OPEN ACCESS

Efficient sequential and parallel algorithms for record linkage

Abdullah-Al Mamun,[1] Tian Mi,[1] Robert Aseltine,[2] Sanguthevar Rajasekaran[1]

[1]Department of Computer Science and Engineering, University of Connecticut, Storrs, Connecticut, USA
[2]Institute for Public Health Research, University of Connecticut, East Hartford, Connecticut, USA

Correspondence to
Dr Sanguthevar Rajasekaran, Department of Computer Science and Engineering, University of Connecticut, 371 Fairfield Road, Unit 2155, Storrs, CT 06269-2155, USA; rajasek@engr.uconn.edu

Received 29 May 2013
Revised 1 October 2013
Accepted 5 October 2013
Published Online First 23 October 2013

ABSTRACT

Background and objective Integrating data from multiple sources is a crucial and challenging problem. Even though there exist numerous algorithms for record linkage or deduplication, they suffer from either large time needs or restrictions on the number of datasets that they can integrate. In this paper we report efficient sequential and parallel algorithms for record linkage which handle any number of datasets and outperform previous algorithms.

Methods Our algorithms employ hierarchical clustering algorithms as the basis. A key idea that we use is radix sorting on certain attributes to eliminate identical records before any further processing. Another novel idea is to form a graph that links similar records and find the connected components.

Results Our sequential and parallel algorithms have been tested on a real dataset of 1 083 878 records and synthetic datasets ranging in size from 50 000 to 9 000 000 records. Our sequential algorithm runs at least two times faster, for any dataset, than the previous best-known algorithm, the two-phase algorithm using faster computation of the edit distance (TPA (FCED)). The speedups obtained by our parallel algorithm are almost linear. For example, we get a speedup of 7.5 with 8 cores (residing in a single node), 14.1 with 16 cores (residing in two nodes), and 26.4 with 32 cores (residing in four nodes).

Conclusions We have compared the performance of our sequential algorithm with TPA (FCED) and found that our algorithm outperforms the previous one. The accuracy is the same as that of this previous best-known algorithm.

INTRODUCTION

Identifying duplicates in voluminous datasets is a crucial problem in many areas of science and engineering. This is especially true for medical records of individuals from different health agencies. Integration of medical records provides a great opportunity to analyze and evaluate disease evolution.[1 2] Methods[3] exist for linking records across multiple medical data centers to identify disease origin and diversity.[4] Copy detection in digital documents also employs data integration techniques to detect similarities.[5 6] Data integration techniques integrate records across different data sources, usually in the absence of any global identifier. This is a way to identify individuals who have records in different datasets. If all the records pertaining to the same individual are exactly correct, the problem of identifying duplicates will be straightforward to solve. Unfortunately, records of the same person might look different owing to errors introduced by typing, phonetic similarity, etc. As a result, the record linkage problem is very challenging. Existing algorithms take a very long time, especially when the data size is large. Thus, it is still an important open problem to discover faster algorithms. In this paper we propose a sequential algorithm that is up to two orders of magnitude faster than one of the prior algorithms, the two-phase algorithm using faster computation of the edit distance (TPA (FCED)).[7] We also present a parallel algorithm that achieves a nearly linear speedup.

A huge number of approaches have been developed in the literature. Most of these algorithms link two datasets at a time. In practice, we have much more than two datasets. If we have two datasets A and B and if n_a and n_b are the numbers of records in them, respectively, then in the worst case we have to process $n_a \times n_b$ record pairs.[8] Some learning algorithms generate comparison vectors and classify them,[9] which takes a large amount of time to generate the vectors.

As the basis for our algorithms, we have used hierarchical clustering,[10 11] which is also widely applied in information theory,[12] gene expression,[13–16] data mining,[17 18] health psychology,[19] and many other fields to identify distributions of corresponding objects or data. Our algorithms use the single linkage method to calculate distances. To reduce load on calculating linkages, we employ radix sort initially on records.[20] Our algorithms also consider different types of errors including typing distance, reversal of the first name and the last name, use of nicknames, truncation of attributes, etc.[7] We have thoroughly tested our algorithms on a large number of synthetic and real datasets. These tests show that the proposed algorithms outperform previous algorithms in terms of time and space. The parallel algorithm achieves a very nearly linear speedup.

BACKGROUND AND SIGNIFICANCE

Record linkage among multiple datasets typically involves millions of records and hundreds of thousands of individuals. The problem of record linkage can be thought of as one of clustering the records such that each cluster has records pertaining to one and only one individual.[21] Clustering, in general, is the process of partitioning objects so that similar objects are grouped into the same group (ie, cluster). A number of clustering methods can be found in the literature, including hierarchical clustering, graph-based clustering, statistical clustering, and centroid based clustering. Any clustering method employs a metric (known as linkage) for defining the distance between two clusters. Distance between two clusters indicates how similar

To cite: Mamun A-A, Mi T, Aseltine R, *et al*. *J Am Med Inform Assoc* 2014;**21**: 252–262.

these two clusters are. In complete linkage, distance between two clusters (of records) A and B is defined as the maximum distance between a record in A and a record in B, while single linkage uses the minimum distance. The distance between two given records can also be defined in a number of ways. Examples include the Levenshtein distance (also known as the edit distance) and the Hamming distance.

Hierarchical clustering can be done in two different ways: (1) The agglomerative approach (bottom-up) starts with n clusters (where n is the number of records or points to be clustered), where each cluster has a single point. From there on clustering happens in iterations where in each iteration the two closest clusters are merged into one. Iterations stop when we have only a single cluster containing all the n points. The sequence of merging steps done in the algorithm can be represented as a tree called a dendrogram. If we have a target number of clusters in mind, we can cut the dendrogram at an appropriate level. The dendrogram can also be cut using a cluster threshold distance. (2) The divisive clustering approach (top-down) starts with a single cluster having all the n points. This cluster is then split hierarchically until we end up with n clusters, each cluster having a single point.

In this paper we employ agglomerative hierarchical clustering, using single linkage. We treat each record as a string of characters and define the distance between two records based on edit distance. Different kinds of common errors have been taken into account: reversal of first and last names, truncation of attributes, etc.[7]

METHODS

Previous methods

A simple brute force approach for record linkage is to compute the distance between every pair of records and identify the pair as a match or a non-match. This would take too much time. Some of the previous methods generate comparison vectors and define classification.[9] Cluster-based entity resolution that uses both relational and attribute information has been shown to perform better than attribute-based record linkage.[22] Linking several datasets using record linkage methods[23] and deduplication[24] to merge records and remove repetitions are popular techniques. A wide range of studies on methods for record linkage have been done.[25] The expectation-maximization (EM) algorithm provides improved decision rule in the Fellegi–Sunter model of record linkage by employing probability estimation.[26] Traditional probabilistic linkage models classify pairs of records as matches if they agree on some of their common attributes, and non-matches otherwise.[27] The probabilistic linkage system AutoMatch results in better linkage quality than some deterministic ones, as shown in a recent study.[8] Many other probabilistic methods also exist.[28–30] Identity uncertainty and citation matching problems have been solved by the relational probability model.[31] Conditional models also cover the problem of identity uncertainty.[32] Conditional random fields have been used to segment and label data.[33] These are also applied in a relational partitioning algorithm.[34] Multi-relational record linkage allows propagation of matches.[35] Personal name matching techniques,[36] distance calculation,[37] matching methods,[38] automated correction of text techniques,[39] longest common substring,[40] and many other techniques are also available for comparisons.

