Clustering-based Scalable Indexing for Multi-party Privacy-preserving Record Linkage*

Thilina Ranbaduge, Dinusha Vatsalan, and Peter Christen

Research School of Computer Science, College of Engineering and Computer Science
The Australian National University, Canberra ACT 0200, Australia
{thilina.ranbaduge,dinusha.vatsalan,peter.christen}@anu.edu.au

Abstract. The identification of common sets of records in multiple databases has become an increasingly important subject in many application areas, including banking, health, and national security. Often privacy concerns and regulations prevent the owners of the databases from sharing any sensitive details of their records with each other, and with any other party. The linkage of records in multiple databases while preserving privacy and confidentiality is an emerging research discipline known as privacy-preserving record linkage (PPRL). We propose a novel two-step indexing (blocking) approach for PPRL between multiple (more than two) parties. First, we generate small mini-blocks using a multi-bit Bloom filter splitting method and second we merge these mini-blocks based on their similarity using a novel hierarchical canopy clustering technique. An empirical study conducted with large datasets of up-to one million records shows that our approach is scalable with the size of the datasets and the number of parties, while providing better privacy than previous multi-party indexing approaches.

Keywords: hierarchical canopy clustering; Bloom filters; scalability.

1 Introduction

Many real-world applications require data from multiple databases to be integrated and combined to improve data analysis and mining. Record linkage (also known as entity resolution or data matching) is the process of identifying matching records that refer to the same entity from multiple databases [3].

The lack of unique entity identifiers in databases requires the use of quasi-identifiers (QIDs) [16], such as first name, last name, address details, etc. to link records across databases. However, due to privacy and confidentiality concerns organizations generally do not want to share any sensitive information regarding their entities with other data sources. Finding records in multiple databases that relate to the same entity or having approximately the same values for a set of QIDs without revealing any private or sensitive information is the research area known as 'privacy-preserving record linkage' (PPRL) [8,16].

* This research is funded by the Australian Research Council under Discovery Project DP130101801.

The naive pair-wise comparison of multiple data sources is of exponential complexity in terms of the number of parties. This makes record linkage applications not scalable to large databases and increasing number of participating parties. Applying indexing is a possible solution aimed at improving scalability [4]. Indexing reduces the large number of potential comparisons by removing as many record sets as possible that correspond to non-matches, such that expensive similarity comparisons are only required on a smaller number of candidate record sets. Indexing for PPRL needs to be conducted such that no sensitive information that can be used to infer individual records and their attribute values is revealed to any party involved in the process, or to an external adversary [18].

We propose a novel indexing mechanism for multi-party PPRL that can provide better scalability, blocking quality, and privacy compared to previous approaches. Our approach efficiently creates blocks across multiple parties without revealing any private information. Specific contributions of our paper are (1) a two-step blocking protocol which consists of (2) a multi-bit splitting approach for Bloom filters and (3) a hierarchical canopy clustering algorithm; and (4) an empirical evaluation using large datasets, and a comparison with other multi-party approaches in terms of efficiency, effectiveness and privacy.

2 Related Work

Can we do efficient and effective indexing for record linkage? This is a problem that has been considered for several decades. According to a survey by Christen [4], a variety of indexing approaches have been developed. Some of these have been adapted for PPRL [16], including standard blocking [1], mapping based indexing [8], clustering [8,10,19], and locality sensitive hashing [7]. However, performing scalable record linkage that provides high linkage quality while preserving privacy is an open research question that needs further investigation.

Bloom filters are commonly used for encoding of records in PPRL due to their capability of computing similarities. Lai et al. [11] proposed a multi-party approach that uses Bloom filters to securely transfer data between multiple parties for private set intersection. This approach was recently adapted by Vatsalan et al. [17] for multi-party PPRL, however their approach does not address blocking.

Schnell [15] introduced a new blocking method for record linkage, based on the concept of a multi-bit tree data structure [9] to hold a set of Bloom filters. This approach was further extended by Ranbaduge et al. [13] as a blocking mechanism for multi-party PPRL. Their experimental results showed that the proposed approach is scalable with the dataset size and the number of parties, and provides better linkage quality and privacy than a phonetic based indexing approach. However, the blocks generated using this approach might miss some true matches due to the recursive splitting of Bloom filter sets.

Clustering is the process of grouping records such that records within a cluster are similar to each other, while dissimilar records are in different clusters. Several clustering approaches have been adapted for private blocking [8,10,19], however neither of these techniques considers blocking of more than two databases.

