DATA POSITION AND PROFILING IN DOMAIN-INDEPENDENT WAREHOUSE CLEANING

C.I. Ezeife*
School of Computer Science, University of Windsor
Windsor, Ontario, Canada N9B 3P4
cezeife@uwindsor.ca

A. O. Udechukwu,
School of Computer Science, University of Windsor
Windsor, Ontario, Canada N9B 3P4

Key words: Data cleaning, Data quality, De-duplication, Dirty data, Field matching

Abstract: A major problem that arises from integrating different databases is the existence of duplicates. Data cleaning is the process for identifying two or more records within the database, which represent the same real world object (duplicates), so that a unique representation for each object is adopted. Existing data cleaning techniques rely heavily on full or partial domain knowledge.
This paper proposes a positional algorithm that achieves domain independent de-duplication at the attribute level. The paper also proposes a technique for field weighting through data profiling, which, when used with the positional algorithm, achieves domain-independent cleaning at the record level. Experiments show that the positional algorithm achieves more accurate de-duplication than existing algorithms.

1 Introduction

Data warehousing provides a way for integrating various source databases originally stored as relational databases, flat files, HTML documents and knowledge bases, for decision support querying (Widom, 1995; Han and Kamber, 2000). Thus, data warehouses store historical, integrated, subject-oriented, and summarized data of an establishment for online analytical processing (OLAP), decision support systems (DSS), and data mining. A data warehouse schema based on the star schema consists of a central table called the fact table and a number of dimension tables. The fact table holds the integrated, historical and subject-oriented data from a number of source data sources with a measure aggregate attribute of interest. Since the subjects of interest appear as foreign keys in the fact table, the dimension tables provide further descriptive attributes for each foreign key subject attribute in the fact table. Building such a data warehouse involves extraction, transformation and loading of data.

Extraction of data from the operational source databases involves changing the source data into the target structure (schema) of the warehouse. Transformation of data involves cleaning data from different data sources and integrating them. Loading of data entails transferring data to operational data store, where further cleaning of dirt due to integration is carried out before uploading to the data warehouse.

Data cleaning, also called data cleansing or scrubbing is the process of detecting and removing errors and inconsistencies from data in order to improve the quality of data (Rahm and Do, 2000). Two main causes of "Dirt" or conflicts in data are *synonyms and homonyms*, though there are many others such as: "incomplete, missing or null values", "spelling, phonetic or typing errors", "Mis-fielding" (e.g., a country's name in a state/province field), "noise or contradicting entry", i.e., values outside the accepted range (e.g., **31**/9/99), "scanning errors", (e.g., alphabetic "I" instead of numeric "1" and vice versa), "type mismatch". While all other causes of data dirt can be as a result of "oversight" or "human errors", synonyms and homonyms are not. For example, a document collection center in a unit of an organization may decide to use entity acronyms/abbreviations, "ACM", while another center may write it in full, as "Association for Computing Machinery". Homonymous dirt arises when the same "term" or "expression" refers to two or more entities, e.g., many occurrences of "John Smith" in a data source may refer to different persons. A ma-

*This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134) and a University of Windsor grant.

jor consequence of dirty data is the existence of duplicates (i.e., multiple entries in the database – standalone or integrated, referring to the same real world entity). The removal of duplicates constitutes a major cleaning task, and is very essential in data warehouses, which, constantly load and refresh very large amounts of data from heterogeneous sources such that the probability of source and integrated data containing "dirty data" is high. Research on the subject is tending towards greater automation of the cleaning process, thus, data cleaning should be supported by tools, which limit manual inspection and programming effort, and which can be easily extended to handle new data sources.