FEBRL is famous for the linkage of two datasets.[41 42] IntelliClean is another framework to identify duplicates by computing the transitive closure under uncertainty and anomalies efficiently.[43] The multi-pass approach for merge/purge problem considers alternate key attributes and applies these results to compute the transitive closure.[44] Many of these techniques use a blocking phase as a preprocessing step where the records are hashed into buckets (or blocks) based on some of the characters in the records, including canopy clustering.[45] Unsupervised and unconstrained partition-based clustering algorithms exist which are different from hierarchical clustering methods.[46] We have improved the TPA (FCED) algorithm, which is one of the fastest known record linkage algorithms, significantly.

Some parallel algorithms for hierarchical clustering have been developed.[47–50] Parallel methods for record linkage also exist.[51–54] P-Swoosh uses match and merge processes, and also uses domain knowledge.[51] An algorithm that performs better than P-Swoosh has been reported.[52] This algorithm achieves an almost linear speedup, for example 6.55–7.49 on eight processors. A different blocking technique in initial data partitioning followed by a matching phase has also been introduced.[53 54] Algorithms that we propose in this paper are based on single linkage hierarchical clustering. Single linkage has been shown to perform better, from a time complexity perspective, over complete linkage and average linkage.[48] An analysis on different linkages in hierarchical clustering has also been conducted.[55]

Our approaches

Naïve algorithms for record linkage take $O(n^2L^2)$ time, where n is the number of records and L is the maximum length of any record. The length of any record is nothing but the total aggregated length of all the attributes employed in the record linkage analysis. When the data size is very large, these algorithms take a very long time. Thus it was an important open problem to devise faster algorithms. To make the record linkage process faster and more reliable, we propose a very fast sequential algorithm and a parallel algorithm.

Sequential algorithm

The proposed algorithm is independent of the number of datasets. Thus, we are able to integrate data from any number of datasets in an elegant way. It is true that any algorithm that links two datasets can be employed to integrate more than two datasets by invoking the algorithm multiple times, each time integrating two. For example, if we have three datasets A, B, and C, we can first merge A and B to get A′ and then merge A′ and C. However, the output and accuracy of this approach will depend on the order in which these pairwise merges are done. In our sequential algorithm called RLA (record linkage algorithm), we collect all the records from all the datasets and form a collection X; we sort X after concatenating some or all of the common attributes (first name, last name, gender, address, etc.) in each record. Using this sorted list exact duplicates are eliminated. Two records are treated as identical if they agree on the common attributes. Note that in any record linkage algorithm record distances are calculated using only these common attributes. Let X′ be the set of records remaining after the elimination of duplicates. Clustering is performed on X′. We use blocking on X′ based on l characters of the last names (for some suitable value of l). Blocking may be done on last name, first name, or any other relevant attribute. In our experiments on real datasets we have realized that the use of last names yields the best accuracy. Each block consists of records that share an l-mer (ie, a substring of length l) in the last names. An l-mer is also referred to as an l-gram in the literature. Two records r_1 and r_2 will be in the same block if they share at least one l-mer in their last names. Since a record might share an l-mer with many other records, it could be in many different blocks. If q is the maximum number of blocks that a record is in and if n′ is

the number of records in X′, then the expected size of each block is $qn'/26^l$, assuming the English alphabet. Single linkage and edit distance are used for the clusters and records, respectively. Instead of constructing the entire dendrogram, we utilize a threshold τ (an input parameter) to generate a partial dendrogram that has only edges with distances no more than τ. Then a graph G(V, E) is generated in which V is X′. Two nodes in V have an edge between them if and only if they are in the same cluster of the partial dendrogram from some blocking. Thus, each connected component of G contains the records pertaining to one individual.

Algorithm 1: RLA

1. Collect all the records from all the datasets and form a single list X.
2. Sort the records in X and form groups such that each group consists of identical records. Pick one record from each such group and let X′ be the resultant collection of records.
3. Do blocking on X′. Specifically, there could be a block for every possible l-mer. (Note that there are 26^l possible l-mers when the alphabet corresponds to English.) Consider one such l-mer y. If two records have y as an l-mer in their last names then these two records will be in the block corresponding to y. If there is an l-mer y′ that does not occur in the last name of any record, then the block corresponding to y′ will be empty. Also, the same record could be in many different blocks. So a record is going to be in (L − l+1) blocks where L is the length of this record and the blocking size is l.
4. Cluster every block obtained in step 3. Employ hierarchical clustering with single linkage. Specifically, two records r_1 and r_2 will belong to the same cluster if the distance between them is no more than τ. We have employed a fast algorithm for computing the edit distance between two records. This algorithm, also used in Mi *et al*,[7] takes O (τk) time where k is the minimum of the two record lengths and τ is the specified threshold.[7]
5. We generate a graph G(V, E) where V is the collection X′. Two records have an edge between them if there exists at least one cluster in at least one block in which both of these records belong.
6. Find the connected components of G(V, E).
7. Output each connected component as a cluster. While outputting a connected component, also output records that are identical to records in the component. (Note that information about identical records is available from step 2).

Analysis

The most time-consuming part of the proposed algorithm is the calculation of linkages between records in blocks to generate the graph G(V, E). Let b be the number of blocks in X′, b_a be the average number of records in a block, L be the maximum length of a record, n′ be the number of records in X′, and τ be the threshold on the distance. The time complexity of algorithm 1 (steps 3–7) is $O(bb_a^2L\tau)$. In practice we have noted that $bb_a=O(n')$ and hence it takes $O(n'b_aL\tau)$ time for steps 3–7. Clearly, the smaller the value of n′ the better will be the run time. Steps 1 and 2 of algorithm 1 take time that is linear in the size of X. We refer to the average number of (identical) duplicates we have for each record as multiplicity. Another prominent idea we have applied is to cache misses. As the cache memory of each processor is limited and most of the times it is not enough to hold all the records, cache misses occur frequently. We handle this issue by copying frequently needed data into a separate array so that these data will be in contiguous memory locations. TPA (FCED) consumes a considerable amount of time in removing duplication of linkages. We have cut this amount of time by considering a graph-based solution where we find connected components in linear time.

Parallel algorithm

We have parallelized the sequential algorithm (parallel record linkage algorithm, or PRLA), which achieves nearly linear speedups. We keep a copy of the input list X with each processor. One of the processors is identified as the master and the other processors are called slaves. Let p be the number of slaves. The steps in the algorithm are enumerated below.

Algorithm 2: PRLA

1. The master broadcasts all the input records to the slave processors.
2. Each processor sorts a portion of X in parallel. Specifically, the records of X are grouped based on the first two characters of the last names. Note that there are 26^2 possible 2-mers of characters and hence there are these many possible groups (some of which could be empty). Each processor sorts $26^2/p$ groups. As a by-product of this sorting, each processor picks a representative from every group of identical records that it sorted. In other words, we form X′. The slaves inform the master about their findings.
3. The master assigns $|X'|/p$ number of records from X′ to each processor for the purpose of blocking. Each processor then performs blocking on its records and sends the blocks information to the master.
4. The master aggregates the blocks. In particular, let y be some possible l-mer. Parts of the block corresponding to y could be with multiple processors. The master merges these partial blocks.
5. Let $B_1, B_2, \ldots, B_t$ be the blocks in X′. Note that $t \leq 26^l$, where l is the blocking size. Let $n_i = |B_i|$, for $1 \leq i \leq t$. The master sorts $n_1^2, n_2^2, \ldots, n_t^2$ values in descending order. Let $s = \sum_{i=1}^{t} n_i^2$. The master then distributes the blocks among the processors so that the work assigned to each processor is nearly even. Specifically, the distribution is such that the sum of squares of block sizes assigned to any processor is nearly s/p.
6. The next task is to generate the graph G(V, E). To do this, each processor finds the edges in its blocks along the same lines as in the sequential algorithm. All of these edges from all the processors are sent to the master.
7. The master finds the connected components in the graph. These connected components together with the initially removed copies of records yield us the clusters of interest.

