

# Efficient Algorithms for Grouping Data to Improve Data Quality

Wing Ning Li  
CSCE department  
University of Arkansas  
Fayetteville, Arkansas 72701  
Email: wingning@uark.edu

Johnson Zhang  
ISYS department  
University of Arkansas  
Fayetteville, Arkansas 72701  
Email: JZhang@walton.uark.edu

Roopa Bheemavaram  
CSCE department  
University of Arkansas  
Fayetteville, Arkansas 72701  
Email: rbheema@uark.edu

**Abstract**—Improving and maintaining data quality has become a critical issue for many companies and organizations because poor data degrades organizational performance whereas quality data results in cost saving and customer satisfaction. Activities such as identifying and removing “duplicate” database records from a single database, and correlating records, which identify the same real world “entity”, from different databases are used routinely to improve data quality. Due to the large size of the data sources having several hundred millions to several billions records, and continuously growing, efficient techniques and algorithms are needed. One approach to speed up the processing is to use a two-step process, where potential candidate records are grouped together in step one and each group is further processed and analyzed in step two. The record grouping problem is a formal formulation of what needs to be done in step one. This paper introduces a record grouping problem called transitive closure problem, and proposes algorithms to solve the problem. The proposed algorithms have been implemented efficiently in several ways. The paper reports on the empirical study of the implementations of the proposed algorithms.

**Key Words:** Record grouping, data quality, transitive closure, record linkage, sorting, disjoint set find and union.

## I. INTRODUCTION

Data quality directly influences information quality. The impact of information quality upon organizations has been indicated via use and user satisfaction [7]. A categorization of data quality problems is provided in [16] and listed below:

- 1) Data “views”: how is the real world captured in the data, such as relevancy, granularity and level of detail.
- 2) Data values: data accuracy, redundancy, consistency, currency and completeness.
- 3) Data presentation: appropriateness of the data format, ease of interpretation.
- 4) Other data issues: data privacy, security and ownership.

Poor information quality directly impacts on organizational performance. According to the study by Data Warehousing Institute, an estimated annual loss of \$600 billion is due to poor data [1]. Another study estimates that the cost associated with poor data quality is 8-12% of the revenue of a typical organization, and informally speculated, 40-60% of a service organization’s expense [16]. The effect of poor data quality is summarized as: “customer dissatisfaction, increased operational cost, less effective decision-making, and a reduced ability to make and execute strategy, hurts employee morale,

breeds organizational mistrust, and makes it more difficult to align the enterprise” [15]. The reader is referred to [2], [3], [8] for further discussion of the impact of poor data and issues concerning data quality management.

Most companies have realized the importance of data quality. According to a survey in 2000, by InformationWeek Research, of 300 IT executives, over 80% agreed improving customer data quality was their top priority [9]. The market for database cleansing service has been growing, up to 20% annual growth rate through 2008 [1]. However, the cost of cleansing data is also very high, in addition to the complexity of implementing such task. The reported cost ranges from “\$100,000 and \$500,000, depending on company size and the amount of data you need to clean.” [1].

Among the four categories of data quality, this paper address the second category, which measures data quality in terms of data *accuracy*, *redundancy*, *consistency*, *currency* and *completeness*. To illustrate some of the issues, let us consider the four records shown in Table I.

Record	Name	Address	Phone	SSN
1	Peg M Smith	123 Main St. Apt. 3	870-123-4567	220-43-1234
2	Peggy Smith	123 Main St.	870-123-4567	220-34-1234
3	Peg R Smith	213 Main St. Apt. 3	870-321-4567	220-43-1233
4	Robert P Smith	213 Main Street	870-321-4566	220-43-1233

TABLE I

A SAMPLE RECORD SET OF SIMILAR RECORDS.

The reader should have no difficulty in observing the similarities among the records in Table I. Further analysis of this record set might reveal that all four records belong to the same household. This realization might lead to another process in which the discrepancy in the address field can be resolved. Hence, data *accuracy* and *consistency* are improved. The same idea might be used to correct the typos or errors in the telephone field in the example. Now, let us imagine that the addresses are all quite different in this example, reflecting the old and new addresses resulted from moves or change of addresses. By realizing that all four records belong to the same household, we might initiate another process that identifies the current address of the household. Hence, data *currency* is maintained.