1.1 Related Work

Hernandez and Stolfo in (Hernandez and Stolfo, 1998) and their earlier work propose a domain-independent data cleaning framework, which utilizes an equational theory for matching and removing duplicates. In (Monge and Elkan, 1996a), Monge and Elkan discuss three domain-independent algorithms for determining duplicates within records. The three algorithms are basic algorithm (a simpler version of the recursive), recursive algorithm, and an adaptation of the Smith-Waterman algorithm (Smith and Waterman, 1981). If words are used as the base case for the recursion, then two words are matched if they are equivalent or one abbreviates the other. In the experiments by Monge and Elkan in (Monge and Elkan, 1996b), the recursive algorithm performed better in precision measures (i.e., a lower percentage of false positives) than the adaptation of the Smith-Waterman algorithm. The recursive algorithm with word-base is however intolerant to errors in words, while the character base algorithm has high level of false positives. The work presented in (Lee et al., 1999) introduces the idea of assigning weights to the fields of the record for the purpose of duplicate elimination.

1.2 Contributions

This paper addresses the problem of duplicate elimination in data-warehouse tables, by proposing a positional algorithm, which is an enhancement to the recursive field-matching algorithm presented in (Monge and Elkan, 1996a), that produces a more accurate result than that in (Monge and Elkan, 1996a). The paper also proposes a scheme for assigning weights (or importance) to the attributes of records in a data table independent of the problem domain, by using data characteristics collected through data profiling. The proposed positional algorithm with weights assigned by the proposed weight assignment scheme, achieves domain independent record level data cleaning. Experiments show that the positional algorithm returns more accurate results than the recursive algorithm.

1.3 Outline of the Paper

The rest of the paper is organized as follows. Section 2 presents the proposed positional algorithm with examples. Section 3 discusses the experimental evaluation of the positional algorithm in comparison with the recursive algorithm, while section 4 finally presents conclusions and future work.

2 The Proposed Positional Algorithm

The problem scope addressed by this algorithm is the elimination of record duplicates in data-warehouse tables. However, the techniques discussed can be used for any de-duplication tasks. The duplicate-elimination problem in data-warehouses can be solved by first eliminating duplicates from the warehouse dimension tables, and then converting all duplicate records in the fact table to their unique de-duplicated version retained in the cleaned dimension tables. Thus, if all the duplicates in all the constituent dimension tables are removed, then cleaning of the data-warehouse is achieved. This paper proposes to solve this problem independent of the domain of the warehouse data.

Two levels of domain independence are identified namely: domain-independence at the attribute level; and domain-independence at the record level. The second level of domain independence identified is domain independence at the record level. The challenge at the record level is identifying the fields in the database schema that should be given higher preference in determining matching records. This paper proposes a premier approach for assigning levels of importance (or weights) to individual fields of the database for purpose of record matching, thus, achieving domain independence at the record level (i.e., the technique will work irrespective of the database schema or data stored in the data tables). The proposed scheme for field weighting is presented in Section 2.3.

2.1 The Main Positional Match-Record Scheme

This scheme contributes a way to decide if any two given records are duplicates or not, and is not concerned with how to bring likely duplicate records in a table close together for record comparisons. Thus, for simplicity of presentation of the proposed method for

identifying any two given duplicate records in a data table, each record in the table is compared with each record below it to decide whether they are duplicates using the positional algorithm. However, any good table level scheme (e.g., (Hernandez and Stolfo, 1998)), for bringing likely duplicates together can be used with the proposed scheme to achieve maximum benefit. Given any data table, the following sequence of steps generates a clean table without duplicate records irrespective of the data domain.

Step 1: Profile the attributes of the data table, assign field weights, and select the fields to use in matching records based on the field weights (using the algorithm discussed in Section 2.3).

Step 2: Starting from the first record, compare each record with all the records below it in the table. The record match function takes in two records and compares the entries in their selected fields for matches using the positional Match-fields function given in Section 2.2. The match score for each of the selected fields is multiplied by the respective field weights to get the field scores. The sum of these field scores is compared to the record threshold to determine if the two records match.

Step 3: Merge identified duplicates in the data table into one unique record.