Analysis

Let n be the number of records and n′ be the number of distinct records in the input. Let L be the maximum length of any record in the input.

In step 1, the broadcasting takes O(n) time. Grouping in step 2 can be done by sorting the records based on two characters and hence this sorting step takes O(n) time as well. Once the groups are formed (based on two characters), we can expect each group to have $n/26^2$ records and hence the sorting of groups takes an expected O(n/p) steps. The communication of the slaves with the master takes O(n) time.

In step 3, the master sends a subset of X′ to each of the slaves. This communication takes O(n′) time. If l is the blocking size, then, each processor spends $O(n'/p(L-l+1))$ time in forming the blocks. Note that there will be a total of 26^l blocks.

Each slave sends the master information about its blocks. In particular, for every block it sends a list of indices of all the records that belong to this block. As a result, the amount of information sent from each slave to the master is $O(n'/p(L-l+1))$. Therefore, the total communication time in this step is O(n′(L −l+1)).

In step 4, aggregation of the blocks received from all the slaves in step 3 is done in O(n′(L−l+1)) time by the master. Then a sorting is done on the list of sizes of the blocks. This takes $O(26^l)$ time using radix sort.

In step 5, the blocks are distributed among the slaves such that the value of s is nearly balanced across the slaves. Note that this problem is NP-complete. We use the sum of squares of block sizes to compute s for the following reason. To compute the edges within each block, in the worst case, each record is compared with every other record. As a result, the worst case time spent on each block is proportional to the square of the block size. We have tried several ways of distributing the blocks. In each of these ways, a block might get split between two adjacent processors to ensure a close partitioning. Therefore, each of the techniques we have employed does not guarantee an exactly even partitioning (or an optimal partitioning). One simple partitioning we have used is to use the sorted list $Q = n_1^2, n_2^2, \ldots, n_t^2$. We will identify a minimum prefix of this sequence whose sum equals or exceeds s/p. If this prefix sum equals s/p, then this prefix sequence of blocks will be assigned to the first processor. If this prefix sum exceeds s/p, then the last block in this prefix sequence will be split between the first and the second processors. The splitting will be done to ensure that the work assigned to the first processor is as close to s/p as possible. By the work assigned to a processor we mean the sum of squares of the blocks assigned to the processor. In the case of the prefix sum exceeding s/p, a portion of the last block in this prefix sequence will be assigned to the second processor. The second processor will also be assigned the next number of blocks in the sorted sequence Q. This number of blocks will be such that the work assigned to this processor is nearly s/p, and so on. The time taken by the master in step 5 is O(t) where t is the number of blocks. If the blocking size is l, then $t \leq 26^l$. After this, the master creates a list of records for each slave to work on. This takes O(n′(L−l+1)) time. Subsequently, the master sends the individual lists to the slaves. This communication also takes O(n′(L−l+1)) time.

In step 6, each processor works on its blocks. The time spent in this step is O(s/p). Note that the expected size of each block is $n'(L-l+1)/26^l$. Also, the time spent in computing the distance between any two records is O(τL). Thus the expected value of s is $((n')^2(L-l+1)^2/26^l)\tau L$. Our empirical results indicate that the total number of edges generated across all the processors is O(n′(L−l+1)). In this case, the communication time is O(n′). As a result, the connected components in step 7 can also be found in O(n′(L−l+1)) time.

In summary, the total expected run time of the algorithm is $O(n + n'(L-l+1) + ((n')^2(L-l+1)^2/p \times 26^l)\tau L)$. It turns out that the last term is the dominating one among the three terms in this time complexity. Table 4 explains why we get a speedup that is close to linear. Please note that blocking is quite useful in reducing the run time. For example, even if L=15, for a value of l=3, the value of $(n')^2(L-l+1)^2/26^l$ is 0.0096 $(n')^2$.

Also, the run times of most of the (sequential and parallel) algorithms found in the literature depend on n^2. Thus the work done by our algorithm is expected to be significantly better than competing algorithms since our run time depends on $(n')^2$. In practice the value of $(n')^2$ is much smaller than that of n^2. Although parallel algorithms exist (see Greiner,[56] for example) for finding connected components, we have not used them here since the time needed for this step is very small.

RESULTS

We have implemented our sequential version for simulated data in C++ to make a better comparison with the parallel version, as PRLA has been implemented using MPI with C++. We have also used C++ implementation of the TPA (FCED) algorithm to compare with our sequential version. As TPA (FCED) was originally implemented in java, we have also implemented our algorithm in java to make a fair comparison with the results in Mi *et al*.[7] Our sequential algorithm outperforms TPA (FCED),[7] especially when the multiplicity is large.

We have tested our algorithms on both synthetic and real data. We have collected real datasets from the Connecticut Health Information Network (CHIN). As TPA (FCED)[7] ensures very high accuracy of record linkage but consumes a large amount of time, our main purpose was to provide a much faster solution. So we have developed our algorithms in such a way that the accuracy remains the same, but the algorithms run much faster. In the blocking phase, we have used 4-mer for all the experiments. The value of l in the blocking phase has to be chosen carefully. If l is low, the accuracy will be high. A higher value will result in a reduction in the run time but the accuracy might suffer.

Results on simulated data for the sequential algorithm

The implementation has been deployed in the HORNET cluster housed in the Booth Engineering Center for Advanced Technology (BECAT), University of Connecticut. This cluster has 64 nodes, each of which has 12 Intel Xeon X5650 Westmere cores, 48 GB of RAM, and 500 GB of local storage.

Running time of our algorithms is independent of the number of datasets as we add all the records to a single list and work with only this list. As in TPA (FCED),[7] we have employed both constant and proportional threshold values in the clustering step. Our algorithm has been tested for each type of distance calculation. The total number of records used for this test ranges from 50 000 to 5 000 000 to reveal the power of our algorithm. Five records have been generated for each individual, in which four are error free. So, on the five records of any individual, exact clustering will find two clusters.

To compare with TPA (FCED), we employ edit distances of two attributes, namely the first name and the last name. TPA (FCED) spends around 650.49 s for 1 000 000 data whereas our algorithm takes only 92.99 s, which is seven times faster for this amount of data. Table 1 summarizes the comparison. Figure 1 provides a graphical representation of this comparison.

When the input data contains a large number of records, TPA (FCED) spends too much time to complete. Table 2 displays the time taken by RLA on various steps.

