A Clustering-Based Framework to Control Block Sizes for Entity Resolution

Jeffrey Fisher
Research School of Computer Science
Australian National University
Canberra ACT 0200
jeffrey.fisher@anu.edu.au

Peter Christen
Research School of Computer Science
Australian National University
Canberra ACT 0200
peter.christen@anu.edu.au

Qing Wang
Research School of Computer Science
Australian National University
Canberra ACT 0200
qing.wang@anu.edu.au

Erhard Rahm
Database Group
University of Leipzig Germany
rahm@informatik.uni-leipzig.de

ABSTRACT

Entity resolution (ER) is a common data cleaning task that involves determining which records from one or more data sets refer to the same real-world entities. Because a pairwise comparison of all records scales quadratically with the number of records in the data sets to be matched, it is common to use blocking or indexing techniques to reduce the number of comparisons required. These techniques split the data sets into blocks and only records within blocks are compared with each other. Most existing blocking techniques do not provide control over the size of the generated blocks, despite this control being important in many practical applications of ER, such as privacy-preserving record linkage and real-time ER. We propose two novel hierarchical clustering approaches which can generate blocks within a specified size range, and we present a penalty function which allows control of the trade-off between block quality and block size in the clustering process. We evaluate our techniques on three real-world data sets and compare them against three baseline approaches. The results show our proposed techniques perform well on the measures of pairs completeness and reduction ratio compared to the baseline approaches, while also satisfying the block size restrictions.

Categories and Subject Descriptors

H.2.8 [**Database management**]: Database applications—*Data mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*

Keywords

Blocking, indexing, data cleaning, record linkage.

KDD'15, August 11–14, 2015, Sydney, NSW, Australia

ACM 978-1-4503-3664-2/15/08..$15.00.
DOI: http://dx.doi.org/10.1145/2783258.2783396.

1. INTRODUCTION

Entity resolution (ER) is a common data cleaning and pre-processing task that aims to determine which records in one or more data sets refer to the same real-world entities [7]. It has numerous applications, including matching customer records following a corporate merger, detecting persons of interest for national security, or linking medical records from different health organisations. In many cases ER is performed in static mode where all matching decisions are conducted at once. However, it can also be performed in real-time, where the task involves finding the most similar record(s) to a given query record [18].

In both static and real-time ER, a key step is to compare the similarities of record pairs [2]. However, comparing all records in a data set, or between multiple data sets, scales quadratically with the number of records in the data sets and can be computationally infeasible if the data sets are large. As a result, blocking or indexing techniques are commonly used to limit the number of comparisons so that only record pairs that have a high likelihood of referring to the same real-world entity are compared [2]. This is done by dividing the data set(s) into (possibly overlapping) blocks and only performing comparisons between records in the same block.

In this paper we study the problem of how to control block sizes when generating blocks for ER. Our study is motivated by the observation that there are various application areas where maximum and minimum block sizes are important.

- One application area is real-time ER [18] where operational requirements mean that only a certain number of comparisons can take place within a specific (or limited) time-span (e.g. sub-second). Therefore, blocking is important to ensure that these comparisons are with the candidate records that most likely correspond to matches. In such cases, having a maximum block size ensures that operational requirements can be satisfied.
- In privacy-preserving record linkage [22], there may be privacy requirements on both the minimum and maximum block size. For example, to guarantee k-anonymous privacy [21] it is necessary that each block contains at least k records. If all blocks have a similar size this reduces the vulnerability of the ER process

