Bulletin of the Technical Committee on

Data Engineering

June 2006 Vol. 29 No. 2

IEEE Computer Society

Letters

Special Issue on Data Quality

Conference and Journal Notices

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.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

There are two Data Engineering Bulletin web sites: http://www.research.microsoft.com/research/db/debull and http://sites.computer.org/debull/.
The TC on Data Engineering web page is http://www.ipsi.fraunhofer.de/tcde/.

Letter from the Editor-in-Chief

New and Retiring Bulletin Editors

Every two years, I appoint a new set of issue editors for the Data Engineering Bulletin. This is the single most important thing that I do as the Bulletin Editor-in-Chief. I am very proud of the distinguished record that these editors have established over the years. I want to thank them, one and all, for the great jobs done over the years.

It is now time to thank the editors who have just completed their terms. Each has done the two issues that our "contract" requires. These retiring editors are Gustavo Alonso of ETH, Minos Garofalakis of Intel Research, Meral Ozsoyoglu of Case Western Reserve, and Jignesh Patel of the University of Michigan. Thank you one and all for a fine job. Special thanks also goes to those who assisted significantly in the editorial process over the past two years, Gultekin Ozsoyoglu of Case Western Reserve and Dan Suciu of University of Washington.

I am very pleased to now introduce the Bulletin issue editors for the next two years. They represent on-going evidence that the Bulletin is held in high regard and continues to attract outstanding members of the database community. These new editors are Anastassia Ailamaki of Carnegie Mellon University, Jayant Haritsa of the Indian Institute of Science, Nick Koudas of the University of Toronto, and Dan Suciu of the University of Washington. Parenthetically, Dan has already been a co-editor of the March 2006 issue. I want to both welcome them to the Bulletin and also thank them for joining us in what, in the final analysis, represents a great cooperative venture to deliver timely reports on the very latest that is happening in the database field.

Nominations for Chair of the TC on Data Engineering

I want to call your attention to the letter following this one from the Nominating Committee for the election of a new chair for the Technical Committee on Data Engineering (TCDE). The TCDE is the organization that sponsors the publication of the Bulletin as well as the International Conference on Data Engineering (ICDE). The chair position has significant responsibilities and TCDE members should take an interest in the election. Please read the next letter and consider nominating someone as TCDE Chair.

The Current Issue

When people ask questions of a web search engine, their expectation is that the results produced will represent a best effort. They anticipate scanning, with human eyeballs, the results in order to discover the most useful and accurate information from a potentially vast sea of possible results. Users of database systems have very different expectations. When they submit a query, they expect that the answer will be precisely what they requested and that it will accurately reflect the situation about which they care. This can be essential, as in the case where it is not a human, but a computer, that will be interpreting and acting on the delivered results.

Unfortunately, even data in dtaabases needs to come from somewhere. And that somewhere is only sometimes other database systems. Ultimately, all data originates from outside of the database system. So what are we to make of this data? It is, in fact, not always accurate, it is sometimes simply imprecise, at other times contradictory, and frequently redundant. This is where one needs to consider, and hopefully solve, the problem of data quality. And it is data quality that is the subject of the current issue.

Nick Koudas knows this area and the researchers, both industrial and academic, who are actively engaged in sorting out the very complex and demanding problems surrounding data quality. The current issue represents a snapshot of some of the very exciting work going on in this area. I want to thank Nick for his efforts in bringing this issue together. I am sure that readers will find articles of immediate interest and will want to come back again to the this issue to re-examine the work in this field. That is, indeed, a core strength of the Bulletin, giving readers a status report on research that represents an on-going effort to deal with the hard problems of our field.

David Lomet
Microsoft Corporation

Call for Nominations Letter

Election of a New Chair for the TC on Data Engineering

The Chair of the IEEE Computer Society Technical Committee on Data Engineering (TCDE) is elected for a two-year period. The mandate of the current Chair, Erich Neuhold, expires at the end of this year and since this is his second term, he is ineligible to be nominated as chair for the next term. A Nominating Committee for electing a chair for the period 2007-2008 consisting of Betty Salzberg (salzberg@ccs.neu.edu), David Lomet (lomet@microsoft.com) and Erich Neuhold (neuhold@ipsi.fhg.de) has been struck. The Nominating Committee invites nominations for the position of Chair from all members of the TCDE. To submit a nomination, please contact any member of the Nominating Committee before August 25, 2006.

More information about TCDE can be found at http://ipsi.fhg.de/tcde. Information about TC elections can be found at http://www.computer.org/tab/hnbk/electionprocedures.htm.

Betty Salzberg, David Lomet, Erich Neuhold
TCDE Nominating Committee

Letter from the Special Issue Editor

This issue of the Data Engineering Bulletin is devoted to the topic of data quality. Data quality is an important topic of vast business importance. Quality problems significantly degrade common business practices (with billing being a prominent example). Typical quality problems include incorrect person or business names, inconsistent address information, poor integrity constraints, missing keys to name a few. It is estimated that data quality problems cause approximately 600 billion dollars of financial loss in the US alone each year. Data quality has been in the center of research interest for more than thirty years. Although several steps have been made towards addressing major problems and an array of industrial tools are available, fundamental problems remain still widely unsolved.

The issue contains seven articles authored by leading experts in the area. The specific collection of articles was chosen in a way to highlight current research directions in data quality as well as new interesting approaches to known problems.

The first two articles present new approaches to the problem of *entity resolution*. The article by Bhattacharya and Getoor addresses the problem of collective entity resolution, namely exploring relationships or ties between entities in a way that related entities are resolved collectively. The article by Benjelloun et. al., is an overview of the approach to entity resolution by the Stanford SERF project. In this approach match predicates are considered as black boxes and algorithms are presented to merge entities efficiently.

The article by Naumann et. al., presents an interesting approach to data fusion by considering a multi-level approach. In particular the authors present the results of their research to merging data from different sources by systematically tracking inconsistencies at the value, tuple as well as schema level. Laure Berti-Equille discusses quality aware query processing. This is a relatively new and highly important problem. Traditional approaches to data quality consist of techniques to 'clean' data in order to enable query processing on the clean image of a data set. Quality aware query processing aims to enable query processing on 'dirty' data (data with various quality problems) and provide tradeoffs between query processing cost and result quality.

Dasu et. al., present work in the context of the Bellman data quality tool from AT&T Research. A very interesting direction in data quality deals with data base exploration for quality problems. It is highly important to provide tools and methodologies to identify data quality problems, in addition to providing techniques to resolve them once they have been identified. The Bellman tool is an important step in this area. The article explores data mining on database dynamics. In particular is performs data mining to explore database changes as a function of time. Chaudhuri et. al., present work in the context of the Data Debugger project at Microsoft Research. The main goal of the project is to identify generic and robust abstractions for data cleaning operators and to support efficient implementations of these abstractions. Doing so will enable composability of operators (relational and non relational) to derive operator trees. In the paper the authors illustrate their approach using the record matching operation.

Finally the article by Milano et. al., deals with addressing quality problems in non relational data sources. In particular it presents approaches for resolving identities for XML data taking into account the specifics of such data and in particular the underlying document structures.

I would like to take this opportunity to thank the contributors to this special issue once again for the time and effort they have put in delivering the articles. I would like to also thank the editor in chief of the publication Dr. David Lomet for his availability and timely response and handling of all matters related to the issue. Mr. Dimitris Tsirogiannis provided valuable assistance in assembling the issue and I would like to thank him for his time. I hope readers will enjoy the article collection as much as I have.

Nick Koudas
University of Toronto
Toronto, Canada

Collective Entity Resolution In Relational Data

Indrajit Bhattacharya and Lise Getoor
Department of Computer Science
University of Maryland, College Park, MD 20742, USA

Abstract

An important aspect of maintaining information quality in data repositories is determining which sets of records refer to the same real world entity. This so called entity resolution problem comes up frequently for data cleaning and integration. In many domains, the underlying entities exhibit strong ties between themselves. Friendships in social networks and collaborations between researchers are examples of such ties. In such cases, we stress the need for collective entity resolution where, instead of independently tagging pairs of records as duplicates or non-duplicates, related entities are resolved collectively. We present different algorithms for collective entity resolution that combine relational evidence with traditional attribute-based approaches to improve entity resolution performance in a scalable manner.

1 Introduction

There has been an increase in automated acquisition and integration for data repositories and information sources and, because completely manual curation is impossible in all but the smallest databases, there has been an increasing dependence on automated techniques for maintaining data integrity and quality of information. While we have seen a surge in research interest in this area over the last decade, the problems are quite challenging. Because accuracy is critical in many applications, there is need for further improvement. In addition to the attributes of records that have traditionally been used by data cleaning and integration algorithms, quite often there may be relationships between different database records. In such cases, the models and algorithms for data cleaning can take such relationships into account to improve performance.

Entity resolution is an important problem that comes up frequently for cleaning and integration. In many databases, records refer to real world entities, and as such databases grow, there can many different records that refer to the same entity. For example, a social network database can have different records with names 'J. Doe', 'Jonathan Doe' and 'Jon Doe' that refer to the same person. In the absence of keys such as social security numbers, this duplication issue [13, 15] leads to many different problems, such as redundant records, incorrectness of computed statistics, and several others. This issue also comes up when integrating data from different heterogeneous sources without shared keys and possibly even different schemas [10]. Broadly, we call such database records *references* to real world entities, and the entity resolution problem is to find the underlying *entities* in the domain and tag the references in the database with the entities to which they correspond.

Entity resolution is a difficult problem and cannot be solved using exact matches on tuple attributes. First, there is the *identification* problem, when different representations arising from recording errors or abbreviations

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

refer to the same entity. In our earlier example, figuring out that 'Jon Doe' and 'Jonathan Doe' are the same person is an instance of this problem. Failure in identification affects completeness of information. The second issue is *disambiguation*. It is possible that two records with name 'J. Doe', the same address and the same age refer to two brothers and not to the same person. This affects precision. Early approaches to entity resolution prescribed fuzzy attribute matches and many sophisticated techniques have been developed. However, attribute-based approaches still cannot satisfactorily deal with the problem of false attribute matches and, in general, it may be hard to improve precision and completeness at the same time using just attributes of records.

In many domains, some underlying entities exhibit strong relational ties to certain other entities. For instance, people interact frequently with their close friends in a social network, while in academic circles, researchers collaborate more with their close associates. When such ties exist between entities, co-occurrences between the references to these entities can be observed in the data. In the social network example, we may have the records for the best friends of 'J. Doe' and 'Jon Doe'. Our goal will be to make use of such relationships between references to improve entity resolution performance. However, the problem is that we do not know the entities for these related records either. So how can we use these relations then? One way is to use the attributes of related records as well when computing fuzzy matches. While this is an improvement, it may not always work. For example, we do not want to merge two person records simply because their best friends have similar names. The correct evidence to use is whether their best friends are in fact the same entity. This is the idea behind *collective entity resolution*, where the entity for any reference depends on the entities for its related references. Computationally, it is a more difficult problem to solve than attribute-based resolution. The database cannot be cleaned with a single-pass approach anymore because of the dependent nature of the resolutions. We need to resort to iterative approaches, where each resolution that we make potentially provides evidence to discover other duplicate references. However, there is also the promise that the resolution accuracy can be significantly improved over traditional techniques. In this article, we present a survey of algorithms we have proposed in earlier work [3, 4, 5, 6] that address the computational challenge of collective resolution and combine attributes of records with relational evidence to improve entity resolution performance.

2 Problem Formulation

In domains where the data contains relationships between different entity references, these may be represented using an auxiliary table of relations. We now introduce a generic notion of a *reference database* that records information about references and the relationships between them that are observed in the data. Then we describe the entity resolution problem in such a reference database using examples to illustrate the various issues involved.

2.1 Reference Database

In the simplest formulation, a reference database contains a table of references, $\mathcal{R} = \{r_i\}$, where each reference has an identifier $\mathcal{R}.id$ and a set of attributes $\{\mathcal{R}.A_1, \ldots, \mathcal{R}.A_k\}$. Also, we have the unobserved domain entities $\mathcal{E} = \{e_j\}$. For any particular reference r_i, we denote the entity to which it maps as $E(r_i)$. We will say that two or more references are **co-referent** if they correspond to the same entity. Note however that the database is unresolved — the references do not have any identifiers that disclose the mapping $E(r_i)$. Further, the domain entities $\mathcal{E}$ and even the number of such entities is not known. To model relationships between references in a generic way, we use a hyper-edge table $\mathcal{H}$ with identifier $\mathcal{H}.id$ and attributes $\{\mathcal{H}.A_1 \ldots \mathcal{H}.A_l\}$. Each hyper-edge connects multiple references. We use a mapping table $\mathcal{M} = \{hid, rid\}$ to associate the reference rid to the hyper-edge hid. For convenience, we use the notation $r \in h$ to mean that a reference $r \in \mathcal{R}$ is associated with a hyper-edge $h \in \mathcal{H}$: $r \in h \iff (r.id, h.id) \in \mathcal{M}$. Note that each reference may be associated with zero or more hyper-edges.

Let us now look at a sample domain to see how it can represented in our framework. We consider as our motivating example a database of academic publications similar to DBLP, CiteSeer or PubMed.[1] We consider the problem of resolving the authors of the publications. Each publication in the database has a set of author names. For each author name, we have a reference r_i in $\mathcal{R}$ and $r_i.Name$ records the observed name of the author in the publication. In addition, we can have attributes $\mathcal{R}$.*Affil* and $\mathcal{R}.Email$ to record the affiliation and email of each author reference if they are available in the paper. Additionally, each publication represents a co-author relationship among the references in it. So we have an entry h_i in the hyper-edge table $\mathcal{H}$ for each publication and an tuple $(h_i.id, r_j.id)$ in the mapping table $\mathcal{M}$ for each reference r_j in a publication h_i. If a publication also comes with additional information, such as title, these are represented as attributes $(\mathcal{H}.Title)$ of the hyper-edge table $\mathcal{H}$. While in general our representation allows each reference to belong to zero or more hyper-edges, in this domain each author-name in a paper is a distinct reference and therefore occurs in exactly one hyper-edge.

As an example, consider the following four papers.

1. W. Wang, C. Chen, A. Ansari, "A mouse immunity model"
2. W. Wang, A. Ansari, "A better mouse immunity model"
3. L. Li, C. Chen, W. Wang, "Measuring protein-bound fluxetine"
4. W. W. Wang, A. Ansari, "Autoimmunity in biliary cirrhosis"

These may be represented in our notation with 10 references $\{r_1, \ldots, r_{10}\}$ in the reference table $\mathcal{R}$, where r_1 is $(id$ 1; $Name$ 'Wang W'), etc. There are 4 entries $\{h_1, \ldots, h_4\}$ in the hyper-edge table $\mathcal{H}$ for the four papers, where h_1 is $(id$ 1; $Title$ 'The mouse immunity model') and so on. The mapping table $\mathcal{M}$ also has 10 entries, one for each reference, to record which reference appears in which paper. For example, the entry $(hid$ 1; rid 1) records that reference r_1 appears in hyper-edge h_1. This is represented pictorially in Figure 1(a).

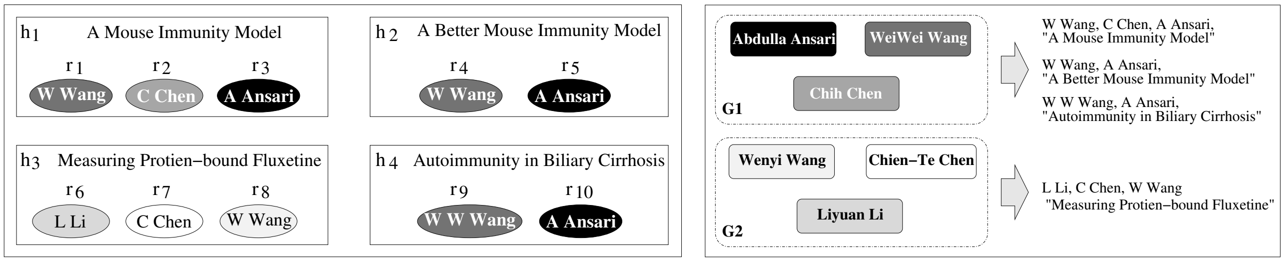
2.2 Entity Resolution Problem in a Reference Database

Given the formulation of a reference database, the entity resolution task is to partition or cluster the references according to their underlying entities. To illustrate this for our example, suppose we have six underlying entities, which are shown in Figure 1(a) using six different shades. All references with name 'A. Ansari' are co-referent, as are all the 'L. Li' references. However, the two 'C. Chen's are *not* co-referent and map to two different entities. More interestingly, the four references with name 'Wang' map to two different entities. References r_1, r_4, and r_9 are co-referent, while r_8 maps to a different entity.

A natural task in a reference database is to take all references with a given name and partition them according to the entities to which they correspond. We refer to this as the **disambiguation task**. Consider the name 'W. Wang'. In our example, there are three author references for 'W. Wang': r_1, r_4, and r_8. Our goal is to partition these identically named references according to entities. Then the correct disambiguation for 'W. Wang' is $\{\{r_1, r_4\}, \{r_8\}\}$ indicating that r_1 and r_4 map to the same entity and r_8 maps to a distinct entity. The complete disambiguation for the database would cover the other references as well.

Observe that the disambiguation task handles one part of the resolution process. In our example, while it finds the co-referent pairs with name 'W. Wang', it does not consider references whose names are not exact matches. However, reference r_9 from the fourth paper is co-referent with r_1, even though it has a different recorded name. So, the r_9 reference from the fourth paper should be included in the same entity cluster as the r_1 reference. Therefore, in addition to disambiguation, we need to 'identify' coreferences with different names as well. To handle this, we define the **entity resolution task** as a partitioning *all* references in the database

[1]However, this entity resolution framework is general enough to handle application domains such as customer relationship management, personal information management and others that involve references, entities and complex relationships.



(a) (b)

Figure 1: (a) Example papers represented as references connected by hyper-edges, with different entities shaded differently (b) Two underlying groups of collaborating entities, with their generated papers listed alongside.

according to the entities to which they correspond. The entity resolution result for our example should return six clusters: $\{\{r_1, r_4, r_9\}, \{r_8\}, \{r_2\}, \{r_7\}, \{r_3, r_5, r_{10}\}, \{r_6\}\}$. The first two clusters correspond to 'Wang', the next two to 'Chen', the fifth to 'Ansari' and the last to 'Li'.

2.3 Entity Resolution Approaches

Different approaches may be used to resolve the references in a database. Here we briefly look at the intuition behind three of the prevalent ones.

1. Attribute-based Entity Resolution: This is the traditional approach where similarity is computed for each pair of references based on their attributes and only those pairs that have similarity above some threshold are considered to be co-referent. Many sophisticated and efficiently computable similarity measures have been proposed for different types of attributes over the years. However, attributes alone often run into problems, as in the case of the three 'W. Wang' references in our example.

2. Naive Relational Entity Resolution: When relations between references are available, this approach considers the attributes of the related references when computing similarity between pairs of references. In our running example, when computing the similarity between 'W. Wang' and 'W. W. Wang', it would take into account that both have co-authors with name 'A. Ansari'.

3. Collective Entity Resolution: While the naive relational approach improves significantly on the attribute-based approach, it can be misled in domains where many entities have the same name and the relationship graph is dense. In our example, the two 'W. Wang' references r_1 and r_8 are not co-referent, though they both have co-authors with name 'C. Chen'. The correct evidence to use here is that the 'Chen's are not co-referent either. In such a setting, in order to resolve the 'W. Wang' references, it is necessary to *resolve* the 'C. Chen' references as well, and not just consider them as attributes. This is the goal of collective entity resolution, where resolutions are not made independently. Instead one resolution decision affects other resolutions via hyper-edges. This increases the computational expense of the resolution process but improves accuracy significantly in ambiguous domains.

For the first two approaches, all that is needed is a similarity measure between pairs of references. Given such a similarity measure, the algorithm for resolving entities is straight-forward — those reference pairs that have similarity above a given threshold are declared to be co-referent. However, collective entity resolution is more involved. Specifically, the dependencies between the different resolution decisions need to be modeled. Also, as we have already mentioned, the algorithm needs to make multiple passes over the references to capture the dependencies. We next describe two approaches to collective entity resolution that we have developed.

3 Algorithms for Collective Resolution

We first describe a clustering approach to collective resolution and then briefly discuss a probabilistic generative model for the same problem and how we can do inference in it.

3.1 Relational Clustering

Given that the goal of entity resolution is to cluster the database references according to their entities, we have developed a relational clustering algorithm for entity resolution (**RC-ER**) [3]. Given a current set of reference clusters $\mathcal{C} = \{c_i\}$, it iteratively merges the pair of clusters that are the most similar. We associate a cluster label $r.C$ with each reference to denote its current cluster membership. Note that it is the similarity measure that distinguishes the different entity resolution approaches. For the attribute-based approach, the similarity only considers the attributes of references. For the naive relational approach, it additionally considers the attributes of related references. The collective approach, in contrast, considers the cluster labels of the related references.

The similarity of two clusters c_i and c_j is defined as

$$sim(c_i, c_j) = (1 - \alpha) \times sim_A(c_i, c_j) + \quad \alpha \times sim_R(c_i, c_j) \quad 0 \leq \alpha \leq 1 \tag{1}$$

where $sim_A()$ is the similarity of the attributes and $sim_R()$ is the relational similarity between the references in the two clusters. The most important and interesting aspect of the collective approach is the dynamic nature of the similarity. In contrast to attribute-based and naive relational resolution, where the similarity between two references is fixed, for collective resolution it depends on the *current* cluster labels of the references and therefore changes with the labels. In our example, the similarity of the two references 'W. Wang' and 'W. W. Wang' increases once the Ansari references are given the same cluster label. Let us now see how the two components of the similarity are computed.

Attribute Similarity: For each reference attribute, we assume the existence some basic similarity measure that takes two reference attributes and returns a value between 0 and 1 that indicates the degree of similarity between them. In addition, if the hyper-edges have attributes, then the attribute similarity of two references can also take into account the attributes of the hyper-edges with which they are associated. Several sophisticated similarity measures have been developed for names, and popular TF-IDF schemes may be used for other textual attributes such as keywords. The measure that works best for each attribute may be plugged in. Finally, a weighted combination of the similarities over the different attributes yields the combined attribute similarity between two reference clusters.

Relational Similarity: For collective entity resolution, relational similarity considers the cluster labels of the references that each cluster is connected to via the hyper-edges. There are many possible ways to define this similarity; here we discuss one of measures that we have proposed [3, 5].

The hyper-edges relevant for a cluster are the hyper-edges for all references in it. Recall that each reference r is associated with one or more hyper-edges in the hyper-edge table $\mathcal{H}$. Therefore, the hyper-edge set $c.H$ for a cluster c of references is defined as

$$c.H = \bigcup_{r \in \mathcal{R} \wedge r.C = c} \{hid \mid (hid, rid) \in \mathcal{M} \ \wedge r.id = rid\} \tag{2}$$

This set defines the hyper-edges that connect a cluster c to other clusters, and are the ones that relational similarity needs to consider. For instance, when all the references in our running example have been correctly clustered as in Figure 1(b), the edge-set for the larger 'W. Wang' cluster is $\{h_1, h_2, h_4\}$, which are the hyper-edges associated with the references r_1, r_4 and r_9 in that cluster.

The different clusters to which any cluster c of references is connected via its hyper-edge set is called the neighborhood $Nbr(c)$ of cluster c.

$$Nbr(c_i) = \bigcup_{h \in c.H, r \in h} \{c_j \mid c_j = r.C\} \tag{3}$$

Returning to our example, the neighborhood of the 'W. Wang' cluster mentioned above consists of the 'Ansari' and the 'Chen' clusters, which are connected by its edge-set. Now, for the relational similarity measure between two clusters, their neighborhoods are compared using set similarity such as Jaccard similarity:

$$sim_R(c_i, c_j) = Jaccard(Nbr(c_i), Nbr(c_j)) \tag{4}$$

Recall that for two sets A and B, their Jaccard similarity is defined as $Jaccard(A, B) = \frac{|A \bigcap B|}{|A \bigcup B|}$. The similarity can be computed and updated efficiently, in time that is linear in the average number of neighbors per cluster.

Clustering Algorithm: Given the similarity measure for a pair of clusters, a greedy agglomerative clustering algorithm is used for collective entity resolution. The algorithm bootstraps the clusters, identifies the candidate set of potential duplicates and iterates over the following steps. At each step, it identifies the current 'closest pair' of clusters (c_i, c_j) from the candidate set and merges them to create a new cluster c_{ij}. It identifies new candidate pairs and updates the similarity measures for the 'related' cluster pairs. All of these tasks are performed efficiently using an indexed priority queue. The algorithm terminates when the similarity for the closest pair falls below a threshold.

3.2 Probabilistic Group Model

In addition to the relational clustering algorithm, we have also developed a probabilistic generative model for collective entity resolution [6], which we call the Latent Dirichlet Allocation model for Entity Resolution, or **LDA-ER** for short. It describes how the author references in any paper might be generated. Instead of modeling pair-wise collaboration relations between author entities, the novelty of the model is that it uses the notion of collaborating *groups of entities*. For our example, the six relevant entities belong to two different groups, as shown in Figure 1(b). The generative process for each paper first selects one or more groups that collaborate to write the paper. Then each author for the paper is chosen from one of these selected groups. The true name of an author entity determines what the reference name in a paper might be. In the example, papers 1, 2 and 4 are generated by collaborating entities from group G1, while paper 3 is written by entities from group G2. Note that for the author entity with true name "WeiWei Wang", the reference name is "W. Wang" in two of the papers and "W. W. Wang" in another.

We have developed a Gibbs Sampling algorithm for doing inference in this model. Starting from an initial assignment of groups and entities for the references, the algorithm repeatedly samples the group and entity for each reference given those for the others until a stationary distribution is reached. In our example, the algorithm is expected to predict that the 'Wang' references in papers 1, 2 and 4 are likely belong to the same group, and therefore they are more likely to map to the same entity. The other 'Wang' reference in paper 3 maps to a different entity, since most probably it belongs to a different group. Also, one interesting aspect of our inference algorithm is that number of entities does not need to specified as a parameter — it automatically determines the most likely number of entities given the reference database. Another important aspect is that the inference algorithm is completely unsupervised. This is significant given the scarcity of training data for this problem.

4 Experimental Results

We have evaluated our collective entity resolution algorithms [3, 4, 5, 6] for the task of author resolution in synthetic as well real-world citation databases such as CiteSeer (2,892 author references from Machine Learning), arXiv (58,515 author references from High Energy Physics) and BioBase (831,991 author references from

Biology). Here we present an overview of our results. The first baseline (**A**) that we compare against uses only attributes of the references for resolution, while the second (**A+N**) additionally uses attributes of neighboring or related references. We also consider the variants **A*** and **A+N*** that take transitive closures over the pair-wise decisions made in **A** and **A*** respectively. For evaluating entity resolution performance, we use the popular F1-measure (the harmonic mean of precision and recall) of the pair-wise decisions over all references.

In Table 1, we show the performance of our relational clustering algorithm algorithm **RC-ER** against the baselines in the three datasets. The best performance for each dataset is shown in bold. We can see that **RC-ER** outperforms the baselines in all cases. Also, the improvement over the baselines increases as we move from CiteSeer to arXiv and then to BioBase. The improvement using collective resolution depends on how densely the references are related to each other and also on what fraction of the references names are ambiguous, or in other words, are shared by more than one entity. The results confirm this since both the density of relations and ambiguity of reference attributes in highest for BioBase and lowest for CiteSeer, which explains the difference in performance. We experimented with different attribute similarity measures and we observed similar improvements with all of them. Performance using our probabilistic model **LDA-ER** is very similar to that of **RC-ER**.