When we have 1 000 000 records, finding clusters using exact matching (steps 3–6 in the sequential algorithm) takes only 8.8 s. The size of X′, after removing duplicates, is only 387 707. From table 1, we see that TPA (FCED) takes around 178.74 s to find clusters for 400 000 records. But RLA clusters 387 707 records by approximate clustering within 83.21 s. This improvement is because of the graph-based solution and the avoidance of cache misses. So clustering of 1 000 000 records takes only 92.99 s. Even when the multiplicity is 1, our algorithm runs around two times faster than TPA (FCED). Since in practice the multiplicity of data is more than 1, our algorithms run much

Table 1 Comparison of results on simulated data

Number of records	Algorithm	Run time in seconds
50 000	TPA (FCED)	7.35
	RLA	1.19
100 000	TPA (FCED)	24.81
	RLA	3.67
200 000	TPA (FCED)	71.47
	RLA	10.25
400 000	TPA (FCED)	178.74
	RLA	26.09
600 000	TPA (FCED)	324.82
	RLA	45.99
800 000	TPA (FCED)	489.43
	RLA	68.67
1 000 000	TPA (FCED)	650.49
	RLA	92.99
2 000 000	TPA (FCED)	1844.52
	RLA	256.51
3 000 000	TPA (FCED)	–
	RLA	490.54
4 000 000	TPA (FCED)	–
	RLA	800.02
5 000 000	TPA (FCED)	–
	RLA	1123.85

–, the algorithm took too long to terminate.

Table 2 Analysis of results on simulated data (RLA)

Number of records	Number of exact clusters	Number of clusters	Exact cluster time	Approx cluster time	Merge time	Total time
50 000	19 582	12 130	0.21	0.95	0.03	1.19
100 000	39 201	23 965	0.48	3.11	0.08	3.67
200 000	78 453	46 487	1.11	8.97	0.17	10.25
400 000	156 934	88 725	2.71	23.04	0.34	26.09
600 000	232 866	130 746	4.67	40.74	0.58	45.99
800 000	309 615	173 617	6.52	61.42	0.73	68.67
1 000 000	387 707	214 912	8.8	83.21	0.98	92.99
2 000 000	771 004	427 269	27.21	227.31	1.99	256.51
3 000 000	1 154 323	639 501	45.07	442.46	3.01	490.54
4 000 000	1 537 531	851 729	61.69	732.89	5.44	800.02
5 000 000	1 920 723	1 064 825	77.14	1041.28	5.43	1123.85

faster as shown in figure 1. Our proposed algorithm is more than 20 times faster than the previous algorithm TPA (FCED) on the datasets of records having a multiplicity of 5. Figure 2 is a graphical representation of table 2.

A similar experiment, which uses reversal edit distance, also shows superiority of the RLA algorithm. Reversal edit distance takes in two groups of attributes, calculates edit distance in both original direction and reversal direction, and returns the smaller one. In our experiments, we aggregate the edit distance of the first attributes of the two records and the edit distance of the second attributes of them. Again we add the edit distance between the first attribute of the first record and the second attribute of the second record and the edit distance between the second attribute of the first record and the first attribute of the second record. We then take the smaller of these two distances, as this is the reversal distance value. Figure 3 shows almost the same efficiency for RLA on this distance as well.

But in this case, both the algorithms take more time than that for the previous distance calculation as two edit distances are needed to be calculated as per the definition of reversal distance.

We have performed another experiment using edit distance as the distance method but adding a parameter, namely truncation count. We have used a truncation count of 2, which means that we only employ the first two characters of any attribute concerned. Both the algorithms produce more clusters in this case. The process is slow since more linkages will have to be dealt with. Figure 4 shows the comparison.

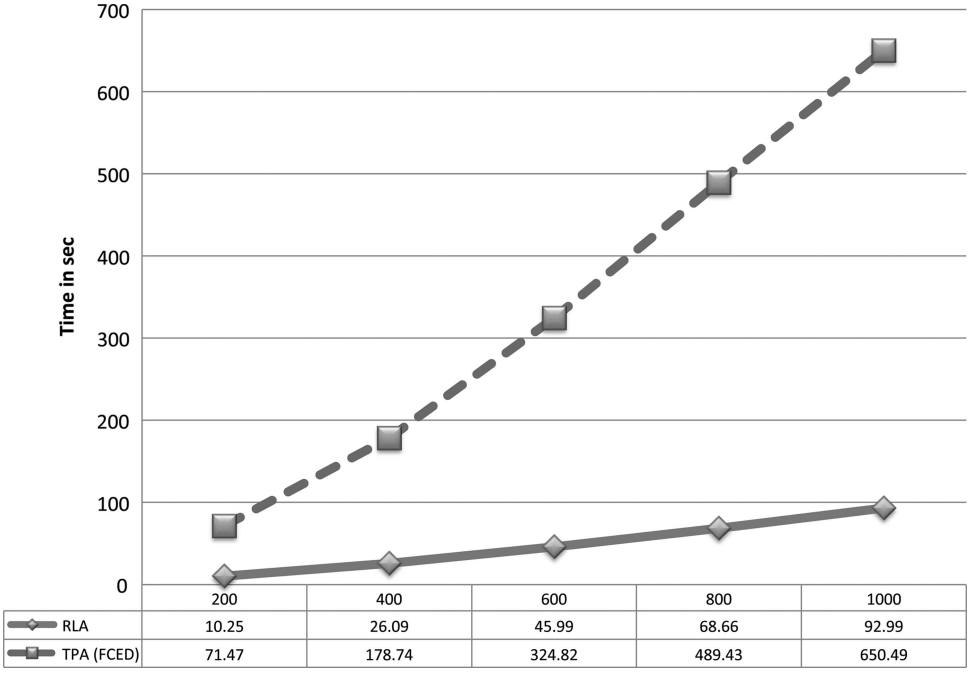


Figure 1 Results on synthetic data (y axis denotes time in seconds; x axis corresponds to number of records in thousands). RLA, record linkage algorithm; TPA (FCED),two-phase algorithm using faster computation of the edit distance.

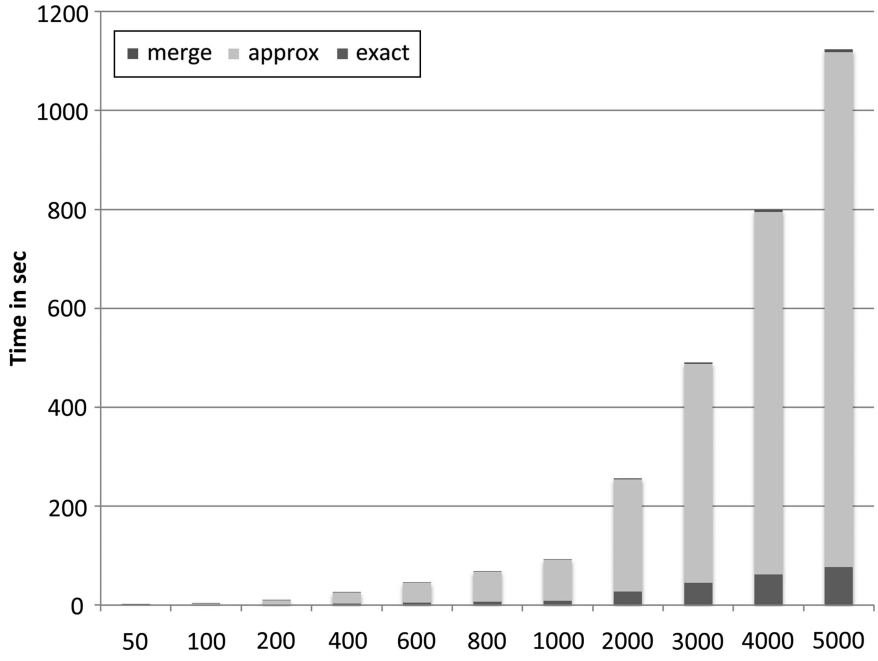


Figure 2 Analysis of results on synthetic data using the record linkage algorithm (y axis denotes time in seconds; x axis corresponds to number of records in thousands).

In the above cases, we have used constant threshold to find clusters. The next test shows results for using proportional threshold, which is dependent on the length of the considered attributes. Results are shown in figure 5.

