United States, the individual states (listed by volume of purchases) needed to form separate patterns to achieve high enough confidence. Here the effect of common titles among different media types did not play as significant a role, since fewer records in each USA state, compared to all of GBR, resulted in almost no conflicts. We also compared the running time of generating this tableau using the on-demand algorithm against that of a standard greedy set cover algorithm that does not employ our optimizations. The performance improvement from computing on-demand was about two-fold, from 550 milliseconds down to 310 milliseconds.

Using 3,601,434 packet traces of IP network attack traffic obtained from the 1999 ACM KDD Cup, including attributes such as service, protocol, flag, attack_name, bytes, we postulated the FD {attack_name} $\rightarrow$ {flag} and conditioned on {service,protocol,attack_name}, which generated the tableau in Table 9 with $\hat{c} = 0.99$ and $\hat{s} = 0.7$. This revealed the homogeneity of icmp traffic

service	protocol	attack_name	conf	cumSupp
icmp	*	*	0.99	0.626
*	private	*	1	0.783

Table 9: Hold Tableau from KDDcup data ($\hat{c} = 0.99$)

img_bits	img_media_type	img_user_text	conf	cumSupp
*	*	ProteinBoxBot	0.971	0.0334
*	AUDIO	*	0.884	0.0407
7	*	*	0.88	0.047
5	*	*	0.885	0.0525
6	*	*	0.875	0.0576
*	*	Blofeld of SPECTRE	0.986	0.0609
*	*	Melesse	0.991	0.0636

Table 10: Hold Tableau from Wikipedia data ($\hat{c} = 0.85$)

which, after examining the data corresponding to pattern (icmp,⋆,⋆), was apparently dominated with echo requests (ecr_i) and very few other protocols. While some of these packets were normal traffic, more than 99% of these ecr_i requests were used for so-called smurf denial-of-service attacks. Examining the data matching pattern (⋆,private,⋆) revealed that the private protocol almost always occurred as a result of the neptune SYN flooding attack, which accounted for almost 93% of private traffic. Once again, the running time of on-demand was cut in half over the unoptimized greedy set cover algorithm, from 13 seconds to 6.3 seconds.

For discovering CINDs, we used two tables from Wikipedia data: the Image table, containing 777,828 records of multimedia objects with attributes such as img_name, img_size, img_bits, img_media_type, img_user_text; and the Imagelinks table, containing 15,532,084 records of pages containing those objects, with attributes il_from and il_to (denoting links from webpages to image files, respectively). We asserted the IND {Image.img_name} ⊆ {Imagelinks.il_to} to identify images that appeared on web pages, and generated the tableau in Table 10 with $\hat{c} = 0.85$ (only the first 7 rows are displayed, collectively giving 0.0636 support). This tableau reveals that audio content submitted to Wikipedia generally "made it to press" (appeared on a web page), as did 5-, 6- and 7-bit content (note: almost 95% of the content in Image is 8-bit). It also revealed some high-frequency users whose multimedia content generally made it to press such as ProteinBoxBot, which is a well known bot for creating and amending a large number of Wikipedia pages corresponding to mammalian genes (and by far the largest contributor to Image). A fail tableau with $\hat{c} = 0.2$ (not shown) revealed that content created using MS-Office tools (pdf, xls, etc.) was generally rejected, as well as some high-frequency users who provided much content that was not linked from any page. The performance improvement on this data was not as dramatic (18 seconds for on-demand compared to 19 for unoptimized greedy set cover), perhaps due to the overhead in reading the primary keys into a hash table.

We also used Caltrans automobile traffic data polled every 30 seconds from road sensors over the span of a day. Here the tables are Polls, containing fields time, id, district, sensor_type, route, road_name, and StationsTimes, containing the entire cross-product of 1K stations and 2880 times during the day that should have been polled. We asserted the IND {StationsTimes.time,StationsTimes.id} ⊆ {Polls.time,Polls.id} and generated the 8-

district	sensor_type	route	street_name	conf	cumSupp
5	*	*	*	0.934	0.165
80	*	3	*	0.904	0.264
51	*	*	*	0.907	0.341
80	*	1	*	0.916	0.412
99	*	1	*	0.914	0.48
50	OR	2	*	0.912	0.501
99	OR	*	*	0.952	0.518
113	ML	*	*	0.945	0.534

Table 11: Hold Tableau from RoadTraffic data ($\hat{c} = 0.9$)

district	sensor_type	route	street_name	conf	cumSupp
*	*	*	Iron Point Rd	0	0.004
			...		
*	*	*	Saw Mill Rd	0	0.057
5	OR	*	Elk Grove Blvd	0	0.058
			...		
*	ML	*	15th St	0.097	0.062
			...		

Table 12: Fail Tableau from RoadTraffic data ($\hat{c} = 0.1$)

row tableau in Table 11 with $\hat{c} = 0.9$. It reveals some districts and streets where the measurement quality was best, as well as route and sensor-type combinations. The fail tableau using $\hat{c} = 0.1$ in Table 12 revealed that most of the missing measurements occur along the same street, and gives many examples of these. For some of these streets, it is only certain sensor-types or along certain routes for which the problems occurred. There were also some patterns specifying no street at all but rather that the problem occurred along multiple streets. The on-demand algorithm took 2m7s whereas the unoptimized greedy set cover heuristic algorithm took 6m45s.

4 Related Work

A great deal of work exists on data quality analysis and data cleaning; see, e.g., [15] for a survey. In particular, various integrity constraints have recently been proposed to enforce data quality, including Conditional Functional Dependencies (CFDs) [7], Conditional Inclusion Dependencies (CINDs) [2] and Conditional Sequential Dependencies (CSDs) [10]. The key concept behind these constraints is the notion of conditioning: rather than requiring the constraint to hold over the entire relation, it need only be satisfied over conditioned subsets of the data summarized by a *pattern tableau*.

The problem of automatically discovering tableaux for a CFD, given an embedded FD, was first studied in [11, 3]. [3, 8] considered variations of CFD discovery, using different frameworks and optimization criteria, where the focus was on discovering individual patterns; [5] considered CIND discovery using their framework.

Here we extend the on-demand algorithm originally proposed in [11] to allow for several generalizations including arbitrary conditioning attributes (as well as other constraints besides CFDs). The notion of decoupling conditioning attributes from those used in the constraint was employed in [1] and [2].

Finally, we point out the orthogonal problem of discovering which integrity constraints hold on a given relation (in contrast to our goal to discover a tableau for a known constraint). Discovering FDs has been studied in [12, 13]; discovering CFDs in [3, 8]; and discovering INDs has been studied in [14].

5 Conclusions

We demonstrated the approach for data quality analysis employed in Data Auditor: hypothesizing a constraint and then discovering (hold and fail) tableaux which capture the data semantics (and violations of them) by conditioning on the supplied multidimensional attributes. The generalized on-demand algorithm for efficiently finding such tableaux enables a more powerful analysis than was previously available on large data sets. We are considering several directions for future work, among them UDF query optimization, applying our tableau discovery framework to other types of constraints, and discovering the underlying constraints in addition to constructing tableaux for them.

References

[1] L. Bravo, W. Fan, F. Geerts, and S. Ma. Increasing the Expressivity of Conditional Functional Dependencies without Extra Complexity. *ICDE 2008*, 516-525.

[2] L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. *VLDB 2007*, 243-254.

[3] F. Chiang and R. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166-1177, 2008.

[4] G. Cormode, L. Golab, F. Korn, A. McGregor, D. Srivastava, and X. Zhang. Estimating the confidence of conditional functional dependencies. *SIGMOD 2009*, 469–482.

[5] O. Cure. Conditional Inclusion Dependencies for Data Cleansing: Discovery and Violation Detection Issues. *QDB Workshop 2009*.

[6] R. Fagin and M. Vardi. The theory of data dependencies - an overview. *ICALP 1984*, 1-22.

[7] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(2):1-48, 2008.

[8] W. Fan, F. Geerts, L. Lakshmanan and M. Xiong. Discovering conditional functional dependencies. *ICDE 2009*, 1231-1234.

[9] L. Golab, H. Karloff, F. Korn, and D. Srivastava. Data Auditor: Exploring Data Semantics using Tableaux *PVLDB*, 3(2): 1641-1644 (2010).

[10] L. Golab, H. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 2(1): 574-585 (2009).

[11] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376-390, 2008.

[12] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100-111, 1999.

[13] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theor. Comput. Sci.*, 149(1):129-149, 1995.

[14] F. D. Marchi, S. Lopes, and J.-M. Petit. Unary and n-ary inclusion dependency discovery in relational databases. *Journal of Intelligent Information Systems*, 32(1):53–73, 2009.

[15] E. Rahm and H. H. Do. Data cleaning: problems and current approaches. *Data Eng. Bulletin*, 23(4): 3-13, 2000.

A Uniform Dependency Language for Improving Data Quality

Wenfei Fan Floris Geerts
University of Edinburgh
{wenfei, fgeerts}inf.ed.ac.uk

Abstract

A variety of dependency formalisms have been studied for improving data quality. To treat these dependencies in a uniform framework, we propose a simple language, Quality Improving Dependencies (QIDs). We show that previous dependencies considered for data quality can be naturally expressed as QIDs, and that different enforcement mechanisms of QIDs yield various data repairing strategies.

1 Introduction

Data quality has been a longstanding line of research for decades, and has become one of the most pressing challenges for data management. As an example, it is estimated that dirty data costs US companies alone 600 billion dollars each year [11]. With this comes the need for studying techniques for improving data quality.

A variety of dependency formalisms have recently been studied to specify data quality rules, *e.g.,* functional dependencies (FDs) [1], conditional functional dependencies (CFDs) [6, 14], order dependencies (ODs) [22, 29, 23], currency dependencies (CDs) [17], sequential dependencies (SDs) [25], matching dependencies (MDs) [18, 13, 4] and editing rules (eRs) [19]. These dependencies have proved useful in detecting and correcting inconsistencies in relational data. However, given these different formalisms, one would naturally ask which dependencies should be used in an application? Is a data quality rule expressible in one formalism but not in another? Is it more expensive to use one formalism than another? How should dependencies be enforced to repair data? These suggest that we study these formalisms in a comparative basis.

This paper takes a first step toward answering these questions. We propose a simple dependency language, referred to as Quality Improving Dependencies (QIDs), and show that all the dependencies mentioned above can be expressed as QIDs. That is, QIDs provide a uniform language to characterize these formalisms. We also introduce different mechanisms for enforcing QIDs as data quality rules, via revisions of the chase process [1]. We show that data repairing algorithms [5, 9, 19, 20] are just implementations of these mechanisms.

In the rest of the paper, we introduce QIDs (Section 2), and show that dependencies studied for data quality are special cases of QIDs (Section 3). In addition, we present data quality problems in terms of QIDs (Section 4), emphasizing data repairing as QID enforcement. We also give an overview of recent work on these issues.

2 Quality Improving Dependencies

We define QIDs in terms of a notion of comparison relations, which compare pairs of tuples based on some of their characteristics. Below we first introduce comparison relations, and then define QIDs.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

2.1 Comparison Relations

Consider a database schema $\mathcal{R} = (R_1, \ldots, R_m)$, in which each relation schema R_i is specified with a set of attributes, denoted by $R_i\big(\mathsf{tid}, A_1, \ldots, A_{\mathsf{n}(i)}\big)$. Here tid is a *tuple identifier* and the domain of attribute A_j is denoted by $dom(A_j)$. We say that two attributes A_j and A_k are *compatible* if $dom(A_j) = dom(A_k)$. An instance $\mathcal{I}$ of $\mathcal{R}$ is $(I_1, \ldots, I_m)$, where for $i \in [1, m]$, I_i is finite set of tuples t with $t[A_j] \in dom(A_j)$ for each $A_j \in R_i$.

A comparison relation is simply a binary relation $\sim \subseteq dom(\mathsf{tid}\,) \times dom(\mathsf{tid}\,)$. In practice, it is often used to group tuples (identified by their tid's) by comparing certain attributes (properties) of the tuples. More specifically, let X be a set of attributes $\{A_1, \ldots, A_k\}$, and $\mathcal{B}$ be a "binary" relation $\mathcal{B} \subseteq dom(X) \times dom(X)$ that compares the X attributes of tuples, where $dom(X) = dom(A_1) \times \cdots \times dom(A_k)$. A *comparison relation via* $(X, \mathcal{B})$, denoted by $\sim^X_{\mathcal{B}}$, is defined such that for any tuples t_1 and t_2 that contain attributes X, $\sim^X_{\mathcal{B}}\ (t_1[\mathsf{tid}], t_2[\mathsf{tid}])$ iff $\mathcal{B}(t_1[X], t_2[X])$. Let $\mathcal{I}$ be an instance of $\mathcal{R}$ and I be the set of all tuples in $\mathcal{I}$ that contain attributes X. Then

$$\sim^X_{\mathcal{B}} := \big\{(\mathsf{tid}_1, \mathsf{tid}_2) \mid t_i \in I,\ t_i[\mathsf{tid}\,] = \mathsf{tid}_i,\ i = 1, 2,\ \mathcal{B}(t_1[X], t_2[X])\big\}$$

for instance $\mathcal{I}$. That is, $\sim^X_{\mathcal{B}}$ relates pairs of tuples by only considering the information present in their X attributes. Observe that a pair of tuples related by $\sim^X_{\mathcal{B}}$ may possibly come from *different* relations in $\mathcal{I}$.

As an example, we consider comparison relations $\sim^A_{\mathcal{B}}$ defined on an attribute A, where $\mathcal{B}$ is a variation of the equality relation on A, *i.e.,* a binary relation in $dom(A) \times dom(A)$ that is reflexive, symmetric and transitive.

Example 1: One can define $\sim^A_{\mathcal{B}}$ to group tuples based on the equality of a certain attribute, via the following $\mathcal{B}$:
(1) $\mathsf{eq}_A = \{(a, a) \mid a \in dom(A)\}$, *i.e.,* for any pair (t_1, t_2) of tuples, $\sim^A_{\mathsf{eq}_A}\ (t_1[\mathsf{tid}], t_2[\mathsf{tid}])$ as long as t_1 and t_2 have the same A attribute value. For a specific constant $c \in dom(A)$, one can define $\mathcal{B}$ to be $\mathsf{eq}_c = \{(c, c)\}$, such that $\sim^A_{\mathsf{eq}_c}$ collects all tuples t in which attribute $t[A]$ is restricted to be the predefined constant c.
(2) $\mathsf{eq}_S = \cup_{c \in S} \mathsf{eq}_c$ for a (finite) set $S \subseteq dom(A)$, *i.e.,* $\sim^A_{\mathsf{eq}_S}$ groups those tuples t in which $t[A] \in S$. Similarly one may define $\mathcal{B}$ as $\mathsf{eq}_{\bar{S}} = (\mathsf{eq}_A \setminus \mathsf{eq}_S)$, such that $\sim^A_{\mathsf{eq}_{\bar{S}}}$ groups tuples t in which $t[A]$ does not belong to S.
(3) When the domain of A is ordered, we may define $\mathcal{B}$ as $\mathsf{eq}_{S(\mathsf{op}\, b)}$ where $S(\mathsf{op}\, b) = \{a \in dom(A) \mid a\, \mathsf{op}\, b\}$, and $\mathsf{op} \in \{<, \leqslant, >, \geqslant, =, \neq\}$; *i.e.,* $\sim^A_{\mathsf{eq}_{S(\mathsf{op}\, b)}}$ groups tuples t such that $t[A] \mathsf{op}\ b$. □

In practice, it is common to compare values of an attribute A based on their similarity rather than equality. In general, a similarity relation $\mathcal{B} \subseteq dom(A) \times dom(A)$ if it is reflexive and symmetric, but not necessarily transitive.

Example 2: Similarity relations are often derived from metrics. Suppose that A is an attribute whose domain is string. We define the similarity relation $\approx^{\mathsf{dl}}_{\theta}$ based on the Damerau-Levenstein metric (with threshold θ):

$$\approx^{\mathsf{dl}}_{\theta} = \{(s, t) \in \mathsf{string} \times \mathsf{string} \mid \text{edit distance}(s, t) \leqslant \theta\}.$$

In fact, any metric such as q-grams or Jaro distance can be turned into a similarity relation (see [10] for a survey on distance functions), from which a comparison relation (*e.g.,* $\sim^A_{\approx^{\mathsf{dl}}_{\theta}}$) can be readily derived. □

The comparison relations defined in terms of equality and similarity relations can be readily extended to a set X of attributes, *e.g.,* $\sim^X_{\mathsf{eq}_X}$ is defined as $\bigcap_{A \in X} \sim_{\mathsf{eq}_A}$, and $\sim^X_{\approx_X}$ as $\bigcap_{A \in X} \sim^A_{\approx_A}$.

A comparison relation $\sim^X_{\mathcal{B}}$ via $(X, \mathcal{B})$ is said to be *decomposable* if there exist $\mathcal{B}_i \subseteq dom(A_i) \times dom(A_i)$ such that $\sim^X_{\mathcal{B}} = \bigcap_{A_i \in X} \sim^{A_i}_{\mathcal{B}_i}$. For instance, the comparison relations $\sim^X_{\mathsf{eq}_X}$ and $\sim^X_{\approx_X}$ given above are clearly decomposable. Nevertheless, not all comparison relations are decomposable, as shown below.

Example 3: Let $X = \{A_1, \ldots, A_k\}$ be a set of ordered attributes, and let $\bar{a} = (a_1, \ldots, a_k)$ and $\bar{b} = (b_1, \ldots, b_k)$ be tuples in $dom(A_1) \times \cdots \times dom(A_k)$. Define $\bar{a}\, \mathsf{lexico}\, \bar{b}$ iff either $a_i = b_i$ for all $i \in [1, k]$ or there exist a $j \in [1, k]$ such that $a_j <_{A_j} b_j$ while $a_i = b_i$ for $i \in [1, j-1]$. Here $<_{A_j}$ denotes an order on A_j. Then the corresponding comparison relation $\sim^X_{\mathsf{lexico}}$ is not decomposable. Indeed, observe that $\sim^X_{\mathsf{lexico}}$ can be written as

$$\Big(\bigcap_{i \in [1,k]} \sim^{A_i}_{\mathsf{eq}}\Big) \cup \bigcup_{j \in [1,k]} \Big(\sim^{A_j}_{<_{A_j}} \cap \bigcap_{i \in [1, j-1]} \sim^{A_i}_{\mathsf{eq}}\Big);$$

in other words, it can be written as a union of decomposable comparison relations, but not as $\bigcap_{i \in [1,k]} \sim^{A_i}_{\mathcal{B}_i}$. This example also shows that comparison relations can be defined in terms of Boolean operations. □

2.2 Quality Improving Dependencies

We next define quality improving dependencies in terms of comparison relations. Let R and R' be two relations in $\mathcal{R}$. Let $X = \{A_1, \ldots, A_k\}$ and $X' = \{A'_1, \ldots, A'_k\}$ be attributes in R and R', respectively, such that A_i and A'_i are compatible for each $i \in [1, k]$; similarly for attributes $Y = \{E_1, \ldots, E_\ell\}$ and $Y' = \{E'_1, \ldots, E'_\ell\}$. Let $\sim^X_{\mathcal{B}_X}$ and $\sim^Y_{\mathcal{B}_Y}$ be comparison relations via $(X, \mathcal{B}_X)$ and $(Y, \mathcal{B}_Y)$, respectively.

