Multi column matching for database schema translation

Robert Warren, Frank Wm. Tompa
School of Computer Science, University of Waterloo
{rhwarren, fwtompa}@uwaterloo.ca

Technical Report CS-2005-24
August 19, 2005

Abstract

We describe a generalised method for discovering complex schema matches involving multiple database columns. The method does not require linked data and is capable of dealing with both fixed- and variable-length field columns. This is done through an iterative algorithm which learns the correct sequence of concatenations of column substrings in order to translate from one database to another. We introduce the algorithm along with examples on common database data values and examine its performance on real-world and synthetic data sets.

1 Introduction

In our work we wish to find a general purpose method capable of resolving complex schema matches requiring information from a number of columns within a database. While heuristics can be attempted for simple translation operations such as concat (firstname, lastname) = fullname, no general purpose solution has yet been devised capable of searching for and generating translation procedures.

Specifically, we wish to find a method capable of discovering a solution for problems as diverse as unknown date formats, unlinked login names, field normalisations and complex column concatenations. Thus, we wish to find a generalisable method capable of identifying complex schema translations of the sort “4 leftmost characters of column lastname + 4 rightmost characters of column birthdate = column userid”.

In spite of the high computational cost of searching for a solution, it is affordable when compared with the time that would need to be spent by humans. It is not uncommon for commercial databases to have thousands of tables with several hundred columns per table; under these conditions computational support for database integration becomes critical.

This paper describes a method that can be used to identify complex, multi-column translations from one database to another in the form of a series of concatenations of column substrings. The algorithm will discover translations as long as there exists some overlap between the translated schema values and the targeted schema values.

2 Previous work

Rahm and Bernstein present a general discussion and taxonomy of column matching and schema translation [1, 2]. They classify column matchers as having “high cardinality” when able to deal with translations involving more than one column. These types of matchers have been implemented on a limited basis in the CUPID system [3] for specific, pre-coded problems of the form “concatenate A and B”.

As a means of abstracting away from the specific data being processed, Doan proposed “format learners” [4]. These infer the formatting and matching of different datatypes, but the idea has not been carried forward to multiple columns. Recently, Carreira and Galhardas [5] looked at conversion algebras required to translate from one schema to another and Fletcher [6] used a search method to derive the matching algebra. Embley et al. [7] explored methods of handling multi column mappings through full string concatenations using an ontology driven method.

The IMAP system [8] takes a more domain oriented approach by utilising matchers that are designed to detect and deal with specific types of data, such as phone numbers. This approach to finding schema translations for numerical data using equation discovery is particularly novel.

We expand on this idea in our approach, but intend to generalise it to string operations without assuming that examples are provided or that linking relationships have been pre-established across databases. We do this by approaching the problem of schema matching and translation through an instance-based approach where the actual values of columns are translated and matched across databases.

Source				Target
first	middle	last	...	Login
robert	h	kerry	...	nawisema
kyle	s	norman	...	jlmalton
norma	a	wiseman	...	rhkerry
...	...	...	...	...
amy	l	case	...	alcase
josh	a	alderman	...	ksokmoan
john	l	malton	...	ksnorman

Table 1: The first sample schema translation problem where login names must be matched to the columns of an unlinked table.

3 Proposed methodology

Let us assume that we have an unordered table T_1 that we term the source table, with columns B_1, B_2, ..., B_n. These columns may or may not be relevant to the translation. Similarly, we have an unordered aggregate table T_2, named the target table, with a single aggregate column A. We attempt to find a solution such that the values in the target column A can be defined as a series of concatenation operations in the manner of $A = \omega_1 + \omega_2 + ... + \omega_i$, where ω_i represents a substring of one of the source columns B_j, and all values are taken from the same row in T_1.

In our formalised model we assume that the tables T_1, T_2 and their respective columns have been previously chosen by an algorithm or heuristics that present potential pairs of tables one by one. Our work is to be implemented as part of a larger database integration framework which applies different integration methods and selects the lowest cost solution.

Furthermore, most current relational databases provide access to the relations using their own extensions to Structured Query Language (SQL). With portability and generalisation in mind, we follow the lead of Koudas et al. [9] and restrict our method to operations that can be implemented with basic SQL commands to ensure the practicality of the resulting algorithm.