Canopy clustering [6,12] is a technique for clustering large high dimensional datasets. It can generate candidate record sets by efficiently calculating similarities between blocking key values. Records are inserted into one or more overlapping clusters based on their similarity to the cluster centroid. Each cluster then forms a block from which candidate sets are generated. However, the use of canopy clustering for indexing in PPRL has so far not been studied.

3 Clustering based Indexing for Multi-party PPRL

We now introduce the building blocks required for our clustering based PPRL indexing approach, and then study the indexing protocol in detail.

3.1 Building Blocks

Bloom Filters: are bit vectors proposed by Bloom [2]. In a Bloom filter initially all the bits are set to 0. To map an element of a set into the domain between 0 and $m-1$ of a Bloom filter of length m, k independent hash functions $h_1, \ldots, h_k$ are used. Furthermore, to store n elements of the set $S = \{s_1, s_2, \ldots, s_n\}$ into a Bloom filter, each element $s_i \in S$ is encoded using the k hash functions and all bits having index positions $h_j(s_i)$ for $1 \leq j \leq k$ are set to 1.

Q-grams: are character sub-strings of length q in a string [3]. For example, the string "PETER" can be padded with character '_' and the resulting q-gram set (for $q = 2$) is $\{\_P, PE, ET, TE, ER, R\_\}$. In our approach we encode q-gram sets of the quasi-identifiers (QIDs) into Bloom filters. First, the selected QIDs of a given record are converted into a q-gram set. Then each q-gram set is encoded into a Bloom filter by using k hash functions.

Secure Summation Protocol: is a method used in secure multi-party computation [5], and has been used in several PPRL approaches [13,17]. The basic idea is to compute a summation of private inputs of P parties without revealing individual values to any other parties, and at the end of the computation no party knows anything except its own input and the final sum [5].

3.2 Basic Multi-bit Indexing Protocol

We now describe our indexing approach for multi-party PPRL. The construction of the index of an individual party consists of two main phases:

1. *Multi-bit Bloom filter splitting*: This phase can be further extended into three steps, which are:
   (a) *Generate Bloom filters for the records in the dataset.*
   (b) *Perform secure summation to find the best splitting bit position.*
   (c) *Split the set of Bloom filters into mini-blocks.*
2. *Merge mini-blocks using clustering.*

Algorithm 1: Multi-bit Bloom filter splitting by party P_i, $1 \leq i \leq P$

Input:
- $\mathbf{D_i}$: Dataset belonging to party P_i
- $\mathbf{A}$: Set of selected attributes
- s_{min}: Minimum *mini-block* split size
- d_{max}: Maximum split degree
- s_{max}: Maximum *mini-block* split size
- bs_t: Bit selection threshold

Output:
- $\mathbf{C}$: Set of Bloom filter mini-blocks

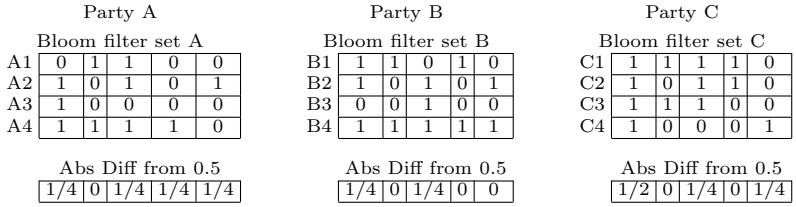
1: B = generateBloomfilters($\mathbf{D_i}$, $\mathbf{A}$)
2: $Q = [B]$ // Initialization of queue
3: **while** $Q \neq \emptyset$ **do:**
4: $b = Q$.pop() // Get the current block
5: R = generateRatios(b) // Generate local bit ratios
6: R_g = secureSummation(R) // Get ratios globally
7: BC_l = getCombinations(R_g,d_{max},bs_t) // Get bit combinations
8: BC_g = secureSummation(BC_l) // Get combinations globally
9: mb_j = splitData(BC_g, b) // Create mini-blocks
10: **if** all ($|mb_j| \geq s_{min}$) **then:** // If mini-blocks large enough
11: **if** any ($|mb_j| > s_{max}$) **then:** // If mini-blocks too large
12: Q.push(mb_j) // Add to queue for further splitting
13: **else:** // Block sizes are acceptable
14: $\mathbf{C}$.add(mb_j) // Add mini-blocks to final set
15: *return* $\mathbf{C}$

Generating large blocks of different sizes makes the comparison step more problematic and requires more computational time. With our approach, a user has control over the block sizes by merging blocks until the size of blocks reaches an acceptable lower limit suitable for comparison. The aim of our protocol is to divide (split) the set of records in the datasets into *mini-blocks* (phase 1) and merge these *mini-blocks* based on their similarity (phase 2). Merging of *mini-blocks* by clustering reduces the overall running time requirement compared to using a clustering technique for blocking the datasets [10,19]. These merged blocks can then be compared using private comparison and classification techniques to determine the matching record sets in different databases [17]. Each party follows these phases to construct the blocks from their own dataset.

