UDC:004.657
Original scientific paperr

INDEPENDENT DE-DUPLICATION IN DATA CLEANING [#]

Ajumobi Udechukwu[1], Christie Ezeife[2], Ken Barker[3]
[1,3] Dept. of Computer Science, University of Calgary, Canada
{ajumobiu, barker}@cpsc.ucalgary.ca

[2]School of Computer Science, University of Windsor, Canada
cezeife@cs.uwindsor.ca

Abstract: *Many organizations collect large amounts of data to support their business and decision-making processes. The data originate from a variety of sources that may have inherent data-quality problems. These problems become more pronounced when heterogeneous data sources are integrated (for example, in data warehouses). A major problem that arises from integrating different databases is the existence of duplicates. The challenge of de-duplication is identifying "equivalent" records within the database. Most published research in de-duplication propose techniques that rely heavily on domain knowledge. A few others propose solutions that are partially domain-independent. This paper identifies two levels of domain-independence in de-duplication namely: domain-independence at the attribute level, and domain-independence at the record level. The paper then proposes a positional algorithm that achieves domain-independent de-duplication at the attribute level, and a technique for field weighting by data profiling, which, when used with the positional algorithm, achieves domain-independence at the record level. Experiments show that the proposed techniques achieve more accurate de-duplication than the existing algorithms.*

Keywords: *Data cleaning, De-duplication, data quality, field-matching, record linkage.*

1. INTRODUCTION

Data cleaning deals with detecting and removing errors and inconsistencies from data in order to improve the quality of data [28]. There are data quality problems existing in single data collections, such as files and databases. When these stand-alone sources are integrated (e.g., in data warehouses, federated database systems or global web-based information systems), the data quality problems are inadvertently escalated. Data quality problems (also referred to as "dirty data") could be as a result of different data formats within distinct sources, data entry errors, or the unavailability of common keys among the different data sources to aid integration. Examples of dirty data include the following:

[#] An earlier, abridged version of this work was presented at the 5th International Conference on Enterprise Information Systems (ICEIS) 2003, with the title "Data position and profiling in domain-independent warehouse cleaning".

- Spelling, phonetic and typing errors (e.g., typing "Smit" instead of "Smith", or "Karl" instead of "Carl")
- Word transpositions (e.g., in a free-form text field for address, there could be two representations of the same address as follows: "Sunset Ave., 401" and "401 Sunset Ave.")
- Extra words (e.g., a name field may have the following two representations for the same name: "Miss Alice Smith" and "Alice Smith")
- Missing data (i.e., fields with null entries)
- Inconsistent values (e.g., having a character entry in a number field)
- Multiple values in a single free-form field (e.g., if a field for cities is free-form, there may be entries of the form "Austin Texas" in that field)
- Mis-fielded values (e.g., entering the last-name in the first-name field, and vice versa)
- Illegal values (e.g., having 13 in a month field, or 200 in a student-age field)
- Synonyms and nicknames (e.g., using both "csc" and "compsci" to represent "Computer Science" in different records of the data)
- Abbreviations, truncation and initials (e.g., entering "IBM" for "International Business Machines" or "CalTech" for "California Institute of Technology")
- Data that do not obey functional dependencies and referential integrity

A number of the instances listed above could be solved by using data integrity checks (e.g., having a minimum and a maximum possible data value), and by using more structured schemas (e.g., avoiding free-form fields and null values whenever possible). Others (e.g., spelling, phonetic and typing errors, abbreviations, etc), however, can only be solved by explicit cleaning routines. A major consequence of dirty data is the existence of duplicates (i.e., multiple entries in the database – stand-alone or integrated, referring to the same real world entity). The removal of duplicates constitutes a major cleaning task. Duplicate elimination is also referred to as record linkage or record matching in research literature [35, 36]. The word *"record"* is used to refer to a syntactic representation of some real-world object, such as a tuple in a relational database. The duplicate elimination problem arises whenever records that are not identical, in a bit-by-bit sense, or in a primary key value sense, may refer to the same real life object [21]. Duplicate records may exist in single-source data tables, or may result from the integration of multiple "clean" data tables. Several approaches to de-duplication have been proposed in the research literature. Most of these approaches are tailored to specific application domains (e.g., mailing lists in the USA). Domain-specific techniques rely of in-depth domain knowledge. Such knowledge may not always be available or sufficient for quality data cleaning, thus the need for reliable domain-independent approaches. Domain-independent approaches also result in data cleaning tools that can be easily reused.

1.1 RELATED WORK

Almost all published work on duplicate elimination emphasize specific application domains, and propose algorithms that directly handle particular domains [1, 11, 12, 13, 14, 19, 22, 24, 25, 35, 36,]. For example, some papers discuss duplicate elimination for customer addresses, census records, or variant entries in a lexicon. Other works on duplicate elimination are not strictly domain specific, but assume that domain-specific knowledge will be supplied by a human for each application domain [17]. Some other techniques aim to improve the domain-specific approaches by using sample labeled sets of duplicates, based on domain knowledge, for training the de-duplication function [2, 16, 32, 33]. Thus, duplicate elimination algorithms vary by the amount of domain specific knowledge that they use.

The work by Monge and Elkan [23, 26] and Monge [20, 21] is quite related to the research presented in this paper. They explore the possibility of cleaning datasets without having a complete knowledge of the source domains. Their work addresses domain-independence at the attribute level, and uses equal weights for all fields at the record level. Such an approach however does not effectively address de-duplication at the record level. The techniques proposed in our work achieve domain-independent de-duplication at both the attribute and record levels. Our technique also achieves better accuracy in de-duplication than earlier approaches.

