A Scalable and Efficient Subgroup Blocking Scheme for Multidatabase Record Linkage [⋆]

Thilina Ranbaduge[1], Dinusha Vatsalan[1,2], and Peter Christen[1]

[1] Research School of Computer Science, The Australian National University
Canberra ACT, Australia. Contact: thilina.ranbaduge@anu.edu.au
[2] Data61, CSIRO, Eveleigh, NSW, Australia.

Abstract. Record linkage is a commonly used task in data integration to facilitate the identification of matching records that refer to the same entity from different databases. The scalability of multidatabase record linkage (MDRL) is significantly challenged with the increase of both the sizes and the number of databases that are to be linked. Identifying matching records across subgroups of databases is an important aspect in MDRL that has not been addressed so far. We propose a scalable subgroup blocking approach for MDRL that uses an efficient search over a graph structure to identify similar blocks of records that need to be compared across subgroups of multiple databases. We provide an analysis of our technique in terms of complexity and blocking quality. We conduct an empirical study on large real-world datasets that shows our approach is scalable with the size of subgroups and the number of databases, and outperforms an existing state-of-the-art blocking technique for MDRL.

Keywords: Entity resolution; Apriori; Depth-first; P-partite graph; Cliques

1 Introduction

Many organisations, including government agencies, businesses, and research centres, collect vast quantities of data on a daily basis [3]. To improve the efficiency and effectiveness of decision making, organisations increasingly require data from different databases to be integrated. Multidatabase record linkage (MDRL) is the process of identifying records that match (i.e. correspond to the same entities) across multiple databases [4]. The process of linking records across different databases is also known as 'data linkage' or 'entity resolution' [3].

A real-world example of MDRL would be a health surveillance system that continuously links data from hospitals and pharmacies. The data collected from these sources can facilitate the investigation of geographical and temporal effects of diseases, or adverse drug reactions in certain patient groups [2]. Such analyses require the linkage across subgroups of hospital and pharmacy databases

[⋆] This work was funded by the Australian Research Council under Discovery Projects DP130101801 and DP160101934. The authors would also like to thank Vassilios Verykios for his valuable feedback.

collected at different locations since the linkage across all databases would not be sufficient to identify subsets of matching records such as cancer patients who visited a certain number of hospitals in a country, but not all hospitals.

In a MDRL context, potentially each record from one database needs to be compared with all records in all other databases to determine if a set of records corresponds to the same entity or not [15]. This becomes computationally expensive as the number of record pair comparisons grows exponentially with the number of databases to be linked [12]. To overcome this issue, blocking is generally applied in the linkage process [10]. Blocking reduces the record comparison space by grouping similar records, that likely correspond to true matches, based on the values of a set of attributes into the same block, while inserting records that likely correspond to non-matches into different blocks.

To identify subsets of matching records in multiple databases, it must be possible to link records from subgroups of databases. Though various blocking techniques have been developed for MDRL [12,15], these techniques cannot identify similar blocks across subgroups of databases because of two reasons. (1) Existing blocking techniques for MDRL are only capable of generating candidate blocks across all the databases, or for subgroups of a specific size [12], and (2) the application of a blocking technique multiple times for linking subgroups of different sizes is computationally infeasible due to the large number of potential subgroup combinations that need to be considered. This makes subgroup linkage for MDRL currently not scalable with an increasing number of databases.

We propose a subgroup blocking approach for MDRL which can efficiently identify blocks of records within a user specific range of subgroup sizes. Assuming d databases to be linked, we introduce two parameters, g_α and g_β, with $2 \leq g_\alpha \leq g_\beta \leq d$, to specify the minimum and maximum number of databases, respectively, that are to be included in a subgroup. Our approach accepts as input the sets of blocks generated from the databases that are to be linked, and it generates a set of candidate block tuples (CBTs) for each subgroup size from g_α to g_β. The records across the blocks in each CBT can then be compared in more detail [3].

To generate CBTs that need to be compared across subgroups of different sizes, we first arrange the sets of blocks from different databases to be linked into a graph structure $\mathbf{G}$. We then introduce two constraints based algorithms for the generation of CBTs by traversing over $\mathbf{G}$. Additionally, our approach allows for subgroups where some of the databases are fixed such that these databases must appear in every subgroup combination that is generated by our approach.

Contributions: We propose (1) a scalable subgroup blocking approach for MDRL that can be used under different real-world blocking scenarios, and (2) two constraint based graph traversal algorithms to generate candidate block tuples for subgroups of different sizes. (3) We analyse our subgroup blocking approach in terms of complexity and blocking quality, and (4) empirically evaluate the approach using large real-world databases with millions of records to validate its efficiency and effectiveness under different subgroup blocking scenarios. The results show that our subgroup blocking approach outperforms a state-of-the-art MDRL blocking approach [12] in terms of efficiency with no loss in effectiveness.