Another analysis of the same record set might also reveal that record 1 and record 2 refer to the same individual. As a result, we might have duplicate records if both records are from the same database. Such information is useful to address the data *redundancy* issue. Now, let us imagine that record 1 and record 2 are from different databases. In addition to the four fields shown, record 1 has an email address field and record 2 has a cell phone field. Knowing record 1 and record 2 refer to the same individual, we might fill in the cell phone information for record 1 if this field is blank or enhance the database of record 1 by adding a cell phone column. Likewise, record 2 might be made more complete by having an email address. We might even build and develop a new, more *complete* database, which "combines" the two databases with enhancement. Notice that the relational operator "joint" may not be performed due to the lack of common key from different data sources.

The proceeding discussion demonstrates the importance of the analysis tools for achieving data quality in terms of data accuracy, redundancy, consistency, currency and completeness. Usually such analysis tools are nontrivial and demand a lot of computing time. The run time of these tools is at least quadratically related to the number of records given and has large coefficients for the polynomial. Such tools might spend days and weeks if not months or years when they are given a data source having billions of records [11].

To use these analysis tools more effectively and efficiently for dealing input records that are in the range of hundred millions or even billions, a fast preprocessing step is needed. Depending on which analysis tools are to be used, the preprocessing step partitions the input records into many smaller groups with similar records ending up in the same group. Only those records within a group might lead to further improvement of the data quality when they are analyzed by additional processes. We can think of the records of Table I as one of the groups produced by a preprocessing step. The preprocessing step makes all "related" records into a single group. Typically, the relation of *relatedness* is an equivalent relation<sup>1</sup>, and the relation partitions the input records into equivalent classes. Another view of these groups is that all the records in a group are "related" to one another. As a result of grouping more relations might be added. For example, suppose we are given two relations (R1, R2) and (R2, R3). Due to transitivity, we conclude that (R1, R3). Therefore the grouping process may be viewed as a process of computing the transitive closure<sup>2</sup>. This paper uses the term "transitive closure" to refer to the problem that is addressed in the preprocessing step. This grouping process is referred to as "blocking method" in [4] where related approaches are discussed.

<sup>1</sup>A relation that is reflexive, symmetric, and transitive is said to be an equivalent relation [12]

<sup>2</sup> $R$  is a relation. The transitive closure of  $R$ , denoted by  $R^+$ , is defined by 1) if  $(a, b)$  is in  $R$ , then  $(a, b)$  is in  $R^+$ ; 2) if  $(a, b)$  is in  $R^+$  and  $(b, c)$  is in  $R^+$ , then  $(a, c)$  is in  $R^+$ ; and 3) Nothing is in  $R^+$  unless it so follows from (1) and (2) [12].

The transitive closure problem is formally introduced in the next section. Algorithms for solving the transitive closure problem are developed in section 3. Experimental results are given in section 4. Future work and related work are discussed next before the acknowledgement section.

## II. PROBLEM DESCRIPTION

*Definition 1:* A *record* is a n-tuple,  $(V_1, V_2, \dots, V_n)$ ,  $V_i$ ,  $1 \leq i \leq n$ , denotes the value of  $i$ th key and  $n$  denotes the number of keys.

In the example of Table I, the first record may be viewed as  $(V_1, V_2, V_3, V_4)$  where  $V_1$ ,  $V_2$ ,  $V_3$ , and  $V_4$  are respectively Peg M Smith, 123 Main St. Apt. 3, 870-123-4567, and 222-43-1234.

*Definition 2:* Two records  $(V_1, \dots, V_n)$  and  $(U_1, \dots, U_n)$  are *related* if there exists  $i$ ,  $1 \leq i \leq n$ , such that  $V_i = U_i$ .

