Incorporating String Transformations in Record Matching

Arvind Arasu Surajit Chaudhuri Kris Ganjam Raghav Kaushik
Microsoft Research
{arvinda, surajitc, krisgan, skaushi}@microsoft.com

ABSTRACT
Today's record matching infrastructure does not allow a flexible way to account for synonyms such as "Robert" and "Bob" which refer to the same name, and more general forms of string transformations such as abbreviations. We expand the problem of record matching to take such user-defined string transformations as input. These transformations coupled with an underlying similarity function are used to define the similarity between two strings. We demonstrate the effectiveness of this approach via a fuzzy match operation that is used to lookup an input record against a table of records, where we have an additional table of transformations as input. We demonstrate an improvement in record matching quality and efficient retrieval based on our index structure that is cognizant of transformations.

Categories and Subject Descriptors
H.2.4 [**Database Management**]: Systems; H.2.8 [**Database Management**]: Database Applications

General Terms
Algorithms, Performance, Experimentation

Keywords
Data cleaning, Record Matching, Transformation Rules

1. INTRODUCTION
Data cleaning is essential in order to use data warehouses for accurate data analysis. For example, owing to various errors in data, the customer name in a sales record may not match exactly with the name of the same customer as registered in the warehouse. This motivates the need for record matching. A critical component of record matching involves determining whether two strings are similar or not: Two records are considered matches if their corresponding (string) attributes are similar. String similarity is typically measured via a similarity function that, given a pair of strings returns a number between 0 and 1 – a higher value indicating a greater degree of similarity with the value 1 corresponding to equality. This function is used to perform a similarity join between two input relations that returns pairs of strings whose similarity is above an input threshold.

As reviewed by Koudas, Sarawagi and Srivastava [1], previously proposed similarity functions focus primarily on the syntactic difference between strings. While this is indeed an indicator of similarity, there are many cases where strings that are syntactically far apart can still represent the same real-world object. This happens in a variety of settings --- street names (the street name *SW 154th Ave* is also known as *Lacosta Dr E* in the postal area corresponding to zipcode *33027*), the first names of individuals (*Robert* can also be referred to as *Bob*), conversion from strings to numbers (*second* to *2nd*) and abbreviations (*Internal Revenue Service* being represented as *IRS*).

Equivalences are only one form of string variations. We also have variations such as abbreviations that lose information – a first initial (such as *J*) can be expanded in multiple ways (*John*, *Jeff*, etc.) that are clearly not equivalent.

In our recently published research paper [2], we show that it is impractical to expect a generic similarity function to be cognizant of variations such as those of the above examples since they are highly domain-dependent. Rather, the matching framework has to be customizable to take these variations as *explicit input*. Our research paper presents such a framework [2]. We have implemented this framework over Microsoft SQL Server 2005 as an operator which is a part of the Extract-Transform-Load platform namely SQL Server Integration Services [6]. Our goal is to demonstrate this operator.

The demonstration proposal is organized as follows. We outline our framework briefly along with the algorithms needed to make the framework computationally efficient in Section 2. We describe the architecture of our system in Section 3 and the demo scenarios in Section 4.

2. TRANSFORMATION BASED FRAMEWORK
We briefly outline our framework to capture string transformations in this section. More details can be found in our research paper [2]. We model strings as a sequence of *tokens*. A given string may be tokenized for instance by splitting it based on delimiters such as white spaces. We use the term *string* to refer to a sequence of tokens and the term *substring* to refer to a subsequence of tokens. A *transformation rule* consists of a triplet (context, lhs, rhs) where each of context, lhs, rhs is a string. An example transformation is: (33027, "SW 154th Ave", "Lacosta Dr E").

We now describe how a string s can be transformed given a set of transformation rules T. A rule (context, lhs, rhs) can be applied to string s if both context and lhs are substrings of s; the result of applying the rule is the string s' obtained by replacing the substring matching lhs with rhs.

We can apply any number of transformations one after another. However, a token that is generated as the result of applying a transformation cannot participate in any subsequent transformation.

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.
ACM 978-1-60558-102-6/08/06.

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

The set of strings generated by *s* is the set of all strings obtained by applying zero or more transformations to s. We can assign a similarity score to two strings given a similarity function f and a set of transformations T. The similarity between strings s1 and s2 under T is defined to be the maximum similarity as measured by f among all pairs s1' and s2' respectively generated by s1 and s2 using T.

Example: Consider the strings ``SW 154th Ave Florida 33027'' and ``Lacosta Dr E, FL 33027''. Under the transformations ("", "FL", "Florida") and ("33027", "SW 154th Ave", "Lacosta Dr E"), their similarity is 1 since both of the above strings generate the same string "Lacosta Dr E, Florida 33027".

