Proceedings of the ACS/IEEE AICCSA 2003 Conference in Tunisa, Tunis. July 14-17, 2003.

ADD: Arabic Duplicate Detector
A Duplicate Detection Data Cleansing Tool

Ramzi A. Haraty and Ralph Varjabedian
Lebanese American University
P.O. Box 13-5053 Chouran
Beirut, Lebanon 1102 2801
Email: rharaty@lau.edu.lb

Abstract
Data mining is a relatively new term; it was introduced in the 1990s. Data mining is the process of extracting useful information from huge amounts of data. It is sometimes referred to as "data discovery" or "knowledge discovery" in databases [1]. What exactly defines useful information depends on the goal that data mining was for in the first place. Useful information can be used to increase revenue and to cut costs. It can also be used for the purpose of research. Advances in hardware and software in the late 1990s made data centralizing possible. Data centralizing is also called "data warehousing" or "data warehouse for the centralized data". With the process of data centralization came a very important issue, the quality of the data that has been centralized, since centralization includes the joining of multiple data sources. The data given as an input for the data mining process should be of high quality in order for the results of the data mining process to be accurate and reliable. Before data could be mined to extract useful information, it goes through a process called data cleansing. This process is as old as the word "data" itself; however, the term regained significance in the 1990s. Data cleansing involves several steps and processes that include one or more algorithms. This paper addresses one important step, which is duplicate data detection. The paper presents a duplicate detection method called the Efficient *k*-way Sorting Method. The paper also presents a tool called Arabic Duplicate Detection, which is based on our method and is tailored for Arabic data.

Keywords: Arabic data, data cleansing, duplicate detection, and knowledge based-systems.

1. Introduction

Data mining is sometimes known as "data knowledge or discovery". Generally speaking, it is the process of analyzing large amounts of data through different perspectives for the purpose of extracting useful information. Defining what useful information is depends on the goal data mining stood for in the first place. Useful information can be used to increase revenue, cut costs, or it can be used for the purpose of research, as in medical research. Data mining software are tools that are used among other things to analyze and categorize data, discover patterns in data and summarize relationships identified. Technically speaking, data mining is the process of finding correlations or patterns among dozens of fields in large databases.

One U.S. Midwest grocery chain, for example, used the data mining capacity of Oracle software to analyze local buying patterns. The mining process revealed that when men bought diapers on Thursdays and

Saturdays, they also tended to buy beer. Further analysis showed that these shoppers typically did their weekly grocery shopping on Saturdays. On Thursdays, however, they only bought a few items. The retailer concluded that they purchased the beer to have it available for the upcoming weekend. The grocery chain could use this newly discovered information in various ways to increase revenue. For example, they could move the beer display close to the diaper display, and they could sell beer and diapers at full price on Thursdays [2].

A term that frequently appears with data mining is data warehouses. Advances in hardware, as well as software, made data centralizing even more possible. When data is centralized, it is called a data warehouse. Data warehousing is defined as a process of centralized data management and retrieval. Data analysis software on these warehouses is the data mining of the data warehouse.

Sometimes, data mining is not the goal. Database administrators clean the data periodically when the amount of data increases rapidly, as in retail, and when the data added to the database is naturally the kind of data that can decrease the quality of the data in the warehouse.

Duplicate detection is an important step in the data cleansing process. It ensures high quality data. This research is conducted primarily for two main reasons. The first is to build on an existing work in the field and enhance it. The second reason is to show that no substantial research or tools were found in the literature that targets data cleansing of Arabic databases. In this work we present a tool for duplicate detection that deals with Arabic data.

The rest of the paper is organized as follows: section 2 discusses related work. Section 3 presents our algorithm and tool. In section 4, we present experimental results. Section 5 concludes the paper.

2. Related Work

Duplicate detection is the process of removing records in the data warehouse that are syntactically different but semantically the same, i.e. they reference the same real world entity but do not match bit by bit in the data warehouse. Many duplicate detection algorithms exist [3][4][5][6][7][8]. In [3] the authors discuss the sorted neighborhood method, which consists of three steps: create keys, sort data, and merge. The sorted neighborhood method performs poorly in detecting duplicate records in cases where the data warehouse does not contain primary keys. In [4], the authors also discuss a technique for duplicate detection. The technique relies on having a distinguishing field to carry out the detection. Consequently, this scheme also suffers when it works on data repositories where no primary keys are present. In [5] the authors present a method for duplicate detection called the "*k*-way Sorting Method". The method detects data duplicates in repositories that do not have any uniquely distinguishing data fields. The steps of the method are as follows:

1. Let *k* be the number of columns to be used for sorting.
2. Select the *k* most meaningful combinations of sort keys based on the *k* selected columns.
3. Assign a record identifier to each record.
4. Sort the records based on the selected sort key combinations.
5. For each sorted set of data, compare adjacent rows within a given window size. If more than half of the *k* columns used for sorting match, then the records should be considered pair wise matches for that sort.