Table 1: Entity resolution performance (F1-measure) of five algorithms on three datasets. Results are for the entire CiteSeer and arXiv datasets and for the 100 most frequent names in BioBase.

	A	**A***	**A+N**	**A+N***	**RC-ER**
CiteSeer	0.980	0.990	0.973	0.984	**0.995**
arXiv	0.974	0.967	0.938	0.934	**0.985**
BioBase	0.701	0.687	0.710	0.753	**0.818**

While Table 1 records improvements over the entire CiteSeer and arXiv datasets, the strength of collective resolution clearly stands out when we look at specific instances of ambiguous names. When a name or its abbreviation is shared between multiple entities, it is hard to resolve different references having that name using attributes alone. In Table 2, we show some examples of ambiguous names from arXiv and the performance of the attribute baselines and our **LDA-ER** model only over references that have this abbreviated name. We can see that for all such cases collective resolution out-performs the baselines by very large margins.

Table 2: Entity resolution performance (F1-measure) for the **LDA-ER** model and the best baseline performance for some example ambiguous names from the arXiv dataset.

	Cho H	Davis A	Sarkar S	Sato H	Shin H	Veselov A	Yamamoto K	Yang Z	Zhang R	Zhu H
Best of A/A*	0.80	0.67	0.67	0.82	0.69	0.78	0.29	0.77	0.83	0.57
LDA-ER	**1.00**	**0.89**	**1.00**	**0.97**	**1.00**	**1.00**	**1.00**	**0.97**	**1.00**	**1.00**

We have done extensive evaluations of the different aspects of our models and algorithms. Figure 2 shows some sample plots. Figure 2(a) shows how performance changes with the combination weight α between attribute and relational similarity for arXiv. We also experimented with synthetic data to see how different structural properties in the data affect the algorithms. Figure 2(b) plots one of the trends, which shows that expected improvements using **LDA-ER** are higher when each relation covers more references on average. Finally, Figure 2(c) shows how **RC-ER** scales with data size once the potential duplicate pairs have been identified. We can see that it takes longer than the attribute baseline, but the growth is still linear.

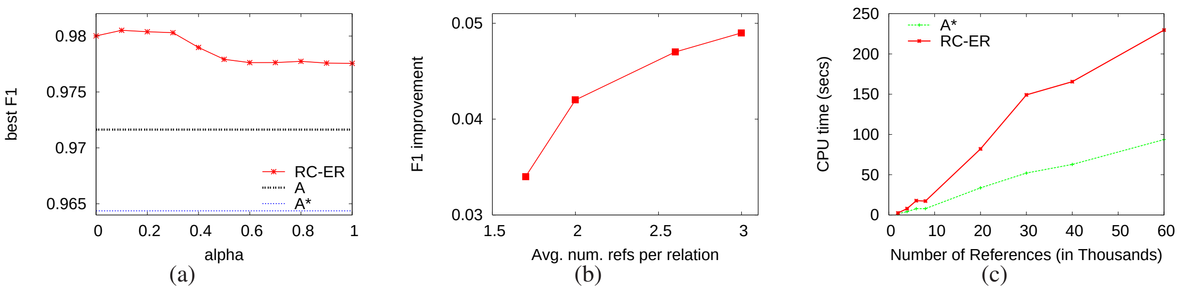


Figure 2: (a) Entity resolution performance using **RC-ER** versus combination weight α for arXiv. (b) Improvement over **A*** using **LDA-ER** against average number of references in each relation. (c) Execution time of **RC-ER** and **A*** for increasing number of references in the data.

5 Related Work

The entity resolution problem has been studied in many different areas under different names — deduplication, record linkage, co-reference resolution, reference reconciliation etc. Most of the work has focused on traditional attribute-based entity resolution. Extensive research has been done on defining approximate string similarity measures [15, 7, 8] that may be used for unsupervised entity resolution. The other approach is to use adaptive supervised algorithms that learn similarity measures from labeled data [18]. The WHIRL system [9] has been proposed for data integration using similarity join queries over textual attributes. Swoosh [2] is generic entity resolution framework that minimizes the number of record-level and feature-level operations when resolving and merging duplicates. Probabilistic techniques have been proposed for quick similarity computation between tuples for fast text-joins [12] and for efficiently looking up candidate matches for incoming tuples [8].

Many recent approaches take relations into account for data integration [1, 3, 5, 14, 11, 16, 17]. Ananthakrishna et al. [1] introduce relational deduplication in data warehouse applications where there is a dimensional hierarchy over the relations. Neville et al. [16] have shown how relations may be combined with attributes for clustering. Kalashnikov et al. [14] enhance attribute similarity between an ambiguous reference and the many entity choices for it with relationship analysis between the entities, such as affiliation and co-authorship. Dong et al. [11] collectively resolve entities of multiple types by propagating relational evidences in a dependency graph, and demonstrate the benefits of collective resolution in real datasets. Singla et al. [17] propose a probabilistic model based on conditional random fields that exploits similar dependencies.

6 Conclusion and Future Directions

Entity resolution is an area that has been attracting growing attention to address the influx of structured and semi-structured data from a multitude of heterogeneous sources. Accurate resolution is important for a variety of reasons ranging from cost-effectiveness and reducing redundancy in data to accurate analysis for critical applications. We have found collective entity resolution to be a powerful and promising technique that combines attribute similarity with relational evidence and significantly improves performance over traditional approaches. The improvements using relations are more dramatic in databases where names are more likely to be ambiguous. While collective resolution is more expensive than attribute-based resolution, the computational cost is not prohibitive. As future directions, we are interested in localized entity resolution, incremental updates and in challenging and important domains such as geo-spatial databases and others with unstructured context as in email archives.

References

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, 2002.

[2] O. Benjelloun, H. Garcia-Molina, Q. Su, and J. Widom. Swoosh: A generic approach to entity resolution. Technical Report, Stanford University, March 2005.

[3] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, June 2004.

[4] I. Bhattacharya and L. Getoor. Relational clustering for multi-type entity resolution. In *KDD Workshop on Multi-Relational Data Mining (MRDM)*, 2005.

[5] I. Bhattacharya and L. Getoor. *Entity Resolution in Graphs*, Chapter in Mining Graph Data (Lawrence B. Holder and Diane J. Cook, eds.). Wiley, 2006.

[6] I. Bhattacharya and L. Getoor. A latent dirichlet model for unsupervised entity resolution. In *SIAM Conference on Data Mining (SIAM-SDM)*, 2006.

[7] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

[8] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, 2003.

[9] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD*, 1998.

[10] W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems*, 18:288–321, 2000.

[11] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, 2005.

[12] L. Gravano, P. Ipeirotis, N. Koudas, and D. Srivastava. Text joins for data cleansing and integration in an rdbms. In *IEEE International Conference on Data Engineering (ICDE)*, 2003.

[13] M. Hernández and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, 1995.

[14] D. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM Conference on Data Mining (SIAM SDM)*, 2005.

[15] A. Monge and C. Elkan. The field matching problem: Algorithms and applications. In *SIGKDD*, 1996.

[16] J. Neville, M. Adler, and D. Jensen. Clustering relational data using attribute and link information. In *Text Mining and Link Analysis Workshop, IJCAI*, 2003.

[17] P. Singla and P. Domingos. Multi-relational record linkage. In *SIGKDD Workshop on Multi-Relational Data Mining (MRDM)*, 2004.

[18] S. Tejada, C. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information Systems Journal*, 26(8):635–656, 2001.

Generic Entity Resolution in the SERF Project

Omar Benjelloun   Hector Garcia-Molina   Hideki Kawai   Tait Eliott Larson
David Menestrina   Qi Su   Sutthipong Thavisomboon   Jennifer Widom

Stanford University

Abstract

The SERF project at Stanford deals with the Entity Resolution (ER) problem, in which records determined to represent the same real-life "entities" (such as people or products) are successively located and combined. The approach we pursue is "generic", in the sense that the specific functions used to match and merge records are viewed as black boxes, which permits efficient, expressive and extensible ER solutions. This paper motivates and introduces the principles of generic ER, and gives an overview of the research directions we have been exploring in the SERF project over the past two years.

1 Introduction

Entity Resolution (ER) (also referred to as deduplication) 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, consider a comparative shopping website, aggregating product catalogs from multiple merchants. Identifying records that *match*, i.e., records that represent the same product is challenging because there are no unique identifiers across merchant catalogs. A given product may appear in different ways in each catalog, 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 also *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.

In the SERF project, we study ER as a "generic database problem". We say we take a generic approach because we do not study the internal details of the functions used to compare and merge records. Rather, we view these functions as "black-boxes" to be invoked by the ER engine. Given such black-boxes, we study algorithms for efficiently performing ER, i.e., we develop strategies that minimize the number of invocations to these potentially expensive black-boxes. An important component of our work is that we identify a set of properties that the black-boxes should have in order to lead to a well-defined single "answer" to the ER problem, as well as to efficient algorithms. For example, associative merges is one such important property: If merges are

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

not associative, the order in which records are merged may impact the final result. Our general framework for ER is introduced in Section 2, as well as our main algorithm.

Except for a few works such as [HS95], most existing work on ER focuses on developing techniques to achieve the best quality for ER, measured in terms of precision and recall, on some class of data or applications. In our generic approach, such metrics are dependent on the black-box functions, and our focus is rather on the framework and algorithms in which these black-boxes are used. Of course, to define a simple coherent setting for ER, we have to make some assumptions (e.g., that matches are computed pairwise), hence we only capture a subset of the (diverse) set of existing techniques for ER, a subset which we believe is useful for a large number of applications. We refer the reader to [BGMJ$^+$05] for a detailed review of related works.

Since ER is computationally expensive, we also developed strategies to distribute its computation across multiple processors. Again, we made our distributed ER algorithm generic, by providing simple abstractions that make the distribution strategy configurable, i.e., capable to accommodate the characteristics of data in specific applications. In particular, our abstractions provide a single unified way to express and leverage common forms of domain knowledge in order to "block" unnecessary record comparisons. For instance, if it is known that records representing products can only match if their prices are close enough, records can be split among processors based on their price, in a way that greatly reduces the communication costs, while still computing the correct result. Our distributed ER algorithm is presented in Section 3.

Because ER is an approximate process, it is often desirable to attach confidence values to the records, and propagate these confidences as matches and merges are performed. However, the meaning of confidences and the way they are propagated may vary. In some applications, the confidence of a record could be interpreted as the probability that it correctly represents an entity, while in others, confidences may measure the precision of data. We extend our model with confidences in a generic way, by leaving to the match and merge functions the responsibility to interpret and propagate confidences. As we illustrate in Section 4, adding confidences implies that some of the properties previously identified for the black-box functions do not hold anymore. For instance, the associativity of merges is often not satisfied, because the order in which records are matched and merged may quite naturally affect the confidence of a derived record. As a consequence, more expensive algorithms are needed for ER with confidences. However, some optimizations may reduce the cost of the ER computation, e.g., when the properties still apply for the data component of the records, or if only records with confidence above some threshold are of interest.

We conclude this paper with a discussion of current and future research directions in Section 5.

2 Generic Entity Resolution

We start by defining our generic model for ER. The input of ER is a set of *records*, and so is its output. We do not make any assumption about the particular form or data model used to represent records.

As an example, in Figure 1 we consider records representing products, along the lines of the comparison shopping scenario mentioned in the introduction. Each product has a name, a price (or price range), and a category. This example is inspired from the data used in our experiments, which consists of actual product descriptions provided to us by the Yahoo! Shopping team.

	name	price	category
r_1	Apple iPod	249	MP3 player
r_2	Apple iPod	299	
r_3	iPod	270	MP3 player

Figure 1: Product records

2.1 Match and Merge

Our generic ER model is based on two black-box functions provided as input to the ER computation: match and merge.

A *match function* M is a function that takes two records as input and returns a Boolean value. Function M returns true if the input records represent the same entity, and false otherwise. Such a match function reflects the restrictions we are making that (i) matching decisions can be made "locally", based on the two records being compared, and (ii) that such decisions are Boolean, and not associated with any kind of numeric confidence (we will revisit this restriction in Section 4). In practice, such functions are easier to write than functions that consider multiple records.

A *merge function* μ is a function that takes in two records and returns a single record. Function μ is only defined for pairs of matching records, i.e., records known to represent the same entity. Its output is a "consolidated" record representing that entity. If a record r is (transitively) obtained through a merge involving a record r', we say that r is *derived* from r'.

When M and μ are understood from the context, $M(r_1, r_2) = \texttt{true}$ (resp. $M(r_1, r_2) = \texttt{false}$) is denoted by $r_1 \approx r_2$ (resp. $r_1 \not\approx r_2$), and $\mu(r_1, r_2)$ is denoted by $\langle r_1, r_2 \rangle$.

To illustrate, we define sample match and merge functions for our comparison shopping example. The match function is based on product names being equal, or all attributes of the records being highly similar. Such a match function can be expressed as:

$$M(r1, r2) = (r_1.name == r_2.name) \vee (M_{name}(r_1, r_2) \wedge M_{price}(r_1, r_2) \wedge M_{category}(r_1, r_2))$$

Let us say that M_{name} computes some similarity n of record names (e.g., an edit-distance) and returns true if $n > 0.8$. M_{price} computes the relative distance p among prices, and returns true if $p > 0.9$. $M_{category}$ returns true if the two records have the exact same category.

With this match function, in Figure 1 $r_1 \approx r_2$, because they have the exact same name. However, $r_1 \not\approx r_3$ because their prices are too far apart, and $r_2 \not\approx r_3$ because r_2 does not have a category value.

For the merge function, let us assume that it has some way to normalize product names into a single name (e.g., by relying on an external source to find the closest reference product name), that it keeps a range for prices, and keeps the union of category values from the base records. This merge function would produce $r_4 = \langle r_1, r_2 \rangle$:

$r4$ = (Apple iPod, [249-299], MP3 Player)

Observe that, unlike r_1 and r_2, the obtained record r_4 may match r_3 because it has combined price and category information from r_1 and r_2. This example illustrates one of the difficulties of ER: it is not sufficient to compare base records to each other. Derived records must be recursively compared to the other records in the dataset.

2.2 Domination

A last notion we need to introduce before defining generic ER is domination. Intuitively, if two records r_1 and r_2 are about the same entity but r_1 holds more information than r_2, then r_2 is useless for representing this entity. We say that r_1 *dominates* r_2. In general, any partial order on records could be used to define domination. For each application, a different notion of domination may be suitable.

To capture the fact that domination is application specific, we rely on the match and the merge functions to define it: We say that r_1 dominates r_2 if $r_1 \approx r_2$ (i.e., the two records match), and $\langle r_1, r_2 \rangle = r_1$. The consistency conditions we will introduce shortly in Section 2.4 ensure that domination is a partial order on records. In our example, the reader can verify that r_4 dominates r_1 and r_2.

2.3 Generic ER

We are now equipped to define entity resolution: Given a set of input records R, an *Entity Resolution* of R, denoted $ER(R)$ is a set of records such that:

- Any record in $ER(R)$ is derived (through merges) from records in R;
- Any record that can be derived from R is either in $ER(R)$, or is dominated by a record in $ER(R)$;
- No two records in $ER(R)$ match, and no record in $ER(R)$ is dominated by any other.

2.4 Consistent ER

If match and merge are arbitrary functions, entity resolution of a set of records R may not exist, may not be unique, and may be infinite. In [BGMJ$^+$05], we introduce simple and practical conditions on the match and merge functions, which guarantee that ER is "consistent", i.e., that it exists, is unique and finite. These properties are the following:

- *Commutativity:* For any pair of records r_1, r_2, $r_1 \approx r_2$ is the same as $r_2 \approx r_1$, and if r_1 and r_2 match, then $\langle r_1, r_2 \rangle$ and $\langle r_2, r_1 \rangle$ produce the same record.
- *Reflexivity/Idempotence:* Any record r matches itself, and $\langle r, r \rangle = r$
- *Representativity:* If $r_3 = \langle r_1, r_2 \rangle$, then r_3 "represents" r_1 and r_2, in the sense that r_3 matches any record that matches r_1 or r_2.
- *Merge associativity:* For any records r_1, r_2, r_3, if $\langle r_1, \langle r_2, r_3 \rangle \rangle$ and $\langle \langle r_1, r_2 \rangle, r_3 \rangle$ exist, then they are equal. Intuitively, this property means that if there exists multiple derivations involving the same set of records, then they should all produce the same result.

As discussed in [BGMJ$^+$05], if some of these properties do not hold, the entity resolution problem becomes much more expensive. For instance, without merge associativity we must consider all possible orders in which records may match and merge. Extending ER with confidences (see Section 4) leads to such a situation.

2.5 The R-Swoosh algorithm

When the four properties introduced above are satisfied, we develop an efficient ER algorithm, R-Swoosh, that reduces the number of invocations to the match and merge functions.

R-Swoosh relies on two sets: R, which initially contains all the input records, and R', which maintains the set of (so far) non-dominated, non-matching records. R-Swoosh successively compares each record in R to all the records present in R'. R-Swoosh performs a merge as soon as a pair of matching records is found. The obtained record is added to R, and the pair of matching records is deleted immediately. The algorithm terminates when R is empty, and R' contains $ER(R)$.

To illustrate, consider the run of R-Swoosh given in Figure 2. The algorithm starts with all the input records in R, and an empty R'. At every round, one record from R is compared to the records in R' and moved to R' if no match is found. Here, r_1 and r_2 are successively moved to R'. At round i, r_3 is compared to r_1 and a match is found. The two records are immediately merged into r_7, which is put back into R, while r_1 and r_3 are deleted. The algorithm ends when R is empty. R' contains the result of ER.

Intuitively, R-Swoosh is efficient because it interleaves matches, merges and deletions of dominated records. R-Swoosh may end up comparing all pairs of records to each other, but it eagerly performs merges and deletions as early as possible, thereby avoiding unnecessary future match comparisons. In [BGMJ$^+$05], it is shown that

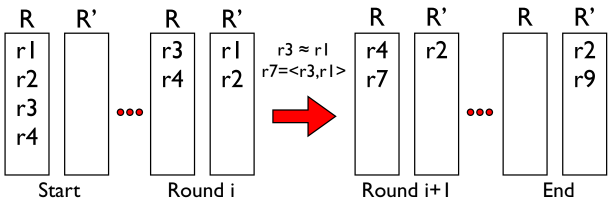


Figure 2: A sample run of the R-Swoosh algorithm

R-Swoosh is optimal, in the sense that no other algorithm can perform fewer record comparisons in the worst case.

In [BGMJ+05], we also give a variant of the algorithm called F-Swoosh (F stands for "feature") that efficiently caches the results of value comparisons when the match function can be expressed as a disjunction of match functions over "features", i.e. parts of the records.

3 Distributed ER

Even though R-Swoosh is optimal, ER is still an expensive process, as it may need to perform (expensive) match comparisons on each pair of records in the (often large) input dataset. To deal with this complexity, we investigated in [BGMK+06].ways to parallelize the ER computation across multiple processors.

Distributing data to processors for the ER computation requires care, as any pair of records is a potential match, and therefore needs to be compared. Also, recall that records produced through merges need to be compared with others, and must be distributed adequately.

In general, there is no optimal, application-independent strategy to distribute data to processors. Depending on the application and the distribution of values, some strategy may be more sensible than others. In particular, it is important to exploit any domain knowledge that saves some comparisons, by reflecting it in the distribution strategy.

As an example, in our comparison shopping application, we may have the domain knowledge that prices for the same product never vary by more than, say, 20% from one vendor to another. We can exploit this knowledge by making each processor responsible for one price segment, with a 20% overlap to account for prices close to segment boundaries. However, prices may not be uniformly distributed, in which case we should distribute the workload of crowded segments across multiple processors.

To support such distribution needs in a generic way, we introduce two abstract functions:

- *scope:* captures the distribution of records to processors, by assigning to each record a set of processors,
- *resp:* determines which processors are responsible of comparing which pairs of records.

We use these functions as primitives in our distributed ER algorithm. To guarantee the correctness of the algorithm, the scope and resp functions need only satisfy a simple *coverage* property: any pair of potentially matching records have scopes that intersect at least at one processor, which is responsible for comparing them.

The D-Swoosh algorithm runs a variant of R-Swoosh at each of the processors. Initially, records are distributed to processors based on the scope function. Then, each processor operates in a similar fashion to R-Swoosh, with the main difference that processors asynchronously exchange messages about which records are added or deleted (again, using the scope function), and keep track of all the records they know have been deleted. The algorithm terminates when all processors are idle and no more messages are exchanged. The result of ER is the union of the R' sets at each processor.

Figure 3 illustrates part of the computation at one of the processors. P_i discovers that records r_3 and r_1 match, and merges them into r_7. It sends an $add(r_7)$ message to P_{27} and P_{67}, the processors in $scope(r_7)$, and delete messages for r_1 and r_3 to the processors in their respective scopes (including itself). Upon receiving these add and delete messages, the processors update their local R and R' sets accordingly. The full D-Swoosh algorithm is described in [BGMK+06].

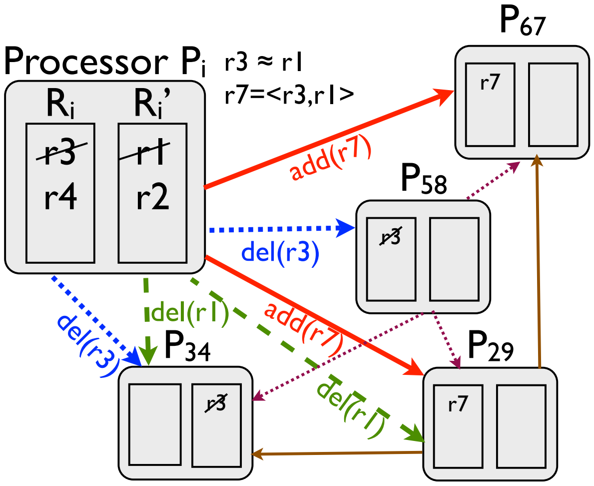


Figure 3: The D-Swoosh algorithm

In [BGMK+06], we also provide a "library" of scope and resp functions, both for the case when no domain knowledge is available, and all pairs of records must be compared, and for the case where domain knowledge of some common forms is available. In the latter case, we are able to express and leverage necessary conditions on pairs of matching records such as the equality on the value of some attribute (such as the category in our example), a linearly ordered attribute with a sliding window (such as our price example), or an ancestor/descendant relationship in a hierarchy. We experimentally compared the different schemes (both with and without domain knowledge) on a comparison shopping dataset from Yahoo!.

4 ER with confidences

In the model we presented so far, everything is certain: records are exact facts, and the match function makes Boolean decision on whether pairs of records represent the same entity. The properties of the match and merge function essentially guarantee that there exists a unique ER solution. However, manipulating numerical confidences (or uncertainties) is often desirable in ER. For instance, input records may come from unreliable sources, and may have confidences associated with them. The match comparisons between records may also yield confidences that represent how likely the records are to represent the same real-world entity. Similarly, the merge process may introduce additional uncertainties, as there may not be a deterministic way to combine the information from different records. In each application domain, the interpretation of confidence numbers may be different. For instance, a confidence number may represent a "belief" that a record faithfully reflects data from a real-world entity, or it may represent how "accurate" a record is.

In [MBGM05], we extended our framework for ER with confidences in a generic way by simply associating a confidence value *conf* (between 0 and 1) to each record. The match and merge functions are responsible for manipulating and propagating confidence values. For instance, our product records may have initial confidences (e.g., reflecting the reliability of their sources), as shown in Figure 4.

	name	price	category	*conf*
r_1	Apple iPod	249	MP3 player	0.8
r_2	Apple iPod	299		0.7
r_3	iPod	270	MP3 player	0.5

Figure 4: Product records with confidences

The merge of r_1 and r_2 may now produce r_4, with a confidence value assigned by the merge function:

$r4$ = (Apple iPod, [249-299], MP3 Player) *conf:* 0.56

Here, the merge function multiplied the confidences of r_1 and r_2 to generate the confidence of r_4. This choice intuitively corresponds to making an independence assumption in a probabilistic interpretation of confidences. Other choices may also be reasonable, such as taking the minimum of the confidences of r_1 and r_2 as a confidence for r_4.

When confidences are present, the notion of domination must be extended. We say that r_1 dominates r_2 (with confidences) if the data component of r_1 dominates that of r_2 and the confidence of r_1 is higher than or equal to that of r_2. In our example, observe that r_4 does not dominate r_1 nor r_2, because it has a lower confidence.

Adding confidences significantly affects the ER process, because some of the properties introduced in Section 2.4 are not satisfied anymore:

- *No representativity:* The fact that two records r_1, r_2 match is inherently an uncertain operation, and therefore the record r_{12} produced by their merge is likely to have a lower confidence than both r_1 and r_2. Even if the data component of r_{12} "represents" that of r_1 and r_2, r_{12} may not match a record that r_1 or r_2 matches, because its has a lower confidence, an information that may be used by the match function. Therefore, the representativity property may not hold.

- *No Merge associativity:* The confidence of a derived record may depend on the sequence of merges that produced it. Given three records r_1, r_2, r_3, $\langle r_1, \langle r_2, r_3 \rangle \rangle$ and $\langle \langle r_1, r_2 \rangle, r_3 \rangle$ may very well have different confidence values, e.g., because one of the derivation was based on "strong" match evidence (therefore yielding high confidence) while the other derivation follows a tenuous connection.

Because representativity and merge associativity may not hold, ER must be performed using a more expensive algorithm than R-Swoosh. Essentially, records participating in a merge cannot be deleted right away, and dominated records can only be removed after all possible matches and merges have been found. In [MBGM05], we provide *Koosh*, a variant of R-Swoosh which is the optimal sequential algorithm for generic ER when representativity and merge associativity do not hold. In a nutshell, Koosh also uses two sets R and R', but always compares records in R to *all* the records in R', and postpones the deletion of dominated records until R is empty.

For the important case where the properties of Section 2.4 do hold for the data component and are only violated because of confidences, we proposed a two-phase algorithm which performs much better than Koosh. The algorithm exploits the fact that matching on the data component is a necessary condition for matching in ER with confidences. It runs a first pass of R-Swoosh on the data component only to partition the base records into "packages", and then runs the more expensive Koosh algorithm on each package separately.

Another optimization we investigated was the use of thresholds to prune the search space: If the user is only interested in records with confidence above some fixed value, and if the meaning of confidences is such that they may only decrease upon merging records, then any record can be discarded as soon as its confidence falls below the threshold, because it cannot contribute to the derivation of any above-threshold record.

5 Conclusion

Entity resolution is a crucial information integration problem. We believe our generic approach provides a clean separation between the quality aspects of ER, encapsulated in the black-box match and merge functions, and its algorithmic and performance aspects, addressed by the sequential and distributed algorithms we developed. To conclude, we would like to mention some of the research directions we are currently investigating in the SERF project:

- *Large scale distributed ER:* We are in the process of deploying our D-Swoosh algorithm on a large-scale, shared-nothing distributed infrastructure consisting of tens of commodity PCs, to run ER on the full Yahoo! comparison shopping dataset (several Gigabytes). We hope to understand the performance and cost trade-offs involved in large scale distributed ER, and to develop optimization strategies to best adapt the distribution scheme (i.e., the scope and resp functions) to particular applications and datasets..
- *Negative ER and uncertainty:* Going beyond our essentially monotonic model for ER, we are incorporating negative information, to express constraints needed by a number of applications. Negative facts essentially lead to modeling uncertainty in the ER process, and embracing the fact that ER may have multiple alternative solutions, possibly with a probability distribution over them. We are building upon the ULDB model for databases with uncertainty and lineage [BSHW06]. Lineage, which keeps track of the derivation history of records is crucial to back-track previously made merge decisions, should new evidence suggest to do so. We are investigating efficient algorithms to compute the most probable ER answer in the presence of such negative information.
- *I/O's and buffer management for ER:* We are investigating strategies to efficiently perform ER when the dataset does not fit in main memory. We are developing and experimenting with various buffer management strategies, and corresponding adaptations of our ER algorithms.
- *Declarative ER:* We believe ER should be specified declaratively using match rules that combine atomic similarity functions on attribute values, and high level constraints able to capture applicable domain knowledge. Based on these specifications, which could be either entered by experts or learned from a training sample, we would like to derive an efficient "execution plan" for performing ER, possibly taking into account statistics on the atomic match and merge black-box "operators" through a suitable cost model.

