Robust temporal graph clustering for group record linkage

Charini Nanayakkara, Peter Christen, and Thilina Ranbaduge

Research School of Computer Science, The Australian National University, Canberra, ACT 2601, Australia. charini.nanayakkara@anu.edu.au

Abstract. Research in the social sciences is increasingly based on large and complex data collections, where individual data sets from different domains need to be linked to allow advanced analytics. A popular type of data used in such a context are historical registries containing birth, death, and marriage certificates. Individually, such data sets however limit the types of studies that can be conducted. Specifically, it is impossible to track individuals, families, or households over time. Once such data sets are linked and family trees are available it is possible to, for example, investigate how education, health, mobility, and employment influence the lives of people over two or even more generations. The linkage of historical records is challenging because of data quality issues and because often there are no ground truth data available. Unsupervised techniques need to be employed, which generally are based on similarity graphs generated by comparing individual records. In this paper we present a novel temporal clustering approach aimed at linking records of the same group (such as all births by the same mother) where temporal constraints (such as intervals between births) need to be enforced. We combine a connected component approach with an iterative merging step which considers temporal constraints to obtain accurate clustering results. Experiments on a real Scottish data set show the superiority of our approach over a previous clustering approach for record linkage.

Keywords: Entity resolution, star clustering, vital records, birth bundling.

1 Introduction

Databases that contain personal information, such as censuses or historical civil registries [18], generally contain records that describe a group of individuals, where each individual can occur with different types of roles [6]. A *baby* is born, then recorded as a *daughter* or *son* in a census, and later she or he might marry (as a *bride* or *groom*) and become the *mother* or *father* of her or his own children. Being able to link such records across different databases will allow the reconstruction of whole populations and open a multitude of studies in the health and social sciences that currently are not feasible on individual databases [4]. Studying these issues is important to identify how societies evolve over time and discover the changes that influenced and contributed for social evolution [14].

The process of identifying the sets of records that correspond to the same individual is known as *record linkage*, *entity resolution*, or *data matching* [5]. Record linkage involves comparing pairs of records to decide if a pair refers to the same entity (known as a *match*) or to different entities (a *non-match*). In this process, generally, the similarities between the values in selected attributes are compared to decide if two records are similar enough to be classified as a match (if for example all similarities are above a given threshold) [5]. For certain applications such a simple pair-wise linkage does however not provide enough information to identify matching records with high accuracy [6].

In contrast to traditional pair-wise record linkage, *group linkage* [17] has recently received significant attention because of its applicability of linking groups of individuals, such as families or households [6,11]. The identification of relationships between individuals can enrich and improve the quality of data, and thus facilitate more sophisticated analysis of different socio-economic factors (such as health, wealth, occupation, and social structure) of large populations [4].

Historical record linkage [19] involves the linkage of records from historical censuses, as well as birth, death, and marriage certificates, to construct longitudinal data sets about a population. This problem has been studied in the past two decades by researchers working in different domains. In 1996 Dillon investigated an approach to link census records from the US and Canada to generate a longitudinal database to examine changes in household structures [9]. IPUMS is a large project which aims to curate and ultimately link large demographic data collections [19]. The Life-M project is another example of transforming historical records into a multi-generational longitudinal database [3]. The Digitising Scotland project [8], which this work is part of, aims to link civil registration events recorded in Scotland between 1856 and 1973 to create a linked database covering the whole population of Scotland spanning more than a century to allow researchers to conduct studies that currently are not possible.

Here we address one specific step used in historical record linkage as conducted by demographers and historians [18]: the *bundling* (clustering) of birth records by the same mother to identify siblings. Once sibling groups have been identified, they can be linked to census, marriage, and death records using group linkage techniques [6,11]. Linked bundles of siblings allow a variety of studies, for example, about fertility and mortality and how these change over time [18].

Contributions: In this paper we investigate how clustering techniques for record linkage [13,20] can be employed for grouping records where temporal constraints exist between record pairs. We propose and evaluate a novel temporal clustering approach which first creates temporally possible connected components with high precision using only links of high similarities [21], and then employs an iterative refinement step that merges those connected components that are highly similar and temporally possible. We conduct an empirical study on a real historical data set which has been extensively linked semi-manually by domain experts [18] providing us with ground truth data to calculate linkage quality. We show that our temporal clustering approach can outperform a state-of-the-art clustering technique for record linkage in terms of linkage quality.

2 Related Work

Classification techniques for record linkage can be categorised into supervised and unsupervised methods. Unsupervised clustering techniques view record linkage as the problem of how to identify all records that refer to the same entity and to group these records into the same cluster. Hassanzadeh et al. [13] presented a framework to comparatively evaluate different clustering techniques for record linkage. Saeedi et al. [20] proposed a framework to perform clustering for record linkage on a parallel platform using Apache Flink. In their evaluation, star clustering [2] was one of the best performing techniques compared to other clustering methods. In star clustering, records that have high similarities with other records are selected as the centres of possibly overlapping clusters, where the overlapping clusters then need to be split in an iterative second step.

Saeedi et al. [21] recently proposed a novel clustering algorithm based on the strengths of links between records as categorised into strong, normal, and weak (as we discuss in the next section). Connected components are formed based only on strong links initially, which are then refined by adding normal links.