3.3 Multi-bit Bloom Filter Splitting

Before performing the clustering algorithm, the set of records needs to be encoded into Bloom filters, and split into sets of smaller blocks (which we call *mini-blocks*). All P parties need to agree upon a bit array length m (length of the Bloom filter); the length (in characters) of grams q, the k hash functions, and a set of QID attributes that are used to block the records. The parameters s_{min} and s_{max} specify the lower and upper bound of the acceptable size of a *mini-block*, respectively. The overall splitting approach for each party P_i, where $1 \leq i \leq P$, is outlined in Algorithm 1.

In line 1, each party iterates over its dataset to encode each record into a Bloom filter. Once all the parties have generated their sets of Bloom filters they are added into a queue Q as a single block. At each iteration the first block of Bloom filters b that is available at the front of Q is processed. In line 5, each party calculates a vector of length m that contains the ratios between the number of



Phase 1.*a*: Bloom filter generation and calculation of 0/1 bit ratios and absolute differences from 50% filled.

Random vector (R) = | 10 | 5 | 12 | 13 | 6 |

10.25	0	12.25	13.25	6.25	→	10.5	0	12.5	13.25	6.25	→	11	0	12.75	13.25	6.5

Secure sums: | 1 | 0 | 0.75 | 0.25 | 0.5 |

Final absolute differences from 50% filled: | 0.33 | **0** | 0.25 | **0.083** | **0.167** |

Selected bit positions: **{2, 4}**

Phase 1.*b*: Secure summation of absolute differences and selecting best bit positions for splitting ($d_{max} = 2$, and $bs_t = 0.2$)

Fig. 1. Selecting $d_{max} = 2$ best bit positions for multi-bit Bloom filter splitting.

0's and 1's for each bit position in the Bloom filters, using $f_{ij} = abs(0.5 - \frac{o_{ij}}{l})$, where f_{ij} is the ratio value of bit position j of party P_i, o_{ij} is the number of 1's in position j, and l is the number of Bloom filters processed in a given block.

Once all parties have computed their own ratio vector, we use an extended secure summation protocol to compute common bit positions suitable for splitting (line 6). The globally summed ratio vector R_g is used to find the d_{max} best splitting bit positions. The bit positions with a sum less than the bit selection threshold bs_t are then selected into the set I_j of splitting bit positions as $I_j = \{j \mid (\frac{P}{2} - \sum_{i=1}^{P} f_{ij}) < bs_t\}$. The d_{max} bit positions in I_j with the lowest ratio values (which we call *match-bits*) are selected as the best splitting bit positions. Fig. 1 illustrates an example of selecting best splitting bits.

Based on the selected bit positions, each party generates all possible bit combinations BC_l (line 7). An example of bit positions $\{j_x, j_y, j_z\}$ would generate the set of combinations $\{\{j_x, j_y, j_z\}, \{j_x, j_y\}, \{j_y, j_z\}, \{j_x, j_z\}, \{j_x\}, \{j_y\}, \{j_z\}\}$. The value of d_{max} needs to be kept small as this generation grows exponentially with d_{max}. For each combination, the Bloom filters in the current block are processed to analyze the sizes of resulting *mini-blocks* with different bit patterns. For a bit combination $\{j_x, j_y\}$, for example, the set of bit patterns is $\{00, 01, 10, 11\}$. Once the current block is processed for all possible bit combinations, we find the common best bit combination BC_g in line 8 in Algorithm 1. The current block is split into *mini-blocks* according to this globally accepted bit combination.

After splitting, if any of the *mini-blocks* contains less than s_{min} records, then these *mini-blocks* will not be included into the final *mini-block* set **C** (line 10). Instead they are merged back into a single block which is added to **C**. If all of the resulting *mini-blocks* contain a number of records greater than s_{min}, then each *mini-block* is checked against the value of s_{max}. All the *mini-blocks* that contain records greater than s_{max} are added to Q for future splitting (line 12 in

Algorithm 1). If the number of records is less than s_{max} at all parties, then these *mini-blocks* are added to $\mathbf{C}$. The parameters s_{min} and s_{max} allow the user to control the number of iterations that occur in the multi-bit splitting algorithm.