References

[BGMJ+05] O. Benjelloun, H. Garcia-Molina, J. Jonas, Q. Su, and J. Widom. Swoosh: A Generic Approach to Entity Resolution. Technical report, Stanford University, 2005. (available at `http://dbpubs.stanford.edu/pub/2005-5`).

[BGMK+06] O. Benjelloun, H. Garcia-Molina, H. Kawai, T. E. Larson, D. Menestrina, and S. Thavisomboon. D-Swoosh: A Family of Algorithms for Generic, Distributed Entity Resolution. Technical report, Stanford University, 2006. (available at `http://dbpubs.stanford.edu/pub/2006-8`).

[BSHW06] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with Uncertainty and Lineage. In *Proc. of VLDB (to appear)*, 2006.

[HS95] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *Proc. of ACM SIGMOD*, pages 127–138, 1995.

[MBGM05] D. Menestrina, O. Benjelloun, and H. Garcia-Molina. Generic Entity Resolution with Data Confidences. Technical report, Stanford University, 2005. (available at `http://dbpubs.stanford.edu/pub/2005-35`).

Data Fusion in Three Steps: Resolving Schema, Tuple, and Value Inconsistencies

Felix Naumann[1] Alexander Bilke[2] Jens Bleiholder[1] Melanie Weis[1]
[1] Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
{naumann|bleiho|mweis}@informatik.hu-berlin.de

[2] Technische Universität Berlin
Strasse des 17. Juni 135, 10623 Berlin, Germany
bilke@cs.tu-berlin.de

Abstract

Heterogeneous and dirty data is abundant. It is stored under different, often opaque schemata, it represents identical real-world objects multiple times, causing duplicates, and it has missing values and conflicting values. Without suitable techniques for integrating and fusing such data, the data quality of an integrated system remains low. We present a suite of methods, combined in a single tool, that allows ad-hoc, declarative fusion of such data by employing schema matching, duplicate detection and data fusion.

Guided by a SQL-like query against one or more tables, we proceed in three fully automated steps: First, instance-based schema matching bridges schematic heterogeneity of the tables by aligning corresponding attributes. Next, duplicate detection techniques find multiple representations of identical real-world objects. Finally, data fusion and conflict resolution merges each duplicate into a single, consistent, and clean representation.

1 Fusing Heterogeneous, Duplicate, and Conflicting Data

The task of integrating and fusing data involves the solution of many different problems, each one in itself formidable: Apart from the technical challenges of accessing remote data, heterogeneous schemata of different data sets must be aligned, multiple but differing representations of identical real-world objects (duplicates) must be discovered, and finally the duplicates must be fused to present a clean and consistent result to a user. In particular this final step is seldomly or inadequately addressed in the literature. Figure 1 shows the three steps and the inconsistencies they bridge.

Each of these tasks has been addressed in research individually at least to some extent: (i) Access to remote sources is now state of the art of most integrated information systems, using techniques such as JDBC, wrappers, Web Services etc. Such technical heterogeneities are not addressed in this article and we assume JDBC or file-based access to the relational data sources. (ii) Schematic heterogeneity has been a research issue for at least

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

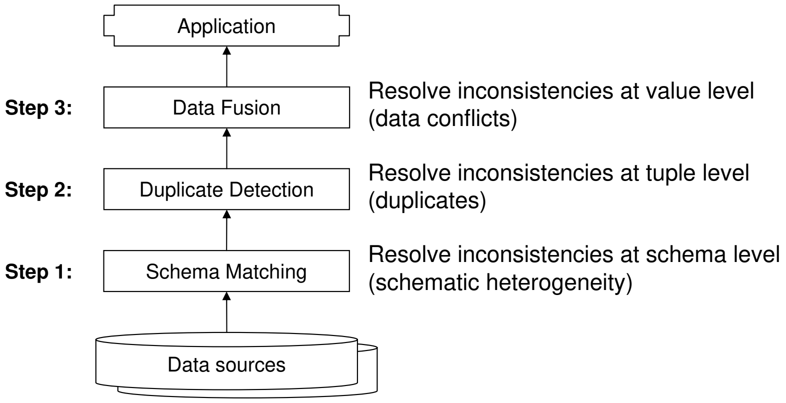


Figure 1: The three steps of data fusion

two decades, first in the context of schema integration and then to automatically generate schema mappings. Especially recently, schema matching techniques have made great progress in automatically detecting correspondences among elements of different schemata. (iii) Duplicate detection is successful in certain domains, in particular in customer relationship management where duplicate customers and their contact information must be detected, and several research projects have presented well-suited domain-independent algorithms. Other research directions have developed domain-independent approaches. All are usually performed as an individual task, such as a separate cleansing step in an ETL procedure. Here we bed duplicate detection into a domain-independent ad-hoc querying environment. (iv) Data fusion, i.e., the step of actually merging multiple, duplicate tuples into a single representation of a real world object, has only marginally been dealt with in research and hardly at all in commercial products. The particular problem lies in resolving value-level contradictions among the different representations of a single real-world object.

We have combined all these techniques under the umbrella of the Humboldt Merger (HumMer) – a one-stop solution for fusing data from heterogeneous sources [2]. A unique feature of HumMer is that all steps are performed in an ad-hoc fashion at run-time, initiated by a user query to the sources; in a sense, we perform *ad hoc, automatic, and virtual ETL*. Apart from the known advantages of virtual data integration (up-to-dateness, low storage requirement), this on-demand approach allows for maximum flexibility: New sources can be queried immediately, albeit at the price of not generating as perfect query results as if the integration process were defined by hand. To compensate, HumMer optionally visualizes each intermediate step of data fusion and allows users to interfere: The result of schema matching can be adjusted, tuples discovered as being border-line duplicates can be separated and vice versa, and finally, resolved data conflicts can be undone and resolved manually.

Ad-hoc and automatic data fusion is useful in many scenarios: Catalog integration is a typical one-time problem for companies that have merged, but it is also of interest for shopping agents collecting data about identical products offered at different sites. A customer shopping for CDs might want to supply only the different sites to search on. The entire integration process, from finding corresponding metadata, to detecting entries for identical CDs, and finally to fuse all conflicting data, possibly favoring the data of the cheapest store, is performed under the covers. In such a scenario, a schema matching component is of special importance, as many web sites use different labels for data fields or even no labels at all.

Another application made possible only by automatic data fusion systems like HumMer is the provision of online data cleansing services. Users of such a service simply submit sets of heterogeneous and dirty data and receive a consistent and clean data set in response. Such a service is useful for individuals trying to compare different data sets, but also for organizations not wanting to employ complex ETL procedures for all data sets.

Finally, an important application is disaster data management. In an area affected by a disaster, data about damages, missing persons, hospital treatments etc. is often collected multiple times (causing duplicates) at different levels of detail (causing schematic heterogeneity) and with different levels of accuracy (causing data conflicts). Fusing such data with the help of a graphical user interface can help speed up the recovery process and for instance expedite insurance pay-outs or detect insurance fraud.

2 Components for Data Fusion

HumMer combines several larger research projects under one umbrella. In the following sections we describe each project in some detail. Related work is references intermittently, but we point out that this article can be no means be a complete survey for each of the vast fields of schema matching, duplicate detection, and data fusion.

2.1 Schema Matching and Data Transformation

When integrating autonomous data sources, we must assume that they do not conform to the same schema. Thus, the first phase in the integration process is the resolution of schematic heterogeneity. This phase proceeds in two sub-steps: schema matching, i.e., the identification of semantically equivalent schema elements, and data transformation, i.e., the bringing the data under a single common schema.

Schema matching is the (semi-automatic) process of detecting attribute correspondences between two heterogeneous schemata. Various approaches that exploit different kinds of information [19], i.e., schema information [15], instances [17], or additional metadata [14], have been proposed. As we assume the databases to contain duplicates according to our scenarios, we apply the DUMAS schema matching algorithm [3]: First, the DUMAS efficiently detects a few duplicates in two (or more) unaligned databases and then derives a schema matching based on similar attribute values of duplicates. This key idea is shown in Figure 2 where two detected duplicate tuples from different sources are used to find a schema matching.

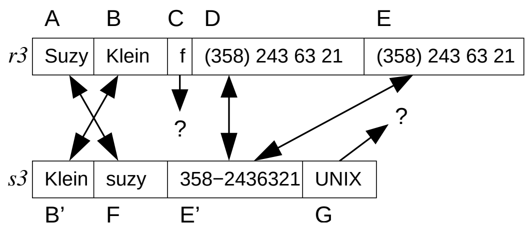


Figure 2: Schema matching using duplicates

Duplicate detection in unaligned databases is more difficult than in the usual setting, because attribute correspondences are missing, i.e., it is not known which attribute values to compare. However, the goal of this phase is not to detect all duplicates, but only as many as required for schema matching. Detecting *all* duplicates is left to the next HumMer component. DUMAS considers a tuple as a single string and applies a string similarity measure to extract the most similar tuple pairs. From the information retrieval field we adopt the well-known *TFIDF similarity* for comparing records. Experimental evaluation shows that the most similar tuples are in fact true duplicates.

These duplicates can be used for schema matching. If two duplicate tuples have the same or a sufficiently similar attribute value, we assume that these attributes correspond. Because two non-corresponding attributes might have a similar value by chance, we use several duplicates instead of only one. Two duplicates are compared field-wise using the *SoftTFIDF similarity measure* [6], resulting in a matrix containing similarity scores for each attribute combination. The matrices of each duplicate are averaged, and the maximum weight matching is computed, resulting in a set of 1:1 correspondences. Correspondences with a similarity score below a given threshold are pruned. HumMer allows users to manually add missing or delete false correspondences simultaneously across multiple data sources.

Like most related approaches, the matcher currently incorporated into HumMer is restricted to 1:1 correspondences. However, we have also developed a matcher that is able to detect complex 1:n or m:n correspondences based on the detected duplicates. The underlying idea is to combine source or target attributes and merge their respective matrix rows or columns in the matching step. The algorithm searches through the space of similarity matrices using greedy pruning, i.e., if a merge does not improve the overall matching, that branch of the search tree is not further considered.

In addition to complex matchings, we have also devised a duplicate-based matcher for schemata consisting of multiple tables. The algorithm starts with a few correspondences and crawls though the schemata by joining neighboring tables. In each step, additional correspondences are detected using the DUMAS matcher, which are used in the following steps to add more tables.

Thus with the schema matching step, schematic inconsistencies are detected and "marked" with appropriate correspondences. In the next sub-step the inconsistencies are overcome by transforming the data so that it appears under a single common schema.

The following *transformation* phase is straightforward because we assume only union-type integration: Without loss of generality, we assume that one schema is the preferred schema, which determines the names of attributes that semantically appear in multiple sources. The attributes in the non-preferred schema that participate in a correspondence are renamed accordingly. All tables receive an additional *sourceID* attribute, which is required in later stages. Finally, the full outer union of all tables is computed.

If correspondences cross multiple relations of source or target schema joins are necessary and a Clio-style data transformation becomes necessary. In this paper we assume that integration is to be performed over relations talking about same types of objects. Only then does duplicate detection and conflict resolution as described in the next sections make sense. Any more complex transformations should be performed in beforehand.

2.2 Duplicate Detection

Duplicate detection is a research area with a long tradition. Beginning with early work on record linkage [10], among many others a prominent technique for domain-dependent duplicate detection is the sorted neighborhood method [13]. More recently, several approached have emerged that regard not only data in a single table, but also data in related tables (or XML elements) to improve accuracy [1, 9, 21].

In [24] we introduce an algorithm that detects duplicates in XML documents. More precisely, duplicate XML elements are detected by considering not only their text nodes, but also those of selected children, i.e., elements involved in a 1:N relationship with the currently considered element. We map this method to the relational world (similar to [1]) to detect duplicates in a table using not only its attribute values, but also "interesting" attributes called *descriptions*, from relations that have some relationship to the current table. In this section, we first describe how descriptions are selected. Then, we introduce the duplicate detection procedure that compares tuples based on their descriptions.

2.2.1 Description Selection

Generally, we consider attributes interesting for duplicate detection being attributes that are (i) related to the currently considered object, (ii) useable by our similarity measure, and (iii) likely to distinguish duplicates from non-duplicates. We developed several heuristics to select such attributes in [25], based on descendant depth, data type, content model, optionality of elements, etc. In the relational data integration scenario descriptions are determined as follows: The attributes related to the currently considered object are attributes of the integrated table and attributes of "children tables". We consider as children all tables that contain foreign keys referencing the tables matched in the previous step. For efficiency, we limit the children tables to direct children only, i.e., no descendants reached by following more than one reference are considered.

The default description selection proposed by HumMer is the set of corresponding attributes between the two schemas, i.e., those that include values for tuples from both sources, opposed to values of non-matching attributes, which are padded with NULL values. Indeed, NULL values do not distinguish duplicates from non duplicates and thus such attributes should not be included in the description.

The heuristic of selecting only matched attributes as descriptions applies both to the already matched tables and attributes from their children tables. Therefore, the children tables need to be matched and integrated as well. Currently, children tables are matched pairwisely and the transitive closure over these pairwise matches is the final result. More specifically, let tables T_1 and T_2 be the two matched tables, and let $\{T_{1,1}, \ldots, T_{1,k}\}$ and $\{T_{2,1}, \ldots, T_{2,m}\}$ be their respective children tables. Then, every pair of tables $(T_{1,i}, T_{2,j}), 1 \leq i \leq k, i \leq j \leq m$ is matched. A given threshold determines if a pair of children tables correspond at all as shown in Figure 3. If they do, their data in the matched attributes can be used for duplicate detection as well.

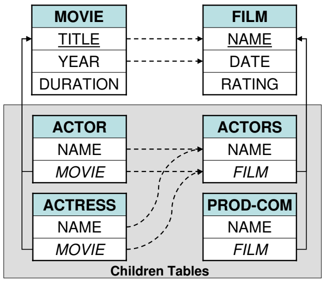


Figure 3: Matching children tables to improve duplicate detection

Once the descriptions is determined automatically, HumMer provides a comfortable means to modify the selection of interesting attributes proposed by our heuristics.

2.2.2 Duplicate Detection

After finalizing the selection of descriptions of an object, tuples are compared pairwisely using a thresholded similarity approach. More specifically, using a similarity measure $sim(t_1, t_2)$ that computes the similarity between two tuples t_1 and t_2, we classify a tuple pair as sure duplicate, possible duplicate, or non-duplicate, using two thresholds θ_{sure} and θ_{poss} in the following duplicate classifier.

$$\Gamma(t_1, t_2) = \begin{cases} t_1 \text{ and } t_2 \text{ sure duplicates} & \text{if } sim(t_1, t_2) > \theta_{sure} \\ t_1 \text{ and } t_2 \text{ possible duplicates} & \text{if } \theta_{poss} \leq sim(t_1, t_2) \leq \theta_{sure} \\ t_1 \text{ and } t_2 \text{ non-duplicates} & \text{otherwise} \end{cases}$$

The currently used similarity measure $sim()$ is proposed in [25] and takes into account (i) matched vs. unmatched attributes, (ii) data similarity between matched attributes using edit distance and numerical distance functions, (iii) the identifying power of a data item, measured by a soft version of IDF, and (iv) matched but contradictory vs. non-specified (missing) data; contradictory data reduces similarity whereas missing data has no influence on similarity. The number of pairwise comparisons are reduced by applying a filter and comparing only the remaining pairs. The filter used in combination with our similarity measure is the filter proposed in [25] that is defined as an upper bound to the similarity measure. Hence, if $sim(t_1, t_2) \leq filter(t_1, t_2) \leq \theta_{poss}$, then we can classify pair (t_1, t_2) as non-duplicate without computing the actual similarity measure, which is more complex to compute than the filter.

Pairs classified as possible duplicates are presented to the user in descending order of similarity. The user can then manually classify the pair as sure duplicate or non-duplicate. Using the descending order of similarity, users can often conclude that after classifying several pairs as non-duplicates, the remaining pairs, which are less similar, are also non duplicates.

When all pairs of sure duplicates are finally available, the transitive closure over duplicate pairs is formed to obtain clusters of objects that all represent a single real-world entity. The output of duplicate detection is the same as the input relation, but enriched by an *objectID* column for identification. Thus, inconsistencies at tuple-level are resolved: The identity of each object and its multiple representations is know. Conflicts among duplicates are resolved during conflict resolution.

2.3 Conflict Resolution

The last step in a data integration process, after *schema matching* and *duplicate detection* has been done, is to combine the different representations of a single real world object into one single representation. This step is referred to as *data fusion* and aims at resolving the still existing conflicts (uncertainties and contradictions) in the attribute values. First we show a query language to specify for each attribute a functions to resolve conflicts. Thereafter we present initial ideas to optimize queries involving data fusion.

2.3.1 Specifying data fusion

We consider *data fusion* as a step in the integration process that is guided by an (expert) user. The user specifies how the different representations and their values are used in determining the final representation, whereas a specific information system, like our HumMer system, carries out the fusion itself. In fusing data from different sources, a user can follow one of several different strategies that are repeatedly mentioned in literature [11, 16, 18, 20, 22] and categorized in [5]. Example strategies are:

- CONSIDER ALL POSSIBILITIES: Conflicts are *ignored* and all possible combinations of values (occasionally creating ones that have not been present in the sources before) are passed on to the user, who finally decides about which "possible world" to choose.
- TRUST YOUR FRIENDS: Specific conflicts are *avoided* by taking a preference decision beforehand and using only values from a specific source, leaving aside the (possibly conflicting) values from other sources.
- CRY WITH THE WOLVES: Choosing the value that is most often used, results in *resolving* a conflict by taking one of the existing values and following the idea that correct values prevail over incorrect ones.
- MEET IN THE MIDDLE: Another possible way of *resolving* the conflict is in creating a new value that is a compromise of all the conflicting values, e.g., an average over several numeric values.

Data fusion in the HumMer system is implemented as a relational operator. It takes as input a number of tables containing multiple representations of a real world object and gives as output one table with exactly one representation for each real world object. This is done by grouping and aggregation, hereby using a global key to group the representations. The key needs to be provided by duplicate detection techniques applied before on the data. In each group conflicts may arise in each column that is not used for grouping. These conflicts are resolved per column by applying a conflict resolution function to the data. Functions that can be used do not only include the standard SQL aggregation functions (min, max, sum, ...) but other more elaborate functions as well, for instance functions that not only use the conflicting values in determining a final value, but also other data from the same attribute, data from other attributes or metadata as given by the query context (e.g., statistics, meta data of sources, etc.). The HumMer system is extensible allowing user defined conflict resolution functions. In the following a brief list of some functions that could be used:

- MAX / MIN: Returns the maximum/minimum value of the conflicting data values.
- GROUP: Returns a set of all conflicting values and leaves resolution to the user.

- SHORTEST / LONGEST: Chooses the value of minimum/maximum length.
- VOTE: Returns the value that appears most often among the present values. Ties could be broken by a variety of strategies, e.g., choosing randomly.
- FIRST / LAST: Takes the first/last value of all values, even if it is a NULL value.
- COALESCE: Takes the first NON-NULL value appearing.
- CHOOSE(SOURCE): Returns the value supplied by the specific source.
- MOST RECENT: Recency is evaluated with the help of another attribute or other metadata.

The fusion operation is expressed with the help of the FUSE BY statement as described in [4]. Defaults, such as using **coalesce** as the default conflict resolution function or using the order of the tables given as preference judgement, as well as implicitly removing subsumed tuples, make it easy to specify conflict resolution in an SQL-like syntax. By issuing such a FUSE BY statement the user is able to accomplish many of the different possible fusion strategies. An example for a FUSE BY statement is:

```
SELECT    Name, RESOLVE(Age, max), RESOLVE(Address, choose(EE_Students))
FUSE FROM EE_Students, CS_Students
FUSE BY   (Name)
```

This statement fuses data two student tables, leaving just one tuple per student. Students are identified by their name, conflicts in the age of the students are resolved by taking the max (assuming that people only get older), and conflicts in the address are avoided by choosing the address from source EE_Students (implementing a TRUST YOUR FRIENDS strategy).

2.3.2 Optimizing data fusion

In an information integration scenario a FUSE BY statement could be seen as a mediator, composing different sources and shaping a consistent view on these sources. Querying such a view results in a query tree combining other relational operators and the fusion operator (e.g., Figure 4). The sources used in the view may themselves be complex queries involving other relational operators as well.

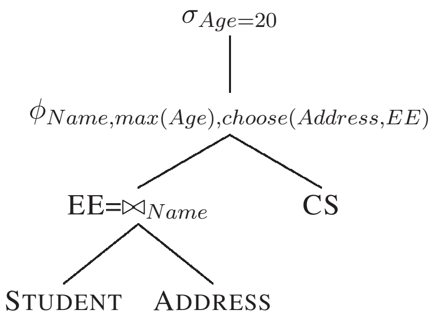


Figure 4: Query on two tables EE and CS involving Fusion and selection, assuming dirty tables CS and Student, and clean table Address (one address per name). The ϕ operator denotes fusion

For ad-hoc queries efficiency is important. To support algebraic query optimization we analyze different properties of the conflict resolution functions, e.g., commutativity, order dependance, decomposability, etc. These play an important role when deciding whether a fusion operator can be pushed down below a join, or a selection can be pushed down below a fusion, etc. Rules for decomposable, order- and duplicate insensitive functions, such as max and min, can be taken from the literature on optimization of grouping and aggregation ([12, 23]) and used in pushing down fusion beyond joins. Likewise, rules for selection pushdown below Group By for these kinds of functions also apply to fusion [7].

An example for such a transformation is in Figure 5, where early selection and fusion decreases the cardinality of intermediate results. We are currently investigating rules for the more complex functions (VOTE, CHOOSE, etc.), eventually making it necessary to use an extended relational algebra that is order-aware. The rules will be included into the query optimizer of the HumMer system. Choosing among different equivalent plans in a cost based fashion (physical query optimization) is currently only supported in the HumMer system by using a tuple-based cost model.

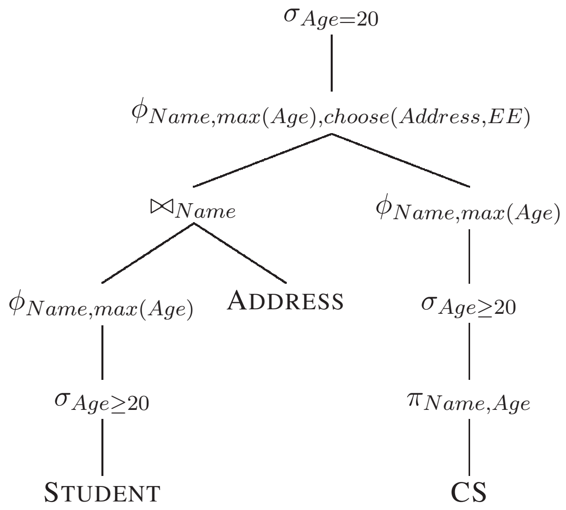


Figure 5: Query from Figure 4, optimized, decreasing the size of intermediate results by pushing selection and fusion down the tree. The ϕ operator denotes fusion

3 Composing the Individual Steps

The Humboldt Merger is implemented as a stand-alone Java application. The underlying engine of the entire process is the XXL framework, an extensible library for building database management systems [8]. This engine together with some specialized extensions handles tables and performs the necessary table fetches, joins, unions, and groupings. On top of the process lies a graphical user interface that drives the user experience. HumMer combines the techniques described in the previous section to achieve all phases of data fusion in a single system. A metadata repository stores all registered sources of data under an alias. Sources can include tables in a database, flat files, XML files, web services, etc. Since we assume relational data within the system, the metadata repository additionally stores instructions to transform data into its relational form. This section briefly describes the architecture and dataflow within the system, as shown in Fig. **??**.

HumMer works in two modes: First, querying via a basic SQL interface, which parses FUSE BY queries and returns the result. Second, querying via a wizard guiding users in a step by step fashion: Given a set of aliases as chosen by the user in a query, HumMer first generates the relational form of each and passes them to the schema matching component. There, columns with same semantics are identified and renamed accordingly, favoring the first source mentioned in the query. The result is visualized by aligning corresponding attributes on the screen. Users can correct or adjust the matching result. Data transformation adds an extra sourceID column to each table to store the alias of the data source and performs a full outer union on the set of tables.

The resulting table is input to duplicate detection. If source tables are part of a larger schema, this component consults the metadata repository to fetch additional tables and generate child data to support duplicate detection. First, the schema of the merged table, along with other tables that still might reside in the databases is visualized as a tree. Heuristics determine which attributes should be used for duplicate detection. Users can optionally adjust the results of the heuristics by hand within the schema. The duplicate detection component adds yet another column to the input table – an objectID column designating tuples that represent the same real-world object. The results of duplicate detection are visualized in three segments: Sure duplicates, sure non-duplicates, and unsure cases, all of which users can decide upon individually or in summary.

The final table is then input to the conflict resolution phase, where tuples with same objectID are fused into a single tuple and conflicts among them are resolved according to the query specification. At this point, the relational engine also applies other query predicates. The final result is passed to the user to browse or use for further processing. As an added feature, data values can be color-coded to highlight uncertainties and data conflicts. Also HumMer collects lineage information for each value, so that users can see the original conflicting values and their data source. Figure 6 shows a screen shot of this final view.