2 Related Work

Papadakis et al. [10] recently provided a survey of blocking techniques that have been proposed for record linkage. Most of these techniques are limited to linking two databases, and only few techniques have been developed for MDRL. Sadinle and Fienberg [15] proposed a probabilistic technique for linking multiple databases by extending the seminal work of Fellegi and Sunter [5]. This technique uses standard blocking [3] to group records into blocks, however it can only be used to match records across all the databases that are being linked.

Kong et al. [9] recently proposed an unsupervised technique to link records from multiple heterogeneous databases. This approach uses locality sensitive hashing (LSH) [7] to block each database to improve efficiency when generating candidate record tuples. The authors then adapted [5] to calculate the likelihood of a candidate record tuple being a match or a non-match based on several attributes. This approach, however, does not scale with the number of databases due to the large number of probability calculations required for each record tuple to be compared, and it cannot perform subgroup matching across databases.

Ranbaduge et al. [12] proposed a distributed blocking technique for privacy-preserving MDRL. This approach allows each owner to block its database independently by conducting a local clustering over their database to generate blocks. These blocks are then hashed using LSH [7] to identify the blocks that need to be compared across databases. While this approach can identify the blocks that need to be compared for a single subgroup size, it has to be run repeatedly for subgroups of different sizes, making the approach neither efficient nor effective in terms of identifying subgroup block tuples of different sizes.

In contrast to these existing multidatabase blocking approaches, our approach generates in one single run all candidate block tuples for subgroups of different sizes. We also allow the user to specify the minimum and maximum size of the subgroups, g_α and g_β, that are to be generated. The approach efficiently generates the most similar candidate block tuples by applying a constraints based pruning technique over a graph structure that is created based on the generated blocks. Fu et al. [6] proposed a graph based approach to match households across time in historical census data, while a MDRL meta-blocking approach recently proposed by Ranbaduge et al. [11] also uses a graph structure to remove redundant record pair comparisons. However, both these approaches are not capable of performing subgroup blocking across databases.

3 Subgroup Blocking Process

Let us assume d (≥ 2) de-duplicated databases are to be linked. We use the notation $\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C$, and so on to represent different databases, while A_i, B_j, C_k and so on to represent the blocks generated for each corresponding database. The aim of our approach is to generate candidate block tuples (CBTs) for subgroup combinations across these d databases. A CBT is a tuple of blocks which consists of a maximum of one block per database, and blocks from at least two databases. As illustrated in Fig. 1, the approach consists of three steps:

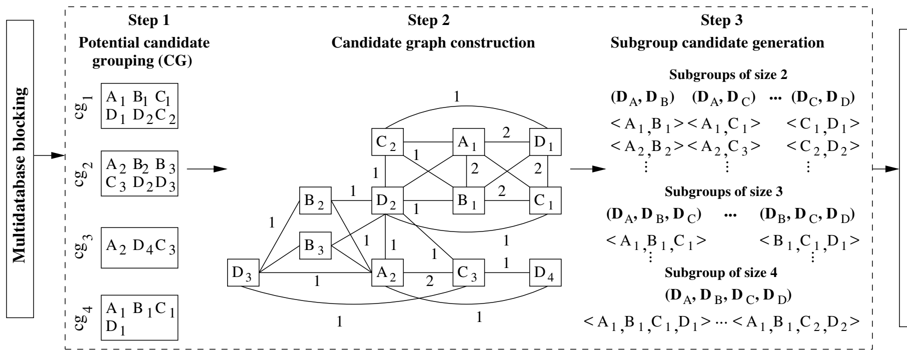
Multidatabase blocking
Step 1
Potential candidate grouping (CG)
Step 2
Candidate graph construction
Step 3
Subgroup candidate generation
Subgroups of size 2
Subgroups of size 3
Subgroup of size 4
Comparison and classification

Fig. 1. Overview of our subgroup blocking approach with its three main steps. First, in step 1 the set of candidate groups (**CG**) is generated. A graph is constructed in step 2, where each block description pair (BDP) in a candidate group $cg \in \mathbf{CG}$ becomes a vertex. Two vertices are connected by an edge if they occur in the same cg. Each edge is assigned with a weight which in this example is the number of $cg \in \mathbf{CG}$ that contain a given pair of BDPs. In step 3, candidate block tuples (CBTs) are identified for subgroups that need to be compared. In this example, (g_α, g_β) is set to $(2, 4)$.

1. *Potential Candidate Grouping*: As we discuss in Sect. 3.1, the block description pairs (BDPs) of each database are grouped into a set of candidate groups (**CG**) based on the similarities between their block representatives.
2. *Candidate Graph Generation*: A candidate graph **G** is constructed based on the generated **CG**. This requires an iteration over each group in the **CG** to create vertices and edges in **G**. Then weights (w) are calculated for each edge in **G**, as we discuss in detail in Sect. 3.2.
3. *Subgroup Candidate Generation*: CBTs are generated for each subgroup combination using **G**. As we describe in Sect. 3.3, a *weight threshold* (w_t) is used to remove low weighted edges ($w < w_t$) in **G** to ensure the block pairs that have a low similarity are excluded from the CBT generation process.