De-duplication is usually supported by strategies and frameworks aimed at bringing likely duplicates together to ensure that they are compared with each other. A naïve strategy may be to compare each record with every other record in the dataset. The records may also be sorted in some way, then neighboring records compared, as in [3, 27]. Some authors have also proposed alternative strategies to improve the comparison process. For example, the sorted neighborhood method [13], the priority queue [26], and the adaptive filtering technique [10]. These strategies are applicable for both domain-specific and domain-independent approaches to de-duplication, and are not the subject of this paper. Some other researchers on data cleaning have focused on how best to integrate the user into the data cleaning process [5, 6, 7, 8, 9, 29, 30, 34]. The emphasis of the research reported in these works is improving the usability of data cleaning systems. User interaction with data cleaning processes is however not the focus of the work presented in this paper.

1.2 RESEARCH CONTRIBUTIONS

This paper proposes effective techniques to achieve domain-independence in de-duplication at both the attribute and record levels. Earlier approaches to domain-independent de-duplication only address attribute level matching (and require user input for record level matching). Secondly, this paper proposes a field-matching algorithm that incorporates the positional ordering of characters and words. Experiments show that the proposed positional technique improves on the accuracy of match results when compared to existing techniques. The rest of this paper is organized as follows. Section 2 presents the proposed de-duplication scheme. In section 3, the accuracy of the proposed scheme is experimentally evaluated. Conclusions are drawn in section 4.

2. THE DE-DUPLICATION SCHEME

Two levels of domain independence are identified in this work, namely: domain-independence at the attribute level, and domain-independence at the record level. Individual attributes of a record constitute data domains that could be cleaned differently. For example, a domain-specific algorithm could be developed to determine when two entries in the age field are equivalent or differ slightly. This could also be done for the other fields in the record. The work by Hernandez and Stolfo [12, 13] uses this approach. Monge and Elkan [23] and Monge [20] propose algorithms that could be used to determine field matches in two records irrespective of the domain of the field (i.e., the same algorithm can be used for the age field, the address field, the name field, etc). Domain independent approaches also are not affected by the inherent characteristics of the data sources (e.g., USA and European addresses can be cleaned with the same techniques without changing parameters). Experiments done by Monge in [20] show that the domain-independent techniques perform as well as the domain-specific techniques in identifying duplicates in the database. The author also experimented the use of classical string matching functions (e.g., edit distance and Smith-Waterman algorithm [31]) for field matching. Classical edit distance functions are however not well suited for field matching in data cleaning because they do not handle acronyms and abbreviations well. For example, the edit distance between "dept" and "department" is 6 (i.e., the number of insertions, deletions, substitutions, etc. required to transform one string into the other). The edit distance between "dept" and "teacher" is also 6, which means that a threshold distance that returns "dept" and "department" as a match will also match "dept" and "teacher". This scenario generally leads to a high level of false matches when classical string matching functions are employed for de-duplication purposes in databases. The experiments by Monge [20] confirm the shortcomings of the classical string matching functions, and show that the recursive algorithm returns more accurate results than earlier techniques. In this paper, we propose a positional algorithm that improves on the results obtained with the recursive algorithm, and also reduces the number of false positives that could have resulted if classical textual similarity functions (such as edit distance) are used.

The second level of domain independence is at the record level. The challenge here is identifying the fields in the database schema that should be given higher preference in determining matching records. For example, the last name attribute could easily be seen as being more important than the gender (sex) attribute in determining if two records in a given database table match. The approach used in commercial data cleaning packages and in most published research, is to have the user or a domain expert specify the levels of importance of the fields in the database. In cases where the packages or algorithms were developed for specific application domains (e.g., names and addresses within the United States), the field importance is hard-coded in the algorithms. In [20], the scheme assigns equal weights to all the fields in the records being compared (except where any of the fields has a null entry, in which case that field is not counted as part of the fields in the records being compared). Such assignment of importance has its obvious shortcomings. Our work proposes a novel, intuitive approach to assign levels of importance (or weights) to the individual fields of the data record for the purpose of de-duplication, thus achieving domain independence at the record level (i.e., the technique will work irrespective of the database schema or the data stored in the data tables). To the best of the authors' knowledge, no previous research had proposed an approach for assigning weights to the fields of the record for the purpose of duplicate elimination.

2.1. POSITIONAL ALGORITHM

The positional algorithm is an enhancement to the recursive algorithm [20, 23, 25]. The basic idea used in the recursive algorithm (with character base) is to locate the characters that make up one entry in a field in the second entry. Thus, if there are two strings "dept" and "department" in two entries within a field, the algorithm checks for the existence of the characters that make up "dept" in "department". For this example, the result is an exact match, which is correct. However, there are instances where this approach fails. For example, given two strings which are entries in the name field of a database:

A = " John Johnson"

B = " Johns Smith"

Using the recursive algorithm with character base, the following steps would be taken to determine if the two strings are duplicates in the database.

1. Tokenize the strings. The resulting tokens are:
   A = {“John” “Johnson”}
   B = {“Johns” “Smith”}
2. Break up each token into its individual atomic characters, as follows:
   A = {“ ‘j’, ‘o’, ‘h’, ‘n’ ” “ ‘j’, ‘o’, ‘h’, ‘n’, ‘s’, ‘o’, ‘n’ ”}
   B = {“ ‘j’, ‘o’, ‘h’, ‘n’, ‘s’ ” “ ‘s’, ‘m’, ‘i’, ‘t’, ‘h’ ”}
3. Take each token in string ‘A’ and compare with each token in string ‘B’. In each case, check for the constituent atomic characters in one token that are contained in the second token. The score is assigned as the ratio of the discovered characters to the length of the word. The highest score achieved for each word in string ‘A’ is taken to be the match score for that word.