Proportional threshold sometimes works better as it is dependent on the data. We omit details on the proportional threshold, as the procedure is similar to the constant threshold.

Clearly, the threshold has a great impact on the accuracy of clusters as a too small or too large threshold will normally yield a low error-rate. That is why a training phase is needed to learn the threshold.

Results on real data for the sequential algorithm

Our experiments on real data have been conducted on the CHIN server for security reasons. The computer has a CPU of Intel(R) Xeon(R) X5460, 3.16 GHz, and 4 GB RAM. The data come from four different datasets having a total of 1 083 878 records.

Table 3 shows the comparison. RLA employs two attributes, namely the first name and the last name. Within 15 s, it outputs 112 404 exact clusters. The rest of the steps take around 19 s. The algorithm terminates within 34.5 s whereas TPA (FCED) spends around 2961 s. RLA is 85 times faster than TPA (FCED) for this real data. The accuracy is 93.0% for both.

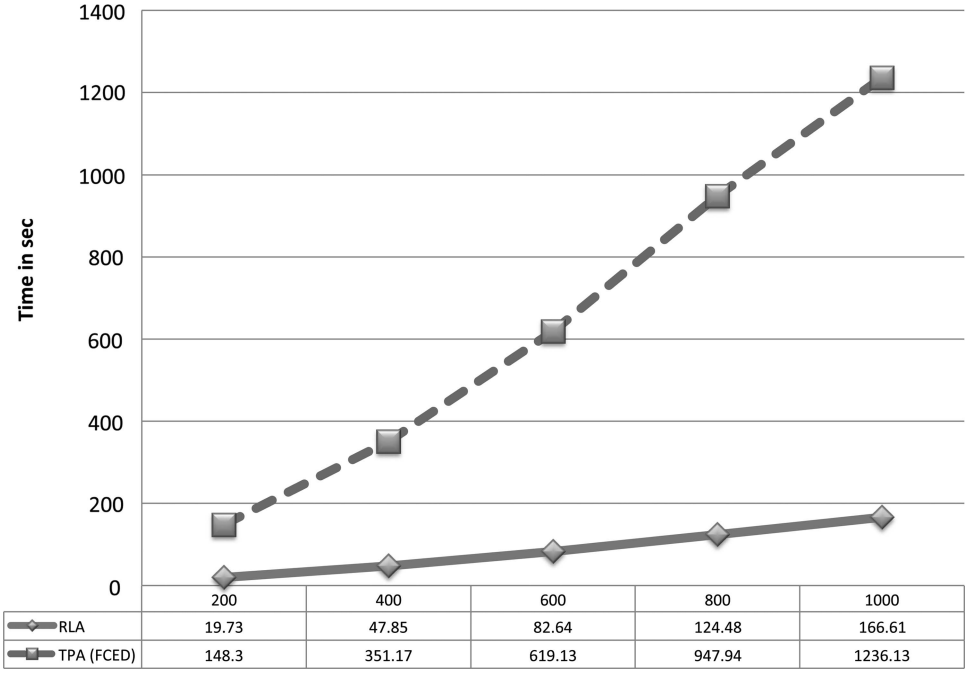
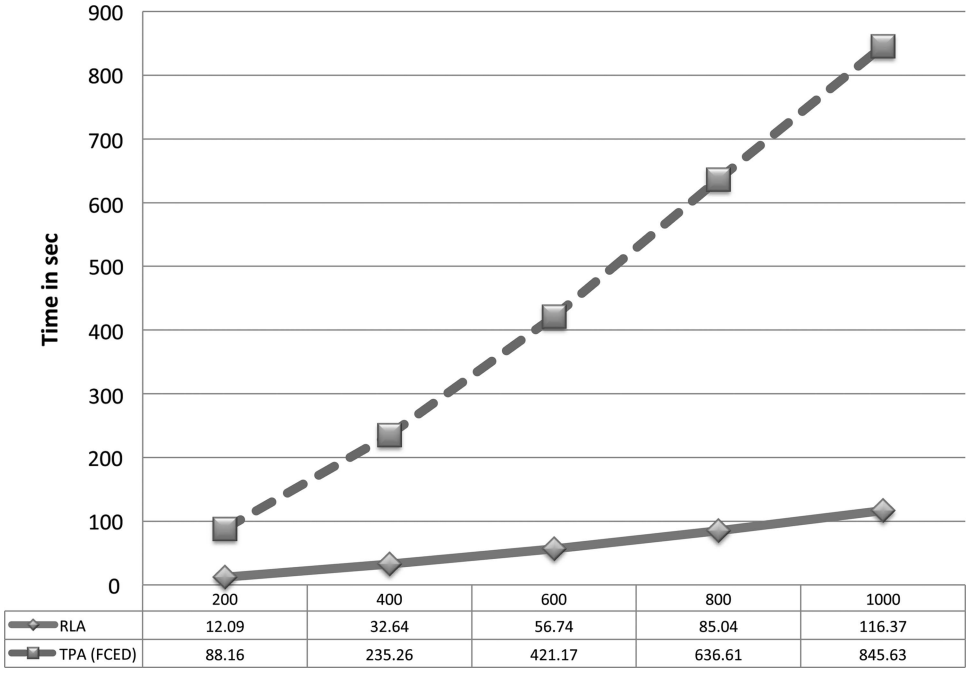


Figure 3 Comparison on reversal edit distance (y axis denotes time in seconds; x axis corresponds to number of records in thousands). RLA, record linkage algorithm; TPA (FCED),two-phase algorithm using faster computation of the edit distance.



	200	400	600	800	1000
RLA	12.09	32.64	56.74	85.04	116.37
TPA (FCED)	88.16	235.26	421.17	636.61	845.63

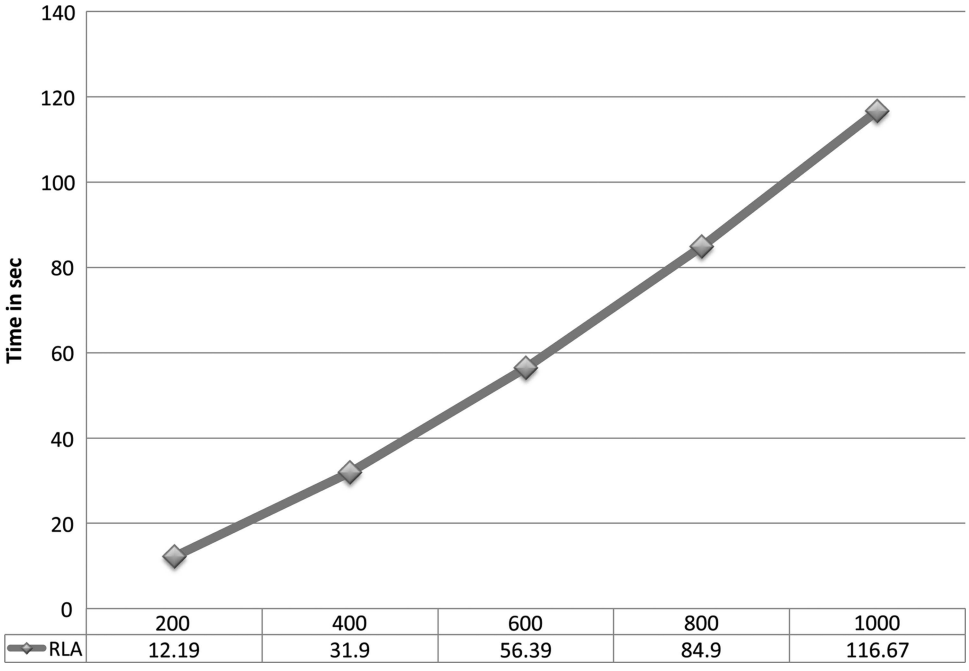
Figure 4 Comparison on truncation edit distance (y axis denotes time in seconds; x axis corresponds to number of records in thousands). RLA, record linkage algorithm; TPA (FCED),two-phase algorithm using faster computation of the edit distance.

