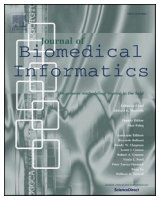
Journal of Biomedical Informatics 130 (2022) 104094

Contents lists available at ScienceDirect

Journal of Biomedical Informatics

journal homepage: www.elsevier.com/locate/yjbin



FIRLA: a Fast Incremental Record Linkage Algorithm☆

Ahmed Soliman, Sanguthevar Rajasekaran

Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269–4155, United States

ARTICLE INFO

Keywords:
Record linkage
Data linkage
Edit distance
Electronic health records
Deterministic linkage

ABSTRACT

Record linkage is an important problem studied widely in many domains including biomedical informatics. A standard version of this problem is to cluster records from several datasets, such that each cluster has records pertinent to just one individual. Typically, datasets are huge in size. Hence, existing record linkage algorithms take a very long time. It is thus essential to develop novel fast algorithms for record linkage. The incremental version of this problem is to link previously clustered records with new records added to the input datasets.

A novel algorithm has been created to efficiently perform standard and incremental record linkage. This algorithm leverages a set of efficient techniques that significantly restrict the number of record pair comparisons and distance computations. Our algorithm shows an average speed-up of 2.4x (up to 4x) for the standard linkage problem as compared to the state-of-the-art, without any drop in linkage performance at all. On average, our algorithm can incrementally link records in just 33% of the time required for linking them from scratch.

Our algorithms achieve comparable or superior linkage performance and outperform the state-of-the-art in terms of linking time in all cases where the number of comparison attributes is greater than two. In practice, more than two comparison attributes are quite common. The proposed algorithm is very efficient and could be used in practice for record linkage applications especially when records are being added over time and linkage output needs to be updated frequently.

1. Introduction

Record linkage refers to the problem of analyzing several input datasets and reporting output clusters of records such that each output cluster constitutes records pertinent to only one entity (e.g., patient). This problem is trivial if a unique primary key exists for each record and all the records are error-free. In practice, medical records usually lack unique patient identifiers (e.g. SSN or a universal patient ID). Also, medical records are not error free. Moreover, state and federal regulations, for example HIPAA in the US and the 2002 Directive on Privacy and Electronic Communications in Europe, severely restrict access to patient identifying information. All of these hurdles, put together, make linking medical records an extremely challenging problem [20].

The importance of record linkage in the medical and healthcare domains has been realized since a long time ago. For instance, in 1900, Alexander Graham Bell linked genealogical records and administrative records from marriages, census results, and other sources to support his familial studies of deafness [1]. In 1929 R. A. Fisher linked public records and family data in the context of human genetics research [2]. Since these works, record linkage has been exploited widely in the biomedical community. We provide some examples next.

In [21], Victor and Mera link records from individual patients and health care providers across time and geography. They employ a combination of exact and probabilistic algorithms. The data used in this research came from insurance claim databases used in health-care related research programs. This dataset had 52 million records representing more than 20 million persons. There are three major components in their algorithm: data standardization, weight estimation and matching. This algorithm has been validated using divergent, convergent, and criterion validity.

Sauleau, Paumier, and Buemi observe that health-care services are making a shift from institution-centered care to consumer centered care [19]. They also assert that unambiguous identification of patients is a critical success factor for health care reform and for the provision of speedy, safe, high quality, comprehensive and efficient health care. Life-critical clinical decisions might depend on positive identification. They have presented an algorithm to detect exact and approximate duplicates within medical identity records [19]. This algorithm helps in attaining a better quality of information and to permit cross-linkage among stand-alone and clustered databases. There are three steps in the algorithm:

☆ C++ source code is available upon request for non-commercial purposes only.

E-mail addresses: ahmed.soliman@uconn.edu (A. Soliman), sanguthevar.rajasekaran@uconn.edu (S. Rajasekaran).

https://doi.org/10.1016/j.jbi.2022.104094
Received 15 November 2021; Received in revised form 2 May 2022; Accepted 4 May 2022
Available online 10 May 2022

The first step is to standardize the data. The second step is to match similar pairs of records. The third step is to generate clusters of coherent related records. The algorithm was run on a dataset of size 300,000. The algorithm identified 240,000 unique clusters.

Padmanabhan, Carty, et al. have described the approach to record linkage used by NHS Digital, a Statutory body in England, and Clinical Practice Research Datalink (CPRD). CPRD provides routine record linkages between primary care data and several health-related datasets within England [16].

One notable application of record linkage in the biomedical domain is that of linking genealogical records with morbidity, mortality, and medical records. Such linking helps researchers identify genetic basis of disease. Furthermore, it enables the understanding of how patients with different genetic predispositions respond to medications [12].

All record linkage algorithms proposed in the literature take time that is quadratic in the total number of records (in all the datasets collectively). Several indexing techniques have been proposed for record linkage to alleviate this high computational complexity. These techniques mitigate the complexity of the matching process by filtering out obviously non-similar pairs [3].