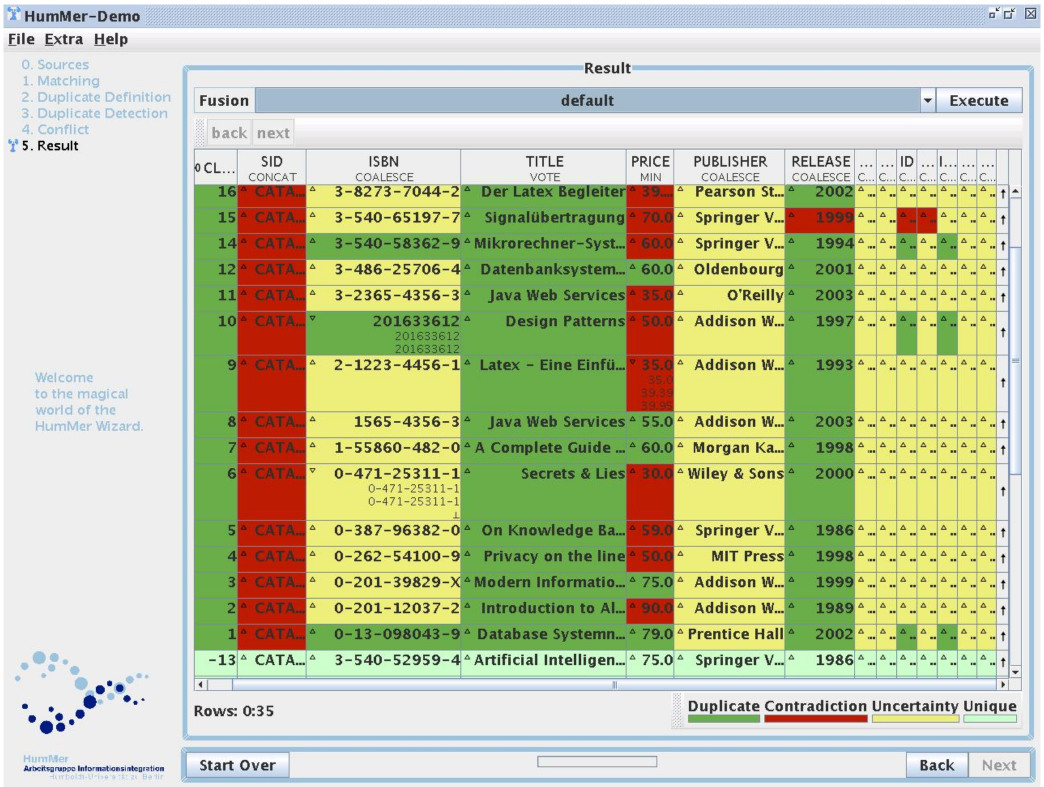


Figure 6: Screenshot of HumMer

4 Outlook

We conclude by reiterating the observation that surprising little work has been done in the field of data fusion to improve the quality of data. A survey of the field of duplicate detection, which would be an obvious place for authors to at least indicate what is to be done once duplicates are detected, yielded no satisfactory approaches. In fact, a common synonymous term for duplicate detection is "duplicate elimination", which precisely describes what many authors propose: Simply remove all but one representative of a duplicate group.

In the field of data integration there are indeed a few concrete approaches to data fusion as cited exemplarily in Section 2.3. Those results have not yet moved to commercial databases and applications yet, despite great efforts of vendors to extend databases to wrap heterogeneous data sources. Data fusion in real-world applications is mostly performed manually or is hard-coded into proprietary applications or ETL scripts. We believe that inclusion of data fusion capabilities into the DBMS kernel is promising.

Acknowledgments. Christoph Böhm and Karsten Draba were very helpful in implementing HumMer. This research was supported by the German Research Society (DFG grants no. NA 432 and GRK 316).

References

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Hong Kong, China, 2002.

[2] A. Bilke, J. Bleiholder, C. Bhm, K. Draba, F. Naumann, and M. Weis. Automatic data fusion with HumMer. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2005. Demonstration.

[3] A. Bilke and F. Naumann. Schema matching using duplicates. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 69–80, Tokyo, Japan, 2005.

[4] J. Bleiholder and F. Naumann. Declarative data fusion - syntax, semantics, and implementation. In *Advances in Databases and Information Systems (ADBIS)*, Tallin, Estonia, 2005.

[5] J. Bleiholder and F. Naumann. Conflict handling strategies in an integrated information system. In *Proceedings of the International Workshop on Information Integration on the Web (IIWeb)*, Edinburgh, UK, 2006.

[6] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb)*, pages 73–78, 2003.

[7] U. Dayal. Processing queries over generalization hierarchies in a multidatabase system. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 342–353, Florence, Italy, 1983. Morgan Kaufmann.

[8] J. V. den Bercken, B. Blohsfeld, J.-P. Dittrich, J. Krämer, T. Schäfer, M. Schneider, and B. Seeger. XXL - a library approach to supporting efficient implementations of advanced database queries. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 39–48, 2001.

[9] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 85–96, 2005.

[10] I. Fellegi and A. Sunter. A theory of record linkage. *Journal of the American Statistical Association*, 64(328), 1969.

[11] A. Fuxman, E. Fazli, and R. J. Miller. Conquer: efficient management of inconsistent databases. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 155–166, New York, NY, USA, 2005. ACM Press.

[12] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *VLDB*, pages 358–369, 1995.

[13] M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[14] W.-S. Li and C. Clifton. SEMINT: a tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data and Knowledge Engineering (DKE)*, 33(1):49–84, 2000.

[15] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Rome, Italy, 2001.

[16] A. Motro, P. Anokhin, and A. C. Acar. Utility-based resolution of data inconsistencies. In *Proceedings of the 2004 international workshop on Information quality in informational systems*, pages 35–43. ACM Press, 2004.

[17] F. Naumann, C.-T. Ho, X. Tian, L. Haas, and N. Megiddo. Attribute classification using feature analysis. In *Proceedings of the International Conference on Data Engineering (ICDE)*, San Jose, CA, 2002. Poster.

[18] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 413–424, Bombay, India, 1996.

[19] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[20] E. Schallehn, K.-U. Sattler, and G. Saake. Efficient similarity-based operations for data integration. *Data Knowl. Eng.*, 48(3):361–387, 2004.

[21] P. Singla and P. Domingos. Object identification with attribute-mediated dependences. In *European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, pages 297–308, 2005.

[22] V. S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. Lu, A. Rajput, T. Rogers, R. Ross, and C. Ward. Hermes: A heterogeneous reasoning and mediator system. Technical report, University of Maryland, 1995.

[23] A. Tsois and T. K. Sellis. The generalized pre-grouping transformation: Aggregate-query optimization in the presence of dependencies. In *VLDB*, pages 644–655, 2003.

[24] M. Weis and F. Naumann. Detecting duplicate objects in XML documents. In *Proceedings of the SIGMOD International Workshop on Information Quality for Information Systems (IQIS)*, Paris, France, 2004.

[25] M. Weis and F. Naumann. DogmatiX tracks down duplicates in XML. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 431–442, Baltimore, MD, 2005.

Contributions to Quality-Aware Online Query Processing

Laure Berti-Équille
IRISA, Campus Universitaire de Beaulieu, Rennes, France

Abstract

For non-collaborative data sources, quality-aware query processing is difficult to achieve because the sources generally do not export data quality indicators. This paper presents a prospective work on the declaration of metadata describing data quality and on the adaptation of query processing for taking into account constraints on the quality of data and finding dynamically the best trade-off between the cost of the query and the quality of the result.

1 Introduction

In both centralized or distributed query applications (e.g., in decision support area or Bussiness Intelligence), a set of interesting data sources may be potentially candidate to answer a query. But these sources are usually non-collaborative and do not export information describing the local cost of their query processing, neither indicators of their quality of service (e.g., resource accessibility, reliability, security, etc.), nor information describing the quality of their content (e.g., data accuracy, availability, freshness, completeness, etc.). Relational query optimization has traditionally relied upon table cardinalities when estimating the cost of query plans they consider. While this approach has been and continues to be successful, the need to consider the various dimensions of data quality (such as accuracy, freshness, completeness, etc.) for query execution requires a particular approach. The dual problem is to fix the query cost and search for the "best quality" result, or to fix the result quality and optimize the query cost.

Data quality awareness when querying single or several distributed data sources in a dynamic and distributed environment raises several interesting questions such as:

- Selecting dynamically the adequate data source: different data sources may answer a global query with different response times, query costs and various levels of data quality. How to define strategies for selecting adaptively the most appropriate sources for answering a query with suitable (or, at least, acceptable) data quality?

- Defining semantically and qualitatively correct distributed query plans: the result of a global query is classically built depending on the particular order for the execution of subquery plans. For ensuring data quality awareness, this technique must combine in a coherent way both information and meta-information from the various data sources (*i.e.*, data quality metadata if available). Data quality levels are often unknown, heterogeneous from one source to another, more or less aggregated or locally non uniform (*i.e.*,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

a source may provide excellent data reliability for one specific area, data subset or data type but not for the others). In this context, one of the problems is to control and merge the data quality indicators in a consistent way.

- Making trade-offs between the query cost and the measurable result quality: because one may accept a query result of lower quality (if it is cheaper or has a shorter response time than if the query cost is higher), it's necessary to adapt the query cost to users' quality requirements. The objective is to measure and optimally reduce the cost and bargain query situations where the system searches for solutions that "squeeze out" more gains (in terms of data quality of the query result) than the query without data quality constraints.

- Developing query cost models to evaluate whether the expected benefits from a "quality-augmented" query compensate for the cost of computing or predicting quality indicators and collecting feedbacks from the source and the environment during execution time. The difficulty is to adapt existing query processing techniques to environments where resource availability, allocation, query cost and data quality may be not decidable at compile time.

Several "static" approaches have been proposed to select the data sources before the query processing; they mainly use metadata describing the source content, structure, and quality [6, 7]. The work presented in [6] studied the alternative distributed query plans for mediation systems. Naumann proposed a distributed query planning algorithm that enumerates query plans in such way that it usually finds the best N plans after computing only a fraction of the total number of plans. Upper quality bounds for partial query plans are constructed and thereby non-promising subplans are early pruned in the search tree. This technique relies on source-specific quality criteria and also query-specific quality criteria for the selection phases in the planning algorithm. In [7], Naumann extends this work and proposes mechanisms to follow the execution of the query and, if necessary, to cancel it or change the query plan execution. In [2], the author proposes to take into account data quality estimates when evaluating the users query and deciding the best manner of carrying out the query (which sources to reach, which server to use, etc).

To the best of our knowledge, the issues of data quality-awareness in online query processing have not been much investigated in the literature. In this short paper, we consider data quality for per-query adaptivity of the query processing, and we attempt to approach the problem of quality-aware online query processing extending our previous work in [1].

The remainder of this paper is organized as follows. Section 2 presents an example that illustrates the need for ensuring data quality for online query results. Section 3 states the problems to tackle for quality-aware query processing and briefly presents our proposed solutions for data quality declaration, data and metadata partitioning and quality-aware online query processing. In Section 4, we conclude and present topics for future research.

2 Revisited Example

Adapting the example of Kossmann *et al.* [5], Figure 1 shows the skyline of seaside hotels of Brittany (France) which are supposed to be cheap and close to the beach without considering data quality. Submitting his query, the user wants to get a *big picture* of hotels satisfying his preferences and then, choose the most "promising" hotel to make his room reservation.

In Figure 1, the connected points represent the hotels that dominate the others in terms of minimal price and distance to the beach. The underlying assumption is that the user fully trusts the quality of the data describing the hotels. Retrieved query points are located in the *2-d Data Space*.

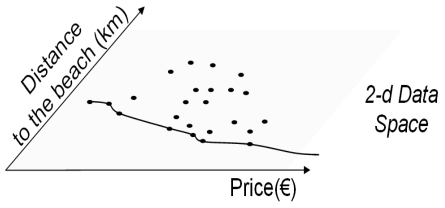


Figure 1: Classical Skyline Query Result

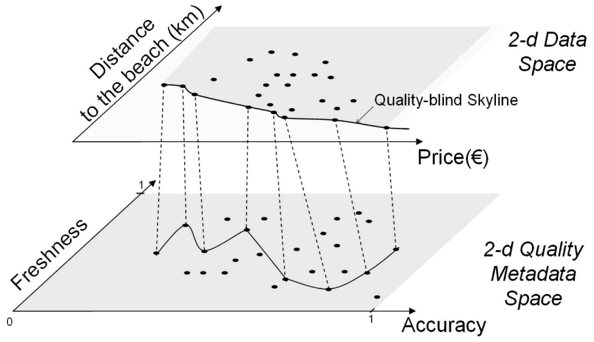


Figure 2: Quality-Blind Skyline Query Result

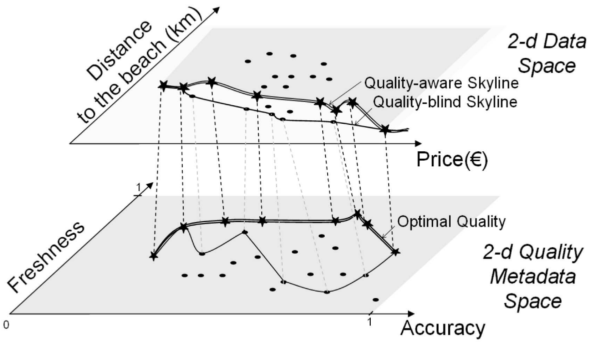


Figure 3: Quality-Aware Skyline Query Result

Consider now data quality as a "multidimensional, complex and morphing concept" [3]. Quality metadata (such as freshness or accuracy, for example) can be associated, at a given time, to each retrieved data point with *(price, distance)* as coordinates. Figure 2 shows the *2-d Quality Metadata Space* describing data accuracy and freshness associated to each retrieved data point of the previous figure. The dotted lines joining the data points from the *2-d Data Space* to the points in the *2-d Quality Metadata Space* represent a scoring function that computes the score (as coordinate) for each dimension of quality (e.g., accuracy and freshness).

Because, the main idea of Skyline queries is to give instantaneously the user the interesting options from a potentially large set of data and, then, let the user make a decision, one might legitimately wonder if a so-called "interesting" points are meaningful when the available data are "dirty" (e.g, not accurate, not up-to-date anymore, or even worse, not complete, or not credible, etc.).

As the probability that a decision uses data increases, the needed data quality increases as well.

Actually, in this example, we observe that interesting points which are part of the initial and "quality-blind" skyline (Fig. 2) have low scores for data accuracy and freshness. In the *2-d Quality Metadata Space* of Figure 3, the optimal quality is represented as a line connecting the points ($\star$) that dominate others points in terms of maximal quality scores for data accuracy and freshness. The corresponding data points in the *2-d Data Space* (called "quality-aware" Skyline) may not be the same retrieved points of the initial "quality-blind" Skyline of hotels with minimal price and distance.

Consequently, answering online queries with data quality-awareness implies to compute interesting points (and recompute them continuously) to produce first results quickly and simultaneously check if they are also optimal in terms of data quality. This may change the initial "quality-blind" skyline of the *2-d Data Space* to produce results with optimal data quality.

3 Problems Statement and Propositions

Ensuring data quality awareness in query processing requires to propose algorithmic, computational, and technical solutions to the five following problems:

1. the definition and both offline and online computation of *generic* and *user defined* functions for measuring or predicting data quality dimensions that could be either specified or called in a declarative and flexible way, or "hard-coded" in the query processor enabling the careful quality-aware query analysis, the preparation of alternative query evaluation plans at compile time, and the selection of quality-aware query plans at run time,

2. the appropriate partition of both k-dimensional data and associated quality metadata; for the case of skyline queries, the partioning method has to enable fast and efficient nearest neighbor searches in the multidimensional data and metadata spaces,

3. the multi-objective optimization of the queries both on data dimensions and on data quality dimensions (for finding the best trade-offs between the cost, the delay of the query, and the quality of the result),

4. the adaptive query processing enabling interactive result presentation to users with possibly changing behaviours when submitting their queries and eventually their requirements or preferences in terms of data quality,

5. the transparent, reversible and explainable result presentation of "quality-augmented" queries, so that these results can be understood and accepted by the users.

In our current work, we attempt to propose several solutions for these requirements for ensuring data quality awareness in the query processing.

3.1 Declaration and Computation of Data Quality Indicators

In [1], we have proposed *XQuaL*, a first version of a quality-extended query language combining SQL and QML (*Quality-of-Service Modeling Language*) proposed by [4] and we have implemented *XQuaL* processor version 2 upon TelegraphCQ V0.2. *XQuaL* is a generic query language extension for describing and manipulating, in a flexible way, data and source quality contracts with `SFW-Qwith` queries.

A quality-extended query (`Qwith-query`) is a SFW query followed by a `Qwith` operator used to declare the required quality constraints based on the notions of quality contract types, contract instances and quality profiles. Table 1(a) presents examples of quality contract types composed of a list of ten named measurable dimensions (e.g., `dataAge` for the contract type named `Freshness`), their corresponding dimension type (noted `dim-type1`, ..., `dim-type10` for the sake of simplicity), the target of the measure that can be applied respectively on values, attribute domains, records, tables or database (noted `ON VALUE, COLUMN, RECORD, TABLE, DATABASE`), and the identifier of the measure function that computes the quality dimension indicator. For example, the function identified by `fid1` for the dimension `dataAge` computes the difference between the current date and the date of creation of each record in the database.

The creation of a contract type associated to a current database implies the execution of the identified functions (*i.e.*, the computation of the declared data quality metrics) and the creation of the contract instances that correspond to the various granularity levels of targeted data. A contract is an instance of a contract type. Table 1(b) gives examples of contract instances, respectively named `fresh`, `accurate`, `complete`, and `available` that respectively correspond to the previously declared contract types named `Freshness`, `Accuracy`, `Completeness`, and `Availability`.

The contract type `Freshness` contract has three dimensions (noted `d1`,$\cdots$, `d3` as comments in Table 1(a) and respectively named `dataAge`, `lastUpdate`, and `updateFrequency`). The type of dimension `updateFrequency` is `dim-type3` composed of a numerical value and a unit `/day` (see Table 1(b)) and it is computed by the function identified by `fid3` applied on each table of the considered database.

Similarly, the dimension `failureMasking` (`d8`) of contract type `Availability` is defined on the whole database and computed by function `fid8`. `dim-type8`, not presented for the sake of simplicity, corresponds

```
CREATE CONTRACTTYPE Freshness(
   dataAge dim-type1 ON RECORD BY FUNCTION fid1, //d1
   lastUpdate dim-type2 ON VALUE BY FUNCTION fid2,//d2
   updateFrequency dim-type3 ON TABLE BY FUNCTION fid3); //d3

CREATE CONTRACTTYPE Accuracy(
   percentageOfContradictions dim-type4 ON VALUE BY FUNCTION fid4, //d4
   NumberOfApproximateMatching dim-type5 ON VALUE BY FUNCTION fid5); //d5

CREATE CONTRACTTYPE Completeness(
   percentageOfNullValues dim-type6 ON VALUE BY FUNCTION fid6, //d6
   numberOfNullValuesPerQuery dim-type7 ON VALUE BY FUNCTION fid7) ; //d7

CREATE CONTRACTTYPE Availability(
   failureMasking dim-type8 ON DATABASE BY FUNCTION fid8, //d8
   serverFailure dim-type9 ON DATABASE BY FUNCTION fid9, //d9
   numberOfFailures dim-type10 ON DATABASE BY FUNCTION fid10) ; //d10
```

(a) Quality Contract Types

```
CREATE CONTRACT fresh Freshness(
   dataAge < 4200 seconds;
   lastUpdate == 4500 seconds;
   updateFrequency == 3 / day);

CREATE CONTRACT accurate Accuracy(
   percentageOfContradictions < 15%;
   NumberOfApproximateMatching < 1 / source);

CREATE CONTRACT complete Completeness(
   PercentageOfNullValues == 8 %;
   NumberOfNullPerQuery == 2 / query);

CREATE CONTRACT available Availability(
   failureMasking IN {noExecution, response};
   serverFailure == initialState ;
   numberOfFailures < 10 / year );
```

(b) Quality Contract Instances

Table 3: Quality Contract Declaration

to the set of possible values among `Omission`, `lostResponse`, `noExecution`, `response`. Creating quality contract types, our objective is to incorporate a set of primitives and data quality functions (e.g., `fid1`, $\cdots$, `fid10`) that can be both computed offline and recomputed or estimated at runtime.

A quality profile can be created in order to specify quality requirements and constraints by combining several instances of contract types with a particular weight. Table 2 gives an example of a profile named `quality` composed of the four previously declared contract instances (Table 1(b)) with a weighted function (`WEIGHT()`).

```
CREATE PROFILE quality(
REQUIRE(fresh, accurate, complete, available)
WEIGHT(fresh 0.4, accurate 0.3, complete 0.2, available 0.1));
```

Table 4: Quality Profile Declaration

One (or several) profile(s) may be used in the `QWITH` part of the *XQuaL* queries and applied on the attributes, tables or database involved in the query. Table 3 presents the naive “nested-loops” way to compute the revisited skyline example with data quality awareness. Price and distance values are queried from table `Hotels` and the profile named `quality` given in Table 2 is applied on the price and distance attributes with the `QWITH` operator.

```
SELECT *
FROM Hotels h
WHERE h.region='Brittany' AND NOT EXITS(
      SELECT *
      FROM Hotels h1
      WHERE h1.region='Brittany' AND
            h1.distance <= h.distance AND
            h1.price <= h.price AND
            (h1.distance < h.distance OR
             h1.price < h.price))
QWITH (PROFILE(quality) ON (price, distance));
```

Table 5: SFW-QWITH Query example

3.2 Quality Metadata and Data Partioning

As soon as a contract type is created and instanciated for the considered database, quality scores are computed for each of the targeted entities (*i.e.*, data values, columns, records, tables of the database). For the sake of simplicity, in the rest of this paper, we choose to focus on the record level among the other data granularity levels and we consider data points as vectors in a d-dimensional data space with d, the number of numerical attributes of the record. Let i ($i \in [1..n]$) representing the data points and j ($j \in [1..k]$) be the required quality profiles (e.g., combining `fresh`, `accurate`, `complete`, and `available`). Let $z_{ij} \in [min_{ij}, max_{ij}]$ be the instances of the declared contract types computed for each data point i on the jth required quality dimension.

Each data point has a scoring function $score_{ij} : [min_{ij}, max_{ij}] \rightarrow [0, 1]$ that gives the score value of the data point i assigned to the quality dimension j in the range of its acceptable values. For convenience, scores are kept in the interval $[0, 1]$.

The relative importance that the user assigns to each dimension is modeled as a weight, w_j, that gives the importance of the quality dimension j (expressed in the declaration of the profile, see Table 2). We assume the weights are normalized, *i.e.*, $\sum_{1 \leq j \leq k} w_j = 1, \forall j \in [0, 1]$. An aggregate scoring function for data point i in the k-dimensional quality space defined by $z = (z_1, \cdots, z_k)$ is defined as : $Score_i(z) = \sum_{1 \leq j \leq k} w_j \cdot score_{ij}(z_i)$.

For analytical purposes we restrict our study to an additive and monotonically increasing or decreasing value scoring function.

As an illustration based on the quality scores of several data points, we can represent a Kiviat graph on a grid of equal-distanced points (*i.e.*, points with coordinates in $[0, 1]$) such that all the vertices are both grid points and score values for each quality dimension. The circumcenter is the point where all quality dimensions are maximal and equal to 1. Figure 4 shows the Kiviat graph for five data points with their corresponding scores on ten quality dimensions (noted, `d1`, $\cdots$, `d10`). A polygon represent the quality of a data point over the specified quality contract types. Searching for the optimal quality data point (for more than three quality dimensions) correspond to find the polygon with the minimal area. Pick's theorem provides a simple formula for calculating the area A of each obtained polygon in terms of the number i of interior points located in the polygon and the number b of boundary points placed on the polygon's perimeter, as: $A = i + b/2 - 1$.

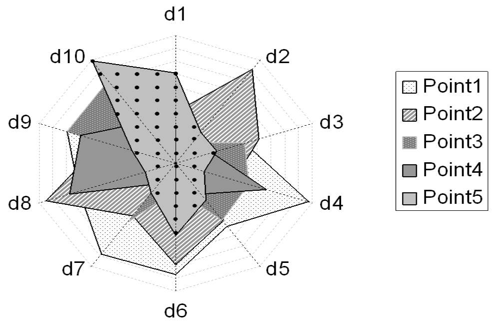


Figure 4: Kiviat Graph for Five Data Points in *10-d Data Quality Metadata Space*

Based on these considerations and faced to high-dimensional data and metadata sets, we propose a method for partitioning data and associated quality metadata. The representation, called *DQ Hyper-Pod*, consists of partitioning data that carves both the d-dimensional data space into homogeneous regions (as hyperspheres) and the k-dimensional quality metadata space into lines for $k \leq 2$ or polygon areas for $k \geq 3$. The *DQ Hyper-Pod* partition method is based on two concepts, the distance from the center of the hypersphere and the projection of the data quality scores that creates a line or a polygonal area whose center is the data point (vector) in the multidimensional data space. The lines or areas representing data quality metadata compose a pod (or envelope) upon each data hypersphere. The *DQ Hyper-Pod* partition method is defined in an Euclidean space and requires an offline phase during which data vectors are first clustered in minimum bounding hyperspheres and outliers isolated.

Consider data points in the *2-d Data Space* representing the vectors for *(price, distance)* of the Skyline of hotels of Figure 1:

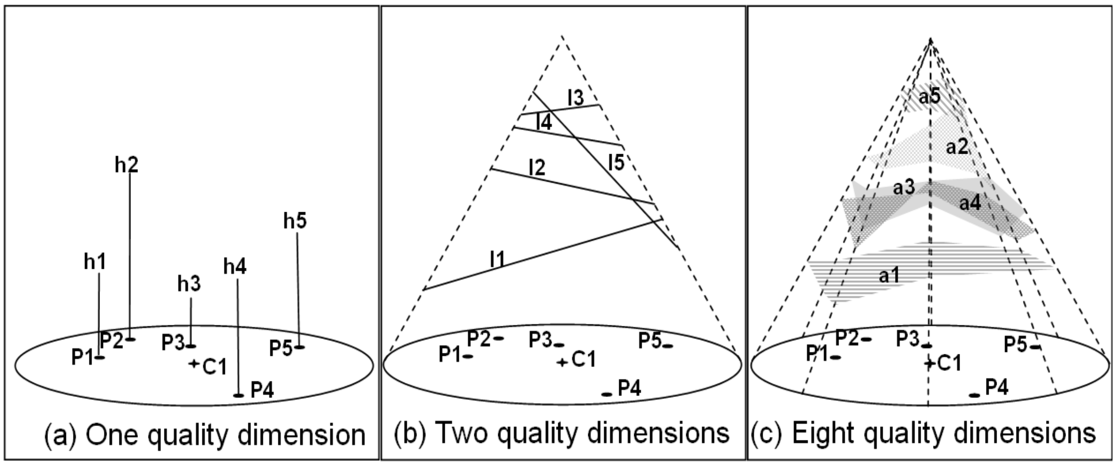


Figure 5: *DQ Hyper-Pod* Representations in *1-d* (a), *2-d* (b), and *8-d* (c) *Quality Space*

- For *1-d Quality Space*, the representation of *DQ Hyper-Pod* partitioning method is hyperspheres with orthogonal lines whose length corresponds to the quality score of each data point for the only one considered quality dimension (see Fig. 5(a)).

- For *k-d Quality Space* ($k \geq 3$), the representation of *DQ Hyper-Pod* partitioning method is right cones with their vertex above the center of their base (also the center of the hypersphere). Each cone of height h and base radius r oriented along the z-axis, with vertex pointing up, and with the base located at $z = 0$. A slant height s_j of the cone is a distance measured along a lateral face from the base to the apex. It supports one quality dimension noted j, such as the score of each data point i is located on it, as: $\forall i, score_{ij} \in [0, s_j]$ with $s_j = \sqrt{h^2 + r^2} = 1$ (see Fig. 5(c)). A polygonal area is defined for each data point joining its scores coordinates per dimension located on respectively on the slant heights of the cone.

- For *2-d Quality Space*, the representation of *DQ Hyper-Pod* partitioning method is a particular case where polygon areas are reduced to lines whose length are in $[0, r]$ and coordinates are defined by the quality scores coordinates on two opposite slant heights of the cone (see Fig. 5(b)).

In Figure 5, we consider the same hypersphere centered on $C1$ with five data points $P1, \cdots, P5$ in the three cases (a), (b), and (c) respectively for one, two or eight quality dimensions for which scores are computed simultaneously with the contract types creation.

3.3 Quality-Aware Online Queries

Back to the initial example, nearest neighbor search is applied in the context of skyline queries [5]. First, let us recall its principle and how *DQ Hyper-Pod* partioning method can be used for data quality-awareness in the online query processing.