A *quality improving dependency* (QID) defined on (R, R') is of the form

$$\varphi \;=\; \big(\sim^X_{\mathcal{B}_X} (R(X), R'(X')) \;\rightarrow\; \sim^Y_{\mathcal{B}_Y} (R(Y), R'(Y'))\big).$$

An instance $\mathcal{I}$ of $\mathcal{R}$ *satisfies* φ, denoted by $\mathcal{I} \models \varphi$, if

$$\forall\, \mathsf{tid} \in I, \mathsf{tid}' \in I' : \sim^X_{\mathcal{B}_X} (\mathsf{tid}, \mathsf{tid}') \;\longrightarrow \sim^Y_{\mathcal{B}_Y} (\mathsf{tid}, \mathsf{tid}'),$$

where I is the instance of R in $\mathcal{I}$, I' is the instance of R' in $\mathcal{I}$ for which attributes of X' and Y' are renamed as their counterparts in X and Y, respectively, *i.e.,* A'_i as A_i for $i \in [1, k]$ and B'_j as B_j for $j \in [1, \ell]$.

When $\sim^X_{\mathcal{B}_X}$ and $\sim^Y_{\mathcal{B}_Y}$ are decomposable as $(\mathcal{B}_1^{A_1}, \ldots, \mathcal{B}_k^{A_k})$ and $(\mathcal{B}_1^{E_1}, \ldots, \mathcal{B}_\ell^{E_\ell})$, respectively, then $\mathcal{I} \models \varphi$ if

$$\forall\, \mathsf{tid} \in I, \mathsf{tid}' \in I' : \bigwedge_{i \in [1,k]} \sim^{A_i}_{\mathcal{B}_i} (\mathsf{tid}, \mathsf{tid}') \;\longrightarrow \bigwedge_{j \in [1,\ell]} \sim^{E_i}_{\mathcal{B}_i} (\mathsf{tid}, \mathsf{tid}').$$

If all comparison relations in a QID φ are decomposable, we say that φ is *decomposable*.

When $R = R'$, $X = X'$ and $Y = Y'$, we say that φ is a *mono-QID*; otherwise we call it a *bi-QID*. In the sequel, we simply write a mono-QID as $R(\sim^X_{\mathcal{B}_X} \rightarrow \sim^Y_{\mathcal{B}_Y})$. We only consider birelational QIDs that relate tuples in one relation R to tuples in another (possibly the same) R'. In practice, two relations often suffice. Nonetheless, one can readily extend QIDs to multiple relations. When developing algorithms for discovering and validating QIDs, and for detecting and correcting errors based on QIDs, it is important to know whether the QIDs under consideration are decomposable or not, and whether one or two relations are required (see Section 4).

3 Capturing Existing Dependency Formalisms with QIDs

As examples of QIDs, we show that a variety of dependencies studied for data quality can be expressed as QIDs, as summarized in Table 3. We first consider formalisms that are expressible as *decomposable mono-*QIDs.

Functional dependencies (FDs) [1]. A functional dependency on a relation R is defined as $R(X \rightarrow Y)$, where X and Y are sets of attributes. An instance I of R satisfies the FD if for any two tuples t_1 and t_2 in I, $t_1[Y] = t_2[Y]$ whenever $t_1[X] = t_2[X]$, *i.e.,* the X attributes of a tuple uniquely determine its Y attributes.

Every FD $R(X \rightarrow Y)$ is a decomposable mono-QID $R(\sim^X_{\mathsf{eq}_X} \rightarrow \sim^Y_{\mathsf{eq}_Y})$, where $\sim^X_{\mathsf{eq}_X}$ is defined in Section 2.

Conditional functional dependencies (CFDs) [14]. CFDs extend FDs by incorporating constant patterns. More specifically, a CFD is defined as $R[(X \rightarrow Y) \mid t_p]$, where $R(X \rightarrow Y)$ is an FD, and t_p is a pattern tuple over $X \cup Y$. For each attribute $A \in X \cup Y$, $t_p[A] \in \mathit{dom}(A)$ or $t_p[A]$ is wildcard, denoted by $\sqcup$. A tuple t in an instance I of R is said to *match* t_p in X, denoted by $t[X] \asymp t_p[X]$, if for each $A \in X$, either $t_p[A] = \sqcup$ and $t[A]$ can be an arbitrary value in $\mathit{dom}(A)$, or $t[A] = t_p[A]$ when $t_p[A]$ is a constant. An instance I of R satisfies the CFD if for any two tuples t_1 and t_2 in I, if $t_1[X] = t_2[X] \asymp t_p[X]$ then also $t_1[Y] = t_2[Y] \asymp t_p[Y]$. For instance, $R[(\mathsf{CC}, \mathsf{zip} \rightarrow \mathsf{street}) \mid (44, \sqcup, \sqcup)]$ is a CFD, which states that when CC (country code) is 44 (indicating UK), zip code uniquely determines street.

Every CFD $R[(X \rightarrow Y) \mid t_p]$ is a decomposable mono-QID $R(\sim^X_{\mathsf{eq}_C} \rightarrow \sim^Y_{\mathsf{eq}_C})$, where $\sim^X_{\mathsf{eq}_C}$ is defined as $\bigcap_{A \in X} \sim^A_{\mathsf{eq}'_A}$. Here $\sim^A_{\mathsf{eq}'_A}$ is $\sim^A_{\mathsf{eq}_A}$ when $t_p[A] = \sqcup$, and $\sim^A_{\mathsf{eq}_c}$ if $t_p[A] = c$ (see Example 1); similarly for $\sim^Y_{\mathsf{eq}_C}$.

Extended CFDs (eCFDs) [6]. This class of dependencies extends CFDs $[R(X \rightarrow Y) \mid t_p]$ such that in the pattern tuples t_p, for each $A \in X \cup Y$, $t_p[A] = S$, $t_p[A] = \bar{S}$ for a finite set of constants $S \subseteq \mathit{dom}(A)$, or $t_p[A] = \sqcup$. We say that $t[X] \asymp t_p[X]$ if for each attribute $A \in X$, $t[A] \in S$ if $t_p[A] = S$, $t[A] \neq S$ if $t_p[A] = \bar{S}$, or $t_p[A] = \sqcup$. The semantics of eCFDs is the same as CFDs except for the revised matching operator.

Dependencies [References]	Quality Improving Dependencies		
	Mono/Bi	decomp.	QIDs (basic comparison relations used)
Functional dependencies (FD) [1]	Mono	Yes	$R(\sim^X_{\mathsf{eq}_X} \rightarrow \sim^Y_{\mathsf{eq}_Y})$ (defined with $\sim_{\mathsf{eq}_A}$)
Conditional FDs (CFD) [14]	Mono	Yes	$R(\sim^X_{\mathsf{eq}_C} \rightarrow \sim^Y_{\mathsf{eq}_C})$ (with $\sim_{\mathsf{eq}_A}$, $\sim_{\mathsf{eq}_a}$)
Extended CFDs (eCFD) [6]	Mono	Yes	$R(\sim^X_{\mathsf{eq}_S} \rightarrow \sim^Y_{\mathsf{eq}_S})$ ($\sim_{\mathsf{eq}_A}$, $\sim_{\mathsf{eq}_S}$,$\sim_{\mathsf{eq}_{\bar{S}}}$)
Similarity FD (SFD) [32, 34, 3, 30]	Mono	Yes	$R(\sim^X_{\approx_X} \rightarrow \sim^Y_{\approx_Y})$ ($\approx_X$, reflexive and symmetric)
Domain dependencies (DDs) [12]	Mono	Yes	$R(\emptyset \rightarrow \sim^A_{\asymp_S})$
Edits [21]	Mono	Yes	$R(\sim^X_{\asymp_S} \rightarrow \sim^{A_k}_{\asymp_{\bar{S}_k}})$ (with $\sim^A_{\asymp_S}$ and $\sim^A_{\asymp_{\bar{S}}}$)
Association rules (AR) [35, 28]	Mono	Yes	$R(\sim^X_{\mathsf{eq}_C} \rightarrow \sim^A_{\mathsf{eq}_a})$ (with $\sim_{\mathsf{eq}_a}$)
Order dependency (OD) [22, 29, 23]	Mono	Yes	$R(\sim^X_{\mathsf{op}_X} \rightarrow \sim^Y_{\mathsf{op}_Y})$ (op: partial orders)
Currency dependencies (CD) [17]	Mono	Yes	$R(\sim^X_{\mathcal{B}_X} \rightarrow \sim^A_{\prec_A})$ ($\sim^A_{\prec_A}$ a currency order)
Binary FDs (BFD) [33]	Bi	Yes	$\sim^X_{\mathsf{eq}_X} (R(X), R'(X')) \rightarrow \sim^Y_{\mathsf{eq}_Y} (R(Y), R'(Y'))$
Matching dependencies (MD) [18, 4]	Bi	Yes	$\sim^X_{\approx_X} (R(X), R'(X')) \rightarrow \sim^Y_{\mathsf{eq}_Y} (R(Y), R'(Y'))$
Editing rules (eR) [19]	Bi	Yes	$\sim^X_{\mathcal{B}_X} (R(X, X_p), R'(X', X_p)) \rightarrow \sim^B_{\mathsf{eq}_B} (R(B), R'(B'))$
Sequential dependencies (SD) [25]	Mono	No	$R(\sim^X_{\mathsf{succ}(<)} \rightarrow \sim^A_{\mathsf{gap}})$

Table 13: Different quality improving dependencies

Each eCFD $[R(X \rightarrow Y) \mid t_p]$ is also a decomposable mono-QID $R(\sim^X_{\mathsf{eq}_S} \rightarrow \sim^Y_{\mathsf{eq}_S})$. Here $\sim^X_{\mathsf{eq}_S}$ is defined as $\bigcap_{A \in X} \sim^A_{\mathsf{eq}'_A}$, where $\sim^A_{\mathsf{eq}'_A}$ is $\sim^A_{\mathsf{eq}_A}$ when $t_p[A] = \sqcup$, $\sim^A_{\mathsf{eq}_S}$ when $t_p[A] = S$, and it is $\sim^A_{\mathsf{eq}_{\bar{S}}}$ when $t_p[A] = \bar{S}$ (see Example 1 for the definitions of $\sim^A_{\mathsf{eq}_S}$ and $\sim^A_{\mathsf{eq}_{\bar{S}}}$); similarly for $\sim^Y_{\mathsf{eq}_S}$.

Similarity functional dependencies (SFDs) [32, 34, 3, 30]. Similarity FDs are a subclass of fuzzy FDs in which equality eq is relaxed to similarity relations that are reflexive and symmetric, but not necessarily transitive. More specifically, for each attribute A in R, a similarity relation $\approx_A$ on $dom(A)$ is defined. A similarity FD $R(X \rightarrow Y)$ is specified the same as an FD. It is interpreted along the same lines as FDs, except that it is based on similarity, and its satisfaction may be a "truth degree" in the range $[0, 1]$ rather than true or false.

Leveraging comparison relations $\sim^X_{\approx_X}$ ($\sim^Y_{\approx_Y}$) defined in Section 2, one can verify that each SFD $R(X \rightarrow Y)$ can be expressed as a decomposable mono-QID $R(\sim^X_{\approx_X} \rightarrow \sim^Y_{\approx_Y})$.

Domain dependencies (DDs)/Edits [12, 21]. A DD [12] is of the form $R(A, S)$, where A is an attribute and S is a set of constants taken from $dom(A)$. It aims to assure that for each tuple t in an instance of R, $t[A] \in S$. An edit e is of the form $R([A_1 : S_1] \times \cdots \times [A_k : S_k])$, where S_i is a set of constants in $dom(A_i)$ [21]. It states that for any tuple t in an instance of R, $t[A_1, \ldots, A_k] \notin S_1 \times \cdots \times S_k$. Observe that edits can be expressed as eCFDs in which pattern tuples contain no wildcards. In contrast not every such eCFD is an edit rule.

To express DDs and edits as QIDs, we use a relation $\asymp_S = \{(a, b) \in dom(A) \times dom(A) \mid a, b \in S\}$. Intuitively, $\asymp_S$ states that elements in S are indistinguishable. We also use $\asymp_{\bar{S}} = dom(A)^2 \setminus \asymp_S$, to identify elements not in S. We define comparison relation $\sim^A_{\asymp_S}$ and $\sim^A_{\asymp_{\bar{S}}}$ in terms of $\asymp_S$ and $\asymp_{\bar{S}}$, respectively.

A DD $R(A, S)$ can be expressed as a decomposable mono-QID $R(\emptyset \rightarrow \sim^A_{\asymp_S})$. An edit $R([A_1 : S_1] \times \cdots \times [A_k : S_k])$ can be specified as a decomposable mono-QID $R(\sim^X_{\asymp_S} \rightarrow \sim^{A_k}_{\asymp_{\bar{S}_k}})$, where $\sim^X_{\asymp_S} = \bigcap_{i \in [1,k-1]} \sim^{A_i}_{\asymp_{S_i}}$.

Association rules with 100% confidence (ARs) [35, 28]. An association rule (AR) is defined by means of an implication $R(([A_1 : a_1], \ldots, [A_k : a_k]) \rightarrow [B : b])$, where A_i and B are attributes, and a_i and b are constants from the corresponding domains. An instance I of R satisfies the AR with 100% confidence if for each tuple t in I, if $t[A_1, \ldots, A_k] = (a_1, \ldots, a_k)$ then $t[B] = b$. Such association rules are a special case of CFDs in which the pattern tuples consist of constants only. Hence, ARs are decomposable mono-QID defined in terms of $\sim^A_{\mathsf{eq}_a}$.

Order dependencies (ODs) [22, 29, 23]. Order dependencies (ODs) are specified as standard FDs in which equality may be replaced by a partial order relations $<, \leq, >, \geq$ on ordered attributes, as well as their complement relations (*e.g.,* $\leq^c$). More specifically, an OD is of the form $R(\tilde{X} \rightarrow \tilde{Y})$ in which $\tilde{X}$ and $\tilde{Y}$ are sets of attributes that are either unmarked (for $=$), or marked with one of the partial order relations. ODs can be expressed as decomposable mono-QIDs defined in terms of $\sim^A_{\mathsf{eq}_A}$ (if the attribute A is unmarked) and $\sim^A_{\mathsf{op}}$ (if A is marked).

Here for $\mathsf{op} \in \{<, \leq, >, \geq, \leq^c\}$, $\sim^A_{\mathsf{op}}$ is defined in terms of relation $\mathsf{op} = \{(a, b) \in \mathit{dom}(A) \times \mathit{dom}(A) \mid a \ \mathsf{op}\ b\}$.

Currency dependencies (CDs) [17]. Currency dependencies (CDs) are constraints of the form

$$\forall t_1, \ldots, t_k : R(\textstyle\bigwedge_{j\in[1,k]}(t_1[\mathsf{eid}] = t_j[\mathsf{eid}] \wedge \psi) \longrightarrow t_u \prec_A t_v),$$

where $u, v \in [1, k]$, $t_u \prec_A t_v$ indicates that $t_v[A]$ is more current than $t_u[A]$; and ψ is a conjunction of predicates of the form $t_j[A]\ \mathsf{op}\ t_h[A]$ with $\mathsf{op} \in \{=, \neq, <, \leq, >, \geq, \prec\}$, $t_j[A] = a$ or $t_j[A] \neq a$ for $a \in \mathit{dom}(A)$. Here eid denotes an entity id attribute, as introduced by Codd [8]. The CD is to assert that when t_i's are tuples pertaining to the same entity, and if condition ψ is satisfied, then tuple t_v must be more up-to-date than t_u in attribute A.

When $k = 2$, CDs can be expressed as decomposable mono-QIDs of the form $R(\sim^X_{\mathcal{B}_X} \rightarrow \sim^A_{\prec_A})$. Here $\sim^A_{\prec_A}$ is a comparison relation defined in terms of relation $\prec_A$, and $\sim^X_{\mathcal{B}_X}$ is an intersection of comparison relations $\sim^E_{\mathsf{op}}$ for $\mathsf{op} \in \{<, \leq, >, \geq, =, \neq\}$, and $\sim^E_{\mathsf{eq}_{S(\mathsf{op}\, a)}}$ for $\mathsf{op} \in \{=, \neq\}$, to express the condition ψ.

There are also *mono*-QIDs that are *not decomposable*.

Sequential dependencies (SDs) [25]. A sequential dependency (SD) is of the form $R(X \rightarrow_g A)$, where X is a set of ordered attributes, A is a numerical (linear ordered) attribute, and $g \subseteq \mathit{dom}(A)$ is an interval. It is to assert that for a predefined permutation π of tuples in an instance I of R that are increasing in X, *i.e.,* $t_{\pi(1)}[X] <_X t_{\pi(2)}[X] <_X \cdots <_X t_{\pi(N)}[X]$, we have that for any $i \in [1, N-1]$, $t_{\pi(i+1)}[A] - t_{\pi(i)}[A] \in g$. A conditional variant of sequential dependencies is studied [25] in which a sequential dependency is to hold only an a subset of tuples in I, where the subset is selected by means of admissible ranges of attribute values in X.