The two user defined parameters, g_α and g_β, with $2 \leq g_\alpha \leq g_\beta \leq d$, specify the minimum and maximum number of databases that are to be included into subgroup combinations, respectively. As an optional parameter, the user can also define the set of fixed databases (F) that must be included in every subgroup that is generated. Based on (g_α, g_β), our approach is capable of generating CBTs for the following three scenarios with d databases in a MDRL context:

1. $g_\alpha = g_\beta = d$: This setting gives the linkage between all databases only, i.e. only sets of blocks that are to be compared across all databases are generated.
2. $2 \leq g_\alpha \leq g_\beta < d$: This setting gives all possible linkages for subgroups with at least g_α to at most g_β databases, i.e. CBTs are generated for every subgroup combination between sizes g_α and g_β across all databases.
3. $F = \{\mathbf{D}_x, \mathbf{D}_y, \cdots, \mathbf{D}_z\}$, $|F| \leq g_\alpha \leq g_\beta < d$: This setting generates CBTs for subgroups with size at least g_α to a maximum size of g_β out of d databases, where databases $\mathbf{D}_x$, $\mathbf{D}_y$, $\cdots$, $\mathbf{D}_z$ must appear in every subgroup.

As shown in Fig. 1, for example, if $F = \{\mathbf{D}_A\}$ and $(g_\alpha, g_\beta) = (2, 3)$ then the subgroup combinations that will be considered in our approach are, for subgroups of size 2: $(\mathbf{D}_A, \mathbf{D}_B)$, $(\mathbf{D}_A, \mathbf{D}_C)$, and $(\mathbf{D}_A, \mathbf{D}_D)$; and for subgroups of size 3: $(\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_C)$, $(\mathbf{D}_A, \mathbf{D}_B, \mathbf{D}_D)$, and $(\mathbf{D}_A, \mathbf{D}_C, \mathbf{D}_D)$.

To perform the linkage across subgroups of databases, as a prerequisite, first each database needs to be blocked. Any blocking technique [3,10] can be used to generate the set of blocks for each database, as long as the same technique is used on all d databases. We assume each database is blocked independently, as this provides flexibility and efficiency over the block generation process [12].

After blocking is completed, a *block description pair* (BDP) is generated for each block from each database. Each BDP (b, b_{rep}) consists of a block identifier (b) and a block representative (b_{rep}). A b_{rep} can be generated in different forms, such as a Min-Hash signature [7], a Bloom filter [12], or a phonetic encoding [3], as long as the same technique is used on all databases to generate the b_{rep}s for all blocks. The set of generated BDPs of each database is then added to an overall set of BDPs, $\mathbf{B}$, which is used as input in our MDRL subgroup blocking approach. Hence, we assume this local multidatabase blocking phase to be a *black box*. We next describe the three steps of our approach in more detail.

3.1 Potential Candidate Grouping

In step 1 of our approach we identify the potential candidates among the sets of blocks of each database by grouping the BDPs in $\mathbf{B}$ into a set of candidate groups ($\mathbf{CG}$). The grouping technique is based on the similarities calculated between the corresponding b_{rep}s that have been generated. For example, a Jaccard based LSH [7] technique can be used with b_{rep}s based on Min-Hash signatures, where blocks that hash to the same bucket become candidates and each bucket is considered as a candidate group (cg) that is added to the overall set $\mathbf{CG}$.

Each $cg \in \mathbf{CG}$ helps to identify the candidate blocks that need to be considered for comparison. If a pair of blocks appears in multiple cgs it is more likely that these blocks are more similar. As in Fig. 1 (step 1), for example, the pair (A_1,C_1) is more likely to be similar compared to (A_1,C_2), because (A_1,C_1) occurs in two cgs while (A_1, C_2) only occurs in one. Hence, this grouping reduces the overall number of block comparisons since only the blocks in a cg will be compared next in the linkage process. Reducing the number of block comparisons therefore reduces comparisons between records that are unlikely to be similar.

3.2 Candidate Graph Construction

In step 2 we construct the candidate graph $\mathbf{G} = (V, E)$ from $\mathbf{CG}$, where $\mathbf{G}$ is an undirected d-partite graph [1]. The construction of $\mathbf{G}$ requires a pass over the $\mathbf{CG}$ where each BDP that appears in a $cg \in \mathbf{CG}$ becomes a vertex $v \in V$ in $\mathbf{G}$.