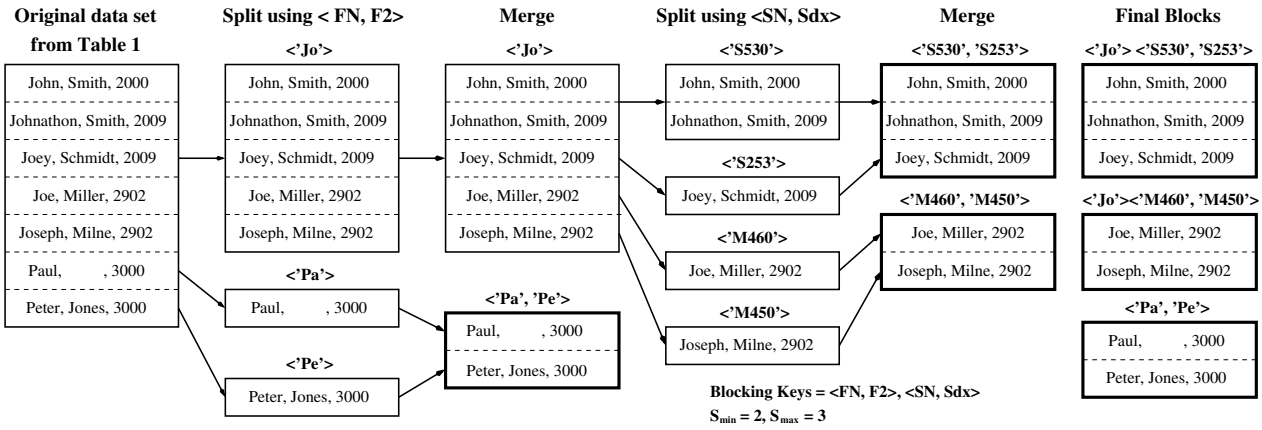

Figure 1: Example of algorithm flow using the data set from Table 1 with $s_{min} = 2$ and $s_{max} = 3$. The initial data set is split using the first blocking key $\langle FN$, F2$\rangle$ (the first two characters of the $FirstName$ attribute) and the small blocks are merged. The block that is still too large after the initial split is split again, this time using the second blocking key $\langle SN$, Sdx$\rangle$ (Soundex encoding of the $Surname$ attribute). The small blocks are again merged until we end up with three blocks in the specified size range (blocks shown with bold frames).

to frequency-based attacks [22]. In this situation it is important to produce blocks in the specified size range.

- Finally, if blocking is being used as a preliminary step for an ER approach that has poor scalability, restricting the maximum size of blocks is very important. Collective entity resolution techniques, such as those proposed by Bhattacharya and Getoor [1], Kalashnikov and Mehrotra [12] and Dong et al. [6], all give high match quality at the expense of scalability. Similarly, techniques such as Markov logic network based ER [20] have very poor scalability for networks above a certain size. Ensuring block sizes are below a certain size threshold allows the matching power of these techniques to be fully harnessed by running them on smaller subsets of large data sets.

In this paper, we propose two recursive clustering approaches for generating blocks within a specified size range. The idea behind our approaches is to use an initial blocking key to split a data set into individual blocks. If some of the blocks are too small we merge them, and if blocks are too large we use a second blocking key to split them. We merge any resulting small blocks, split any that are still too large using a third blocking key, and so on. Our two approaches differ in how we perform the clustering during the merge steps and as a result, give different distributions of block sizes, as we will present in Section 4.

Motivating Example. Throughout the rest of this paper we make use of the example data set in Table 1 to help illustrate this process. Figure 1 shows the algorithm flow of our approaches on this small data set.

Contributions. In this paper, we develop a novel blocking framework based on recursive agglomerative clustering to produce blocks in a specified size range. We then propose a novel penalty function which allows us to relax the hard block size restrictions and gives us control over the block generation process by selecting a trade-off between block size and block quality. We have conducted experiments on three real-world data sets and the results show our proposed approaches perform well on measures of block quality (both pairs completeness and reduction ratio [2]) in comparison to three baseline approaches, and can effectively generate blocks within the specified size range.

Record ID	First Name	Surname	Postcode
r_1	John	Smith	2000
r_2	Johnathon	Smith	2009
r_3	Joey	Schmidt	2009
r_4	Joe	Miller	2902
r_5	Joseph	Milne	2092
r_6	Paul		3000
r_7	Peter	Jones	3000

Table 1: Example data set.

Outline. We next discuss recent literature relating to iterative blocking and clustering methods. In Section 3 we describe the notation we use and formally define our problem. In Sections 4 and 5 we describe our blocking approaches and a penalty function which allows a trade-off between block size and block quality. In Section 6 we conduct an experimental evaluation of our approaches and we finish with conclusions and directions for future work in Section 7.

2. RELATED WORK

Blocking (also called *indexing*) for ER is an area of active research and several recent surveys have been conducted [3, 7, 15]. In the following we briefly describe some key prior research that relates to our work, in particular, the blocking techniques that adopt an iterative approach or that aim to control the size of blocks.

