IMPROVING DATABASE QUALITY THROUGH ELIMINATING DUPLICATE RECORDS

***Mingzhen Wei*[1*]*, Andrew H. Sung* [2]*, Martha E. Cather* [3]

[1*] *University of Missouri-Rolla, 1870 miner Circle, Rolla, MO, 65409, USA, email: weim@umr.edu*

[2] *New Mexico Institute of Mining and Technology, Socorro, NM, 87801, USA, email: sung@cs.nmt.edu*

[3] *New Mexico Petroleum Recovery Research Center/New Mexico Tech, Socorro, NM, 87801, USA, email:Martha@prrc.nmt.edu*

ABSTRACT

Redundant or duplicate data are the most troublesome problem in database management and applications. Approximate field matching is the key solution to resolve the problem by identifying semantically equivalent string values in syntactically different representations. This paper considers token-based solutions and proposes a general field matching framework to generalize the field matching problem in different domains. By introducing a concept of String Matching Points (SMP) in string comparison, string matching accuracy and efficiency are improved, compared with other commonly-applied field matching algorithms. The paper discusses the development of field matching algorithms from the developed general framework. The framework and corresponding algorithm are tested on a public data set of the NASA publication abstract database. The approach can be applied to address the similar problems in other databases.

Key Words: Field matching, General field matching framework, String matching points, String matching patterns

1. INTRODUCTION

With the development of WWW and databases, data are becoming more available from different resources. To benefit the data storing and retrieving, majority of useful data is stored in large databases under certain database management systems (DBMS). Due to problems in different stages of data flow (Dasu and Johnson, 2003), it is very easy to introduce data quality problems in databases. Based on Kukich (1992a, 1992b), the average error rate is 1─3% in typed data, 1─16% in optical character recognition (OCR) processed data, and 5─6% in data obtained by voice communication, respectively. The errors are caused by misspellings, typos, abbreviations both standardized and non-standardized, character recognition problems or phonetic problems inherent in these data acquisition methods. Rahm and Do (2000) analyze data quality problems in single and integrated data sources, as data quality problems being classified as single-sourced and multiple-sourced problems. Single-sourced data quality problems include data quality problems caused

by data acquisition and data modeling problems, as illegal values and inter-field mapping problems, duplicate records; while multi-sourced data quality problems mainly refer to duplicate or redundant records that are caused by different data conventions and formats in multiple databases.

For above mentioned reasons, data quality problems block recording of real-world objects correctly. When occurring in the identifying attribute domains, these problems also obstruct the efficient data access. Problems created by low quality data is often shown by an example of customer data, as described in Ananthakrishna, Chaudhuri and Ganti (2002). When multiple variations of a customer's contacts are saved in databases, as [Lisa Simpson, Seattle, WA, USA, 98025] and [Lisa Simson, Seattle, WA, USA, 98025], duplicated information will significantly increase the costs on direct mailing. In addition, such duplicates will cause incorrect results in data analyses, such as a survey of customers of a specific chain store or the customers' shopping pattern identification. With all stunning large numbers in its costs for enterprises and industries in white papers, as cited in a white paper from Trillium Software (2004), data quality problems are drawing peoples' attention.

In this paper, the central problem of field matching in database management is discussed. A general field matching framework is proposed to consider token-based field matching in general. By introducing the concept of String Matching Points (SMP) in string comparison, improved efficiency and accuracy are observed. Major string matching problems in different attribute domains are analyzed. A corresponding field matching algorithm is developed for testing data set from the NASA publication abstract database. Further improvement of computational efficiency is discussed in the experiments. Promising results are observed.

The rest of this paper is organized as follows. Section 2 reviews the related work on the field matching problem and related algorithms; Section 3 presents the concept of String Matching Points in string comparison; Section 4 presents the proposed field matching framework; Section 5 depicts the experimental setups and results; and Section 6 concludes the paper.

2. RELATED WORK

The nature of field matching is to identify the similarities of domain values in string representations for determining their equivalency. String matching has been studied intensively for its applications in signal processing, computational biology, information retrieval, intrusion detection and others, as summarized in Sankoff and Kruskal (1983). In those areas, the goal of approximate string matching is to identify the Longest Common Subsequence (LCS), or to identify one feature in one sequence from the other sequence, or to identify the abnormality in the targeting sequences. Many algorithms have been developed to resolve approximate matching problems in these fields. Navarro (2001) provides a comprehensive overview of these algorithms.

Since field matching is to identify equivalent string values in attribute domains in databases, it is required to develop algorithms based on the characteristics of string matching problems and the goals. It is to identify semantically equivalent (identifying) attribute values in syntactically different representations. As in other application areas, the equivalency of two string values is modeled by their similarity degree in the range of [0,1], with one as equivalent, zero no similarity. Selection of a proper approximate matching algorithm is a domain-dependent task, and should be done according to the nature of quality problems in selected domains.