We assume we focus on two specific hyperspheres S_i and S_j that cluster data points in the *2-d Data Space* (see Figure 6). The minimum hypersphere bounding the data points, called S_i, is centered on C_i and its radius is r_i. $d_{\min_i}$ denotes the minimum distance between a query point q and C_i, the center of the hypersphere S_i. Based on geometrical properties of these data regions, a classical NN algorithm will first use filtering rules and discard the hyperspheres which the minimum distance to the query q is greater than the farest points of another hypersphere, as: if $d_{\min}(q, C_i) \geq d_{\max}(q, C_j)$ then disguard S_i. Then, the NN algorithm will rank the hyperspheres based on $d_{\min}$, the distance to the query point. For each hypersphere, the distances between the data points and the query point are computed and ranked. Finally, the sequential search is stopped when $d_{\min}(q, C_i) \geq d(q, nn_k)$ with nn_k the kth retrieved query points.

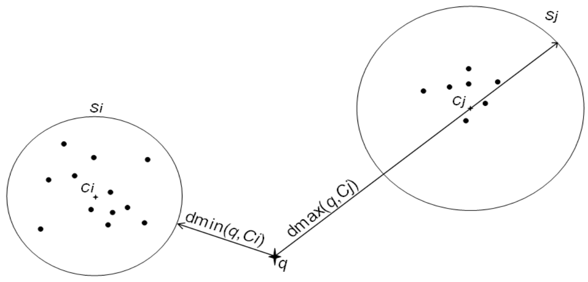


Figure 6: Quality-Blind NN Search

For ensuring data quality-awareness in this processing, we use the *DQ Hyper-Pod* partioning method and consider the quality metadata computed for each data point. Depending on the number of quality dimensions considered in the "quality-augmented" query, the NN algorithm is adapted and applied to the appropriate *DQ Hyper-Pod* representation in the different cases of one, two or more quality dimensions. Similarly to data points, the query is applied the the d-dimensional space and the k-dimensional quality space. Thus, it is a vector in $\mathbb{R}^d \times \mathbb{R}^k$. The adaptation of the NN algorithm mainly consists of two steps:

Step 1: compute the minimal distance to the query point (without considering quality, *i.e.*, $z = 0$) and rank the hyperspheres only based on the data space for retrieving the nearest neighbor,

Step 1: for each hypersphere in the list, consider the quality axis (*i.e.*, z-coordinate) and:

- in the *1-d Quality Space*, rank the data points based on the distance between each data point P_i with x-, y-, and z-coordinates, noted $P_i = (x_i, y_i, z_i)$ and the query $q = (q_x, q_y, q_z)$, as: $d(q, P_i) = \sqrt{(x_i - q_x)^2 + (y_i - q_y)^2 + (z_i - q_z)^2}$ for z_i and $q_z \in \mathbb{R}$,
- in the *2-d Quality Space*, rank the data points based on the length L_i defined by $(z_1 \cdot \frac{r}{h} + d(q, C_i), 0, z_1)$ and $(z_2 \cdot \frac{r}{h} + d(q, C_i), 0, z_2)$ with z_1 and z_2 the scores of the data point i on the two quality dimensions ($z_i \in \mathbb{R}^2$) in the hypershere centered on C_i with radius r and height h and $d(q, C_i)$ the distance between the query coordinates and the center of the hypersphere,
- in the *k-d Quality Space* with $k \geq 3$, rank the data points based on the area A_i of the polygon defined by $z = (z_1, \ldots, z_k)$ the vector of quality scores of each data point ($z_i \in \mathbb{R}^k$).

Figures 7, 8 and 9 respectively show the three case of NN search considering the quality spaces with one, two and four dimensions. In these cases, the query requires hotels with minimal price and distance to the beach and maximal data quality on the declared dimensions in the `QWITH` part of the query, that's the reason why the quality-augmented query is reduced to a single point (e.g.,$((0,0),(1,1,1,1))$ in Fig. 9), and the algorithm rank the data points based on the distance between the apex of each cone in the *DQ Hyper Pod* representation and their aggregate quality scores represented as lines or polygons.

In Figure 7, the algorithm will rank the data points inside each hypersphere based on the distance between the quality-augmented query point and the point defined by the score on the considered dimension. In Figure 8, the algorithm will rank the data points inside each hypersphere based on the length of the lines defined by the scores on the two considered dimensions.

In Figure 9, the algorithm will rank the data points inside each hypersphere based on the area of the polygon associated to each data point and defined by the scores on the various considered dimensions.

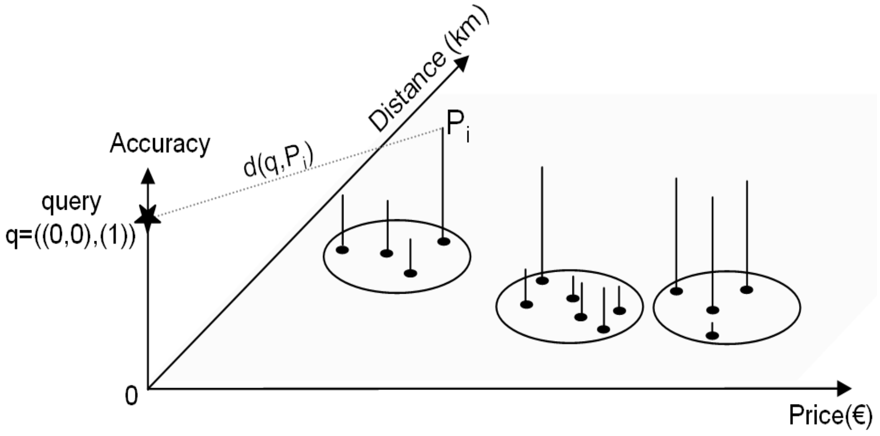


Figure 7: Quality-Aware NN search Considering One Quality Dimension

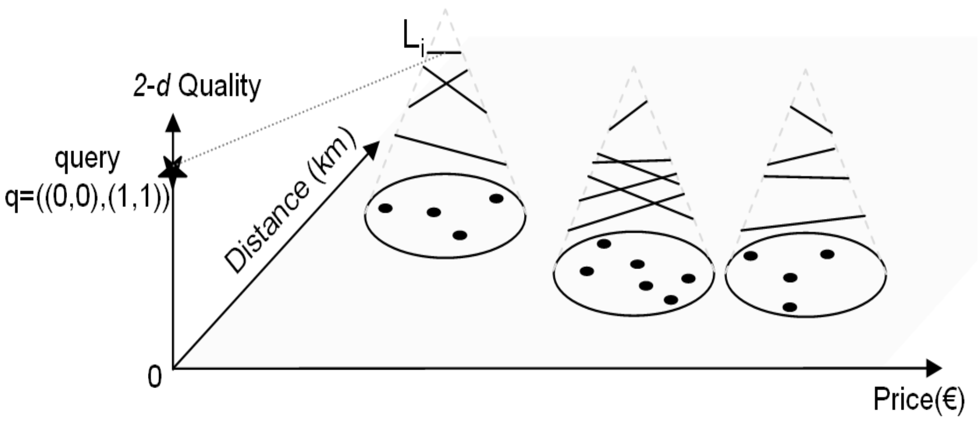


Figure 8: Quality-Aware NN search Considering Two Quality Dimensions

This technique can be easily extended to the cases where the quality scores required in the query are defined by the user and are not necessarly maximal. Hence, the `QWITH` part of the query will be evaluated and the corresponding quality scores coordinates, line length or polygon area of the query will be computed and respectively compared to the list of quality scores coordinates, line lengths or polygon areas that were pre-computed for the data points clustered in the hyperspheres of the *DQ Hyper Pod* partition.

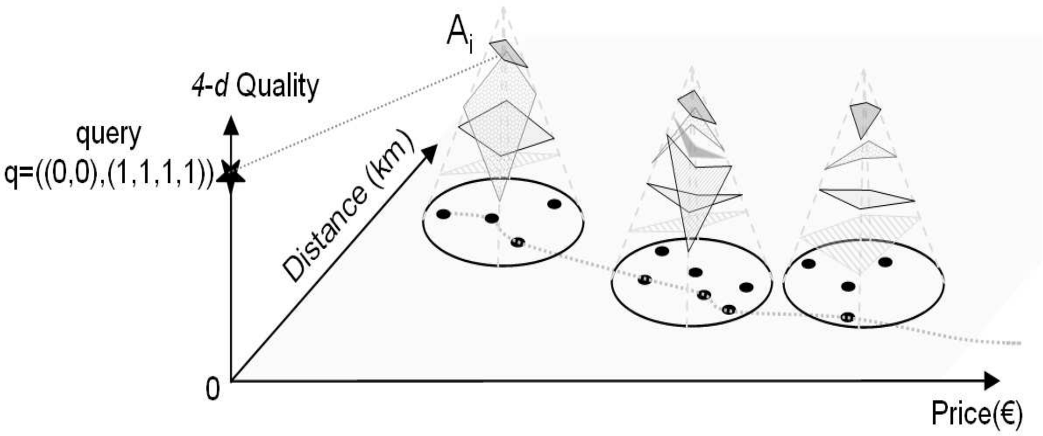


Figure 9: Quality-Aware NN search Considering Four Quality Dimensions

4 Conclusion and Future Research

As mentioned in this short paper, providing efficient access to information sources received a sustained interest since several decades but an interesting research direction in optimizing queries for single- and multi-source data management systems is the use of data quality. Few approaches have been proposed to deal with the various issues of quality-aware query processing mainly in distributed environments (HiQIQ - [7, 6]; ObjectGlobe - [2]). These issues are particularly challenging due the characteristics of the sources, including autonomy, volatility, amounts of data, large heterogeneity spectrum on *i)* data type (e.g., multimedia, XML, relational records, etc.), *ii)* on database schema, and *iii)* on the quality of data and the quality of data management services. An initial motivation is that the constraints on data quality may reflect the user's needs better in such environments. And constraints on information quality are of crucial importance for some critical applications (e.g., homeland security, business intelligence, etc.). A challenging research and development direction is to build quality-aware query processing infrastructures and this requires addressing several research issues as outlined in the following:

- *Quality of data and quality of service extended query languages*. Devise a declarative query language that targets quality of data and also quality of data management service with the advantage that the same quality-constrained query specification holds whatever underlying information is available.

- *Computation model*. In a multi-source infrastructure, the resolution of any "quality-augmented" query may involve an iterative process between the different systems. We need to devise a computation model for the interaction of the different (sub-) systems (e.g., wrapper/mediator systems, sources/data warehouse, peers, Web portals/Web services/Web providers, etc.) in order to ensure data quality awareness (through quality of data and quality of service contract negotiation, for example), not only for the query processing but also for the entire data management and processing chain.

- *Optimization model*. Performance has a prime importance in successfully deploying a quality-aware query processing infrastructure. It mainly relates to query optimization. One challenge is to define appropriate metrics to characterize and measure QoS and QoD depending on the application domain, the systems capabilities and the required performance. The different query planning strategies focus generally on finding feasible and optimal sub-goal orderings based on available bindings and supported conditions at the data sources. Proposed techniques assume a full knowledge of the query capabilities of every participating source. They rely heavily on the way that information sources are described and the objective function of the optimizer (e.g., number of sources, response time, etc.). Using the same source description and quality description models may not always be possible across a large spectrum of data sources. The optimization of quality-aware query processing on structured data (*i.e.*, relational records) as well as on semi-structured data (XML) has to be considered. XML quality-aware query processing is still at its infancy and constitutes a very interesting trend in the near future.

- *Optimization heuristics*. In most of the real world applications, it is quite natural that "quality-augmented" query should meet a number of different and potentially conflicting quality dimensions. Optimizing a particular objective function may sacrifice optimization of another dependent and conflicting objective. An interesting perspective is the study the quality-aware query processing problem from the perspective of multi-objective optimization.

- *Quality-aware adaptive query processing*. Another interesting trend is the use of adaptive and dynamic approaches for dealing with quality-aware query optimization. This is motivated by the intrinsic dynamics of the distributed and autonomous sources where unpredictable events may occur during the execution of a query. The types of actions that are considered in these approaches fall into one of the following cases: *i)* change the query execution plans in order to privilege data quality of query results, *ii)* change the scheduling of operations in the same query execution plan or in different concurrent query plans, *iii)* introduce new operators to cater for the unpredictable events (e.g., improvement or degration of data quality), or *iv)* modify the order of inputs of binary operators. Adaptive techniques have yet to demonstrate their applicability to various real applications with large numbers of information sources. There is also a need to show how they react under heavy quality of data and quality of data management service fluctuations.

References

[1] Berti-Équille L., Quality-Adaptive Query Processing over Distributed Sources. Proc. of the 9th International Conference on Information Quality (IQ04), Cambridge, MA, USA, 2004.

[2] Braumandl R., Keidl M., Kemper A., Kossmann D., Kreutz A., Seltzsa S., Stocker K., ObjectGlobe: Ubiquitous Query Processing on the Internet. The VLDB Journal, 10(1), 2001.

[3] Dasu T., Johnson T., Exploratory Data Mining and Data Cleaning. Wiley, New York, 2003.

[4] Frølund S. and Koistinen J., QML: A Language for Quality of Service Specification. Tech. Rep. HPL-98-10, Software Technology Laboratory, Hewlett-Packard, 1998.

[5] Kossmann D., Ramsak F., Rost S., Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. Proc. of the 28th Intl. Conf. on Very Large Data Bases (VLDB), pp. 275–286, Hong Kong, China, 2002.

[6] Naumann F., Leser U., Freytag J., Quality-Driven Integration of Heterogeneous Information Systems. Proc. of the 25th Intl. Conf. on Very Large Data Bases (VLDB), pp. 447–458, Edinburgh, Scotland, 1999.

[7] Naumann F., Quality-Driven Query Answering for Integrated Information Systems. LNCS 2261, Springer, 2002.

Database Exploration Using Database Dynamics

Tamraparni Dasu, Theodore Johnson, Amit Marathe
AT&T Labs – Research
{tamr, johnsont, marathe}@research.att.com

Abstract

Data analysts often work at a "distance" from the databases they analyze. DBAs of production databases generally do not give logins to data analysts, who might run large disabling queries. Since large and complex databases are often poorly documented, the analyst faces significant problems in understanding the process of how a database is updated, or even whether the processes are correctly implemented. The current trend of outsourcing compounds the problem, as the outsourcing provider has a strong incentive to reveal as little as possible about their operations. In this paper, we explore the possibility of data mining on database dynamics – that is, performing data mining to explore how a databases changes over time. Two uses for the results are data quality monitoring to verify that the database is being properly maintained, and database reverse engineering to give the analyst additional insight into the structure of a very large but poorly documented database. Our methods use summaries of the database which are generated by our database profiling system, Bellman. This approach allows a user with limited computational resources to mine database dynamics.

1 Introduction

Database users frequently work at a distance from their data sources, and do not know how the large and complex databases they use are maintained and updated. In a large enterprise (commercial, scientific, government, or otherwise), the data producers are often in a different organization than the data consumers and analysts. Database management issues increase the separation between users and their database. The DBAs of production databases generally do not give logins to data analysts, out of concern that the analyst might run a crippling query which interferes with ongoing operations. The current trend of outsourcing compounds the problem, as the outsourcing provider has a strong incentive to reveal as little as possible about their operations. As a result, the data user lacks a crucial piece of metadata: by what process is the database updated? Even if the analyst knows what the processes are supposed to be, she doesnt know whether these processes are being followed. In the case of outsourced databases, no-one in the enterprise may know whether or not the outsourcing provider is maintaining critical information correctly.

Large and complex enterprises are usually supported by correspondingly large and complex databases. The most difficult and time-consuming part of a data mining study is often the identification, acquisition, and cleaning of the data to be mined. Similarly, a complex and time consuming first step in database integration is to

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

understand the databases to be integrated and how they relate to each other. The basic problem is one of complexity. Production databases which support large and complex enterprises have hundreds to thousands of tables; a large and complex enterprise will have dozens of such databases. These databases are often poorly and incompletely documented, and existing documentation might be out of date. Even developers who are familiar with a database can encounter problems while adding functionality to a database to support new applications due to the complexity of the existing database (the disorder and complexity in a database is often the result of the continuing need to adapt an existing database to new requirements). As a result, the DBA of a large, complex database might not be fully aware of the database update process, or whether the process is working correctly. A variety of work has been performed with the goal or side-effect of enabling database exploration and automatic extraction of metadata. For example, database profiling has been studied in the literature [10] and is the basis for some commercial database exploration systems [1, 34]. Data integration systems often include facilities for exploring data integration query results (e.g., Potters Wheel [32], and Clio [18]). Additional relevant research is discussed in the related works section.

In this paper, we propose to aid in the exploration of a database by providing information about database dynamics – that is, how the database changes over time. Our methods for mining database dynamics use the kind of database summary information generated by database profiling systems. By mining database changes over database summaries instead of database snapshots, we can inexpensively perform large-scale analyses of database changes. An enterprise data warehouse might gather copies of dozens of production databases together for correlation, analysis, and data mining. End-users often do not have the resources to host multiple copies of these snapshots; and in any case many organizations have rules which prohibit the proliferation of database copies (for version management and data security reasons). Since the database summaries are small, an end-user can inexpensively support time series analyses of database changes even in very large data warehouses. Further, end-users can store the summaries while they might be prohibited from storing database copies.

There are many applications of database dynamics information; we roughly group them into two types. In database data quality monitoring, we try to detect anomalous or suspicious behaviors while in reverse engineering, we try to understand properties of the database.

Database change information is useful for data quality monitoring. We can place a number of a-priori constraints on the database, then watch for violations. Have important tables or fields been dropped? Did the unfilled orders table fail to get updated? Conversely, was there a change in a table which should be static? Are new values infiltrating a field which should have a limited domain? We can also detect possible data quality problems by searching for deviations from past behavior. For example, a previously static table or field which exhibits a significant change can indicate either a data glitch or a change in procedures. An active entity which fails to change can indicate a data warehouse loading problem or a problem in production processes.

One common example of a reverse engineering activity is to try to find the important tables in the database. This task is surprisingly difficult, e.g. in a poorly documented database with hundreds or thousands of tables. The database explorer can use a variety of clues to find interesting tables: table and field names, documentation, the number of rows in a table, and samples of row and field values. Several database exploration systems provide this kind of information [1, 10, 34], but there is still a missing piece: the update activity on the table. Is the table regularly updated or is it static? If the table is regularly updated, what is the nature of the update – small or large; added and deleted rows, or changes in field values? Which fields of a table are being updated? These properties indicate how the table is used. For example, a static table might be a relic that is no longer in use. A table of unfulfilled orders should exhibit large changes from week to week, while a table of customers should exhibit relatively small changes.

In this paper, we explore the use of time series of database profiles, generated by our Bellman profiling tool, to mine information about database dynamics, which to our knowledge has not been done before (except for some degree in [23]). Our contributions are:

1. Evaluating the effectiveness of different types of profile data for performing database change detection,

2. Developing methods for mining information from profile time series, and

3. Evaluating these methods on profile time series from two production databases used by a large enterprise.

Our analysis techniques are deliberately simple exploratory methods – the contribution of this paper is the extraction and exploration of a novel data set. Nevertheless, these techniques can find many interesting properties of the databases we explore.

1.1 Related Work

The results in this paper use data gathered by a database profiling system [13] small but informative summaries of database contents. Commercial providers of profiling systems include Similarity Systems [34] and Ascential [1]. For our research we use data collected by Bellman [10] developed and used at AT&T. These summaries can include keys and functional dependencies [6, 20, 30, 33]. In [1] the authors present techniques for creating summaries based on information-theoretic clustering. However, Bellman does not incorporate this kind of profile data so we do not explore its use for change detection in this paper.

Database reverse-engineering is a pre-requisite for schema mapping and database integration. Several schema mapping efforts have incorporated database exploration tools [12, 17, 18, 29, 30]. Some schema mapping tools also allow the exploration of the database integration results [18, 32].

Recent work has developed tools for keyword search in a database [1, 3, 5, 21, 22]. Other types of advanced search tools include regular expression indexing [8, 19]. A related topic is that of approximate join for which we refer the interested reader to the tutorial by Koudas and Srivastava [16].

Some of the results in this paper can be considered to be data mining for data quality. Other methods include the use of multiple independent models [27], non-parametric outlier identification [26], and automated learning to detect data quality problems [25].

Most commercial DBMSs optionally allow the logging of all queries submitted to the system. An analysis of the query log could provide a more direct method for reverse-engineering a DBMS [28]. However, query logs are usually not available to end-users, and their contents can be difficult to interpret.

Some recent work has addressed detecting changes in data sets [14, 15, 24]. These papers treat the contents of a table (or stream) as a multivariate distribution and look for changes in the distribution (in [14] by measuring distance through classification models, in [15] by partition-wise comparison, while [24] tests stationarity by measuring distance between distributions). In [7], the authors use a sampling method to detect the frequency by which a data source (web pages or data warehouses) changes. The focus of this paper is to minimize the cost keeping a local copy of the data up-to-date. A side effect of the algorithm presented in [7] is to estimate the rate at which data sources change, but requires a copy of the data set. The most closely related work is [23], which monitors the rate of change in text databases by measuring longitudinal change in a summary (an inverted list index). However, this study only measured table changes, used a single summary, and did not attempt to infer properties of the table (the issue was finding a schedule for loading a local copy). In contrast to previous work, we present a method for mining database dynamics which extracts information about database management and update procedures, and is the first study of its kind that we are aware of.

2 Profile Data

The profile data that we use for our study is that collected by the Bellman profiler. The basic profile data collected by Bellman is:

- The schema of the database.

- The number of rows in each table.

- The number of distinct values of each field.
- The number of NULL values of each field.
- The frequency distribution of each field.
- The top ten most common values of each field, and each values frequency of occurrence.

Basic profile data is collected by the commercial database profiling systems that we are aware of [1, 34]. The commercial database profilers generally collect some additional information, for example the keys and functional dependencies in the tables. While Bellman can also collect key and FD data, the longitudinal data set that we had access to did not include them (computing keys and functional dependencies is expensive, so they were not collected). Commercial profiling systems can also collect row samples; Bellman does not. While it is clear that, e.g. keys and functional dependencies are very useful, e.g. for data quality monitoring, we cannot report any results about them.

Bellman also collects minhash signatures and sketches of some database entities. While these approximation techniques are well-discussed elsewhere [10], we provide a brief review here to make the paper self-contained.

The resemblance between two sets is the size of the intersection of the sets divided by the size of their union.

$$\rho = \frac{|A \cap B|}{|A \cup B|} \tag{5}$$

The resemblance of two sets can be approximated by a minhash signature: Let hi be a hash function defined over the domain of set A. Let $s_i(A) = \min\{h_i(a)|a \in A\}$. Then $\Pr[s_i(A) = s_i(B)] = \rho$. We repeat this experiment N times, and the minhash signature of a set A is the collection of these experiments: $S(A) = \{s_1(A), \ldots, S_N(A)\}$. Bellman collects minhash signatures over multisets (e.g., the multiset of values of a field), providing an opportunity to collect an approximation to the frequency distribution of the multiset: $M(A) = \{m_1(A), \ldots, m_N(A)\}$, where $m_i(A)$ is the number of times that $h_i^{-1}(s_i(A))$ occurs in A.

Bellman also makes use of random vector projection sketches. Let V be a d-dimensional vector, and X a d by k dimensional matrix, $k \ll d$. Then the sketch of V is

$$SK(V) = V \cdot X \tag{6}$$

Where SK(V) is a k-dimensional vector. The L_2 distance between two vectors V_1 and V_2 can be approximated by

$$L_2(\hat{V_1, V_2}) = L_2(SK(V_1), SK(V_2)) \tag{7}$$

New fast sketch techniques, e.g. the count-min sketch [9], have been proposed, but they have not been incorporated into Bellman at the time of our study.

The sketch and signature profile data that Bellman collects is

- The minhash signature of value each field.
- The minhash signature of the values of the rows of each table.
- The minhash signature of the set of 3-grams of each field.
- The sketch of the frequency vector of the multiset of 3-grams of each field.

3 Experimental Setup

Bellman is used on a variety of projects within AT&T, providing a source of data for our experiments. One project in particular is a data warehouse consisting of copies of production databases, to be used for intensive data mining analyses. Currently, this data warehouse contains 15 databases updated weekly, with a total of 2899 tables. We selected two of these databases, which we call A and N, to study. We chose these particular two because among the large and complex databases in the warehouse, they were the most reliably updated and the best understood by the local analysts (but the documentation for these databases was quite meager). These databases also represent the distance problem that the analysts face, as corporate policy forbade the analysts from making a local copy (for version management and data security reasons), and the management of these databases had been outsourced.

We arranged with the Bellman developers to collect snapshots of the weekly profiles, timestamped with the profile collection time, for two of the fifteen databases. At the time of this writing, the first database (A) contains 385 tables and 3522 fields, while the second database (N) contains 508 tables and 3547 fields. We collected 33 profile snapshots, allowing for 32 successive weeks of comparison data.

We supported our experiments by making a pre-analysis of the data, computing time series of table and field self-resemblances by value, by q-gram, and by q-gram distance; and also summaries of changes in other field statistics. Databases A and N contain more than 420 million rows and represent about 100 Gbytes of data; computing their profiles requires about 8 hours of computing time (on a 2.8 GHz, 2-processor server) and results in about 500 Mbytes of profile data. Computing the summary tables requires about an additional hour of compute time, after which all queries used to generate the data in this paper ran in real time. Since the data warehouse is profiled weekly anyway, the results in this paper provide database dynamics information almost ”for free”.

4 Experiments

In this section, we explore the efficacy of different types of profile data to detect various types of changes in a database. This section is divided into subsections based on the type of change we are trying to detect.

We do not examine the use of schema information in this paper. Although schema information is quite useful for detecting added or dropped tables or fields, detecting these changes is straightforward (this kind of a data quality alerting facility was added to Bellman without the need for research). We note that we found many examples of added and dropped tables and fields in the course of our study.

This study is limited by the types of databases we study, the types of profile data collected on them, and the local knowledge about these very complex databases (but obtaining the data used for the experiments was a multi-year effort). The results of the experiments indicate the type of information that can be found by mining database change information. Different applications will have different characteristics, so one cannot expect the precise results of this study to be universally applicable. Still, databases A and N show remarkably similar behavior, and provide an indication of what one might find in other databases.

4.1 Database Change Detection

In the context of a data warehouse receiving weekly updates, an important question is, did the database refresh occur successfully? Although it would seem that this is a simple question to answer (e.g., look at the server logs), in a large enterprise there can be complications. The data warehouse we studied is managed on a shoestring budget by an organization remote from its users. The data loads are accomplished using automated scripts, and the DBAs do not have the resources to analyze the loading process. The database load might fail for many reasons: the backup image was not transferred from the production database, resource limitations such as lack of disk space, a failure in an ETL process, and so on. So, the task of evaluating the quality of the database refresh falls to the users.

There are two types of profile data which naturally suggest that they can be used to measure a table change: the change in the number of rows in the table, and the resemblance of the row values of a table to themselves (table self-resemblance). Self-resemblance would seem to be the more sensitive of the two tests, as it can detect changes to a table made using row updates only. However, we approximate the table self-resemblance using minhash signatures, which cannot reliably detect small changes. To evaluate these two methods, we collected 12249 week-to-week comparisons of profile data for the A and N databases. We counted the number of table changes detected using combinations of row count change and a table self-resemblance of less than 1. A summary is in Table 6.

Row count and self resemblance	2256 (18.4%)
Row count only	2857 (23.2%)
Self resemblance only	2520 (20.6%)
Row count or resemblance	3121 (25.5%)

Table 6: Number of table changes detected.