   In the example being used, “john” from string ‘A’ is compared to “johns” in string ‘B’ with a score of 1. Also, “john” from string ‘A’ is compared to “smith” in string ‘B’ with a score of 0.25. The higher score, which is 1, is taken as the match score for “john” in string ‘A’. Going by the recursive algorithm, the second token in string ‘A’, which is “johnson”, is compared with all the tokens in string ‘B’. First, “johnson” is compared with “johns”, resulting in a score of 1, then “johnson” is compared with “smith” with a score of 0.29. The higher of the scores, which is 1, is taken as the match score for this token. Taking the average of the match scores for string ‘A’, a match of 1 (exact match) is arrived at in error.

   Such errors can also occur at the word-match level, for example, given two strings:

   A = “Tims”   and   B = “Smith”

   Going by the recursive algorithm with character base, a match is found if all the characters that constitute a token in one string are found in the second string. In this example, all the individual characters that constitute string ‘B’ can be found in string ‘A’, thus an exact match is declared in error.

The recursive algorithm may also be implemented with a word base. For the word-base case, the first shortcoming discussed above is still present (i.e., when A = {“John” “Johnson”} and B = {“Johns” “Smith”}). The second shortcoming is not encountered because the words are not broken into characters. However, the recursive algorithm with word base is very rigid and does not tolerate errors in words (i.e., permits only exact word-matches and prefixes). The rigidity in the word-base recursive algorithm may be addressed with classical string similarity functions (e.g., edit distance or Smith-Waterman algorithm). However, as discussed earlier in section 2, these classical functions result in high levels of false-positive matches. The positional algorithm overcomes the shortcomings of the recursive algorithm. The positional algorithm tolerates errors in words while maintaining high accuracy levels, and also supports the matching of acronyms and abbreviations. The steps in the positional algorithm are detailed below.

The steps in the positional algorithm (for field and word matching) are outlined below:

Step 1: Tokenize the two strings (or fields) being compared into words. Denote the string with fewer word-tokens as the first string, and the other string as the second.

Step 2: If one of the strings has only one word-token and the other string has more than one word-token (e.g., A = "IBM" and B = {"International", "Business", "Machines"}), then, break up the word-token in the string with one token into its constituent atomic characters (i.e., A = {"I", "B", "M"}). Compare each atomic character in the string with one word-token, with the first character of the word-tokens in the second string in order. Declare a match if all the atomic characters in the first string are matched with the first characters of the tokens in the second string. End if there is a match (i.e., skip steps 3 and 4).

Step 3: Starting with the first word-token in the first string (the first string is assigned as the string with less word-tokens), compare each word-token in the first string to the word-tokens in the second string. (The technique for comparing two words is discussed in step 4 below). If a match is found (i.e., the threshold is reached) between any two word-tokens being compared (i.e., one word-token from each of the two strings), a match score of 1 is assigned to the word-token, the comparison is stopped and the token position in the second string is marked as the last match position and is un-matchable for subsequent matches, and subsequent word-tokens from the first string are compared first with word-tokens in the second string that are positioned to the right of the matched position, and subsequently the rest of the word-tokens if a match is not found right of the last matched position.

Step 4: How to match two words: At the word token level, the match problem is reduced to a search of the constituent characters of the shorter word token in the longer word token. A match score of the word is returned as the total match score for all characters in the shorter word divided by the number of characters in the shorter word. The characters in the shorter string are searched for in the longer string one after the other. If the length of the shorter word is less than or equal to half the length of the longer string, then the first characters of both words must match for the process to continue. Once a match of a character in the first word is found in the second word, the position of the last character that got a match is noted in the second word. For example, to match the words "John" and "Johns", once the 'J' in the first word, "John" matches the 'J' in the second word, "Johns", position 1 in the second word is noted so that the next search for the following character 'o' will begin at position 2 and not start again from the beginning. Thus, the last matched position is used to indicate the start position (the first character right of the matched position) and the end position (the last character left of the matched position) of the next search. The position tracking is also used to charge penalties for characters that appear out of order or with a gap because other non-target characters are separating them from previously matched characters. If a character matches, a score of 1 is added to the match score of the word being searched for. However, a disorder penalty of –1 is charged for each character disorder encountered in the match. For example, in matching the word "tims" with the word "smith", character 't' matches the 4th character of the second word, but the next character 'i' in the first word is found not right of the character 't' in the second word, but, left of it, causing a position disorder penalty of –1 to be charged. The effective score when such a disorder occurs is 0. Finally, if there is a gap between two matches of the constituent characters, a gap penalty of –0.2 is charged in addition, for each gap in the first set of gaps. The penalty charge for each gap doubles every time a new set of gaps is encountered. For example, in matching the two words "dean" and "department", the first two characters 'd' and 'e' in "dean" are matched with a match score of 2 (i.e., 1 + 1). However, the third character 'a' in "dean" is not the third character in “department” as there is a character gap (the first set for this word) between the last matched character ‘e’, and ‘a’ leading to a gap penalty charge of –0.2 for the one gap. Also, the last character ‘n’ is four characters away from the last matched character ‘a’ and this is the second set of gaps. Thus, the penalty charged to each of these four gaps is double the previous penalty (in this case –0.2). The total penalty for the four gaps is –0.2 * 2 * 4 = -1.6. This means that the total match score for the matched word “dean” is (4 (from the 4 characters found) + 0 (no disorder penalties charged) + (-0.2 – 1.6) (gap penalties))/4 = 2.2/4 = 0.55. The decision as to whether any match score is considered a match is dependent on a word match threshold assigned by the user. A score lower than match threshold returns a “no match” result for the word with a word match score of 0, the word match score is 1 otherwise. The penalty is set at –0.2 for each character space in the first occurrence of a gap, and is doubled each time a new set of gaps is encountered.