Neither of these clustering approaches, however, has considered temporal constraints. In our recent work [16] we have considered temporal aspects as an improvement to star clustering. While this improved star clustering algorithm was able to achieve better results compared to a greedy temporal clustering approach, it still resulted in low linkage quality due to the requirement of splitting overlapping clusters. Our aim is to improve linkage quality using a novel temporal clustering method which employs the concepts of link strength [21], and integrates them with temporal constraints and an iterative cluster merging step.

3 Overview of Temporal Graph Clustering

Our methodology to conduct temporal graph clustering for group linkage consists of three major phases. In this section, we first describe how we model temporal constraints, and then detail how we generate the initial similarity graph. In Sect. 4 we then propose a connected component generation phase, and in Sect. 5 an iterative refinement phase which merges similar temporally consistent connected components. For notation we use bold letters for clusters, lists, and sets (with upper-case bold letters for sets and lists of sets, lists, and clusters), and normal type letters for numbers and strings. Lists are shown with square and sets with curly brackets, where lists have an order but sets do not.

Modelling Temporal Constraints: One aspect of all three phases of our temporal clustering approach is the consideration of temporal constraints of which pairs of records to consider for linkage. Temporal constrains between records can include that the birth record of a person must be before their death record, a marriage record can only occur once an individual has reached a certain age, or (for our clustering problem) the same mother can only give birth to several babies according to certain biological limitations (at least nine months apart or within a few days for multiple births such as twins) [18].

We model such temporal constraints as a list $\mathbf{T}$ of time intervals where it is *plausible* (or not) for two records to be linked (such as a mother to give birth to two babies). We assume each record $r_i \in \mathbf{R}$ includes a time-stamp, $r_i.t$, such as a date of birth, marriage, or death. Based on these time-stamps we can calculate the temporal difference $\Delta t_{i,j} = r_i.t - r_j.t$ between two records where $\Delta t_{i,j}$ is positive if $r_i.t > r_j.t$ (i.e., r_i refers to a life event that occurred after r_j).

The list $\mathbf{T}$ contains time intervals and *temporal plausibilities*, p, where $p = 1$ means two records are temporally plausible and $p = 0$ means they are not, in the form of tuples $(\Delta t^{start}, p^{start}, \Delta t^{end}, p^{end})$, with $\Delta t^{start} < \Delta t^{end}$. For example, for birth records, $\mathbf{T}$ = [(0,1,2,1), (3,0,273,0), (274,1,12783,1), (12784,0,99999,0)] means that two births by the same mother up-to two days apart are plausible, as are two births nine months to 35 years (274 to 12,783 days) apart, but not two births between three days to nine months or more than 35 years apart.

We calculate the temporal plausibility, $p_{i,j}$, for a pair of records (r_i, r_j), by first identifying the corresponding time difference interval in $\mathbf{T}$ for $\Delta t_{i,j}$, and then calculating the pair's temporal plausibility using linear interpolation as:

$$p_{i,j} = \begin{cases} p^{start}, & \text{if } \Delta t_{i,j} = \Delta t^{start}, \\ p^{end}, & \text{if } \Delta t_{i,j} = \Delta t^{end}, \\ (p^{end} - p^{start}) \cdot \frac{(\Delta t_{i,j} - \Delta t^{start})}{(\Delta t^{end} - \Delta t^{start})} + p^{start}, & \text{if } \Delta t^{start} < \Delta t_{i,j} < \Delta t^{end}. \end{cases} \tag{1}$$

If the calculated $p_{i,j}$ is below a given minimum temporal plausibility threshold p_{min} (provided as input to Algo. 1), then the corresponding record pair is deemed not to be temporally plausible and it will not be compared. Currently we assume the list $\mathbf{T}$ of temporal constraints is provided by domain experts. As future work, we aim to develop techniques to learn such temporal constraints using true matching record pairs available in ground truth data.

Similarity Graph Generation: In the first phase of our approach, as detailed in Algo. 1, we calculate pair-wise record similarities. This is a standard record linkage approach [5] using techniques such as approximate string comparisons and a locality sensitive hashing (LSH) [15] based blocking approach.

The main input to the algorithm is a list of records, $\mathbf{R}$, which we aim to link and cluster (in our case we aim to determine which birth records are by the same mother). In order to calculate the pair-wise similarity between record pairs, we use a list of attributes $\mathbf{A}$ and approximate string comparison functions $\mathbf{S}$, such as Jaro-Winkler and edit distance [5], as appropriate to the type of data in an attribute. The calculated attribute similarities may or may not be weighted using the provided list of weights $\mathbf{w}$ (unweighted if all elements of $\mathbf{w}$ are set to 1). In general record linkage, assigning different weights to attributes can increase the quality of the generated links between records [5]. Higher weights can, for example, be assigned to first and last name similarities compared to occupation because names are more likely to help identify matching record pairs.