In the literature, field matching algorithms can be classified into three broad categories: a) character-based, b) q-gram-based and c) token-based algorithms.

2.1 Character-based Field Matching

Character-based string matching is a large family for its wide application areas, as summarized by Navarro (2001). Although designed with different strategies, character-based string matching takes strings as sequences of characters, as in $w = c_1 c_2 ... c_n$, and compares two strings character by character. The most commonly applied algorithm in this category is the Levenshtein algorithm (Levenshtein, 1966), also called simple edit distance algorithm, which calculates the minimum number of atomic editing operations (e.g. insertions, deletions, transpositions and substitutions) that are required to transform one string to the other. The edit distance of two strings can be denoted as ED(*str1*,*str2*). An example is $ED("university","universty") = 1$. Levenshtein algorithm is often applied by its normalized variations (Hernandez and Stolfo, 1995, Navarro et al., 2001, Elfeky, Verykios and Elmagarmid, 2002, Chaudhuri et al., 2003). The most popular variation is to normalize the edit distance of two strings by their maximum length, formulated as $sim(s_1, s_2) = 1 - \frac{ED(s_1, s_2)}{\max(|s_1|, |s_1|)}$.

Character-based field matching is capable of comparing strings with slight edit variations created in data entry and OCR processing. There are other character-based field matching algorithms, such as the Smith-Waterman algorithm (Smith and Waterman, 1981) and the Jaro algorithms (Jaro, 1989). Some algorithms are developed to consider first character errors, as a variation of the Jaro algorithm described in Winkler (1990). Many are designed for searching certain string pattern in long textual files in online search, as a two-way string comparison algorithm (Crochemore and Perrin, 1991). Online pattern search is a different prospect from the field matching problem, which will not be addressed in this paper.

2.2 Q-Gram-Based Field Matching

Given a string *S* and an integer *q*, the set of *q*-gram of *S*, denoted as $G_q(S)$, is obtained by sliding a window of length *q* over the characters of string *S*. For instance, the 3-gram set of "shackleford" can be represented as $G_3("shackleford") = \{'sha','hac','ack','ckl','kle','lef','efo',' for','ord'\}$. String matching using *q*-grams is based on following observation: if two strings are similar to each other they share a large number of *q*-grams in common (Ukkonen, 1992). The similarity degree of two strings is usually modeled as the ratio of common q-grams to the total number of distinct q-grams in two strings.

Combined with database management systems, Q-grams can be efficiently applied in string matching, as reported in Gravano et al. (2001). Q-gram-based methods employ the sequential substrings to consider the similarity of two strings. Positional *q*-grams consider the out-of-order or word transposing problems in string matching. But *q*-gram is an inherent space expensive technique, with (*m*-*q*+1) q-grams for a string with length of m. High space consumption means high computational cost in database systems.

2.3 Token-Based Field Matching

Token-based string matching considers strings as consequences or sets of words, as in $S = w_1 w_2 ... w_m$ and $S = \{w_1, w_2, ..., w_m\}$ respectively; and each word as a sequence of characters as in character-based string matching. This adds great flexibility in string comparison for two obvious reasons:

1) Stop words and punctuation, which have no significant meanings in content representation, can be easily removed from strings, and only meaningful strings are left to compare;
2) Abbreviations can be taken care by expressing words as sequences of characters, as in $blvd \rightarrow boule\,\mathrm{var}\,d$ and $NMT \rightarrow New\ Mexico\ Tech$.

The importance and requirement of token-based field matching in resolving field matching problems has been addressed in literature (Kukich, 1992a, Navarro et al., 2003, Han et al., 2004). Both simple and sophisticated algorithms have been developed and applied in field matching. The simplest token-based field matching algorithm is the Jaccard similarity metrics, which counts the number of common words N_C and the number of distinct words N_D of two strings in comparison, and takes the ratio of N_C/N_D as the similarity degree of two strings. The simple field matching algorithm (Monge, Elkan, 1996) is very similar to the Jaccard metrics, in which the similarity degree is calculated by a simple formula $sim(str_1, str_2) = \frac{N_c}{|\ str_1\ | + |\ str_2\ |} \times 2$, where $|str_1|$ is the number of words in str_1, and $|str_2|$ is the number of words in str_2. Given proper thresholds, these algorithms are capable to resolve some equivalence errors without introducing high false positives (e.g., irrelevant string values identified), but they generate a large number of true negatives (e.g., semantically equivalent string values unidentified).