2.1 Algorithm

The main computational problem we addressed is the following: given a table R, called the *reference* table, consisting of strings and a set of transformation rules T, and a string s and similarity threshold α, find all strings in R whose similarity under T with s is greater than or equal to α. Our solution to this problem involves building an index over R.

The index structure and index creation algorithm are based on the techniques described in [2] where we focused on the *join* problem. In the join problem, we have two tables R and S and a set of transformations T. Our goal is to find all pairs of strings in R and S whose similarity under T is greater than or equal to α. We now summarize our techniques.

There are previously known algorithms for computing the similarity join [1]. A naive adaptation of these algorithms is as follows. Consider the expanded relations ExpandR and ExpandS obtained by applying the transformations to each string in R and S respectively. We can compute the similarity join between ExpandR and ExpandS by using any previously known technique and then find all distinct (r,s) returned by this join. The main problem with this approach is that the expansion factor can be prohibitively high.

In our research paper [2], we focused on a class of signature-based algorithms [3] which includes several well-known techniques such as locality-sensitive hashing [5]. In order to address the expansion problem mentioned above, we considered several optimizations to compress the number of signatures. We also proposed optimizations for the problem of checking whether two strings have a similarity greater than or equal to α under T by making connections to the problem of bipartite matching. We refer the reader to [2] for these details.

3. SYSTEM ARCHITECTURE

Our system implements two operations incorporating transformations: (1) creating an index over a given reference table for a given set of transformations T, and (2) looking up an input string using the index to find all strings in the relation whose similarity under T is more than threshold α. The underlying similarity function is *jaccard* similarity [1] where the strings are tokenized into sets and the similarity is computed by taking the ratio of the (weighted) intersection to their union.

We have developed our system as a stand-alone Windows *dll* (dynamically linked library). The primary usage of this system is in the context of loading data into a data warehouse where the reference table is stored in a database and the data being loaded has to be cleansed first. Therefore, we package the dll as an operator over Microsoft SQL Server 2005 as a part of SQL Server Integration Services (SSIS) [6]. Microsoft SQL Server 2005 Integration Services is a platform designed to make the development of scalable Extract, Transform and Load (ETL) workflows easier. A typical data flow pipeline involves data sources (such as a flat file or a database query) connected to a number of operators connected ultimately to data destinations (such as the final warehouse database). Data flows from the sources through the operators and is then output at the destinations.

The Fuzzy Lookup operator in SSIS that performs approximate string matching has been demonstrated previously [4] and we incorporate our system into this operator, thus enriching Fuzzy Lookup with transformations. This operator can be seamlessly combined with numerous other operators to create very powerful ETL solutions.

4. DEMO SCENARIOS

Our demonstration will highlight the benefits of our transformation-based framework as well as the performance of our algorithm.

4.1 Scenario 1: Benefit of Transformations

We will first demonstrate (1) how we can leverage transformation rules to match strings that are syntactically far apart, and (2) how a similarity function that is not cognizant of transformations will fail to achieve this.

This will be shown visually through a graphical tool. Figure 1 shows a screenshot of our tool. As we can see, we have an input string "Bill Gates, One Microsoft Way, Bldg 34, Redmond Washington 98052" being matched against a reference table of strings. Immediately below the input string is a set of matching records from the reference relation. As can be seen, we obtain several matches to strings that are syntactically far apart, such as "BillG, Microsoft Main Campus, Seattle WA 98052". This is the result of using the set of transformations shown on the side.

A special case arises when we have no explicit transformations. In this case, our matching algorithm reduces to jaccard similarity. We will illustrate this special case to show why taking transformations into account is essential for matching quality.

We will also demonstrate the wide applicability of our transformation rules framework by using real-life data from several different databases including addresses, bibliographic citations, and product names.

4.2 Scenario 2: Lineage

Next, we will demonstrate how our tool displays the set of transformations that led to two strings being matched. This is shown in the bottom two panes in the screenshot in Figure 1. Thus, a user of the tool can understand the lineage of the match produced. For example, the input record is matched with the reference record "Bill Gates, 1 Msft Wy, Redmond Washington