Blocking is another technique that can be used to reduce the run time of record linkage algorithms. The basic idea of blocking is to group the input records into, not necessarily disjoint, blocks. A record might belong to multiple blocks. Then, only candidate records, residing within each individual block, are considered for linkage. Another effective strategy is the use of filtering. Filters are designed to find similar record pairs where similarity is calculated based on a cut-off value under a specific similarity metric. A recent survey on both blocking and filtering techniques could be found in [17].

Two main approaches for record linkage could be found in the literature namely, *deterministic linkage* and *probabilistic linkage* [6]. In deterministic linkage, the grouping of records is based on a set of deterministic rules. The use of statistical theory to make an informed decision for linking is known as probabilistic linkage. In this approach, matching weights are calculated based on probabilities followed by selecting a suitable threshold value. Data *standardization* and *cleansing* are usually done as a preprocessing step to ensure high quality linkage results. The *data matching* phase includes the basic step of comparing record pairs.

A record can be viewed as a collection of attributes. For instance, a medical record may constitute the following attributes: first name, last name, date of birth, height, gender, blood group, address, etc. For comparing record pairs, a suitable subset of attributes, known as comparison attributes, might suffice. Also, an appropriate similarity measure should be carefully selected.

Clusters are widely used for holding related records and keeping track of linkage intermediate and output results. Linkage decisions must be made on whether any cluster pair should be merged. These linkage decisions are based on a distance metric and a threshold value. Single linkage and Complete linkage are two inter-cluster distance metrics widely employed in the literature. For two arbitrary clusters A and B, let the distances vector D denotes the set of distances between all pairs of records a and b where $a \in A$ and $b \in B$. Single (Complete) linkage defines the distance between A and B as $min(D)$ ($max(D)$). Edit distance is commonly used for measuring the similarity between record pairs. Edit distance is defined as the cost of the minimum number of string operations required to transform one string into another. So, each comparison attribute of a record is treated as a string and the distance between two records is then defined as the sum of the edit distances between each pair of comparison attributes.

We live in a period of time when voluminous data get generated in every walk of life (not restricted to science and engineering alone). Most of these data could become extremely useful if linked properly. For example, linked health records could aid medical and social researchers in their studies [5]. This is because *record linkage* enhances any deduced statistics produced from data records. It leads to more accurate and more significant statistical information [7].

A *duplicate* is a record that refers to the same entity already referenced by another record in the dataset. The reason duplicates exist is mainly because data usually come from various sources. Detecting and removing/merging such duplicate records, also known as *de-duplication*, is an important step in the record linkage process. The relative frequency of duplicate records is known as the *rate of duplicates*. In real datasets, it has been reported that the rates of duplicates can reach 20 percent [22]. As the data sources grow, this rate is expected to keep increasing over the years. In our approach we leverage such a high rate of duplicates in a way that improves the speed of the linkage process.

Challenges and Innovations: Records to be linked may not have unique patient identifiers. There could be several sources of errors. Also, regulations such as HIPAA restrict access to patient identifying information. All of these issues make the problem of record linkage extremely challenging [20]. Also, the problem of incremental record linkage is even more challenging as evidenced by the long run times of the best known algorithms. In this paper we create record linkage algorithms that outperform prior algorithms. Some of the innovations we introduce are: 1) Our algorithms are inherently incremental; 2) We keep a set of representatives for each cluster to speedup the run times; 3) We use multimodal attributes for linking; and 4) Many record comparisons are avoided with a pruning technique that only uses the lengths of the records.

2. Methods

2.1. Previous methods

Different record linkage solutions have been built and made available by several researchers. Examples include Reclink, AtyImo, Fril, and CIDACS-RL. Reclink is a probabilistic database linkage system written in C++ [4]. AtyImo is a hybrid probabilistic linkage software [18]. It is designed particularly for linking massive datasets with a high accuracy. An accuracy of 93%-97% in true matches have been reported as a result of applying AtyImo for linking 114 million individuals in less than nine days using Spark [18]. Fril is another integration and linkage Java-based tool. It supports different distance metrics, search, and analysis tools [10].

Since the amount of data is usually very big in most practical applications, several techniques have been developed to make linking more scalable. Karapiperis, et al. have proposed parallel and distributed techniques to deal with the scalability issue. They have introduced *LSHDB*, their parallel data engine, which is based on locality sensitive hashing [11]. Another way of improving the scalability of the record linkage processes is through leveraging some heterogeneous architectures (for example, a mix of CPUs and GPUs) [18].

Mi et al. [15] have introduced a hierarchical clustering based solution for record linkage. In their work, four techniques have been applied to improve both the runtime and space requirement of the linking algorithms. One prominent technique is based on the Faster Computation of the Edit Distance (FCED). FCED employs upper bounds on edit distance to predict the distance given a specific threshold. A Two-Phase record linkage Algorithm (called TPA(FCED)) that employs FCED can efficiently integrate large datasets from multiple sources.

Mamun et al. [14] have introduced efficient sequential and parallel algorithms for record linkage. These algorithms are based on hierarchical clustering and used radix sorting on selected attributes to detect and eliminate identical records. Also, a graph has been created that represents the links of similar records. This graph is subsequently analyzed to find the connected components. Each connected component represents a set of records belonging to a unique entity.