The method functions in four steps: selecting an initial source column B_k, creating an initial translation recipe that isolates a substring ω_x from it, iterating for additional columns, and finally ranking the translation formulas for best matching ability. The overall algorithm is shown in the Appendix as Algorithm 1.

In the first step, all source columns are scored to identify those most likely to be part of the target column at a certain substring size. This information is used to create an initial translation formula which partially maps the source column to the target column. Using this coarse translation formula, we can then iterate through additional selections until either a complete translation formula has been found, or the addition of columns no longer provides additional information.

In the following sections, we review each of these steps in detail while following an example based on the data contained with Table 1.

3.1 Beginning the search

In order to choose candidate columns and generate possible translation formulas for very large tables, we need a method to sample values from the source columns. The objective is not to get an optimal column selection as much as identifying a feasible one. In effect we are trying to "bootstrap" the translation with a single useful column from which the search can begin. We present here a heuristic which we use to make this initial selection, with the detailed algorithm typeset in the Appendix as Algorithm 2.

In Table 1, we would prefer to pick the column last which matches, on average, the most caracters from the target column. This would seem to be a simple problem, but because of computational costs we can only afford to "score" a sample of the values of each column. This must also be done in a way that can be implemented using SQL.

We address this problem by sampling a fraction of the unique values of source column B_k and then selecting candidates from target column A based on all the possible sequences of consecutive characters of length q (that is, we use a q-gram approach [10]). As an example, for a row value of *possible* and a search length of 4 characters, we could extract the values from Column A that contain any of the substrings "poss", "ossi", "ssib", "sibl" or "ible". In general we get $n - q + 1$ substrings of length q from a string of length n.

$$\left(\sum_{j=1}^{t} \frac{HitCount(j)}{RecordCount(B_i) * CharCount(j)} \right)^{q} \tag{1}$$

This formula (1) computes a score where the number of matches for a column (B_i) based on the length of the common substring and the average record overlap between both tables. Experimentally, we chose a value of 2 for the q parameter.

For each candidate source column, we sample a pre-determined fraction of the distinct values within the column, yielding t values, and use each of them to yield a set of $n - q_i + 1$ length search keys for the target column. We taken steps to prevent statistical outliers from overwhelming all other rows by attempting to normalise the result.

The number of distinct hits for each key ($HitCount(j)$) is averaged over the length of the key ($CharCount(j)$) and normalised to the total count of distinct values ($RecordCount(B_i)$) within the source column. Lastly, we then take this value to the power of q, to take into account the decrease in probability of this string occurring randomly.

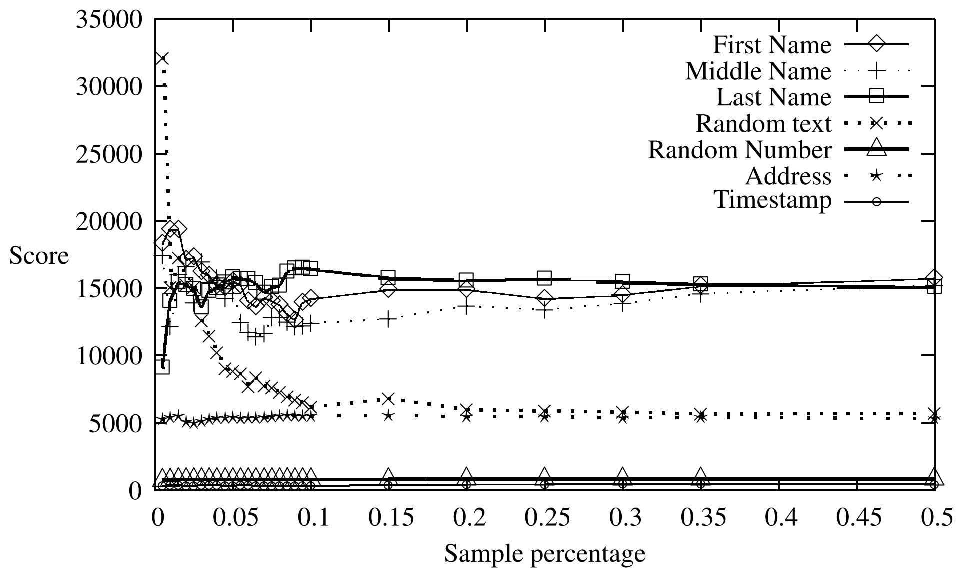

Figure 1: Sampling size is not critical and good result can be obtained with as little as 10% of the total unique rows.