In Lee et al. (1999), an algorithm was proposed to improve the field matching accuracy. First, it sorts tokens in each attribute value in the selected domain, and then sorts records based on selected domain(s) to bring duplicate records as close as possible. The concept of Record Similarity (RS) was proposed to calculate the similarity of records by combining the similarity of tokens and similarity of field values. By defining some string matching patterns, this algorithm improved the field matching accuracy significantly. Because of the sorting process, the algorithm fails some matches, as in processed strings *comput science department new mexico institute of mining and technology* and *cs dept new mexico tech*, all lowercased. After removing the stop words and being sorted correspondingly, two sequences become *comput institute mexico new science technology* and *cs dept mexico new tech*. It is impossible to match *cs* and *comput science*, and hence to identify them as completely equivalent strings by all means, because the original allocation of words is changed.

From above discussion, it is obvious that word allocation plays significant roles in string matching. Under different contexts, one word means and functions differently. For example, a same word *new* has different semantic meanings in fragments of *a new cloth* and *new mexico state*. Therefore, it is important to keep the original allocations in string comparison.

A more sophisticated field matching algorithm was presented in Monge and Elkan (1996, 1997), the recursive field matching algorithm. This algorithm recursively compares substrings of two strings str_A and str_B to obtain the maximum degree of similarity, and calculates the overall similarity by averaging the total similarity degrees, as shown in formula $sim(str_A, str_B) = \frac{1}{|\ str_A\ |} \sum_{i=1}^{|str_A|} \sum_{j=1}^{|str_B|} sim(str_{Ai}, str_{Bj})$. In the recursive string matching algorithm, strings are considered as sequences of words, and best matching is achieved by

comparing substrings iteratively. By cooperating with string matching patterns, this algorithm greatly improves the string matching accuracy. But it is an extremely expensive algorithm due to its recursive comparison of substrings. Another essential problem is that oftentimes it is difficult to find the best matching substrings, especially when related fields are messed up.

Hence, an ideal field matching algorithm should consider the allocation of words in strings, and define how strings match each other by defining string matching patterns, and have an approach to identify potential matching substrings fast for the purpose of improving the string matching efficiency. In this paper, we propose a field matching algorithm that tries to address these problems and to improve the string matching accuracy and efficiency.

3. THE CONCEPT OF STRING MATCHING POINTS

The concept of String Matching Points (SMP) was inspired by the process of comparing two strings using our inborn eyesight, in which we obtain a coarse similarity degree of two strings by two steps. In the first step, two strings are investigated briefly, and potential matching substrings are identified. In the second step, two potential matching substrings are compared carefully in characters or based on pre-defined string matching rules. Repeat these two steps and achieve the overall similarity of two strings.

More formally, SMP are defined as follow: given two strings str_A and str_B, a SMP of a substring *sub* in str_A is defined as the index of tokens *tsmp* in str_B, where substring *sub* and token *tsmp* share the same first characters. By the definition, it is possible that one substring has multiple SMP or has no SMP in the other string. For example, the SMP of substrings of *cs dept new mexico tech* in *comput science department new mexico institute mining technology* (all lowercased), will be 1, 3 and 4 consequently, since *cs* shares the first character with *comput*, *dept* with *department*, and *new* with *new*.

It is obvious that each substring will spot its SMP if two correct strings match each other, even with abbreviations and shorter representations. Sometimes spotting SMP can be a problem for some erroneous variations when the errors occur to the first letters. But based on Kukich (1992a), we can trust the first letter of a word over the others in string comparison. Hence those with SMP will dominate the matches in equivalent strings.

By introducing SMP, string matching is turned into an iterative procedure of following two steps: 1) looking for the string matching points and 2) comparing following substrings detailedly. As the SMP searching is as simple as character comparison, it is an efficient strategy to improve string matching efficiency. As string matching processes, SMP move forward for unmatched tokens.

Figure 1 shows the string matching process with the assistance of SMP. SMP searching make identification of maximum matching substrings easier, rather than a trial-and-error process in the recursive field matching algorithm proposed by Monge and Elkan (1997).

Given: String *A*: *CS dept NMT*
String *B*: *Comput Science department New Mexico Technology*
Compare two strings starting from *A*→*B* from leftmost to the rightmost:

1. match substrings *cs dept nmt* and *comput science department new mexico technology*
   1) Spot the SMP of *cs*: the matching point is 1.
   2) Match *cs* and the other substring: the matched substring in ***B*** is *comput science*.
   3) Mark all matched tokens or words:
      cs dept nmt
      1 0 0
      comput science department new mexico technology
      1 1 0 0 0 0
2. match substrings *dept nmt* and *department new mexico technology*
   1) Spot the SMP of *dept*: the matching point is 3.
   2) Match *dept* and the other substring: the matched substring in ***B*** is *department*.
   3) Mark all matched tokens:
      cs dept nmt
      1 1 0
      comput science department new mexico technology
      1 1 1 0 0 0
3. match substrings *nmt* and *new mexico technology*
   1) Spot the SMP of *nmt*: the string matching point is 4.
   2) Match the *nmt* and the other substring: the matched substring in ***B*** is *new mexico technology*.
   3) Mark all matched tokens:
      cs dept nmt
      1 1 1
      comput science department new mexico technology
      1 1 1 1 1 1

Figure 1. An example of string matching using the concept of string matching points.