By plotting the number of changes over time, we can track aggregate database behavior. In Figure 1, we plot the week-to-week fraction of the tables in database A with a change detected by self-resemblance, change in row count, and both. Figure 2 shows the same chart for database N.

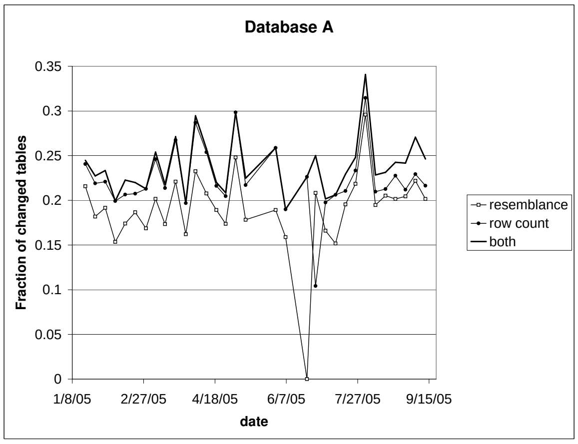


Figure 1 : Tables with a detected change.

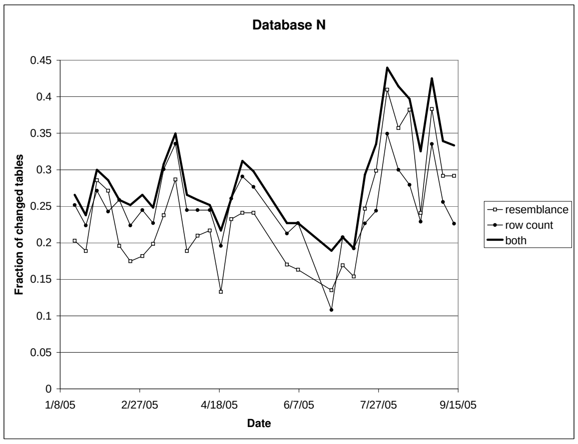


Figure 2 : Tables with a detected change.

One striking feature of these databases is that most of the tables are static – only 20% to 30% of the tables in Database A and 20% to 40% of the tables in Database N change from week to week. We found this property to hold for all of the 15 databases in the data warehouse. Therefore, looking for changing tables can be an e ffective way to reverse engineer a database. We explore this topic further in Section 4.2.

In addition, Figure 1 indicates a data refresh problem in Database A for its 6/21/05 data point. Using self-resemblance indicates no week-to-week change, while using row count does. Further investigation shows that while the average percentage change in the row count (of tables whose row count changed week to week) is 5.9%, the row count change during that week is only .4%. We later confirmed a data refresh problem.

A comparison of the changed tables detected by self-resemblance and row count indicates a subtle but distinct change in database dynamics when we compare pre-6/21 to post-6/21 behavior. The fraction of tables which change increases dramatically in Database N. In addition, self-resemblance accounts for a larger number of detected changed tables. For example, in Database N more changes are detected by self-resemblance than by row count after 6/21, but fewer before 6/21. We confirmed that data loading procedures were improved during this period in response to user complaints.

4.2 Table Change Detection

The results from Section 4.1 suggest that row counts are generally a better indicator than self-resemblance for detecting changed tables. However, in that section we are primarily interested in determining whether or not tables did change. In this section, we will also be interested in the degree by which the table changed. Therefore, we need to determine which measure is more sensitive in detecting the degree of change. In Figure 3, we plot the degree of table change detected using table self-resemblance vs. the degree detected using row counts for the 3121 instances where we detected any table change using either method. Unsurprisingly, self-resemblance is almost always a more sensitive indicator of change (as shown by the fact that almost all data points are below the diagonal).

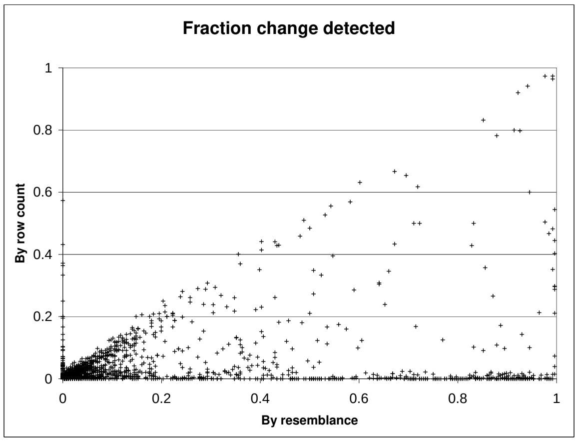


Figure 3 : Fraction of change detected in tables using row count and resemblance.

We can try to classify the type of updates to a table by the correlation of the fraction of change detected by row count to the fraction of change detected by self-resemblance. If the change is primarily due to insertions or deletions, the resemblance change and the row count change should be strongly correlated as indicated by the diagonal line in Figure 3. If the change is due primarily to updates, the row count should stay the same while the resemblance should show a large change – as indicated by the horizontal line in Figure 3. Changes due to a mixture of inserts, deletes, and updates should lie between these two lines.

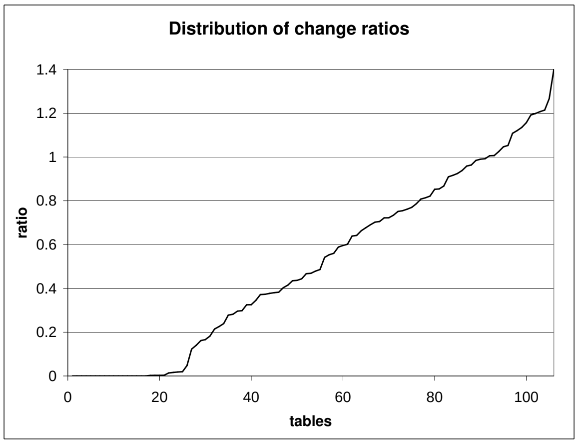
Next, we can characterize how a table changes: Is it mostly updates, inserts or deletes, or a mixture. Is the mixture consistent? We collected the 2520 instances where table self-resemblance indicates a week-to-week change in a table. Next, we computed the ratio of the fraction change in row count to the fraction change in self-resemblance:

$$ratio = \frac{|count_{old} - count_{new}|}{1 - \rho} \tag{8}$$

We next analyzed this data on a per-table basis, for the set of tables with at least four week-to-week changes (80 tables in database A, 66 tables in database N).

We computed the coefficient of variation (square root of the mean squared error divided by the mean) for the fraction change ratio, and found a wide range of coefficients. To limit our scope to well-behaved time series, we selected those tables with at least four changes detectable by resemblance and a coefficient of variation less than .75 (an arbitrary value that we selected by eyeball). We are left with 46 tables in database A, 60 tables in database N.

In these 106 tables, we found 26 with a ratio of 5% or less, and 27 with a ratio of 85% or greater (Figure 4 shows the distribution of the ratio, by sorting tables on their change ratio). Since we chose these tables because their change ratio time series is reasonably well behaved, we find that we can distinguish between these tables based on the types of updates they normally receive. For example, consider the change ratio time series for the three tables in Figure 5. One table ($ratio < .05$) changes primarily through updates, another ($ratio = .5$) through a mix of inserts, deletes, and/or updates, and the third ($ratio > .85$) primarily through inserts (we verified that its size increases, not decreases, over time).



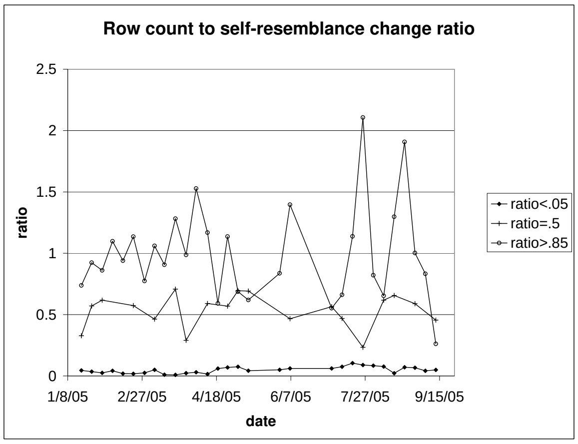


Figure 4 : Row count vs self-resemblance change.

Figure 5 : Change ratios for three tables.

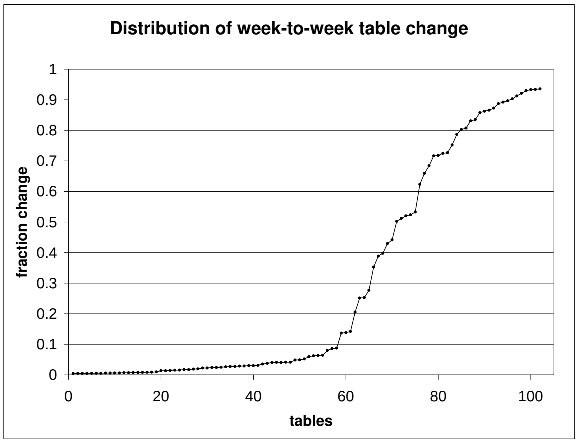
We can also classify tables by the degree to which they change from week to week. We selected the tables which changed at least six times as indicated by self-resemblance, computing the change to be $1 - \rho$. The result is 63 tables in Database A and 60 tables in Database N. Next, we computed the trimmed coefficient of variation for these tables[1] and selected those whose coefficient of variation is .75 or less. The result is 54 tables from Database A and 48 tables from Database N.

These tables show a wide variation in the fraction by which they change week to week, as is shown in Figure 6. 22 of the 102 tables change by an average of 1.5% or less per week, and 13 of the tables change by 85% or more. Figure 7 shows the week-to-week fraction of change for a table with small, medium, and large week-to-week change, respectively (the large-change table was added in July 2005).

Yet another way to classify tables is by whether or not they tend to recycle their rows. We call a table which recycles its rows convergent, while a table which tends to create new rows divergent. We look for convergent tables by comparing the time series of four week resemblance changes to an estimate of the four week resemblance change computed using 1-week resemblance changes. We estimate the 4-week resemblance by the product of the previous four 1-week resemblances, and the 4-week change by one minus the four week resemblance.

We found that this simple estimator generally works well. That is, we found that most tables are divergent. For the 63 tables in Database A and 60 tables in Database N which have at least six changes indicated by resemblance, we computed the ratio of the actual four-week change to the estimated four-week change. For

[1]For this analysis, we ignore zero change ($\rho = 1$) data points, because they might indicate only a failure to refresh.



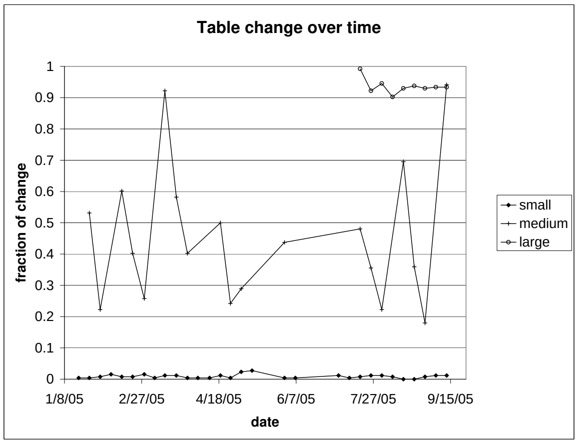
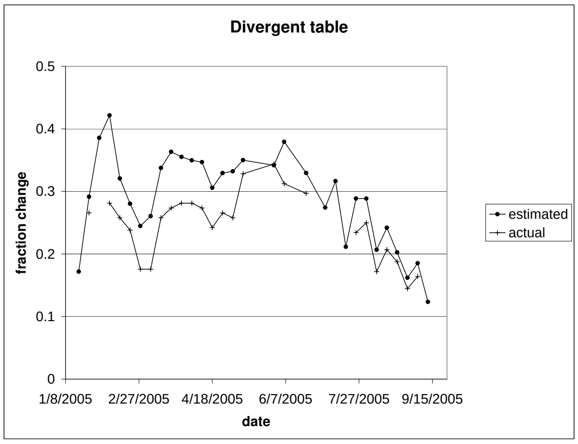


Figure 6 : Fraction of change week-to-week. Figure 7 : Small, medium, and large week-to-week change.

each table, we counted the number of times that this ratio is .7 or less. We selected the tables with a count of six or higher as the convergent tables. We found 10 convergent tables in database A, and 6 convergent tables in database N.

Figure 8 shows a comparison between the estimated and actual 4-week resemblance changes in a divergent table. This table actually shows a change in its behavior. Before 5/9/2005, the actual change is somewhat less than the estimated change (about 77% of estimated change, above our somewhat arbitrary 70% threshold). On 5/9/2005 and after, the estimated change closely tracks the actual change. Figure 9 shows estimated and actual four-week changes for a convergent table. Its behavior is different than that shown in Figure 9, as actual percentage change is persistently about half the estimated change – indicating that this table tends to recycle its rows.



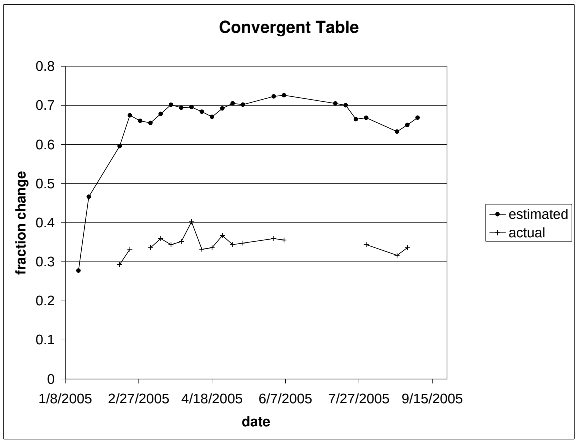


Figure 8 : Divergent table, 4-week changes. Figure 9 : Convergent table, 4-week changes.

There is one more type of information we can use for classifying changes in tables changes in the q-grams of their rows considered as strings. Recall from Section 2 that we have two types of q-gram information: q-gram resemblance and q-gram distance. To evaluate these metrics, for each of the 123 tables with at least six changes detected by value self-resemblance, we computed the ratio of the week-to-week change detected by value self-resemblance to the change detected by q-gram self-resemblance, and to the change detected by q-gram distance. The result is shown in Figure 10.

Neither q-gram resemblance nor q-gram distance is a good metric for determining that a table has changed. However, the distribution of ratios for q-gram resemblance suggests that this ratio can classify tables based on the rate at which they change their textual content (the peaks in both charts are due to only two tables). The changes detected by q-gram resemblance versus q-gram distance are only moderately correlated: we found an R^2 value of only .33 when performing a linear regression between the two variables, and that only after removing two outliers. Since row q-gram resemblance is more sensitive in detecting textual changes, we recommend it as the better mechanism for classifying this kind of table change.

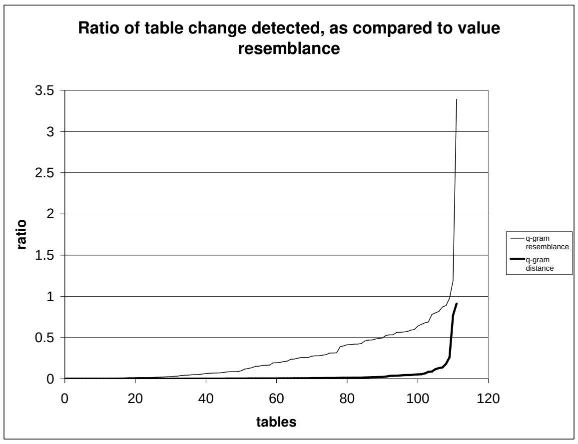


Figure 10 : Row value change versus row q-gram change.

4.3 Field Change Detection

For detecting changes in fields, we have a wealth of information. In addition to the value and q-gram resemblances, and q-gram distance information, we have the count of the number of distinct items, the count of the NULL values, the top-10 values, and an approximation to the frequency distribution of the values of the field. We limit the scope of our experiments to fit within page limits and minimize reader fatigue. We will use value self-resemblance and frequency distribution information to characterize the nature of the updates to the tables, and q-grams, count distinct, count NULL, and top-10 information to discover unusual events.

In Section 4.2, we search for "hot" tables – those that get updated – and found that a small fraction of the tables in the database get updated from week to week. A natural extension is to search for "hot" fields. That is, can we identify a small collection of fields in a table which account for the updates? For every occasion in which a table shows a table self-resemblance change, we compute the fraction of that tables fields which also show a value self-resemblance change. We plot this fraction for the 2732 week-to-week table changes in Figure 11. We find that in a large number of tables, the values in only a small set of fields have value changes on 130 occasions, there are no value changes in any fields. Conversely, on 274 occasions, 90% or more of the tables fields show a value resemblance change.

Next we can ask whether there are fields of a table that are persistently "hot", in terms of value resemblance change. We selected the 123 tables which showed a table self-resemblance change on six or more occasions and analyzed the number of times each of their fields show a change. We only considered those fields which exist in every observation (there was a small but significant number of added and dropped fields). The result

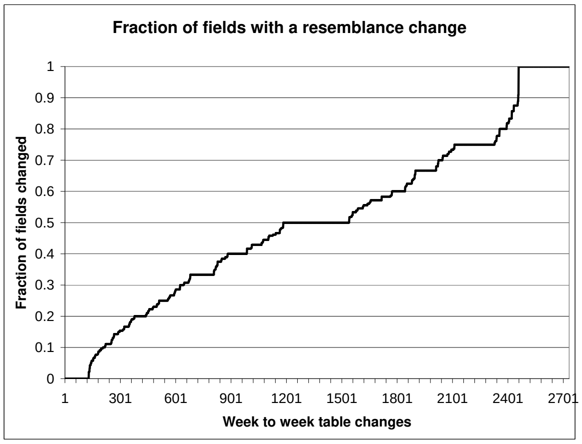


Figure 11 : Fieldswith a value resemblance change.

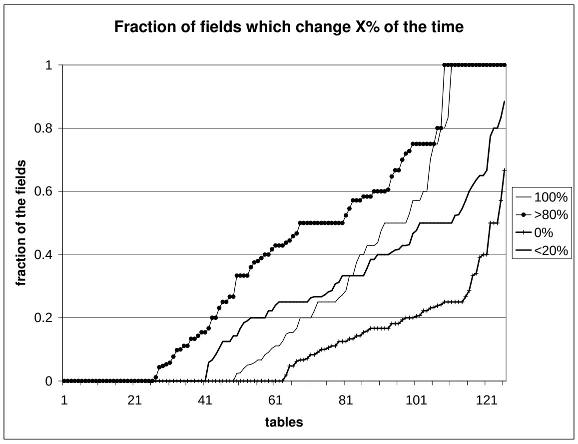


Figure 12 : Fields with persistent changes.

is in Figure 12. Although there are a significant number of occasions in which none of a tables fields show a resemblance change, there are no tables whose fields persistently never, or even rarely, change. However, for most tables there is a subset of hot (frequent value change) and cold fields (infrequent value change).

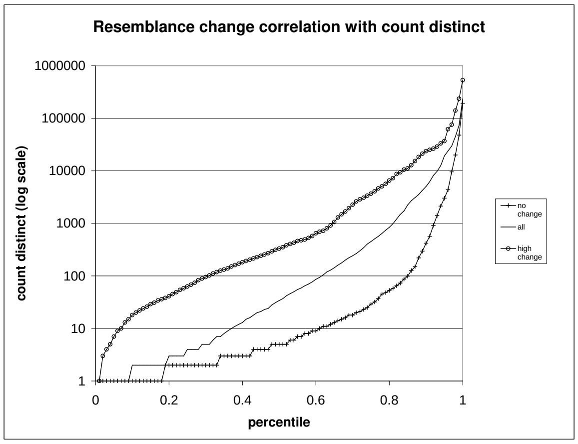
Is there significance to the set of fields whose values are not changing? In most databases there is a large collection of fields with only a few values, e.g. Male/Female. One would expect that values in these fields do not change. To answer this question, we correlated the number of distinct items in a field to its resemblance change during the occasions where the fields table has a week-to-week resemblance change. We plot the distribution all of fields, for those fields which do not have a resemblance change, and for those which have a high resemblance change (larger than or equal to the table resemblance change) in Figure 13. As expected, fields with large changes tend to have significantly more distinct values than usual, and fields which do not change tend to have significantly fewer distinct values than usual. However, there is a significant fraction of fields with few distinct values but a large resemblance change, and also of fields with many distinct values but no resemblance change.

We can use the change in the distribution of a fields values as a more sensitive metric. Of the 34727 field/occasions under consideration, the self-resemblance is 1.0 in 20327 of the fields, but the distribution is identical in only 6555 of these. A failure for a field to change in distribution strongly suggests that its table is changed by updates on other fields. Are there fields which persistently do not change their distribution? We collected 1474 fields which occur six times or more in tables[2] and which show a resemblance change, and counted the number of times when each of these fields showed zero change in resemblance and distribution. We plot the results, in terms of frequency of no change, in Figure 14. We find 90 fields which never change, and 144 fields which do not change 80% of the time or more when their table changes. These non-changing fields occurred in 24 of the 207 tables which showed six or more resemblance changes.

For the remainder of the field change data (q-gram resemblance, q-gram distance, number distinct, number null, top-10), we search for significant changes in the field. We collected field change data such that the fields table showed a week-to-week change and for which we could join field statistics, table row counts, q-gram resemblance, and q-gram distance. The result was 18078 rows.

- We compared the week-to-week change in the number of NULL values of each field, and divided by the (ending) number of rows in the table. We found 59 occurrences in which the number of NULL values changed by 20% or more.

[2]We need this restriction because some fields were added or deleted during the course of the study.



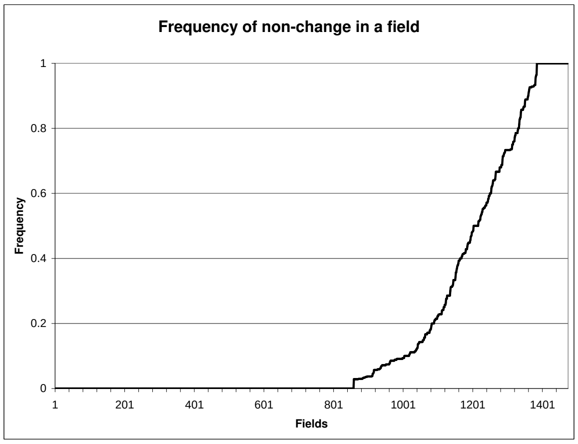


Figure 13 : Distribution of field count distinct values. Figure 14 : Frequency on non-change.

- We compared the week-to-week number of unique values of each field, and divided by the (ending) number of distinct values. We found 407 occasions in which the number of unique values changed by 20% or more.
- We computed the number of top-K values which were common week-to-week and divided by the number of top-K values at the end of the week. We found 1308 occurrences in which 20% or more of the top-K values changed.
- We found 630 occurrences in which the week-to-week q-gram resemblance indicates a change of 20% or more.
- We found 186 occurrences in which the week-to-week q-gram distance is .2 or more.

Unsurprisingly, these indicators of unusual behavior in a field are strongly correlated.

4.4 Time Series Analysis

Since we have a time series of table change data, we can try to look for tables which change in similar ways. If the changes are strikingly similar enough, we might even suspect that there is a similar process which updates the tables. To do this, we need to cluster tables based on their time series. For this experiment, we use the week-to-week table self-resemblances.

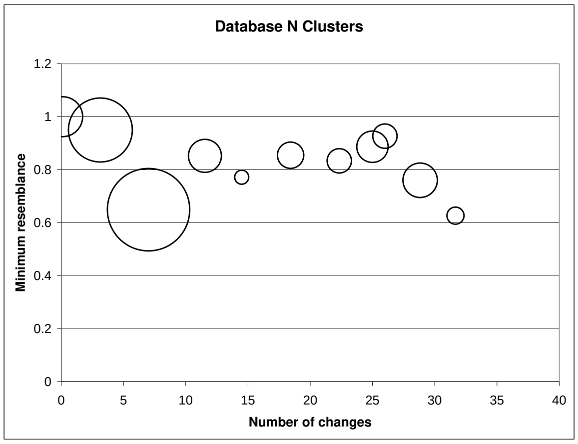
A common approach to clustering time series involves summarizing the time series using summaries such as Fourier transforms and clustering the resulting fixed length vectors. However, model based clustering can fail when the assumptions are not true. We know very little about how tables are expected to change from week to week; therefore we use the fast, simple, and nonparametric method to cluster the time series, described in [11]. This method computes simple nonparametric summaries of the time series, and then clusters time series based on their summaries.

In this specific instance, we use the following three descriptors: average resemblance, the number of times the resemblance is less than 1.0, and the smallest resemblance (i.e. the biggest change). We then run a fast k-means clustering algorithm (FASTCLUS from SAS) with 10 and 20 clusters. While seemingly simple, the technique is very fast and effective, resulting in clusters that clearly identify different types of behavior. The

”goodness-of-fit” criteria provided by the software (e.g. R-square of 0.98) all indicate that clustering was effective in explaining the variability of the three clustering attributes. Note that our aim here is to create approximate groupings which explain behavior, rather than exact, optimally separated clusters. The approximate clusters enable us to partition the data into smaller pieces for further examination of the peculiarities of individual time series.

Since Database A and Database N show similar behavior, we concentrate on Database N. We found that 20 clusters produced many singleton clusters (i.e., capturing outliers), so we chose to examine the 10-cluster results. Figure 15 is a bubble chart showing the minimum resemblance and number of detected changes axes for database N (the bubble size indicates the size of the cluster.

We picked a few of the 10 clusters to show as representatives. Figure 16 shows a low-activity cluster, while Figure 17 shows a high activity cluster, demonstrating that clustering technique does tend to group similar behaving time series. In Figure 18, the tables are mostly quiescent, but many of them show a significant change at week 27. These tables all show an anomalous change of similar magnitude at the same time period, suggesting that there is a similar process behind this update.



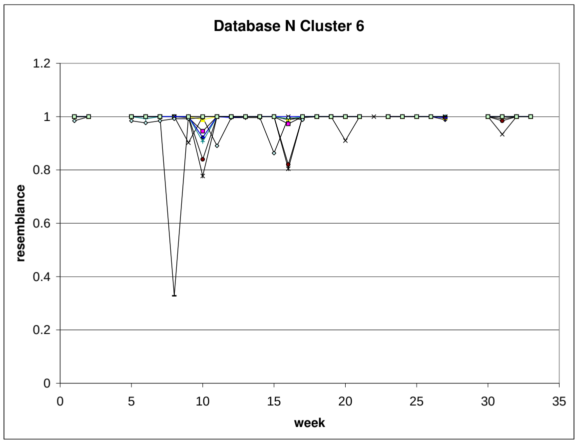
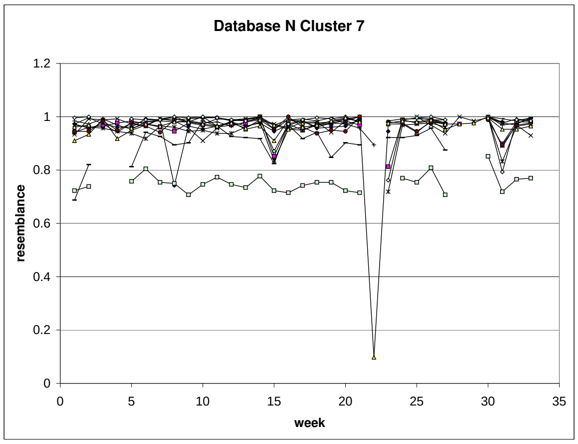


Figure 15 : Clusters in Database N’s value resemblance time series. Figure 16 : Low activity cluster.

