Matching Algorithms within a Duplicate Detection System

Alvaro E. Monge
California State University Long Beach
Computer Engineering and Computer Science Department,
Long Beach, CA, 90840-8302

Abstract

Detecting database records that are approximate duplicates, but not exact duplicates, is an important task. Databases may contain duplicate records concerning the same real-world entity because of data entry errors, unstandardized abbreviations, or differences in the detailed schemas of records from multiple databases – such as what happens in data warehousing where records from multiple data sources are integrated into a single source of information – among other reasons. In this paper we review a system to detect approximate duplicate records in a database and provide properties that a pair-wise record matching algorithm must have in order to have a successful duplicate detection system.

1 Introduction

Many of the current technological improvements has lead to an explosion in the growth of data available in digital form. The most significant of these being the popularity of the Internet and specifically the world wide web. There are other more traditional sources of data however that has also contributed to this exponential growth. The commercial success of relational databases in the early 1980's has lead to the efficient storage and retrieval of data. Hardware improvements have also contributed significantly now that external storage has faster access times and continue to increase in density.

Such technological changes allow for easy and widespread distribution and publishing of data. The availability of these data sources increases not only the amount of data, but also the variety of and quality in which such data appears. These factors create a number of problems. The work presented here concentrates on one such problem: the detection of multiple representations of a single entity. Following are examples where solutions to such a problem are needed:

- As more and more data becomes available, it is desirable to integrate the data whenever possible. For example, one data source may contain bibliographic data for published scientific papers. Another data source may contain a white pages service for web pages of people. By integrating the data from these two sources, one can go directly to the authors' web pages to obtain information about the more recent work by the authors. Such an integration is much like the *join* operation in relational databases [1, 2]. While the join of two relations is based on an exact match of corresponding attributes, here we relax that condition and allow for an approximate match.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

- Another typical example is the prevalent practice in the mass mail market of buying and selling mailing lists. Such practice leads to inaccurate or inconsisten data. One inconsistency is the multiple representations of the same individual household in the combined mailing list. In the mass mailing market, this leads to expensive and wasteful multiple mailings to the same household

Relational database systems disallow the entry of records containing duplicate primary key values. Unfortunately, due to errors such as from data entry, whenever the value of the primary key attributes is affected by these errors the relational database system can no longer guarantee the non-existence of duplicate records.

The next section reviews the system for detecting approximate duplicate database records [3, 4, 5]. One important module in the system is the algorithm to detect pair-wise approximate duplicates. The system uses the algorithm to establish "is a duplicate of" relationships between pairs of records leading to its adaptability among different data sources and different dynamics of the database records. Section 2 looks at the different matching algorithms that could be plugged in to the duplicate detection system. The article concludes in Section 4 with final remarks about this work.

2 Algorithms to match records

The word *record* is used to mean a syntactic designator of some real-world object, such as a tuple in a relational database. The record matching problem arises whenever records that are not identical, in a bit-by-bit sense – or in a primary key value sense – may still refer to the same object. For example, one database may store the first name and last name of a person (e.g. "Jane Doe"), while another database may store only the initials and the last name of the person (e.g. "J. B. Doe").

The record matching problem has been recognized as important for at least 50 years. Since the 1950s over 100 papers have studied matching for medical records under the name "record linkage." These papers are concerned with identifying medical records for the same individual in different databases, for the purpose of performing epidemiological studies [16]. Record matching has also been recognized as important in business for decades. For example tax agencies must do record matching to correlate different pieces of information about the same taxpayer when social security numbers are missing or incorrect. The earliest paper on duplicate detection in a business database is by [17]. The "record linkage" problem in business has been the focus of workshops sponsored by the US Census Bureau [12, 18]. Record matching is also useful for detecting fraud and money laundering [19].

Almost all published previous work on record matching is for specific application domains, and hence gives domain-specific algorithms. For example, papers discuss record matching for customer addresses, census records, or variant entries in a lexicon. Other work on record matching is not domain-specific, but assumes that domain-specific knowledge will be supplied by a human for each application domain [20, 7].