Several iterative blocking techniques have been studied for ER in recent years [5, 18, 23]. Whang et al. [23] proposed an iterative blocking process in order to perform ER. Rather than processing each block individually, the approach propagates the results from processed blocks (i.e. where records

have been compared) to inform decisions in subsequent blocks. Once two records are determined as a match, they are merged, and the resulting new record is propagated into other blocks where the combination of attributes may cause previously undetected matches to be found. The results of previous comparisons are stored so that comparisons are not repeated unnecessarily. However, these techniques give no control over the size of the blocks that are produced.

Das Sarma et al. [5] also developed an iterative blocking approach that combines splitting and merging to efficiently block large-scale data sets for ER. The work makes use of labelled training examples to generate blocking schemas in an automated fashion. The authors performed a post-processing step of merging small canopies (blocks) to increase recall based on a heuristic of minimising combined size and maximising the number of matches. While this technique gives some control of the block sizes, it does not enforce hard size contraints and also requires labelled training examples, whereas our approaches are unsupervised.

Ramadan et al. [18] modified the sorted neighbourhood approach [2] for real-time ER to allow for updating a blocking key value tree in response to a query record. The authors examined an adaptive window-size approach to vary the number of candidate records returned for comparison based on either a similarity or a size threshold. The similarity between neighbouring nodes in a tree can be pre-calculated to reduce query times. This approach does not enforce minimum and maximum size constraints nor does it generate individual blocks which makes it unsuitable for applications such as privacy-preserving record linkage.

Zhu et al. [25] examined the clustering problem under size constraints, although not in the context of ER. They proposed an approach to produce clusters of a certain size, which can also be relaxed to a size range. Nevertheless, the authors only tested their approach on small data sets that have less than one thousand records or no more than three clusters. Their approach also requires computing the complete similarity between all pairs of records in a data set, which limits its usefulness for blocking in ER tasks where the aim is specifically to avoid this complete pairwise comparison. Work by Ganganath et al. [10], Malinen and Fränti [16] and Rebollo-Monedero et al. [19] have the same limitations. In contrast to their work, our approach aims to support larger data sets, and does not require a complete pairwise comparison of all records.

3. PROBLEM STATEMENT

We assume that a data set $\mathbf{R}$ consists of records, each of which is associated with a set $\mathbf{A}$ of attributes. The value of an attribute $a \in \mathbf{A}$ in a record $r \in \mathbf{R}$ is denoted as $r.a$.

To split a set of records $\mathbf{R_x} \subseteq \mathbf{R}$ into blocks we make use of one or more blocking keys. A *blocking key*, $k_{i,j} = \langle a_i, f_j \rangle$ is a pair consisting of an attribute $a_i \in \mathbf{A}$ and a function f_j. The function f_j takes as input an attribute value and returns another value, such as a phonetic encoding, a substring, or a reversed string. For a given blocking key $k_{i,j} = \langle a_i, f_j \rangle$, we generate a *blocking key value* (BKV) for record $r_y \in \mathbf{R_x}$ by applying function f_j to $r_y.a_i$, denoted $v_{i,j,y} = f_j(r_y.a_i)$. For example, possible functions include the first two characters (F2), exact value (Ext) and a Soundex encoding (Sdx) [7].

To illustrate this using the example in Table 1, we consider the following blocking keys: the first two characters of the *FirstName* attribute $\langle FN, \mathsf{F2} \rangle$, a Soundex encoding of the *Surname* attribute $\langle SN, \mathsf{Sdx} \rangle$ and the exact value of the *Postcode* attribute $\langle PC, \mathsf{Ext} \rangle$. The BKV of $\langle FN, \mathsf{F2} \rangle$ applied to r_1 is 'Jo' (the first two characters of 'John'), the BKV of $\langle SN, \mathsf{Sdx} \rangle$ applied to r_1 is 'S530' (the Soundex encoding of 'Smith') and the BKV of $\langle PC, \mathsf{Ext} \rangle$ applied to r_1 is '2000'.

To split a set of records $\mathbf{R_x}$ into blocks we use a blocking key $k_{i,j}$ to generate a BKV $v_{i,j,y}$ for each $r_y \in \mathbf{R_x}$ and we create one block for each unique BKV generated. We insert each record into the block corresponding to its BKV. This means two records r_y, $r_z \in \mathbf{R_x}$ will be inserted into the same block if and only if they generate the same BKV using blocking key $k_{i,j}$, i.e. $f_j(r_y.a_i) = f_j(r_z.a_i)$. During our approaches we also need to merge blocks. This results in a single block being associated with multiple BKVs. We denote the set of BKVs associated with block $\mathbf{b}_i$ as $V(\mathbf{b}_i)$.