Algorithm 1: ***Pair-wise similarity graph generation***

Input:	
- $\mathbf{R}$:	List of records to be linked
- $\mathbf{A}$:	List of attributes from $\mathbf{R}$ to be compared
- $\mathbf{S}$:	List of similarity functions to be applied on attributes from $\mathbf{A}$
- $\mathbf{w}$:	List of weights given to attribute similarities, with $\lvert\mathbf{w}\rvert = \lvert\mathbf{S}\rvert$
- $\mathbf{T}$:	List of temporal constraints
- b, r	Number of bands and band size for min-hash based LSH blocking
- p_{min}:	Minimum temporal plausibility for record pairs to be compared
- s_{pmin}:	Minimum pair-wise similarity for record pairs to be added to the generated graph
Output:	
- $\mathbf{G}$:	Undirected pair-wise similarity graph

```
1:  $\mathbf{V} = \emptyset$, $\mathbf{E} = \emptyset$, $\mathbf{G} = (\mathbf{V}, \mathbf{E})$                         // Initialise an empty graph
2:  $\mathbf{L} = MinHashLSHIndexing(\mathbf{R}, b, r)$                                  // Generate a min-hash index
3:  for $\mathbf{l} \in \mathbf{L}$ do:                                                  // Loop over all min-hash blocks
4:      for $(r_i, r_j) : r_i \in \mathbf{l}, r_j \in \mathbf{l}, r_i.id < r_j.id$ do:          // Loop over all record pairs in a block
5:          if $IsTempPlausible(r_i, r_j, \mathbf{T}, p_{min})$ then:      // Check the temporal plausibility of pair
6:              $\mathbf{s}_{i,j} = CompareRecords(r_i, r_j, \mathbf{A}, \mathbf{S}, \mathbf{w})$             // Calculate record pair similarity
7:              $s_{i,j} = NormaliseSim(\mathbf{s}_{i,j}, \mathbf{w})$                              // Normalise the similarity
8:              if $s_{i,j} \geq s_{pmin}$ then:
9:                  UpdateGraph($\mathbf{G}, (r_i, r_j), s_{i,j}$)    // Add edge and nodes (if they do not exist) to G
10: return G
```

To prevent a full comparison of every possible record pair $(r_i, r_j) : r_i, r_j \in \mathbf{R}$ we employ blocking using min-hash based LSH [15] as parameterised using b (the number of min-hash bands) and r (the band size). Only record pairs (r_i, r_j) in the same LSH block will be compared in detail (line 6 in Algo. 1). Furthermore, before comparing records, in line 5 we check if a pair of records is temporally plausible with regard to the list $\mathbf{T}$ of temporal constraints, as described above. The generated undirected similarity graph, $\mathbf{G}$, basically contains records as nodes and edges between records if the calculated normalised similarity, $s_{i,j}$ between two compared records r_i and r_j is at least the provided minimum threshold, s_{pmin}, and the two records are also temporal plausible with regard to $\mathbf{T}$.

4 Temporal Connected Component Clustering

In the second phase of our approach, based on the ideas of link strength (described below) as proposed by Saeedi et al. [21], we generate a set of connected components (clusters) using the similarity graph $\mathbf{G}$, where every pair of records in a cluster must be temporally consistent. The original connected component based clustering approach by Saeedi et al. [21] differs from ours in that it does not consider temporal constraints and also assumes the linkage of records across multiple data sources only. The requirement of incorporating temporal constraints makes the problem much more complex, since simply obtaining the connected components does not ensure temporal consistency between all records within a component, as the example in Fig. 1 shows.

Extending the ideas described by Saeedi et al. [21], and using a minimum cluster similarity threshold, s_{cmin}, with $s_{cmin} \geq s_{pmin}$ (the pair-wise similarity threshold used in Algo. 1), we categorise the edges in $\mathbf{G}$ into three types:

- **Strong**: An edge (r_i, r_j) is *strong* if (1) the corresponding similarity $s_{i,j}$ is the highest similarity for both records r_i and r_j with regard to any other edges they have with other records in $\mathbf{G}$, and (2) $s_{i,j} \geq s_{cmin}$.

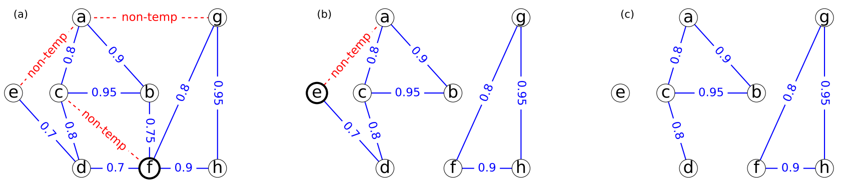

Fig. 1: Example iterative temporal cluster refinement in the base cluster generation phase, as detailed in Algo. 2, where in each step we identify the best edge(s) to be removed that most improve the temporal consistency of the cluster(s).

- **Norm**: An edge (r_i, r_j) is *normal* if (1) the corresponding similarity $s_{i,j}$ is the highest similarity for either record r_i or r_j (but not both) with regard to any other edges they have with other records in $\mathbf{G}$, and (2) $s_{i,j} \geq s_{cmin}$.
- **WeakHigh**: An edge (r_i, r_j) is *weak high* if (1) it is neither strong nor normal, and (2) $s_{i,j} \geq s_{cmin}$.

As detailed in Algo. 2, one or several of these edge types are used to create the initial connected components (named *base clusters*). Edges (r_i, r_j) with similarity $s_{i,j} < s_{cmin}$ are ignored. The temporal implausible base clusters are then split further until all are temporally consistent.

First, in lines 1 to 7 of Algo. 2, we generate the connected components based on the edges in $\mathbf{G}$ of the selected edge type(s) bt (one or several of **Strong**, **Norm** and **WeakHigh**, as described above) which we retrieve in the set $\mathbf{E}_b$ in line 2. We then check, in line 5, if all pairs of records in a connected component $\mathbf{c}_i$ are temporal plausible. If they are then $\mathbf{c}_i$ is added to the set of base clusters, $\mathbf{C}_b$, and removed from the set of connected components $\mathbf{C}_{cc}$. At the end of this step the clusters left in $\mathbf{C}_{cc}$ are those that contain record pairs that are temporally implausible (like two birth records five months apart).