Record matching algorithms vary by the amount of domain-specific knowledge that they use. The pairwise record matching algorithms used in most previous work have been application-specific. Many algorithms use production rules based on domain-specific knowledge. The process of creating such rules can be time consuming and the rules must be continually updated whenever new data is added to the mix that does not follow the patterns by which the rules were originally created. Another disadvantage of these domain-specific rules is that they answer whether or not the records are or are not duplicates, there is no in between.

The goal of this work is to create a detection system that is of general use. Record matching algorithms should use as little domain-specific knowledge as possible and should also provide a measure of the strength of the match. This information is crucial to the efficiency with which the detection system can process the data.

2.1 The record matching problem

In this work, we say that two records are *equivalent* if they are equal semantically, that is if they both designate the same real-world entity. Semantically, this problem respects the reflexivity, symmetry, and transitivity properties. The record matching algorithms which solve this problem depend on the syntax of the records. The syntactic calculations performed by the algorithms are approximations of what we really want – semantic equivalence. In such calculations, errors are bound to occur and thus the semantic equivalence will not be properly calculated. However, the claim is that there are few errors and that the approximation is good.

Equivalence may sometimes be a question of degree, so a function solving the record matching problem returns a value between 0.0 and 1.0, where 1.0 means certain equivalence and 0.0 means certain non-equivalence. Degree of match scores are not necessarily probabilities or fuzzy degrees of truth. An application will typically just compare scores to a threshold that depends on the domain and the particular record matching algorithm in use.

2.2 Algorithms based on approximate string matching

One important area of research that is relevant to approximate record matching is approximate string matching. String matching has been one of the most studied problems in computer science [21, 22, 23, 24, 25, 26]. The main approach is based on edit distance [27]. Edit distance is the minimum number of operations on individual characters (e.g. substitutions, insertions, and deletions) needed to transform one string of symbols to another [28, 23]. In [23], the authors consider two different problems, one under the definition of equivalence and a second using similarity. Their definition of equivalence allows only small differences in the two strings. For examples, they allow alternate spellings of the same word, and ignore the case of letters. The similarity problem allows for more errors, such as those due to typing: transposed letters, missing letters, etc. The equivalence of strings is the same as the mathematical notion of equivalence, it always respects the reflexivity, symmetry, and transitivity property. The similarity problem on the other hand, is the more difficult problem, where any typing and spelling errors are allowed. The similarity problem then is not necessarily transitive; while it still respects the reflexivity and symmetry properties. Edit distance approaches are typically implemented using dynamic programming and run in $O(mn)$ time where m and n are the lengths of the two records. Thus the importance on avoiding unnecessary calls to the record matching function by the duplicate detection system.

Any of the approximate string matching algorithms can be used in place of the record matching algorithm in the duplicate detection system. In previous work we have used a generalized edit-distance algorithm. This domain-independent algorithm is a variant of the well-known Smith-Waterman algorithm [29], which was originally developed for finding evolutionary relationships between biological protein or DNA sequences.

The Smith-Waterman algorithm is domain-independent under the assumptions that records have similar schemas and that records are made up of alphanumeric characters. The first assumption is needed because the Smith-Waterman algorithm does not address the problem of duplicate records containing fields which are transposed, see [5] for solutions to this problem. The second assumption is needed because any edit-distance algorithm assumes that records are strings over some fixed alphabet of symbols. Naturally this assumption is true for a wide range of databases, including those with numerical fields such as social security numbers that are represented in decimal notation.

3 System to detect duplicate database records

This section summarizes the system used in detecting approximately duplicate database records [3, 4, 5]. In general, we are interested in situations where several records may refer to the same real-world entity, while not being syntactically equivalent. A set of records that refer to the same entity can be interpreted in two ways. One way is to view one of the records as correct and the other records as duplicates containing erroneous information. The task then is to cleanse the database of the duplicate records [6, 7]. Another interpretation is to consider each matching record as a partial source of information. The aim is then to merge the duplicate records, yielding one record with more complete information [8]. The system described here gives a solution to the detection of approximately duplicate records only. In particular, it does not provide a solution to the problem of consolidating the detected duplicate records into a single representation for the real-world entity.