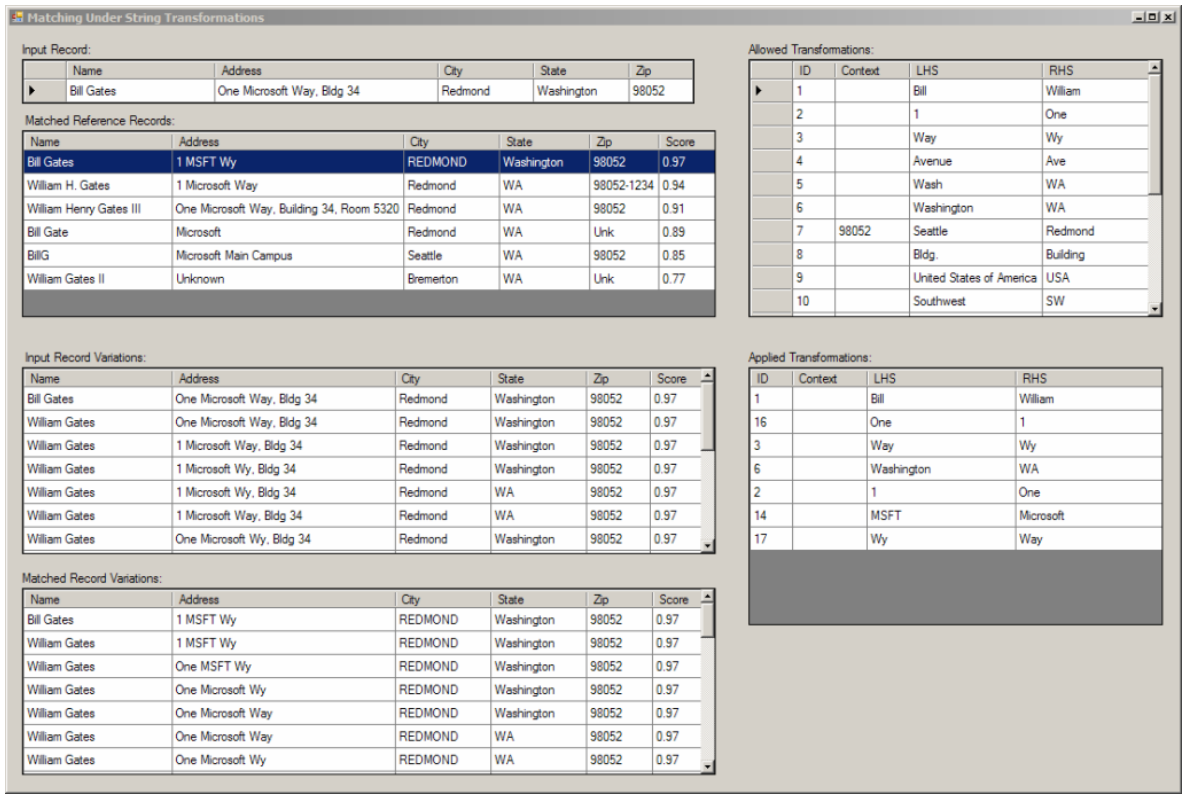

Figure 1: Screenshot of interactive tool

98052" via the rules ("", "Microsoft", "Msft"), ("", "Way", "Wy") and ("", "One", "1").

4.3 Scenario 3: Interactively Changing Transformations

We will then demonstrate how we can interactively change the set of transformations to improve the overall data quality. For example, the transformations used in Section 4.2 are not adequate to match the input record with the record "BillG, Microsoft Main Campus, Seattle WA 98052". However, adding transformations ("98052", "One Microsoft Way", "Microsoft Main Campus") and ("", "BillG", "Bill Gates") leads to a match.

4.4 Scenario 4: Keyword Search

Finally, we will demonstrate a stand-alone keyword search application where we are searching for a string over a reference table of strings. This scenario is primarily intended to illustrate the computational efficiency of our algorithms. Specifically, the application will illustrate the scalability of our algorithms both in the input size (indexed reference table) and the number of transformation rules. We will use a reference relation consisting of a large number of individual names and addresses as shown in Figure 1, and transformation rules obtained from USPS.

We will show the response time of our indexing algorithm for various sets of transformations. Two useful reference points are the performance of a linear scan and the performance of our algorithm when the transformation set is empty.

By considering larger sets of transformations systematically, we will show the improvements in response time yielded by our algorithm and also the overheads imposed by transformations.

5. REFERENCES

[1] Nick Koudas, Sunita Sarawagi, Divesh Srivastava. *Record linkage: Similarity Measures and Algorithms.* ACM SIGMOD 2006.

[2] A. Arasu, S. Chaudhuri and R. Kaushik. *Transformation based Framework for Record Matching.* IEEE ICDE 2008.

[3] A. Arasu, V. Ganti and R. Kaushik. *Efficient Exact Set Similarity Joins*. VLDB 2006.

[4] S. Chaudhuri, K. Ganjam, V. Ganti, R. Kapoor, V. R. Narasayya, T. Vassilakis. *Data cleaning in Microsoft SQL server 2005*. ACM SIGMOD 2005.

[5] A. Gionis, P. Indyk and R. Motwani. *Similarity search in high dimensions via hashing.* VLDB 1999.

[6] Microsoft SQL Server Integration Services. *http://msdn2.microsoft.com/en-us/library/ms141026.aspx*