We next process the clusters in $\mathbf{C}_{cc}$ (lines 8 to 19) one by one. We pick one $\mathbf{c}_j \in \mathbf{C}_{cc}$ (line 9) and generate a list $\mathbf{N}$ which for each node $v_i \in \mathbf{c}_j$ contains its average similarity with the other nodes in $\mathbf{c}_j$, its neighbours in $\mathbf{c}_j$, and the other nodes in $\mathbf{c}_j$ it is temporally not plausible with. In line 13, using the function $GetNodeToRefineCluster()$ we identify from $\mathbf{N}$ the best $v_i \in \mathbf{c}_j$ to process which reduces by most the number of temporal implausible edges in $\mathbf{c}_j$.

To select the best node v_{ref}, in line 13 we first attempt to find the first node in $\mathbf{N}$ with a non-empty intersection between its set of neighbours $\mathbf{n}_{ref}$ and the set of neighbours of nodes which v_{ref} is temporal inconsistent with, $\mathbf{nnt}_{ref}$. If the intersection is empty for all nodes in $\mathbf{N}$, v_{ref} will be the node with the lowest average similarity in the cluster, the lowest number of neighbours, and that is involved in the highest number of implausible edges in $\mathbf{c}_j$. In the example shown in Fig. 1, assuming the nodes with non-temporal connections are ordered as $\mathbf{N} = [f, e, a, g, c]$, we select node f first since it is the first node in $\mathbf{N}$ with a non-empty intersection ($\mathbf{n}_{ref} \cap \mathbf{nnt}_{ref} = \{b, d\}$). Subsequently (Fig. 1 (b)), we check nodes e and a in that order, for non-empty intersection. However, since the intersection is empty for both nodes a and e, node e is selected for removal.

Algorithm 2: ***Connected component base cluster generation***

```
Input:
- $\mathbf{G}$:        Undirected pair-wise similarity graph
- $\mathbf{T}$:        List of temporal constraints (as discussed in Sect. 3)
- $p_{min}$:   Minimum plausibility threshold for record pairs to be added to a cluster
- $s_{cmin}$:  Minimum similarity for record pairs to be added to a cluster
- $bt$:        Type(s) of edges to use to create base clusters
Output:
- $\mathbf{C}_b$:       Set of generated temporal consistent base clusters

1:  $\mathbf{C}_b = \{\}$                                          // Initialise an empty set of clusters
2:  $\mathbf{E}_b = FindTempEdges(\mathbf{G}, bt, s_{cmin})$                   // Get temporal edges of type $bt$
3:  $\mathbf{C}_{cc} = GetConnComp(\mathbf{G}, \mathbf{E}_b)$                  // Get the set of connected components
4:  for $\mathbf{c}_i \in \mathbf{C}_{cc}$ do:                      // Iterate through the connected components
5:    if $IsTempPlausibleCluster(\mathbf{c}_i, \mathbf{T}, p_{min})$ do:
6:        $\mathbf{C}_b = \mathbf{C}_b \cup \{\mathbf{c}_i\}$                   // Add the cluster to the final cluster set
7:        $\mathbf{C}_{cc} = \mathbf{C}_{cc} \setminus \{\mathbf{c}_i\}$                        // Remove the processed cluster
8:  while $\mathbf{C}_{cc} \neq \emptyset$ do:              // Iterate through the temporal inconsistent clusters
9:     $\mathbf{c}_j = \mathbf{C}_{cc}.pop()$                            // Get the first connected component
10:    $\mathbf{N} = [\,]$                   // Initialise a list to hold cluster nodes and node information
11:    for $v_i \in \mathbf{c}_j$ do:                            // Iterate through the nodes in cluster $\mathbf{c}_j$
12:       $\mathbf{N}.add((CalcSim(v_i, \mathbf{c}_j, \mathbf{G}), GetNeigh(v_i, \mathbf{c}_j),$
                  $GetTempNotPlausible(\mathbf{c}_j, v_i, \mathbf{T}, p_{min}), v_i))$
13:    $\mathbf{n}_{ref}, \mathbf{nnt}_{ref}, v_{ref} = GetNodeToRefineCluster(\mathbf{N})$          // Select node to refine $\mathbf{c}_j$
14:    $\mathbf{C}_r = GetTempImproved(\mathbf{c}_j, v_{ref}, \mathbf{n}_{ref}, \mathbf{nnt}_{ref})$          // Partition $\mathbf{c}_j$ based on $v_{ref}$
15:    for $\mathbf{c}_i \in \mathbf{C}_r$ do:
16:       if $IsTempPlausibleCluster(\mathbf{c}_i, \mathbf{T}, p_{min})$ do:
17:           $\mathbf{C}_b = \mathbf{C}_b \cup \{\mathbf{c}_i\}$                // Add cluster to the final base cluster set
18:       else:
19:           $\mathbf{C}_{cc} = \mathbf{C}_{cc} \cup \{\mathbf{c}_i\}$            // If not temporal, add cluster to $\mathbf{C}_{cc}$ to be refined
20: return $\mathbf{C}_b$
```

Based on the selected node v_{ref}, and sets $\mathbf{n}_{ref}$ and $\mathbf{nnt}_{ref}$, we then partition the cluster $\mathbf{c}_j$ (line 14) using the function *GetTempImproved*() which returns the set $\mathbf{C}_r$ of two or more temporally improved clusters. In lines 15 to 19 we check each cluster $\mathbf{c}_i \in \mathbf{C}_r$ if it is temporally consistent (in which case we add it to the set of base clusters, $\mathbf{C}_b$) or not (in which case we add it to the set of clusters $\mathbf{C}_{cc}$ to be processed further). In Fig. 1, the edges that node f has with its neighbours $\{b, d\}$ are removed first, and then edges of node e are removed next resulting in the three temporally consistent clusters shown in Fig. 1 (c).