2.2. Formal definition

We start by giving a formal definition for the record linkage problem

in regards to the biomedical domain. Let $D = [D_1, D_2, \ldots, D_m]$ be a set of m datasets. Each dataset D_i constitutes a set of records $D_i = [r_1, r_2, \ldots, r_{|D_i|}]$. Each record r_i defines a real-world entity, a particular patient in a medical record linkage setting. Each record constitutes a set of attributes $r_i = [a_1, a_2, \ldots, a_n]$. Example attributes are: Social Security Number (SSN), Patient Identification Number, First Name, Last Name, Date of Birth, and Date of Death, …etc. The record linkage problem is the problem of identifying a set of clusters (group of records) $C = [c_1, c_2, \ldots, c_p]$ such that, for each cluster c_j all member records $r \in c_j$ belong to one and only one patient. Two records will be placed in the same cluster if they are very similar (as measured by the distance between the two records).

2.3. Our approaches

In this paper we focus on deterministic record linkage. In other words, we focus on *exact matching* rather than *statistical matching*. The most important basic operation in the linkage process is comparing a pair of records. The brute-force algorithm is to compare each and every pair of records. Obviously, this is impractical since there is a quadratic growth in the number of comparisons as the total size of the datasets increases. One very well known technique to restrict the number of comparisons is the blocking technique. A record always consists of several attributes. One field is designated as a blocking field. Based on the value of this field a set of blocks are constructed. A record is then associated with a set of blocks, corresponding to all the l-mers (substrings of size l) in it. Only pairs of records sharing one or more blocks need to be compared. This dramatically reduces the number of record comparisons. Hence, the execution time decreases and the linkage process becomes more practical.

In this article we introduce several useful techniques which improve the execution speed further. In the following subsections we will present these techniques.

2.3.1. Inherently incremental linking

The record linkage algorithms found in literature usually iterate over the blocks in their outermost loop. For each block, all pairs of records within the block are then compared. This has been the general framework for most of these linking algorithms. In real applications it is usually the case that new records are being generated and saved to record sets over time. We call such new records *incremental records*. To consider the incremental records in the linking process (i.e., update the linkage output as a result of appending new records to the datasets) the usually adopted block-by-block scanning must be restarted from scratch. Of course, this is inefficient. To better handle incremental records, we adopt a new linkage framework. We call this new framework *Inherently Incremental Linking*. Next, we describe this proposed framework in detail.

In each outermost iteration of our algorithm, we pick a single incremental record r. The blocks B associated with r are identified. We compile a list of candidate clusters C' sharing one or more blocks from B. Then we compile a list of candidate records R' which is the union of all unique representative records in C'. Then based on the edit distances $D = ED(r, r')$ where $r' \in R'$, record r will be linked according to one of two scenarios.

Scenario 1: There exists at least one distance in D less than the threshold value θ. This means record r should be linked with one or more related clusters. In this case, the related clusters are merged into a new cluster first. Next, record r is added to that newly created cluster and becomes available for comparison in subsequent iterations.

Scenario 2: None of the candidate clusters are related to record r. In this case, record r will be added alone in a new singleton cluster. Again, this new cluster will become available for comparison in subsequent iterations.

In summary, at any given time during linkage, there are two partitions of records. The first partition constitutes the records linked so far. And the second partition is the set of incremental records awaiting linkage. Another feature of our algorithm is the ability to save and load clustering information for the previously linked datasets. This is especially useful for incremental linkage applications where the linkage time could be drastically reduced after loading this clustering information. The linkage time is reduced because there is no need for pair-wise comparisons among any of the previously linked records. Only incremental records need to be compared against the linked records. Please note that deletion is not supported yet in our algorithm. It is part of our future work.

Fig. 1 illustrates the steps of our incremental linkage process using a simple working example.

2.3.2. Comparison skipping technique

In our work, record pairs comparisons are based on edit distances (see e.g., [9]). Two records R_i and R_j are linked if and only if the edit distance between them $ED(R_i, R_j)$ is less than or equal to a threshold value θ. Let $R_i.f$ ($R_j.f$) denotes the comparison attributes of record R_i (R_j). The edit distance is computed as:

$$ED(R_i, R_j) = \sum_{k=1}^{K} ED(R_i.f_k, R_j.f_k)$$

Where K is the total number of comparison attributes. We present a novel idea that significantly reduces the number of distance computations needed during record linkage. The idea is an extension of the observation stated in [15] namely, the sizes of the comparison attributes are compared first. The idea is as follows: before calling the function that computes $ED(R_i, R_j)$, a much cheaper string length comparisons are done first. Denote by $strlen(R_i.f_k)$ the string length of the field $R_i.f_k$. If, for any k, $|strlen(R_i.f_k) - strlen(R_j.f_k)| > \theta$ or $\sum_{k=1}^{K} |strlen(R_i.f_k) - strlen(R_j.f_k)| > \theta$ then comparison is immediately skipped. Thus, R_i and R_j are considered unrelated.