3.1 Standard duplicate detection

The well known and standard method of detecting *exact duplicates* in a table is to sort the table and then to check if consecutive records are identical. Exact duplicates are guaranteed to be next to each other in the sorted order regardless of which part of a record the sort is performed on. The approach can be extended to detect approximate duplicates. The idea is to do sorting to achieve preliminary clustering, and then to do pairwise comparisons of nearby records [9, 10, 11]. In this case, there are no guarantees as to where duplicates are located relative to each other in the sorted order. At best, the approximate duplicate records may not be situated next to each other but will be found nearby. In the worse case they will be found in opposite extremes of the sorted order. Such results are possible due to the choice of field to sort on and also to the errors present in the records. In order to capture all possible duplicate records, every possible pair of records must be compared, leading to a quadratic number of comparisons – where the comparison performed is typically an expensive operation as we will see in section 2. This gives rise to an inefficient and possibly infeasible solution since the number of records in the database may be in the order of hundreds of millions.

To avoid so many comparisons, we can instead compare only records that are within a certain distance from each other. For example, in [7], the authors compare nearby records by sliding a window of fixed size over the sorted database. As a window of size W slides over the database one record at a time, the new record is compared against the other $W - 1$ records in the window. Now, the number of record comparisons decreases from $O(T^2)$ to $O(TW)$ where T is the total number of records in the database.

There is a tradeoff here between the number of comparisons performed and the accuracy of the detection algorithm. The more records the window contains (large value of W) the better the system will do in detecting duplicate records. However this also increases the number of comparisons performed and thus leads to an increase in running time. An effective approach is to scan the records more than once but in a different order and apply the fixed windowing strategy to compare records and combine the results from the different passes[9, 12]. Typically, combining the results of several passes over the database with small window sizes yields better accuracy for the same cost than one pass over the database with a large window size.

One way to combine the results of multiple passes is by explicitly computing the transitive closure of all discovered pairwise "is a duplicate of" relationships [7]. If record R_1 is a duplicate of record R_2, and record R_2 is a duplicate of record R_3, then by transitivity R_1 is a duplicate of record R_3. Transitivity is true by definition if duplicate records concern the same real-world identity, but in practice there will always be errors in computing pairwise "is a duplicate of" relationships, and transitivity will propagate these errors. However, in typical databases, sets of duplicate records tend to be distributed sparsely over the space of possible records, and the propagation of errors is rare. The experimental results confirm this claim [7, 13, 4, 5].

3.2 An adaptive and efficient duplicate detection system

Under the assumption of transitivity, the problem of detecting duplicates in a database can be described in terms of keeping track of the connected components of an undirected graph. Let the vertices of a graph G represent the records in a database of size T. Initially, the graph will contain T unconnected vertices, one for each record in the database. There is an undirected edge between two vertices if and only if the records corresponding to the pair of vertices are found to match according to the pairwise record matching algorithm. At any time, the connected components of the graph G correspond to the transitive closure of the "is a duplicate of" relationships discovered so far. To incrementally maintain the connected components of an undirected graph we use the union-find data structure[14, 15].

The standard algorithm has another weakness in that the window used for scanning the database records is of fixed size. If a cluster in the database has more duplicate records than the size of the window, then it is possible that some of these duplicates will not be detected because not enough comparisons are being made. Furthermore if a cluster has very few duplicates or none at all, then it is possible that comparisons are being done which may not be needed. An approach is needed that responds adaptively to the size and homogeneity of the clusters discovered as the database is scanned, in effect expanding/shrinking the window when necessary. To achieve this, the fixed size window is replaced by a priority queue of duplicate records.

The system scans the sorted database with a priority queue of record subsets belonging to the last few clusters detected. The priority queue contains a fixed number of sets of records. Each set contains one or more records from a detected cluster. For efficiency reasons, entire clusters should not always be saved since they may contain many records. On the other hand, a single record may be insufficient to represent all the variability present in a cluster. Records of a cluster will be saved in the priority queue only if they add to the variability of the cluster being represented. The set representing the cluster with the most recently detected cluster member has highest priority in the queue, and so on.