4 Merge Mini-blocks by Clustering

Our clustering of *mini-blocks* is based on a canopy technique [6,12]. The generated *mini-blocks* are merged into larger clusters by inserting records into one or more overlapping clusters based on their similarity to their nearest cluster centroid. This allows us to perform computationally efficient indexing. We use a normalized Hamming distance based similarity calculation for computing the similarity of clusters: $sim_H(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{|i|}{m} \mid \boldsymbol{x_i} \neq \boldsymbol{y_i}$ *and* $1 \leq i \leq m$, where $sim_H(\boldsymbol{x}, \boldsymbol{y})$ is the normalized Hamming distance similarity between the two bit vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, $sim_H(\boldsymbol{x}, \boldsymbol{y}) = 1$ if and only if $\boldsymbol{x} = \boldsymbol{y}$.

Each *mini-block* generated by the multi-bit splitting algorithm is initially considered as a separate cluster, which we refer to as a *mini-cluster*. A Bloom filter a_c is selected as the centroid for each *mini-cluster* which has the highest similarity to all other Bloom filters in the cluster. For this selection process we use a maximum average Hamming distance based similarity calculation which is shown in (1).

$$a_c = \underset{a_i}{\operatorname{argmax}} \left\{ \frac{\sum sim_H(a_i, a_j)}{|\mathbf{C}|} \mid a_i, a_j \in \mathbf{C},\ 1 \leq i, j \leq |\mathbf{C}|, and\ i \neq j \right\} \quad (1)$$

We use threshold-based canopy clustering for merging similar *mini-clusters*. Before starting the clustering algorithm, all parties need to agree upon the tight similarity threshold s_t, loose similarity threshold s_l, and the maximum merge size ms_{max} which controls the size of merged clusters (blocks) and indirectly controls the number of iterations of the merging process. For merging of *mini-clusters* we suggest two canopy based clustering algorithms, which are:

- *Standard canopy clustering (SCC)*: *Mini-clusters* are merged until the resulting cluster size increases to ms_{max}. This algorithm merges a set of similar *mini-clusters* greedily in a given iteration.
- *Hierarchical canopy clustering (HCC)*: The merging of *mini-clusters* is based on an agglomerative clustering approach until ms_{max} is met. In a given iteration, the two most similar *mini-clusters* are merged.

4.1 Merge Mini-clusters with Standard Canopy Clustering

The suggested SCC approach for merging of *mini-clusters* is shown in Algorithm 2. In line 2, each party iterates over the set of *mini-clusters* $\mathbf{C}$. At each iteration the *mini-cluster* at the front of $\mathbf{C}$ is processed as the initial cluster (line 3). As discussed above, the centroid is computed for the initial cluster (line 4).

Algorithm 2: Merge mini-clusters using standard canopy clustering

Input:
- s_t: Tight similarity threshold
- s_l: Loose similarity threshold
- ms_{max}: Maximum merge size
- **C**: Set of mini-clusters ($[c_1, \ldots, c_l]$)
- $sim_H(\cdot, \cdot)$: Similarity comparison function

Output:
- **O**: Set of merged clusters

```
1:  O = ∅
2:  while C ≠ ∅ do:
3:    c_x = C.pop()
4:    a_x = getCentroid(c_x)
5:    c_xy = c_x
6:    while |c_xy| ≤ ms_max do:
7:      c_y = C.next()
8:      a_y = getCentroid(c_y)
9:      s = sim_H(a_x, a_y)
10:     if s ≤ s_t then:
11:       c_xy = c_xy + c_y
12:       del c_y
13:     if s ≤ s_l then:
14:       c_xy = c_xy + c_y
15:   del c_x
16:   O.add(c_xy)
```

Algorithm 3: Merge mini-clusters using hierarchical canopy clustering

Input:
- s_t: Tight similarity threshold
- s_l: Loose similarity threshold
- ms_{max}: Maximum merge size
- **C**: Set of mini-clusters ($[c_1, \ldots, c_l]$)
- $sim_H(\cdot, \cdot)$: Similarity comparison function