For our final experiments, we attempt to find anomalous behavior in the week-to-week table self-resemblances using time series analysis. We use the table self-resemblance time series and search for anomalous table updates. At first we tried to capture the structure in the resemblance time series using linear regression models of various sorts. However, the data was not amenable to such models due to high correlations and singularity issues. We then switched to a control chart type of approach, modified slightly to meet our needs.

We computed a set of four-step moving averages for each time series. We then computed the average and standard deviation of these moving averages to serve as a central reference line of the control chart and basis for the confidence or error bounds respectively. We used 2-sigma limits, and flag an alert whenever the self-resemblance is outside these bounds. Analogously, we also computed the median of the moving averages and 10th and 90th quantiles to serves as error bounds. However, we believe that these bounds are less reliable than the mean and standard deviation based ones due to the limited amount of data.

The control chart technique raised at least one alert on almost half of the time series. Many of these alerts indicated an anomalous update to a table; Figure 19 shows an example. There were also many alerts as shown in Figure 20. Here the most extreme alert indicates a failure to update the table.



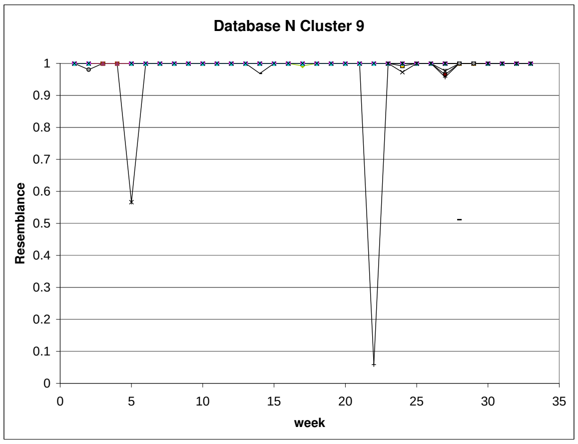
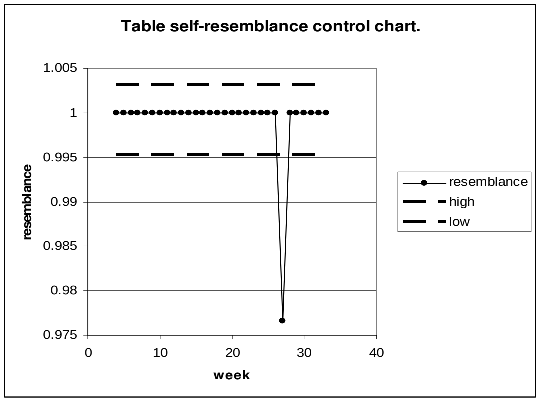


Figure 17 : High activity cluster. Figure 18 : Cluster indicating an anomolous correlated change in tables.



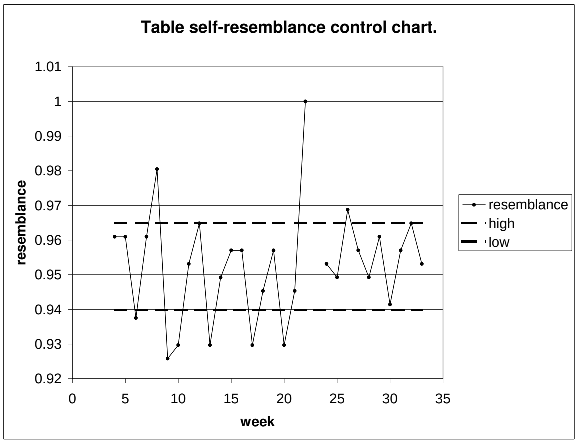


Figure 19 : Anomalous table update. Figure 20 : Failure to update a table.

5 Conclusions

A common problem faced by database users and maintainers is that of understanding what is in the database. A production database for a complex enterprise is generally a quite complex thing itself, separated from the end-user, and often outsourced. Frequently, even the DBAs of a database dont completely understand its operations, sometimes leading to disastrous mistakes in maintenance and upgrades.

Data warehouse maintainers and users have even fewer resources than the DBAs for understanding what is happening in a database. In fact, this project was motivated by problems encountered in using a large data warehouse used for cross-database analysis and data mining; in particular to determine whether or not weekly database refreshes to the (remotely managed) warehouse were successful.

Recent research has developed new tools for exploring the contents of complex databases. In particular, database profiling has been developed both in research efforts and as commercial offerings. These tools provide a wealth of information about the structure and contents of a snapshot of a database. However, an important dimension of information is missing the database dynamics. That is, what parts of the database are changing, and in what way are they changing.

Data warehouse maintainers and users typically do not have access to query logs of production databases, which can provide direct evidence of database dynamics. Maintaining historic snapshots of database contents in a data warehouse is often not feasible in a large federated warehouse, and analyzing large database snapshots for differences can be prohibitively expensive. A data warehouse maintainer might perform periodic database profiling on the data warehouse contents, to provide metadata to the warehouse users and to screen for data quality problems. Historic database snapshot profiles provide an excellent data source for data mining to discover database dynamics. Being small but informative summaries of database contents, they are easily stored and mined even on inexpensive equipment, and can reveal trends in databases changes.

In this paper, we explore possibilities of mining database snapshot profiles. Because of the novel nature of the data set, our analyses use deliberately simple exploratory techniques. Nevertheless, we are able to extract from our sample databases a significant amount of reverse engineering and data quality information, including:

- Baselines for changes to a database.
- Identification of hot tables.
- Classification of changes to tables by a variety of criteria, including rate of change, insertions vs. updates, and convergent versus divergent change.
- Identification of hot fields within a table.
- Identification of suspicious changes to a field.
- Identification of tables with correlated updates.
- Identification of suspicious updates to a table.

Because of the wealth of information which can be extracted for free from a database profiling system, the Bellman developers are using the results of this study to add new browsing and alerting functionality to the Bellman system.

5.1 Future Work

A great deal of future work is possible; in particular when specific domain knowledge is available. Our particular interest is to make a deeper exploration of the meaning of what we can discover. For example, can we correlate the anomalous events we found in Section 4.3 with particular data quality problems. Conversely, if we identify data quality problems, can we correlate them to events observed in the database change data?

Our access to this kind of domain-specific information was very limited (and formed part of the motivation for this research). We hope that the results of this research will motivate the development of database change detection systems which, when put into production use by subject matter experts, can yield more detailed experimental results.

References

[1] G. Antoshenkov. Byte-aligned data compression. U.S. Patent number 5,363,098.

[1] Ascential. ibm.ascential.com

[2] BANKS: Browsing and Keyword Searching in Relational Databases. B Aditya et al. VLDB, 2002.

[3] DBXplorer: A System for Keyword-Based Search over Relational Databases. S. Agrawal, S. Chaudhuri, and G. Das. ICDE 2002. pg 5-16.

[4] Information-Theoretic tools for mining database structure from large data sets. P Andritsos, RJ Miller, and P Tsaparas, SIGMOD 2004.

[5] ObjectRank: Authority-Based Keyword Search in Databases. A Balmin, V Hristidis, Y Papakonstantinou. VLDB 2004.

[6] A feasibility study and performance study of dependency inference. D. Bitton, J. Millman, S. Torgerson, Proc. ICDE 1989 pg. 635-641.

[7] Effective change detection using sampling. J. Cho and A. Ntoulas. VLDB 2002.

[8] A Fast Regular Expression Indexing Engine. J. Cho and S. Rajagopalan. ICDE 2002.

[9] An Improved Data Stream Summary: The Count-Min Sketch and Its Application. G. Cormode and S. Muthukrishnan. LATIN 2004, pg. 29-38.

[10] Mining database structure; or, how to build a data quality browser. T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. SIGMOD 2002, 240-251.

[11] Grouping multivariate time series: A case study. T. Dasu, D. F. Swayne and D. Poole. Intl. Conf. Data Mining, Workshop on Temporal Data Mining 2005.

[12] Reconciling schemas of disparate data sources: A machine learning approach. A. Doan, P. Domingos, and A. Levy. SIGMOD 2001, pg. 509-520.

[13] Evoke Software. Data profiling and mapping, the essential first step in data migration and integration projects. http://www.evoke.com/pdf/wtpprDPM.pdf, 2000.

[14] A Framework for Measuring Changes in Data Characteristics. V. Ganti, J. Gehrke, and R. Ramakrishnan. PODS 1999 pg 126-137

[15] Comparing Massive High-Dimensional Data Sets. T. Johnson, T. Dasu. KDD 1998, pg. 229-233.

[16] Approximate Joins: Concepts and Techniques. N. Koudas, D. Srivastava. VLDB 2005.

[17] SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. W.-S. Li, C. Clifton. Data and Knowledge Engineering 33(1):49-84, 2000.

[18] Schema mapping as query discovery. M. Hernandez, R. Miller, and L. Haas. SIGMOD 2001, pg. 607.

[19] Indexing Text Data under Space Constraints. B Hore, H Hacigumus, B Iyer, S Mehrotra CIKM 2004.

[20] Efficient discovery of functional dependencies and approximate dependencies using partitions. Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Proc. ICDE 1998, pg. 363-372.

[21] Efficient IR-Style Keyword Search over Relational Databases. V Hristidis, L Gravano, Y Papakonstantinou VLDB, 2003.

[22] DISCOVER: Keyword Search in Relational Databases. V Hristidis, Y Papakonstantinou. VLDB, 2002.

[23] Modeling and Managening Content Changes in Text Databases, P. Ipeirotis, A. Ntoulas, J. Cho, L. Gravano. ICDE 2006, pg 606-617.

[24] Detecting Change in Data Streams. D. Kifer, S. Ben-David, and J. Gehrke. VLDB 2004, pg. 180-191.

[25] Checks and Balances: Monitoring Data Quality Problems in Network Traffic Databases. F. Korn, S. Muthukrishnan, Y. Zhu. VLDB 2003 536-547.

[26] Mining Deviants in a Time Series Database. H.V. Jagadish, N. Koudas, S. Muthukrishnan, VLDB 1999 102-112.

[27] Outliers in process modeling and identification. Pearson, R.K. IEEE Transactions on Control Systems Technology 10(1):66-63 , Jan 2002.

[28] Practical query analyzer. http://pgfoundry.org/projects/pqa/.

[29] Semantic and schematic similarities between database objects: A context-based approach. V. Kashyap and A. Seth, VLDB Journal 5(4):276-304, 1996.

[30] Issues and approaches of database integration. C. Parent and S. Spaccepietra. CACM 41(5):166-178, 1998.

[31] Efficient search of string partial determinations. S. Kramer and B. Pfahringer. Proc. KDD 1996 pg. 371-378.

[32] Potters Wheel: An interactive data cleaning system. VLDB 2001.

[33] Bottom-up induction of functional dependencies from relations. I. Savnik and P. Flach, Proc. AAAI Knowledge Discovery in Databases, 1993 pg. 174-185.

[34] Similarity Systems. www.similaritysystems.com

[35] Efficiently inducing determinations: A complete and systematic search algorithm that uses pruning. J. Schlimmer. Proc. AAAI Knowledge Discovery in Databases 1993. pg. 284-290.

Data Debugger: An Operator-Centric Approach for Data Quality Solutions

Surajit Chaudhuri Venkatesh Ganti Raghav Kaushik
Microsoft Research
{surajitc,vganti,skaushi}@microsoft.com

1 Introduction

Data cleaning is an essential step in populating and maintaining data warehouses. Owing to likely differences in conventions between the external sources and the target data warehouse as well as due to a variety of errors, data from external sources may not conform to the standards and requirements at the data warehouse. Therefore, data has to be transformed and cleaned before it is loaded into the warehouse so that downstream data analysis is reliable and accurate. This is usually accomplished through an Extract-Transform-Load (ETL) process.

Typical data cleaning tasks include record matching [10, 12, 6, 9, 14, 16], deduplication [16], and column segmentation [5, 1] which often go beyond traditional relational operators. This has led to development of utilities that support data transformation and cleaning. Such software falls into two broad categories. The first category consists of verticals such as Trillium [20] that provide data cleaning functionality for specific domains, e.g., addresses. By design, these are not generic and hence cannot be applied to other domains. The other category of software is that of ETL tools such as Microsoft SQL Server Integration Services (SSIS) [22] and IBM Websphere Information Integration [23] that can be characterized as "horizontal" *platforms* that are applicable across a variety of domains. These platforms provide a suite of operators including relational operators such as `select, project` and `equi-join`. A common feature across these frameworks is *extensibility*—applications can plug in their own custom operators. A data transformation and cleaning solution is built by composing these (default and custom) operators to obtain an operator tree or a graph. This extensible operator-centric approach is also adopted in research initiatives such as Ajax [13] and Morpheus [11].

While the second category of software can in principle support arbitrarily complex logic by virtue of being extensible, it has the obvious limitation that most of the data cleaning logic potentially needs to be incorporated as custom code since creating optimized custom code for data cleaning software is nontrivial. It would be desirable to extend its repertoire of "built-in" operators beyond traditional relational operators with a few core data cleaning operators such that with very less extra code, we can obtain a *rich variety* of data cleaning solutions.

In our Data Debugger project, we seek to achieve the above goal. Thus, we aspire to identify key primitive data cleaning operators and then ensure their efficient implementation on horizontal ETL engines such as SSIS. Thus, we adopt the approach of developing a domain-neutral framework of generic data cleaning operators. We believe that decomposing a data cleaning solution into simpler well-defined operators makes it easier to compose data cleaning operators with each other and with other (relational and non-relational) operators. In particular, it will be possible to easily customize and to analyze the overall data cleaning solution.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

While this is the long-term goal of our project, we illustrate our approach with the record matching operation in this paper. Section 2 goes over prior approaches to this problem and we examine our solution in Section 3. We conclude in Section 4 with a brief discussion of next steps.

2 Record Matching: An Overview

The *record matching* problem forms the basis for record linkage (e.g. [10, 12, 6, 9, 14]) and identifying approximately duplicate entities in databases (e.g., [16]). This is a very important problem in data cleaning. We illustrate our approach for developing data cleaning solutions using this problem.

Record matching is the operation of joining "similar" data. For example, consider a sales data warehouse. Owing to likely errors in the data such as typing mistakes, differences in conventions, product names and customer names in sales records may not match exactly with records in the master product catalog and reference customer registration tables, respectively. For example, two records "[Microsoft Corp.,, Redmond, WA, 98052]" and "[Microsoft Corporation, One Microsoft Way, Redmond, WA, 98052]" are not equal even though they identify the same organization.

The record matching problem has been considered in several domains such as web informatics [3, 15] and genome sequencing [17]. At the core of this problem, across all domains, is the concept of a measure that can be used to compare records, called a *similarity function*. The similarity function returns a value between 0 and 1, a higher value indicating that the two records are closer to being the same. A value of 1 indicates exact equality. A similarity join between two relations R and S returns all pairs of records, one each from R and S, such that the similarity function when applied to them is greater than a threshold [12, 16, 10]. By setting this threshold to a "high" value, we find all pairs that we can consider to be matches. Examples of similarity functions include edit distance, cosine similarity, Jaccard similarity and the recently proposed generalized edit distance [6]. Recent work on record matching [2] has also proposed the use of information beyond the text of records in order to determine whether they are duplicates. An example of this use of contextual information is to determine whether two state names (say "Washington" and "WA") are the same by checking similarity of the co-occurring sets of cities present in these states. Note that these notions of similarity apply independent of the textual content in the records. We informally use the term "co-occurrence similarity function" to apply to these notions of similarity.

Each of the above similarity functions has a unique characteristic and it is well-known that no single similarity function is universally applicable across all domains and scenarios [18]. For example, the characteristics of an effective similarity function for matching products based on their part names, where the errors are usually spelling errors, would be different from those matching street addresses because even small differences in the street numbers such as "148th Ave" and "147th Ave" are crucial, which would be different from similarity functions for matching person names based on their sounds. In fact, techniques based on machine learning for combining several similarity functions have been shown to be effective and is an active area of research [18, 21, 4, 19].

A general purpose data cleaning platform has to efficiently support this whole variety of notions of similarity. A naive implementation is to apply the similarity predicate after the cross product between the input relations. Obviously, this approach is prohibitively expensive when the input relations are large.

Several prior techniques provide efficient implementations for specific similarity functions [14, 9]. Naturally this option is tedious and it does not enable us to get to a more general solution. Alternatively, *blocking* heuristics have been proposed as a replacement for cross products. The idea is to first "group" potential candidate duplicates together and then apply similarity joins among small groups of candidates. Candidate groups are identified by grouping all records which match exactly on "approximate key" attributes. These approximate key attributes may either be user-specified [16] or be artificially generated via heuristic signature generation schemes [20]. These techniques are heuristic in nature. They do not necessarily guarantee that all pairs of records which satisfy the similarity predicate will be returned. Thus, the independence between the operator specification and the implementation is lost.

We are left with the challenging option of supporting a foundational primitive which can be used as a building block to implement a broad variety of notions of similarity. Our approach to record matching is driven by the desire to support a wide variety of standard similarity functions like Jaccard similarity with (IDF) weights on tokens, edit similarity. We also consider the generalized edit similarity function [6], which builds upon edit distance and jaccard containment measures with (IDF) weights on tokens, and has been shown to provide better matches in several scenarios. More importantly, we want to support all of these functions using a uniform abstraction so that we may then focus on efficient support for this abstraction. We achieve this using the SSJoin operator described in the next section. Our second goal that distinguishes us from the above approaches based on blocking is to separate, as in the case of relational operators, the operator specification from its implementation. Our implementation of similarity join returns all results required to be returned by its specification.

3 Record Matching Using the SSJoin Operator

As noted in Section 2, our goal is to efficiently support a foundational primitive which can be used as a building block to implement a broad variety of notions of similarity. The *SSJoin* primitive proposed in [7] and discussed below enables such a general implementation. The idea is to model strings as sets and perform a set-similarity join over these sets (hence the name S(et)S(imilarity)Join). In this way, most of the the effort in a similarity join can be done by the SSJoin operator. The benefit of an efficient implementation of the SSJoin operator simultaneously translates to a variety of similarity functions.

OrgName	3-gram	Norm
Microsoft Corp	mic	12
Microsoft Corp	icr	12
Microsoft Corp	cro	12
...	...	...
Microsoft Corp	cor	12
Microsoft Corp	orp	12

R

OrgName	3-gram	Norm
Mcrosoft Corp	mcr	11
Mcrosoft Corp	cro	11
Mcrosoft Corp	ros	11
...	...	...
Mcrosoft Corp	cor	11
Mcrosoft Corp	orp	11

S

Figure 1: Example sets from strings

We now describe the SSJoin operator. The operator works with sets and intuitively finds all pairs of sets that have a high overlap. We refer to the size of the intersection between two sets s_1, s_2 to be their overlap similarity, denoted $Overlap(s_1, s_2)$. While our discussion below focuses on unweighted sets, the discussion extends to weighted sets as in [7].

We assume that sets are represented in First Normal Form by storing every element in a separate record. Consider relations $R(A, B)$ and $S(A, B)$ where A and B are subsets of columns. Each distinct value $a_r \in R.A$ defines a group, which is the subset of tuples in R where $R.A = a_r$. Call this set of tuples $Set(a_r)$. Similarly, each distinct value $a_s \in S.A$ defines a set $Set(a_s)$. The simplest form of the SSJoin operator joins a pair of distinct values $\langle a_r, a_s \rangle$, $a_r \in R.A$ and $a_s \in S.A$, if the projections on column B of the sets $Set(a_r)$ and $Set(a_s)$ have a high overlap. The formal predicate is $Overlap(\pi_B(Set(a_r), \pi_B(Set(a_s))) \geq \alpha$ for some threshold α. We denote $Overlap(\pi_B(Set(a_r), \pi_B(Set(a_s)))$ as $Overlap_B(a_r, a_s)$. Hence, the formal predicate is $Overlap_B(a_r, a_s) \geq \alpha$. We illustrate this through an example.

Example 1: *Let relation $R(OrgName, 3\text{-}gram)$ and $S(OrgName, 3\text{-}gram)$ shown in Figure 1 associate the strings "Microsoft Corp" and "Mcrosoft Corp" with their 3-grams. Denoting* OrgName *by A and* 3-gram *by B, the SSJoin operator with the predicate $Overlap_B(a_r, a_s) \geq 10$ returns the pair of strings ⟨"Microsoft Corp", "Mcrosoft Corp"⟩ since the overlap between the corresponding sets of 3-grams is 10.*

In general, we may wish to express conditions such as: the overlap similarity between the two sets must be 80% of the set size. Thus, in the above example, we may wish to assert that the overlap similarity must be higher than 80% of the number of 3-grams in the string "Microsoft Corp". We may also wish to be able to assert that the overlap similarity be higher than say 80% of the sizes of *both* sets. We now formally define the SSJoin operator as follows, which addresses these requirements.

Definition 1: Consider relations $R(A, B)$ and $S(A, B)$. Let $pred$ be the predicate $\bigwedge_i Overlap_B(a_r, a_s) \geq \ e_i$, where each e_i is an expression involving only constants and columns from either $R.A$ or $S.A$ (but not both). We write R SSJoin_A^{pred} S to denote the following result: $\{\langle a_r, a_s\rangle \in R.A \times S.A | pred(a_r, a_s)$ is true $\}$. We also denote $pred$ as $\{Overlap_B(a_r, a_s) \geq e_i\}$.

We illustrate this through the following examples based on Figure 1, where the third column *Norm* denotes the length of the string. In general, the *norm* could denote either the length of the string, or the cardinality of the set, or the sum of the weights of all elements in the set. Several similarity functions use the norm to normalize the similarity.

Example 2: *As shown in Figure 1, let relations $R(OrgName, 3\text{-}gram, Norm)$ and $S(OrgName, 3, Norm)$ associate the organization names with (1) all 3-grams in each organization name, and (2) the number of 3-grams for each name. The predicate in the SSJoin operator can be used to capture different notions of similarity by varying the predicate specification.*

- *Absolute overlap: $Overlap_B(a_r, a_s) \geq 10$ joins the pair of strings $\langle$"Microsoft Corp", "Mcrosoft Corp"$\rangle$ since the overlap between the corresponding sets of 3-grams is 10.*
- *1-sided normalized overlap: $Overlap_B(\langle a, norm\rangle_r, \langle a, norm\rangle_s) \geq 0.8 \cdot R.norm$ joins the pair of strings $\langle$"Microsoft Corp", "Mcrosoft Corp"$\rangle$ since the overlap between the corresponding sets of 3-grams is 10, which is more than 80% of 12.*
- *2-sided normalized overlap: $Overlap_B(\langle a, norm\rangle_r, \langle a, norm\rangle_s) \geq \{0.8 \cdot R.norm, 0.8 \cdot S.norm\}$ also returns the pair of strings $\langle$"Microsoft Corp", "Mcrosoft Corp"$\rangle$ since 10 is more than 80% of 12 and 80% of 11.*

3.1 Exploiting SSJoin for Similarity Joins

First, note that the above definition of SSJoin can be directly used to capture notions of similarity based on co-occurrence [2] where it has been shown to be very effective for identifying approximate duplicates. We illustrate with an example below.

Example 3: *Suppose we have two tables, say from different sources that are being integrated, of author names joined with the titles of the papers, say with the schema* <PTITLE, ANAME>. *Since we want a unified view of all authors, we are interested in identifying author names that are likely to represent the same author. Now, if the naming conventions in the two sources are entirely different, it is quite likely that the textual similarity between the author names is only a partial indicator of their similarity. We are forced to rely on alternative sources of information for identifying duplicate author entities. In this instance, we can use the set of paper titles associated with each author to identify authors. The idea is that if two authors are the same, then the set of paper titles co-occurring with them must have a large overlap.*

The notion of set overlap can also be used to capture various string similarity functions—edit distance, jaccard similarity and generalized edit similarity as shown in [7]. The way the SSJoin operator is used for string similarities is outlined in Figure 2. Let Rbase(A) and Sbase(A) be relations where A is a string-valued attribute. The goal is to find pairs $\langle$Rbase$.A$, Sbase$.A\rangle$ where the string similarity, according to a specific similarity function, is above a threshold α. We first convert the strings Rbase(A) and Sbase(A) to sets, construct normalized representations $R(A, B, norm(A))$ and $S(A, B, norm(A))$, and then perform an SSJoin between the normalized representations. The SSJoin predicate, especially the overlap threshold α', is chosen so that all pairs whose

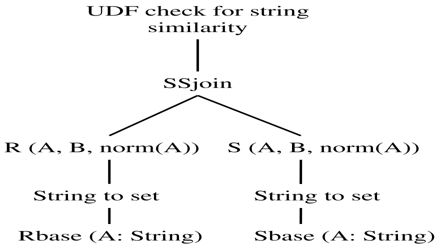


Figure 2: String Similarity Join using SSJoin

string similarity is greater than α are *guaranteed* to be in the result of the SSJoin. Hence, the SSJoin operator provides a way to efficiently produce a small superset of the correct answer. We then compare the pairs of strings using the actual similarity function, declared as a UDF within a database system, to ensure that we only return pairs of strings whose similarity is above α.

Note that a direct implementation of the UDF within a database system without SSJoin is most likely to lead to a cross-product where the UDF is evaluated for all pairs of tuples. Hence, this is an impractical approach. On the other hand, an implementation using SSJoin exploits the support within database systems for equi-joins to result in a significant reduction in the total number of string comparisons. This results in orders of magnitude improvement in performance [7]. Ajax [13] proposed a matching primitive based on a user-specified similarity function. First, this focuses purely on record level similarity and hence cannot capture co-occurrence similarity functions. Further, they do not identify any common features across all similarity functions they support. Hence, they run the risk that their implementation would reduce to a cross-product followed by a UDF check and hence would also be impractical.

We now illustrate how similarity join with respect to edit distance can be performed using SSJoin. The edit distance between strings is the least number of edit operations (insertion and deletion of characters, and substitution of a character with another) required to transform one string to the other. For example, the edit distance between strings 'microsoft' and 'mcrosoft' is 1, the number of edits (deleting 'i') required to match the second string with the first.

We illustrate the connection between edit distance and overlap through the following example.

Example 4: *Consider the strings "Microsoft Corp" and "Mcrosoft Corp". The edit distance between the two is 1 (deleting 'i'). The overlap similarity between their 3-grams is 10, more than 80% of the number of 3-grams in either string.*

The intuition is all q-grams that are "far away" from the place where the edits take place must be identical. Hence, if the edit distance is small, then the overlap on q-grams must be high. This intuitive relationship between edit distance and the set of q-grams can be formalized as follows [14].

Property 2: [14] Consider strings σ_1 and σ_2, of lengths $|\sigma_1|$ and $|\sigma_2|$, respectively. Let $QGSet_q(\sigma)$ denote the set of all contiguous q-grams of the string σ. If σ_1 and σ_2 are within an edit distance of ϵ, then $Overlap(QGSet_q(\sigma_1), QGSet_q(\sigma_2)) \geq \max(|\sigma_1|, |\sigma_2|) - q + 1 - \epsilon \cdot q$

Thus, in the above example, the edit distance is 1, and Property 2 asserts that at least 9 of all the 3-grams from the two strings have to be common. Based on this property, we can implement similarity join based on edit distance using SSJoin, as detailed in [7].

3.2 Top-K Similarity Join

Consider the example where we want to match a product name in a sales transaction reported by a reseller against the standardized set of products in a product reference table. In this context, we know that the incoming products have to match with those in the reference table. If there is no record in the reference table which does not match exactly with an input product name, we want to match it with the "best matching" record in the reference table. This operation has been shown to be very effective in dealing with input errors [6, 20]. The intuitive notion of match quality can again be quantified using a similarity function and thus this operation is like the similarity join between the input relation and a reference table. However, unlike the similarity join where a user specifies the threshold, we want the most similar record or in general the K most similar records to each record in the input relation.