In other words, two records are *related* if for some key the two records share the same value. Notice that from the above definition, all keys are treated the same. As long as one key value is identical, two records are *related*. This may be desirable in the preprocessing step that intends to capture all possibly related information.

*Definition 3:* Two records  $V = (V_1, \dots, V_n)$  and  $U = (U_1, \dots, U_n)$  are *transitively related* if either the two records are related or there exists records  $R_1, R_2, \dots, R_k$  for  $k \geq 1$  such that  $V$  and  $R_1$ ,  $R_k$  and  $U$ , and  $R_i$  and  $R_{i+1}$  are related,  $1 \leq i < k$ .

In the example of Table I, records 3 and 4 are related because the value of their 4th key, 220-43-1233, is the same. Suppose we only use first name and last name to test the value of the name field. Then records 1 and 3 are related. From transitively related definition, records 1 and 4 are transitively related because record 1 relates to record 3, and record 3 relates to record 4.

*Definition 4:* Let  $S$  be a set.  $P_1, P_2, \dots, P_k$  are *partitions* (*clusters, transitive closures*) of  $S$  iff  $S = P_1 \cup P_2 \cup \dots \cup P_k$  and  $P_i \cap P_j = \emptyset$ ,  $i \neq j$ ,  $1 \leq i, j \leq k$ .

### Transitive Closure Problem

**Input:** A set of records.

**Output:** Partitions of input record set such that all transitively related records are in one partition.

## III. ALGORITHMS

Any algorithm that solves the transitive closure problem must address two fundamental problems. One is to determine if two records are related or not for any two records. This task amounts to computing the complete relation using definition 2. The other is to partition the records based on the relation. Consider the sample data file shown in Fig. 1 and the same information and the corresponding solution depicted in Fig. 2. Fig. 2 shows 2 transitive closures for the 12 records shown Fig. 1. The first cluster includes R1, R2, R3, R5, R6, R9, R10, R11, and R12. The second one includes R4, R7 and R8. Fig. 2 also shows which key or keys make a pair of records related to one and another. For example, R4 and R7 are related to one

another by Key 2, which means their second keys are identical. Similarly, R9 and R10 are related to one another by Key 1 and Key 3, indicating not only their first keys are identical, but the third keys as well. In this example, computing all the edges of the graph in Fig. 2 corresponds to the first problem. Circling the nodes of the graph corresponds to the second problem.

RecordNo.	Key 1	Key 2	Key 3	Key 4	Set Number
R1	C1K1V4	C1K2V10	C1K3U30	C1K4U31	1
R2	C1K1V4	C1K2U2	C1K3U3	C1K4V2	2
R3	C1K1V4	C1K2V5	C1K3U8	C1K4V5	3
R4	C2K1U18	C2K2V3	C2K3V3	C2K4V3	4
R5	C1K1V5	C1K2V7	C1K3U21	C1K4V7	5
R6	C1K1V5	C1K2V4	C1K3U3	C1K4V4	6
R7	C2K1V9	C2K2V3	C2K3U5	C2K4V4	7
R8	C2K1V16	C2K2V3	C2K3U6	C2K4V5	8
R9	C1K1V7	C1K2U13	C1K3U14	C1K4U15	9
R10	C1K1V7	C1K2V42	C1K3U14	C1K4V34	10
R11	C1K1V23	C1K2V6	C1K3U14	C1K4V6	11
R12	C1K1V23	C1K2V1	C1K3U14	C1K4V5	12