4. GENERAL FIELD MATCHING FRAMEWORK

4.1 General Framework

As discussed above, the general field matching framework simulates two steps in string comparison. The general field matching framework follows the basic steps of token-based field matching algorithms, as shown in Figure 2.

Input: two strings str_A and str_B
Output: similarity degree in range of [0, 1] of two strings

General framework:

1. Tokenize str_A and str_B into two sequences of tokens;
2. Remove stop words and punctuations from two sequences based on domain knowledge;
3. Assume processed two strings as str_A' and str_B' and $|str_A'| \leq |str_B'|$

4. String matching
   a. String matching with SMP starting from str_A'
      for each unmatched token W_{Ai} in string str_A':
         a1) Identify the token W_{Bj} as the SMP;
         a2) Construct substring in str_A' from W_{Ai} and substring in str_B' from W_{Bj};
         a3) Compare constructed substrings based on pre-defined string matching patterns;
         a4) Mark all matched tokens in str_A' and str_B'.
   b. String matching without SMP starting from str_A'
      for each unmarked token W_{Ai} in string str_A' :
         a1) Search substrings left in str_B';
         a2) Construct substring in str_A' from W_{Ai} and substring in str_B' from W_{Bj};
         a3) Compare constructed substrings based on pre-defined patters;
         a4) Mark all matched tokens.
5. Calculate the similarity degree of str_A and str_B using formula:

$$sim(str_A, str_B) = \frac{total\ \ number\ \ of\ \ marked\ \ tokens\ \ in\ \ str_A'\ \ and\ \ str_B'}{total\ \ number\ \ of\ \ tokens\ \ in\ \ str_A'\ \ and\ \ str_B'}$$

Figure 2. The workflow of general token-based field matching framework.

As a token-based approach, the framework takes in the advantages of token-based string matching methods and keeps only meaningful contents to compare by removing stop words and punctuation in two strings. Stop words refer to words that mean no significance to the real contents of strings, such as <of, no, and>, and so on, which are determined based on domain knowledge. Punctuation can be {*.& %$#@"\/[],:} in string values, also dependent on the domain knowledge. Removing stop words and punctuation aids in obtaining a correct picture of the equivalency of two strings in comparison.

The actual string matching starts from the shorter string because the shorter string is likely more compressed by abbreviations, and hence less SMP search is required. The framework considers string matches with SMP and matches without SMP. As discussed above, in two equivalent strings, the matches in the first category are expected to be the majority; those in the second category are complementary to pick up matches with errors in the first characters.

In string matching with SMP, first step is to identify the SMP for unmatched tokens, and then construct substrings from the SMP to implement the actual comparisons. In actual string matching applications, the field matching algorithm can be developed from the framework by customizing string matching patterns according to data quality problems in the selected domain. Matches without SMP are caused by the errors in the first characters. No matches in this category fit for pre-defined string matching patterns for matches without SMP. Therefore it is required to compare each unmatched substring in one string with all other unmatched substrings in the other string to find the maximum matches, as in recursive field matching algorithm. Since matches in this category are infrequent, they would not cost much.

4.2 String Matching Patterns

To resolve string matching problems, it is important to define rules or string matching patterns to match syntactically different substrings. As different domains have different types of equivalence errors, different string matching patterns are required. We classify all possible string matching patterns into following categories for facilitating the algorithm development:

1) One-to-one matching: matches between two single words, which include:
   - P1: exact match as *match* → *match*;
   - P2: prefixed match as *sci* → *science*;
   - P3: concatenation of prefix and postfix as *dept* → *department*;
   - P4: shorter representation as *blvd* → *boulevard*;
   - P5: error resilient as *institute* → *instutite*;

2) One-to-n matching: matches between a single word and a multi-word string, which include:
   - P6: acronymic match as *NMT* → *New Mexico Tech*;
   - P7: concatenation of prefixes as *CalTech* → *California Institute of Technology*;

3) N-to-n matching: matches between two multi-word strings, which include:
   - P8: word boundary problem as *New Mexico Tech* → *NewMexico Tech*.

The above string matching patterns are discussed in resolving field matching problems in publications (Monge, Elkan, 1996, 1997, Lee et al.,1999). It is obvious that not all patterns are required for all field matching applications and this list is not a complete one for all field matching problems.