Repeat this step with all subsequent windows of records until all records have been examined.
6. Draw *k* graphs, one for each sort, with undirected connectors between the record identifiers that were identified to be pair wise matches.
7. Examine the *k* graphs collectively: For all pair wise matches, if the matches occur between the same record identifiers on more than half of the *k* graphs then they should be mapped onto the summation graph. The summation graph should represent all pair wise matches that existed on more than half of the *k* sort graphs.
8. The summation graph should then be handled by computing the transitive closure utilizing the Find/Union processes.

The method requires multiple sorting repetitions of the data repository using a different multiple data sort key for each sort.

In [6] the authors concentrate on an algorithm that is domain-independent. The paper uses a similar method to the *k*-way sorting method", but enhances it by using the Edit_Distance matching method. It also uses the Union/Find structure instead of the graphing method used by [5] to deduce the final set of results.

Three key ideas distinguish the I-Match algorithm presented in [7]. Firstly, a version of Smith-Waterman algorithm for computing minimum distance is used as a domain-independent method to recognize pairs of approximately duplicate records. Secondly, the Union/Find algorithm is used to keep track of clusters of duplicate records incrementally, as pair wise duplicate relationships are discovered. Thirdly, the algorithm uses a priority queue of cluster subsets to respond adaptively to the size and homogeneity of the clusters discovered as the database is scanned.

The main goal of I-Match is to specifically reduce the search engine information and for discovering the information that is duplicates in the engine's database.

In [8] the authors discuss the field matching problem in details. They explain three important techniques used to do smart field matching. The first technique is the "basic field matching algorithm". When two fields are compared, the degree to which two fields match is the number of their matching atomic strings divided by their average number of atomic strings. Two atomic strings match if they are the same string or if one is a prefix of the other. The second technique, which is a slight modification from the first technique, is a recursive field matching algorithm. The third technique discussed is the Smith-Waterman algorithm.

3. Duplicate Detection

In this section we present our algorithm, which is an adaptation of the *k*-way sorting method. Our algorithm is called "Efficient *k*-way Sorting Method". It has many enhancements and new features. Although the efficient *k*-way sorting method could effectively cleanse a data repository containing a uniquely identifying key field, it specifically focuses on identifying entries in data warehouses that do not have defining key elements. We also build a tool called ADD (Arabic Duplicate Detection), which is based on our algorithm. ADD can be used for any data warehouse. However, it has features tailored for Arabic data.

3.1 Efficient *k*-way Sorting Method

In the next few paragraphs, we outline our method and then compare it with the *k*-way sorting method. The efficient *k*-way sorting method works as follows:

```
for each sort order
 {
  open the current sort order excluding
  records that are already in the result set.
   while (not EOF)
    {
     for each search column in the current
     record
      {
       compare each search column with the
       previous columns within the window
       size.
       if more than half of the search
       columns match with the matching
       function specific for the search
       column then
        {
         we have a pair of records that are
         considered to be a match in this
         sort order.
         take the identifiers of the records
         pair and open the other k sort sets
         (incrementally) excluding the
         records that are already in the result
         set, and search for the two column
         identifiers apart from each other by
         the same window size.
         if more than half of the k-sort orders
         also match then we have a definite
         match. This definite match will be
         added to the result set.
        }
      }
    }
 }
```

The efficient *k*-way sorting method picks first the table that should be scanned for duplicates in the database. If more than one table is picked, then those tables are combined to form a single table by the use of views.

Next, the record identifier column is picked. If a record identifying column is not present, most database systems will let you add one.

The search columns are picked next. They can range from 1 to *n* (where *n* is the total number of columns in the table – 1). These columns are called the search columns and the number of columns picked will be designated as *k*. The most meaningful *k* sort orders are then picked.

Then, a window size is picked. Generally speaking, the window size must not be large, and typically it is either two or three. We call this window size *w*. The window size generally denotes the strictness or the forgiveness of the whole system in matching records.