Fig. 1. An example of a transitive closure instance

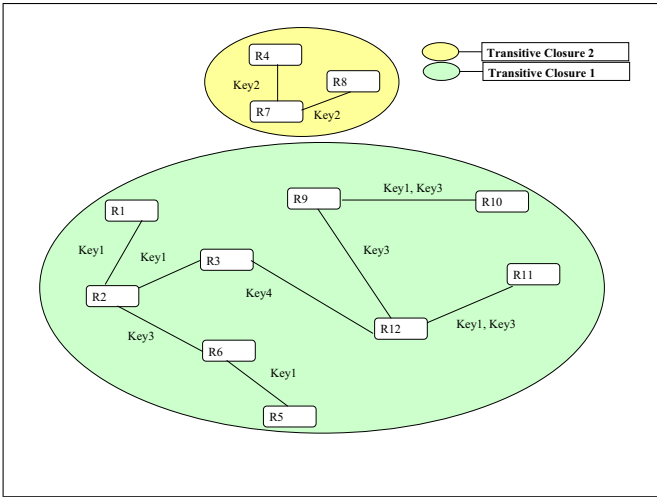


Fig. 2. A graph representation of transitive closure

The algorithms that solve the transitive closure problem, therefore, may be classified into two groups. One group is to solve both problems at the same time, and the other is to solve the two problems one after another in two steps. The output of the first step will be used as the input for the second step. In an earlier preliminary paper [18], we proposed and studied two algorithms that solve both problems at the same time. One algorithm uses pairwise comparison to solve the first problem and at the same time performing breadth first search to solve the second problem by computing connected components. The overall time complexity of the algorithm is  $O(n^2)$  due to pairwise comparison, where  $n$  is the number of records. The other algorithm uses sorting of each key to solve

the first problem and at the same time performing disjoint set find and union to solve the second problem. The overall time complexity of the algorithm is  $O(n \log n)$ . The reader is referred to [18] for more details of both algorithms and their implementations.

In the current paper, the second algorithm is investigated further due to its superior performance over the other one. Based on the idea of solving the problem in two steps, another algorithm and approach is proposed.

For ease of reference and illustration, the pseudo code of the second algorithm in [18] is given in Fig. 3.

The algorithm passes through the record set  $k$  times, where  $k$  is the number of keys. In each pass, it considers a new key and uses that key to further the transitive closure computed thus far using earlier keys. The refinement is carried out very efficiently using disjoint set find and union data structures. Due to definition 2, sorting of a given key can be used to derive all record relations due to that key. If a perfect hashing is available, the same information may be derived using hashing, which is much more efficient than sorting. However, for non-perfect hashing, collisions may happen, which introduces fictitious relatedness between records. These new relations might lead to fewer transitive closures with bigger sizes. Nonetheless, the idea of using hashing is very interesting and will be investigated further. Sorting and disjoint set find and union are the two key ideas used to solve the problem. Both topics are covered in details in [6]. Taking advantage of definition 2, we use sorting to speed up the process of identifying related information, and avoid conducting a pairwise comparison for all the records. Disjoint set find and union is used to derive the transitive closures. The disjoint set data structures maintain a group of sets that are pairwise disjoint. Given an element find returns the set in which the element belongs. Given two elements from two disjoint sets, union combines the two sets into a single one.

**Algorithm 1: computing the transitive closures or partitions**

Assume  $n$  records from 1 to  $n$  and  $k$  keys from 1 to  $k$ . Each record also has a record number field for identification purpose.  $djs$  is the data structures for disjoint set find and union. It is assumed that  $djs$  is initialized with each record forming a set of its own

```

0 for ( i = 1; i ≤ k i++)
1   sort all records based on key i;
2   prevKey = r[1].key[i];
3   recNo = r[1].recNo;
4   for ( j = 2; j ≤ n j++)
5     if (prevKey == r[j].key[i] and djs.find(recNo) ≠ djs.find(r[j].recNo) )
6       djs.union(recNo, r[j].recNo);
7     prevKey = r[j].key[i];
8     recNo = r[j].recNo;
9 for ( i = 1; i ≤ n i++)
10  r[i].partition = djs.find(r[i].recNo);

```

Fig. 3. An algorithm finding transitive closures from a record file in one step.

Note from the pseudo code, that after the current keys are sorted the first test of the if statement of line 5 tells us the

current record and the previous record are related. The second test of the if statement tells us that both records are currently in different clusters and the corresponding clusters should be combined. As can be seen, the related information and the transitive closure are derived almost simultaneously. Sometimes the two may be computed independently. Conceptually, as shown above, the first test may be used to generate a pair of records or a pair of keys if the two records are related. Therefore, a set of key pairs may be used to represent the related information and get computed in the first step of a two-step algorithm. Again the 12-record example will be used to demonstrate how this may work. In the example, key 1 is used to relate different records and the set of key pairs are key 1 pairs.