Table 2 and Figure 1 represent empirical experiments computed on the sample dataset (Table 1), sized at 8,000 rows. To verify the correctness of the method we added several noise columns including: a column containing random text, a time-and-date value, a random number and a random street address.

Column	Score
first (B_1)	**14194.4**
middle (B_2)	**12391.7**
last (B_3)	**16374.0**
rnd_text	6151.6
timedate	354.0
rnd_numb	792.9
rnd_addr	5505.3

Table 2: Score results generated with formula (1) with a 10% sample.

The selection of a q-gram size of 2 and using 10% of the distinct values in each source column resulted in acceptable performance for most of the experiments we attempted, though future work should automate the optimal selection of these parameters.

The scores resulting from formula (1) are used to create a preference order with which the source columns should be considered for generating an initial translation formula. The formula works surprisingly well even with a very small sample in large data sets. Figure 2 plots the results of the column selection formula on a dataset containing over 700,000 concatenated first and last names. Even with a very small sample of several hundred rows, the column selection order is accurate.

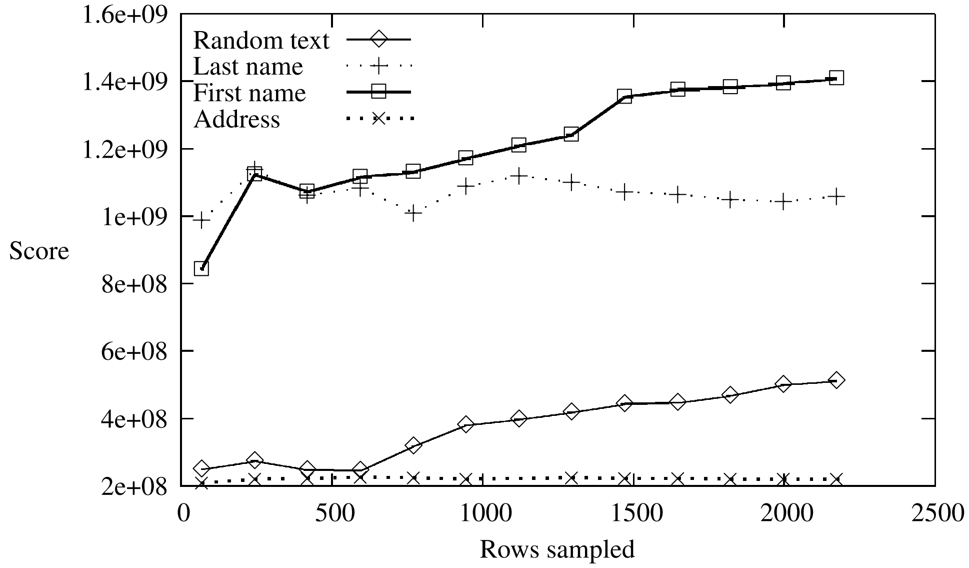

Figure 2: With large datasets (700K rows) the formula does well with very small samples.

3.2 Creating an initial translation formula

With column B_k selected as a starting point, we next create a partial translation formula by comparing sampled values from columns B_k and A. We use equal distance sampling of unique columns values by first selecting a value and then skipping a number of rows before selecting a new one.

An edit-distance method is used to search for commonly repeating patterns between pairs of values selected from these two columns from which we generate a "recipe" that translates specific substrings of values in the source B_k to values in the target column A. The actual insertion, deletion and copy costs assigned to the edit distance methods have not been shown to be critical within our experiments, where we implemented a simple edit distance method as described by Monge [11]. However, any enhancement to the ability to recognise the longest common substring is likely to lower the amount of noise, and increase our capacity of making an accurate recipe choice. We used here a very simple edit distance method supporting basic case-insensitive string operations such as cut, copy and insert. With little effort, advanced operations such as caracter set translation can also be added.