The algorithm ends once all clusters in $\mathbf{C}_{cc}$ have been processed and the set of temporally consistent base clusters, $\mathbf{C}_b$, that is to be refined in the next phase of our approach, is returned in line 20 of Algo. 2.

5 Iterative Cluster Merging

In the final phase of our approach we merge base clusters that have high overall similarities between all their individual records. This process is iterative, in that merged clusters will be further compared until no cluster is highly similar with any other cluster. We ensure all merged clusters are temporally consistent.

As detailed in Algo. 3, we use a priority queue and sets of similar clusters to keep track of cluster pairs that are similar in order to prevent a full pair-wise recalculation of cluster similarities each time a merged cluster is generated. We start the algorithm (lines 1 and 2) by initialising the empty set of final clusters to be generated, $\mathbf{C}_f$, and the empty priority queue, $\mathbf{Q}$, which will hold cluster pairs and their similarities.

Algorithm 3: ***Similar base cluster merging***

```
Input:
- $\mathbf{G}$:        Undirected pair-wise similarity graph
- $\mathbf{C}_b$:      Base clusters (as generated in Algo. 2)
- $\mathbf{T}$:        List of temporal constraints (as discussed in Sect. 3)
- $p_{min}$:   Minimum plausibility threshold for record pairs to be considered temporal consistent
- $s_{mmin}$:  Minimum similarity threshold for clusters to be merged
- $mt$:        Type of edges to use to merge base clusters
- $mm$:        Method to merge base clusters (cluster similarity calculation method)
- $w_{sim}$:   Weight to assign to cluster similarity versus cluster coverage
Output:
- $\mathbf{C}_f$:      Final set of generated clusters
1:  $\mathbf{C}_f = \{\}$                                   // Initialise an empty set of final clusters
2:  $\mathbf{Q} = []$              // Initialise an empty priority queue which will be sorted by similarity
3:  $\mathbf{E}_m = FindTempEdges(\mathbf{G}, mt, s_{mmin})$                  // Get temporal edges of type $mt$
4:  for $(\mathbf{c}_i, \mathbf{c}_j) \in \mathbf{C}_b, i < j$ do:                     // Loop over all cluster pairs in $\mathbf{C}_b$
5:      $\mathbf{Q}.add((CalcSim(\mathbf{c}_i, \mathbf{c}_j, \mathbf{G}, \mathbf{E}_m, mm, w_{sim}), \mathbf{c}_i, \mathbf{c}_j))$  // Add cluster pair and its similarity
6:  while $\mathbf{Q} \neq \emptyset$ do:                                   // Iterate through cluster pairs in $\mathbf{Q}$
7:      $s_{x,y}, \mathbf{c}_x, \mathbf{c}_y = \mathbf{Q}.pop()$                    // Get the most similar cluster pair from $\mathbf{Q}$
8:      $\mathbf{S}_x = \{\mathbf{c}_p : (\mathbf{c}_p, \mathbf{c}_x) \in \mathbf{Q} \ \wedge \ s_{p,x} \geq s_{mmin}, \ \mathbf{c}_p \neq \mathbf{c}_y\}$  // Set of clusters highly similar to $\mathbf{c}_x$
9:      $\mathbf{S}_y = \{\mathbf{c}_q : (\mathbf{c}_q, \mathbf{c}_y) \in \mathbf{Q} \ \wedge \ s_{q,y} \geq s_{mmin}, \ \mathbf{c}_q \neq \mathbf{c}_x\}$  // Set of clusters highly similar to $\mathbf{c}_y$
10:     $RemoveAllTuplesWithCluster(\mathbf{Q}, \mathbf{c}_x)$              // Remove tuples containing $\mathbf{c}_x$ from $\mathbf{Q}$
11:     $RemoveAllTuplesWithCluster(\mathbf{Q}, \mathbf{c}_y)$              // Remove tuples containing $\mathbf{c}_y$ from $\mathbf{Q}$
12:     if $(s_{x,y} \geq s_{mmin})$ and $IsTempPlausibleClusterPair(\mathbf{c}_x, \mathbf{c}_y, \mathbf{T}, p_{min})$ do:
13:         $\mathbf{c}_{mer} = \mathbf{c}_x \cup \mathbf{c}_y$ // Merge clusters if similarity is high enough and if temporally plausible
14:         if $\mathbf{S}_x \cup \mathbf{S}_y = \emptyset$ do:                       // If no clusters are similar with $\mathbf{c}_x$ or $\mathbf{c}_y$
15:             $\mathbf{C}_f = \mathbf{C}_f \cup \{\mathbf{c}_{mer}\}$                     // Add merged cluster to final clusters
16:         else:
17:             for $\mathbf{c}_z \in \mathbf{S}_x \cup \mathbf{S}_y$ do:          // Add $\mathbf{c}_{mer}$ with clusters similar to $\mathbf{c}_x$ or $\mathbf{c}_y$ into $\mathbf{Q}$
18:                 $\mathbf{Q}.add((CalcSim(\mathbf{c}_z, \mathbf{c}_{mer}, \mathbf{G}, \mathbf{E}_m, mm, w_{sim}), \mathbf{c}_z, \mathbf{c}_{mer}))$
19:     else:
20:         $\mathbf{C}_f = \mathbf{C}_f \cup \{\mathbf{c}_x, \mathbf{c}_y\}$    // Add cluster pair $\mathbf{c}_x$ and $\mathbf{c}_y$ to final clusters if non-mergeable
21: return $\mathbf{C}_f$
```