2.1.1 Example Matching of Fields and Words in Database Records

Example 1: Are the string field values A and B given below, from the *Name* field of a database table, duplicates, assuming word and field match thresholds of 0.75?

A = “John Johnson” and B = “Johns Smith”

The positional algorithm would first break each *Name* field value into its constituent word tokens. The field with fewer word tokens is designated the shorter string and each word token of the shorter string is matched with the word tokens in the second string to obtain a match score for the word token. The total match score of the entire field value is finally obtained as the average of the word match scores of its word tokens. The two field values are declared a match only if this field-match score is greater than or equal to a field match threshold. Thus, the positional field match algorithm will call the positional word match function to compare “John” in string A with “Johns” in string B. At the word comparison level, the constituent characters in the shorter word are compared with those of the longer word with match scores assigned, last match position remembered, and disorder as well as gap penalties assigned where necessary. Now, comparing the word “John” with the word “Johns” yields a character match score of 1 for ‘J’, 1 for ‘o’, 1 for ‘h’, and 1 for ‘n’. The average word match score for this word then is 4/4 = 1and since this is higher than the word match threshold of 0.75, this is declared a match and a match score of 1 is recorded for the word “John”. Since there is a match, all subsequent comparisons would not involve “Johns” in string B. Thus, “Johnson” in string A is never compared with “Johns” in string B. Next, “Johnson” in string A is compared with “Smith” in string B because the last match position is at the first word, which is now declared unmatchable for subsequent comparisons. A match score of 0 is returned for the word “Johnson” because it does not achieve the word match threshold with the word “Smith”. This now means that the overall field match score for string A is (1+0)/2 = 0.5. Since this is lower than the field match threshold of 0.75, the decision returned by the positional field matching function for whether the strings A and B are duplicates is false. The differences between the positional algorithm and the recursive algorithm are that (1) position of character and word tokens are remembered and used to disallow re-matching characters/words that had participated in previous matches, (2) positions are used to charge disorder and gap penalties to matched words. These differences improve the accuracy of the positional algorithm.

Example 2: Are the string word values A and B given below, from the *Name* field of a database table, duplicates, assuming a word match threshold of 0.75?

A = "tims" and B = "smith"

The search problem would be to locate the characters of the shorter string in the longer string; thus, the constituent characters in "tims" would be searched for in "smith". The search locates 't' in the 4th position in "smith", a score of 1 is given, 't' is marked as already matched, and the match position recorded. The next character is 'i', and it is searched for in character positions right of 't' in "smith". In this case, there is no 'i' right of 't', so the search wraps around to the first character of "smith". The character 'i' is then located at the 3rd position in "smith", a score of 1 is assigned for matching, but a disorder penalty of –1 is charged because the character is found left of the previous match position of 4. The current match position recorded for further searches (i.e., the match position becomes the last match position referenced to obtain the start position for the next search). The characters 'm' and 's' are also searched for, each yielding a net score of 0 due to positional disorder relative to the most recent match position. Finally, although all the characters in "tims" were located in "smith", only 't' got a score of 1. Thus, total score for the token is 1 divided by the number of characters (i.e., 4) in the token (in this case, the token score is 0.25) indicating a "no match" answer for the two words.

The examples above demonstrate how duplicates are detected at the word and field levels. At the record level, the match-score for any given pair of records is computed as the sum of the products of the field-match scores and the field weights for the selected fields. The sum of the assigned field weights must be equal to 1, and the field-match scores range between 0 and 1, thus the resultant sum for the record-match score would always range between 0 and 1 (where 0 depicts a no-match and 1 depicts a perfect match). The positional algorithm (just like the recursive algorithm) has quadratic time complexities $O(nm)$ at both the word-match level and the field-match level, where n and m are the number of tokens (characters or words respectively) in the strings. At the record-match level, the recursive algorithm also has a quadratic time complexity $O(n^2)$, where n is the number of fields in each record. This is because each field in one record is matched with every field in the second record. The positional algorithm however compares only corresponding entries in the selected fields (with assigned weights); thus, the number of fields in the records being compared does not affect its time complexity. The result is an $O(k)$ time complexity at the record level, where k is the number of fields with assigned weights. Data profiling is used to select the fields that participate in record matching, as well as assign weights (or importance) to the selected fields. Details of the data profiling technique proposed in this work are discussed in section 2.2.

2.2. ALGORITHM FOR ASSIGNING FIELD IMPORTANCE USING DATA PROFILING

Data profiling is simply the characterization of data through the collection and analysis of data content statistics [18]. The scheme used in this work is based on the discriminating power of the attributes of the records in a database to uniquely identify the individual records of the database. The technique gathers data content statistics from a subset of the data table to be cleaned, and uses the information to assign weights to the attributes of the data table. Thus, the technique adapts well to changing data domains. The scheme is outlined in the following steps:

1. Given a database table of *N* records, where *N* is a large number, select a subset of the table (*n* records) and collect the following statistics from the data contained in the

fields of the *n* records: uniqueness, presence of null values, and field length. The major characteristic of interest here is uniqueness. Uniqueness measures the percentage of the data contained in each of the attributes of the n records that are without repetition. In determining the uniqueness factor, each field in the *n* subsets of the database table is grouped into clusters. Two entries are merged into a cluster if they are exact matches or one is a prefix of the other. Also, fields with lengths greater than 100 are grouped into one cluster. The ratio of the number of resulting clusters to the number of records in the subset is the percentage uniqueness (note that one resulting cluster is equivalent to zero uniqueness).