In order to improve the string matching efficiency, substring construction in the general framework is done based on the defined string matching patterns, as defined as followings:

- for one-to-one matching, words W_{Ai} and W_{Bj} are taken to compare;
- for one-to-n matching, continuously unmarked tokens from W_{Ai} in str_A and continuously unmarked tokens from W_{Bj} in str_B are concatenated as sub_{Ai} and sub_{Bj} respectively, and string comparisons are implemented between W_{Ai} and sub_{Bj}, or W_{Bj} and sub_{Ai};
- For n-to-n matching, continuously unmarked tokens from W_{Ai} in str_A and continuously unmarked tokens from W_{Bj} in str_B are concatenated as sub_{Ai} and sub_{Bj} respectively, and string comparisons are implemented between sub_{Ai} and sub_{Bj}.

Because comparisons in one-to-one, one-to-n and n-to-n categories are not same in computational costs, it is important to select proper string matching patterns to resolve emerged string matching problems in specific databases. For example, if defined matches are in one-to-one match, it is not necessary to concatenate the neighboring words into multi-word substrings; and two tokens obtained from two strings are enough. For n-to-n matches, it is required to concatenate continuously unmatched tokens into multi-word substrings for both strings. It is obvious that word concatenation and matching multi-word substrings will cost more than one-to-one matches. Statistical results of equivalence errors in a set of 2,000 distinct affiliation names and a set of 2000 distinct authors' names from the NASA publication abstract database on Astronomy and Astrophysics, and a set of oil producers' names from New Mexico ONGARD database are

shown in Table 1. They are obtained by analyzing the equivalence errors in equivalent string values of these domains.

Table 1. The Distribution of Equivalence Errors in Three Different Domains

Patterns	Affiliation[1]	Name[2]	Oil Producers' Name[3]
P1	93.741	28.178	72.504
P2	3.938	69.932	19.515
P3	0.558	0.156	1.344
P4	0.294	0.734	2.713
P5	0.029	0.165	0.779
P6	0.500	0.028	1.105
P7	0.235	0.028	0.421
P8	0.705	0.780	1.620
P1+P2	97.679	98.11	92.019
P1+P2+P3+P4+P5	98.56	99.165	96.855
P6+P7	0.735	0.056	1.526

Note: 1. Author's affiliation data from the NASA public publication databases
2. Author's name from the NASA public publication databases
3. Oil producer's name or property name from the New Mexico ONGARD database

As seen in Table 1, in three different domains, exact and prefixed matches dominate the equivalence errors, the percentages are 97.68%, 98.11% and 92.02% respectively. It reveals that P1 and P2 are the most important string matching patterns in field matching. The percentages from one-to-one patterns are more impressive, with 98.56%, 99.17% and 96.86% respectively. It indicates that for majority of field matching problems, one-to-one patterns will be sufficient to consider majority of equivalence errors.

5 EXPERIMENTS AND RESULTS

5.1 Setups

A set of affiliation names is abstracted from NASA publication abstract database on Astronomy and Astrophysics. Since it is an official publication database on Astronomy and Astrophysics, it plays important roles in literature retrieval in related research. Data standardization will benefit greatly the data management and other research conducted on the database. Among authors' name, authors' affiliation, paper title, publication title, and other domains, authors' affiliation is the most problematic domain, containing many string matching problems as summarized in Table 1.

The experiments are set up to classify equivalent affiliations into one class, which can great aid in data standardization or removing duplicates in the later stages of data cleaning. To eliminate the effect of searching algorithms, a complete comparison is applied, in which every string compares with every other string to determine their equivalency. Strings with similarity degrees higher than pre-defined threshold are classified into one class. The string matching accuracy is evaluated by the accuracy of the classification.

In this practice, each resulting class belongs to one of following types, compared with a validated classification result:

Type 1: All representations in a class are of the same affiliation, and the class contains all representations of the same affiliation, which is the perfect case for classification.

Type 2: All representations in a class are of same affiliation, and the class does not contain all representations of the same affiliation, which is still correct classification.

Type 3: The other cases that include a class contain representations of multiple affiliations, and totally wrong classification.

5.2 Customized Field Matching Algorithm

Based on Table 1, equivalence errors in one-to-one category is essential to resolve majority field matching problems. By considering one-to-one equivalence errors, including P1, P2, P3, P4 and P5, in matching with SMP and error resilient type of equivalence errors (e.g., P5) in matching without SMP, majority string matching problems can be resolved. By considering these string matching patterns, the actual field matching algorithm is customized as follows in Figure 3.