Based on a pre-defined list of blocking keys $K = \langle k_{i,j}, k_{l,m}, \ldots \rangle$, we aim to adaptively split $\mathbf{R}$ into to a set of blocks $\mathbf{B}$ by using the BKVs generated by one or more blocking keys in K. However, we also want to control the size of the blocks we produce. The *size* of a block $\mathbf{b}$, denoted as $|\mathbf{b}|$, is the number of records in the block. To control the size of blocks, we use two size parameters: s_{min} and s_{max} with $s_{min} \leq s_{max}$, to specify the minimum and maximum block sizes that are permitted, respectively.

Problem statement. Given a data set $\mathbf{R}$, two size parameters s_{min} and s_{max}, and a list of blocking keys $K = \langle k_{i,j}, k_{l,m}, \ldots \rangle$, the problem we study is to use K to partition the records in $\mathbf{R}$ into a set $\mathbf{B}$ of non-overlapping blocks such that for each $\mathbf{b} \in \mathbf{B}$, $s_{min} \leq |\mathbf{b}| \leq s_{max}$, and within each block the number of true matches is maximised and the number of true non-matches is minimised.

In practice, s_{min} and s_{max} can be set in accordance with operational requirements such as computational limitations, real-time efficiency requirements, or privacy-preserving requirements. As is common with other blocking techniques, we can also improve the robustness of our approaches by running them multiple times with different lists of blocking keys for a single ER task [3]. This reduces the impact that a single typographical error or incorrect value has on the ER process [2]. In future work we intend to further investigate the impact of different blocking keys and whether the optimal list of keys can be discovered automically.

4. CLUSTERING APPROACHES

We propose two recursive clustering approaches for generating blocks within a specified size range. The idea behind our approaches was illustrated in Figure 1. We iteratively split and merge blocks until the size parameters s_{min} and s_{max} are satisfied. The first approach processes blocks in order of decreasing block similarity (i.e., similarity-based), and the second approach in order of increasing block size (i.e., size-based). In Section 4.1 we briefly describe the way we calculate the similarity between BKVs as well as between blocks (clusters) during clustering, then in Sections 4.2 and 4.3 we describe our two approaches, and in Section 4.4 we discuss their respective advantages and disadvantages.

4.1 Similarity Functions

During clustering we aim to merge blocks with similar BKVs together, since this is more likely to bring true matches

Algorithm 1: SimilarityBasedClustering(R, K, ς, n, s_{min}, s_{max})

Input:
- Set of records: $\mathbf{R}$
- List of blocking keys: K
- Block similarity function: ς
- Current recursion depth: n // Set as $n = 1$ for first call to algorithm
- Minimum block size: s_{min}
- Maximum block size: s_{max}

Output:
- Set of correct sized blocks: $\mathbf{B}^*$

1: $\mathbf{B}$ = GenerateBlocks($\mathbf{R}$, $K[n]$) // Generate blocks using the n^{th} blocking key in K
2: $\mathbf{B}^-$, $\mathbf{B}^*$, $\mathbf{B}^+$ = SizePartition($\mathbf{B}$, s_{min}, s_{max}) // Partition $\mathbf{B}$ into too small, correct size, too large blocks
3: Q = GeneratePriorityQueue() // Create empty queue, ordered by similarity
4: **for** $\mathbf{b}_i$ **in** $\mathbf{B}^- \cup \mathbf{B}^*$ **do**:
5: **for** $\mathbf{b}_j$ **in** $\mathbf{B}^- \cup \mathbf{B}^* \setminus \mathbf{b}_i$ **do**:
6: **if** $|\mathbf{b}_i| + |\mathbf{b}_j| \leq s_{max}$ **then**:
7: Q.Insert($\varsigma(\mathbf{b}_i, \mathbf{b}_j)$, $\mathbf{b}_i$, $\mathbf{b}_j$) // Insert correct sized pairs into the queue
8: **while** $Q \neq \emptyset$ **do**:
9: sim, $\mathbf{b}_i$, $\mathbf{b}_j$ = Q.Pop() // Get the first pair from the queue
10: $\mathbf{b}_{ij}$ = MergeBlocks($\mathbf{b}_i$, $\mathbf{b}_j$)
11: **for** $\mathbf{b}_k$ **in** $\mathbf{B}^- \cup \mathbf{B}^*$:
12: **if** $|\mathbf{b}_{ij}| + |\mathbf{b}_k| < s_{max}$ **then**:
13: Q.Insert($\varsigma(\mathbf{b}_{ij}, \mathbf{b}_k)$, $\mathbf{b}_{ij}$, $\mathbf{b}_k$) // Put back in queue with new block similarity
14: **if** $|\mathbf{b}_{ij}| < s_{max}$ **then**:
15: $\mathbf{B}^* = \mathbf{B}^* \cup \mathbf{b}_{ij}$ // Add to correct size blocks
16: **for** $\mathbf{b}_i$ **in** $\mathbf{B}^+$ **do**: // Process the too large blocks
17: $\mathbf{B}_i$ = **SimilarityBasedClustering**($\mathbf{b}_i$, K, ς, $n+1$, s_{min}, s_{max}) // Call recursively with $n+1$
18: $\mathbf{B}^* = \mathbf{B}^* \cup \mathbf{B}_i$
19: **return** $\mathbf{B}^*$