Observe key 2 of the example of Fig 1 and that R8, R7 and R4 have identical key 2 value. For these records the following related information or key 1 pairs: (C2K1V16, C2K1V9), (C2K1V9, C2K1U18) are derived. Applying the same logic to key 3 and key 4, the following pairs: (C1K1V7, C1K1V7), (C1K1V7, C1K1V23), (C1K1V23, C1K1V23) for R9, R10, R11, R12, (C1K1V4, C1K1V5) for R2 and R6, and (C1K1V4, C1K1V23) for R3 and R12 are derived respectively. The key 1 pairs: (C2K1V16, C2K1V9), (C2K1V9, C2K1U18), (C1K1V7, C1K1V7), (C1K1V7, C1K1V23), (C1K1V23, C1K1V23), (C1K1V4, C1K1V5), (C1K1V4, C1K1V23) capture the exact related information. Now if we apply disjoint set find and union on these key pairs, we will obtain 2 partitions: C2K1V16, C2K1V9, C2K1U18 and C1K1V7, C1K1V23, C1K1V4, C1K1V5. Based on key 1, the records are partitioned into R4, R7, R8 and R1, R2, R3, R5, R6, R9, R10, R11, R12, which is identical to the result of Fig. 2.

The set of key pairs can be derived very efficiently through massive parallel computing in propriety grid computer architecture [17]. Therefore, in addition to files that conceptually similar to that of Fig. 1, a set of key pairs such as those shown earlier could be another form of input to the transitive closure problem, Algorithms that take a set of key pairs as input are developed and empirically evaluated. The proposed algorithms take the form of the pseudo code shown in Fig 4 .

---

**Algorithm 2: computing the transitive closures or partitions**

Assume first key is used for key pairs and each line of input has two keys. djs is the data structures for disjoint set find and union. It is assumed that djs is initialized with each key forming a set of its own. Two files are processed.

```

0 while more input to read in the pair file
1   read the two keys, key1 and key2;
2   if (djs.find(key1) ≠ djs.find(key2) )
3     djs.union(key1, key2);
4 while more input to read in the record file
5   read next record i;
10  r[i].partition = djs.find(r[i].key[1]);

```

---

Fig. 4. An algorithm finding transitive closures from a key pair file. The second step of a two-step approach.

## IV. EXPERIMENTAL RESULTS

All algorithms are implemented in C++ and empirically studied. The record files used in the experiment are generated by a synthetic data generator, called Ruby data generator [17]. The example of Fig. 1 is also generated by the Ruby data generator.

Cluster Size	Total records	Sorted Algo1 Time in sec	Sorted Algo2 Time in sec	Random Algo1 Time in sec	Random Algo2 Time in sec
small	100,000	568	3	565	3
median	100,000	610	3	610	3
large	100,000	612	3	612	3

TABLE II

EXPERIMENTAL RESULTS FOR RUBY DATA FILES.

The experimental results for Ruby Data Files are shown in Table II. Many files are used in the experiment. Files are classified into three categories based on the number of records in the final clusters or transitive closures. For small size distribution (second row of the table), cluster sizes are in the range of 10 to 100. For median size distribution, cluster sizes are in the range of 200 to 2000. For large size distribution, cluster sizes are in the range of 10,000 to 100,000. The run time is the average time of processing all the files in that category. Column 2 lists the number of records in each file. In the files generated by Ruby data generator, all records belonging to the same transitive closure (cluster) are located together. In other words, records in a file are "sorted" according to their transitive closures. The "sorted" property might affect the evaluation and produce bias results. A tool, developed in house, is used to randomize the output files from the Ruby data generator. Both the "sorted" files and the corresponding randomized files are used in the experiment. Columns 3 and 4 list the run times for sorted files, and columns 5 and 6 for unsorted files.