The *k*-way sorting method delays finding or combing duplicates from the *k*-sorts until the last step, and then combines the results. However, our method does not delay this step to the end. Once a duplicate is found in one *k*-sort, it is then tested immediately on the other *k*-sorts. This approach is more beneficial since the already identified duplicates can be accumulated, and the information can be used to minimize on the search sorts along the way. Since the search sorts are not cached in our method, they are generated as needed.

Moreover, the *k*-way sorting method performs the same record matching scheme on each of the *k*-sorts, which we believe is a waste of time. While our method, once it finds a duplicate in one *k*-sort, it uses the duplicate identifiers to find the records in the other *k*-sorts within the window size. With our method it is possible to take this step and integrate it in the database, (using, for example, a stored procedure) which will further enhance the speed and performance of the duplicate detection process.

In our method, at the end of the *k*-sorts being examined, the data is already ready, while in the original *k*-way sorting method further

processing is needed, like graph combining and transitive closure.

3.2 Complexity of the Method

Consider the following parameters:

r = number of records in the table
w = window size
k = columns picked
t = time of matching algorithm

Then, our method is in the order of: $(r * w) * k * t * k$. $r * w$ is the number of records multiplied by the window size. Then, for each record we have k columns to compare, and each column takes time t (matching algorithm) and finally multiplied by k once more because we have k sets (sorted sets).

3.3 Arabic Duplicate Detector

A tool was built, which we call ADD or Arabic Duplicate Detector based on our efficient k-way sorting method. The matching functions in our method are implemented as plug-ins in the tool. Our tool comes with eight plug-ins. Out of the eight plug-ins, three of them are written for the Arabic language support and the rest are for general purpose. A brief look at each plug-in follows:

A. StringCmp:

This plug-in compares strings bit by bit. They either match or they do not. The comparison can either be case-sensitive or case-insensitive.

B. Soundex:

This is the standard soundex function, which compares strings according to the way they sound and not to the way they are written.

C. KeyboardDistance:

This plug-in measures the difference between how the strings in two fields are actually spelled compared to how badly they could possibly be misspelled. First, the horizontal and vertical actual distance between the corresponding characters in both fields is calculated. Then for each character that does not match, if it is close to the original character on the keyboard, then it is considered accepted. If it is not close, then the plug-in returns immediately with no match at all. For example, ‘Smith’ and ‘Snith’ match at 80%.

D. EditDistance:

This plug-in assesses equality between fields by determining how much editing would be required to convert one field value into the other. It compares the number of characters that need to be changed to the total number of characters of the longer field value to compute a match percentage. For example, the strings ‘Smith’ and ‘Snith’ match at 80% while ‘Smith’ and ‘Snit’ match at 60%.

E. CharacterFrequency:

This plug-in compares the frequency of occurrence of the different characters in each field. It creates a list of the characters and the number of occurrences of each character for each field and then compares the lists to compute a match percentage. The plug-in is useful for dealing with transpositions of characters. For example, the strings ‘Pierce’ and ‘Peirce’ match 100%, whereas the strings ‘Fierce’ and ‘Pier’ match at only 60%.

F. ASingleName:

This plug-in compares Arabic single names. It has three key points in the way it deals

with words. First, the plug-in removes any 'AL-AL TA3REEF' from the beginning of the words to ensure they will not affect the result. Next, it removes the ending character if it is a 'HAMZE' and is preceded by an 'ALEF'. Then, the algorithm compares the strings keeping in mind that the strings may begin and end with different characters even though they are the same.

G. ArabicMiddleName:

This plug-in performs middle name comparison. If one field has the initial of the middle name only, and if the other field has the complete middle name, it will compare the first character. It will match the 'ALEF' with 'HAMZE' and the 'ALEF' without a 'HAMZE'

H. ArabicAddress:

This is perhaps the most complicated plug-in, and it is the only plug-in that is content-driven. This plug-in has its own database file which it uses to decide for context matches. This plug-in cleans the strings from white spaces and other punctuation marks and extracts the substrings. It fills in the substrings into a substring list. Then for each substring in the list it decides if the substring is a noise word and if so removes it. It decides on the noise words from a database table it has. Noise words are words that do not add to the meaning of the string but append the meaning of another word. For example, "OUZA3EE Street" and "OUZA3EE" are the same, the word street is considered a noise word.

After this step, it attempts to match how many substrings from each string match the other substrings from the other string. For example, if string_1 has three substrings and string_2 has also three substrings, when all of the substrings match then the match result is 100%. If string_1 has only two substrings, and string_2 has three substrings and the substrings match completely then the match is 2/3 or 66% and so on.