2. After processing the *n*-records subset of the data table, a scheme is used to determine the score of each attribute. This scheme assigns to each field, (1) a uniqueness score between 100 (highly unique) and 0 (not unique), (2) a null value score between -0 (lowest null value presence, no penalty) and -30 (highest null value presence), and (3) a field length score between -0 (short field length) and –30 (long field length with more than 50 characters). The aim is to select fields with a high percentage of unique data entries and low presence of null entries. The scheme also avoids selecting fields with excessive entries (e.g., a memo field). Table 1 shows the scoring scheme used in our work.

3. Given a table with *K* attributes, if all the *K* attributes performed perfectly well in step 2 above, then the total score of all the attributes would be 100**K*. The fields are then ranked in descending order of their scores. The first *l* fields that achieve a total score of (100**K*)/2 (minimum of 2 fields if $K \geq 2$), up to a maximum of 5 fields, are given weights relative to their scores and used in the field-matching algorithm. If all the fields do not achieve a total of at least (100**K*)/2 (i.e., the attributes have a low discriminating score), then the top (*K*/2) + 1 attributes from the ranked list of attributes will be assigned weights relative to their scores and used in the match.

4. If *l* fields are selected from step 3 above, each with a score of *t*, the total score for the selected fields, *T* is then the sum of the t_l's. The weight *w* for each field is then computed as: $w = t/T$

Table 1: Scheme for Scoring Field Weights

Characteristic	Percentage Achieved	Score
Uniqueness	95% - 100%	100
	90% - 94%	90
	80% - 89%	80
	70% - 79%	70
	60% - 69%	60
	50% - 59%	50
	40% - 49%	40
	30% - 39%	30
	20% - 29%	20
	10% - 19%	10
	0% - 9%	0
Null Values	< 5%	- 0
	5% - 10%	- 5
	11% - 20%	- 10
	21% - 30%	- 15
	31% - 40%	- 20
	41% - 50%	- 25

	> 50%	- 30
Field Length (> 50 Characters)	< 5%	- 0
	5% - 10%	- 5
	11% - 20%	- 10
	21% - 30%	- 15
	31% - 40%	- 20
	41% - 50%	- 25
	> 50%	- 30

The major activity in the data-profiling algorithm is the clustering of the entries in each field to determine uniqueness. To determine clusters, each entry in a particular field of the database would be compared to all the previous clusters in that field. In the worst case, all the entries would be unique clusters, resulting in $\frac{n}{2}(n-1)$ comparisons, where n is the number of records in each subset of the data-table selected for profiling. This results in an $O(n^2)$ time complexity for scoring each field in the subset of the database. Given that the data table has k attributes, then since each attribute needs to be scored, the time complexity for scoring each subset of the database would be $O(kn^2)$. Once all the attributes are scored, the other major activity would be the ranking/sorting of the field scores, which has an $O(k \log k)$ time complexity. The dominant cost is thus $O(kn^2)$, and this is the time complexity of profiling each selected subset of the data table. If the algorithm is set up such that the profiling is done m times, then the time complexity for the entire data profiling process would be $O(kmn^2)$. In practical scenarios, n is set as a very small fraction of the entire dataset, such that the profiling phase does not add any significant cost to the entire de-duplication process.

The four steps outlined above must be performed in the first instance before the duplicate elimination process begins. The user can then set an interval after which the data may be profiled again and the scores averaged with the existing scores. The algorithm adapts to the new scores and assigns weights accordingly. For example, given a data table with 100,000 records, we could choose to profile the first 100 records in the first instance. The scores obtained from this first profiling are then used to assign weights to the fields of the database. We can also choose to profile the data after every 20,000 records; thus, in the first pass of the multi-pass merge/purge technique [13] we could profile the data 5 times. Each of the profiling operations updates the scores of the data fields and consequently, the weights and selection of fields that should participate in the matching decision.

3. EXPERIMENTAL EVALUATION

This section presents the results of experimental evaluation of the positional algorithm, recursive algorithm with word-base, and recursive algorithm with character-base. Two data sets were used to test the algorithms for accuracy. The datasets were set up from real-world subscription lists using techniques similar to those used by Hernandez and Stolfo [13] and Monge [20, 21]. The base subscription list consists of 500 unique records, which can generate 124,750 record pairs (i.e., pairing each record with every other record). Subsets of the base list are randomly selected to create the two datasets used in our experiments. Testing for accuracy of de-duplication techniques requires controlled introduction of duplicate records. The errors introduced in the duplicate records vary from small typographical errors to word (and field) transpositions. We draw from popular studies on errors and spelling correction algorithms [4, 15]. The set-up of the experiments and the results achieved for accuracy are discussed in section 3.1.

3.1. EXPERIMENTS FOR ACCURACY

Two datasets were used to test the algorithms for accuracy. The experiments use two measures of accuracy to evaluate the effectiveness of the algorithms in de-duplicating the datasets. The first measure of accuracy used is recall. Recall is the ratio of true matches returned by a matching algorithm to the number of duplicates in the original database. The second measure of accuracy used is precision. Precision measures the ratio of true matches returned by a matching algorithm to the total number of matches returned by the algorithm. Thus, precision gives a measure of the level of false positives in the returned results. The three algorithms were run using threshold values ranging from 0.1 to 1.0. The data tables were set up such that the first column is a unique identifier. This is useful in evaluating the results of the various runs. The results of the experimental runs on the two data sets are discussed in the paragraphs below.