We have also used the date of birth attribute in addition to the above two attributes. The running time is also impressive. RLA takes only 48.7 s whereas TPA (FCED) takes 3402 s. In this case, RLA is 70 times faster; 97.8% accuracy is achieved for these data since the use of a larger number of attributes removes many occurrences of false positives.

Results on simulated data for the parallel algorithm

In our experiments, we have used at most 32 cores from four nodes, eight from each node. In this case, we have used another set of synthetic data, in which the multiplicity is nearly 1.

An algorithm is fully parallel when the speedup is linear. We have optimized our algorithm to make it almost linear. Table 4



	200	400	600	800	1000
RLA	12.19	31.9	56.39	84.9	116.67

Figure 5 Results on synthetic data using proportional threshold (t=0.1, y axis denotes time in seconds; x axis corresponds to number of records in thousands). RLA, record linkage algorithm.

Table 3 Results on real datasets (1 083 878 records)

Number of attributes	Algorithm	Time (s)	Created clusters	Correct clusters	Number of individuals	Accuracy %	Com. %
2	TPA (FCED) RLA	2961 34.5	94 381	87 756	108 800	93.0	80.7
3	TPA (FCED) RLA	3402 48.7	101 864	99 562	108 800	97.8	91.6

analyzes the running time of PRLA for 6 million records. The first column shows the number of cores used. The total time spent in broadcast operations that take place in steps 1, 3, and 5 is shown as bcast. The total time for the other communications that happen in steps 2, 3, 4, and 6 is shown as *comm*. As we can readily see, these communication overheads are very low. Master performs certain tasks on its own in steps 3, 4, 5, and 7. This total time is displayed as *master* in table 4. The time for sorting and finding duplicates in step 2 is *dedup*. The total time for blocking (*block*, in step 3), merging (*merge*, in step 4), distribution of blocks (*dist*, in step 5), and finding connected components (*concomp*, in step 7) is very low as well. Generating edge lists is the major time consuming step. This time is shown as *edgelist*. The fact that this step dominates the entire run time is also revealed in our time complexity analysis above. The first row, *seq*, shows the runtime consumed by sequential RLA. Figure 6 graphically describes the data in table 4.

The time results are also shown in figure 7. The x-axis represents the number of cores used and the y-axis shows time in seconds.

Our results show that the speedup is around 7.5 for eight cores (that reside in a single node), 14.1 for 16 cores (residing in two nodes), and 26.4 for 32 cores. Values show almost linearity in speedup (figure 8). We have tested on 1, 2, 4, 8, 16, and 32 cores.

DISCUSSION

Our algorithms ensure the same accuracy as the previous algorithm TPA (FCED). Accuracy and completeness have been calculated on real dataset. Social Security Number (SSN) or DDS identification number was available for these records that we

Table 4 Distribution of running time for 6 000 000 records

Pr Seq	bcast	comm	master	dedup	block	merge	dist	edgelist	concomp	Total time 4861.7	Speedup
1	0	0	3.95	74.44	4.07	0.26	50.22	4719.8	0.69	4853.5	1.00
2	0.06	0.8	3.75	54.56	2.21	0.16	49.15	2329.0	0.6	2440.3	1.99
4	0.1	0.38	3.19	21.3	1.19	0.11	19.68	1220.9	0.54	1267.4	3.84
8	0.23	0.34	2.96	12.04	0.7	0.08	10.03	622.18	0.54	649.1	7.50
16	2.56	1.99	2.76	8.35	0.4	0.07	2.72	325.58	0.67	345.1	14.1
32	3.48	2.08	2.52	5.73	0.21	0.06	0.71	169.0	0.53	184.3	26.4

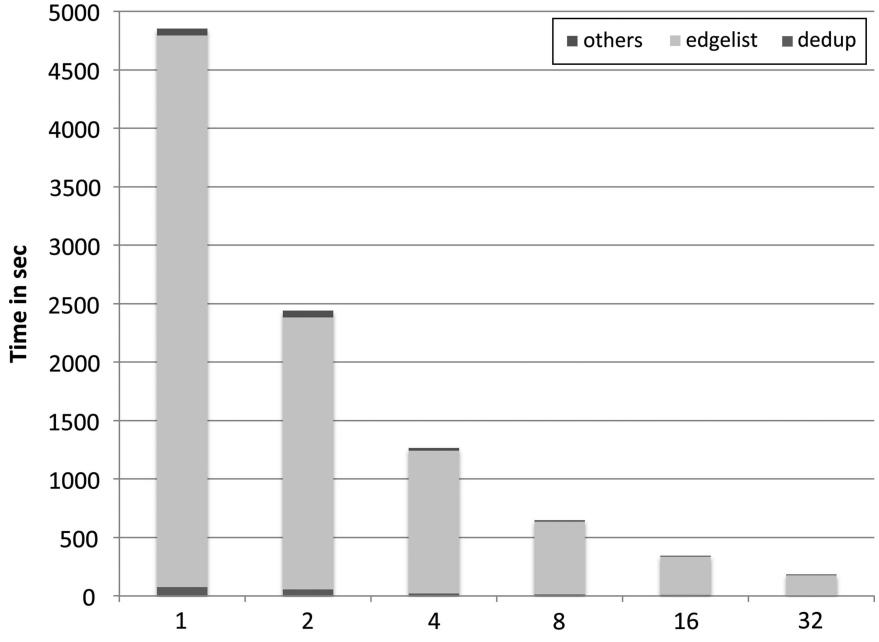
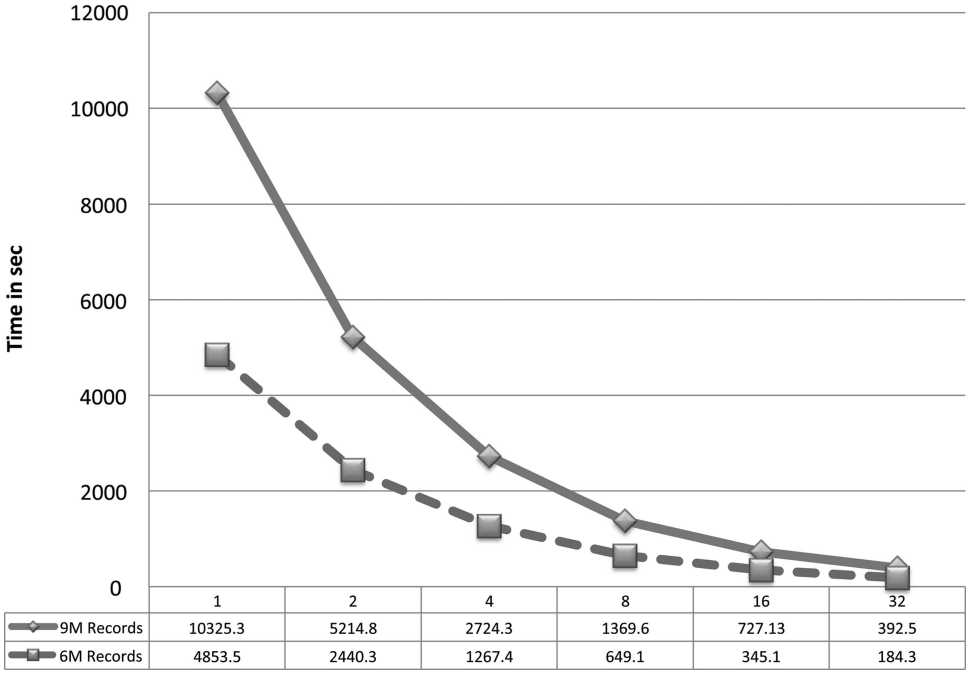


Figure 6 Analysis of results on synthetic data using the parallel record linkage algorithm (y axis denotes time in seconds; x axis corresponds to number of processors).