The scoring formula of Section 3.1 has already ensured that an overlap exists between values in these two columns. Given a value key_j from B_k, we can again select all values from A that match at least one q-gram from key_j. For any one of these candidates target values v_i, we can identify all possible substring translations that would map key_j to v_i, which we call "recipes". We repeat this process for the sampled values of B_k while collating the occurrence of the different possible recipes as in Algorithm 3. Thus, if the value of key_j was "warner" and the values of v_i was "rhwarner", the recipe would be two unknow characters followed by characters 1 through 6 of key_j.

For fixed-field data, it is straight-forward to identify the commonly repeating recipes because the absolute locations of the overlapping substrings will always align across recipes. Any superfluous matches (that is, other characters matching the overlapping field) will occur infrequently enough that the outlier recipes can be discarded.

For variable-length fields, however, the problem is slightly more difficult as the absolute locations of the matching values are not aligned. The relative ordering in which the matching and non-matching substrings are found does however remain consistent. It is by making use of this property that we are able to design a solution process capable of dealing with both fixed-length and variable-length data simultaneously.

To characterise the data from the edit-distance algorithm, we define the concept of a "region" as a series of left-to-right contiguous characters from the source column, or a specific w_x element. Should a character repeat, or the character sequence be broken, a separate region is created. Algorithm 4 outlines the process used to convert the edit-distance recipes to a set of translation formulas.

As each recipe is processed, its known and unknown character sequences are translated into a series of regions. Each region w_x represents a string element either from an unknown source or from specific character positions within a designated source column. We term the sequence of these regions $\omega_1+\omega_2+...+\omega_i$ to be a translation formula which provides a partial method to translate the information from the set $\mathcal{B}$ of source columns to the target column A.

These translation formulas are collated as they are generated from the recipes, but their occurrence counts are incremented by the score of their original recipe. In the specific case where a region within a recipe ends on the last character of a source column, two translation formulas are generated: one with the original absolute column character positions and a second with an end-of-string marker. This specific exception allows us to deal with non-fixed length columns by allowing us to refer to relative positions within the source columns.

Table 3 presents typical regions generated using some of the values from our running example. The typesetting convention used is % for any unmatched region and column[n] for matched characters, where n refers to the nth character of the source column named Column.

The translation formulas are collated and the most frequently occurring translation is selected. This partial translation then becomes the starting point for a search for additional regions within the translation formula.

3.3 Selecting additional columns

We now begin an iterative process which will search for columns matching the unknown regions within the translation formula. Because the source columns are related through table T_1, we are able to retrieve associated values for any column that could form part of the same target column value. It is the relation that allows us to restrict our search to values and columns likely to form part of the target column translation.

We search for additional columns by considering each potential column, generating alternate translation formulas based on the obtained recipes and selecting the highest scored translation formula for

Column		Edit distance recipe
B_3	A	
warner	rhwarner	$\%B_3$[123456]
		or $\%B_3$[1-n]
	klwarder	$\%B_3$[123]$\%B_3$[56]
		or $\%B_3$[123]% B_3[5-n]
	ghkarer	$\%B_3$[23]B_3[56]
		or $\%B_3$[23]B_3[5-n]
amy	laramy	$\%B_3$[1]$\%B_3$[123]
		or $\%B_3$[1]$\%B_3$[1-n]
	amyrose	B_3[123]%
		or B_3[1-n]%
	camyro	$\%B_3$[123]%
		or $\%B_3$[1-n]%
wang	mkwang	$\%B_3$[1234]
		or $\%B_3$[1-n]
wayne	opwayne	$\%B_3$[12345]
		or $\%B_3$[1-n]

Table 3: Sample edit distance recipes for the login data, where B_3 is used in place of last_name.

source		target	Translation	
B_3	B_1	A	Partial	Candidate
kerry	robert	rhkerry	$\%b_3$[1-n]	b_1[1]$\%b_3$[1-n]
	robert	klkerry	$\%b_3$[1-n]	$\%b_3$[1-n]
	robert	gkerry	$\%b_3$[1-n]	$\%b_3$[1-n]
wyn	mike	mkwyn	$\%b_3$[1-n]	b_1[1]b_1[3]b_3[1-n]
kyle	otto	opkyle	$\%b_3$[1-n]	b_1[1]$\%b_3$[1-n]

Table 4: Translation formulas are added to based on partial edit distance recipes

all considered columns. For each considered candidate column we begin by sampling the relation joining the columns already assigned to the translation as well as the candidate column.