For efficient implementation of this technique, sizes of all comparison attributes are evaluated just once at the beginning of our algorithm and stored in a vector. The exact skipping fraction (i.e., the fraction of skipped record pair comparisons) depends on the distribution of string lengths of the records being linked. In our experiments, on an average, the skipping fraction (due to employing the distance skipping technique) was about 35% of the total number of pair comparisons being considered (for $\theta = 1$).

2.3.3. Representative records

Another improvement is to keep track of what we call *a set of representative records* for each cluster. The idea is to efficiently deal with duplicate records. Instead of comparing against each record in a cluster, comparison involves only this set of records. The set is formed as follows: Whenever a new record enters a cluster, if its distance to any record in the cluster is zero (i.e., this new record is a duplicate) then this record is not considered a representative record. Otherwise, the new record carries added information. Thus, it is added to the set of representative records.

2.3.4. Treating dates as integers

To the best of our knowledge, date attributes (e.g., birth dates) were treated as strings of characters in all previous works. One of our contributions is to parse these date attributes as integers. As shown in Section 3, this led to further speedup in the linkage process. In the preparation phase of our algorithm, all date attributes are optionally parsed into *boost::gregorian::date* objects, one of the *Boost C++ Libraries* (https://www.boost.org/), which consumes only 4 bytes of memory per date field. These date objects internally use adjusted *Julian day* count which is exactly what we need to compare records. Compare this with one of the widely used date formats, e.g. *mmddyyyy*, which consumes 8 characters. Of course, there is a trade-off in using string versus integer representation of date attributes. String representations have the

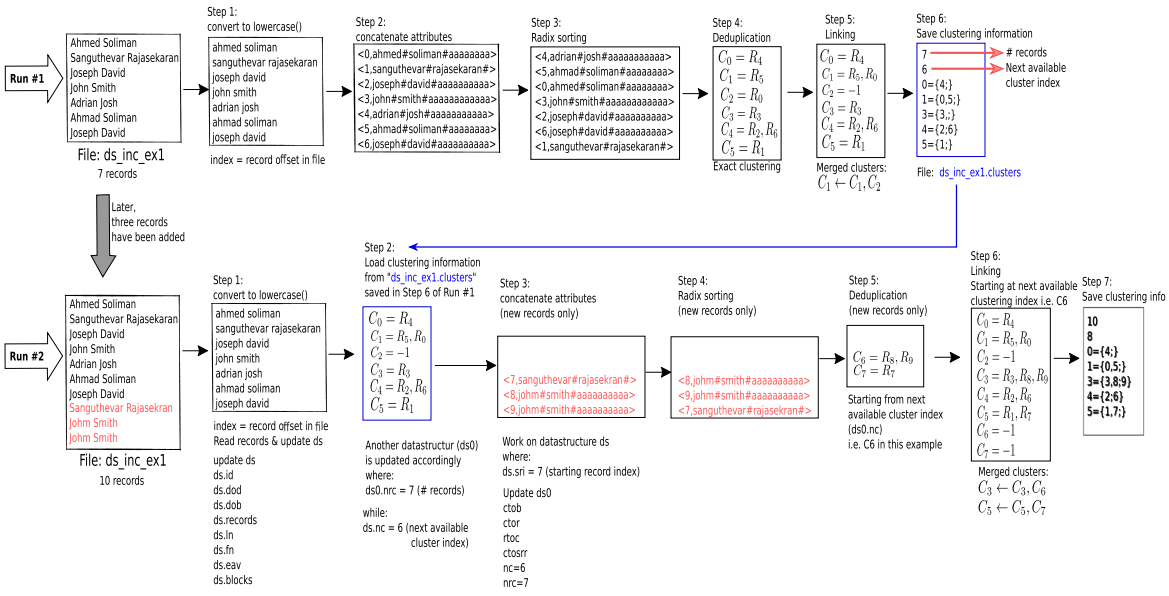


Fig. 1. Incremental Record Linkage using "FIRLA" (A working example).

advantage of detecting errors in date entries and might link more records. However, assuming extremely low data entry error rates associated with date entries, our proposed representation leads to a similar linkage performance. Note that medical institutions usually verify the birth date several times with the patient at each visit, thus reinforcing our assumption.

2.4. Our algorithm

Algorithm 1. Fast Incremental Record Linkage Algorithm (**FIRLA**)

Input: Records dataset DS and distance threshold θ.
Output: Cluster of records as a result of single-linkage based on the given input threshold θ.

1: Concatenate the comparison attributes for each record $R \in DS$;
2: Radix-sort the list of concatenated strings;
3: Deduplicate the sorted list;
4: Extract and store exactly matched records into clusters.
5: Store separate records into singleton clusters.
6: **for** each cluster C **do**
7: isClustered ← False;
8: **for** each representative record $R \in C$ **do**
9: **for** each block $B \in R$ **do**
10: **for** each candidate cluster C' sharing the same block B with cluster C **do**
11: **for** each candidate record $R' \in C'$ **do**
12: **if** *canSkipComparison*(R, R') **then**
13: Continue;
14: **end if**
15: **if** *Distance*$(R, R') > \theta$ **then**
16: Continue;
17: **else**
18: $C' \leftarrow C' + C$; ▷merge clusters;
19: isClustered ← True;
20: break;
21: **end if**
22: **end for**
23: **end for**
24: **end for**
25: **end for**
26: **if** isClustered == False **then**
27: Create a new cluster for record R
28: **end if**
29: **end for**

In this paper we present a novel record linkage algorithm, called **FIRLA**[1]. The pseudo-code for the algorithm is presented in Algorithm 1. FIRLA invokes all the novel techniques described above and the ones presented below. Please note that the function *canSkipComparison*(R, R'), in line 12, implements the comparison skipping and signature-based pruning techniques described in Sections 2.3.2 and 2.5, respectively.