Output:
- **O**: Set of merged clusters

```
1:  O = ∅
2:  while C ≠ ∅ do:
3:    c_x = C.pop()
4:    a_x = getCentroid(c_x)
5:    for c_y ∈ C do:
6:      a_y = getCentroid(c_y)
7:      s = sim_H(a_x, a_y)
8:      if s ≤ s_t then:
9:        c_xy = c_x + c_y
10:       del c_x, c_y
11:     if s ≤ s_l then:
12:       c_xy = c_x + c_y
13:       del c_x
14:   if |c_xy| ≥ ms_max then:
15:     O.add(c_xy)
16:   else:
17:     C.add(c_xy)
```

The initial cluster is compared and merged with other *mini-clusters* in the set until the size of the cluster reaches ms_{max} as shown in lines 5 to 15. At the merging step, the computed similarity value s (line 9) is checked against s_t and s_l. As per lines 10 to 12, if $s \leq s_t$, then the *mini-clusters* will be merged. The merged *mini-clusters* are removed from the set **C** if they are within s_t (line 12).

If $s \leq s_l$, then the merge is performed between the *mini-clusters* but only the initial cluster is deleted from the set **C** (line 15). Once the size of the resulting merged cluster c_{xy} reaches ms_{max}, the cluster is added to the set of merged clusters **O** (line 16). Therefore at each iteration a set of *mini-clusters* which are similar to the initial cluster are merged until the overall cluster size reaches ms_{max}. Each cluster generated by this approach will become a block to be used in the comparison step in the PPRL pipeline.

4.2 Merge Mini-clusters with Hierarchical Canopy Clustering

In the SCC algorithm described in Sect. 4.1, depending on the sizes of the *mini-clusters* that are merged, the final cluster size can grow beyond ms_{max} which will result in more record set comparisons. We propose a novel threshold-based hierarchical canopy clustering (HCC) approach which guarantees that clusters are only merged up-to the size limit ms_{max} as shown in Algorithm 3.

To merge clusters, each party iterates over its set of *mini-clusters* **C** (line 2). At each iteration one *mini-cluster* is selected and the centroid is computed as discussed in Sect. 4 (lines 3 and 4). A similarity value s is computed between the initial cluster and other *mini-clusters* in **C** (line 7). The computed value s

is checked against s_t and s_l for merging (lines 8 to 13). Similar to Algorithm 2, *mini-clusters* are merged if the value s satisfies the threshold values.

After each iteration, the size of the resulting merged cluster is checked against ms_{max}. If the size of the merged cluster is less than ms_{max}, the cluster is added back into $\mathbf{C}$ as a new *mini-cluster* c' (line 17). This enables c' to be merged further with other close *mini-clusters*. Therefore, at each iteration the two most similar *mini-clusters* are merged into one. Once the size of a merged cluster reaches ms_{max}, it is added to the final set of merged clusters $\mathbf{O}$.

5 Analysis of the Protocol

We now analyze our protocol in terms of complexity, privacy, and quality.

Complexity: By assuming there are N records in a dataset with each having an average of n q-grams, we analyze the computational and communication complexities in terms of a single party.

In the first phase of our protocol all records are encoded using k hash functions. The Bloom filter generation for a single party is of $O(k \cdot n \cdot N)$ complexity. In the multi-bit splitting step, the parameters s_{min}, s_{max} and d_{max} are used to control the size of *mini-blocks* generated. Suitable values for the parameters s_{min} and s_{max} need to be set as the size of the *mini-blocks* decides the number of iterations that occur in the splitting phase. At each iteration, a set of Bloom filters is split into $2^{d_{max}}$ *mini-blocks*. The number of iterations in the splitting phase can be calculated as $log_2(N/s_{max})/d_{max}$. Therefore, the splitting of N Bloom filters into a set of *mini-blocks* is of $O(N \cdot log_2(N/s_{max})/d_{max})$.

In the second phase of our protocol, merging *mini-clusters* requires the processing of $|\mathbf{C}|$ merged clusters. The computation of the centroid for all *mini-clusters* is of $O(s^2_{max} \cdot |\mathbf{C}|)$ complexity. Merging *mini-clusters* using the SCC approach requires a total computation of $O(ms_{max}/s_{max} \cdot |\mathbf{C}|)$, where at each iteration ms_{max}/s_{max} clusters are merged. At each iteration in the HCC approach the two most similar *mini-clusters* are merged which requires a total of $O(|\mathbf{C}|^2)$ computations.

The parties only need to communicate with each other to perform the secure summation protocol in the phases of multi-bit splitting and the merging of *mini-clusters*, with each message of length m and $|\mathbf{C}|$, respectively. By assuming each party directly connects to all other parties, the P parties require P messages to be sent in each iteration. Therefore the entire protocol has a communication complexity of $O(m \cdot P \cdot log_2(N/s_{max})/d_{max} + m \cdot |C|)$ for P parties.