As done previously, we sample a percentage of the number of unique relation values in equidistant intervals (we used 10% of the distinct rows). Using this sample, we form a search expression based on the current partial translation formula which will retrieve target column values matching the known regions.

For example, for the recipe %last[1-n] and a source column value of wayne, our search expression would then be select A from T_2 LIKE '%wayne'. The set of target column values returned is then analysed with respect to the source column under consideration, again using an edit distance method. We do this while ensuring that any target column character previously allocated to a known region cannot be matched to a candidate column value.

Thus if we assume that a target column value of opwayne was retrieved using the search '%wayne', we would only allow the 'op' characters to be matched by the edit distance method as in Table 4. In essence we are constraining the search to a specific area of the candidate string while enforcing the record link between related source column values from one tuple. This enforcement is what enables us to extract a valid translation recipe from several independent edit-match methods under noisy conditions.

It is also possible to further restrict the retrieval of unlikely values from the database by making use of the SQL position function and ensuring that at least one q-gram of the candidate column is present in order for a value to be retrieved.

The translation of the collated recipes to a new set of candidate translations is done slightly differently than previously done in Algorithm 4. In the modified Algorithm 5, the sections previously allocated to known regions are not matched to any candidate column.

The resulting translation formulas are then collated and scored according to a second scoring formula (2) which scores the translation (T_j) based both on their occurrence and the source column (B_i) in use.

$$\frac{NormalisedCount(T_j)}{max(1, avgcolumnlength(B_i) - 2)} \tag{2}$$

The formula scores the candidate translations based on a per-column normalised occurrence score, but also penalises the score for large width columns. We found experimentally that with large and wide columns of random characters(e.g. column rtext, the resulting serendipitous matches would increase noise to unacceptable levels. Hence a penalty term ($max(1, avgcolumnlength(B_i) - 2)$) was added to the formula to account for these cases.

The candidate translations are then ranked according to scores and the top ranked formula is retained. A question is whether we should make use of the 2nd or 3rd ranked translations in case of failure. At this point we have concentrated our research on obtaining a single valid translation formula, but plan to look at multiple, simultaneous formulas.

This iterative process continues until a translation formula is completed which has no unknown regions and which provides a complete method to translate the information from T_1 to T_2.

3.4 Experimental results

We used this methods on several different data sets, beginning with a listing of users real names and Unix login names (Table 1) taken from Waterloo's Faculty of Mathematics undergraduate computing systems. Both tables have the same number of rows (about 7,000) and 3 different translation formulas are known to exist to create login names from the actual names. The most common of the login name generation formulas (about 3,000 rows) was returned by the search as login = first[1-1] + last[1-n], and the equivalent SQL translation query was generated as:

```
select substring( first from 1 for 1 ) || last as login from table
where first is not null and char_length( substring( first_name from 1
for 1 )) = 1 and last_name is not null and char_length( last_name ) ≥
1
```

If we remove from both datasets the records translated by this formula, then the method returns the next dominant translation login = first[1-1] + middle[1-1] + last[1-n] which covers about 1,200 rows.

Note that within our implementation the resulting SQL statements enforce the assumptions of the translation formulas. The source columns utilised by the formula cannot be null and they must have the required string lengths to ensure that the translation formula provides a proper mapping and translation from one table to another.

Table 5 was used for further experiments using 10,000 randomly generated time-stamps which were then merged into a single string. In this case we used simple concatenations because of the small width of the fields.