together into the same block. This requires a way of measuring the similarity between two BKVs. In addition, once blocks are merged, each block can be associated with multiple BKVs as shown in Figure 1, so we also require a way of combining the pairwise similarities between BKVs into an overall block similarity measure.

To calculate the similarity between two BKVs v_1 and v_2, denoted as $\zeta(v_1, v_2)$ we use traditional string comparison functions such as Jaro-Winkler or Jaccard similarity [2]. However, this does not always give good results for blocking keys using functions such as Soundex encodings or the first two characters of an attribute. For example, none of the above similarity functions give a good indication of the similarity between the Soundex codes 'S530' and 'S550.' In order to obtain a better similarity measure, we use the original unencoded attribute values and apply a traditional string comparison function on them instead. For the above example we take the values that encode to 'S530' (such as 'Smith' and 'Smythe') and compute their similarity with values that encode to 'S550' (such as 'Simon' and 'Simeon'). If possible, we calculate all pairwise combinations of all values in a data set with these encodings to get a weighted average similarity between pairs of Soundex codes.

However, if the full pairwise calculation is computationally infeasible, we randomly sample a selection of original values for each code and take the average similarity between these. In practice we found that even small sample sizes still produced results that were nearly identical to those of the complete calculation. We discuss this further in Section 6.

To combine the pairwise similarity between BKVs into an overall *block similarity* measure, denoted as $\varsigma(\mathbf{b}_1, \mathbf{b}_2)$, we make use of three traditional approaches [24]: (1) *single link* $\varsigma(\mathbf{b}_1, \mathbf{b}_2) = \max(T)$, (2) *average link* $\varsigma(\mathbf{b}_1, \mathbf{b}_2) = \text{mean}(T)$ and (3) *complete link* $\varsigma(\mathbf{b}_1, \mathbf{b}_2) = \min(T)$, where $T = \{\zeta(v_1, v_2) : v_1 \in V(\mathbf{b}_1) \text{ and } v_2 \in V(\mathbf{b}_2)\}$.

4.2 Similarity-Based Blocking Approach

The similarity-based blocking approach is described in Algorithm 1. To begin, we set $n = 1$ and take the set of records as $\mathbf{R}$. We generate a set $\mathbf{B}$ of blocks using the n^{th} blocking key in K (line 1). One block is created for each unique BKV. Next, $\mathbf{B}$ is partitioned into three disjoint sets $\mathbf{B}^-$, $\mathbf{B}^*$ and $\mathbf{B}^+$, with $\mathbf{b}_i \in \mathbf{B}^-$ if $|\mathbf{b}_i| < s_{min}$, $\mathbf{b}_i \in \mathbf{B}^*$ if $s_{min} \leq |\mathbf{b}_i| \leq s_{max}$ and $\mathbf{b}_i \in \mathbf{B}^+$ if $|\mathbf{b}_i| > s_{max}$ (line 2). We place each pair of blocks in $\mathbf{B}^- \cup \mathbf{B}^*$ into a priority queue Q, in order of their decreasing block similarity (lines 4-7). We retrieve from Q the pair of blocks ($\mathbf{b}_i$, $\mathbf{b}_j$) with maximum $\varsigma(\mathbf{b}_i, \mathbf{b}_j)$ (line 9). We merge $\mathbf{b}_i$ and $\mathbf{b}_j$ into $\mathbf{b}_{ij}$ where $V(\mathbf{b}_{ij}) = V(\mathbf{b}_i) \cup V(\mathbf{b}_j)$. We then calculate $\varsigma(\mathbf{b}_{ij}, \mathbf{b}_k)$ for all $\mathbf{b}_k$ s.t. $|\mathbf{b}_k| + |\mathbf{b}_{ij}| \leq s_{max}$ and reinsert these new pairs of blocks into Q (line 13). We then proceed with the pair of blocks with the second highest block similarity (loop back to line 9), and continue this process until no more merges are possible. For each $\mathbf{b}_i \in \mathbf{B}^+$ (i.e. blocks that are too large, $|\mathbf{b}_i| > s_{max}$) we call the algorithm recursively with $\mathbf{b}_i$ as the new set of records and using the next blocking key in K to generate new BKVs (lines 16-18).