An edge $e_{i,j} \in E$ is created between two vertices, v_i and v_j, if their corresponding BDPs (BDP_i and BDP_j) appear in the same cg, with the constraint that edges are created only between BDPs from different databases. As shown in Fig. 1 (step 2), a weight $w_{i,j}$ is calculated for each $e_{i,j}$. The weight $w_{i,j}$ of

Algorithm 1: Apriori Candidate Generation

```
Input:
- G:      Undirected candidate graph
- F:      Set of fixed databases
- d:      Number of databases
- w_t:    Weight threshold
- g_α, g_β: Minimum and maximum subgroup size

Output:
- SBT: Inverted index of subgroup block tuples
 1: $\mathbf{SBT} \leftarrow \{\}$, $K \leftarrow \{\}$
 2: $k \leftarrow 2$
 3: if $k < g_\alpha$ then:
 4:     $K$.add(getCombinations($k, g_\alpha, d, F$))
 5: $K$.add(getCombinations($g_\alpha, g_\beta, d, F$))
 6: foreach $S \in K[2]$ do:
 7:     $E \leftarrow$ identifyEdges($\mathbf{G}, S, w_t$))
 8:     $C_2$.add($E$)
 9: $\mathbf{SBT}[2] \leftarrow C_2$
10: $k \leftarrow 3$
11: while $k \leq g_\beta$ and $C_{k-1} \neq \emptyset$ do:
12:     $C_k \leftarrow$ getCliques($\mathbf{G}, K[k], w_t, C_{k-1}$)
13:     if $g_\alpha \leq k$ then:
14:         $\mathbf{SBT}[k] \leftarrow C_k$
15:     $k \leftarrow k + 1$
16: return SBT
```

Algorithm 2: Depth-first Candidate Generation

```
Input:
- G:      Undirected candidate graph
- F:      Set of fixed databases
- d:      Number of databases
- w_t:    Weight threshold
- g_α, g_β: Minimum and maximum subgroup size
- B:      Set of block description pairs
Output:
- SBT: Inverted index of subgroup block tuples
 1: $\mathbf{SBT} \leftarrow \{\}$, $K \leftarrow \{\}$
 2: $K$.add(getCombinations($g_\alpha, g_\beta, d, F$))
 3: foreach $k \in K.keys()$ do:
 4:     foreach $S \in K[k]$ do:
 5:         $\mathbf{SBT}[k]$.add(genCandidates($\mathbf{G}, S, w_t, \mathbf{B}$))
 6: return SBT
Function genCandidates($\mathbf{G}, S, w_t, \mathbf{B}$):
 7: if $|S| = 2$ then:
 8:     return identifyEdges($\mathbf{G}, S, w_t$)
 9: else:
10:     $\mathbf{C} \leftarrow []$
11:     $\mathbf{D}, L_{BDP} \leftarrow$ getDBWithMinBlocks($\mathbf{B}, S$)
12:     foreach $(b_i, b_{rep}) \in L_{BDP}$ do:
13:         $L_v \leftarrow$ getNeighbours($\mathbf{G}, \{S - \mathbf{D}\}, w_t, b_i$)
14:         $C \leftarrow$ genCandidates($\mathbf{G}, \{S - \mathbf{D}\}, w_t, L_v$)
15:         $\mathbf{C}$.add(updateCandidates($C, b_i$))
16:     return C
```

edge (BDP_i, BDP_j) can be computed in different ways based on the generated b_{rep}s, such as the similarity between the corresponding block representatives b^i_{rep} and b^j_{rep}, or the normalised cardinality of (BDP_i, BDP_j) which is $w_{i,j} = |\{cg : \forall_{cg \in \mathbf{CG}}\ (BDP_i, BDP_j) \in cg\}|\ /\ |\mathbf{CG}|$, where $|\cdot|$ represents the cardinality of a given set. These weights are used in the next step to generate the CBTs.

3.3 Subgroup Candidate Generation

In step 3 of our approach, CBTs are generated for each subgroup combination required. For a given subgroup of size g_α a CBT contains a maximum of one block identifier per database, and identifiers from at least g_α databases. Each CBT is a *clique* $c \in \mathbf{G}$, where each $c \subseteq V$, such that all pairs of vertices in c must be connected by an edge, i.e., $\forall v_i, v_j \in c :\ (v_i, v_j) \in E$. For generating CBTs we propose two candidate generation algorithms, as detailed below:

- *Apriori based Candidate Generation*: CBTs for subgroup sizes from g_α to g_β are generated using an Apriori based breadth-first search over $\mathbf{G}$ [1,8].
- *Depth-first based Candidate Generation*: CBTs for subgroup sizes from g_α to g_β are generated using a depth-first traversal through graph $\mathbf{G}$ [1,14].

Apriori based Candidate Generation (ACG): The proposed ACG approach is outlined in Algorithm 1. In lines 3 to 5, the function *getCombinations()* generates all the required subgroup combinations from g_α to g_β to be considered in the candidate generation process which are then added to an inverted index K using the subgroup sizes as keys. For example, with $(g_\alpha, g_\beta) = (2, 3)$ of a linkage between databases $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}_C$, $K[2]$ contains the list of subgroup combinations ($\mathbf{D}_A$,$\mathbf{D}_B$), ($\mathbf{D}_A$,$\mathbf{D}_C$), and ($\mathbf{D}_B$,$\mathbf{D}_C$), while $K[3]$ contains the subgroup combination ($\mathbf{D}_A$, $\mathbf{D}_B$, $\mathbf{D}_C$). If $g_\alpha > 2$ (line 3) ACG needs to generate the set of subgroups of sizes 2 to g_α because an Apriori based iterative approach [1,8] is used to identify cliques (CBTs) of size k from the cliques of size $k-1$ that were identified in the previous iteration (starting from pairs, i.e. $k = 2$).