The same background "noise" columns were used as for the first dataset in Table 1. The returned SQL translation query was:

```
select substring( hour from 1 for 2) || substring( minutes from 1 for
2) || substring( seconds from 1 for 2) as fulltime from table where hour
is not null and char_length( substring( hour from 1 for 2)) = 2 and minutes
is not null and char_length( substring( minutes from 1 for 2)) = 2 and
seconds is not null and char_length( substring( seconds from 1 for 2))
= 2
```

Source				Target
secs.	mins.	hrs.	...	time
55	59	02	...	345407
43	23	05	...	330011
12	55	07	...	135741
...	...	...	...	...
33	00	11	...	004107
34	54	07	...	192609

Table 5: Time-stamps represented in single and multiple columns.

Source			Target
first	last	...	full
robert	kerry	...	robertkerry
kyle	norman	...	kylenorman
norma	wiseman	...	normawiseman
...	...	...	...
amy	case	...	amycase
josh	alder	...	joshalder
john	galt	...	johngalt

Table 6: Merged names dataset.

We used a list of names to create Table 6 where the first and last names are merged into a single column. The table is about 700,000 rows long with about 70,000 unique values in both source columns. The same "noise" columns as in Table 1 were used to verify that our was functioning properly. The target column full was generated using the translation full = first[1-n] + last[1-n].

As expected, the SQL translation query returned was:

select `first || last` **as** `full` **from** `table` **where** `first is not null` **and** `char_length( first )` $\geq$ `1` **and** `last_name is not null` **and** `char_length( last_name )` $\geq$ `1`

Beyond the two baseline examples, we also used the DBLP [12] article and Citeseer [13] citation indexes to provide additional real-world text data. The DBLP dataset (article only) contains about 233,000 records and the Citeseer dataset contains about 526,000 records. Even though it was expected that erroneous or missing column information would be in both datasets, no data-cleaning (or normalisation) was performed except for a global conversion to lowercase.

For both datasets we created a new target column citation in the form of year [1-n] + title[1-n] + first author[1-n] and inserted it into a new target table. We then used our method on the generated target table and the source columns to extract what it believed to be the translation formula. As expected, the correct translation formula used to create the synthetic target columns was extracted for both datasets.

As a further test of the method's resilience to noisy data, we then compared the citation column from the DBLP dataset to the source columns from the Citeseer dataset using our method. This is a difficult problem because both datasets are noisy and unlikely to match exactly for most records. This increases the uncertainty in extracting a translation formula because of the amount of noise introduced into the edit distance method.

As a gross means of estimating the amount of overlap between both datasets, we counted the number of common titles (exact match) between both datasets to be about 2,480. With a source column sample size of 1%, the method returned the translation formula year [1-n] + title[1-n] + second author[1-n] which translated about 378 rows between both tables. Choosing the second author was not our expected translation of year [1-n] + title[1-n] + first author[1-n], which would have translated about 714 rows, and we carefully looked at both datasets to establish why.

Both translation formulas are "correct" in that we confirmed through inspection that they translated the proper row instances between both datasets.

Suprisingly, the translation formula returned by the method converts a block of articles within the Citeseer dataset that have their first author and second author inverted. In this case, we are in the odd situation of correctly matching a sizeable number of incorrectly loaded records. Should we remove these erroneous rows, then the method would return the expected translation formula.

A concern is why a translation was returned which occurs for half as many rows as the expected translation. This can be explained by the relatively low number of common records coupled with a sample of 1% of the source columns. By chance the sampling missed locating the dominant, but still rare, translation.

Additionally, a series of experiments were conducted using random sets of dictionary words as column data and a randomly generated translation formulas. We used our method to attempt to recover the translation formula from a both tables with 10,000 rows.

While we succeeded at identifying relevant columns in about 60% of cases for 500 experiments, we noticed that the creation of the translation formula for a single column is sometimes incomplete in high noise environments.

Hence, while the translation may be partially correct, it may be missing a few characters from the source columns, especially when dealing with variable length columns. In our current approach, we are unable to recover from this because we do not revisit columns and the translation remains incomplete. We plan to modify Algorithm 5 to recognise such situations and properly correct the translation formula.