Algo1 and Algo2 are two implementations of the same algorithm proposed earlier (see Fig. 3), which solves the transitive closure problem directly from the record data files. The only difference between the two is the implementation of disjoint set find and union data structures. Algo1 uses C++ template class design to implement disjoint set to allow set element to be of any object. Algo2 restricts set elements to be integers from 1 to  $n$  and takes advantage of such restrictions to optimize the design and implementation of disjoint set. The experiment shows that the implementation with integer set runs 200 times faster than that with an object set. Regardless the input files are "sorted" or not, the performance of the algorithms remains the same. The program ran on a single 1.3 GHz Dell computer with 512 MB main memory running Windows XP.

Based on the experimental results of Table II, to better the performance, integers should be used as keys.

The proposed algorithm (see Fig. 4) of taking in a file containing key pairs to compute the transitive closure is implemented. The implementation uses integer disjoint set as

Distinct Keys	Total pairs	Text File Time sec	Binary File Time sec	Text File Size mb	Binary File Size mb
30,728,760	110,664,705	222	96	2019	844
61,457,520	221,329,410	502	138	4205	1688
92,186,280	331,994,115	755	153	6225	2532
122,915,040	442,658,820	1257	184	8490	3376

TABLE III  
EXPERIMENTAL RESULTS FOR INTEGER PAIR FILES.

Algo2 does. Hence, in the experiment the key pair files contain a sequence of integer pairs. Assuming that 32-bit integers are used, two ways of storing a pair of integers are possible. One way is to store an integer as a string of digits in a text file and an integer key pair is stored as two character strings separated by white space. The other way is to store an integer using its machine image in a binary file and an integer key pair is stored as eight bytes data corresponding to its machine images.

The experimental results are shown in Table III. The key pair files are derived from the first keys of the original ruby data files. All the ruby data files have the same cluster distribution: 52.5% of the clusters have 5 records, 42.5% has 20 records, 4% has 5,000 records and 1% has 50,000 records. The following test statistics are shown in Table III: the number of distinct first keys (column 1), the number of key pairs in the input file (column 2), the run time for the text file (column 3), the run time for the binary file (column 4), the text file size in disk (column 5), and the binary file size in disk (column 6). The results show substantial improve in run time when binary file is used. It is believed that the saving is due to more efficient I/O in binary file and elimination of date type conversions. As we can see, it takes about 1.5 minutes to finish the transitive closure computation for an input file having about 100 millions key pairs (110,664,705 to be exact). The largest file size is about 8GB with 442 million pairs in the experiment. For this size, the main memory used is no more than 500MB. The memory usage depends on the number of distinct keys and does not depend on the number of pairs. The program ran on a single 2.8 GHz Dell computer with 1 GB main memory running Windows XP.

In all implementations of the proposed algorithms, that the processing can be done internally is assumed, meaning all the needed information can be stored in main memory or virtual memory. However, the assumption is invalid for files with billions of records and each record takes hundreds of bytes. For example, for many 32-bit systems, the maximum virtual memory an application can have is either 2GB or 4GB. To overcome the problems caused by the huge data size, single machine solutions may have to use external processing and external files and multi-machine solutions, combining memory of each machine into a large memory, may have to use distributed processing. Due to space limitation, the reader is referred to [5] for our preliminary results of distributed algorithm for the transitive closure problem. In any event, the proposed sequential algorithms may be applied to improve the performance of both single and multi-machine solutions

because both solutions must have an internal processing component.

## V. CONCLUSION

In the paper, the importance of data quality is considered. Why analysis tools could improve data quality in terms of data *accuracy, redundancy, consistency, currency* and *completeness* is articulated and illustrated through an example. To improve the performance of the analysis tools, transitive closure problem is introduced and formulated. Transitive closure processing reduces the number of records fed to the analysis tools by grouping related records into groups. The size of each group is almost always much smaller than the size of the original data source, where the groups are derived. The analysis tools gain performance by being applied to the records within each group instead of the total records in the data source. Algorithms are proposed to solve the transitive closure problem, based on the ideas of sorting and disjoint set find and union. The algorithms are programmed and the implementations are studied empirically. The experimental results strongly favor using integers for disjoint set, using integers as record identifiers, and using binary or machine representation for storing integer pairs in a file. Hence, whenever possible integers should be used as record identifiers. Using integer pairs as record identifiers should improve the performance of transitive closure computation.