Input: two strings str_A and str_B
Output: similarity degree in range of [0, 1] of two strings

Procedure:

1. Tokenize str_A and str_B into two sequences of tokens;
2. Remove stop words and punctuation from two sequences based on domain knowledge;
3. Assume two processed strings as str_A' and str_B' and $|str_A'| \leq |str_B'|$
4. String matching
   a. String matching with SMP
      for each unmatched token W_{Ai} in string str_A':
      a1) Identify the token W_{Bj} as the SMP;
      a2) Compare W_{Ai} and W_{Bj} according to one-to-one patterns;
      a3) Mark W_{Ai} and W_{Bj} if they match each other.
   b. String matching without SMP
      for each unmarked token W_{Ai} in string str_A' :
      a1) Search unmatched token W_{Bj} in str_B';
      a2) Compare W_{Ai} and W_{Bj} according to error resilient pattern;
      a3) Mark W_{Ai} and W_{Bj} if they match.
5. Calculate the similarity degree of str_A and str_B using formula:

$$sim(str_A, str_B) = \frac{total\ number\ of\ marked\ tokens\ in\ str_A'\ and\ str_B'}{total\ number\ of\ tokens\ in\ str_A'\ and\ str_B'}$$

Figure 3. The field matching algorithm applied in experiments.

5.3 Results and Discussions

5.3.1 Comparison of Field Matching Algorithms

In comparison of field matching algorithms, we consider classification accuracy and time efficiency. Time efficiency is evaluated by computational costs, and classification accuracy is evaluated by both perfect classification and correct classification. Results are shown in Figures 4, 5 and 6.

From Figure 4, it is obvious that time complexity of the new algorithm is between the simple token-based algorithms, including Jaccard similarity metrics and the simple field matching algorithm, and complicated token-based and character-based algorithms, including normalized edit distance (NED), Jaro and record similarity algorithm (Lee et al., 1999). It indicates that the new algorithm belongs to sophisticated token-based algorithms that consider multiple string matching patterns in string matching; also the new algorithm is an efficient algorithm by skipping a large number of comparisons by searching the SMP before detailed comparison. The new algorithm is a less expensive algorithm compared with NED, and it is more sophisticated in considering potential problems in string matching, such as word transpositions and abbreviations and single-error misspellings.

Results of perfect classification (type 1) are the best results revealing the classification accuracy. As shown in Figure 5, they all achieve their peak performance when threshold is 0.85, which indicates similarity threshold of 0.85 can be considered as the optimized threshold in identifying equivalent values using these algorithms. The best performed algorithms are NED, Jaro and the new algorithm, with the new algorithm being much cheaper.

Correct classification, including classification in Type 1 and Type 2, is also an indicator of the performance of a field matching algorithm. As shown in Figure 6, the NED, Jaro and the new algorithms perform best. This observation discovers that token-based algorithms work no better than character-based algorithms unless necessary string matching patterns are designed, and it confirms the conclusion that proper string matching patterns are required in string matching. Because correct classification only considers classes that are correctly clustered, with one perfect cluster maybe split into several correct clusters, the results are more complicated. For the new algorithm, its accuracy of correct classification is basically climbing as the increase of similarity thresholds, up to its peak at threshold of 0.85.

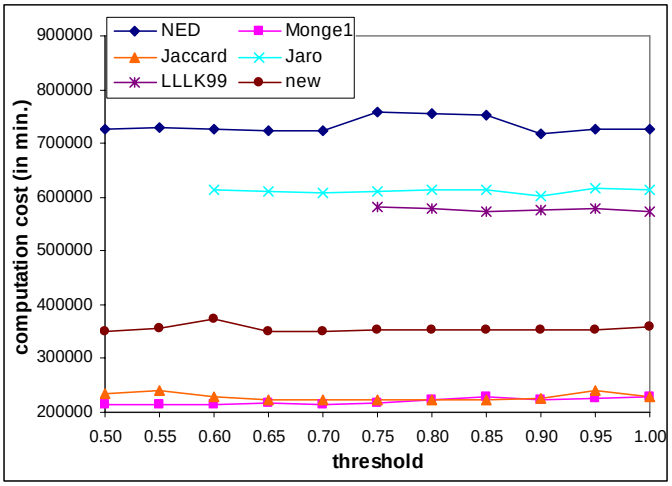

Figure 4. Computational costs using different field matching algorithms.

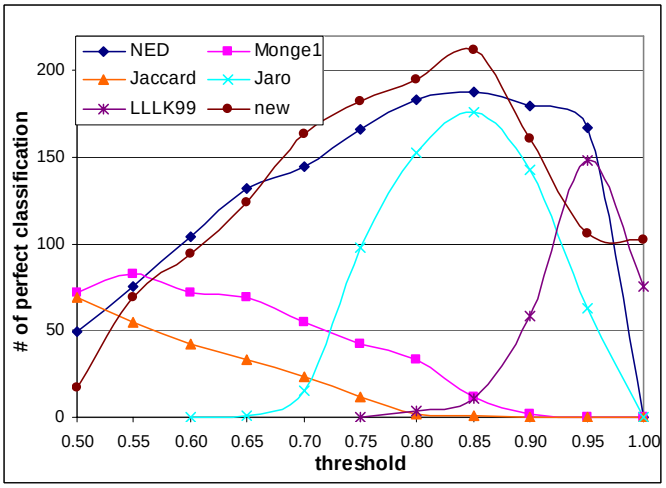

Figure 5. Perfect classification results using different field matching algorithms.