In lines 3 to 5, we calculate the similarities between every pair of base clusters in $\mathbf{C}_b$, where we consider a set of edge types, mt, different to Algo. 2, with one or several of **Strong**, **Norm**, and **WeakHigh**, as described before.

In line 5 we calculate the similarity between a cluster pair $\mathbf{c}_i$ and $\mathbf{c}_j$ using the function $CalcSim()$ which also takes as input a merge method, mm, and a cluster similarity weight, w_{sim}. mm determines how the overall similarity between clusters is calculated, where it can be one of the aggregation functions minimum, average or maximum. The aggregated similarity between two clusters is assigned a weight of w_{sim} whereas a weight of $1-w_{sim}$ is assigned for coverage. The coverage is the ratio between the number of edges across $\mathbf{c}_i$ and $\mathbf{c}_j$ in $\mathbf{E}_m$, and the number of edges across $\mathbf{c}_i$ and $\mathbf{c}_j$ in $\mathbf{G}$, which reflects the proportion of edges covered in our similarity calculation. The cluster similarity, $s_{x,y}$, returned by $CalcSim()$ is the weighted sum of similarity and coverage.

The main loop starts in line 6 and iterates over each cluster pair tuple in the queue $\mathbf{Q}$. For both clusters in the tuple, $\mathbf{c}_x$ and $\mathbf{c}_y$, we next (lines 8 and 9) identify all other clusters that they are similar with, and we keep these clusters in two sets $\mathbf{S}_x$ and $\mathbf{S}_y$, respectively. We then remove all tuples in $\mathbf{Q}$ that contain $\mathbf{c}_x$ or $\mathbf{c}_y$ since they should not be re-processed. In line 12 we check if the similarity between $\mathbf{c}_x$ and $\mathbf{c}_y$ is at least the minimum cluster merge similarity s_{mmin} and if they are temporally consistent with each other. If this is the case we merge clusters $\mathbf{c}_x$ and $\mathbf{c}_y$ into $\mathbf{c}_{mer}$ in line 13.

If both $\mathbf{c}_x$ and $\mathbf{c}_y$ are not similar with any other clusters (i.e. both $\mathbf{S}_x$ and $\mathbf{S}_y$ are empty; the test in line 14), then based on the triangular inequality we know that the merged cluster $\mathbf{c}_{mer}$ cannot be merged further with any other clusters. Therefore $\mathbf{c}_{mer}$ is added to the set of final clusters, $\mathbf{C}_f$, in line 15. Otherwise, in line 17 we calculate the similarity of the merged cluster, $\mathbf{c}_{mer}$, with each cluster in $\mathbf{S}_x$ and $\mathbf{S}_y$ and add new tuples into the queue in line 18.

If a cluster pair in the queue was not similar enough or not temporally consistent, we do not merge $\mathbf{c}_x$ and $\mathbf{c}_y$ but instead add both into $\mathbf{C}_f$ in line 20. Finally we return the set of merged and temporally consistent clusters, $\mathbf{C}_f$.

6 Experimental Evaluation

We evaluated our temporal clustering approach using a real Scottish data set, as provided by [18], that covers the population of the Isle of Skye over the period from 1861 to 1901. This data set consists of 17,614 birth certificates, where each of these contains personal details about a baby and its parents such as their names, address, marriage date, occupations, and the birth date. As with other historical data [1,11], this data set has a very small number of unique name values (2,055 first and only 547 last names), and the frequency distributions of names are also very skewed. Many records have missing addresses or occupations, and for unmarried women the details of a baby's father are mostly missing.

This data set has been extensively curated and linked semi-manually by demographers who are experts in the domain of linking such historical data [18]. Their approach followed long established rules for family reconstruction, leading to a set of linked birth records. We thus have a set of manually generated links of births that allows us to compare the quality of the different clustering techniques, and to evaluate how temporal constraints can improve linkage quality.

We used three different subsets of attributes to compare record pairs and generate the similarity graph $\mathbf{G}$ discussed in Sect. 3: *All* (parents names, addresses, occupations, and marriage dates), *Parent names and addresses*, and *Parent names* only. To compare attribute values we used approximate string comparison functions such as Jaro-Winkler and edit-distance [5]. We used both unweighted (UW) and weighted (W) similarities, where weights were calculated based on the traditional Fellegi and Sunter record linkage approach [10]. We thus ended up with six different similarity graphs $\mathbf{G}$ where we set $s_{pmin} = 0.7$ to only include pair-wise links with at least this normalised similarity.