2.5. Signature-based pruning

This section details our *signature-based pruning* technique. This technique is one of our crucial contributions in this paper. This technique estimates edit distances between two strings without computing it. As shown in Section 3, this idea alone has led to an extremely fast linkage. Edit distance computation is the most critical and time-consuming step in the linkage process. So, every attempt to skip edit distance computation will result in a speedup. Of course, speedup is guaranteed provided that the extra code, which implements the pruning technique, does not take more than the edit distance computation time alone. Although this technique is presented and tested in the context of English alphabet, it is applicable to other languages. The technique has two steps, namely: signature preparation and distance estimation. Next, we describe these steps in detail.

2.5.1. Signature preparation

In this step, each comparison attribute is mapped into a bit-vector. We call this vector the *signature* of the field. The signatures are formally defined as follows: Let F_j^i denote the j^{th} comparison field/attribute of record i. First, all characters are converted into lowercase. In the context of English language, F_j^i is a string defined over Σ, where $\Sigma = a,b,c,\ldots,z$. Since $|\Sigma| = 26$, signature S_j^i is a bit-vector of size 26 (i.e., $S_j^i = [b_{25}b_{24}\ldots b_0]$) where, the alphabet characters $a, b, \ldots, z$ are mapped into bits $b_0, b_1, \ldots b_{25}$, respectively. First, S_j^i is initialized to all zeros. Then, F_j^i is mapped into S_j^i as follows: the characters of F_j^i are scanned and the corresponding bits in the signature are set to 1. For example, if $F_j^i =$ "*Joseph*" Then,

[1] C++ source code is available upon request for non-commercial purposes only

$S_j^i = 00000001001100001010010000$

Note that only bits: $b_9, b_{14}, b_{18}, b_4, b_{15}$, and b_7 are set to 1. These bits correspond to the characters: j, o, s, e, p, and h, respectively.

2.5.2. Distance estimation

Now, record pairs are compared as follows: If comparison cannot be skipped (see Section 2.3.2) then we apply the logical XOR operator to compute the Hamming distance H between the respective signatures. Next, the number of ones in the result $Ones(H)$ is efficiently estimated. If $Ones(H) > 2\theta$ then the distance computation could be skipped. Otherwise, the edit distance is computed. Otherwise, the edit distance computation is done. The validity of this technique is established in the following Lemma.

Lemma 1. *Let F_j^a and F_j^b denote the j^{th} comparison attributes belonging to two records R_a and R_b, respectively. Let S_j^a and S_j^b denote the respective signatures for these comparing attributes. Let the Hamming distance between both the signatures be denoted by H_j, where $H_j = S_j^a \oplus S_j^b$. Assume that we use Edit-distance-based record linkage, where the edit distance threshold is θ. R_a and R_b can be determined to be unrelated (without calculating the edit distance) if there exists at least one field j such that, $Ones(H_j) > 2.\theta$, where $Ones(H_j)$ is the number of ones in the bit-vector H_j.*

Proof. Let S_j^a be the set of characters in F_j^a that are not in F_j^b and let S_j^b be the set of characters in F_j^b that are not in F_j^a. If $Ones(H_j) > 2\theta$, it means that $|S_j^a| + |S_j^b|$ is $> 2\theta$. In this case it is easy to see that the edit distance between F_j^a and F_j^b has to be greater than θ. The edit distance introduced by the characters in S_j^a and S_j^b is minimized when $|S_j^a|$ is nearly the same as $|S_j^b|$. Even in this case, the edit distance between F_j^a and F_j^b is $> \theta$ (assuming that the edit distance introduced by the other characters is zero). □

3. Results

Experimental data: We have used real people data to evaluate our algorithms. The original datasets have been downloaded from *Social Security Death Master File* courtesy of *SSDMF.INFO* [2]. Each record consists of the following attributes: social security number, last name, first name, date of birth, and date of death. A modified version of the FEBRL dataset generator program [3] has been used to introduce errors in the dataset. Note that the datasets used in this study are the same datasets used in evaluating state-of-the-art linkage algorithms [13]. All datasets used in this study could be downloaded from [23].

Datasets generation: The number of records belonging to the same entity in a dataset is known as "Multiplicity." To generate a dataset of size N and multiplicity m, we arbitrarily pick N/m records from the original datasets. Then, each record is duplicated m times. Next, for each entity we introduce errors by corrupting one out of its m records. The way data is corrupted is described in the following paragraph in detail. Now, for each entity we have $m-1$ original records that remained intact plus one corrupted record. All records in the generated datasets are then shuffled. Finally, we save the generated datasets into files which are used later as input for our experiments.