	1	2	4	8	16	32
9M Records	10325.3	5214.8	2724.3	1369.6	727.13	392.5
6M Records	4853.5	2440.3	1267.4	649.1	345.1	184.3

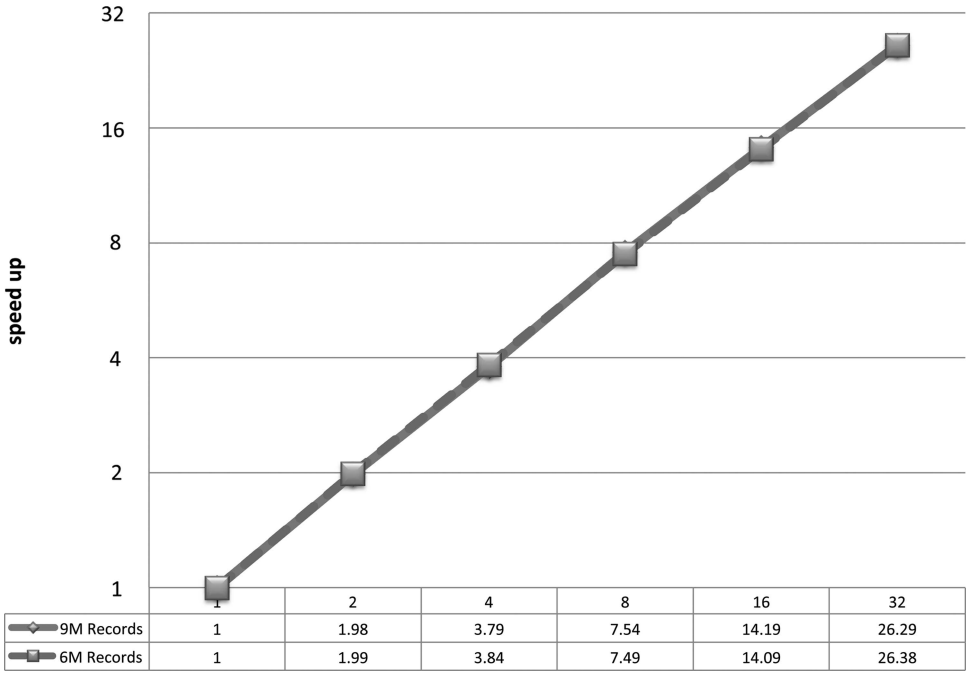
Figure 7 Results on simulated data (for 6 million and 9 million records; y axis denotes time in seconds; x axis corresponds to number of processors).

utilized for calculating the accuracy. These numbers were revealed to us only after our algorithms produced the results.

To cluster records more accurately, an appropriate threshold value is necessary. Such a threshold can be obtained in a learning process as described in Mi *et al*.[7] The idea is to have a training phase in which records for which the right clustering is known will be utilized. The whole procedure is described elaborately in Mi *et al*.[7] We have used a constant threshold value of 1 and a proportional threshold value of 0.1.

Besides using edit distance, we have also employed reversal edit distance and truncation distance. A common error occurring in records is the reversal of the first and last names. In these cases, reversal edit distance will yield better results. Truncation distance is used when a specific portion of records is sufficient for determining the clusters. All of these distance calculations make our algorithms versatile.

We experimented on four real datasets of total size 1 083 878 records. Two datasets came from the University of Connecticut's



	1	2	4	8	16	32
9M Records	1	1.98	3.79	7.54	14.19	26.29
6M Records	1	1.99	3.84	7.49	14.09	26.38

Figure 8 Speedup (for 6 million and 9 million records; y axis denotes speed up; x axis corresponds to number of processors).

Dental Clinic (UCHC) and two from the Connecticut Department of Development services (DDS).

To generate simulated data, we collected 200 000 records of dead people from ssdmf.info. Each record has SSN, last name, first name, middle name, date of death, and date of birth attributes. Then we introduced 2–3 new characters in the first name or last name for 90% of the records. For the others, we have altered 1–3 characters of the first name or last name. We have thus generated 1 000 000 records. Then we replicated the file three times. We also generated another eight datasets of 1 000 000 records, introducing errors using the above procedure.

CONCLUSIONS

To integrate a huge number of records across multiple datasets, especially from diverse medical and health datasets, our algorithms ensure very fast solutions with high accuracy.

The overall runtime of our algorithms depends on the multiplicity. Even for a multiplicity of 1, our algorithm is faster than TPA (FCED) by a factor of 2. For larger multiplicities, our algorithm achieves impressive speedups over TPA (FCED). For instance, if the multiplicity is 10, then the speedup is more than 100. Runtime and accuracy of our algorithms also depend on the value of l used for blocking. In general, some learning techniques should be applied to figure out a good threshold value. The parallel algorithm achieves a nearly linear speedup.

Contributors AM contributed to implementation, algorithm development, and data analysis. TM contributed to implementation and data analysis. RA contributed data and to data analysis. SR worked on algorithm development and data analysis.

Funding This work was supported in part by the NIH grant R01-LM010101.

Competing interests None.

Provenance and peer review Not commissioned; externally peer reviewed.

Data sharing statement We are ready to release the software for anyone on request.