Quality: We analyze the quality of our protocol in terms of effectiveness and efficiency [19]. The SCC approach merges *mini-clusters* greedily which can generate clusters with sizes larger than ms_{max}. This results in the SCC approach to have higher effectiveness and lower efficiency compared to the HCC approach. Both the SCC and HCC approaches retrieve more similar records compared to a previous approach [13], as the similarity between *mini-clusters* is used to merge the clusters up-to ms_{max}.

For $|\mathbf{C}|$ merged clusters, with each containing ms_{max} records, the number of candidate record sets generated for each party is $ms_{max} \cdot |\mathbf{C}|$. The parameter ms_{max} limits the size of the clusters generated by one of the clustering algorithms, which indirectly determines the number of merged clusters generated by the protocol. In the worst case scenario a merged cluster can be of size $2(ms_{max} - 1)$ if the two *mini-clusters* merged are each of size $ms_{max} - 1$. A suitable value for ms_{max} therefore needs to be set by considering factors such as the dataset size and the number of parties, such that both high effectiveness and high efficiency are achieved while guaranteeing sufficient privacy as well.

Privacy: We assume that each party follows the honest-but-curious adversary model [18]. All parties participate in a secure summation protocol for exchanging of ratio values of Bloom filters with other parties. During these summations, each party computes the sum of its ratio values but neither of the parties is capable of deducing anything about the other parties' private inputs [5].

Our protocol performs a generalization strategy on clusters that makes reidentification not possible. The parameter ms_{max} is used to guarantee that every resulting cluster contains at least ms_{max} records. This ensures all clusters that are generated have the same minimum number of records, which guarantees k-anonymous mappings ($k = ms_{max}$) privacy [8,19]. The merging of *mini-blocks* makes the protocol more secure and harder for dictionary and frequency attacks [18]. A higher value for ms_{max} provides stronger privacy guarantees but requires more computations as more candidate record sets will be generated.

6 Experimental Evaluation

We evaluated our protocol using the North Carolina Voter Registration (NCVR) database[1]. We based our experiments on the datasets used in and provided by [13], which contain from $5,000$ to $1,000,000$ records for 3, 5, 7 and 10 parties. In each of these sub-sets, 50% of records were matches. Some of these datasets included corrupted records which allowed us to evaluate how our approach works with 'dirty' data. The corruption levels were set to 20% and 40%.

We implemented a prototype of our protocol using Python (version 2.7.3). All experiments were run on a server with 64-bit Intel Xeon (2.4 GHz) CPUs, with 128 GBytes of main memory and Ubuntu 14.04. The programs and test datasets are available from the authors. We used four attributes commonly used for record linkage as QIDs: Given name, Surname, Suburb (town) name, and Postcode. We set the Bloom filter parameters as $m = 1000$ bits, $k = 30$ hash functions, and $q = 2$ by following earlier Bloom filter work in PPRL [13,14]. For comparative evaluation we used the single-bit tree (SBT) multi-party PPRL blocking approach by Ranbaduge et al. [13] as to our knowledge there are no other blocking approaches for multi-party PPRL available. We also used a phonetic based blocking approach (PHO) as a baseline [8,17] to comparatively evaluate the level of privacy. We named our multi-bit splitting, standard canopy clustering, and hierarchical canopy clustering as MBS, SCC, and HCC, respectively.

[1] Available from: `ftp://alt.ncsbe.gov/data/`

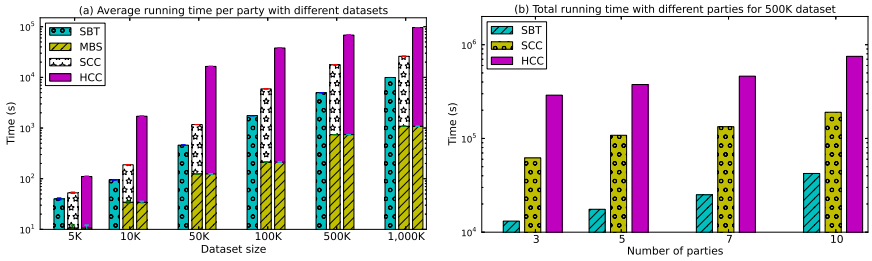


Fig. 2. (a) Average blocking runtime per party for different dataset sizes and (b) total blocking runtime for the $500K$ dataset with different number of parties.