To express SDs as QIDs, we define the following two comparison relations: (1) $\sim^A_{\mathsf{gap}}$ in terms of relation $\mathit{gap} = \{(a, b) \in \mathit{dom}(A) \times \mathit{dom}(A) \mid b - a \in g\}$, which is neither reflexive, symmetric nor transitive; and (2) $\sim^X_{\mathsf{succ}(<)} = \sim^X_{\pi} \setminus (\sim^X_{\pi\_\mathsf{strict}} \cap (\sim^X_{\pi\_\mathsf{strict}})^{\circ})$, where $\sim^X_{\pi}$ is the comparison relation derived from the predefined order π, $\sim^X_{\pi\_\mathsf{strict}} = \sim^X_{\pi} \setminus \sim^X_{\mathsf{eq}}$, and S° denotes the reversal operation on binary relation S; observe that $\sim^X_{\mathsf{succ}(<)}$ is defined as a Boolean combination of comparison relations, and is not decomposable.

An SD $R(X \rightarrow_g A)$ can be readily expressed as a non-decomposable mono-QID $R(\sim^X_{\mathsf{succ}(<)} \rightarrow \sim^A_{\mathsf{gap}})$. In addition, for the conditional extension of SDs given in [25], one can construct an appropriate comparison relation $\sim^X_{\mathsf{range}}$ that restricts $\sim^X_{\mathsf{succ}(<)}$ and groups tuples together that fall in the admissible range.

We next consider *decomposable bi*-QIDs.

Binary FDs (BFDs) [27, 33]. A binary FD is an FD that relates attributes in two different relations. More specifically, a BFD is of the form $R[X] = R'[X'] \rightarrow R[Y] = R'[Y']$ where X, X' and Y, Y' are lists of pairwise compatible attributes in R and R', respectively. For an instance (I, I') of (R, R'), it is to assure that for any pair of tuples $t \in I$ and $t' \in I$, if $t[X] = t'[X']$ then also $t[Y] = t'[Y']$.

We can express BFDs as decomposable bi-QIDs of the form $\sim^X_{\mathsf{eq}_X}(R(X), R'(X')) \rightarrow \sim^Y_{\mathsf{eq}_Y}(R(Y), R'(Y'))$, where $\sim^X_{\mathsf{eq}_X}$ and $\sim^Y_{\mathsf{eq}_Y}$ are defined in terms of $\sim^A_{\mathsf{eq}}$ as given earlier for the case of FDs, but across R and R'.

Matching dependencies (MDs) [18, 13, 4]. Matching dependencies (MDs) were proposed to specify record matching rules in [18]. MDs were originally defined in terms of a dynamic semantics. Here we attempt to redefine MDs as QIDs, and interpret their dynamic semantics in terms of their enforcement strategies in Section 4.

When specified as QIDs, MDs can be viewed as an extension of both SFDs and BFDs. They compare certain attributes of tuples (possibly across different relations) in terms of similarity, and if these attributes are similar enough to each other, then identify some other attributes of the tuples. More specifically, an MD is of the form

$$(\bigwedge_{i\in[1,k]} R[A_i] \asymp_i R'[A'_i]) \longrightarrow (\bigwedge_{j\in[1,\ell]} R[B_j] = R'[B'_j]).$$

For an instance (I, I') of (R, R'), it assure that for any tuples $t \in I$ and $t' \in I'$, if $t[A_i] \asymp_i t'[A'_i]$ for $i \in [1, k]$, where $\asymp_i$ is a similarity relation, then $t[B_j]$ and $t'[B'_j]$ must be identified for $j \in [1, \ell]$, indicated by equality.

MDs can be written as decomposable bi-QIDs of the form $\sim^X_{\approx_X}(R(X), R'(X')) \rightarrow \sim^Y_{\mathsf{eq}_Y}(R(Y), R'(Y'))$, where $\sim^X_{\approx_X}$ and $\sim^Y_{\mathsf{eq}_Y}$ are given earlier for SFDs and FDs, respectively.

Editing rules (eRs) [19]. Editing rules (eRs) were introduced for correcting errors in a data relation (R) with accurate values from a master relation (R'). An eR is of the form $[(R(X), R'(X') \rightarrow R(B), R'(B')) \mid t_p[X_p]]$, where X and X' are lists of compatible attributes in schemas R and R', respectively, and $|X| = |X'|$. Moreover, B is an attribute of R but $B \notin X$, and B' is an attribute of R' but is not in X'. Finally, t_p is a pattern tuple over a set of distinct attributes X_p in R, where for each $A \in X_p$, $t_p[A]$ is either $\sqcup$ or a constant a drawn from $dom(A)$, similar to pattern tuples in CFDs. Intuitively, for an instance (I, I') of (R, R') and for any tuple $t \in I$, if $t[X_p] \asymp t_p[X_p]$, where $\asymp$ is the match relation defined for CFDs, and moreover, if there exists $t' \in I'$ such that $t[X] = t'[X']$, then we update $t[B]$ by letting $t[B] := t'[B]$, *i.e.,* taking the value $t'[B]$ from the master tuple t'.

Like MDs, eRs also have a dynamic semantics. Nevertheless, we can express eRs as QIDs in terms of a standard static semantics, by using equality to indicate updates. More specifically, an eR can be written as decomposable bi-QIDs of the form $\sim^X_{\mathcal{B}_X}\ (R(X, X_p), R'(X', X_p)) \rightarrow \sim^B_{\mathsf{eq}_B}\ (R(B), R'(B'))$, where $\sim^X_{\mathcal{B}_X}$ is defined as $\bigcap_{A \in X} \sim^A_{\mathsf{eq}_A} \cap \bigcap_{E \in X_p} \sim^E_{\mathsf{eq}'_E}$. Here $\sim^A_{\mathsf{eq}_A}$ is the equality relation on attributes of tuples across R and R', and $\sim^E_{\mathsf{eq}'_E}$ is defined along the same lines as its counterpart for CFDs on R attributes only, to enforce patterns.

4 Data Quality Problems

We have proposed QIDs as data quality rules, to catch and fix errors. Below we address several central data quality issues associated with QIDs, and provide an overview of recent advances in the study of these issues.

Discovering QIDs. To use QIDs as data quality rules, it is necessary to have techniques in place that can *automatically discover* QIDs from real-life data. Indeed, it is often unrealistic to rely solely on human experts to design data quality rules via an expensive manual process. This suggests that we settle the following problem.

Given a database $\mathcal{I}$, *the profiling problem* is to find a *minimal cover* of all QIDs that $\mathcal{I}$ satisfies, *i.e.,* a non-redundant set of QIDs that is logically equivalent to the set of all QIDs that hold on $\mathcal{I}$. We want to learn informative and interesting data quality rules from (possibly dirty) data, and prune away insignificant rules.

Several algorithms have been developed for discovering FDs (*e.g.,* [26]), CFDs [7, 15, 24] and MDs [31]. These algorithms allow a considerable pruning of the search space when QIDs are decomposable.

Validating QIDs. A given set Σ of QIDs, either automatically discovered or manually designed, may be inconsistent itself. In light of this we have to find those QIDs from Σ that are consistent and non-redundant, to be used as data quality rules. These highlight the need for studying the following classical problems for dependencies.

The *satisfiability problem* for QIDs is to determine, given a set Σ of QIDs defined on a relational schema $\mathcal{R}$, whether there exists a nonempty instance $\mathcal{I}$ of $\mathcal{R}$ such that $\mathcal{I} \models \Sigma$, *i.e.,* $\mathcal{I} \models \varphi$ for *each* QID $\varphi \in \Sigma$.

The *implication problem* for QIDs is to decide, given a set Σ of QIDs and a single QID φ defined on a relational schema $\mathcal{R}$, whether Σ entails φ, *i.e.,* whether for all instances $\mathcal{I}$ of $\mathcal{R}$, if $\mathcal{I} \models \Sigma$ then $\mathcal{I} \models \varphi$. Effective implication analysis allows us to remove redundancies and deduce new data quality rules from Σ.

These problems have been studied for FDs [1], CFDs [14], SFDs [3], edits [21], ODs [29], BFDs [27], MDs [13] and editing rules [19]. These issues are, however, nontrivial. When CFDs are concerned, for instance, both problems are intractable [14]. This calls for the study of effective heuristic algorithms for their analyses.

Error detection (localization). After a consistent set of data quality rules is identified, the next question concerns how to effectively catch errors in a database by using the rules. Given a set Σ of QIDs and a database $\mathcal{I}$, we want to *detect inconsistencies* in $\mathcal{I}$, *i.e.,* to find all tuples and attributes in $\mathcal{I}$ that violate some QID in Σ. Error detection can be done more efficiently for decomposable mono-QIDs due to the absence of intra-relational joins.

Error detection methods have been developed for centralized databases [14] and distributed data [16], based on CFDs. Nevertheless, it is rather challenging to precisely localize errors in the data. Dependencies typically help us catch a pair of tuples (or attributes) that are inconsistent with each other, but fall short of telling us which attribute is erroneous. As an example, consider a simple CFD $R[(\mathsf{CC}, \mathsf{AC} \rightarrow \mathsf{city}) \mid (44, 131, \mathrm{Edi})]$, which states that in the UK (CC = 44), when the area code (AC) is 131, then the city must be Edinburgh. A tuple t: (CC = 44.

AC = 131, city = Ldn) violates this CFD: AC is 131 but the city is London. The CFD finds that $t[\mathsf{CC}]$, $t[\mathsf{AC}]$ or $t[\mathsf{city}]$ is incorrect, *i.e.,* an error is present in the tuple. However, it does not tell us which of the three attributes is wrong and to what value it should be changed. To rectify this problem, we need to adopt a stronger semantics when enforcing QIDs for data imputation, as will be seen below when we discuss editing rules [19].

Data repairing (data imputation). After the errors are detected, we want to automatically correct the errors and make the data consistent. This is known as *data repairing* [2]. Given a set Σ of QIDs and a database instance $\mathcal{I}$ of $\mathcal{R}$, it is to find a candidate *repair* of $\mathcal{I}$, *i.e.,* another instance $\mathcal{I}'$ of $\mathcal{R}$ such that $\mathcal{I}' \models \Sigma$ and $\mathsf{cost}(\mathcal{I}, \mathcal{I}')$ is minimum. Here $\mathsf{cost}()$ is a metric defined in terms of (a) the distance between $\mathcal{I}$ and $\mathcal{I}'$ and (b) the accuracy of attribute values in $\mathcal{I}$ (see, *e.g.,* [5, 9]). When a value v in $\mathcal{I}$ is changed to v' in $\mathcal{I}'$ in data repairing, the more accurate the original value v is and the more distant the new value v' is from v, the higher the cost of the change is. That is, we want to avoid changing accurate values in $\mathcal{I}$, and want the repair $\mathcal{I}'$ to *minimally differ* from $\mathcal{I}$.

No matter how important, the data repairing problem is nontrivial: it is intractable even when only a fixed set of FDs is considered [5]. In light of this, several heuristic algorithms have been developed, to repair data based on different strategies for enforcing FDs, CFDs, MDs and eRs [5, 9, 19, 20].

These QID enforcement strategies can be uniformly characterized in terms of a revised chase process (see, *e.g.,* [1] for details about chase). More specifically, assume that for each attribute $A \in \mathcal{R}$, a *fixing function* $\sigma_A : \mathit{dom}(\mathsf{tid}) \to \mathit{dom}(A)$ is defined. Given a tuple t in $\mathcal{I}$ identified by tid, the function is used to change the values of $t[A]$ to the value $\sigma_A(\mathsf{tid})$. Let $\bar{\sigma}$ denote a set of fixing functions σ_A, one for each $A \in \mathcal{R}$. Consider a QID $\varphi = (\sim^X_{\mathcal{B}_X} (R(X), R'(X')) \to \sim^Y_{\mathcal{B}_Y} (R(Y), R'(Y')))$, and two tuples t_1, t_2 in $\mathcal{I}$. We say that an instance $\mathcal{I}'$ of $\mathcal{R}$ is the *result of enforcing φ on t_1 and t_2 in $\mathcal{I}$ using $\bar{\sigma}$*, denoted by $\mathcal{I} \to^{\bar{\sigma}}_{\varphi,(t_1,t_2)} \mathcal{I}'$, if

- in $\mathcal{I}$: we have that $\sim^X_{\mathcal{B}_X} (t_1[\mathsf{tid}], t_2[\mathsf{tid}])$ holds, whereas $\sim^Y_{\mathcal{B}_Y} (t_1[\mathsf{tid}], t_2[\mathsf{tid}])$ does not hold;
- in $\mathcal{I}'$: $t_1[A_i] = \sigma_{A_i}(t_1[\mathsf{tid}])$, $t_2[A_i] = \sigma_{A_i}(t_2[\mathsf{tid}])$ for all $A_i \in X \cup Y$, and both $\sim^X_{\mathcal{B}_X} (t_1[\mathsf{tid}], t_2[\mathsf{tid}])$ and $\sim^Y_{\mathcal{B}_Y} (t_1[\mathsf{tid}], t_2[\mathsf{tid}]))$ hold; and finally,
- $\mathcal{I}$ and $\mathcal{I}'$ agree on every other tuple and attribute value.

For a set Σ of QIDs, we say that $\mathcal{I}'$ is a Σ-*repair* of $\mathcal{I}$ if there is a finite *chasing sequence* $\mathcal{I}_0 \ldots, \mathcal{I}_k$ such that

$$\mathcal{I}_0 = \mathcal{I} \longrightarrow^{\bar{\sigma}^1}_{\varphi^1,(t^1_1,t^1_2)} \mathcal{I}_1 \longrightarrow^{\bar{\sigma}^2}_{\varphi^2,(t^2_1,t^2_2)} \cdots \longrightarrow^{\bar{\sigma}^k}_{\varphi^k,(t^k_1,t^k_2)} \mathcal{I}_k = \mathcal{I}',$$

and furthermore, $\mathcal{I}' \models \Sigma$, where $\varphi^i \in \Sigma$ and $\bar{\sigma}^i$ is a set of fixing functions for $i \in [1, k]$. A Σ-repair $\mathcal{I}'$ of $\mathcal{I}$ is minimal if $\mathsf{cost}(\mathcal{I}, \mathcal{I}')$ is minimum. We say that an enforcement strategy of QIDs is *Church-Rosser* if for any set Σ of QIDs, all Σ-repairs obtained via different finite chasing sequences result in the same instance of $\mathcal{R}$.

A repairing algorithm is essentially an implementation of the "chase"-process, determined by (a) how fixing functions are defined, (b) in what order the QIDs in Σ are enforced, and (c) for each QID φ, how it is enforced on the tuples in $\mathcal{I}$ that violate φ. Below we outline how existing algorithms compute Σ-repairs by enforcing QIDs. Here decomposable QIDs again simplify the chase process because fixing functions can be defined independently for different attributes in a decomposable QID.

(Conditional) Functional dependencies [5, 9]. Given a set Σ of FDs and an instance $\mathcal{I}$, the repairing algorithm of [5] computes a Σ-repair by enforcing the FDs in Σ one by one, as follows. (1) For each FD $\varphi = R(\sim^X_{\mathsf{eq}_X} \to \sim^B_{\mathsf{eq}_B})$ in Σ and each tuple t in $\mathcal{I}$, it finds the set $V_{\varphi,t}$ of all tuples t' in $\mathcal{I}$ such that $\sim^X_{\mathsf{eq}_X} (t[\mathsf{tid}], t'[\mathsf{tid}])$ but not $\sim^Y_{\mathsf{eq}_Y} (t[\mathsf{tid}], t'[\mathsf{tid}])$. (2) It defines fixing functions $\bar{\sigma}$ such that for all $B \in Y$ and for all $t' \in V_{\varphi,t} \cup \{t\}$, $\sigma_B(t[\mathsf{tid}])$ is the same constant c, which is a value drawn from $\{t''[B] \mid t'' \in V_{\varphi,t} \cup \{t\}\}$ such that the total cost of changing $t'[B]$ to c is minimum. That is, it enforces φ on all tuples in $V_{\varphi,t}$ consecutively, by changing the values of their attributes in the right-hand side of φ to the same value. Furthermore, in order for the repairing process to terminate, a *hard constraint* is imposed, which requires that once σ_B identifies attributes of a pair of tuples, these attributes remain identified throughout the repairing process. Note that this still allows adjustments to the values of those attributes at a later stage, *i.e.,* different fixing functions $\bar{\sigma}^i$ may be used in the process. This enforcement strategy guarantees to find a Σ-repair for a set of FDs, but is not Church-Rosser.

This is extended by the algorithm of [9] for CFDs, in which fixing functions allow attribute values to be changed to a constant specified by a CFD, provided that the hard constraints imposed so far is not violated. For instance, tuple $t : (\mathsf{CC} = 44, \mathsf{AC} = 131, \mathsf{city} = \text{Ldn})$ can be changed to $(\mathsf{CC} = 44, \mathsf{AC} = 131, \mathsf{city} = \text{Edi})$ as specified by CFD $R[(\mathsf{CC}, \mathsf{AC} \to \mathsf{city}) \mid (44, 131, \text{Edi})]$, unless Ldn was assigned earlier by a fixing function σ_{city}. In the latter case, σ_{city} will not make any change to the right-hand side attribute, but instead a fixing function on the left-hand side attributes is applied; *e.g.,* t could be changed to $(\mathsf{CC} = 44, \mathsf{AC} = 020, \mathsf{city} = \text{Ldn})$ by σ_{AC}. Termination is also assured, but the enforcement strategy is not Church-Rosser.

Matching dependencies [4, 18]. MDs were proposed in [18] to infer matching rules. An enforcement strategy was introduced in [4] for repairing data with MDs. It enforces a set Σ of MDs one by one on an instance $\mathcal{I}$ as follows. Given an MD $\varphi = \sim^{X}_{\approx_X} (R(X), R'(X')) \to \sim^{Y}_{\mathsf{eq}_Y} (R(Y), R'(Y'))$, for each $B \in Y$, it employs matching functions $m_B : \mathit{dom}(B) \times \mathit{dom}(B) \to \mathit{dom}(B)$ that are idempotent ($m_B(b, b) = b$), commutative ($m_B(b, b') = m_B(b', b)$) and associative ($m_B(b, m_B(b', b'')) = m_B(m_B(b, b'), b'')$). For a pair (t_1, t_2) of tuples in $\mathcal{I}$ that violate φ, it defines fixing functions σ_B such that $\sigma_B(t_1[\mathsf{tid}]) = \sigma_B(t_2[\mathsf{tid}]) = m_B(t_1[B], t_2[B])$, and hence, computes $\mathcal{I} \to^{\bar{\sigma}}_{\varphi,(t_1,t_2)} \mathcal{I}'$, where $\bar{\sigma}$ consists of σ_B for all $B \in Y$. It is shown in [4] that m_B imposes a bounded lattice-structure on $\mathit{dom}(B)$ and that $\mathcal{I}'$ is an instance that dominates $\mathcal{I}$ in the corresponding bounded lattice on instances of $\mathcal{R}$. From this follows the termination of the process. The strategy is not Church-Rosser.