To control the number of CBTs generated for each subgroup, we use a constraint, named as *weight threshold* (w_t), on the weight $w_{i,j}$ of each $e_{i,j} \in E$, which specifies the minimum weight that each $e_{i,j}$ must have in order to be considered in the CBT generation. w_t helps to control the density of $\mathbf{G}$ by efficiently pruning block pairs that have a low similarity. In practice, different w_ts can be specified for different subgroup combinations depending on user requirements.

The function *identifyEdges()* generates the trivial cliques of size $k = 2$ for each subgroup S which are the set of edges $E \in \mathbf{G}$ that satisfy w_t (line 7). In line 12, the function *getCliques()* traverses through $\mathbf{G}$ to identify all cliques of size k that satisfy w_t. These cliques are then added to the set C_k. Following the Apriori principle [1,8], in lines 11 to 15, ACG continues until k reaches g_β or no cliques were generated in the previous iteration (i.e. $C_{k-1} = \emptyset$). These generated CBTs are added to an inverted index $\mathbf{SBT}$ using subgroup combinations as keys.

Depth-first based Candidate Generation (DCG): The proposed DCG approach uses an iterative deepening depth-first search algorithm [14], as detailed in Algorithm 2. DCG generates CBTs from size g_α to g_β by incrementally expanding the size of subgroups. DCG uses multi-branch recursion that allows $\mathbf{G}$ to be searched progressively for similar blocks from the corresponding databases of a subgroup combination until the required CBT size is reached.

Similar to ACG, DCG starts with generating subgroup combinations for all databases by using the function *getCombinations()*. These combinations are then added to K (line 2 of Algorithm 2). For each subgroup S in K the recursive function *genCandidates()* is called to generate the set of CBTs (in lines 3 to 5). Similar to Algorithm 1, the function *identifyEdges()* is used to get the set of edges from $\mathbf{G}$ for each subgroup of size $|S|$ that satisfy the threshold w_t (line 8).

For subgroup sizes greater than 2, the function *genCandidates()* first selects the database $\mathbf{D}$ with the minimum number of blocks using a function *getDB-WithMinBlocks()*. This function returns $\mathbf{D}$ and its list of BDPs L_{BDP} in $\mathbf{B}$. This selection minimises the number of recursive branches in the CBT generation (line 11). Next, for each (b_i, b_{rep}) in L_{BDP} the function *getNeighbours()* retrieves the neighbouring vertices for the remaining set of databases that connect with b_i (line 13). Those pairs of vertices that have an edge weight greater than or equal to w_t are added to the list of neighbouring vertices L_v.

For each b_i in L_{BDP} the function *genCandidates()* is called recursively with the list L_v, w_t, and the set of remaining databases as inputs (line 14). Each of these recursive calls returns a list of block tuples C, where each tuple in C is updated with the current processed b_i using the function *updateCandidates()* (line 15). This allows DCG to progressively generate CBTs until they reach the required size k. These CBTs are finally added to $\mathbf{SBT}$ (line 5).

4 Analysis of Subgroup Blocking

We analyse our approach in terms of complexity and blocking quality. We neither consider the block generation nor the comparison and classification techniques since they are outside the scope of our approach. Let us assume d databases are to be linked and the blocks of all these databases are added into the set $\mathbf{B}$.

Complexity: Though the generation of the set of candidate groups ($\mathbf{CG}$) in step 1 depends on the grouping technique used, it would require a complexity of $O(|\mathbf{B}|)$. We assume each candidate group $cg \in \mathbf{CG}$ contains d BDPs from different databases. In step 2, $\mathbf{CG}$ is used to construct the graph $\mathbf{G}$. This requires to iterate over each $cg \in \mathbf{CG}$ to add a vertex v to $\mathbf{G}$, and to create edges between vertices if they share the same cg. A cg with d BDPs generates $d(d-1)/2$ edges in $\mathbf{G}$. Hence, the construction of $\mathbf{G}$ has a complexity of $O(|\mathbf{CG}| \cdot d^2)$.

In ACG the CBT generation would require a complexity of $O(n^{g_\alpha})$ [8] if each of the g_α databases generates $n = |\mathbf{B}|/d$ BDPs. In line 10 of Algorithm 1, the generation of cliques of size k depends on the number of $(k-1)$ size cliques, C_{k-1}, generated previously. Hence, the complexity of ACG is $O(|E| + \sum_{g_\alpha=3}^{g_\beta} \binom{g_\alpha}{g_\beta} |C_{g_\alpha-1}|^2)$, where $|E|$ is the number of edges (pairs of blocks) in $\mathbf{G}$. This becomes computationally infeasible when n, and g_β are increasing.