In the PHO approach we used Soundex [4] as encoding function for all QIDs except Postcode where the first three digits of the value were used as the blocking key. Based on a set of parameter evaluation experiments we set the MBS parameters $d_{max} = 3$, $bs_t = 0.1$, $s_{max} = 50$ and $s_{min} = s_{max}/2$. We set the SCC and HCC parameters of s_t, s_l, and ms_{max} to 0.9, 0.8, and 500, respectively, as these values gave us the minimum overlap between clusters.

We measured the average total runtime for the protocol to evaluate the complexity of blocking. The reduction ratio (RR), which is the fraction of record sets that are removed by a blocking technique, and pairs completeness (PC), which is the fraction of true matching record sets that are included in the candidate record sets generated by a blocking technique, were used to evaluate the blocking quality. These are standard measures to assess indexing in record linkage [3].

Fig. 2 illustrates the scalability of our approach in terms of the average time required with different dataset sizes and number of parties. As expected the MBS-SCC approach requires less runtime than the MBS-HCC approach but both show a linear scalability with the size of the datasets and number of parties.

Fig. 3(a) illustrates that RR remains close to 1 for all dataset sizes and for different number of parties. This shows our approach significantly reduces the total number of candidate record sets that need to be compared. Fig. 3(b) to (d) illustrate the PC of our approach with different dataset sizes and corruption levels, indicating that our approach can provide significantly better blocking quality than the earlier proposed SBT approach [13].

To evaluate the privacy of our approach we use the measure *probability of suspicion* (P_s) [18], which is defined for a value in an encoded dataset as $1/n_g$, where n_g is the number of values in a global dataset ($\mathbf{G}$) that match with the corresponding value in the encoded dataset $\mathbf{D}$. As shown in Fig. 4(a), the MBS-SCC and MBS-HCC approaches both provide significantly better privacy compared to the PHO approach which has a maximum P_s of 1 (under the worst case assumption of the global dataset $\mathbf{G}$ being equal to the linkage dataset i.e. $\mathbf{G} \equiv \mathbf{D}$). By increasing the parameter ms_{max} stronger privacy can be guaranteed in our approach. Fig. 4(b) shows that the PHO approach [8,17] creates a large number of blocks of size 1 which makes this approach not suitable for PPRL.

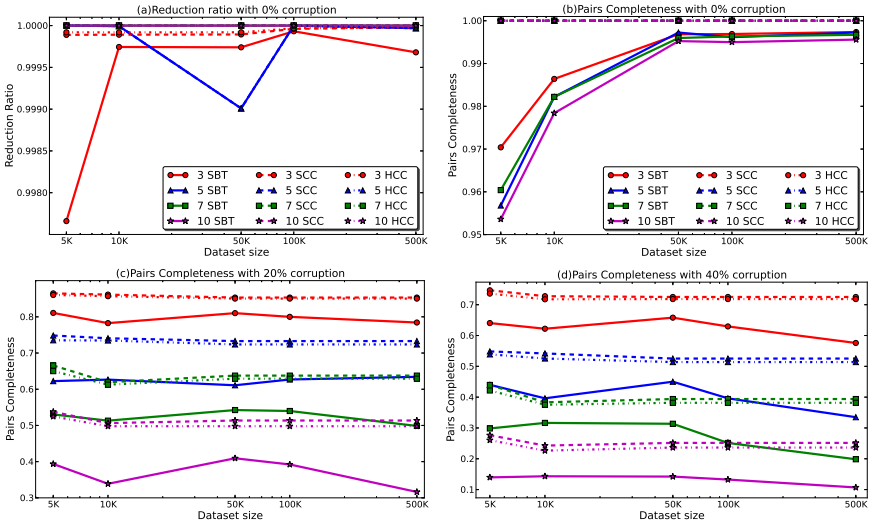


Fig. 3. (a) Reduction ratio with 0% corruption and (b) to (d) pairs completeness with 0%, 20%, and 40% corruption for different dataset sizes. Note the different y-axis scales.

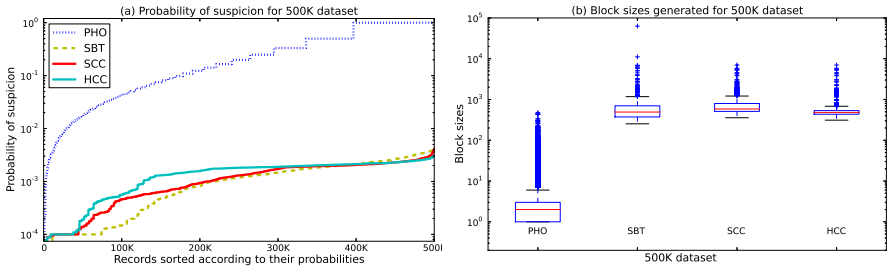


Fig. 4. (a) Probability of suspicion (P_s) and (b) block sizes generated by the different approaches using the $500K$ dataset.