Employing both CFDs and MDs, a data repairing algorithm was developed in [20], with fixing functions defined for CFDs and MDs as above. It enforces a set of CFDs and MDs by interleaving data repairing and record matching operations; by leveraging their interaction, the algorithm improves the accuracy of repairs generated.

Editing rules [19]. As remarked earlier, dependencies typically only detect the presence of errors, but do not locate precisely what attributes are wrong and how to fix the errors. As a result, repairing algorithms often fail to correct errors and worse still, may even introduce new errors in the repairing process. To rectify this problem, a repairing strategy was proposed in [19] that aims to find a certain fix of a tuple, *i.e.,* a fix that is guaranteed correct, by using editing rules (eRs) and master data. Consider an eR $\varphi = \sim^{X}_{\mathcal{B}_X} (R(X, X_p), R'(X', X_p)) \to \sim^{B}_{\mathsf{eq}_B} (R(B), R'(B'))$, a tuple t_1 in a data instance I of R, and a tuple t_2 in a master relation of schema R' that provides consistent and correct data values. It defines a fixing function σ_B that equalizes $t_1[B]$ and $t_2[B]$ by always setting $\sigma_B(t_1[\mathsf{tid}])$ and $\sigma_B(t_2[\mathsf{tid}])$ to the master value $t_2[B]$. Furthermore, it allows $\mathcal{I} \to^{\sigma_B}_{\varphi,(t_1,t_2)} \mathcal{I}'$ by enforcing φ only if (a) $t_1[X, X_p]$ has been assured correct (*i.e.,* validated), either by users or via inference from fixes generated earlier; and (b) enforcing different eRs on t_1 and master tuples yields the same repair. In addition, after an attribute $t_1[B]$ is updated, it is no longer changed since it is already validated.

For instance, tuple $t : (\mathsf{CC} = 44, \mathsf{AC} = 131, \mathsf{city} = \text{Ldn})$ is changed to $(\mathsf{CC} = 44, \mathsf{AC} = 131, \mathsf{city} = \text{Edi})$ only if (a) $t[\mathsf{CC}, \mathsf{AC}, \mathsf{city}]$ is already validated, and (b) for all eRs φ in a given set Σ and all master tuples t' in I', either φ does not apply to t and t', or φ requires that $t[\mathsf{city}]$ and $t'[\mathsf{city}]$ be equalized and moreover, $t'[\mathsf{city}] = \text{Edi}$.

From these it follows that the enforcement strategy guarantees the termination of repairing processes. Furthermore, it is per definition Church-Rosser and assures that the fixes generated are certain.

These algorithms are just examples of the revised chase. We expect that different QID enforcement strategies could be developed along the same lines, and yield data repairing algorithms for a wider range of applications.

Acknowledgments. Fan is supported in part by the RSE-NSFC Joint Project Scheme, an IBM scalable data analytics for a smarter planet innovation award, and the National Basic Research Program of China (973 Program) 2012CB316200.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, 1999.

[3] R. Belohlávek and V. Vychodil. Data dependencies in Codd's relational model with similarities. In *Handbook of Research on Fuzzy Information Processing in Databases*. IGI Global, 2008.

[4] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *ICDT*, 2011.

[5] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 2005.

[6] L. Bravo, W. Fan, F. Geerts, and S. Ma. Increasing the expressivity of conditional functional dependencies without extra complexity. In *ICDE*, 2008.

[7] F. Chiang and R. Miller. Discovering data quality rules. In *VLDB*, 2008.

[8] E. F. Codd. Extending the database relational model to capture more meaning. *TODS*, 4(4):397–434, 1979.

[9] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, 2007.

[10] M. M. Deza and E. Deza. *Encyclopedia of Distances*. Springer, 2009.

[11] W. W. Eckerson. Data quality and the bottom line: Achieving business success through a commitment to high quality data. The Data Warehousing Institute, 2002.

[12] R. Fagin. A normal form for relational databases that is based on domains and keys. *TODS*, 6:387–415, 1981.

[13] W. Fan, H. Gao, X. Jia, J. Li, and S. Ma. Dynamic constraints for record matching. *The VLDB Journal*, 2011.

[14] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(2), 2008.

[15] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *TKDE*, 23(5):683–698, 2011.

[16] W. Fan, F. Geerts, S. Ma, and H. Müller. Detecting inconsistencies in distributed data. In *ICDE*, 2010.

[17] W. Fan, F. Geerts, and J. Wijsen. Determining the currency of data. In *PODS*, 2011.

[18] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *PVLDB*, 2(1):407–418, 2009.

[19] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *PVLDB*, 3(1):173–184, 2010.

[20] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Interaction between record matching and data repairing. In *SIGMOD*, 2011.

[21] I. P. Fellegi and D. Holt. A systematic approach to automatic edit and imputation. *Journal of the American Statistical Association*, 71(353):17–35, 1976.

[22] S. Ginsburg and R. Hull. Order dependency in the relational model. *TCS*, 26(1-2):149 – 195, 1983.

[23] S. Ginsburg and R. Hull. Sort sets in the relational model. *J. ACM*, 33:465–488, 1986.

[24] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. In *VLDB*, 2008.

[25] L. Golab, H. J. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 2(1):574–585, 2009.

[26] Y. Huhtala, J. Kärkk ainen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[27] E. Komissartschik. Restructuring and dependencies in databases. In *MFDBS*, 1989.

[28] M. Luxenburger. Implications partielles dans un contexte. *Mathématiques, Informatique et Sciences Humaines*, 29(113):35–55, 1991.

[29] W. Ng. Ordered functional dependencies in relational databases. *Inf. Syst.*, 24(7):535 – 554, 1999.

[30] K. V. S. V. N. Raju and A. K. Majumdar. Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. *TODS*, 13:129–166, 1988.

[31] S. Song and L. Chen. Discovering matching dependencies. In *CIKM*, 2009.

[32] M. I. Sözat and A. Yazici. A complete axiomatization for fuzzy functional and multivalued dependencies in fuzzy database relations. *Fuzzy Sets Syst.*, 117:161–181, 2001.

[33] J. Wang. Binary equality implication constraints, normal forms and data redundancy. *Inf. Process. Lett.*, 101(1):20–25, 2007.

[34] A. Yazici, E. Gocmen, B. Buckles, R. George, and F. Petry. An integrity constraint for a fuzzy relational database. In *FUZZ*, 1993.

[35] M. J. Zaki. Mining non-redundant association rules. *Data Min. Knowl. Discov.*, 9(3):223–248, 2004.

Towards a Domain Independent Platform for Data Cleaning

Arvind Arasu Surajit Chaudhuri Zhimin Chen Kris Ganjam Raghav Kaushik Vivek Narasayya
Microsoft Research
{arvinda,surajitc,zmchen,krisgan,skaushi,viveknar}@microsoft.com

Abstract

We present a domain independent platform for data cleaning developed as part of the Data Cleaning project at Microsoft Research. Our platform consists of a set of core primitives and design tools that allow a programmer to develop sophisticated data cleaning solutions with minimal programming effort. Our primitives are designed to allow rich domain and application specific customizations and can efficiently handle large inputs. Our data cleaning technology has had significant impact on Microsoft products and services and has been successfully used in several real-world data cleaning applications.

1 Introduction

Modern enterprises rely on sophisticated data-driven *Business Intelligence (BI)* technologies [6] for their decision making. An integral component of the BI architecture is the *data warehouse*, where data from a variety of sources is aggregated prior to analysis. Similar data aggregation also occurs in offline data processing pipelines of search verticals such as Bing Maps and Bing Shopping [5]. The data provided by sources can have data quality issues; for example, the data can have missing values and typographical errors. Further, different sources can use different representations for the same real-world entity. For example, the state *Washington* can be represented using `Washington` in one source and `WA` in another. It is important to fix such errors and reconcile representational inconsistencies since they can adversely affect the quality of data analysis (in BI) and search (in verticals such as Bing Maps). The process of fixing errors and inconsistencies in data is often referred to as *data cleaning*. Since integration into a warehouse typically involves large data volumes, performance and scale are important considerations in data cleaning.

Data cleaning encompasses a variety of high-level *tasks* such as *record matching*, *deduplication*, *segmentation*, and *null-filling* [16, 11]. The goal of record matching and deduplication is to identify *matching* records, defined to be records that correspond to the same real-world entity. The output of record matching is pairs of matching records while the output of deduplication is clusters of matching records. The goal of segmentation is to extract structured records from unstructured text, e.g., extracting details such as street name, city, and postal code from an address string. The goal of null-filling is to identify missing values in records, e.g., identifying a missing postal code using information from city and state. As we describe later, these data cleaning tasks are related to each other; e.g., an implementation of record matching might use segmentation as a sub-module.

Data cleaning is applicable in a wide variety of domains such as addresses, products, health records, publications and citations, and music meta-data. Most current commercial solutions can be viewed as *verticals* and cater

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

to a single domain. For example, Trillium [18] provides data cleaning functionality only for address domain. The aspirational goal of our Data Cleaning project at Microsoft Research [17] (which has been ongoing since 2000) is to design and develop a *domain independent, horizontal platform* for data cleaning, such that vertical solutions such as the functionality provided by Trillium can be developed over the platform with little programming effort. The basis for this goal is the observation that some fundamental concepts such as textual similarity are part of most data cleaning solutions across different domains, and our platform is designed to capture and support these concepts.

Our current platform, shown in Figure 1, consists of a set of core *primitives* and a set of *design tools*. A programmer develops a data cleaning solution on our platform by composing and customizing one or more primitives, and she can leverage the design tools to identify a good way to customize and compose the primitives. Our primitives include a customizable notion of textual similarity along with an efficient lookup operation based on this similarity called *Fuzzy Lookup*, a clustering primitive called *Fuzzy Grouping*, and a parsing (segmentation) primitive. We have designed our primitives so that they can be customized to a specific domain—e.g., we can customize Fuzzy Lookup to recognize that `Bob` and `Robert` are synonymous while matching people names. Further, our primitives have been designed for high performance and can handle large inputs and complex customizations efficiently. While other research efforts have focused on horizontal approaches to data cleaning (see [11] for a survey), one important differentiator of our approach is the focus on customizability and performance.

Our design tools include a suite of *learning tools* to help a programmer configure and customize a data cleaning solution using the primitives available in our platform. The learning tools rely mostly on labeled examples to navigate the space of possible configurations. Our platform also includes a *Data Profiling* tool that assesses data quality of a source and identifies properties such as keys and functional dependencies satisfied by data. As noted earlier, functional dependencies can be useful for null-filling.

Our data cleaning technology has had significant product impact within and outside of Microsoft [17]: (1) Fuzzy Lookup and Fuzzy Grouping were shipped as part of SQL Server Integration Services (SSIS), Microsoft SQL Server ETL system, in 2005, and they were the first data cleaning technology to appear in SSIS. (2) In the aftermath of Hurricane Katrina, Fuzzy Lookup was used to develop a matching solution that allowed matching the Government's evacuee database with the database of missing people, thereby helping with the search and identification of missing people. Similar solutions have been developed in recent years, e.g., by Google [13]. (3) Data Profiling tool was shipped as part of SSIS 2008. (4) The Bing maps service currently uses Fuzzy Lookup to match a user query against a large database (containing tens of millions of records) of landmarks and locations. Bing maps service has stringent performance requirements and Fuzzy Lookup currently has an average latency of 3 ms per match query. (5) The group within Microsoft that performs *master data management* of the Microsoft's customers currently uses a solution based on Fuzzy Lookup for record matching over customer data. This solution replaced an earlier commercial third party solution after their evaluations showed that it had better performance and quality. (6) The Bing shopping team currently uses the segmentation primitive as part of its offline data cleaning pipeline before product data from various sources is loaded into its warehouse. (7) Fuzzy lookup is also available as an add-in for Microsoft Excel [12].

This paper provides an overview of our data cleaning primitives and design tools and highlights key design decisions. In particular, we focus on Fuzzy Lookup (Section 2) and provide a brief overview of segmentation (Section 3) and design tools (Section 4). We emphasize that this paper is not a general overview of data cleaning and we refer the reader to [11] for an extensive survey. An overview of our project as of 2006 appears in [8].

2 Fuzzy Lookup

The same real world entity could be represented in different ways, e.g., due to typographical errors, differences in conventions, and missing attributes. For example, the real world organization `Microsoft` could be represented both as ⟨`Microsoft Corp, 1 Microsoft Way, Redmond, WA, 98052`⟩ and ⟨`Microsoft`

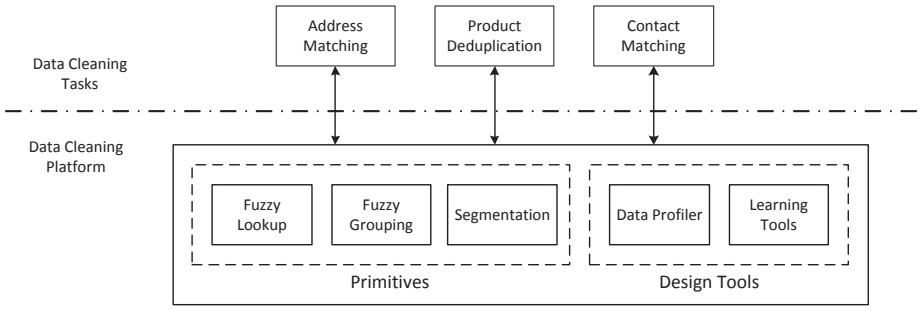


Figure 1: Data cleaning primitives and design tools.

`Corporation, One Microsoft Way, Redmnd, Wash, 98052`⟩. The goal of a record matching task is to identify pairs of matching records that represent the same real-world entity.

At the core of record matching is the notion of a *similarity* function that given two strings returns a numeric value (typically between 0 and 1) quantifying their similarity, a higher value indicating greater similarity. We describe our similarity function in Section 2.1. Fuzzy Lookup supports the following functionality—given a *reference table* R of strings, a query string s, and a threshold parameter θ, find all strings in R whose similarity with s is greater than or equal to θ. To support efficient lookups, Fuzzy Lookup builds an *index* over R. The index structure and lookup algorithm are discussed in Section 2.2.

2.1 Similarity Function

The standard way in which similarity is computed is to use the *textual similarity* between strings using functions such as edit distance, Hamming distance, and Jaccard similarity [11]. Textual similarity is a strong indicator of string similarity. Indeed, the first version of our Fuzzy Lookup primitive [7] relied exclusively on textual similarity. However, in course of our experience in using Fuzzy Lookup with various scenarios including the Hurricane Katrina scenario described above, we discovered the need for customizing the similarity function. For example, there are many cases where strings that are syntactically far apart can still represent the same real-world object. This happens in a variety of settings—street names (the street name `SW 154th Ave` is also known as `Lacosta Dr E` in the postal area corresponding to zipcode 33027), the first names of individuals (`Robert` can also be referred to as `Bob`), conversion from strings to numbers (`second` to `2nd`) and abbreviations (`Internal Revenue Service` being represented as `IRS`). Therefore we modified the design of our similarity function to start with a textual similarity function and extended it to allow a rich customization that enables it to match strings that are syntactically far apart. The current version of Fuzzy Lookup which is used, e.g., in Bing Maps, supports such customizations. We first discuss the core measure of textual similarity followed by the various customizations we expose.

2.1.1 Set Similarity

As our core measure of textual similarity, we use Jaccard similarity. We model a string as a *set of tokens*. The Jaccard similarity between two sets is the size of their intersection as a fraction of the size of the union. The Jaccard Similarity between the two strings `University Avenue, Seattle, WA` and `University Ave, Seattle, WA` tokenized at word boundaries is 3/5. Our rationale for choosing Jaccard similarity is that in addition to being an interesting similarity function in its own right, we can also reduce other similarity functions such as edit distance to it. The intuition is as follows: consider tokenizing a string into it's q-grams. All q-grams that are "far away" from the place where the edits take place must be identical. Hence, a small edit distance implies large Jaccard similarity between the sets of q-grams. For example, consider the strings

Microsoft and Mcrosoft; their edit distance is 1 and the Jaccard similarity between their 1-grams is 8/9. The above relationship between edit distance and Jaccard similarity has been formalized in prior work [14].

In general, we can customize Jaccard similarity by specifying how we tokenize a string. For example, we could tokenize a string into a set of words or a set of q-grams or design language specific tokenizations. We also draw upon the notion of *token weighting* present in previous work on Information Retrieval (IR). Jaccard similarity can be extended to handle token weights by computing a weighted intersection and union. We can use token weights to further customize Jaccard similarity. For example, by assigning higher weights to the tokens Microsoft and Oracle and lower weights to Corporation and Corp, we can ensure that the similarity between the pair (Microsoft Corporation, Microsoft Corp) is greater than that between the pair (Microsoft Corporation, Oracle Corporation). Intuitive weights could be assigned using the IR notion of inverse document frequency (idf).

2.1.2 Transformation Rules

As noted above, there are many cases (e.g., the first names of individuals where Robert can also be referred to as Bob) where strings that are syntactically far apart represent the same real-world object. We posit that it is impractical to expect a generic similarity function to be cognizant of variations such as those of the above examples since they are highly domain-dependent. Rather the matching framework has to be customizable to take the variations as an *explicit input*. Prior work addresses this issue primarily using *token standardization* [16, 11]. The idea is to pre-process the input data so that all variations of a string are converted into a uniform (standard) format. Thus, for example, the input could be pre-processed so that all occurrences of Bob are converted to Robert. However, this approach is inadequate since it does not capture variations that are not equivalences; for example a first initial (such as J) can be expanded in multiple ways (John, Jeff, etc.) that are clearly not equivalent [1].

We therefore introduce the notion of *transformation rules (transformations)*. A transformation rule consists of a triplet $(context, lhs \rightarrow rhs)$ where each of *context*, *lhs*, and *rhs* is a string. (Transformations with empty context are represented $lhs \rightarrow rhs$.) Examples of transformations are: (Bob $\rightarrow$ Robert) and (33027, SW 154th Ave $\rightarrow$ Lacosta Dr E). We now describe how a string s can be transformed given a set of transformation rules T. A rule $(context, lhs \rightarrow rhs)$ can be applied to s if both *context* and *lhs* are substrings of s; the result of applying the rule is the string s' obtained by replacing the substring matching *lhs* with *rhs*. We can apply any number of transformations one after another. However, a token that is generated as the result of applying a transformation cannot participate in any subsequent transformation.