Test Data 1

Test data 1 consists of 7021 pairs of records, with fourteen pairs of duplicated records. The duplicates were set up with acronyms, character substitutions, deletions, insertions, and transpositions, with multiple errors in each duplicated record. There were no exact duplicates in this data set, and each duplicated record was allowed only one duplicate. There were twelve attributes in each record (record id, name, address, city, state/province, postal code, telephone, membership year, level of education, discipline, occupation, and gender). Table 2 presents the results achieved when the three algorithms were run on test data 1. Figures 1 and 2 depict the percentage precision and percentage recall achieved for varying thresholds respectively. Results for test data 1 (Table 2) show that the positional algorithm achieves a higher precision than the other two algorithms for all levels of recall. The highest recall achieved by the recursive algorithm with word base for test data 1 is 87.51%, and its best precision for this threshold is 0.2%. The recursive algorithm with character base achieved a 100% recall, but its best precision at that recall level is 0.22%. Test data 1 was clearly challenging for all three algorithms; however, the positional algorithm outperforms the other two algorithms with respect to accuracy on this dataset.

Table 2: Experimental Results on Test data 1

Threshold	**Positional Algorithm**		**Recursive Algorithm with word-base**		**Recursive Algorithm with character-base**	
	%Recall	%Precision	%Recall	%Precision	%Recall	%Precision
0.1	100	0.20	85.71	0.20	100	0.20
0.2	100	0.20	50	0.14	100	0.20
0.3	100	0.26	50	0.25	100	0.20
0.4	100	2.89	42.86	0.47	100	0.20
0.5	100	15.73	28.57	0.87	100	0.20
0.6	100	50	14.29	2.15	100	0.22
0.7	42.86	37.5	14.29	9.52	92.86	0.29
0.8	14.29	50	14.29	33.33	57.14	0.86
0.9	0	N/A	0	N/A	21.43	25
1.0	0	N/A	0	N/A	0	N/A

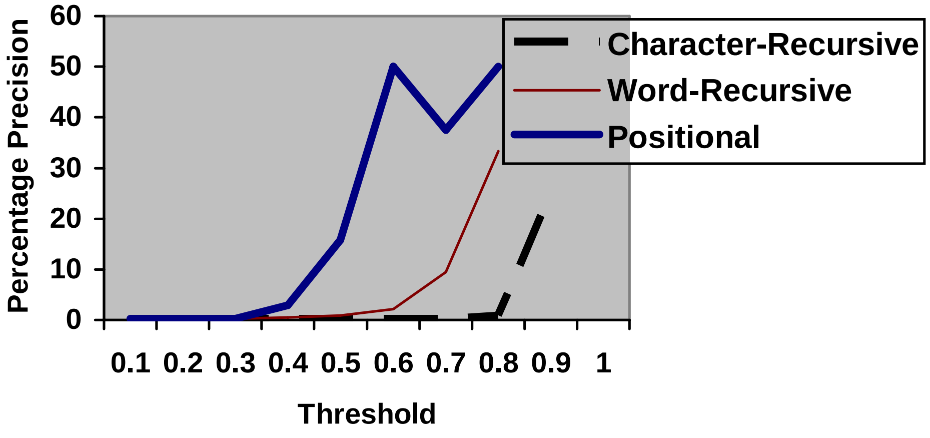

Figure 1: Percentage Precision versus Threshold on Test data 1

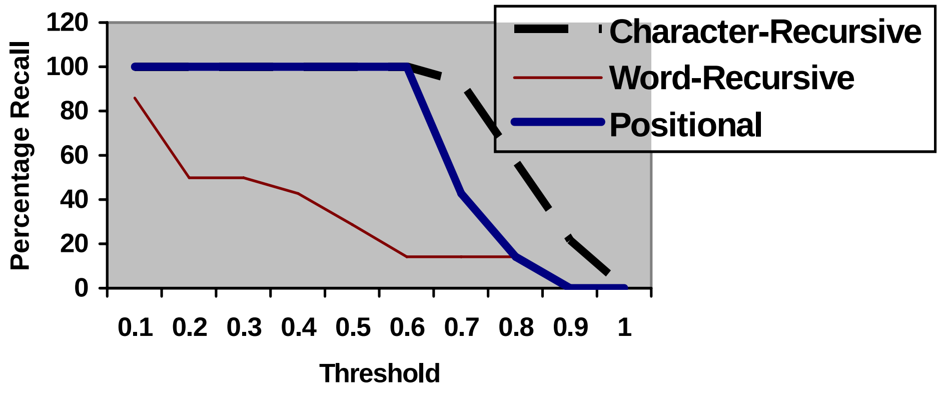

Figure 2: Percentage Recall versus Threshold on Test data 1

Test Data 2

Test data 2 consists of 9045 pairs of records, with fourteen pairs of duplicated records. The duplicates in this case were developed with word transpositions, deletions, substitutions, insertions, as well as exact duplicates, thus it is expected that a hundred percent precision can be achieved at a recall that is higher than zero. Each record in the dataset consists of two fields (record id and name). The record-id field was not used in any of the matching algorithms. Its function was in the evaluation of match results. This data set was set up especially to evaluate the field-matching effectiveness of the various algorithms irrespective of other constraints such as field weights. The results from running the three algorithms on this data set are presented in Table 3. Figures 3 and 4 depict the percentage precision and percentage recall achieved for varying thresholds respectively. The three algorithms performed better with test data 2 than with test data 1. The positional algorithm however, still outperforms the other two algorithms on accuracy, achieving a higher precision for every level of recall. As with test data 1, the recursive algorithm with word base failed to attain a 100% recall, achieving a maximum recall of 71.43%. The best precision achieved at this level of recall for the recursive algorithm with word base is 6.21%. The recursive algorithm with character base achieved 100% recall levels, but with much lower precision than the other two algorithms. The best precision achieved by the recursive algorithm with character base at 100% recall is 0.47%.