As discussed in Sect. 4, we evaluated different combinations of the edge types **Strong**, **Norm**, and **WeakHigh** in our approach. For Algo. 2 we generated base clusters with only **Strong** edges because these clusters showed much higher precision (95%) in set-up experiments compared to using other edge type combinations. In Algo. 3 we used '**Norm**', '**Norm** with **WeakHigh**' (where base clusters were merged using both edge types in the same run), as well as '**Norm** and **WeakHigh**' (where base clusters were first merged using **Norm** edges and then the resulting clusters were merged again using **WeakHigh** edges). As discussed in Sect. 5, we used the three cluster merge methods (*mm*): **Min** (minimum pair-

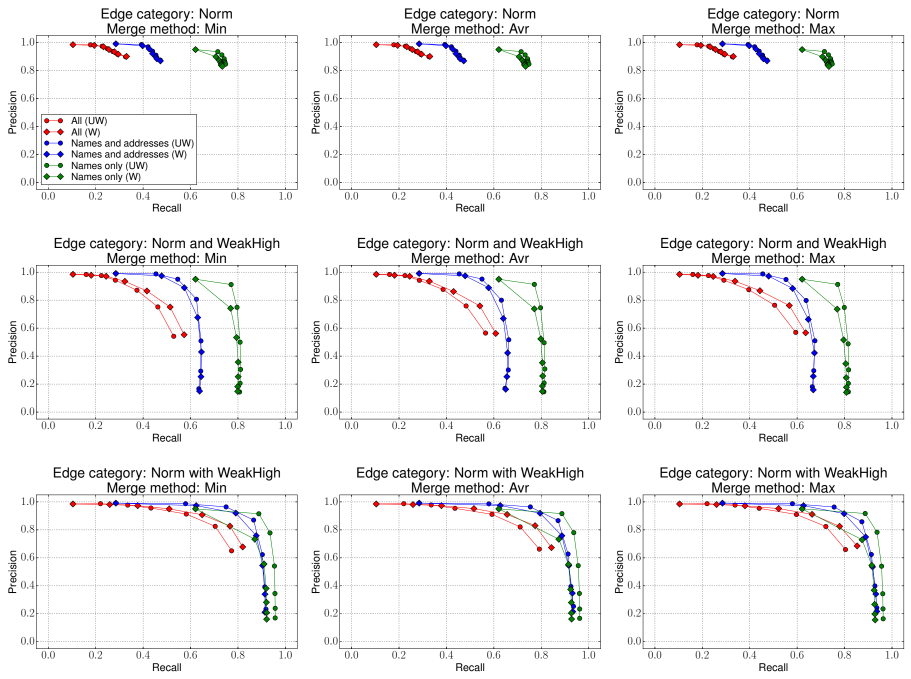

Fig. 2: Precision and recall of our approach with different merge methods (as discussed in Sect. 5) for different similarity graphs as described in Sect. 6.

wise record similarity across two clusters), **Avr** (average pair-wise similarity), and **Max** (maximum pair-wise similarity). We also ran experiments where we did not consider any temporal constraints, i.e. we set $\mathbf{T} = []$ in all algorithms.

We calculated linkage quality as precision (the ratio of true links identified against all links within clusters) and recall (the ratio of true links identified against all true links) [5]. We do not present F-measure results given recent work has identified some problematic aspects with this measure when used for record linkage [12]. Instead we present the area under the precision-recall curve (AUC-PR) which has shown to be robust for class imbalance problems [7].

We compared the proposed approach with our recent star clustering based method [16], as this is the only approach we are aware of that uses temporal aspects for group linkage. We applied LSH [15] (blocking) to limit the number of record pairs being compared, resulting in a recall of 99.7% of true matches for the similarity graph **G**. We set the similarity threshold in Algos. 2 and 3 from 1.0 to 0.7 in 0.05 steps, and the weight w_{sim} in Algo. 3 to 0.5. We implemented all techniques using Python 2.7, and the programs and similarity graphs are available from the authors to facilitate repeatability.

Fig. 2 shows precision and recall curves of our approach with different edge combinations. As can be seen, the edge combination '**Norm** with **WeakHigh**' provided the best linkage quality compared to other edge combinations. The reason for this is that when using **Norm** edges only, many true links that are in

Table 1: The area under the precision-recall curve (AUC-PR) results (averages and standard deviations) of our approach and star based clustering [16], with (T) and without (NT) temporal constraints for different similarity graphs.

Similarity graph	ConnComp (T)	Star (T)	ConnComp (NT)	Star (NT)
All (UW)	0.72 ± 0.012	0.70 ± 0.003	0.64 ± 0.005	0.63 ± 0.003
All (W)	0.77 ± 0.014	$\mathbf{0.74} \pm 0.006$	0.69 ± 0.005	0.68 ± 0.004
Names and addresses (UW)	0.87 ± 0.006	0.70 ± 0.014	0.83 ± 0.002	0.73 ± 0.003
Names and addresses (W)	0.86 ± 0.007	0.69 ± 0.016	0.80 ± 0.003	0.72 ± 0.007
Names only (UW)	$\mathbf{0.88} \pm 0.002$	0.72 ± 0.018	$\mathbf{0.85} \pm 0.001$	$\mathbf{0.78} \pm 0.015$
Names only (W)	0.80 ± 0.002	0.65 ± 0.016	0.73 ± 0.001	0.69 ± 0.019
Averages	$\mathbf{0.82} \pm 0.064$	0.70 ± 0.030	0.76 ± 0.083	0.71 ± 0.051

the **WeakHigh** category are ignored. It also appears that merging clusters in a single run using the '**Norm** with **WeakHigh**' method provided better results than conducting the cluster merging in two phases (as done in the '**Norm** and **WeakHigh**' method). We also noted that the quality of clustering does not change much with the merge method *mm* (**Min**, **Avr**, or **Max**) because most clusters generated only contained between 2 to 4 records.