Suppose that record R_j is the record currently being considered. The algorithm first tests whether R_j is already known to be a member of one of the clusters represented in the priority queue. This test is done by comparing the cluster representative of R_j to the representative of each cluster present in the priority queue. If one of these comparisons is successful, then R_j is already known to be a member of the cluster represented by the set in the priority queue. We move this set to the head of the priority queue and continue with the next record, R_{j+1}. Whatever their result, these comparisons are computationally inexpensive because they are done just with *Find* operations.

Next, in the case where R_j is not a known member of an existing priority queue cluster, the algorithm uses the matching algorithm to compare R_j with records in the priority queue. The algorithm iterates through each set in the priority queue, starting with the highest priority set. For each set, the algorithm scans through the members R_i of the set. R_j is compared to R_i using the matching algorithm. If a match is found, then R_j's cluster is combined with R_i's cluster, using a $Union(R_i, R_j)$ operation. In addition, R_j may also be included in the priority queue set that represents R_i's cluster – and now also represents the new combined cluster. Intuitively, if R_j is very similar to R_i, it is not necessary to include it in the subset representing the cluster, but if R_j is only somewhat similar then including R_j in the subset will help in detecting future members of the cluster.

On the other hand, if the comparison between R_i and R_j yields a very low score then the system continues directly with the next set in the priority queue. The intuition here is that if R_i and R_j have no similarity at all, then comparisons of R_j with other members of the cluster containing R_i will likely also fail. If the comparison still fails but the score is close to the matching threshold, then it is worthwhile to compare R_j with the remaining members of the cluster. These heuristics are used to counter the errors which are propagated when computing pairwise "is a duplicate of" relationships.

Finally, if R_j is compared to members of each set in the priority queue without detecting that it is a duplicate of any of these, then R_j must be a member of a cluster not currently represented in the priority queue. In this case R_j is saved as a singleton set in the priority queue, with the highest priority. If this action causes the size of the priority queue to exceed its limit then the lowest priority set is removed from the priority queue.

Earlier research showed that this adaptive duplicate detection system performs much fewer comparisons than previous work [3, 4, 5]. Fewer comparisons usually translates to decreased accuracy. However, similar accuracy was observed because the comparisons which are not performed correspond to records which are already members of a cluster, most likely due to the transitive closure of the "is a duplicate of" relationships. All experiments show that the improved algorithm is as accurate as the standard method while significantly performing many fewer record comparisons – as much as a 75% savings over the methods by [7, 13].

4 Conclusion

The integration of information sources is an important area of research. There is much to be gained from integrating multiple information sources. However, there are many obstacles that must be overcome to obtain valuable results from this integration.

This article has explored the problem of approximate duplicate detection. To integrate data from multiple sources, one must first identify the information which is common in these sources. Different record matching algorithms were presented that determine the equivalence of records from these sources. Section 2 presents the properties of such record matching algorithms to be applied to alphanumeric records that contain fields such as names, addresses, titles, dates, identification numbers, and so on.

The duplicate detection system described in this work and in [4, 5] improve previous related work in two significant ways. The first contribution is to show how to compute the transitive closure of "is a duplicate of" relationships incrementally, using the union-find data structure. The second contribution is a heuristic method for minimizing the number of expensive pairwise record comparisons that must be performed while comparing individual records with potential duplicates. These two contributions can be combined with any pairwise record matching algorithm. It is desirable for the record matching algorithms to be as domain-independent as possible and also for the algorithms to indicate the strength of the match (or non-match) between the records.

References

[1] A. E. Monge and 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*, p. 151, AAAI Press, Nov. 1995.

[2] A. E. Monge and C. P. Elkan, "The WebFind tool for finding scientific papers over the worldwide web," in *Proceedings of the 3rd International Congress on Computer Science Research*, (Tijuana, Baja California, México), pp. 41–46, Nov. 1996.

[3] A. E. Monge and C. P. Elkan, "An efficient domain-independent algorithm for detecting approximately duplicate database records," in *Proceedings of SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, (Tucson, Arizona), May 1997.

[4] A. Monge, *Adaptive detection of approximately duplicate database records and the database integration approach to information discovery*. Ph.D. thesis, Department of Computer Science and Engineering, University of California, San Diego, 1997. Available from University Microfilms International.