Based on the g_α and g_β settings, DCG requires to generate CBTs for $n_c = \sum_{g_\alpha=2}^{g_\beta} \binom{d}{g_\alpha}$ subgroup combinations. For each combination, DCG uses a multi-branch recursion to generate CBTs. In the function *genCandidates()* of Algorithm 2, at each recursion a database $\mathbf{D}$ out of g_α databases with the minimum number of vertices is selected. Without loss of generality, let us assume the number of BDPs selected for $\mathbf{D}$ is m. m defines the maximum recursion branch factor of $\mathbf{D}$. The total number of vertex traversals in $\mathbf{G}$ for a given subgroup combination of size g_α can be calculated as $g_\alpha \cdot m + (g_\alpha - 1) \cdot m^2 + \cdots + 2 \cdot m^{g_\alpha - 1} + m^{g_\alpha}$ which is $\sum_{i=1}^{g_\alpha} (g_\alpha + 1 - i) \cdot m^i$. Hence, for n_c subgroup combinations DCG has a complexity of $O(\sum_{g_\alpha=2}^{g_\beta} \sum_{i=1}^{g_\alpha} \binom{d}{g_\alpha} (g_\alpha + 1 - i) \cdot m^i)$. However, DCG would require a complexity of $O(n_c \cdot (|\mathbf{B}|/d)^{g_\alpha})$ if $\mathbf{G}$ is a complete graph with $\forall e \in E : \ e.w \geq w_t$.

Blocking Quality: In step 1 the effectiveness of the candidate grouping depends on the b_{rep}s and grouping technique used on those b_{rep}s. In step 2 the density of the graph $\mathbf{G}$ depends on the number of blocks in each $cg \in \mathbf{CG}$ from different databases to be linked. A large number of blocks in a cg will increase the effectiveness of CBT generation as more edges are created in $\mathbf{G}$.

In step 3 of our approach, the weight threshold w_t provides a trade-off between the quality and efficiency of the CBT generation process. The threshold w_t is used to prune edges (block pairs) with weights lower than w_t from the candidate generation process. A lower w_t will generate more cliques (CBTs) as more edges are considered in the CBT generation process for a given subgroup. This will increase the number of true matches as more block tuples are generated as cliques to be compared in the comparison and classification step, which will improve the effectiveness of the overall linkage. However, a lower w_t will potentially increase the overall runtime and space requirements of our approach because more edges are considered for a given subgroup combination.

Table 1. Datasets use in our experimental evaluation. 'Dataset Size (min-max)' is the minimum and maximum number of records in the databases of a dataset, and 'Avg. overlap' is the average percentage of records matched across the databases in a dataset.

Datasets	Number of databases (d)	Dataset size (min-max)	Avg. overlap	Provenance
NC-CLN	16	5,614,747 – 7,453,886	90%	Real
NC-DRT	16	72,903 – 1,308,796	20%	Real
NC-SYN	10	5,000 – 1,000,000	50%	Synthetic
UKCD	6	17,033 – 31,059	5,000 records	Real

5 Experiments and Discussion

For evaluation purposes we use two real-world datasets as outlined in Table 1. NC contains registration records of around 8 million voters from the US state of North Carolina (available from: `http://dl.ncsbe.gov/`). We use *given name*, *surname*, *city*, and *zipcode* as the blocking key attributes, as these are commonly used for record linkage [3,13].

We use 16 voter databases, collected at different points in time with two months interval between each pair of databases. The records in these databases can be grouped into three categories: (1) *exact matching*: those records (about the same person) that are exactly matching with each other, (2) *unique*: those records that are only appearing in one database, and (3) *updated*: those records where at least one attribute value has changed across two consecutive databases.

We use three different variations of the NC datasets, named as NC-CLN, NC-DRT, and NC-SYN. For NC-CLN we extracted *unique* and *exact matching* records from each database. Due to the skewness of *exact matching* records NC-CLN is only used to evaluate the scalability of our approach. For NC-DRT we extracted *unique* and *updated* records from each database. Since each update in an attribute value is considered as a modification in a record, NC-DRT is used to evaluate the blocking quality of our approach. To evaluate our approach with different levels of data quality, we use NC-SYN that contains 10 synthetic databases, as used in and provided by [12], which was created by extracting records from the original NC dataset. Some of these databases included corrupted records, where the corruption levels were set to 20% and 40% [12].

UKCD is another real dataset used in and provided by [6], consisting of census records collected from the years 1851 to 1901 in 10 year intervals for the town of Rawtenstall and surrounds in the United Kingdom. It contains approximately 150,000 records of 32,000 households with partial gold standard data (records manually linked by domain experts) for testing. Both NC and UKCD have been used for the evaluation of various other RL approaches [6,11,12] and we are not aware of any other available large real-world datasets that contain records from more than two databases that could be used to evaluate MDRL.