The related work is briefly discussed in the following. For the analysis tools mentioned in the introduction, two fundamental questions that need to be answered are: 1) Does the two records refer to the same real-world entity of interest? 2) Are two fields similar, same, or approximately same? To see how two records compare, we select the same set of fields from both records and base the results of the field comparison to determine the outcome of the record comparison [11], [14], [4]. Approximate string match technique and its variations are typically applied to address the field comparison question [13], [10]. Several commercial products (for example, Acxiom, Trillium, Vality) have proprietary algorithms, which might leverage characteristics peculiar to a particular domain, such as address, for matching addresses and individual or business records.

We are currently investigating how to further speed-up, scale-up the proposed algorithm by parallel and distributed processing using grid computers, and developing a prototype of a distributed algorithm [5] to evaluate the performance of the algorithm empirically. Another idea that requires further study is to use hashing for obtaining relations and the impact of fictitious relations on the final transitive closure result. Other competing definitions of relatedness and other parameters that might affect the effectiveness of transitive closure computation need to be investigated, for example, applying different weights to different keys. Last, but not the least, we need to improve the analysis tools, which process the records in a transitive closure, so that ultimate data quality can be achieved. One technique could be to use clustering analysis techniques to subdivide the over-group records in a transitive closure.

## ACKNOWLEDGMENT

This research was supported in part by Axiom Corporation through the Axiom Laboratory for Applied Research. We would also like to thank our Axiom Champions Tanton Gibbs and Tom Schweiger for bringing the transitive closure problem to our attention.

## REFERENCES

- [1] D. Agostino. Getting clean. *CIO-Insight*, 42:72–76, 2004.
- [2] D Ballou. Enhancing data quality in data warehousing environment. *Comm. ACM*, 42(2):73–78, 1999.
- [3] D Ballou, H. Wang, and G. Pazer. Modeling information manufacturing systems to determining information product quality. *Management Science*, 44(4):462–484, 1998.
- [4] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD '03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 25–27, Washington, DC, 2003.
- [5] R. Bheemavaram, J. Zhang, and W. Li. A parallel and distributed approach for finding transitive closures of data records: A proposal. In *Axiom Laboratory for Applied Research Conference on Applied Research in Information Technology*, pages 61–70, 2006.
- [6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction To Algorithms Second Edition*. McGraw-Hill Higher Education, 2001.
- [7] W. Delone and E. Mclean. Information systems success: The quest for the independent variable. *Information Systems Research*, 3(1):60–95, 1992.
- [8] B. Depompa. Scrub data clean. *InformationWeek*, 610:88–92, 1996.
- [9] M. Faden. Data cleansing helps e-business run more efficiently. *InformationWeek*, 781:136–139, 2000.
- [10] P Hall and G. Dowling. Approximate string matching. *ACM Computing Survey*, 13(4):381–402, 1980.
- [11] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of data*, pages 127–138, 1995.
- [12] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, And Computation*. Addison-Wesley, 1979.
- [13] P. Jokinen, J. Tarhio, and E. Ukkonen. A comparison of approximate string matching algorithms. *Software Practice and Experience*, 26(12):1439–1458, 1996.
- [14] G. Navarro, R. Baeza-Yates, E. Sutinen, and J. Tarhio. Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin*, 24(4):19–27, 2001.
- [15] T. Redman. *Data Quality for the Information Age*. Artech House, 1996.
- [16] T. Redman. The impact of poor data quality on the typical enterprise. *Comm. ACM*, 41(3):79–82, 1998.
- [17] Project Sponsor. private communication, 2006.
- [18] J. Zhang, R. Bheemavaram, and W. Li. Transitive closure of data records: Application and computation. In *Axiom Laboratory for Applied Research Conference on Applied Research in Information Technology*, pages 71–81, 2006.