[5] A. E. Monge, "An adaptive and efficient algorithm for detecting approximately duplicate database records," June 2000. Submitted for journal publication.

[6] A. Silberschatz, M. Stonebraker, and J. D. Ullman, "Database research: achievements and opportunities into the 21st century." A report of an NSF workshop on the future of database research, May 1995.

[7] M. Hernández and S. Stolfo, "The merge/purge problem for large databases," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 127–138, May 1995.

[8] J. A. Hylton, "Identifying and merging related bibliographic records," M.S. thesis, MIT, 1996. Published as MIT Laboratory for Computer Science Technical Report 678.

[9] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James, "Automatic linkage of vital records," *Science*, vol. 130, pp. 954–959, Oct. 1959. Reprinted in [12].

[10] I. P. Fellegi and A. B. Sunter, "A theory for record linkage," *Journal of the American Statistical Association*, vol. 64, pp. 1183–1210, 1969.

[11] C. A. Giles, A. A. Brooks, T. Doszkocs, and D. Hummel, "An experiment in computer-assisted duplicate checking," in *Proceedings of the ASIS Annual Meeting*, p. 108, 1976.

[12] B. Kilss and W. Alvey, eds., *Record linkage techniques, 1985: Proceedings of the Workshop on Exact Matching Methodologies*, (Arlington, Virginia), Internal Revenue Service, Statistics of Income Division, 1985. U.S. Internal Revenue Service, Publication 1299 (2-86).

[13] M. Hernández, *A Generalization of Band Joins and the Merge/Purge Problem*. Ph.D. thesis, Columbia University, 1996.

[14] J. E. Hopcroft and J. D. Ullman, "Set merging algorithms," *SIAM Journal on Computing*, vol. 2, pp. 294–303, Dec. 1973.

[15] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. MIT Press, 1990.

[16] H. B. Newcombe, *Handbook of record linkage: methods for health and statistical studies, administration, and business*. Oxford University Press, 1988.

[17] M. I. Yampolskii and A. E. Gorbonosov, "Detection of duplicate secondary documents," *Nauchno-Tekhnicheskaya Informatsiya*, vol. 1, no. 8, pp. 3–6, 1973.

[18] U. C. Bureau, ed., *U.S. Census Bureau's 1997 Record Linkage Workshop*, (Arlington, Virginia), Statistical Research Division, U.S. Census Bureau, 1997.

[19] T. E. Senator, H. G. Goldberg, J. Wooton, and M. A. C. *et al.*, "The financial crimes enforcement network AI system (FAIS): identifying potential money laundering from reports of large cash transactions," *AI Magazine*, vol. 16, no. 4, pp. 21–39, 1995.

[20] Y. R. Wang, S. E. Madnick, and D. C. Horton, "Inter-database instance identification in composite information systems," in *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences*, pp. 677–84, Jan. 1989.

[21] R. S. Boyer and J. S. Moore, "A fast string-searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762–772, 1977.

[22] D. E. Knuth, J. H. M. Jr., and V. R. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, vol. 6, no. 2, pp. 323–350, 1977.

[23] P. A. V. Hall and G. R. Dowling, "Approximate string matching," *ACM Computing Surveys*, vol. 12, no. 4, pp. 381–402, 1980.

[24] Z. Galil and R. Giancarlo, "Data structures and algorithms for approximate string matching," *Journal of Complexity*, vol. 4, pp. 33–72, 1988.

[25] W. I. Chang and J. Lampe, "Theoretical and empirical comparisons of approximate string matching algorithms," in *CPM: 3rd Symposium on Combinatorial Pattern Matching*, pp. 175–84, 1992.

[26] M.-W. Du and S. C. Chang, "Approach to designing very fast approximate string matching algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, pp. 620–633, Aug. 1994.

[27] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics – Doklady 10*, vol. 10, pp. 707–710, 1966.

[28] J. Peterson, "Computer programs for detecting and correcting spelling errors," *Communications of the ACM*, vol. 23, no. 12, pp. 676–687, 1980.

[29] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 1981.