Table 3: Experimental Results on Test data 2

Threshold	Positional Algorithm		Recursive Algorithm with word-base		Recursive Algorithm with character-base	
	%Recall	%Precision	%Recall	%Precision	%Recall	%Precision
0.1	100	0.17	71.43	6.17	100	0.16
0.2	100	0.19	71.43	6.17	100	0.17
0.3	100	0.31	71.43	6.21	100	0.20
0.4	100	0.53	64.29	8.26	100	0.28
0.5	100	1.06	64.29	8.26	100	0.47
0.6	100	77.78	14.29	28.57	85.71	0.92
0.7	100	87.5	7.14	16.67	71.43	2.11
0.8	71.43	100	7.14	16.67	71.43	8.40
0.9	35.71	100	7.14	16.67	42.86	35.29
1.0	21.43	100	7.14	16.67	14.29	33.33

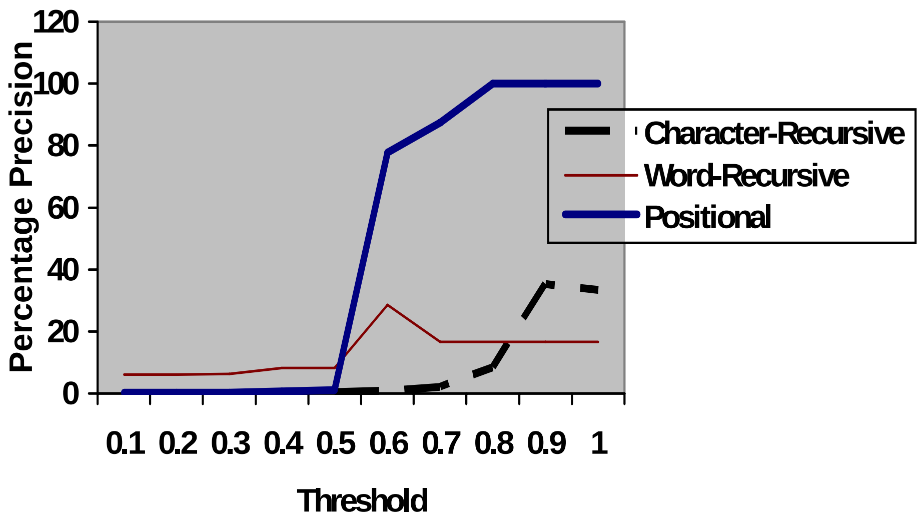

Figure 3: Percentage Precision versus Threshold on Test data 2

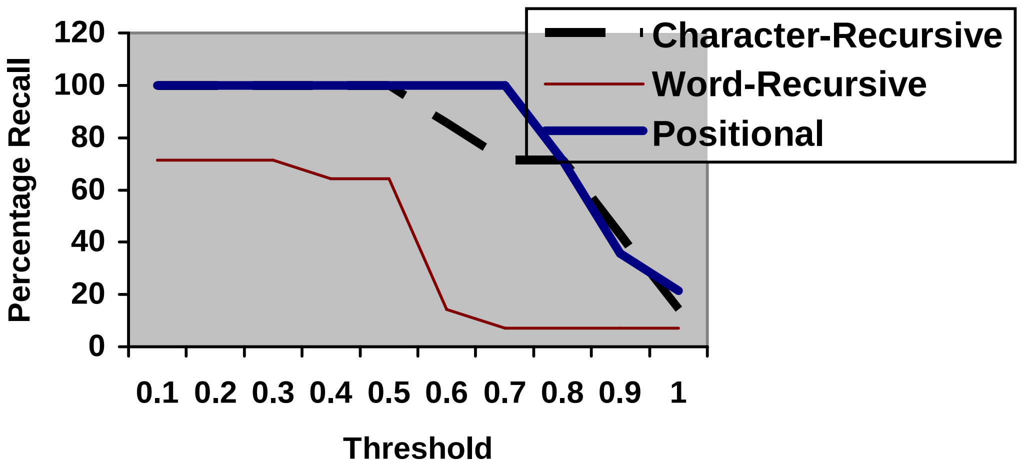

Figure 4: Percentage Recall versus Threshold on Test data 2

4. CONCLUSIONS

This paper contributes a wholesome treatment of domain-independent de-duplication in data cleaning. Domain-independent approaches to data cleaning present an interesting and worthwhile field of research. The domain-independent approaches reduce repetitive efforts required in data cleaning operations, and from a Software Engineering perspective, offer reuse and standardization of data cleaning tools. The technique proposed in this work achieves domain-independence at the field and record levels, and is shown by experiments to generate more accurate results than existing approaches to domain-independent de-duplication.

The concept of domain independence used in this work applies only to data from the unicode character set. An area of future research is to extend the domains covered to include image data, spatial/map data, biological data, *etc*.

REFERENCES

[1] R. Ananthakrishna, S. Chaudhuri, V. Ganti, Eliminating Fuzzy Duplicates in Data Warehouses, Proc. of the 28th VLDB Conference, Hong Kong, China, 2002.

[2] M. Bilenko, R.J. Mooney, Adaptive Duplicate Detection Using Learnable String Similarity Measures, Proc. of the ACM SIGKDD'03, pages 39 – 48, August 2003.

[3] D. Bitton, D.J. DeWitt, Duplicate Record Elimination in Large Data Files, ACM Transactions on Database Systems, 8(2): 255 – 65, 1983

[4] K.W. Church, W.A. Gale, Probability Scoring for Spelling Correction, Statistics and Computing, 1, pp. 93-103, 1991

[5] H. Galhardas, D. Florescu, D. Shasha, E. Simon, Declaratively Cleaning your Data using AJAX, In Journees Bases de Donnees, Oct. 2000.