4 Conclusion

Whereas previous approaches took specialised domain-specific matchers to form the matches and translations, we present here a generalised algorithm for most string-based matches. This method attempts to find a translation formula that composes a target column from the concatenation of an arbitrary number of column substrings. We do this without user training or explicit linkage between table rows and experimental results validate the approach for realistic data.

Because the method matches complex column translations and because it is computationally expensive, it must function within a framework of a schema integration system. We make an explicit assumption that a certain overlap exists between both datasets and that the framework is able to provide us with both the target column and a set of candidate columns. Experimentally, we determined that the size of the overlap is not critical.

We are currently working on the discovery of separator constants such as spaces, slashes and hyphens within the translation formula as well as the recognition of space-padded fields. So far, we have had promising results in extracting the separator by querying the unknown regions of the target column instances.

Furthermore, we wish to automate the selection of q-gram and sampling parameters within the method. Another promising research direction is the identification of missing information within the source table and the creation of translation columns to support direct record linkage between databases.

References

[1] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[2] E. Rahm and P. Bernstein, "On matching schemas automatically," Tech. Rep. MSR-TR-2001-17, Microsoft Research, Redmond, WA, February 2001.

[3] J. Madhavan, P. A. Bernstein, and E. Rahm, "Generic schema matching with cupid," in *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pp. 49–58, Morgan Kaufmann Publishers Inc., 2001.

[4] A. Doan, P. Domingos, and A. Y. Halevy, "Reconciling schemas of disparate data sources: a machine-learning approach," in *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pp. 509–520, ACM Press, 2001.

[5] P. Carreira and H. Galhardas, "Execution of data mappers," in *IQIS '04: Proceedings of the 2004 international workshop on Information quality in information systems*, (New York, NY, USA), pp. 2–9, ACM Press, 2004.

[6] G. H. L. Fletcher, "The data mapping problem: Algorithmic and logical characterizations," in *International Special Workshop on Databases For Next Generation Researchers (SWOD) at ICDE 2005*, 2005.

[7] D. W. Embley, L. Xu, and Y. Ding, "Automatic direct and indirect schema mapping: experiences and lessons learned," *SIGMOD Rec.*, vol. 33, no. 4, pp. 14–19, 2004.

[8] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos, "imap: discovering complex semantic matches between database schemas," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 383–394, ACM Press, 2004.

[9] N. Koudas, A. Marathe, and D. Srivastava, "Flexible string matching against large databases in practice.," in *VLDB* (M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, eds.), pp. 1078–1086, Morgan Kaufmann, 2004.

[10] E. Ukkonen, "Approximate string-matching with q-grams and maximal matches," *Theor. Comput. Sci.*, vol. 92, no. 1, pp. 191–211, 1992.

[11] A. E. Monge and C. Elkan, "An efficient domain-independent algorithm for detecting approximately duplicate database records.," in *DMKD*, pp. 0–, 1997.

[12] "Dblp citation index," May 2005. http://dblp.uni-trier.de/xml/.

[13] "Citeseer. ist scientific literature digital library," May 2005. http://citeseer.ist.psu.edu/oai.html.

A Algorithms

Data: For a set $\mathcal{B}$ of columns B_1, B_2, ..., B_n and target A
Find column B_{start} most likely part of A;
Generate a translation τ partially translating B_{start} to A;
while *Transformation* τ *has unknowns* **do**
    foreach *Columns* B_k *in* B_1, B_2, ..., B_n **do**
        Sample columns from T_1 and select values of A matching partial translation τ';
        Generate a new τ' partially translating B_k to A;
    end
    Score each τ', B_k;
    Insert highest ranked into Transformation τ;
end

Algorithm 1: Overall algorithm

set B_{best} to null;
set $score_{best}$ to 0;
foreach *column* B_k *of* T_1 **do**
    count distinct values of B_k as $dcount$;
    $hitcount$ =0;
    for *j=1* **to** $dcount/10$ Step $\frac{dcount/90}{dcount/10-1}$ **do**
        get value key from B_k ;
        $localc =$ count T_2 where A like q-grams of key;
        $hitcount += \dfrac{localc}{length(key)}$;
    end
    score(B_k) = $\left(\dfrac{HitCount}{dcount/10}\right)^q$;
    if *score*(B_k)> $score_{best}$ **then**
        $score_{best}$ = score(B_k);
        B_{best} = B_k;
    end
end