Example 1: We can use the transformation (Drive $\rightarrow$ Dr) to generate the string Lacosta Dr E from the string Lacosta Drive E. However, we cannot further convert Lacosta Dr E to Lacosta Doctor E using the transformation (Dr $\rightarrow$ Doctor).

The set of strings generated by s is the set of all strings obtained by applying zero or more transformations to s. We combine set similarity with transformations to obtain an overall similarity function. Given a set of transformations T, the similarity between strings s_1 and s_2 *under* T is defined to be the *maximum* Jaccard similarity among all pairs s_1' and s_2' respectively generated by s_1 and s_2 using T. We illustrate the above definition through the following example.

Example 2: Consider the strings SW 154th Ave Florida 33027 and Lacosta Dr E FL 33027 and the transformations (FL $\rightarrow$ Florida) and (33027, SW 154th Ave $\rightarrow$ Lacosta Dr E). Each of the strings derives two strings as shown in Figure 2. The overall similarity is the maximum Jaccard similarity among the four pairs of derived strings which in this instance is 1 since both of the above strings generate the same string Lacosta Dr E Florida 33027.

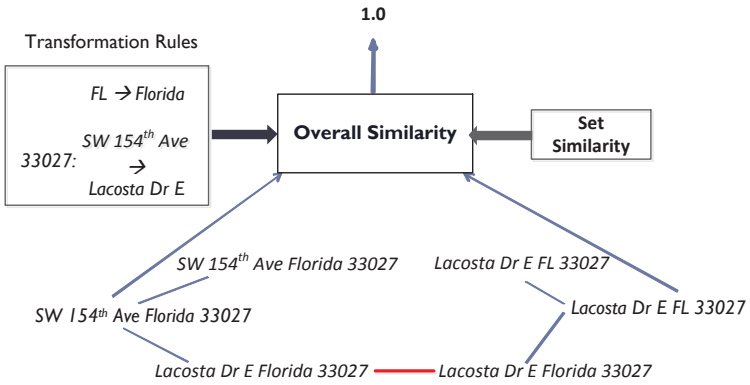


Figure 2: Combining Set Similarity With Transformations.

In general, we allow different sets of the transformations for the reference table and to the input query. Although we assume that the set of transformations is an input, they need not be provided in the form of a table. For example, we could handle transformations obtained by concatenating words (e.g., `Disney Land` → `DisneyLand`) programmatically by generating them dynamically at query time. Since the number of possible word concatenations is large, generating them dynamically may be preferable to materializing them ahead of time. Similarly, we could programmatically capture word level edits (e.g., `Mcrosoft`→`Microsoft`) and abbreviations (e.g., `J`→`Jeff`). Fuzzy lookup provides an interface where we can plug in user-defined transformations. We next discuss how we address the efficiency challenges posed by our notion of similarity.

2.2 Efficient Lookups under the similarity function

Recall that Fuzzy Lookup provides the following functionality—given a *reference table* of strings, R, a set of transformation rules, T, a query string s, and a threshold parameter θ, find all strings in R whose Jaccard similarity under T with s is $\geq \theta$. To support efficient lookups, Fuzzy Lookup builds an *index* over R. The indexing algorithm is based on the idea of *signature schemes* [3]. A signature scheme, given a string, generates a set of signatures with the property that: if the Jaccard similarity between r and s is greater than or equal to θ, then they share a common signature. A signature based algorithm operates as follows. It generates signatures for each record in the reference table R which are indexed using an inverted index. Given a query string s, the lookup algorithm finds candidate strings r such that the signatures of r and s overlap. In a post-filtering step, the similarity predicate is checked and the matching records are returned.

Several signature schemes have been proposed in the literature for Jaccard similarity (without transformations) [16] including signature schemes developed by us [3, 9]. We now briefly discuss a signature scheme developed by us [9] called *prefix filtering*. Consider a total ordering over all tokens. Given a weighted set of tokens r and $0 \leq \beta \leq 1$, define $prefix_\beta(r)$ as follows: sort the elements in r by the global token ordering and find the shortest prefix such that the sum of the weights of the tokens in the prefix exceeds β times the weight of r. We have the following result [9]: if the jaccard similarity between strings r and s is greater than or equal to θ, then $prefix_{1-\theta}(r) \cap prefix_{1-\theta}(s) \neq \phi$. Fuzzy Lookup supports the signature schemes developed by us as well as the well-known *locality sensitive hashing (LSH)* which is a probabilistically approximate signature scheme. Multiple signature schemes enable Fuzzy Lookup to operate under different precision performance trade-offs.

In the presence of transformations, a naive adaptation of the above algorithms is as follows. Consider the expanded relation $ExpandR$ obtained by applying the transformations to each string in R. Given a query string s, we can similarly generate an expanded set of strings, look up each against $ExpandR$ and return the distinct matching rows. One problem with this approach is that the size of the expanded sets can be prohibitively high when we have a large number of transformations.

Id	*ProductName*
1	ThinkPad T43 Centrino PM 750 1.8GHz/512MB/40GB/14.1"TFT
2	ThinkPad T43 Centrino PM 740 1.7GHz/512MB/40GB/14.1"TFT
3	Lenovo ThinkPad T43 Notebook (1.7GHz Pentium M Centrino 740, 512MB, 40 GB, 14.1TFT)

(a)

Id	*Brand*	*Line*	*Model*	*Processor*	*Processor Model*	...
1		ThinkPad	T43	Centrino PM	750	...
2		ThinkPad	T43	Centrino PM	740	...
3	Lenovo	ThinkPad	T43	Pentium M Centrino	740	...

(b)

Figure 3: Sample Product Records

Example 3: Consider the citation: "N Koudas, S Sarawagi, D Srivastava. Record linkage: Similarity Measures and Algorithms. Proceedings of the 2006 ACM SIGMOD International Conference. Chicago, IL, USA." Suppose that we consider the set of rules {N→Nick, S→Sunita, D→Divesh, SIGMOD→"Special Interest Group on Management of Data", ACM→"Association for Computing Machinery", IL→Illinois}. The number of strings generated by this citation under these rules is $2^6 = 64$.

Since each lookup of a derived string in turn generates signatures that are then looked up, we can improve the performance by only looking up the *distinct* signatures over all the derived strings.

Example 4: Consider Example 3. Suppose the set (of last names) {Koudas, Sarawagi, Srivastava} is a signature generated by all the 64 generated records. This signature need not be replicated 64 times; one copy suffices.

Similarly, in the presence of transformations, even computing the similarity function becomes challenging. The straightforward method is to enumerate all generated strings which is worst-case exponential. For the large class of single-token transformations where both the `lhs` and `rhs` are single tokens (e.g., `St`→`Street`), we show that it is possible to compute the Jaccard similarity between two strings (under the transformations) efficiently (in polynomial time) by reducing this problem to bipartite matching [1]. Using the above optimizations and additional optimizations proposed in our previous work [1], we are able to support highly efficient record matching. For example, over a database of 10 million US landmarks used by Bing Maps, in the presence of a total of 24 million transformations, the average response time per lookup is less than 3 ms.

3 Segmentation

In this section, we discuss another primitive called *segmentation (parsing)*. The segmentation primitive takes as input unstructured text and produces as output a structured record obtained by assigning labels to various segments of the text. Figure 3(a) shows a set of sample laptop product names and Figure 3(b) shows the desired output of segmenting the product names. Segmentation is a useful data cleaning (transformation) primitive since structured records enable richer search compared to unstructured text. Further, segmentation primitive can be useful for record matching. Consider record matching directly over the records in Figure 3(a). Here, textual similarity is very misleading: The records with id 1 and 2 are textually very similar but are not matches since they have different processor models. Conversely, records 1 and 3 are textually dissimilar but correspond to the same laptop configuration. The segmented records of Figure 3(b) let us overcome the limitations of textual similarity. We can design sophisticated record matching logic that, for example, checks if two processor models are equal before classifying the corresponding records as matches. More generally, segmentation can be used as a preprocessing step before using primitives such as Fuzzy Lookup in the design of record matching packages. Our current implementation of the segmentation operator involves *parsing* an input string using a regular expression and using the parse to identify segments and assign labels. The regular expression can reference *dictionaries*

which encode domain knowledge. As an illustrative (and highly simplified) example, we can segment address strings (e.g., "`123 Main St Seattle WA 12345`") using the regular expression:

$$\underbrace{[\text{digit}]+}_{Street\ Num}\ \underbrace{[\text{letter}]+\ \text{“St”}}_{Street\ Name}\ \underbrace{[\text{City}]}_{City}\ \underbrace{[\text{State}]}_{State}\ \underbrace{[\text{digit}]+}_{PostalCode}$$

where [digit] and [letter] represent digits and letters, and [City] and [State] refer to dictionaries of city and state names. We note that dictionaries are a form of customization input similar to transformations for Fuzzy Lookup.

4 Design Tools

Recall from Section 1 that designing a data cleaning solution in our platform involves composing various primitives into a *package (or program)* and customizing each primitive in the package for the domain of interest. For complex data cleaning tasks, manually identifying the best way to compose and customize primitives can be difficult, motivating the need for design tools that reduce the manual effort involved.

Customizing Fuzzy Lookup involves specifying a tokenization scheme, token weights, and transformation rules. For most record matching scenarios, whitespace-based tokenization and IDF-based token weights are reasonable defaults. Identifying a useful set of transformations is more challenging. We have recently developed a tool [2] that automatically identifies candidate transformations from examples of matching records. The basic idea is to identify a concise set of transformations that explain textual "differences" between the example records. For example, the two matching addresses `123 Main St` and `123 Main Street` suggest the transformation `St`→`Street`. Similarly, customizing the segmentation primitive involves identifying appropriate dictionaries. We recently developed techniques to automatically identify dictionaries (e.g., list of cities) given a few examples (e.g., `San Francisco`, `New York`) using Web data [15]. The technique uses the observation that many lists on the web contain members of a concept, e.g., names of cities. In the example above, other strings that occur in the same lists as `San Francisco` and `New York` are likely to be cities, and can therefore be identified. Developing a general tool to automatically identify the best way to combine primitives for a given data cleaning task is challenging since the space of possible compositions is quite large. We have taken some initial steps in this direction, and have a tool that identifies a good record matching package that uses only Fuzzy Lookup primitive [4]. Ideally, it is desirable to have a single design tool for both customizing primitives and combining them; developing such a tool would be an interesting avenue for future work.

When integrating data from various sources into a warehouse, it is useful to be able to assess the quality of data being integrated, and we have developed a *Data Profiling* tool for this purpose. The Data Profiling tool efficiently identifies various properties of a dataset such as value distributions, (approximate) keys, functional dependencies, foreign keys, and regular expression patterns in data [10]. These properties can help identify data quality issues and also serve as input for subsequent data cleaning steps. For example, functional dependencies can be exploited for filling missing values and regular expressions can be used for segmentation (see Section 3).

5 Conclusions

In this paper, we provided a brief overview of the data cleaning project at Microsoft Research where we have developed primitives and design tools that allow a programmer to develop data cleaning solutions with minimal programming effort. In particular, we presented two primitives for data cleaning, namely, Fuzzy Lookup and segmentation. While our primitives are generic, they allow rich customizations that can be domain or application specific and can efficiently handle large datasets. Our primitives have been successfully used in several large real-world data cleaning applications.

Acknowledgments

Venkatesh Ganti was a (founding) member of our Data Cleaning project until 2009. He was involved in several research initiatives and the shipping of the initial version of Fuzzy Lookup in SSIS.

References

[1] A. Arasu, S. Chaudhuri, and R. Kaushik. Transformation-based framework for record matching. In *ICDE*, 2008.

[2] A. Arasu, S. Chaudhuri, and R. Kaushik. Learning string transformations from examples. *PVLDB*, 2(1), 2009.

[3] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set similarity joins. In *VLDB*, 2006.

[4] A. Arasu, M. Götz, and R. Kaushik. On active learning of record matching packages. In *SIGMOD*, 2010.

[5] Bing shopping. `http://www.bing.com/shopping`.

[6] S. Chaudhuri, U. Dayal, and V. Narasayya. An overview of business intelligence technology. *CACM*, 54(8), 2011.

[7] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the ACM SIGMOD*, June 2003.

[8] S. Chaudhuri, V. Ganti, and R. Kaushik. Data debugger: An operator-centric approach for data quality solutions. *IEEE Data Eng. Bull.*, 29(2), 2006.

[9] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, 2006.

[10] Z. Chen and V. R. Narasayya. Efficient computation of multiple group by queries. In *SIGMOD*, 2005.

[11] A. Elmagarmid, P. G. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Trans. on Knowledge and Data Engg.*, 19(1), 2007.

[12] Fuzzy lookup add-in for excel. `http://www.microsoft.com/bi/en-us/Community/BILabs/Pages/FuzzyLookupAddInforExcel.aspx`.

[13] Person finder: 2011 japan earthquake. `http://japan.person-finder.appspot.com/`.

[14] L. Gravano, P. G. Ipeirotis, et al. Approximate string joins in a database (almost) for free. In *VLDB*, 2001.

[15] Y. He and D. Xin. Seisa: set expansion by iterative similarity aggregation. In *WWW*, pages 427–436, 2011.

[16] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, 2006.

[17] Data cleaning project. http://research.microsoft.com/en-us/projects/datacleaning/default.aspx.

[18] Trillium Software. www.trilliumsoft.com/trilliumsoft.nsf.

Developments in Generic Entity Resolution

Steven Euijong Whang
Stanford University
swhang@cs.stanford.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

Abstract

Entity resolution (ER) is the problem of identifying which records in a database refer to the same entity. Although ER is a well-known problem, the rapid increase of data has made ER a challenging problem in many application areas ranging from resolving shopping items to counter-terrorism. The SERF project at Stanford focuses on providing scalable and accurate ER techniques that can be used across applications. We introduce generic ER and explain the recent advances made in our project.

1 Introduction

Information integration is one of the most important and challenging computer science problems: Information from diverse sources must be combined, so that users can access and manipulate the information in a unified way. One of the central problems in information integration is that of *Entity Resolution (ER)* (sometimes referred to as deduplication). ER is the process of identifying and merging records judged to represent the same real-world entity. ER is a well-known problem that arises in many applications. For example, mailing lists may contain multiple entries representing the same physical address, but each record may be slightly different, e.g., containing different spellings or missing some information. As a second example, a comparative shopping website may aggregate product catalogs from multiple merchants.

Identifying records that *match* (e.g., records that represent the same product) is challenging because there are no unique identifiers across sources (e.g., merchant catalogs that contain the information of products). A given record may appear in different ways in each source, and there is a fair amount of guesswork in determining which records match. Deciding if records match is often computationally expensive, e.g., may involve finding maximal common subsequences in two strings. How to *merge* records, i.e., combine records that match is often application dependent. For example, say different prices appear in two records to be merged. In some cases we may wish to keep both of them, while in others we may want to pick just one as the "consolidated" price. Such ER techniques find direct use in disciplines like computer science, biology, medicine, and even counter-terrorism.

The SERF project [7] at Stanford focuses on providing a general framework for ER that can span multiple applications. We do not study the internal details of the functions that determine if one or more records represent the same real-world entity. Rather, we view such functions as "black boxes", making our solutions useful across applications. In the initial phases of our project [1], we studied black-boxes that compared pairs of records

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

only (pair-wise matching functions). If two records matched, they were deemed to represent the same real-world entity and were merged into a composite record (via a merge function). As we will see (Section 2), more recently we have considered more general black-boxes that simply take as input a set of records (or an initial partition of records) and produce a partition of the records, where records in the same output partition are deemed to represent the same real-world entity. We call such a black-box the "core ER algorithm." The core algorithm could use pairwise matching as its strategy, but could use any other scheme.

After defining our model, we introduce recent developments for generic ER in the SERF project. We first consider scaling ER on a single data type where all the records have the same schema (e.g., all records refer to people). Many blocking techniques [3] have been proposed for scaling ER where the entire data is divided into (possibly overlapping) blocks, and the core ER algorithm is run on one block at a time. However, the downside is that one may miss matching records across blocks. One solution is to consider the interaction between blocks where resolving one block may help resolve others. In Section 3, we illustrate how this approach can maintain accuracy while providing scalability.

We also study scaling ER when resolving multiple types of data. For example, some records may refer to papers while others refer to authors. Since papers and authors are related to each other, resolving papers may help resolving authors and vice versa. While many ER solutions in the literature focus on resolving one type of records, few of them study the interactions between resolving different types of records. In Section 4 we illustrate joint ER and discuss scalable solutions.

In practice, a core ER function continuously "evolves" as the application developer has a better understanding on the application and data. For instance, when comparing people records, the application developer may at first think that comparing the names and zip codes of people is a good choice. After resolving the records, however, the developer may realize that comparing the names and phone numbers of people produces better ER results. (And this decision may further change as well.) When the ER algorithm logic changes, the naïve solution for ER is to simply re-run the ER algorithm from scratch, which may be prohibitively expensive when resolving large datasets. In Section 5, we provide incremental solutions for quickly updating ER results when the ER algorithm logic evolves.

Finally, we study a configurable and scalable quality measure for ER results. Many ER measures used in the literature "conflict" in a sense that, depending on the measure, different ER algorithms appear to perform better than others. However, many existing ER works tend to arbitrarily choose a measure "out of the hat" and evaluate their algorithms. In Section 6, we illustrate how conflicts may occur and introduce a scalable and configurable measure that can capture different qualities of ER results.

2 Core ER Model

A core ER algorithm takes as input a set of records R and groups together records that represent the same real world entity. We represent the output of the ER process as a partition of the input. We do not assume any particular form or data model for representing records.

To illustrate a possible core algorithm, consider the records of Table 14. Note that the structure of these records and the way we determine if records refer to the same real-world entity (via a pair-wise comparison) are part of the example. For this example, the core ER algorithm works as follows: A function compares the name, phone and email values of pairs of records. If the names are very similar (above some threshold), the records are said to match. The records also match if the phone and email are identical.