For comparison we use the state-of-the-art MDRL blocking technique proposed in [12] (named HDC for *Hashing based Distributed Clustering*) as this is the only existing technique we are aware of that can be used for subgroup block-

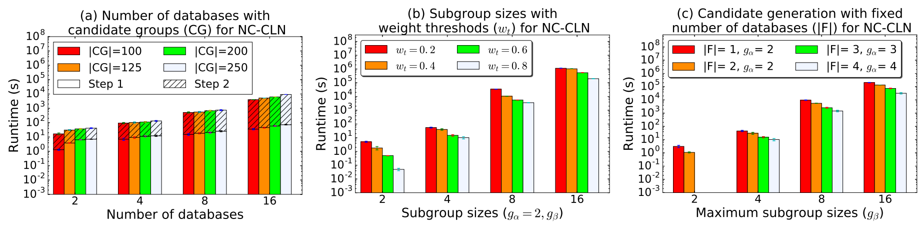

Fig. 2. The average runtime required with different (a) number of databases; (b) weight thresholds (w_t); and (c) number of fixed databases ($|F|$) for the NC-CLN datasets.

ing. For the prerequisites of our blocking approach, we use the same steps and parameter settings as used in HDC. Each database is blocked using a hierarchical clustering approach to generate an average of 100 to 1,000 blocks per database. Next, Min-Hash signatures are generated for each block as b_{rep}s. As in HDC, we hash these b_{rep}s into a set of buckets using locality sensitive hashing (LSH) [7] in step 1. Each bucket is added to the set $\mathbf{CG}$ as a candidate group. To measure the similarity between the b_{rep}s in step 2 we use the Jaccard coefficient [3].

We evaluated the complexity using runtime and memory, while the blocking quality was measured using pairs completeness (PC) and reduction ratio (RR) [3]. PC was calculated as the ratio of the number of matched records against the total number of true matched records across all databases. RR measures the reduction in the number of compared record pairs against the total number of record pairs. All experiments were conducted on a server with 64-bit Intel Xeon (2.4GHz) CPUs, 128 GBytes of memory, and Ubuntu 14.04. We implemented all approaches using Python (version 2.7), and to allow repeatability the programs and test datasets are available from the authors.

Discussion: As shown in Fig. 2 (a), the average runtime required for steps 1 and 2 of our approach increases linearly with the number of databases d. We noted that the average runtime also increases linearly with the number of candidate groups ($|\mathbf{CG}|$), which suggests that more edges are being generated in the graph $\mathbf{G}$. The average runtime decreases with an increase in the weight threshold (w_t) because the edges with lower similarity between their corresponding b_{rep}s are not considered in the CBT generation (see Fig. 2 (b)). However, the runtime increases linearly with the size of subgroups as more combinations are considered in the CBT generation while the runtime decreases when more databases are included in the set of fixed databases F as shown in Fig. 2 (c).

Figures 3 (a) and (d) show the total runtime increases exponentially as the number of subgroup combinations grows exponentially with d, while the runtime grows linearly with d for a given subgroup size. Similar to Fig. 2 the average runtime decreases with an increase in w_t as shown in Fig. 3 (b) and (e). Figures 3 (c) and (f) show DCG requires less runtime compared to HDC and ACG, which suggests that DCG is more efficient for CBT generation. However, ACG is still competitive with DCG if the number of blocks from each database remains small. We also measured the memory required for the CBT generation (not shown due

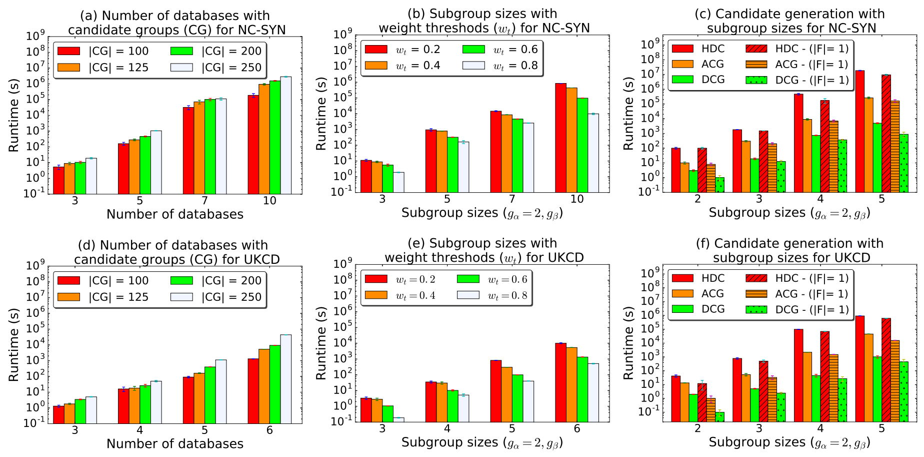

Fig. 3. Runtime results, where plots (a) to (c) and (d) to (f) show the results for the NC-SYN and UKCD datasets, respectively. Plots (a) and (d) show the total runtime with different number of databases, and (b) and (e) show the average runtime with different weight thresholds for different subgroup sizes. Plots (c) and (f) show the average runtime required for DCG compared with HDC and ACG.

to limited space) where in average DCG only uses below 10% of the total memory required by ACG and HDC. We were unable to conduct experiments for HDC and ACG with subgroup size larger than 5 due to their memory requirements.