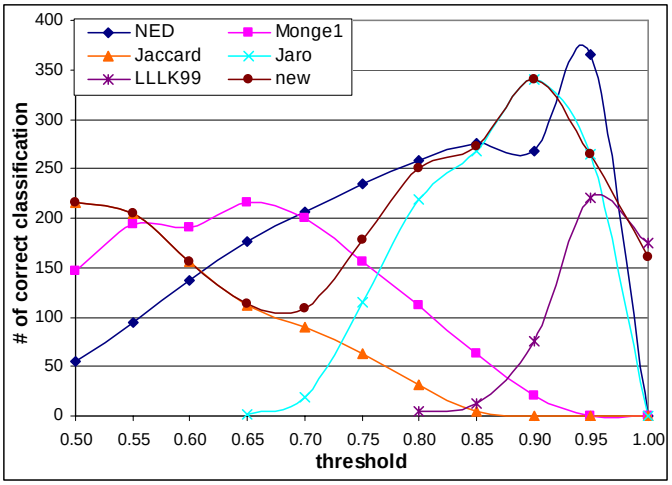

Figure 6. Correct classification using different field matching algorithms.

5.3.2 Effect of String Matching Patterns

Effect of string matching patterns is studied using the same problem, as shown in Fig. 7 and 8. Different field matching algorithms developed from the general field matching framework, as [new 1], [new 2], [new 3], [new 4] and [new 5], consider different string matching patterns. More detailedly, the string matching patterns considered in these algorithms are:

- new 1: P1, P2, P3, P4, P8 with only SMP;
- new 2: P1, P3, P4, P8 with only SMP ;
- new 3: P1, P2, P3, P4, P5, P6, P8 with only SMP;
- new 4: P1, P3, P4, P5, P6, P8 with only SMP;
- new 5: P1, P3, P4, P5, P6, P8 with SMP, and P8 without SMP.

From the results, P2, which is prefixed matches, matters significantly in string matching because [new 1] outperforms [new 2] greatly in perfect classification results before the edit distance threshold as 0.85. This is because that prefixed match is an important pattern, as indicated in Table 1, and ignoring the prefixed matches would cause loss in identifying equivalent values. But it does not necessarily mean down performance in correct classification because correct classification does not require all equivalent values be clustered in one cluster.

Results also reveal that once the most important string matching patterns are considered, no substantial increases in accuracy are observed by adding one or more other string matching patterns. This conclusion also accords with the statistical results in Table 1, which says that one-to-one patterns dominate matches in different attribute domains and other equivalence errors are really minus problems in string matching.

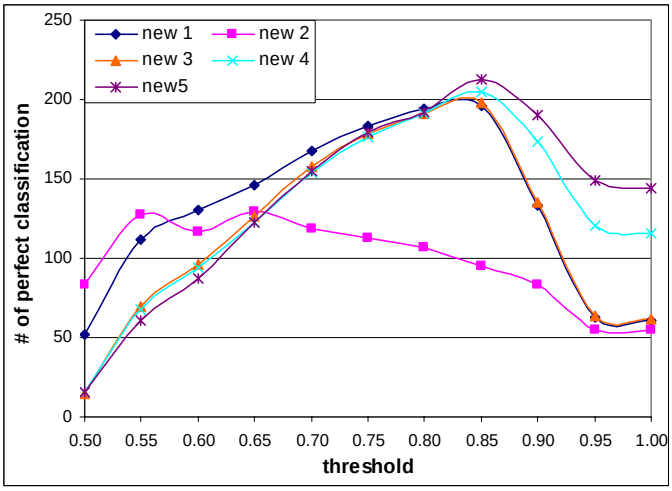

Figure 7. Perfect classification of different field matching algorithms developed using the general field matching framework by considering different string matching patterns.

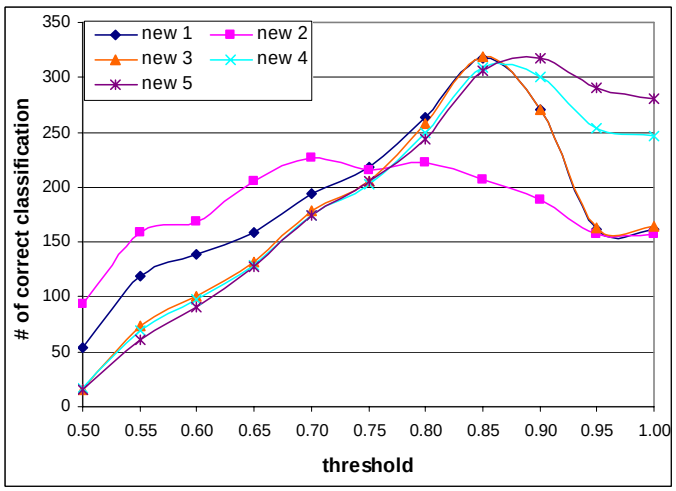

Figure 8. Correct classification of different field matching algorithms developed using the general field matching framework by considering different string matching patterns.