For our sample data, the ER algorithm determines that r_1 and r_2 match, but r_3 does not match either r_1 or r_2. For instance, the function finds that "John Doe" and "J. Doe" are similar, but finds "John D." not similar to anything (e.g., because John is a frequent first name). Thus, r_1 and r_2 are clustered together to form the partition $\{\{r_1, r_2\}, \{r_3\}\}$ where the curly brackets denote the clusters in the output partition.

Since records in an output cluster represent some real world entity, the cluster can be considered a "compos-

	Name	Phone	E-mail
r_1	John Doe	235-2635	jdoe@yahoo
r_2	J. Doe	234-4358	
r_3	John D.	234-4358	jdoe@yahoo

Table 14: A set of records representing persons

ite" new record. In some cases we may apply a merge function to actually generate the composite record. For example, suppose that the ER algorithm combines the names into a "normalized" representative, and performs a set-union on the emails and phone numbers. Then the records r_1 and r_2 will merge into the new records $\langle r_1, r_2 \rangle$ as shown below. (Here we denote a merged record with angle brackets.)

$\langle r_1, r_2 \rangle$	John Doe	234-4358, 235-2635	jdoe@yahoo

In this case, the ER result is the set of records $\{\langle r_1, r_2 \rangle, r_3\}$. Even if the records are merged, the lineage of the new record can be used to identify the original records in the cluster. Notice that in this case, we can also iteratively match $\langle r_1, r_2 \rangle$ with r_3 and merge them together because they now have the same phone and email. Throughout this paper, we will either cluster or merge records depending on the given setting.

3 Iterative Blocking

We now show how a core ER algorithm can be extended to become more scalable while maintaining accuracy. Blocking [3] is a common technique used in the ER literature where the data is divided into (possibly overlapping) blocks and a core ER algorithm is used to compare records within the same block, assuming that records in different blocks are unlikely to match. For example, we might partition a set of people records according to the zip codes in address fields. We then only need to compare the records with the same zip code. Since a single blocking criterion may miss matches (e.g., a person may have moved to places with different zip codes), several blocking criteria (i.e., dividing the data in several ways) are typically used to ensure that all the likely matching records are compared, improving the accuracy of the result.

Although many previous works focus on finding the best blocking criteria, most of them assume that all the blocks are processed separately one at a time. In many cases, however, it is useful to exploit an ER result of a previously processed block. First, when two records are grouped together (by the core algorithm) in one block, their composite may then be grouped with records in other blocks. Second, an ER result of a block can be used to reduce the processing time of another block. That is, the same record comparisons made by the core algorithm in one block may be repeated in another block, and hence can be avoided. To address these two points, we propose an iterative blocking framework [12] where the ER result of a block is immediately reflected to other blocks. Unlike previous blocking techniques, there is an additional stage where newly created partitions (i.e., groups of matching records) of a block are distributed to other blocks. Since the propagation of ER results can generate new record matches in other blocks, the entire operation becomes iterative in the sense that we are processing blocks (possibly the same block multiple times) until we arrive at a state where the resulting partition of records is stable. Our work is thus focused on effectively processing the blocks given a blocking criteria.

As an example, consider the five people records shown in Table 15, and a core algorithm that works by comparing pairs of records. We would like to merge (i.e., place in the same output partition) records that actually refer to the same person. Suppose that records r_1 and r_2 match with each other because their names are the same, but do not match with r_3 because the strings differ too much. However, once r_1 and r_2 are merged into a new record $\langle r_1, r_2 \rangle$, the combination of the address and email information of r_1 and r_2 may lead us to discover a new match with r_3, therefore yielding an initially unforeseen merge $\langle r_1, r_2, r_3 \rangle$. Notice that, in order to find this new merge, we need to compare the merged result of r_1 and r_2 with all the other records again.

Record	Name	Address(zip)	Email
r_1	John Doe	02139	jdoe@yahoo
r_2	John Doe	94305	
r_3	J. Foe	94305	jdoe@yahoo
r_4	Bobbie Brown	12345	bob@google
r_5	Bobbie Brown	12345	bob@google

Table 15: Customer records

In reality, our dataset can be very large, and it may not be feasible to compare all pairs of the dataset. Hence, we divide the customer records in Table 15 into blocks. We start by dividing the records by their zip codes. As a result, we only need to compare customers that are in the same geographical region. In Table 16, the first blocking criterion SC_1 uses zip codes to divide the records. Records r_2 and r_3 have the same zip code and are assigned to the block 2 (denoted as $b_{1,2}$) and records r_4 and r_5 are assigned to $b_{1,3}$ while r_1 is assigned to $b_{1,1}$. Since we may miss matches for people who have moved to several places with different zip codes, say we also divide the customer records according to the first characters of their last names. Hence, even if two records referring to the same person have different zip codes, we will have a better chance of comparing them because their last names might be similar. In Table 16, the matching records r_1 and r_2 can be compared because, although they have different zip codes, they have the same last name. After processing all the blocks, the final result of blocking is $\{\langle r_1, r_2\rangle, r_3, \langle r_4, r_5\rangle\}$.

Criterion	Partitions by	$b_{-,1}$	$b_{-,2}$	$b_{-,3}$
SC_1	zip code	r_1	r_2, r_3	r_4, r_5
SC_2	1^{st} char of last name	r_1, r_2	r_3	r_4, r_5

Table 16: Multiple blocking

Although the blocking of Table 16 reduces the number of records to compare, it misses the iterative match between $\langle r_1, r_2\rangle$ and r_3. Iterative blocking can find this match by distributing the newly created $\langle r_1, r_2\rangle$ (found in block $b_{2,1}$) to the other blocks. Assuming that $\langle r_1, r_2\rangle$ contains the zip codes of both r_1 and r_2 (i.e., 02139 and 94305), $\langle r_1, r_2\rangle$ is then assigned to both blocks of SC_1. In $b_{1,2}$, $\langle r_1, r_2\rangle$ can then be compared with r_3, generating the record $\langle r_1, r_2, r_3\rangle$. Eventually, the final iterative blocking solution becomes $\{\langle r_1, r_2, r_3\rangle, \langle r_4, r_5\rangle\}$. Thus, the iterative blocking framework helps find more record matches compared to simple blocking.

Iterative blocking also provides fast convergence. For example, once the records r_4 and r_5 are merged in $b_{1,3}$, they do not have to be compared in $b_{2,3}$. While the blocks in Table 16 are too small to show any improvements in runtime, iterative blocking can actually run faster than blocking for large datasets when the runtime savings exceed the overhead of iterative blocking. Intuitively, the more work we do for each block, the runtime savings for other blocks become significant.

In reference [12], we formalize the iterative blocking model and present two iterative blocking algorithms:

- *Lego:* An in-memory algorithm that efficiently processes blocks within memory.
- *Duplo:* A scalable disk-based algorithm that processes blocks from the disk.

We also experimentally evaluate Lego and Duplo and show that iterative blocking can improve over blocking both in accuracy and runtime.

4 Joint Resolution

The core ER algorithms can also be extended to jointly resolve multiple datasets of different entity types. Since relationships between the datasets exist, the result of resolving one dataset may benefit the resolution of another dataset. For example, the fact that two (apparently different) authors a_1 and a_2 have published the same papers could be strong evidence that the two authors are in fact the same. Also, by identifying the two authors to be the same, we can further deduce that two papers p_1 and p_2 written by a_1 and a_2, respectively, are more likely to be the same paper as well. This reasoning can easily be extended to more than two datasets. For example, resolving p_1 and p_2 can now help us resolve the two corresponding venues v_1 and v_2. Compared to resolving each dataset separately, joint ER can achieve better accuracy by exploiting the relationships between datasets.

To illustrate joint ER, consider two datasets P and V (see Table 17) that contain paper records and venue records, respectively. (Note that Table 17 is a simple example used for illustration. In practice, the datasets can be much larger and more complex.) The P dataset has the attributes Title and Venue while V has the attributes Name and Papers. For example, P contains record p_1 that has the title "The Theory of Joins in Relational Databases" presented at the venue v_1. The Papers field of a V record contains a set of paper records because one venue typically has more than one paper presented.

Paper	Title	Venue
p_1	The Theory of Joins in Relational ...	v_1
p_2	Efficient Optimization of a Class of ...	v_1
p_3	The Theory of Joins in Relational ...	v_2
p_4	Optimizing Joins in a Map-Reduce ...	v_3

Venue	Name	Papers
v_1	ACM TODS	$\{p_1, p_2\}$
v_2	ACM Trans. Database Syst.	$\{p_3\}$
v_3	EDBT	$\{p_4\}$

Table 17: Papers and Venues

Suppose that two papers are considered the same and are clustered if their titles and venues are similar while two venues are clustered if their names and papers are similar. Say that we resolve the paper records first. Since p_1 and p_3 have the exact same title, p_1 and p_3 are considered the same paper. We then resolve the venue records. Since the names of v_1 and v_2 are significantly different, they cannot match based on name similarity alone. Luckily, using the information that p_1 and p_3 are the same paper, we can infer that v_1 and v_2 are most likely the same venue. (In fact "ACM TODS" and "ACM Trans. Database Syst." stand for the same journal.) We can then re-resolve the papers in case there are newly matching records. This time, however, none of the papers match because of their different titles. Hence, we have arrived at a joint ER result where P was resolved into the partition $\{\{p_1, p_3\}, \{p_2\}, \{p_4\}\}$ while V was resolved into $\{\{v_1, v_2\}, \{v_3\}\}$. Notice that we have followed the sequence of resolving papers, venues, then papers again.

Given enough resources, we can improve the runtime performance by exploiting parallelism and minimizing unnecessary record comparisons. For example, if we have two processors, then we can resolve the papers and venues concurrently. As a result, p_1 and p_2 match with each other. After the papers and venues are resolved, we resolve the venues again, but only perform the incremental work. In our example, since p_1 and p_2 matched in the previous step, and p_1 was published in the venue v_1 while p_2 was published in the venue v_2, we only need to check if v_1 and v_2 are the same venue and can skip any comparison involving v_3. Notice that the papers do not have to be resolved at the same time because none of the venues merged in the previous step. However, after v_1 and v_2 are identified as the same venue, we resolve the papers once more. Again, we only perform the incremental work necessary where we resolve the three records p_1, p_2, and p_3 (because v_1 and v_2 matched in the previous step), but not p_4. In total, we have concurrently resolved the papers and venues, then incrementally resolved the venues, and then incrementally resolved the papers. If the incremental work is much smaller than resolving a dataset from the beginning, the total runtime may improve.

Among the existing works on joint ER [2], few have focused on scalability, which is crucial in resolving large data (e.g., hundreds of millions of people records crawled from the Web). The solutions that do address

scalability propose custom joint ER algorithms for efficiently resolving records. However, given that there exists ER algorithms that are optimized for specific types of records (e.g., there could be an ER algorithm that specializes in resolving authors only and another ER algorithm that is good at resolving venues), replacing all the ER algorithms with a single joint ER algorithm that customizes to all types of records may be challenging for the application developer.

Instead, we propose a flexible, modular resolution framework where existing ER algorithms developed for a given record type can be plugged in and used in concert with other ER algorithms. While many previous joint ER works assume that all the datasets are resolved at the same time in memory using one processor, our framework allows efficient resource management by resolving a few datasets at a time in memory using multiple processors as illustrated above. A conceptually similar ER technique is blocking where the entire data of one type is resolved in small subsets or blocks. Similarly, our framework resolves multiples types of data and divides the resolution based on the data type. Our approach may especially be useful when there are many large datasets that cannot be resolved altogether in memory. Thus one of the key challenges is determining a good sequence for resolution. For instance, should we resolve all venues first, and then all papers and then all authors? Or should we consider a different order? Or should we resolve some venues first, then some related papers and authors, and then return to resolve more venues? And we may also have to resolve a type of record multiple times, since subsequent resolutions may impact the work we did initially. These issues (and others) are explored in a technical report [10] that describes our ongoing work in this area.

5 Incremental Resolution

We now show how certain properties of the core ER algorithm enable incremental solutions when the core ER algorithm itself changes frequently. In practice, an entity resolution (ER) result is not produced once, but is constantly improved based on better understandings of the data, the schema, and the logic that examines and compares records. In particular, here we focus on changes to the logic that compares two records in the core ER algorithm. We call this logic the *rule*, and it can be a Boolean function that determines if two records represent the same entity, or a distance function that quantifies how different (or similar) the records are. Initially the core ER algorithm starts with a set of records S, then produce a first ER result E_1 based on S and a rule B_1. Some time later rule B_1 is improved yielding rule B_2, so we need to compute a new ER result E_2 based on S and B_2. The process continues with new rules B_3, B_4 and so on.

A naïve approach would compute each new ER result from scratch, starting from S, a potentially very expensive proposition. Instead, we explore an incremental approach [9], where for example we compute E_2 based on E_1. Of course for this approach to work, we need to understand how the new rule B_2 relates to the old one B_1, so we can understand what changes incrementally in E_1 to obtain E_2.

For example, consider the set of people records S shown in Table 18. The first rule B_1 (see Table 18) says that two records match (represent the same real world entity) if predicate p_{name} evaluates to true. Predicates can in general be quite complex, but for this example assume that predicates simply perform an equality check using its attribute. The ER algorithm calls on B_1 to compare records and groups together records with name "John," producing the result $\{\{r_1, r_2, r_3\}, \{r_4\}\}$.

Record	Name	Zip	Phone
r_1	$John$	54321	123-4567
r_2	$John$	54321	987-6543
r_3	$John$	11111	987-6543
r_4	Bob	$null$	121-1212

Comparison Rule	Definition
B_1	p_{name}
B_2	$p_{name} \wedge p_{zip}$
B_3	$p_{name} \wedge p_{phone}$

Table 18: Records and Comparison Rules

Next, say users are not satisfied with this result, so a data administrator decides to refine B_1 by adding a predicate that checks zip codes. Thus, the new rule is B_2 shown in Table 18. The naïve option is to run the same ER algorithm with rule B_2 on set S to obtain the partition $\{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}$. (Only records r_1 and r_2 have the same name and same zip code.) This process repeats much unnecessary work: For instance, we would need to compare r_1 with r_4 to see if they match on name and zip code, but we already know from the first run that they do not match on name (B_1), so they cannot match under B_2.

Because the new rule B_2 is "stricter" than B_1 (i.e. all records that match according to B_2 also match according to B_1), we can actually start the second ER from the first result $\{\{r_1, r_2, r_3\}, \{r_4\}\}$. That is, we only need to check each cluster separately and see if it needs to split. In our example, we find that r_3 does not match the other records in its cluster, so we arrive at $\{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}$. This approach only works if the ER algorithm satisfies certain properties and B_2 is stricter than B_1. One of the properties that the ER algorithm must satisfy is being able to resolve clusters "independent" of each other. In our example above, splitting the cluster $\{r_1, r_2, r_3\}$ must have no affect on the cluster $\{r_4\}$. If B_2 is not stricter and the ER algorithm satisfies different properties, there are other incremental techniques we can apply.

A complementary technique is to "materialize" auxiliary results during one ER run, in order to improve the performance of future ER runs. To illustrate, say that when we process $B_2 = p_{name} \wedge p_{zip}$, we concurrently produce the results for each predicate individually. That is, we compute three separate partitions, one for the full B_2, one for rule p_{name} and one for rule p_{zip}. The result for p_{name} is the same $\{\{r_1, r_2, r_3\}, \{r_4\}\}$ seen earlier. For p_{zip} it is $\{\{r_1, r_2\}, \{r_3\}, \{r_4\}\}$. The cost of computing the two extra materializations can be significantly lower than running the ER algorithm three times, as a lot of the work can be shared among the runs.

The materializations pay off when rule B_2 evolves into a related rule that is not quite stricter. For example, say that B_2 evolves into $B_3 = p_{name} \wedge p_{phone}$, where p_{phone} checks for matching phone numbers. In this case, B_3 is not stricter than B_2 so we cannot start from the B_2 result. However, we can start from the p_{name} result, since B_3 is stricter than p_{name}. Thus, we examine each cluster in $\{\{r_1, r_2, r_3\}, \{r_4\}\}$, splitting the first cluster because r_2 has a different phone number. The final result is $\{\{r_1, r_3\}, \{r_2\}, \{r_4\}\}$. Clearly, materialization of partial results may or may not pay off, just like materialized views and indexes may or may not help.

In reference [9], we formalize rule evolution and identify desirable properties of ER algorithms that enable fast rule evolution and categorize existing ER algorithms based on the properties they satisfy. We then propose efficient rule evolution techniques that use one or more of the properties. We then experimentally evaluate the rule evolution algorithms for various ER algorithms and show that rule evolution can be significantly faster than the naïve approach with reasonable time and space overheads.

6 Quality Measure

Properly evaluating ER results is important for comparing the performances of various ER algorithms. Usually when we compare entity resolution algorithms, we run them on a data set and compare the results to a "gold standard." The gold standard is an ER result that we assume to be correct. In many cases, the gold standard is generated by a group of human experts. On large data sets where the task is too large to be handled by a human, it is not uncommon to run an exhaustive algorithm to generate a result, and treat that result as the gold standard. Then we can compare the results of other approximate or heuristic-based algorithms to this standard in the same manner we would compare them to a human-generated gold standard.

A key component of this type of evaluation is a method of assigning a number to express how close a given ER result is to the gold standard. Many ER measures have been proposed for comparing the results of ER algorithms, but there is currently no agreed standard measure for evaluating ER results. Most works tend to use one ER measure over another without a clear explanation of why that ER measure is most appropriate. The pitfall of using an arbitrary measure is that different measures may disagree on which ER results are the best.

For example, consider an entity resolution problem with an input set of records $I = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7,$

r_8, r_9, $r_{10}\}$. Three possible ER results are shown in Table 19, along with the gold standard.