Figures 4 (a) and 5 (a) show RR increases with w_t which suggests that less CBTs are generated for a given subgroup size. This results in PC to decrease as true matches are missed due to the lower number of CBT comparisons (see Fig. 4 (b)). However, a lower w_t value increases PC by generating more CBTs, which increases the overall runtime of our approach (see Fig. 2 and Fig. 3). Also, PC increases with $|\mathbf{CG}|$ which suggests that CBT generation becomes more fine grained as the graph $\mathbf{G}$ becomes more dense (see Fig. 4 (c) and Fig. 5 (b)). After step 3 of our approach, we applied the same ranking algorithm as used in HDC to compare ACG and DCG with HDC [12]. HDC ranks the CBTs for comparison according to an approximation of RR. As shown in Fig. 5 (c), we observed that ACG and DCG achieve the same PC as HDC which suggests that our approach can perform subgroup blocking more efficiently with no loss in effectiveness.

6 Conclusions and Future work

We proposed a subgroup blocking approach for multidatabase record linkage (MDRL) based on a graph structure that is used for generating candidate block tuples using cliques. The evaluation on real datasets showed that our approach is scalable with the size of subgroups and it outperforms an existing MDRL blocking approach in terms of subgroup blocking. In future we aim to adapt pattern growth methodologies [1] and parallelisation into our blocking approach.

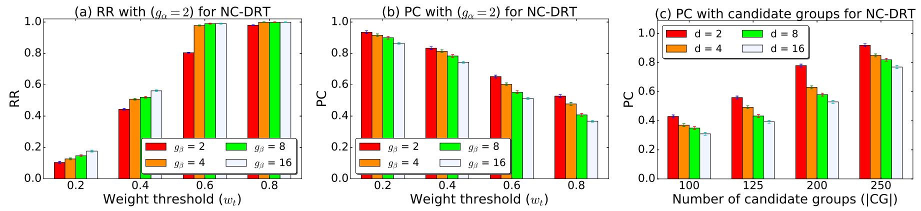

Fig. 4. (a) RR and (b) PC for different weight thresholds for different subgroup sizes, and (c) PC with different number of candidate groups (|**CG**|) for the NC-DRT datasets.

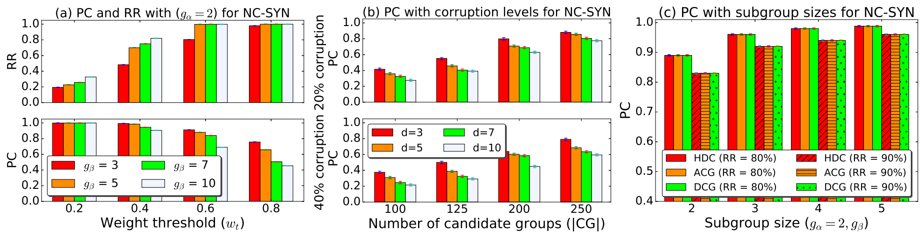

Fig. 5. (a) PC and RR for different weight thresholds (with 0% corruption), and PC with different (b) corruption levels and (c) subgroup sizes for the NC-SYN datasets.

References

1. Aggarwal, C., Wang, H.: Managing and Mining Graph Data. Springer (2010)
2. Boyd, J., Ferrante, A., O'Keefe, C., et al.: Data linkage infrastructure for cross-jurisdictional health-related research in australia. BMC Health Serv Res (2012)
3. Christen, P.: Data Matching. Springer (2012)
4. Elmagarmid, A., Ipeirotis, P., Verykios, V.: Duplicate record detection: A survey. IEEE TKDE (2007)
5. Fellegi, I., Sunter, A.: A theory for record linkage. JASA (1969)
6. Fu, Z., Christen, P., Zhou, J.: A graph matching method for historical census household linkage. In: PAKDD (2014)
7. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: Theory of Computing (1998)
8. Inokuchi, A., Washio, T., Motoda, H.: An Apriori-based algorithm for mining frequent substructures from graph data. In: PKDD (2000)
9. Kong, C., Gao, M., Xu, C., Qian, W., Zhou, A.: Entity matching across multiple heterogeneous data sources. In: ACM DASFAA (2016)
10. Papadakis, G., Svirsky, J., et al.: Comparative analysis of approximate blocking techniques for entity resolution. VLDB Endow. (2016)
11. Ranbaduge, T., Vatsalan, D., Christen, P.: Scalable block scheduling for efficient multi-database record linkage. In: IEEE ICDM (2016)
12. Ranbaduge, T., Vatsalan, D., Christen, P., Verykios, V.: Hashing-based distributed multi-party blocking for privacy-preserving record linkage. In: PAKDD (2016)
13. Randall, S., Ferrante, A., Boyd, J., Semmens, J.: The effect of data cleaning on record linkage quality. BMC Med Inform Decis Mak (2013)
14. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach (2009)
15. Sadinle, M., Fienberg, S.: A generalized Fellegi-Sunter framework for multiple record linkage with application to homicide record systems. JASA (2013)