Corrupting data: We have used an unbiased coin to generate the probabilities of introducing errors. This is a realistic model since it is less likely to find more than one typo in a single record. For 90% of the records, we insert two (three) new characters into first name or last name fields with probability of 90% (10%). For the remaining 10% of the records, we alter one, two, or three characters of first name or last name fields equally with probabilities of 80%, 10%, and 10% respectively.

Scalability: 11 dataset sizes, namely: 50K, 100K, 200K, 400K, 600K, 800K, 1M, 2M, 3M, 4M, and 5M records have been employed in our experiments. We have used a multiplicity of 5. Please note that multiplicity is defined before introducing errors (i.e., a corrupted record is counted in) as it still belongs to the same original entity.

Experimental Setup: All the experiments were done on a Dell OptiPlex-9020 PC with 4 Intel i7-4770CPU @3.40 GHz cores and 24 GB of RAM. The operating system running was Ubuntu 18.04.4 LTS. The programs have been written in standard C++.

Evaluation Metrics: We use the following metrics to assess and compare performance of linking algorithms:

1. **Execution time:**
   Time was measured by taking the CPU clock time which gives the instruction level elapsed time a program takes.
2. **Linkage Performance:**
   We start evaluating linkage performance by computing the confusion matrix shown in Table 1. Then the following four metrics are calculated.
   (a) **Linkage Precision, P:** The ratio of true matches to the total predicted matches.
   $$P = d/(d+b)$$
   (b) **Linkage Recall, R:** The ratio of successfully predicted, recalled, matches to the total number of true matching record pairs.
   $$R = d/(d+c)$$
   (c) **Linkage Accuracy, A:** This is defined as the proportion of correctly identified links.
   $$A = (a+d)/(a+d+b+c)$$
   (d) **F1 Score, F_1:** This is calculated from the precision and recall as the harmonic mean defined by,
   $$F_1 = 2 \times \frac{PR}{P+R}$$

As pointed out in [8], for large databases, linkage performance evaluations which are solely based on the linkage accuracy are discouraged since a remarkably high accuracy can be easily obtained with sloppy algorithms. However, we have included the linkage accuracy in our evaluations for ease of comparison with previous work.

3.1. Standard linkage results

In our experiments, record linkage is performed for each dataset using three different single linkage algorithms: The algorithm by Mamun, et al. [14] ($SL1$), our first proposed algorithm ($SL2$) which implements all of our contributions discussed in this paper except for treating dates as integers, and our second proposed algorithm ($SL3$) that treats dates as integers in addition. Fig. 2 and Table 2 show linking times in seconds when three comparing attributes were used, namely: first

Table 1
Confusion matrix.

Predicted link status	True link status	
	Match	Non-match
Match	d (true match)	b (false match)
Non-match	c (false non-match)	a (true non-match)

[2] http://ssdmf.info/download.html
[3] http://sourceforge.net/projects/febrl/

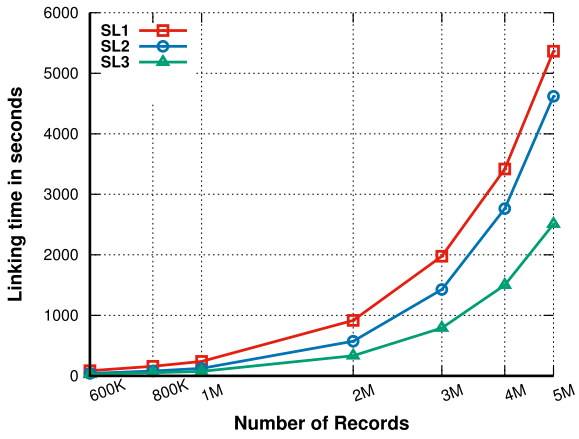


Fig. 2. Linkage time in seconds when using three comparison attributes: first name, last name, and date of birth. **SL1** is the algorithm by Mamun, et al. [14], **SL2** is the first variation of our proposed algorithm (FIRLA) which implements all our contributions discussed in this paper except for treating dates as integers, and **SL3** is the full-featured version of our proposed algorithm (FIRLA) that treats dates as integers in addition.

name, last name, and date of birth.

Another experiment has been conducted for linking records using four comparison attributes namely, first name, last name, date of birth, and date of death. The linkage time is displayed in Fig. 3.

The linkage performance in all standard linkage experiments is summarized in Table 3 using F_1 score. For more details using the other performance metrics please refer to the tables in the supplementary file.

3.2. Incremental linkage results

The following experiment shows how our algorithm performs incremental linkage. The experiment uses a total of $2M$ records. We simulate records being added over time. First, the input file contains the first $1M$ records to start with. The program is run to link those $1M$ records. Then, we consider the addition of $100K$ records. Now, the job is to link all $1.1M$ records. (i.e., link the new $100K$ records with previously linked records). Table 4 shows the run times for incrementally linking $100K$ to $1M$ records over a base dataset of $1M$ records.