2.2 Positional Match_Fields and Match_Words Functions

The design aim in developing the positional algorithm is to establish a data-cleaning technique that has both a high recall (i.e., the ability to identify duplicate records, as well as be error tolerant), and a high precision (i.e., the ability to return only correct matches). A framework that would intrinsically handle acronyms is also a design objective. The positional algorithm does not allow multiple matches with the same tokens, unlike the recursive algorithm which takes the highest match score, irrespective of the number of tokens that had been matched to the token earlier. For example, given two strings which are entries in the name field of a database table:

A = " John Johnson"

B = " Johns Smith"

The recursive algorithm would match "John" in string A with "Johns" in string B, and would also match "Johnson" in string A with the same "Johns" in string B. Matches with already matched tokens are disallowed in the positional algorithm. At the word-match level, the scheme used by the positional algorithm copes with errors in words by charging gap penalties. Penalties are also charged for characters with positional disorder. The detailed scheme used in the positional algorithm is discussed below:

Step 1: Tokenize the two strings (or fields) being compared into words such that the string with fewer word tokens is the first string, and the other string is 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"}). If the number of characters in 'A' equals the number of words in 'B', then compare each atomic character in the string having 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 was a match, skipping steps 3 and 4.

Step 3: Starting with the first word-token in the first string, compare each word-token in the first string with the word-tokens in the second string.

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. If the length of the shorter word is less than or equal to half the length of the longer word, 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 "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 marked position is used to indicate the start position (the first character right of the marked position) and the end position (the last character left of the marked 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, if there is a character disorder in the word, a disorder penalty of -1 is charged. 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, but the penalty charge is doubled for each gap in the next subsequent set of gaps. 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 of -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) + (-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 the word match threshold assigned by the user or a threshold algorithm. A match score lower than match threshold returns a 'no match' result for this word with a word match score of 0 for this word token, but a match score of 1 otherwise.

Once a match is found between the first word token (from the first string) and the second word token (from the second string), because the threshold is reached, a match score of 1 is assigned to the word-token, 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 last matched position, and subsequently the rest of the word-tokens to the left of the last matched position if a match is not found right of the last matched position.

2.2.1 Example Matching of Fields and Words of Data Records

Example 1: Are the string values A and B given below, from the *Name* field of a database table, duplicates, assuming a character and word match thresholds of 0.75?
A = "John Johnson"
B = "Johns Smith"
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 short word are compared with those of the long 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 = 1 and since this is higher than the character match threshold of 0.75, this is declared a match and a match score of 1 is recorded for the word "John" at the field level. 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 with the first word, which is now declared unmatchable for subsequent comparisons. A match score of 0 is returned for the word "Johnson" at the field level because it does not match 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 basic 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 that may lead to overall incorrect matches. These differences improve on accuracy reached by the positional algorithm. The following examples will show how the second difference is used to disallow false matches.

Example 2: Are the string 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"
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 4^{th} 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 3^{rd} position in "smith", a score of 1 is assigned for matching, but a disorder penalty of -1 is assigned because the word is found left of the previous match position of 4. The current match position is recorded for further searches (i.e., the match position becomes the last match position referenced as 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 last 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.

Example 3: Are the string values A and B given below, from the *Name* field of a database table, duplicates, assuming a word match threshold of 0.75?
A = "department"
B = "dean"
Penalties are also charged for gaps between matching characters that are in order. The gap penalty is set to -0.2 for each character position in the first set of gaps, and the penalty is set to double the previous penalty (for each character position) for all subsequent sets of gaps. In the two strings given above, all the characters in "dean" are also contained in "department". Thus, a match score of 4 (from 1+1+1+1) is assigned. It is also worth noting that the characters in "dean" are in the same positional order in "department" so penalties for positional disorder will not be effective in this case. The positional algorithm uses the gap penalty to handle instances like this. The first set of gaps is 1 gap between the last matched position 'e' and target character 'a' in the string "department". This causes a gap penalty of (-0.2 * 1). The second set of gaps has 4 gaps between the last matched position 'a' and the new target character 'n', each with a new gap penalty of (-0.2 * 2), which is double the gap penalty of the last set of gaps for this word. The total match score for this word token is then (4 - 0.2 - 1.6)/4 = 0.55.