The plug-in decides if two substrings match using two methods: parent-child relationship - if substring_1 is defined as the parent of substring_2 in the table, then these substrings will match. If the substrings do not match by the parent-child table, then the plug-in will use another function that works almost the same as the ASingleName plug-in.

The database design of this plug-in is described in Figure 1.

Areas	
PK,I1	ID
I2	ParentID Word

IgnoreVocab	
PK,I1	ID
	Word

Figure 1 – Arabic address database tables.

The table "Areas" contains the parent-child relationships of the words. For example, Beirut is the parent record and Hamra, Dikwaneh, and Koraitem are the child records, because these two areas lie within Beirut. The second table "IgnoreVocab" is the table that is used for the noise words.

4. Experimental Results

A sample database for Arabic names and addresses was tested. The schema of the database is shown in Figure 2.

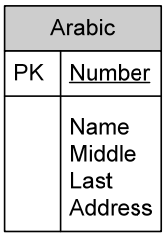

Figure 2 – Arabic sample database schema.

The table has the column [Number], which is used as the record identifier column, the column [Name] represents the Name of the person, column [Middle] represents the middle name of the person, column [Last] represents the last name of the person and finally the column [Address] represents the address of the person.

The experimental database has 13000 records. Out of the 13000 records, 9300 records are duplicate records. The 9300 records were the result of careful manual duplicate detection. We ran the software using the following settings:

Window Size (*w*):	3
Language Locale:	Arabic
Columns (*k*):	4

Table 1 presents the settings of the plug-ins:

Table 1 – Settings of plug-ins.

Column	Plug-in	Variable Comparison
Name	ASingleName	--
Middle	ArabicMiddleName	--
Last	ASingleName	--
Address	ArabicAddress	66%

The output of the ADD was 9100 records were found to be duplicates, only 200 records were missed which gives us an accuracy of 98.46 %.

5. Conclusion

An efficient *k*-way sorting method was presented in this paper. The method is an improvement over other existing methods and it tackles Arabic data. A tool called ADD was built based on the method. Add was written in C++. The results were impressive! ADD achieved an accuracy of 98.4 6%.

As for future work, we plan to allow expressions to be written for any column; expressions can combine any number of plug-ins, using the AND, OR, and NOT logical operators. Also, we plan to have the support for segmenting the fields into multiple tokens to be handled by the tool itself instead of leaving this job for the plug-in. This way the tool can control whether to call the plug-in given the whole field or call the plug-in given the tokens of the field.

References

[1] Manolopoulos, Y. *Data Mining: Challenges and Opportunities*. http://heldinet.dbnet.exe.ntua.gr/activities/workshops/w3/apth.pdf. 2003.

[2] Frand, J. *Data Mining: What is Data Mining?* The Anderson School at UCLA. http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/datamining.htm. 1998.

[3] Hernandez, M., and Stolfo, S. *The Merge/Purge Problem for Large Databases*. Proceedings of the ACM SIGMOD International Conference on Management of Data. pp. 127-133. May 1995.

[4] Monge, A. and Elkan, C. *An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate*

Database Records. Proceedings of the ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery. pp. 23-29. May 1997.

[5] Feekin, A. and Zhengxin, C. *Duplicate Detection Using K-way Sorting Method*. Proceedings of the ACM Scientific Applications Conference. Como, Italy. 2000.

[6] Monge, A. *An Adaptive and Efficient Algorithm for Detecting Approximately Database Records*. Bulletin of the Technical Committee . IEEE Computer Society Letters. Volume 23, 1. December 2000.

[7] Monge, A. and Elkan, C. *An Efficient Domain Independent Algorithm for Detecting Approximately Duplicate Database Records*. Proceedings of the ACM SIGMOD Workshop on Data Mining and Knowledge Discovery. 1997.

[8] Monge, A. and Elkan, C. *The Field Matching Problem: Algorithms and Applications*. KDD-96 Technology Spotlight. pp. 267-270. 1996.

Biography

Ramzi A. Haraty is an Assistant Professor of Computer Science at the Lebanese American University in Beirut, Lebanon. He received his B.S. and M.S. degrees in Computer Science from Minnesota State University - Mankato, Minnesota, and his Ph.D. in Computer Science from North Dakota State University - Fargo, North Dakota. His research interests include database management systems, artificial intelligence, and multilevel secure systems engineering. He has well over 40 journal and conference paper publications. He is a member of Association of Computing Machinery, Arab Computer Society and International Society for Computers and Their Applications.

Ralph Varjabedian M.S. degree in Computer Science form the Lebanese American University – Beirut, Lebanon. His research interests include software engineering and database management systems.