Algorithm 2: Initial column selection using a fixed q gram size

Data: A candidate column B_k.
Result: Edit distance recipes $\mathcal{R}$.
count distinct values of B_k as $dcount$;
$Recipes$ = null;
for *j=1* **to** $dcount/10$ Step $\frac{dcount/90}{dcount/10-1}$ **do**
    get value key from B_k ;
    Create set $\mathcal{A}$ from T_2 where A like q-grams of key;
    foreach $candidate$ **in** $\mathcal{A}$ **do**
        Recipe $R =$ edit-distance(key, $candidate$) ;
        search $\mathcal{R}$ for Recipe matching R ;
        if *found* **then**
            increase count of R entry by 1 ;
        else
            create new entry in $\mathcal{R}$ with score 1;
        end
    end
end

Algorithm 3: Creating an initial set of recipes from a candidate column.

```
Data: Edit distance recipes 𝓡.
Result: Partial translation formulas 𝒯.
foreach R in 𝓡 do
    create empty T ;
    begin region ;
    foreach char in R do
        if key chars still in sequence then
            region continues ;
        else if 1st char is from key then
            region continues ;
        else if region still unknown then
            region continues ;
        else if 1st char unknown then
            region continues ;
        else if known region ends on key boundary then
            clone region ;
            mark cloned region as end-of-string;
            link both regions to end of T chain ;
            begin region ;
        else
            (un)known region or recipe ends
        end
        link regions to end of T chain ;
    end
    search 𝒯 for translation matching T ;
    if found then
        increase count of 𝒯 entry by 1 ;
    else
        create new entry in 𝒯 with score 1;
    end
end
```

Algorithm 4: Generation of translation formulas from recipes.

```
Data: A set of candidate columns 𝓑, a partial translation T
Result: A new translation T
Init T_best;
foreach column B_i in 𝓑 do
    𝓡 = AlgorithmSix(B_i, T);
    foreach R in 𝓡 do
        create empty T_new ;
        begin region ;
        foreach char in R do
            if key chars still in sequence then
                region continues ;
            else if 1st char froms part of T then
                region continues ;
            else if region still unknown then
                region continues ;
            else if 1st char unknown then
                region continues ;
            else if known region ends on key boundary then
                clone region ;
                mark cloned region as end-of-string;
                link both regions to end of T_new chain ;
                begin region ;
            else
                (un)known region or recipe ends
            end
            link regions to end of T_new chain ;
        end
        search 𝒯 for translation matching T_new ;
        if found then
            increase count of 𝒯 entry by 1 ;
        else
            create new entry in 𝒯 with score 1;
        end
        foreach T_new in 𝒯 do
            Score T_new on formula 2;
        end
        if Score(T_new) > Score(T_best) then
            T_best = T_new;
        else

        end
    end
    return T_best;
end
```

Algorithm 5: Selecting additional columns for a partial translation.

Data: A candidate column B_k, a candidate translation T
Result: Edit distance recipes $\mathcal{R}$.
for B_k *and all columns in* T **do**
  count distinct relations as $dcount$;
end
for *j=1* **to** $dcount/10$ Step $\frac{dcount/90}{dcount/10-1}$ **do**
  Initialize SearchPattern;
  foreach *region in* T **do**
    if *region is known* **then**
      Get value of region column;
      Extract substring from column;
      Add substring to SearchPattern;
    else
      SearchPattern =+ '%';
    end
  end
  get value key from B_k ;
  Create set $\mathcal{A}$ from T_2 where A like SearchPattern and contains q-grams of key;
  foreach $candidate$ **in** $\mathcal{A}$ **do**
    Apply SearchPattern mask to edit-distance;
    Recipe $R =$ edit-distance(key, $candidate$) ;
    search $\mathcal{R}$ for Recipe matching R ;
    if *found* **then**
      increase count of R entry by 1 ;
    else
      create new entry in $\mathcal{R}$ with score 1;
    end
  end
end

Algorithm 6: Creating edit distance recipes for a new candidate column.