[6] H. Galhardas, D. Florescu, D. Shasha, E. Simon, AJAX: An Extensible Data Cleaning Tool, Proc. ACM SIGMOD Conference, page 590, 2000.

[7] H. Galhardas, D. Florescu, D. Shasha, E. Simon, An Extensible Framework for Data Cleaning, In Proceedings of the International Conference on Data Engineering

(ICDE), San Diego, CA, pages 312 – 344, 2000

[8] H. Galhardas, D. Florescu, D. Shasha, E. Simon, C. Saita, Improving Data Cleaning Quality using a Data Lineage Facility, In Proc. of the Inter. Workshop on Design and Management of Data Warehouses (DMDW'2001), pages 3 – 16, 2001.

[9] H. Galhardas, D. Florescu, D. Shasha, E. Simon, C. Saita, Declarative Data Cleaning: Language, Model, and Algorithms, Proceedings of the 27th VLDB Conference, Roma, Italy, pages 371 – 380, 2001.

[10] L. Gu, R. Baxter, Adaptive Filtering for Efficient Record Linkage, SIAM Data Mining Conference, 2004.

[11] M. Hernandez, A Generalization of Band Joins and the Merge/Purge Problem, Ph.D. Thesis, Columbia University, 1996.

[12] M. Hernandez, S. Stolfo, The Merge/Purge Problem for Large Databases, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 127 – 138, May 1995.

[13] M.A. Hernandez, 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] J. Hylton, Identifying and Merging Related Bibliographic Records, Master's Thesis, Massachusetts Institute of Technology, June 1996

[15] K. Kukich, Techniques for Automatically Correcting Words in Text, ACM Computing Surveys, 24(4):377-439, 1992.

[16] M.L. Lee, T.W. Ling, W.L. Low, IntelliClean: A Knowledge-Based Intelligent Data Cleaner, In Proc. ACM SIGKDD'2000, pages 290 – 294, Boston, 2000.

[17] M.L. Lee, H. Lu, T.W. Ling, Y.T. Ko, Cleansing Data for Mining and Warehousing, In Proceedings of the 10th International Conference on Database and Expert Systems Applications (DEXA), pages 751 – 760, 1999.

[18] W. Li, Knowledge Gathering and Matching in Heterogeneous Databases, Working Notes of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, pages 116 – 121, 1995.

[19] J.I. Maletic, A. Marcus, Data Cleansing: Beyond Integrity Analysis, in: Proceeding of The Conference on Information Quality (IQ 2000), pages 200 – 209, October 2000.

[20] A. Monge, Adaptive Detection of Approximately Duplicate Database Records and The Database Integration Approach to Information Discovery, Ph.D. Thesis, Dept. of Comp. Sci. and Eng., Univ. of California, San Diego, 1997

[21] A.E. Monge, Matching Algorithms within a Duplicate Detection System, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, pages 14 – 20, 2000.

[22] A.E. Monge, C.P. Elkan, WebFind: Automatic Retrieval of Scientific Papers over the World Wide Web, In Working Notes of the Fall Symposium on AI Applications in Knowledge Navigation and Retrieval, page 151, AAAI Press, 1995.

[23] A.E. Monge, C.P. Elkan, The Field Matching Problem: Algorithms and Applications, in: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, pages 267 – 270, 1996.

[24] A.E. Monge, C.P. Elkan, Integrating External Information Sources to guide Worldwide Web Information Retrieval, Technical Report CS96 – 474, Dept. of Computer Science and Engineering, Univ. of California, San Diego, January 1996.

[25] A.E. Monge, C.P. Elkan, The WEBFIND Tool for Finding Scientific Papers over the Worldwide Web, In Proc. of the Third Inter. Congress on Computer Science Research, pages 41 – 46, Tijuana, Mexico, 1996.

[26] A.E. Monge, C.P. Elkan, An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records, Proceedings of the SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, pages 23 – 29, Tucson, Arizona, May 1997.

[27] H.B. Newcombe, J.M. Kennedy, S.J. Axford, A.P. James, Automatic Linkage of Vital Records, Science, Vol. 130: 954 – 959, October 1959.

[28] E. Rahm, H.H. Do, Data Cleaning: Problems and Current Approaches, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, pages 3 – 13, 2000.

[29] V. Raman, A. Chou, J.M. Hellerstein, Scalable Spreadsheets for Interactive Data Analysis, ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD), May 1999.

[30] V. Raman, J.M. Hellerstein, Potters Wheel: An Interactive Framework for Data Cleaning and Transformation, Proc. of the International Conference on Very Large Databases (VLDB), pages 381 – 390, Roma, Sept. 2001.

[31] T.F. Smith, M.S. Waterman, Identification of Common Molecular Subsequences, Journal of Molecular Biology, 147:195 – 197, 1981.

[32] S. Sarawagi, A. Bhamidipaty, A. Kirpal, C. Mouli, ALIAS: An Active Learning led Interactive Deduplication System, Proc. of the 28th VLDB Conf., Hong Kong, China, 2002.

[33] S. Tejada, C.A. Knoblock, S. Minton, Learning Domain-Independent String Transformation Weights for High Accuracy Object Identification, Proc. of the 8th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, Edmonton, Alberta, 2002.

[34] P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, T. Sellis, ARKTOS: A Tool for Data Cleaning and Transformation in Data Warehouse Environments, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, pages 42 – 47, 2000.

[35] W.E. Winkler, Advanced Methods for Record Linkage, Technical Report, Statistical Research Division, Washington DC: US Bureau of the Census, 1994.

[36] W.E. Winkler, Matching and Record Linkage, In Brenda G. Cox, editor, Business Survey Methods, pages 355 – 384, Wiley, 1995.

Received: 21 October 2004
Accepted: 21 September 2005