Note that the last column corresponds to linking from scratch. For example, standard linking of the entire 2 M records from scratch takes 391 s (about 6 and a half minutes). Using "FIRLA" cuts this linkage time down to 187 s (about 3 min). On an average, our algorithm incrementally links in about 33% of the time required for linking from scratch. Note that this speedup does not affect any of the performance metrics. The linkage output is identical for incremental and one-shot linking.

4. Discussion

Our proposed algorithm "FIRLA" outperforms the state-of-the-art algorithms [14,13] in all cases where the number of comparison attributes is greater than two. Even when the number of comparison attributes equals two, just one dataset took more linkage time, namely *ds*11 of size five million records. Of course, record linkage time is data dependent. Datasets with different distributions of multiplicity factors will be linked in different times.

Also, multiplicity m affects linkage time. We have used $m = 5$ in our experiments because we expect at least $m = 5$ is common in real applications. As multiplicity increases, more speed up is expected from our algorithms mostly because we only compare against representative records from each cluster. Even for $m = 1$ (i.e., each entity is represented by one and only one record in the datasets collectively) our algorithm is still faster. In the latter case, the speedup of our algorithm is due to the combination of the other techniques that we have introduced and implemented therein such as comparison skipping and signature-based pruning.

5. Conclusions

In this paper we have presented several techniques that can be used to speedup record linkage algorithms. We have created algorithms that exploit our speedup techniques. Experimental results reveal that our algorithms outperform existing algorithms. As noted from the experimental results, performance of our algorithms improves as the multiplicity factor increases. This is the case for most datasets in real applications. For instance, each individual can have on an average 5 to 10 medical records distributed among different health-care institutional databases. Hence, we believe that our algorithms will perform very well in real applications.

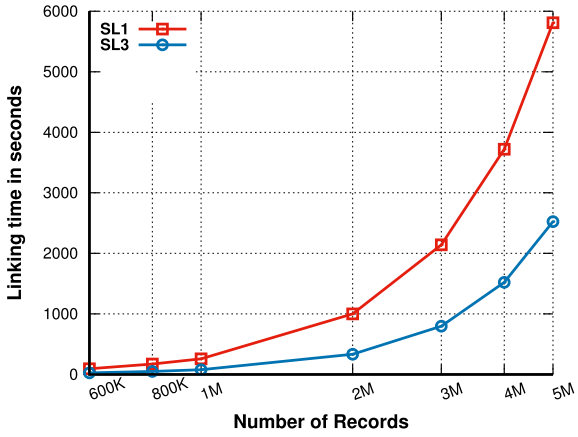


Fig. 3. Linkage time in seconds for four comparison attributes: first name, last name, date of birth, and date of death. **SL1** is the algorithm by Mamun, et al. [14] and **SL3** is our algorithm (FIRLA) which implements all contributions presented in this paper.

Table 2
Linkage Time in seconds.

#CA	DS / #Rec / Algo	ds01 50 K	ds02 100 K	ds03 200 K	ds04 400 K	ds05 600 K	ds06 800 K	ds07 1 M	ds08 2 M	ds09 3 M	ds10 4 M	ds11 5 M
2	**SL2**	0.0	0.0	4.0	15.5	34.0	61.0	93.0	375.0	850.5	1573.0	2711.0
	SL1	1.0	3.0	9.0	31.0	62.0	106.5	156.0	525.5	1025.5	1655.5	2486.5
3	**SL3**	0.0	1.0	3.0	12.0	27.0	51.0	76.0	333.0	793.0	1500.0	2510.0
	SL2	0.0	1.0	4.0	19.5	42.0	81.5	125.0	573.5	1428.0	2763.0	4620.0
	SL1	1.0	3.0	12.0	43.0	87.0	159.0	238.5	917.5	1977.5	3418.0	5364.0
4	**SL3**	0.0	1.0	3.0	12.3	26.3	50.3	78.3	333.0	797.3	1521.0	2523.3
	SL1	1.0	3.0	13.0	46.0	94.0	171.0	258.0	1001.0	2142.0	3720.0	5811.0

Table 3
F1 Score.

#CA	DS #Rec Algo	ds01 50 K	ds02 100 K	ds03 200 K	ds04 400 K	ds05 600 K	ds06 800 K	ds07 1 M	ds08 2 M	ds09 3 M	ds10 4 M	ds11 5 M
2	**SL1**	0.93	0.88	0.81	0.72	0.66	0.55	0.46	0.10	0.03	0.01	0.01
	SL2	0.93	0.89	0.82	0.74	0.69	0.62	0.58	0.40	0.29	0.22	0.18
3	**SL1**	0.97	0.97	0.97	0.97	0.97	0.96	0.96	0.95	0.94	0.92	0.90
	SL2	0.97	0.97	0.97	0.97	0.97	0.96	0.96	0.96	0.95	0.94	0.94
	SL3	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97
4	**SL1**	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97
	SL3	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97

Table 4
Incremental Linkage Time (in seconds).