REFERENCES

1 Clark DE. Practical introduction to record linkage for injury research. *Inj Prev* 2004;10:186–91.
2 Fayyad U, Piatetsky-Shapiro G, Smyth P. From data mining to knowledge discovery in databases. *AI Mag* 1996;17:37–54.
3 Victor TW, Mera R. Record linkage of health care insurance claims. *J Am Med Inform Assoc* 2001;8:281–8.
4 Maizlish N, Herrera L. A record linkage protocol for a diabetes registry at ethnically diverse community health centers. *J Am Med Inform Assoc* 2005;12:331–7.
5 Brin S, Davis J, Garcia-Molina H. Copy detection mechanisms for digital documents. In: *Proceedings of the ACM SIGMOD Annual Conference*: 22–25 May 1995; San Jose CA, Carey MJ, Schneider DA. eds. New York: ACM; 1995:398–409.
6 Shivakumar N, Garcia-Molina H. Building a scalable and accurate copy detection mechanism. In: *Proceedings of the 1st ACM International Conference on Digital Libraries*: 20–23 March 1996; Bethesda, MD. Fox EA, Marchionini G, eds. New York: ACM; 1996:160–8.
7 Mi T, Rajasekaran S, Aseltine R. Efficient algorithms for fast integration on large data sets from multiple sources. *BMC Med Inform Decis Mak* 2012;12:59.
8 Gomatam S, Carter R, Ariet M, *et al*. An empirical comparison of record linkage procedures. *Stat Med* 2002;21:1485–96.
9 Gu L, Baxter R, Vickers D, *et al*. Record linkage: current practice and future directions. *CSIRO Math Info Sci Tech Rep* 2003;3:83.
10 Winkler WE, Thibaudeau Y. An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census. Technical Report Statistical Research Report Series RR91/09, U.S. Bureau of the Census, Washington, DC, 1991.
11 Zhao Y, Karypis G. Evaluation of hierarchical clustering algorithms for document datasets. In: *Proceedings of the 11th international conference on Information and knowledge management: 4–9 November 2002; McLean, VA*. New York: ACM; 2002:515–24.
12 Vinh NX, Epps J, Bailey J. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In: *Proceedings of the 26th International Conference on Machine Learning: 14–18 June 2009; Montreal, Quebec, Canada*. New York: ACM; 2009:1073–80.
13 Heyer LJ, Kruglyak S, Yooseph S. Exploring expression data: identification and analysis of coexpressed genes. *Genome Res* 1999;9:1106–15.
14 Hawse JR, Hejtmancik JF, Huang Q, *et al*. Identification and functional clustering of global gene expression differences between human age-related cataract and clear lenses. *Mol Vis* 2003;9:515–37.
15 Huang DW, Sherman BT, Lempicki RA. Systematic and integrative analysis of large gene lists using DAVID bioinformatics resources. *Nature Protoc* 2009; 4:44–57.
16 Dennis G Jr, Sherman BT, Hosack DA, *et al*. DAVID: Database for Annotation, Visualization, and Integrated Discovery. *Genome Biol* 2003;4:3.
17 Wong W, Liu W, Bennamoun M. Tree-traversing ant algorithm for term clustering based on featureless similarities. *Data Min Knowl Disc* 2007;15:349–81.
18 Ng RT, Han J. Efficient and effective clustering methods for spatial data mining. In: *Proceedings of the 20th International Conference on Very Large Data Bases*: 12–15 September 1994; Santiago de Chile, Chile. Bocca JB, *et al*. eds. San Francisco: Morgan Kaufmann Publishers; 1994:144–55.
19 Clatworthy J, Buick D, Hankins M, *et al*. The use and reporting of cluster analysis in health psychology: A review. *Br J Health Psychol* 2005;10:329–58.
20 Sedgewick R. *Algorithms in C++, third edition*. Addison-Wesley Professional, 1998:424–7.
21 Christen P. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer, 2012.
22 Bhattacharya I, Getoor L. Collective entity resolution in relational data. *ACM TKDD* 2007;1.
23 Winkler WE. Overview of record linkage and current research directions. Technical Report Statistical Research Report Series RRS2006/02, U.S. Bureau of the Census, Washington, DC, 2006.
24 Christen P, Goiser K. Quality and complexity measures for data linkage and deduplication. *Qual Measures Data Mining* 2007;43:127–51.
25 Winkler WE. The state of record linkage and current research problems. Technical Report Statistical Research Report Series RR99/04, U.S. Bureau of the Census, Washington, DC, 1999.
26 Winkler WE. Improved decision rules in the Fellegi-Sunter model of record linkage. *Proceedings of on Survey Research Methods*. American Statistical Association 1993;1:274–9.
27 Fellegi IP, Sunter AB. A theory for record linkage. *J Am Stat Assoc* 1969;64:1183–210.
28 Elmagarmid AK, Ipeirotis PG, Verykios VS. Duplicate record detection: a survey. *IEEE Trans Knowl Data Eng* 2007;19:1–16.
29 Winkler WE. *Matching and record linkage, business survey methods*. New York: Wiley, 1995:355–84.
30 Winkler WE. *The state of record linkage and current research problems*. Statistical Research Division, U.S. Census Bureau, 1999.
31 Pasula H, Marthi B, Milch B, *et al*. Identity uncertainty and citation matching. *Advances in Neural Information Processing Systems* 2003:1401–8.
32 McCallum A, Wellner B. Conditional models of identity uncertainty with application to noun coreference. *Advances in Neural Information Processing Systems* 2005:905–12.
33 Lafferty J, McCallum A, Pereira F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of 18th International Conference on Machine Learning*. 2001:282–9.
34 Culotta A, McCallum A. Joint deduplication of multiple record types in relational data. *14th ACM International Conference on Information and Knowledge Management* 2005:257–8.
35 Parag , Domingos P. Multi-relational record linkage. *Tenth International Conference on Knowledge Discovery and Data Mining*. 2004:31–48.
36 Christen P. A comparison of personal name matching: Techniques and practical issues. *Second International Workshop on Mining Complex Data* 2006:290–4.
37 Levenshtein VI. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Phys, Doklady* 1966; 10:707–10.
38 Jaro MA. Advances in record linkage methodology as applied to matching the 1985 census of Tampa, Florida. *J Am Stat Assoc* 1989;84:414–20.
39 Kukich K. Techniques for automatically correcting words in text. *ACM Comput Surv* 1992;24:377–439.
40 Friedman C, Sideli R. Tolerating spelling errors during patient validation. *Compu Biomed Res* 1992;25:486–509.
41 Christen P, Churches T, Hegland M. Febrl—a parallel open source data linkage system. *Lect Notes Comput Sc* 2004;3056:638–47.
42 Christen P. Febrl—a freely available record linkage system with a graphical user interface. *Second Australasian workshop on Health data and knowledge management* 2008;80:17–25.

43 Lee ML, Ling TW, Low WL. Intelliclean: A knowledge-based intelligent data cleanser. *Sixth International Conference on Knowledge Discovery and Data Mining* 2000:290–4.
44 Hernandez MA, Stolfo SJ. The merge/purge problem for large databases. *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*. 1995:127–38.
45 McCallum A, Nigam K, Ungar LH. Efficient clustering of high-dimensional data sets with application to reference matching. *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*. 2000:169–78.
46 Hassanzadeh O, Chiang F, Lee HC, *et al*. Framework for evaluating clustering algorithms in duplicate detection. *Proc VLDB Endowment* 2009;2:1282–93.
47 Li X. Parallel algorithms for hierarchical clustering and cluster validity. *IEEE Trans Pattern Anal Mach Intell* 1990;12:1088–92.
48 Olson CF. Parallel algorithms for hierarchical clustering. *Parallel Comput* 1995;21:1313–25.
49 Rajasekaran S. Efficient parallel hierarchical clustering algorithms. *IEEE Trans Parallel Distrib Syst* 2005;16:497–502.
50 Wu C-H, Horng S-J, Tsai H-R. Efficient parallel algorithms for hierarchical clustering on arrays with reconfigurable optical buses. *J Parallel Distributed Comput* 2000;60:1137–53.
51 Kawai H, Garcia-Molina H, Benjelloun O, *et al*. P-Swoosh: parallel algorithm for generic entity resolution. Technical Report, Department of Computer Science, Stanford University, 2006.
52 Kim H, Lee D. Parallel Linkage. *ACM CIKM* 2007:283–92.
53 Kirsten T, Kolb L, Hartung M, *et al*. Data partitioning for parallel entity matching. *Proc VLDB Endowment* 2010;3.
54 Bianco GD, Galante R, Heuser CA. A fast approach for parallel deduplication on multicore processors. *Proceedings of the 2011 ACM Symposium on Applied Computing* 2011:1027–32.
55 Mi T, Aseltine R, Rajasekaran S. Data integration on multiple data sets. *Proceedings of the 2008 IEEE International Conference on Bioinformatics and Biomedicine* 2008:443–6.
56 Greiner J. A comparison of parallel algorithms for connected components. *Symposium on Parallel Algorithms and Architectures* 1994:16–25.