According to Fig. 4(b), the MBS-HCC approach provides clusters within the acceptable size limit of ms_{max} which results in better block structures compared to the SBT and MBS-SCC approaches. This illustrates that our novel MBS-HCC technique provides better privacy than the other compared approaches while achieving higher results for both RR and PC.

7 Conclusion

We proposed a novel blocking protocol for multi-party PPRL based on multi-bit Bloom filter splitting and canopy clustering. We suggested a novel agglomerative hierarchical canopy clustering algorithm which generates canopies (blocks) within a specific size range. We demonstrated the efficiency and effectiveness of

our approach on datasets containing up-to one million records. The evaluation results indicated that our approach is scalable with both the size of the datasets and the number of parties. Our approach outperforms a previous multi-party private blocking and a phonetic based indexing approach in terms of blocking quality and privacy. A limitation in our approach is the assumption of the semi-honest adversary model. We plan to extend our protocol to adversary models that are applicable for malicious parties and evaluate the privacy over other attack methods applicable to PPRL [18]. We will also investigate the parallelization of our approach to improve its performance.

References

1. Al-Lawati, A., Lee, D., McDaniel, P.: Blocking-aware private record linkage. In: ACM IQIS. pp. 59–68. Baltimore (2005)
2. Bloom, B.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7), 422–426 (1970)
3. Christen, P.: Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer (2012)
4. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE TKDE 24(9), 1537–1555 (2012)
5. Clifton, C., Kantarcioglu, M., Vaidya, J., Lin, X., Zhu, M.: Tools for privacy preserving distributed data mining. SIGKDD Explorations 4(2), 28–34 (2002)
6. Cohen, W.W., Richman, J.: Learning to match and cluster large high-dimensional data sets for data integration. In: ACM SIGKDD. pp. 475–480. Edmonton (2002)
7. Durham, E.: A framework for accurate, efficient private record linkage. Ph.D. thesis, Faculty of the Graduate School of Vanderbilt University, Nashville, TN (2012)
8. Karakasidis, A., Verykios, V.: Secure blocking+secure matching = secure record linkage. Journal of Computing Science and Engineering 5, 223–235 (2011)
9. Kristensen, T.G., Nielsen, J., Pedersen, C.N.: A tree-based method for the rapid screening of chemical fingerprints. Algo. for Molecular Biology 5(1), 9 (2010)
10. Kuzu, M., Kantarcioglu, M., Inan, A., Bertino, E., Durham, E., Malin, B.: Efficient privacy-aware record integration. In: ACM EDBT. Genoa, Italy (2013)
11. Lai, P., Yiu, S., Chow, K., Chong, C., Hui, L.: An efficient Bloom filter based solution for multiparty private matching. In: SAM. Las Vegas (2006)
12. McCallum, A., Nigam, K., Ungar, L.H.: Efficient clustering of high-dimensional data sets with application to reference matching. In: ACM SIGKDD. Boston (2000)
13. Ranbaduge, T., Vatsalan, D., Christen, P.: Tree based scalable indexing for multi-party privacy-preserving record linkage. In: AusDM, CRPIT 158. Brisbane (2014)
14. Schnell, R., Bachteler, T., Reiher, J.: Privacy-preserving record linkage using Bloom filters. BMC Medical Informatics and Decision Making 9(1) (2009)
15. Schnell, R.: Privacy-preserving record linkage and privacy-preserving blocking for large files with cryptographic keys using multibit trees. In: JSM. Montreal (2013)
16. Vatsalan, D., Christen, P., Verykios, V.: A taxonomy of privacy-preserving record linkage techniques. JIS 38(6), 946–969 (2013)
17. Vatsalan, D., Christen, P.: Scalable privacy-preserving record linkage for multiple databases. In: ACM CIKM. pp. 1795–1798. Shanghai (2014)
18. Vatsalan, D., Christen, P., O'Keefe, C.M., Verykios, V.: An evaluation framework for privacy-preserving record linkage. JPC 6(1), 35–75 (2014)
19. Vatsalan, D., Christen, P., Verykios, V.: Efficient two-party private blocking based on sorted nearest neighborhood clustering. In: ACM CIKM. San Francisco (2013)