Observe that we stipulate one of the two relations involved as a *reference table*, and for each record in the other relation we want the K most similar records from the reference table. Thus, this operation is inherently asymmetric unlike the similarity join, which can be symmetric if the similarity function is. As with the similarity join operator, we ideally want to be able to support the top-K similarity join operator over many similarity functions discussed earlier. Currently, we support an index-based implementation (with probabilistic guarantees) of this operation for the generalized edit similarity in SQL Server Integration Services 2005. We call this specific implementation the *fuzzy lookup* transform. Other details of this implementation are discussed in [6].

4 Conclusions

In this paper, we discussed the Data Debugger framework for data cleaning. The main goal is to identify generic and robust abstractions for data cleaning operators, and to support efficient implementations of these abstractions within the platform. We are now able to compose these operators with other relational and non-relational operators to derive operator trees.

We illustrated the Data Debugger framework for the specific problem of record matching, which is only one important operation in data cleaning. Deduplication (also called entity resolution) [16, 8], which partitions a relation into groups of records based on their pairwise similarities, is also an important data cleaning operation. Another operation is one which takes an attribute value and segments it into constituent attribute values [5, 1]. Commercial address cleansing tools (e.g., Trillium) rely heavily on this operation, which they call "parsing". We want to extend the Data Debugger framework to these other important data cleaning operations.

References

[1] E. Agichtein and V. Ganti. Mining reference tables for automatic text segmentation. In *Proceedings of ACM SIGKDD*, 2004.

[2] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of VLDB*, 2002.

[3] K. Bharat and A. Z. Broder. Mirror, mirror on the web. *Computer Networks*, 1999.

[4] M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of ACM SIGKDD*, 2003.

[5] V. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *Proceedings of ACM SIGMOD*, 2001.

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

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

[8] S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *Proceedings of ICDE*, 2005.

[9] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of ACM SIGMOD*, 1998.

[10] W. W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on information systems*, 2000.

[11] T. Dohzen, M. Pamuk, S.-W. Seong, J. Hammer, and M. Stonebraker. Data integration through transform reuse in the morpheus project. In *Proceedings of ACM SIGMOD*, 2006.

[12] I. P. Felligi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Society*, 1969.

[13] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C. Saita. Declarative data cleaning: Language, model, and algorithms. In *Proceedings of VLDB*, 2001.

[14] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *Proceedings of VLDB*, 2001.

[15] T. H. Haveliwala, A. Gionis, and P. Indyk. Scalable techniques for clustering the web. In *Proc. of the 3rd Intl. Workshop on Web and Databases*, 2000.

[16] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *Proceedings of ACM SIGMOD*, 1995.

[17] M. Narayanan and R. M. Karp. Gapped local similarity search with provable guarantees. In *Proc. of the 4th Intl. Workshop on Algorithms in Bioinformatics*, 2004.

[18] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of ACM SIGKDD*, 2002.

[19] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *Proceedings of ACM SIGMOD*, 2004.

[20] Trillium Software. www.trilliumsoft.com.

[21] S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information Systems*, 2001.

[22] Microsoft SQL Server 2005 integration services. http://msdn.microsoft.com/sql.

[23] IBM Websphere Information Integration. http://ibm.ascential.com.

Structure-Aware XML Object Identification

Diego Milano*, Monica Scannapieco*† and Tiziana Catarci*

* Università degli Studi di Roma "La Sapienza"
Dipartimento di Informatica e Sistemistica
Via Salaria 113, 00198 Roma
{milano, monscan, catarci}@dis.uniroma1.it

† Istituto Nazionale di Statistica
Via Cesare Balbo 6, 00184 Roma
scannapi@istat.it

Abstract

The object identification problem is particularly difficult for XML data, due to its structural flexibility. Tree edit distances have been used to perform approximate comparisons among XML trees. However, such distances ignore the semantics implicit in element labels and nesting relationships of XML data. Furthermore, the use of tree edit distances for unordered trees, that would be more suitable for this task, is computationally infeasible. In this paper, we define a new distance for XML data, the structure aware XML distance*, that overcomes these issues, and present a polynomial algorithm to calculate it.*

1 Introduction

The *object identification problem* is a central problem arising in data cleaning and data integration, where different objects must be compared to determine if they refer to the same *real-world entity*, even in the presence of errors such as misspellings.

As the spread of the XML format as a data model increases, the need to develop effective strategies for XML object identification grows. As a data model, XML is half a way between completely semistructured models, in which nothing is known in advance about the structure of the data, and structured ones, like the relational model. XML documents often represent complex, nested data. However, they are usually required to conform to some kind of structural specification, expressed in schema languages like DTD and XML Schema. Hence, their structure, though flexible, usually exhibits a certain degree of regularity.

Flexibility is one of the major issues in XML object identification. XML data representations may allow for optional values, and lists of values whose length is not known schema-wise. Functions for approximate XML data comparisons must thus be able to cope both with errors at the level of textual data values and with this structural flexibility. The hierarchical nature of XML data has lead to the use of *tree edit distances*([1]) to compare XML documents for various purposes, like approximate DTD matching and detection of differences in versions of XML documents ([3]). Some proposals also address the object identification problem ([4]).

Tree edit distances in their original form give great importance to topological features of trees, but are not well suited when node labels and their nesting have semantics and data structure is somewhat regular. The proposals cited above adapt or extend tree edit distances to handle XML documents, but ignore this problem.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Another issue is that, due to the infeasibility of tree edit distance measures for unordered trees([18]), such proposals are usually based on versions of tree edit distance for ordered trees. Notice that, while the XML data model is indeed ordered, the presence of unbounded lists of values and optional elements in the data motivates for the adoption of unordered comparisons when looking for approximate matches. As an example, consider an element defined by the following DTD element definition:

```
<!ELEMENT SHOP (NAME, ADDRESS?, PHONENUM*)>
```

Here, an object representing a shop may contain zero or more phone numbers. The order in which phone numbers are listed is irrelevant, or however unspecified, so two objects representing the same shop might contain the same set of phone numbers in different order. Requiring that elements correspond to each other in an ordered way may lead to miss some of the similarities among those objects. We believe that unordered comparison are much more suited than ordered comparisons to perform approximate matches in data-oriented XML.

We are currently investigating how to overcome these issues. This paper presents the first results of our ongoing research. In particular, we propose a novel distance measure for XML data, the *structure aware XML distance*, and present an algorithm to compute it. This distance copes with the flexibility which is usual for XML documents, but takes into proper account the semantics implicit in structural information. It allows for comparison of XML data as unordered tree, and is thus particularly suited to identify objects that may differ also for the order of the data values they contain. Nonetheless, differently from other distances for unordered trees, it can be computed in polynomial time. The structure aware XML distance can be used as the basis for approximate comparisons in an XML object identification framework.

The rest of this paper is organized as follows. In Section 2 we review some related work. In Section 3 we first motivate the introduction of a new distance, showing with examples how approaches based on classical tree edit distance fail to respect the semantics of XML data. We then define formally the structure aware XML distance. In Section 4 we present an algorithm to calculate the distance on two XML trees and review its time complexity. In Section 5 we draw some conclusions, and describe some issues we plan to consider in our future work.

2 Related Work

The *object identification* problem has been extensively studied for relational data (with the name of record matching or record linkage problem), but the correspondent problem for semi-structured data has only recently drawn some attention. Most proposals for XML object identification are *structure oblivious*, in the sense that they rely on some kind of flattening of document structure to perform comparisons. In [16], XML objects are flattened and compared using string comparison functions. In the DOGMATIX framework([15]), data is extracted from an XML document and stored in relations called *object descriptions*. Tuples of two object descriptors containing data with the same XPath are classified as similar or contradictory using string edit distance, and object descriptions similarity is assessed taking into account the number of similar and contradictory tuples. The approach in [13] is similar, but objects have, beside flat object descriptors, also nested objects called *descendants*. Comparisons among objects are performed level by level in a bottom up fashion, and approximate similarity results at lower-levels are considered when comparing objects at upper levels. In particular, the similarity of two objects is influenced by how many of their descendants are supposed to be similar. Notice that, in contrast to this approach, the distance we define in this paper takes into account the entire structure of an XML tree at once. *Structure aware* approaches proposed for XML object identification rely on distance measures based on the tree structure of XML, like *tree edit distances* (see [1] and below in this section). In particular, the authors of [4] integrate string comparison functions into the classic tree edit distance for ordered trees to compute approximate joins on XML documents. The paper is mainly concerned with heuristics to filter unneeded comparisons, based on efficient computation of bounds for the distance.

It is worthwhile to notice that tree edit distances have been used to measure similarity of XML documents also in proposals not directly related to object identification. As an example, in [3] the authors use a version of tree edit distance that allows moving entire subtrees, in order to detect changes in hierarchically structured documents. In [9], an analogous version of edit distance is used to cluster XML documents by similarity.

The notion of *tree edit distance* for ordered trees has been introduced in its most widely known form by Tai ([14]), though a restricted version already appeared in [11]. Since then, the problem has been extended to unordered trees ([18, 12]) and many variations have been proposed (see e.g. [5, 17, 10]). Most versions of the edit distance problem allow polynomial-time algorithms in the case of ordered trees, but become NP-hard in the unordered case([18]). The *tree alignment distance*([5]) is a restricted version of edit distance. In tree alignment, trees are first made isomorphic (ignoring node labels) with the *insertion* of nodes labelled with *spaces*, and then overlayed. A cost function is defined on pairs of labels and the cost of an alignment is the sum of the costs of opposing labels. An *optimal alignment* is an alignment of minimum cost. Differently from the distance we propose in this paper, tree alignment considers insertions of nodes and overlays nodes with different labels. The alignment problem has polynomial cost for ordered trees, but becomes NP-Hard for unordered trees. In [10] a *structure respecting edit distance* is presented. Despite the similarity with the name of the distance introduced here, the proposal in [10] is a tree edit distance, with the added constraint that disjoint subtrees are mapped to disjoint subtrees, and does not take into account the semantics of structure, as in our case. We refer to [1] the reader interested in a survey on various versions of tree edit distances.

3 A Structure-Aware Approach to XML Object Identification

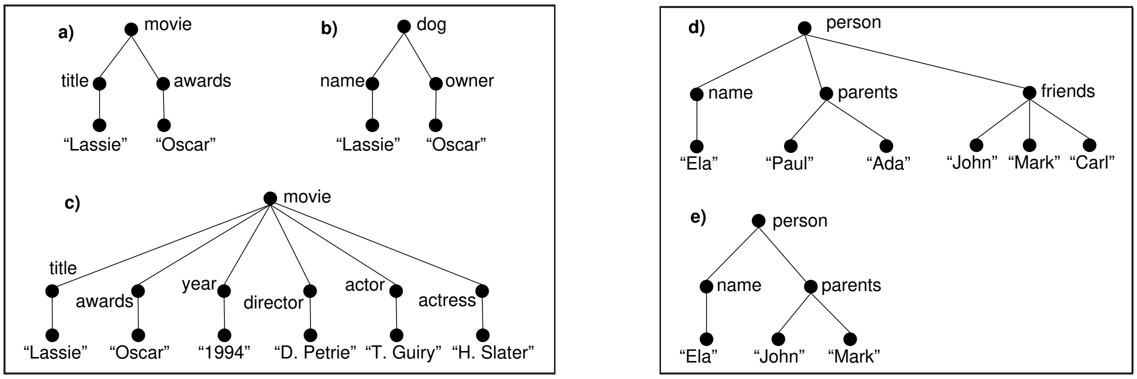
Approaches to solve the object identification problem generally make use of some kind of distance function to detect the similarity of two objects. In record matching techniques proposed for the relational model, attribute values are often compared using *string comparison functions* ([8, 6]). XML documents can be modelled as node labelled trees. This hierarchical, tree-like nature justifies the proposal of similarity measures that integrate string comparison functions with *tree edit distances* ([1]). Tree distances have been introduced to measure structural similarity among trees. However, they are not fully able to capture the semantics of XML data, as they do not keep into account the semantics and structural relationships among XML elements.

In this section, we first show some weaknesses that classic tree edit distances suffer when used to compare XML data, and then define a new notion of distance for XML data, the *structure-aware XML distance*, as the basis of an approach to XML object identification.

3.1 Tree Distances

Given a set of edit operations on labelled trees (i.e. node insertions, deletions and relabellings) and a function that assigns a cost to each operation, the *tree edit distance* between two trees is defined ([14]) as the minimum cost sequence of tree edit operations required to transform one tree to another. Other variants of this notion have been proposed in the literature, including versions where edit operations are allowed on entire subtrees.

Comparison of XML data based on tree distances has been proposed for various purposes ([4, 3, 9]). As an example, in [4] the authors adapt the tree distance defined above to perform approximate XML joins by adding comparisons of text node labels based on a string comparison function. With respect to other proposals for XML object identification, this approach has the advantage of keeping into account the tree structure of XML data. However, the use of tree edit distance for this purpose has some drawbacks. The following examples illustrate two of them. First, consider the XML data trees represented in Figure 1(a). Tree **c)** represents the same data as tree **a)**, and also contains some additional information. Tree **b)**, instead, represents different data. However, as it can be easily verified, the tree distance between **a)** and **c)** is greater than the distance between **a)** and **b)**. Let us now consider the example shown in Figure 1(b). Here, a person can be represented with its



(a) (b)

Figure 1: Two issues in classical tree edit distance-based XML comparisons

parents and an optional list of friends. When measuring the distance between the two trees **d)** and **e)**, a minimal distance is obtained by deleting from tree **d)** the entire *parent* subtree, relabelling node *friends* into *parents* and matching its leaves to two of the nodes of the *parent* subtree of tree **e)**. This behaviour clearly violates the semantics implicit in node labels. These problems, in addition to the need of performing unordered comparisons efficiently, motivate the introduction of a new distance measure for XML data.

3.2 XML Structure-aware Distance

In this section, we first give an intuitive description of our approach to distance measurement, and then we formalize the distance itself. Our aim is to overcome the problems highlighted in the previous section by taking into full account the presence of structural information in XML data.

The above examples show that, when comparing XML trees, a good choice is to match subtrees that have similar structure and that are located under the same path from the root. These can be indeed interpreted as clues of the same semantics. If two trees have exactly the same structure, and only differ by the textual values present on the leaves, we can *overlay* the trees so that nodes with the same path match. When multiple overlays are possible, then we choose one such that the distance among textual values on the leaves is minimal. If the two trees have different structure, due to the presence of additional information, we can match those subtrees that exhibit the least distance on leaves, and ignore those that cannot be matched. Intuitively, we are trying to realize an overlay as above by deleting extra subtrees that do not match well. In the remainder of this section we formalize these intuitions, and use them to define a new distance for XML data.

An *overlay* O of two data trees T_1 and T_2 is a non-empty set of couples of nodes from T_1 and T_2 with the following properties. Let $v_i, v_i' \in T_i, n_i \in (T_i - leaves(T_i)), i = 1, 2$:

$$\begin{cases} \text{if } \langle v_1, v_2\rangle, \langle v_1', v_2'\rangle \in O, \text{ then } v_1 = v_1' \text{ iff } v_2 = v_2'; \\ \text{if } \langle v_1, v_2\rangle \in O, \text{ then } path(v_1) = path(v_2); \\ \langle n_1, n_2\rangle \in O \text{ iff } \exists v_1, v_2 \text{ s.t. } n_1 = parent(v_1) \wedge n_2 = parent(v_2) \wedge \langle v_1, v_2\rangle \in O. \end{cases}$$

Where, $path(v)$ denotes the sequence of node labels $label(root) \ldots label(v)$ encountered when traversing the tree from the root to node v. If $\langle v_1, v_2\rangle \in O$ we say that v and w *match*. If a node is not matched with any other node, we say that it is *deleted*. Informally, an overlay matches nodes from T_1 to nodes from T_2 one-to-one, so that nodes or leaves are matched only if they have the same path from the root. Two nodes can be matched iff they are ancestors of two leaves that are matched. Notice that this implies that, if a node is deleted, all its descendants are also deleted. It also implies that an overlay of two trees exists only if there exist two leaves $l_1 \in T_1$ and $l_2 \in T_2$ with the same path from the root. We say that two trees are *comparable* if they have at least one overlay. An overlay O of two trees is *complete* if there is no other overlay O' such that $O \subset O'$. In the rest of this paper, when we refer to an overlay, we implicitly assume that it is a complete overlay.

Let $sdist(s_1, s_2)$ be a string comparison function. If v is a leaf node, we denote with $text(v)$ its string value. The *cost of match* for two nodes v, w is:

$$\mu_{v,w} = \begin{cases} sdist(text(v), text(w)) & \text{if } v, w \text{ are leaves} \\ 0 & \text{otherwise} \end{cases}$$

The *cost of an overlay O* is defined as $\Gamma_O = \sum_{\langle v,w \rangle \in O} \mu_{v,w}$. An overlay O of two trees is *optimal* if it is complete and there is no other complete overlay O' such that $\Gamma_{O'} < \Gamma_O$.

The *structure aware XML distance* of two comparable XML trees T_1 and T_2 is defined as the cost of an optimal overlay of T_1 and T_2. In Section 4, we describe an algorithm that measures the structure aware XML distance among two trees. Differently from other tree-distances, the distance defined here can be computed in polynomial time, even if the trees are unordered. Therefore, this distance is suitable for object identification, which requires a high number of pairwise comparisons among trees.

Notice that, when applied to the trees in the example given above, this distance works as expected. Trees **a)** and **b)** in Figure 1(a) are incomparable, while the distance of trees **a)** and **c)** is zero. In the case of Figure 1(b), the distance only considers the differences among those leaves that is meaningful to compare, giving as a result the least distance between names present under the nodes *parents*.

4 Structure Aware XML Distance Measurement

In this section, we introduce an algorithm to measure the structure aware XML distance defined in Section 3. Before presenting it in details, we describe some properties of overlays that are useful to understand how the algorithm works. We denote with T_1, T_2 two comparable data trees, with r_1, r_2 their roots, and with $v_i, w_j, i \in [1, deg(r_1)], j \in [1, deg(r_2)]$ the children of r_1 and r_2, respectively. Furthermore, given a node v, we denote with $T(v)$ the tree rooted at v.

Let O be an overlay of T_1, T_2. It can be easily shown that if $\langle v, w \rangle \in O$, then the set $O_{v,w} = \{\langle y, z \rangle \in O | y \in T(v), z \in T(w)\}$ is an overlay of $T(v)$ and $T(w)$. Therefore, O can be written $O = \langle r_1, r_2 \rangle \cup (\bigcup_{\langle v_i, w_j \rangle \in O} O_{v_i,w_j})$. It follows that the cost of O is the sum of the costs of all the overlays O_{v_i,w_j}. It is also immediate to see that O is optimal only if the overlays O_{v_i,w_j} are all optimal.

A complete overlay of T_1 and T_2 can be obtained by first matching r_1 with r_2 and then matching a children of r_1 with a children of r_2 until no more matches are possible. Notice that nodes can be matched only if they have the same label. Nodes that are not matched are deleted, along with all their descendants. If v_i and w_j are matched, then an overlay is built for $T(v_i)$ and $T(v_j)$ by applying the same process, recursively, up to the leaves. From the above considerations, it follows that, in order to obtain an optimal overlay, the children of r_1 and r_2 must be matched so that the sum of the costs of optimal overlays for subtrees rooted at matched nodes is minimal. In other words, an algorithm must choose, among all possible assignments of subtrees rooted at the children of r_1 to subtrees rooted at the children of r_2, one that minimizes the sum of the distances of all couples of subtrees.

Algorithm 1 analyzes two comparable trees recursively, starting from the roots. If the roots are leaf nodes, a distance measure for their associated text values is returned. Such function is denoted by the procedure $compareStrings()$ in the algorithm. Otherwise, the algorithm considers their children, and computes a distance for each couple of subtrees rooted at children with the same label, recursively. After all distances have been calculated, the algorithm must assign each node to another node with the same label, minimizing the overall cost. This is an assignment problem and can be solved using a variation of the well-known Hungarian Algorithm ([7],[2]). In the algorithm, this task is performed by a call to procedure $findAssignment()$. In particular, given a matrix of distances, the procedure returns a set of assignments containing couples of indices of assigned

Algorithm 1 $StructureAwareXMLDist(T_1, T_2)$

```
if isLeaf(r_1) and isLeaf(r_2) then
  returncompareStrings(text(r_1), text(r_2))
else
  structDist := ∞
  for all l in labels(children(r1) ∪ children(r_2)) do
    for all v_i ∈ children_l(r_1) do
      for all w_j ∈ children_l(r_2) do
        children_lDistance[i, j] := StructureAwareXMLDist(T(v_i), T(w_j))
      end for
    end for
    assignment_l := findAssignment(children_lDistance[])
    for all ⟨h, k⟩ ∈ assignment_l do
      if structDist = ∞ then
        structDist := 0
      end if
      structDist := structDist + children_lDistance[h, k]
    end for
  end for
  return structDist
end if
```

nodes. For ease of presentation, in the algorithm we denote the set of all children of node v having label l with $children_l(v)$. Results of distance calculations for a certain set of children having label l are stored in an array named $children_lDistance$. The distance is initially set to ∞, and reset to 0 only in the case that there is at least one assignment of root children.

In order to understand the cost of the algorithm, let us consider a case in which all the leaves of the tree have the same path, and the data trees are complete. We consider distance calculation among two trees T_1 and T_2. We denote with deg_1 and deg_2 their respective degrees and with L_1, L_2 their sets of leaves.

Let T_1', T_2' be two subtrees of T_1 and T_2 rooted at level l, and let r_1', r_2' be their roots. In order to compute their distance, we must choose a match among the children of r_1' and r_2' such that the the sum of distances for corresponding subtrees is minimal. Assuming that we have already calculated all pairwise distances, we need to solve an instance of the linear assignment problem. The Hungarian algorithm gives a solution in cubic time, so the cost of an assignment is $O((deg_1 + deg_2)^3)$.

To compute all distance measurements, we proceed bottom up, starting from the leaves and calculating all pairwise distances among all nodes at each level. At level $depth - 1$, before performing the assignment phase we must compute distances among textual values. These are computed in constant time (w.r.t. the size of the trees). At upper levels, we already know the distances among nodes at lower levels, so we just need to perform the assignment phase. In total, the assignment phase is repeated $(|T_1| - |L_1|) \times (|T_2| - |L_2|)$ times. Thus, the overall cost is $O((|T_1| - |L_1|) \times (|T_2| - |L_2|) \times (deg_1 + deg_2)^3$

When there is more than one path for leaf nodes, the calculation is less expensive. For example, if the sets of children of two nodes are partitioned according to their label in two sets of equal cardinality s, in order to compute the distance among the two nodes the algorithm will have to calculate $(s^2 + s^2)$ distances among children, instead than $(s + s)^2$, and calculate two assignments at the cost of $O((2s)^3)$ instead of a single one at cost $O((4s)^6)$.

5 Conclusions

XML data has tree-like nature and flexible structure. These features have lead to proposals for XML object identification that exploit tree-edit distances to perform approximate comparisons among XML trees. However, tree edit distances suffer from certain drawbacks, since they ignore the semantics implicit in the element labels and nesting relationships. Furthermore, while tree-distances for unordered trees are better suited to perform approximate comparisons of XML data, their use is computationally infeasible. In this paper, we have defined a new distance for XML data, the *structure aware XML distance*, that overcomes these issues. The distance compares only portions of XML data trees whose structure suggest similar semantics. Furthermore, it performs comparison on unordered trees, without incurring in high computational costs. We have presented an algorithm to measure the distance between two trees, and discussed its complexity, that is polynomial.

In our future work, we will perform experiments to determine the effectiveness of our distance as the basis of an object identification approach. We also plan to investigate other interesting issues related to the use of tree distances for XML comparison. As an example, in [4] the authors suggest the use of ontology based techniques to evaluate the cost of relabelling element nodes. How to balance the effects of string- comparison-based and ontology- based cost evaluation seems far from trivial.

References

[1] Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3), 2005.

[2] Francois Bourgeois and Jean-Claude Lassalle. An extension of the munkres algorithm for the assignment problem to rectangular matrices. *Commun. ACM*, 14(12):802–804, 1971.

[3] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6*, 1996.

[4] Sudipto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu. Approximate xml joins. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6*, 2002.

[5] Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of trees - an alternative to tree edit. *Theor. Comput. Sci.*, 143(1):137–148, 1995.

[6] Nick Koudas and Divesh Srivastava. Approximate joins: Concepts and techniques. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2*, 2005.

[7] James Munkres. Algorithms for assignment and transportation problems. *SIAM Journal on Computing*, 5(1), March 1957.

[8] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[9] Andrew Nierman and H. V. Jagadish. Evaluating structural similarity in xml documents. In *Proceedings of the Fifth International Workshop on the Web and Databases, WebDB 2002, Madison, Wisconsin, USA, June 6-7, 2002, in conjunction with ACM PODS/SIGMOD 2002*, pages 61–66, 2002.

[10] Thorsten Richter. A new measure of the distance between ordered trees and its applications. Technical Report 85166-CS, 1997.

[11] Stanley M. Selkow. The tree-to-tree editing problem. *Inf. Process. Lett.*, 6(6):184–186, 1977.

[12] Dennis Shasha, Jason Tsong-Li Wang, Kaizhong Zhang, and Frank Y. Shih. Exact and approximate algorithms for unordered tree matching. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4), 1994.

[13] Felix Naumann Sven Puhlmann, Melanie Weis. Xml duplicate detection using sorted neighborhoods. In *Proceedings of EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31*. Springer Verlag, LNCS 3896, 2006.

[14] Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3), 1979.

[15] Melanie Weis and Felix Naumann. Dogmatix tracks down duplicates in xml. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*.

[16] Melanie Weis and Felix Naumann. Detecting duplicate objects in xml documents. In *IQIS 2004, International Workshop on Information Quality in Information Systems,Paris, France*, 2004.

[17] Kaizhong Zhang. A constrained edit distance between unordered labeled trees. *Algorithmica*, 15(3), 1996.

[18] Kaizhong Zhang, Richard Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Inf. Process. Lett.*, 42(3), 1992.

Call for Participation

VLDB2006
32nd International Conference on Very Large Data Bases
Very Large Data Bases

The 32nd International Conference on Very Large Data Bases

The Convention and Exhibition Center (COEX), Seoul, Korea
September 12-15, 2006
http://www.vldb2006.org/

VLDB is the premier international conference on database technology, organized every year by the VLDB Endowment. VLDB 2006 will be held in Seoul, the biggest city in Korea. A city with a population of 11 million, Seoul has been the capital city of Korea for nearly 600 years and has many historical relics and tourist resorts, attracting over 5,000,000 annual visitors.

The VLDB 2006 has assembled an excellent program that consists of **2 keynote speakers**, **83 research papers, 11 industrial papers, 29 demonstrations, 2 panels, 5 tutorials**, **12 co-located workshops, and the 10-year best paper talk.** For the detailed program, go to the main home page of VLDB 2006.

The registration for VLDB 2006 is now open at the VLDB 2006 home page. Early registration deadline for reduced rates is July 31st, 2006.

VLDB 2006 is sponsored by LG Electronics, Microsoft, IBM, Samsung Electronics, Oracle, SAP, SK Telecom, Google, Naver, HP, Samsung SDS, Asian Office of Aerospace R&D, US Army ITC-PAC Asian Research Office, and Aju Information Technology. VLDB 2006 is hosted by KAIST and AITrc, Korea.

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

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