Finally, Table 1 shows AUC-PR results of our approach with different similarity graphs. The average and standard deviations of AUC-PR values across the three merge methods are reported for the best edge combination '**Norm** with **WeakHigh**' from Fig. 2. As can be seen, our approach achieved the highest AUC-PR value of 0.88 with temporal constrains while it resulted in an AUC-PR value of 0.85 without any temporal constraints. We conducted a t-test to evaluate the statistical significance between the AUC-PR values of 0.88 and 0.85, which resulted in a p-value less than 0.0001. Such high statistical significance confirms that the use of temporal constraints can improve the overall linkage quality in our approach. Further, as this table shows, our approach outperformed both temporal and non-temporal star clustering in terms of linkage quality for all similarity graphs which indicates our approach is suitable to cluster records, such as the births by the same mothers in the context of historical record linkage, with high linkage quality.

7 Conclusions and Future Work

We have presented a temporal clustering approach for group record linkage. Our approach first generates a graph that represents the similarities calculated between individual records, and then generates temporally consistent connected components which are merged to obtain a set of high quality clusters. Our experimental evaluation on a real data set from Scotland has shown that our approach can substantially outperform a previous temporal clustering approach for record linkage. In the future we aim to conduct empirical evaluations for different data sets and parameter settings. We further plan to conduct a complexity analysis on the proposed algorithms and also learn temporal constraints for different time intervals using ground truth data based on true matching record pairs.

Acknowledgements

This work was supported by ESRC grants ES/K00574X/2 *Digitising Scotland* and ES/L007487/1 *ADRC-S*. We like to thank Alice Reid of the University of Cambridge and her colleagues Ros Davies and Eilidh Garrett for their work on the Isle of Skye database, and their helpful advice on historical Scottish demography. This work was partially funded by the Australian Research Council under DP160101934.

References

1. Antonie, L., Inwood, K., Lizotte, D.J., Ross, J.A.: Tracking people over time in 19th century Canada for longitudinal analysis. Machine Learning **95**, 129–146 (2014)
2. Aslam, J.A., Pelekhov, E., Rus, D.: The star clustering algorithm for static and dynamic information organization. J. Graph Algorithms Appl. **8**, 95–129 (2004)
3. Bailey, M., Cole, C., et al.: How well do automated methods perform in historical samples? Evidence from new ground truth. Tech. rep., NBER (2017)
4. Bloothooft, G., Christen, P., Mandemakers, K., Schraagen, M.: Population Reconstruction. Springer (2015)
5. Christen, P.: Data Matching. Springer (2012)
6. Christen, V., Groß, A., Fisher, J., Wang, Q., Christen, P., Rahm, E.: Temporal group linkage and evolution analysis for census data. In: EDBT. Venice (2017)
7. Davis, J., Goadrich, M.: The relationship between Precision-Recall and ROC curves. In: ACM ICML. pp. 233–240. Pittsburgh (2006)
8. Dibben, C., Williamson, L., Huang, Z.: Digitising Scotland (2012), `http://gtr.rcuk.ac.uk/projects?ref=ES/K00574X/2`
9. Dillon, L.Y.: Integrating nineteenth-century Canadian and American census data sets. Computers and the Humanities **30**(5), 381–392 (1996)
10. Fellegi, I.P., Sunter, A.B.: A theory for record linkage. Journal of the American Statistical Association **64**(328), 1183–1210 (1969)
11. Fu, Z., Christen, P., Zhou, J.: A graph matching method for historical census household linkage. In: PAKDD. Tainan, Taiwan (2014)
12. Hand, D., Christen, P.: A note on using the f-measure for evaluating record linkage algorithms. Statistics and Computing **28**(3), 539–547 (2018)
13. Hassanzadeh, O., Chiang, F., Lee, H.C., Miller, R.J.: Framework for evaluating clustering algorithms in duplicate detection. PVLDB **2**(1), 1282–1293 (2009)
14. Kum, H.C., Krishnamurthy, A., Machanavajjhala, A., Ahalt, S.C.: Social genome: Putting big data to work for population informatics. IEEE Computer **47**(1) (2014)
15. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Cambridge University Press (2014)
16. Nanayakkara, C., Christen, P., Ranbaduge, T.: Temporal graph-based clustering for historical record linkage. In: MLG, held at ACM SIGKDD. London (2018)
17. On, B.W., Koudas, N., Lee, D., Srivastava, D.: Group linkage. In: IEEE ICDE. Istanbul (2007)
18. Reid, A., Davies, R., Garrett, E.: Nineteenth-century Scottish demography from linked censuses and civil registers. History and Computing **14**(1-2) (2002)
19. Ruggles, S., Fitch, C.A., Roberts, E.: Historical census record linkage. Annual Review of Sociology **44**(1), 19–37 (2018)
20. Saeedi, A., Peukert, E., Rahm, E.: Comparative evaluation of distributed clustering schemes for multi-source entity resolution. In: ADBIS. pp. 278–293. Nicosia (2017)
21. Saeedi, A., Peukert, E., Rahm, E.: Using link features for entity clustering in knowledge graphs. In: ESWC. pp. 576–592. Heraklion, Crete (2018)