6 CONCLUSION

In this paper, a central problem of entity identity, also named as object identification or field matching in data cleaning community is addressed. Following conclusions are achieved:

- The purposes of a general field matching framework are to facilitate the field matching problems in different domains and databases.

- By introducing SMP, string matching is converted to a two-step process: firstly identifying potential matching starting-points for particular substrings, and then matching substrings based on pre-defined string matching patterns.
- Algorithm development from the general framework is addressed, and an algorithm is developed for a set of affiliation from NASA publication abstract database.
- Experimental results discover the advantages of new field matching strategy over other field matching algorithms, both in string matching accuracy and computational efficiency.
- With the proposed general framework, field matching problems in different domains and databases should be addressed more easily and more accurately.

REFERENCES

Ananthakrishna R., Chaudhuri S., Ganti V., (2002) Eliminating fuzzy duplicates in data warehouses, in the *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.

Chaudhuri, S., Ganjam, K., Ganti, V., Motwani, R., (2003) Robust and Efficient Fuzzy Match for Online Data Cleaning, *proceedings of SIGMOD 2003*, June 9-12, San Diego, CA.

Crochemore M., Perrin D., (1991) Two-way string matching, *Journal of the ACM*.

Dasu, T., Johnson, T., (2003) *Exploratory data mining and data cleaning*, New York: Wiley-Interscience.

Gravano L., Ipeirotis P. G., Jagadish H. V., Koudas N., Muthukrishnan S., Srivastava D., (2001) Approximate string joins in a database (almost) for free, *Proceedings of the 27th International Conference on Very Large Databases*, pp: 491-500.

Han H., Giles L., Zha H., Li C., Tsioutsiouliklis K., (2004) Two supervised learning approaches for name disambiguation in author citations, *Proceedings of the 2004 joint ACM/IEEE conference on Digital Libraries*, Tuscon, AZ, USA.

Hernandez, M. A., Stolfo, S. J., (1995) The Merge/Purge Problem for Large Databases, *proceedings of ACM SIGMOD/PODS*, San Jose, CA.

Jaro M. A., (1989) Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida, *Journal of the American Statistical Association*, Vol. 84 (406).

Kukich, K., (1992a) Techniques for automatically correcting words in text, *ACM Computing Surveys*, Vol. 24(4).

Kukich, K., (1992b) Spelling correction for the tele-communications network for the deaf, *Communications of the ACM*, Vol. 35(5).

Levenshtein V., (1966) Binary codes capable of correcting deletions, insertions and reversals, *Soviet Physics Doklady*, Vol. 10.

Lee, M.L., Lu, H., Ling, T. W., Ko, Y. T. (1999) Cleansing data for mining and warehousing, *proceedings of the 10th DEXA*, pp:751-760.

Monge, A.E., Elkan, C.P. (1996) The field matching problem: algorithms and applications, *Proceedings of the 2nd SIGKDD*, Portland, Oregon, USA.

Monge, A.E., Elkan, C.P. (1997) An efficient domain-independent algorithm for detecting approximately duplicate database records, *Proceedings of DMK97*, Tucson, Arizona.

Navarro G., (2001) A guided tour to approximate string matching, *the ACM Computing Survey*, vol. 33(1), pp: 31-88.

Navarro G., Baeza-Yates R., Arcoverde J., (2003) Matchsimile: a flexible approximate matching tool for searching proper names, *Journal of the American Society for Information Science and Technology*, vol. 54(1).

Navarro, G., Baeza-Yates, R., Sutinen, E., Tarhio, J., (2001) Indexing Methods for Approximate String Matching, *IEEE Data Engineering Bulletin*, Vol. 24(4), pages 19-27.

Rahm, E., Do, H., (2000) Data cleaning: problems and current approaches, *IEEE Bulletin of the Technical Committee on Data Engineering*, Vol. 23(4).

Sankoff D., Kruskal J., (1983) *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison*, Addison-Wesley, Reading, MA.

Smith T. F., Waterman M. S., (1981) Identification of common molecular subsequences, *Journal of Molecular Biology*, Vol. 147, pp195-197.

Trillium Software, (2004) How data profiling & analysis saves companies $millions, white paper in *Data Integration and Data Quality Management*.

Ukkonen E., (1992) Approximate string-matching with q-grams and maximal matches, *Journal of Computer Science 92*, pp. 191-211.

Winkler W. E., (1990) String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage, Proceedings of the Section on Survey Research Methods, *American Statistical Association*, 1990, pp:354-359.