Figure 1 illustrates this process applied to the example data set in Table 1 with $K = \langle\langle FN, \mathsf{F2}\rangle, \langle SN, \mathsf{Sdx}\rangle\rangle$ and $s_{min} = 2$ and $s_{max} = 3$. We start by splitting the records into blocks using the first blocking key $\langle FN, \mathsf{F2}\rangle$ (the first two characters of $FirstName$). The blocks that have a size smaller than 2 (s_{min}) are clustered and merged. Any blocks with size greater than 3 (s_{max}) are split using the second blocking key $\langle SN, \mathsf{Sdx}\rangle$ (the Soundex encoding of $Surname$). Then, in a second merging phase, blocks that are smaller than size 2 (s_{min}) are again clustered. In this case this finishes the algorithm since all blocks are now in the correct size range. However, if there were still blocks with size greater than 3 they would be split using a third blocking key, for example $\langle PC, \mathsf{Ext}\rangle$, any resulting small blocks would again be clustered and merged, and so forth. This continues until no blocks remain with size greater than 3 or we run out of blocking keys in K.

The main drawback of the similarity-based approach is the need to calculate $\varsigma(\mathbf{b}_i, \mathbf{b}_j)$ for each pair of blocks in $\mathbf{B}^- \cup \mathbf{B}^*$ and store them in Q. In addition, as blocks are merged, the block similarity needs to be calculated between the new

Algorithm 2: SizeBasedClustering($\mathbf{R}$, K, ς, n, s_{min}, s_{max})

Input:
- Set of records: $\mathbf{R}$
- List of blocking keys: K
- Block similarity function: ς
- Current recursion depth: n // Set as $n = 1$ for first call to algorithm
- Minimum block size: s_{min}
- Maximum block size: s_{max}

Output:
- Set of correct sized blocks: $\mathbf{B}^*$

1: $\mathbf{B}$ = GenerateBlocks($\mathbf{R}$, $K[n]$) // Generate blocks using the n^{th} blocking key in K
2: $\mathbf{B}^-$, $\mathbf{B}^*$, $\mathbf{B}^+$ = SizePartition($\mathbf{B}$, s_{min}, s_{max}) // Partition $\mathbf{B}$ into too small, correct size, too large blocks
3: Q = GeneratePriorityQueue() // Create empty queue, ordered by block size
4: **for** $\mathbf{b}_i$ **in** $\mathbf{B}^-$ **do**:
5: Q.Insert($\mathbf{b}_i$) // Put the small blocks into the queue
6: **while** $Q \neq \emptyset$ **do**:
7: $\mathbf{b}_i = Q$.Pop() // Get the first block from the queue
8: $\mathbf{b}_j = Argmax(\varsigma(\mathbf{b}_i, \mathbf{b}_k)), \forall \mathbf{b}_k \in \mathbf{B}^- \cup \mathbf{B}^*$ such that $|\mathbf{b}_i| + |\mathbf{b}_k| \leq s_{max}$ // Get nearest block of correct size
9: $\mathbf{b}_{ij}$ = MergeBlocks($\mathbf{b}_i$, $\mathbf{b}_j$)
10: **if** $|\mathbf{b}_{ij}| < s_{min}$ **then**:
11: Q.Insert($\mathbf{b}_{ij}$) // Put new block back into the queue
12: **else**:
13: $\mathbf{B}^* = \mathbf{B}^* \cup \mathbf{b}_{ij}$ // Add to correct size blocks
14: **for** $\mathbf{b}_i$ **in** $\mathbf{B}^+$ **do**: // Process the too large blocks
15: $\mathbf{B}_i$ = **SizeBasedClustering**($\mathbf{b}_i$, K, ς, $n+1$, s_{min}, s_{max}) // Call recursively with $n+1$
16: $\mathbf{B}^* = \mathbf{B}^* \cup \mathbf{B}_i$
17: **return** $\mathbf{B}^*$

block and all remaining blocks. This reduces the scalability of the approach and also leads to high memory overhead since Q can become large, $O(|\mathbf{B}|^2)$. Next we present an alternative approach with better scalability that removes the need to store all pairwise combinations of blocks in memory.