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).

2.3 Algorithm for Assigning Field Importance

Data profiling is simply the characterization of data through the collection and analysis of data content statistics (Lee et al., 1999). The scheme is based on discriminating power of attributes of records in a database to uniquely identify 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. The technique adapts well to changing data domains. The steps involved in the proposed algorithm for assigning field importance are:
Step 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. Uniqueness measures the percentage of data values contained in each of the attributes of the n records that are without repetition. In determining the uniqueness factor, the n subsets of the database table are sorted and 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).
Step 2: After processing the n-record subset of the data table, a scheme is used to determine the score of each attribute, by adding the values assigned to its uniqueness, null value and field length as determined by their respective computed percentages. This scheme assigns to each profiled field, (1) a uniqueness score between 100 (highly unique for uniqueness percentage value of $95 - 100\%$) and 0 (not unique for $0-9\%$), (2) a null value score between -0 (lowest null value presence for null value percentage less than 5%) and -30 (highest null value presence for greater than 50%), (3) a field length score between -0 (short field length with length percentage less than 5%) and -30 (long field length with more than 50 characters and 50%). The sum of these three scores for each field constitutes the score for the attribute.
Step 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 $(100 * K)/2$, then the top $(K/2) + 1$ attributes from the ranked list of attributes, up to a maximum of 5 fields, will be assigned weights relative to their scores and used in the match.
Step 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: $w = t/T$. The algorithm adapts to 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, and to select the initial sort key.

3 Experimental Evaluation

The positional algorithm overcomes some of the shortcomings of the recursive algorithm by reducing the possibilities of false positives in duplicate matching (that allows errors in words), thus improving the accuracy of the cleaning process. This section presents the results of experimental evaluation of the positional algorithm, recursive algorithm with word-base, and recursive algorithm with character-base. All the experiments were performed on a 1GHz Intel Celeron PC with 256 megabytes main memory, running Windows XP Home Edition. All the programs are written in Java. The data sets were stored as Oracle data tables (on Oracle 8i Personal Edition), and were retrieved using JDBC^{TM}. Two data sets were used to test the algorithms for accuracy. The data sets were set up from real-world subscription lists and the duplicates were introduced to especially increase the probability of false positives in the match results. The set-up of the experiments and the results achieved are discussed in the subsections that follow.

3.1 Experiments for Accuracy

The first measure of accuracy used is recall. Recall is the ratio of correct 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 correct 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 field weights used for the positional algorithm were assigned using the data-profiling scheme discussed in Section 2.3. The data tables were set up such that the first column is a unique identifier. This is useful in evaluating the results of 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 resulted in 7021 pairing 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. Table 1 presents the results achieved when the three algorithms were run on test data 1. Figures 1 and 2 show graphical presentations of the percentage recall and percentage precision achieved for varying thresholds respectively for test data 1. Results for test data 1 show that the positional algorithm achieves a higher precision than the other two algorithms for all levels of recall.

Table 1: Percentage Recall (Rec) and Percentage Precision (Prec) with Test Data 1

Threshold	Algorithms					
	Positional		Word-base		Character	
	Rec	Prec	Rec	Prec	Rec	Prec
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

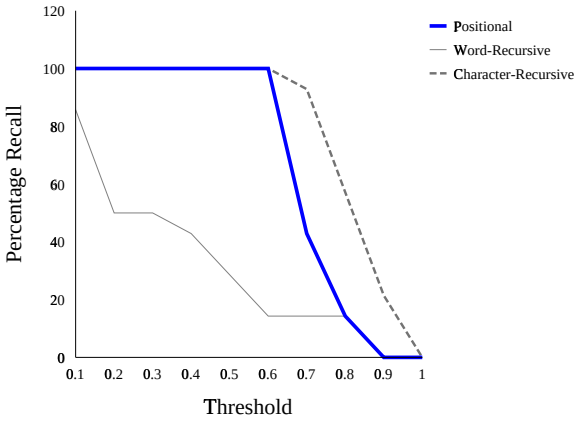

Figure 1: Percentage Recall versus Threshold on Data 1

Test Data 2
Test data 2 resulted in 9045 pairing of records, with fourteen pairs of duplicated records in the one hundred and thirty five records in the sample data set. 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. The results from running the three