Set	ER Result
Gold Standard	$\{\{r_1, r_2\}, \{r_3, r_4\}, \{r_5, r_6, r_7, r_8, r_9, r_{10}\}\}$
R_1	$\{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}, \{r_5, r_6, r_7, r_8, r_9, r_{10}\}\}$
R_2	$\{\{r_1, r_2\}, \{r_3, r_4\}, \{r_5, r_6, r_7\}, \{r_8, r_9, r_{10}\}\}$
R_3	$\{\{r_1, r_2, r_3, r_4\}, \{r_5, r_6, r_7, r_8, r_9, r_{10}\}\}$

Table 19: ER results and Gold Standard

Suppose we are evaluating two ER results R_1 and R_2, against the gold standard G. Using an ER measure that evaluates a result based on the number of record pairs that match, R_1 could be a better solution because it found 15 correct pairs (i.e., all record pairs in $\{r_5, r_6, r_7, r_8, r_9, r_{10}\}$) while R_2 only found 8 correct pairs. On the other hand, if we use a measure that evaluates results based on correctly resolved entities in the gold standard, R_2 could be considered better than R_1 because R_2 contains two correctly resolved entities $\{r_1, r_2\}$ and $\{r_3, r_4\}$ while R_1 only has one correct entity $\{r_5, r_6, r_7, r_8, r_9, r_{10}\}$. As another example, suppose that we compare R_2 and R_3. One measure could be more focused on high precision and prefer R_2 over R_3 because R_2 has only found correctly matching records while R_3 has found some non-matching records (e.g., r_1 and r_3 do not match). On the other hand, another measure might consider recall to be more important and prefer R_3 over R_2 because R_2 has not found all the matching record pairs (unlike R_3).

Surprisingly, such conflicts between ER measures can occur frequently. It is tempting to suggest that when conflicts arise, one of the measures involved must be faulty in some way. However, since different applications may have different criteria that define the "goodness" of a result, we cannot simply claim one measure to be better than another.

In reference [6], we provide a survey of ER measures that have been used to date, experimentally demonstrates the frequency of conflicts between these measures, and provides an analysis of how the measures differ.

We also propose a configurable measure inspired by the edit distance of strings. Rather than the insertions, deletions and swaps of characters used in edit distance, our measure is based upon the elementary operations of merging and splitting clusters. We therefore call this measure "merge distance." A basic merge distance simply counts the number of splits and merges. For example, R_1 has a distance of 2 from G because it takes two cluster merges to convert R_1 to G. On the other hand, R_3 only has a distance of 1 because one cluster split is required to convert R_3 to G. But as we have mentioned, no single ER measure is better than all the others. We thus generalize merge distance by letting the costs of merge and split operations be determined by "cost functions," arriving at an intuitive, configurable measure that can support the needs of a wide variety of applications. For example, the cost of splitting a cluster could be defined to be twice as large as the cost of merging two clusters. Or the costs of merging and splitting clusters could be proportional to the sizes of the input clusters. While differently configured merge distance measures may still conflict, we now have a better understanding of what quality each configured measure is evaluating. Surprisingly, at least two state-of-the-art measures are closely related to generalized merge distance: the Variation of Information (VI) [5] clustering measure is a special case of generalized merge distance while the pairwise F_1 [4] measure can be directly computed using generalized merge distance. We also propose an efficient linear-time algorithm that computes generalized merge distance for a large class of cost functions that satisfy reasonable properties.

7 Conclusion

In this paper, we have introduced the recent advances on generic ER in the SERF project. We have first explained our basic ER model where a core ER algorithm resolves a set of records. We have then shown two methods – iterative blocking and joint ER – that can scale ER while maintaining accuracy using the core ER algorithms. We

have also shown incremental ER solutions for evolving ER rules. Finally, we have proposed a configurable and scalable quality measure for evaluating ER results. Although not covered in this paper, we have also studied the problems of correcting inconsistencies [8] and producing good partial ER results within a limited runtime [11].

References

[1] O. Benjelloun, H. Garcia-Molina, H. Kawai, T. E. Larson, D. Menestrina, Q. Su, S. Thavisomboon, and J. Widom. Generic entity resolution in the serf project. *IEEE Data Eng. Bull.*, 29(2):13–20, 2006.

[2] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD Conference*, pages 85–96, 2005.

[3] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD Conference*, pages 127–138, 1995.

[4] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[5] M. Meila. Comparing clusterings by the variation of information. In *COLT*, pages 173–187, 2003.

[6] D. Menestrina, S. E. Whang, and H. Garcia-Molina. Evaluating entity resolution results. *PVLDB*, 3(1):208–219, 2010.

[7] Stanford Entity Resolution Framework. http://infolab.stanford.edu/serf/.

[8] S. E. Whang, O. Benjelloun, and H. Garcia-Molina. Generic entity resolution with negative rules. *VLDB J.*, 18(6):1261–1277, 2009.

[9] S. E. Whang and H. Garcia-Molina. Entity resolution with evolving rules. *PVLDB*, 3(1):1326–1337, 2010.

[10] S. E. Whang and H. Garcia-Molina. Joint entity resolution. Technical report, Stanford University, available at http://ilpubs.stanford.edu:8090/1002/.

[11] S. E. Whang, D. Marmaros, and H. Garcia-Molina. Pay-as-you-go entity resolution. Technical report, Stanford University, available at http://ilpubs.stanford.edu:8090/979/.

[12] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD Conference*, pages 219–232, 2009.

Extracting, Linking and Integrating Data from Public Sources: A Financial Case Study

Doug Burdick[1], Mauricio Hernández[1], Howard Ho[1], Georgia Koutrika[1], Rajasekar Krishnamurthy[1]
Lucian Popa[1], Ioana R. Stanoi[1], Shivakumar Vaithyanathan[1], Sanjiv Das[2]

[1] IBM Research – Almaden [2] Finance Department, Santa Clara University

Abstract

We present Midas, a system that uses complex data processing to extract and aggregate facts from a large collection of structured and unstructured documents into a set of unified, clean entities and relationships. Midas focuses on data for financial companies and is based on periodic filings with the U.S. Securities and Exchange Commission (SEC) and Federal Deposit Insurance Corporation (FDIC). We show that, by using data aggregated by Midas, we can provide valuable insights about financial institutions either at the whole system level or at the individual company level.

The key technology components that we implemented in Midas and that enable the various financial applications are: information extraction, entity resolution, mapping and fusion, all on top of a scalable infrastructure based on Hadoop. We describe our experience in building the Midas system and also outline the key research questions that remain to be addressed towards building a generic, high-level infrastructure for large-scale data integration from public sources.

1 Introduction

During the last few years, we have observed an explosion in the number and variety of public data sources that are available on the web: research papers and citations data (e.g., Cora, Citeseer, DBLP), online movie databases (e.g., IMDB), etc. Accurate extraction and integration of key concepts from these sources is challenging since their contents can be distributed over multiple web sites and vary from unstructured (or text) to semi-structured (html, XML, csv) and structured (e.g., tables). Even highly regulated sources, such as government regulatory data from the SEC and FDIC, still pose challenges since a large number of filings are in text.

In this paper, we present our experience with building and applying Midas, a system that unleashes the value of information archived by SEC and FDIC, by extracting, integrating, and aggregating data from semi-structured or text filings. We show that, by focusing on high-quality financial data sources and by combining three complementary technology components – information extraction, information integration, and scalable infrastructure – we can provide valuable insights about financial institutions either at the system level (i.e., systemic analysis) or at the individual company level. Midas is a system that starts from a document-centric archive, as provided by SEC and FDIC, and builds an entity-centric repository (similar to the "web of concepts" [12]) where the main entities are companies, their key people, loans, securities, together with their relationships.

There is a plethora of research in information extraction [11, 14], entity resolution [15], schema mapping [16, 19] and, in general, information integration [17]. While Midas relies on technologies and ideas from these

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

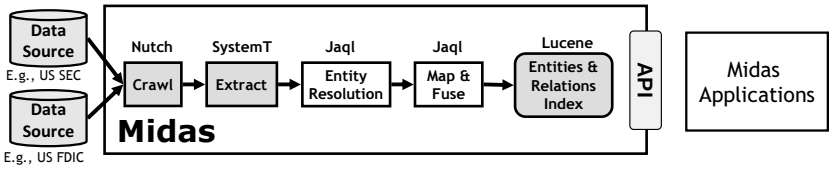


Figure 1: The Midas Data Flow

areas, our main contribution can be seen in the synergistic use of both unstructured and structured information integration to build a comprehensive solution for the financial domain that brings out the value of the data in public sources. This paper is organized as follows. Section 2 gives a high-level view of the Midas system. We describe in Section 3 two important types of applications that Midas enables. Section 4 gives further details on the integration stages implemented in Midas. We conclude in Section 5 with a discussion of remaining challenges.

2 Midas Overview

Figure 1 gives a high-level view of Midas, our system for extracting and integrating information from public data. Midas takes its input from multiple types of documents (ranging from text to HTML and XML) that are archived by SEC and FDIC. As output, Midas produces a set of integrated and cleansed entities and relationships, which are used by applications like the ones described in Section 3. The input data can be large (Peta-bytes of information, overall) with new incremental updates arriving daily. Hence, all components in the Midas data flow must be able to process large amounts of data efficiently and should scale well with increasing data sizes. To address these challenges, Midas is designed to run on Hadoop, and many of its operations are expressed in Jaql [5], a high-level language that compiles data transformations as map/reduce jobs.

Crawl is in charge of retrieving documents from the archives and storing them in our local file system. Instances of **Crawl** are implemented using Nutch, a widely used open-source crawler (http://nutch.apache.org/). We run Nutch as Hadoop jobs to parallelize the fetching of documents. We have currently crawled close to 1,000,000 SEC documents related to financial companies and 77,000 FDIC reports for active banks.

Extract is in charge of extracting facts from the unstructured documents. Here, we leverage a large library of previously existing information extraction modules (annotators) implemented on top of SystemT, a rule-based information extraction system developed at IBM Research and built around a declarative rule language (AQL) [9]. SystemT can deliver an order of magnitude higher annotation throughput compared to a state-of-the-art grammar-based IE system [9]. Furthermore, high-quality annotators can be built in SystemT with accuracy that matches or outperforms the best published results [10]. AQL rules are applied to each input document and produce a stream of annotated objects, making this operation trivially parallelizable with Hadoop. After applying extraction rules to each input document, we obtain structured records containing the extracted attributes with their values (e.g., a person name, a job title, a company name), as well as associated meta-data (e.g., the file id, date, text location of each extracted attribute, etc).

Entity Resolution identifies and links extracted records that correspond to the same real-world entity. The data required to build an entity (e.g., a person) is spread across many documents that mention various aspects of that entity, at various times. Recognizing that these separate mentions refer to the same entity requires a complex and domain-dependent analysis in which the exact match of values may not work. For instance, person names are not always spelled the same; moreover, the documents might not explicitly contain a key to identify the person. Entity Resolution, which appears in the literature under other names (Record Linkage, Record Matching, Merge/Purge, De-duplication) [15], is often solved with methods that score fuzzy matches between two or more candidate records and use statistical weights to determine when these records indeed represent the same entity. Other methods explicitly express when two or more candidate records match using *rules*. Our implementation of Midas uses the latter approach, where we implemented the matching rules as Jaql scripts.

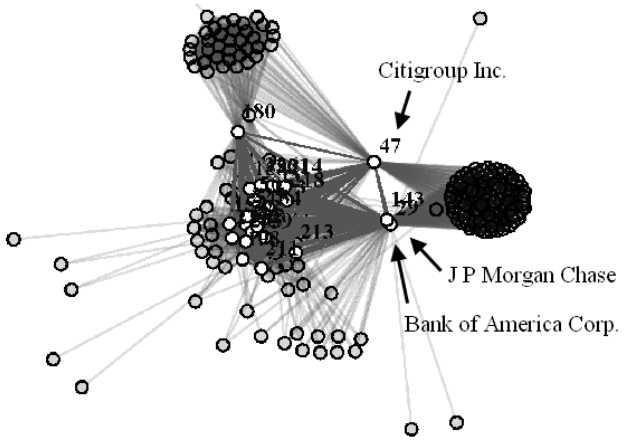


Figure 2: Co-lending network for 2005.

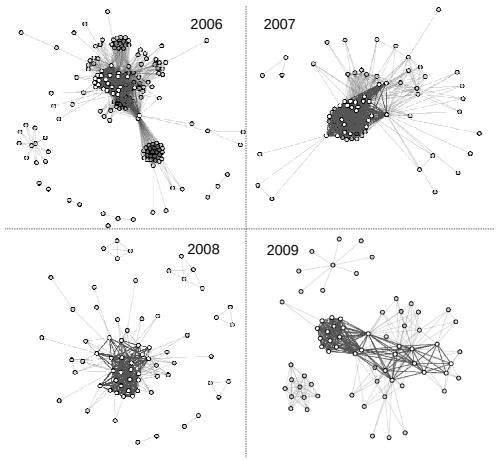


Figure 3: Co-lending networks for 2006–2009.

Map & Fuse transform the extracted (and possibly linked) records into aggregated entities. All necessary queries to join and map the source data into the expected target schema(s) are implemented as part of this operator. Since data is collected from multiple sources, duplicate values for certain fields are inevitable and must be solved to determine which value survives. A more complex type of fusion that takes place as part of Map & Fuse is the temporal aggregation for certain complex attributes of an entity that have a temporal aspect (e.g., the employment history of key executives over the years).

We note that in the Midas implementation, there is no fixed ordering between Entity Resolution and Map & Fuse. Instead, there are multiple instantiations of these components that apply in a flow. As we see in Section 4, Midas first creates an initial set of integrated entities via an application of Map & Fuse on a set of input records, which happen to have global keys for the main entities. Thus, there is no need for entity resolution in this first phase. However, subsequently, Entity Resolution needs to be applied so that other types of extracted records (from text, with no keys) are linked to the initial set of integrated entities. Based on the generated links, the extracted records are then integrated into the appropriate target entities by applications of Map & Fuse.

3 Midas: The Applications

In this section, we discuss the types of financial applications that the data aggregated by Midas enables. We group these applications into two types (one systemic, and one at the individual company level).

3.1 Systemic Risk Analysis

Systemic analysis is defined as the measurement and analysis of relationships across entities towards understanding their impact on the system as a whole. The failure of a major player in a market that causes the failure/weakness of other players is an example of a systemic effect, such as the one experienced with the bankruptcy of Lehman Brothers in 2008. A major challenge to performing quality systemic analysis is the paucity of data for the *entire* system. Current approaches rely on a few proprietary data sets of limited scope [1, 2, 6, 20]. Midas addresses this challenge in a major way by leveraging unstructured or semi-structured public data archived by SEC and FDIC to provide much richer information across the entire system of financial institutions. In turn, this enables finance researchers to develop new and powerful risk analysis techniques. As an example, we will use co-lending relationships to construct networks of relationships between banks, and then use network analysis to determine which banks pose the greatest risk to the financial system.

Co-lending Systemic Risk. Using loan-related data extracted from SEC/FDIC filings from 2005 to 2009, we construct a network of connections between financial firms based on their co-investment in loans made to other corporations or financial institutions. Concretely, if five banks made a joint loan, we add undirected edges (each

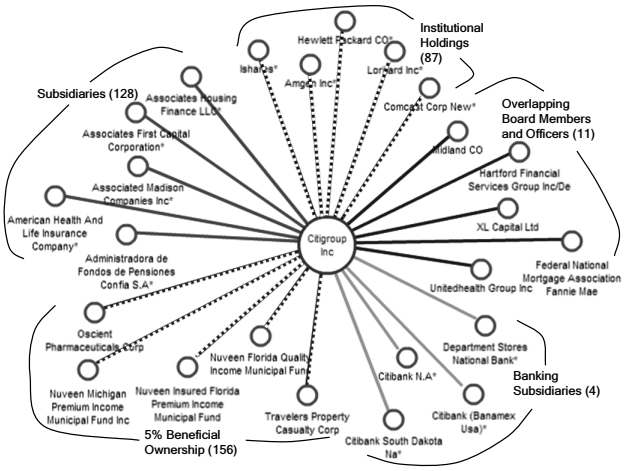


Figure 4: Companies related to Citigroup.

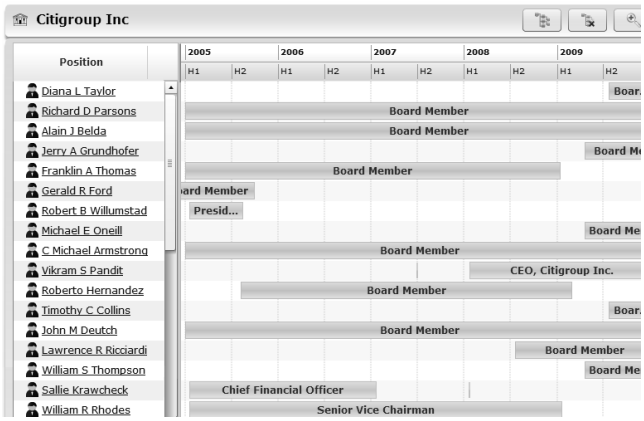


Figure 5: Key people for Citigroup.

with a score of 1, initially) for all pairs of the five banks. Overall, we create an undirected network with the banks as nodes, and where the edges have the total count of pairwise co-lending, aggregated across all loans. A bank failure will directly impact the co-lending activity of all banks it is connected with, and will indirectly impact the banks that are connected to the ones it is directly connected with. Therefore, even if a bank has very few co-lending relationships itself, it may impact the entire system if it is connected to a few major lenders.

Figure 2 shows the resulting co-lending network for 2005. We see that there are three large components of co-lenders that are clustered together, and three hub banks, with connections to the large components. To determine which banks in the network are most likely to contribute to systemic failure, we compute for each bank the normalized eigenvalue centrality score that is described in [7]. The three nodes with the highest centrality are seen to be critical hubs in the network—these are J.P. Morgan (node 143), Bank of America (node 29), and Citigroup (node 47). They are bridges between all banks, and contribute highly to systemic risk. Figure 3 shows how the network evolves in the four years after 2005. Comparing 2006 with 2005, we see that there are still large components connected by a few central nodes. From 2007 onwards, as the financial crisis begins to take hold, co-lending activity diminished markedly. The diameter of the co-lending graph becomes marginally smaller as the network shrinks. Also, all high centrality banks tend to cluster into a single large component in the latter years (see Figure 3, especially years 2007 and 2008).

This type of analysis is just one illustration of many possible techniques that can be developed on top of extracted and integrated data from SEC and FDIC. The particular framework we used, based on co-lending and centrality, can also be extended into a full risk management system for regulators.

3.2 Drill-Down into Individual Entities

While the previous section has given a view of the financial companies at the whole system level, in this section we describe additional views that are centered around the individual entities. For example, once a company such as Citigroup Inc. has been identified as a critical hub for the financial system, a regulator may want to dive deeper into various aspects of Citigroup: its relationships with other companies (subsidiaries, competitors, borrowers, etc.), its key executives (officers and directors), its aggregated financial data (loans, investments, etc.).