Starting size	Incremental size	Incremental linkage time	Total linkage time
1 M	100 K	28	213
1 M	200 K	51	255
1 M	400 K	97	301
1 M	600 K	135	339
1 M	800 K	163	367
1 M	1 M	187	391

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This research has been supported in part by the National Science Foundation (NSF) Grants 1743418 and 1843025. This work was partially supported by the United States Census Bureau under Award Number CB21RMD0160003. The content is solely the responsibility of the authors and does not necessarily represent the official views of the US Census Bureau.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at https://doi.org/10.1016/j.jbi.2022.104094.

References

[1] AG Bell. The deaf. In: US Department of Commerce and Labor, Bureau of the Census. Special Reports: The blind and the deaf, 1900.
[2] Joan Fisher Box, R.A. Fisher, the Life of a Scientist, Wiley, New York, 1978.
[3] Peter Christen, A survey of indexing techniques for scalable record linkage and deduplication, IEEE Trans. Knowl. Data Eng. 24 (9) (2012) 1537–1555.
[4] Kenneth R. de Camargo Jr. and Cláudia M. Coeli. Reclink: aplicativo para o relacionamento de bases de dados, implementando o método probabilistic record linkage. Cadernos de Saúde Pública, 16(2), 439–447, jun 2000.
[5] Rinku Dewri, Toan Ong, R. Thurimella, Linking health records for federated query processing, Proc. Privacy Enhan. Technol. 2016 (2016) 23–24.
[6] Harron K. Doidge JC. Demystifying probabilistic linkage: Common myths and misconceptions. Int J Popul Data Sci., 3, 2018.
[7] Halbert L. Dunn. Record Linkage. American Journal of Public Health and the Nations Health, 36(12), 1412–1416, December 1946. Publisher: American Public Health Association.
[8] David Hand, Peter Christen, A note on using the F-measure for evaluating record linkage algorithms, Stat. Comput. 28 (3) (2018) 539–547.
[9] E. Horowitz, S. Sahni, S. Rajasekaran, Computer algorithms, Silicon Press (1998).
[10] Pawel Jurczyk, James J. Lu, Li Xiong, Janet D. Cragan, Adolfo Correa, Fine-grained record integration and linkage tool, Birth Defects Res. Part A: Clin. Mol. Teratol. 82 (11) (2008) 822–829.
[11] D. Karapiperis, A. Gkoulalas-Divanis, V. Verykios, Lshdb: a parallel and distributed engine for record linkage and similarity search, in: 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), 2016, pp. 1–4.
[12] Daijin Kim, Steven Labkoff, Samuel H Holliday, Opportunities for electronic health record data to support business functions in the pharmaceutical industry—a case study from pfizer, inc, J. Am. Med. Inform. Assoc. 15 (5) (2008) 581–584.
[13] Abdullah-Al Mamun, Robert Aseltine, Sanguthevar Rajasekaran, Efficient record linkage algorithms using complete linkage clustering, PLOS ONE 11 (4) (2016) 1–21, 04.
[14] Abdullah-Al Mamun, Tian Mi, Robert Aseltine, and Sanguthevar Rajasekaran. Efficient sequential and parallel algorithms for record linkage. Journal of the American Medical Informatics Association, 21(2), 252–262, sep 2014.
[15] Tian Mi, Sanguthevar Rajasekaran, Robert Aseltine, Efficient algorithms for fast integration on large data sets from multiple sources, BMC Med. Inform. Decis. Mak. 12 (1) (2012).
[16] Shivani Padmanabhan, Lucy Carty, Ellen Cameron, Rebecca E. Ghosh, Rachael Williams, Helen Strongman, Approach to record linkage of primary care data from clinical practice research datalink to other health-related patient data: overview and implications, Eur. J. Epidemiol. 34 (2019) 91–99.
[17] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey. ACM Comput. Surv., 53(2), March 2020.
[18] R. Pita, C. Pinto, S. Sena, R. Fiaccone, L. Amorim, S. Reis, M.L. Barreto, S. Denaxas, M.E. Barreto, On the accuracy and scalability of probabilistic data linkage over the brazilian 114 million cohort, IEEE Journal of Biomedical and Health Informatics 22 (2) (2018) 346–353.
[19] Erik A. Sauleau, Jean Philippe Paumier, Antoine Buemi, Medical record linkage in health information systems by approximate string matching and clustering, BMC Med. Inform. Decis. Mak. 5 (2005) 1–13.
[20] Gulzar H. Shah, Kaveepan Lertwachara, Anteneh Ayanso, Record Linkage in Healthcare, Int. J. Healthcare Deliv. Reform Initiat. 2 (3) (2011) 29–47.
[21] Timothy W. Victor, Robertino M. Mera, Record linkage of healthcare insurance claims, Stud. Health Technol. Inform. 84 (3) (2001) 1409–1413.
[22] William E. Winkler. Matching and Record Linkage. In Brenda G. Cox, David A. Binder, B. Nanjamma Chinnappa, Anders Christianson, Michael J. Colledge, and Phillip S. Kott, editors, Wiley Series in Probability and Statistics, pages 353–384. John Wiley & Sons Inc, Hoboken, NJ, USA, October 2011.
[23] A. Soliman, Record Linkage Datasets (2022), https://doi.org/10.6084/m9.figshare.19500671.v1.