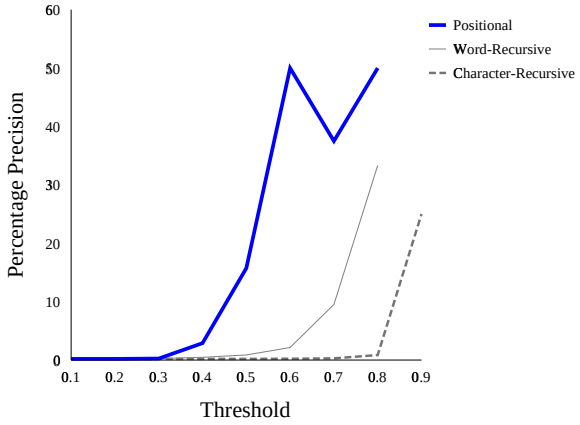

Figure 2: Percentage Precision versus Threshold on Test Data 2

Table 2: Percentage Recall (Rec) and Percentage Precision (Prec) with Test Data 2

Threshold	Algorithms					
	Positional		Word-base		Character-base	
	Rec	Prec	Rec	Prec	Rec	Prec
0.1	100	0.17	71.4	6.17	100	0.16
0.2	100	0.19	71.4	6.17	100	0.17
0.3	100	0.31	71.4	6.21	100	0.20
0.4	100	0.53	64.3	8.26	100	0.28
0.5	100	1.06	64.3	8.26	100	0.47
0.6	100	77.8	14.3	28.6	85.7	0.92
0.7	100	87.5	7.14	16.7	71.4	2.11
0.8	71.4	100	7.14	16.7	71.4	8.40
0.9	35.7	100	7.14	16.7	42.9	35.3
1.0	21.4	100	7.14	16.7	14.3	33.3

algorithms on this data set are presented in Table 2. The positional algorithm achieved 87.5% precision at a 100% 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%.

4 Conclusions and Future Work

This paper contributes an enhancement to the recursive field-matching algorithm for domain-independent field matching an also proposed data-profiling technique for assigning field importance to attributes of a database table. The enhanced approach is shown by experiments, to overcome the major shortcomings of the recursive field-matching algorithm with respect to accuracy. The approach however does not handle cases of field mismatches (for example entering the first name in the last name field and vice versa). Furthermore, the concept of domain independence used here applies only to data from the unicode character set. An area of future research is to extend the domains covered to include image data, biological data.

REFERENCES

Han, J. and Kamber, M. (August 2000). *Data Mining: Concepts and Techniques* . Morgan Kaufmann Publishers., New York, 1st edition.

Hernandez, M. and Stolfo, S. (1998). Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37.

Lee, M., Lu, H., Ling, T., and Ko, Y. (1999). Cleansing Data for Mining and Warehousing. In *Proceedings of the* 10^{th} *International Conference on Database and Expert Systems Applications (DEXA)*, pages 751–760. DEXA press.

Monge, A. and Elkan, C. (1996a). The Field Matching Problem: Algorithms and Applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 267–270.

Monge, A. and Elkan, C. (1996b). The WEBFIND Tool for finding Scientific Papers on the Worldwide Web. In *Proceedings of the Third International Congress on Computer Science Research, Tijuana, Mexico*, pages 41–46. Congress.

Rahm, E. and Do, H. (2000). Data Cleaning: Problems and Current Approaches. In *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, pages 3–13. IEEE press.

Smith, T. and Waterman, M. (1981). Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147:195–197.

Widom, J. (1995). Research Problems in Data Warehousing. In *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM)*. CIKM press.