Company Relationships. Figure 4 shows how Citigroup is related to other companies through investment, lending, ownership, as well as shared insiders. For each relationship type, along with a count of the number of related companies in that category, we show up to five representative companies. **Banking subsidiaries** lists the four banks that Citigroup has registered with the FDIC. This information was obtained by integrating data from SEC and FDIC. **Subsidiaries** is an exhaustive list of Citigroup's global subsidiaries, as reported in the latest annual report (typically in text or html format). **5% Beneficial Ownership** enumerates the securities in which Citigroup has more than 5% ownership based on analysis of SC-13D and SC-13G text filings made

by Citigroup and its subsidiaries. **Overlapping board members/officers** represents key insiders (directors or officers) that are shared between Citigroup and other companies. **Institutional Holdings** represents securities in which Citigroup has invested more than $10 million based on analysis of 13F text filings. While the relationship graph provides a birds-eye view on Citigroup, one can further drill down into any of the individual relationships. We explore next the key insider aspect of a company.

Insider Analysis and Employment Histories. Understanding management structure of companies and relationships across companies through common officers and board of directors is relevant in firm dynamics and corporate governance. Connected firms appear to end up merging more [8], and understanding post-merger management structures based on earlier connections between the managers of the merged firms is also being studied [18]. To enable such analysis, Midas exposes detailed employment history and trading information for insiders (i.e., key officers and directors) of individual companies. Figure 5 shows some of the key officers and directors associated with Citigroup over the last several years. For each key person, the various positions held in Citigroup along with the time periods are also displayed. This profile is built by aggregating data from individual employment records present in annual reports, proxy statements, current reports and insider reports.

Additional views for insider holdings, insider transactions, and lending exposure are described in [4].

4 Midas Integration Flow: Further Details

We now give concrete details of the Midas flow that integrates information related to financial companies. We discuss in Section 4.1 the initial construction of a reference or core set of company and people entities from insider reports (Forms 3/4/5). Since these forms are in XML and contain structured and relatively clean data, the resulting core set of entities forms the backbone of the rest of the integration flow. In Section 4.2, we detail how further information from unstructured forms is extracted, linked and fused to the core set of entities. The final result is a set of entities with rich relationships, including detailed employment histories of key people, lending/co-lending relationships among companies, and all the other relationships we discussed in Section 3.2.

4.1 Constructing Core Entities

We now discuss the initial construction and aggregation of company and key people entities from the XML files that correspond to insider reports (Forms 3/4/5).

Extraction of records from XML forms. We use Jaql to extract (and convert to JSON) the relevant facts from XML Forms 3/4/5. Each fact states the relationship, as of a given reporting date, between a company and a key officer or director. The relevant attributes for the company are: the SEC-assigned key (or cik) of the company, the company name and address, the company stock symbol. The relevant attributes for the person are: the SEC key or cik, name, an attribute identifying whether the person is an officer or a director, and the title of the person (i.e., "CEO", "Executive VP", "CFO", etc) if an officer. Other important attributes include the reporting date, a document id, a list of transactions (e.g., stock buys or sells, exercise of options) that the person has executed in the reporting period, and a list of current holdings that the person has with the company.

Aggregation of company and people entities. In this step, we process all the facts that were extracted from XML forms and group them by company cik. Each group forms the skeleton for a company entity. The important attributes and relationships for a company are aggregated from the group of records with the given company cik. As an example of important attribute of a company, we aggregate the set of all officers of a company such as Citigroup Inc. This aggregation is with respect to all the forms 3/4/5 that Citigroup Inc. has filed over the five years. Additional fusion must be done so that each officer appears only once in the list. Furthermore, for each officer, we aggregate all the positions that the respective person has held with the company. As an example, a person such as Sallie Krawcheck will result in one occurrence within the list of officers of Citigroup, where this occurrence contains the list of all the positions held by Sallie Krawcheck with Citigroup (e.g., CFO, CEO of Global Wealth Management). Since positions are strings that vary across forms, normalization code is used to identify and fuse the "same" position. Finally, each position is associated with a set of dates, corresponding to

all the filings that report that position. The earliest and the latest date in this set of dates is used to define the time span of the position (assuming continuous employment). The result of this analysis is illustrated in Figure 5.

To give a quantitative feel about the above processing, there are about $400,000$ facts extracted from the 3/4/5 forms. These $400,000$ facts result in about $2,500$ company entities, each with a rich structure containing officers with their position timelines (within the company), directors (with similar timelines), and also containing an aggregation of transactions and holdings (to be discussed shortly). A separate but similar processing generates, from the same $400,000$ facts, an inverted view where people are the top-level entities. We generate about $32,000$ people entities, corresponding to the officers or directors that have worked for the $2,500$ financial companies. Each person entity is also a complex object with nested attributes such as employment history, which spans, in general, multiple companies. For example, the person entity for Sallie Krawcheck has an employment history spanning both Citigroup Inc. (where she served as CFO and then CEO of Global Wealth Management) and Bank of America (which she joined later as President of Global Wealth and Investment Banking).

Fusion of insider transactions and holdings. The aggregation of transaction and holding data over the collection of forms 3/4/5 requires a detailed temporal and numerical analysis. First, we need to ensure that we group together securities of the same type. In general, there are multiple types of securities (derivatives or non derivatives), types of ownership (direct or indirect), and types of transactions (acquired, disposed, granted, open market purchase, etc.). The various values for such types are reported in text and have variations (e.g., "Common Stock" vs. "Class A stock" vs. "Common shares"). In order to avoid double counting of transactions and to report only the most recent holding amount for each type, we developed normalization code for types of securities and for types of ownership. Subsequent processing summarizes, for each company and for each year, the total amount of transactions of certain type (e.g., open market purchase) that company insiders executed in that year. Examples of views that can be constructed from results of such aggregation can be found in [4].

4.2 Incorporating Data from Unstructured Forms

We now discuss the processing involved in the extraction and fusion of new facts from unstructured data into the core entities. The new facts, which are extracted from either text or HTML tables, describe new attributes or relationships, and typically mention a company or a person by name without, necessarily, a key. Thus, before the new information can be fused into the existing data, entity resolution is needed to perform the linkage from the entity mentions to the actual entities in the core set. Consider the example of enriching a person's employment history. In addition to the insider reports, information about a person's association with a company occurs in a wide variety of less structured filings. This information ranges from point-in-time facts (when an officer/director signs a document) to complete biographies of a person. To extract and correctly fuse all the needed pieces of information, we must address several challenges.

Extract. Employment history records need to be extracted from various contexts such as biographies, signatures, job change announcements, and committee membership and compensation data. These records are typically of the form (person name, position, company name, start date, end date) for each position mentioned in the text. However, not all of the attribute values may be present or extracted successfully. For instance, extraction may result in records such as (James Dimon, Chairman, JP Morgan Chase, –, –), (James Dimon, Chief Executive Officer, JP Morgan Chase, –, –), (James Dimon, Director, JP Morgan Chase, 2000, –) and (Mr. Dimon, Chairman, unknown, "December 31, 2006", –). All of these records have to be linked and fused by the next stages.

Using biographies as an example, we give a flavor of the challenges one typically encounters in extracting employment records from unstructured documents in SEC. First, biographies typically appear as short paragraphs within very large HTML documents (100s KBs to 10s MBs) and within HTML tables, where individual employment facts may be formatted in different ways. For instance, a position with a long title may span multiple rows while the corresponding person's name may align with only one of these rows, depending on the visual layout. Moreover, a person may have multiple positions linked with a single organization (e.g., Chairman and CEO of JP Morgan Chase); these positions sometimes are associated with the same start date (e.g., CEO and

President since 12/31/2005) or may have different timelines (some of which may have to be inferred later in subsequent stages). Furthermore, individual sentences may refer to an individual via a partial name (e.g., "Mr. Dimon") or by using pronouns (e.g., "he"). Sometime the name of a related individual may be mentioned in the biography. Hence, extracting the person name itself may be a challenge. All of these challenges are addressed in Midas by carefully designed and customized AQL rules that are often complemented by Java functions.

Entity Resolution. As mentioned, the attributes extracted for biographies include the name of the person, the name of the filer company (also the cik, since this is associated with the filing entity) and the biography text itself. However, information in biographies does not contain the key (cik) for the person and we need entity resolution to link each extracted biography record to a person cik. Entity resolution is an iterative process that requires understanding the data, writing and tuning the matching rules, and evaluating the resulting precision (are all matches correct?) and recall (did we miss any matches and why?). We summarize next the main issues in matching people mentioned in biographies to the actual person entities.

No standardization in entity names. People names come in different formats (e.g. "John A. Thain" vs. "Thain John" vs. "Mr. Thain", or "Murphy David J" vs. "Murphy David James III"). Hence, exact name matching will only find some matches and we need approximate name matching functions to resolve more biographies. On the other hand, two people with similar names (even when working for the same company) may be in fact two different people. For example, "Murphy David J" and "Murphy David James III" are two different people. To tackle this challenge, we designed specialized person name normalization and matching functions that cater for variations in names, suffixes such as "Jr.", "II", and allow matching names at varying precision levels. We iterated through our data and entity resolution results several times to fine-tune our functions.

Achieving high precision. To improve precision beyond just the use of name matching, we observed that for a biography record, we typically know the cik of the company (since it is the filing entity). As a result, we were able to develop matching rules that exploit such contextual information. In particular, the rules narrow the scope of matching to only consider the people entities that are already known to be officers or directors of the filing company (as computed from Forms 3/4/5). The final resulting precision was close to 100%.

Improving recall. To improve recall, in general, one needs multiple matching rules. For example, there are cases where the filer company is not in the employment history of a person (based on Forms 3/4/5). Hence, we must include other, more relaxed rules that are based just on person name matching. Having multiple rules, we prioritized them so that weaker matches are kept only when there are no matches based on stronger evidence. For instance, if we matched a "Thain John A" mentioned in a biography to both a "John A. Thain" and a "Thain John" in key people, via two different rules, we will only keep the first match since it is based on a rule that matches first/last name *and* middle name initial. Our initial rules achieved a 82.29% recall, that is, 82.29% of $23,195$ biographies were matched to a person cik. At the end of the tuning process, we raised that to 97.38%.

After Entity Resolution, a subsequent instantiation of the Map & Fuse stage takes place to perform the actual mapping and fusion of the linked records. The process is similar to what was described in the earlier Section 4.1, except that fusion is now with respect to an already existing set of entities. We omit any further details.

5 Future Directions

Even though aimed at a specific (financial) domain, building the target integrated dataset for Midas was a highly non-trivial task, which required about a year of intensive research and development that used multiple languages (e.g., Java, AQL, and Jaql) and packages on top of the Hadoop platform. We also note that traditional ETL or SQL/XML processing cannot be directly applied since the data is highly heterogeneous, with large variation in the number of attributes and data values; moreover the scale of the data can be daunting.

The longer-term vision here is to take a Midas-type of system[1] to the next level where one can easily develop and customize a sophisticated data integration flow in any given domain. Ideally, a small team of data analysts should be able to identify the important concepts or entity types that they are interested in, and then specify at

[1]Other examples, such as DBLife [13], exist here.

the high-level *what* needs to be accomplished rather than *how* to implement it. The *what* type of rules would incrementally specify the shape of the target entities and where they can be extracted and mapped from. They would also have to spell out, in a declarative way, the rules for linking between the various entity or entity references. Some research components already exist: tools for schema mapping [16], emerging formalisms for declarative entity resolution [3]. However, these often assume the existence of pre-defined schemas; moreover, they trade down the more advanced features (e.g., fusion, solving of conflicts, temporal aggregation) that are needed in a Midas-like system, in order to achieve simplicity or ease of use. The challenging aspect is in retaining the needed expressive power and flexibility while still having a high-level framework that spans all the stages of data integration, from extraction to entity resolution and mapping and fusion.

References

[1] V. Acharya, L. Pedersen, T. Philippon, and M. Richardson. Measuring Systemic Risk. SSRN: http://ssrn.com/abstract=1573171, 2010.

[2] T. Adrian and M. Brunnermeier. CoVaR. http://www.princeton.edu/∼markus/research/ papers/CoVaR.pdf, Princeton University, 2009.

[3] A. Arasu, C. Ré, and D. Suciu. Large-Scale Deduplication with Constraints Using Dedupalog. In *ICDE*, pages 952–963, 2009.

[4] S. Balakrishnan, V. Chu, M. A. Hernández, H. Ho, R. Krishnamurthy, S. Liu, J. Pieper, J. S. Pierce, L. Popa, C. Robson, L. Shi, I. R. Stanoi, E. L. Ting, S. Vaithyanathan, and H. Yang. Midas: Integrating Public Financial Data. In *SIGMOD*, pages 1187–1190, 2010.

[5] K. Beyer and V. Ercegovac. Jaql: A Query Language for JSON, 2009. http://code.google.com/p/jaql/.

[6] M. Billio, M. Getmansky, A. Lo, and L. Pelizzon. Econometric Measures of Systemic Risk in the Finance and Insurance Sectors. SSRN: http://ssrn.com/abstract= 1648023, 2010.

[7] P. Bonacich. Power and Centrality: A Family of Measures. *American Journal of Sociology*, 92(5):1170–1182, 1987.

[8] Y. Cai and M. Sevilir. Board Connections and M&A Transactions. SSRN: http://ssrn.com/abstract=1491064, 2009.

[9] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, and S. Vaithyanathan. SystemT: An Algebraic Approach to Declarative Information Extraction. In *ACL*, 2010.

[10] L. Chiticariu, R. Krishnamurthy, Y. Li, F. Reiss, and S. Vaithyanathan. Domain Adaptation of Rule-Based Annotators for Named-Entity Recognition Tasks. In *EMNLP*, 2010.

[11] L. Chiticariu, Y. Li, S. Raghavan, and F. Reiss. Enterprise Information Extraction: Recent Developments and Open Challenges. In *SIGMOD*, pages 1257–1258, 2010.

[12] N. N. Dalvi, R. Kumar, B. Pang, R. Ramakrishnan, A. Tomkins, P. Bohannon, S. Keerthi, and S. Merugu. A Web of Concepts. In *PODS*, pages 1–12, 2009.

[13] P. DeRose, W. Shen, F. Chen, Y. Lee, D. Burdick, A. Doan, and R. Ramakrishnan. DBLife: A Community Information Management Platform for the Database Research Community. In *CIDR*, pages 169–172, 2007.

[14] A. Doan, J. F. Naughton, R. Ramakrishnan, A. Baid, X. Chai, F. Chen, T. Chen, E. Chu, P. DeRose, B. J. Gao, C. Gokhale, J. Huang, W. Shen, and B.-Q. Vuong. Information Extraction Challenges in Managing Unstructured Data. *SIGMOD Record*, 37(4):14–20, 2008.

[15] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate Record Detection: A Survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[16] R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema Mapping Creation and Data Exchange. In *Conceptual Modeling: Foundations and Applications, Essays in Honor of John Mylopoulos*, pages 198–236. Springer, 2009.

[17] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data Integration: The Teenage Years. In *VLDB*, pages 9–16, 2006.

[18] B.-H. Hwang and S. Kim. It Pays to Have Friends. *Journal of Financial Economics*, 93(1):138–158, 2009.

[19] P. G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *PODS*, pages 61–75, 2005.

[20] M. Kritzman, Y. Li, S. Page, and R. Rigobon. Principal Components as a Measure of Systemic Risk. SSRN: http://ssrn.com/abstract=1582687, 2010.

CALL FOR PAPERS

Data Management in the Cloud (DMC) Workshop 2012

In Conjunction with the IEEE International Conference on Data Engineering (ICDE 2012)

April 5, 2012, Washington DC, USA

http://www.nec-labs.com/dm/dmc2012/

Workshop Organizers

Hakan Hacıgümüş, NEC Labs
Bijit Hore, University of California, Irvine
Jagan Sankaranarayanan, NEC Labs

Program Committee

Amr El Abbadi, UC Santa Barbara
Michael Carey, UC Irvine
Brian Cooper, Google
Carlo Curino, MIT
Sudipto Das, UC Santa Barbara
Michael Franklin, UC Berkeley
Donald Kossmann, ETH Zurich
Tim Kraska, UC Berkeley
Paul Larson, Microsoft
David Lomet, Microsoft
Sam Madden, MIT
Jayant Madhavan, Google
Hyun Jin Moon, NEC Labs
Jeffrey Naughton, University of Wisconsin
Jignesh Patel, University of Wisconsin
Neoklis Polyzotis, UC Santa Cruz
Donavan Schneider, SalesForce.com
Prashant Shenoy, University of Massachusetts, Amherst
Swami Sivasubramanian, Amazon
Junichi Tatemura, NEC Labs

The DMC Workshop

The cloud computing has emerged as a promising computing and business model. By providing on-demand scaling capabilities without any large upfront investment or long-term commitment, it is attracting wide range of users. The database community has also shown great interest in exploiting this new platform for data management services in a highly scalable and cost-efficient manner. As a result, the cloud computing presents challenges and opportunities for data management. The DMC workshop aims at bringing researchers and practitioners in cloud computing and data management systems together to discuss the research issues at the intersection of those areas, and also to draw more attention from the larger data management research community to this new and highly promising field.

Topics of Interest

The DMC workshop calls for contributions that address fundamental research and system issues in cloud data management including but are not limited to the following topics.

- Elasticity for Could Data Management Systems
- Resource and Workload Management in Cloud Databases
- Multi tenancy
- High Availability and Reliability in Cloud Databases
- Cloud computing infrastructure designs for cloud data services
- Transactional models for Cloud Databases
- Distributed and Massively Parallel query processing
- Storage architectures and technologies for cloud databases
- Privacy and security in cloud data management
- Mobile cloud data management
- Cross-platform interoperability
- Service Level Agreements
- Economic/business models and pricing policies
- Novel data-intensive/data-rich computing applications
- Virtualization and Cloud Databases

Submissions Guidelines

Authors should submit papers reporting original works that are currently not under review or published elsewhere. All submissions must be prepared in the IEEE camera-ready format. The workshop solicits both full papers and short papers (e.g. experience reports, preliminary reports of work in progress, etc.). Full papers should not exceed 8 pages in length. Short papers should not exceed 4 pages.

All accepted papers will be published in the ICDE proceedings and will also become publicly available through the IEEE Xplore.

All papers should be submitted in PDF format using the online submission system at:

https://cmt.research.microsoft.com/DMC2012/

Important Dates

Paper submission deadline: October 1, 2011
Author Notification: November 1, 2011
Final Camera-ready Copy Deadline: November 08, 2011

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398