4.3 Size-Based Blocking Approach

The size-based blocking approach is described in Algorithm 2. The initial setup for this approach is identical to that of the similarity-based blocking approach. However, in the size-based case the priority queue Q contains individual blocks, which are ordered by increasing block size (line 5). This is an important distinction since it significantly reduces the size of Q from $O(|\mathbf{B}|^2)$ to $O(|\mathbf{B}|)$. In the main loop of the algorithm (lines 6-13) we remove the smallest block $\mathbf{b}_i$ from Q (line 7), determine the block $\mathbf{b}_j$ such that $|\mathbf{b}_i| + |\mathbf{b}_j| \leq s_{max}$ and $\varsigma_1(\mathbf{b}_i, \mathbf{b}_j)$ is maximised (line 8). Essentially we find the most similar block to $\mathbf{b}_i$ such that their combined size would be less than s_{max}. We merge $\mathbf{b}_i$ and $\mathbf{b}_j$ into $\mathbf{b}_{ij}$ (line 9) and if $|\mathbf{b}_{ij}| \leq s_{min}$, we reinsert $\mathbf{b}_{ij}$ into Q. We then proceed to the next smallest block (loop back to line 6) and continue this process until no blocks remain with size less than s_{min}. As with the similarity-based approach, for each block in $\mathbf{B}^+$ the algorithm is called recursively with $n = n + 1$ and using the next blocking key in K.

4.4 Discussion and Algorithm Complexities

We now discuss the characteristics of the two approaches and present their computational complexities. Depending on the settings of s_{min} and s_{max}, it is possible that our approaches may generate some blocks that are outside the desired size range. For example, if $s_{min} = 0.8 * s_{max}$, some blocks may have a size in the range $0.5 * s_{max}$ to $0.8 * s_{max}$. Merging any two of these blocks would result in a block size greater than s_{max}, so none of them end up being merged. However, if s_{min} and s_{max} satisfy $s_{max} \geq 2 * s_{min}$ then we are guaranteed that at most one block at the end will be smaller than s_{min} because if two blocks were left, they could be merged as their combined size would still be below s_{max}.

If blocks are left at the end of either algorithm which are larger than s_{max}, then there must exist some unique combination of BKVs that occurs more frequently than s_{max} and our only option is to add another blocking key to K.

The similarity-based blocking approach ensures that pairs of blocks with high block similarity are merged together. In practice, the approach often creates many blocks that are close in size to s_{max} which makes it effective for load balancing in parellel ER applications [14]. However, if there is a block left at the end which is too small, it may be quite small in comparison to s_{min}, which may make this approach less suitable in applications where s_{min} is important. The running time of the similarity-based approach is also typically longer than that of the size-based approach.

In practice, if $s_{max} \geq 2 * s_{min}$, the size-based approach tends to produce blocks that are more evenly distributed within the size range, with potentially a single block that is too small. Since the merging is done iteratively from smallest to largest, if there is a block that is smaller than s_{min}, its size is typically close to s_{min}, although this closeness is not mathematically guaranteed. This means that for situations where minimum block size is important the size-based approach is a good candidate. However, the size-based blocking approach is not as successful when there are multiple large blocks with different BKVs from values that are quite similar. For example, depending on the blocking keys used, the first names 'John' and 'Johnathon' may generate different BKVs but we would prefer to combine them into the same block. However, because blocks are processed in order of size and both blocks may be quite large, neither block will be considered until late in the merging process. As a result, by the time they are compared one of them may have already grown too large (due to other merges) for them to be merged. This situation can be partially overcome by the penalty function detailed in the next section.

The selection of the blocking keys in K is important for both approaches and has a significant effect on the running time and the blocking quality. At present we rely on domain expertise to select the blocking keys, taking into account such factors as completeness, size of the domain, distribution of values and general data quality. As part of our future work we intend to investigate methods for automatically se-

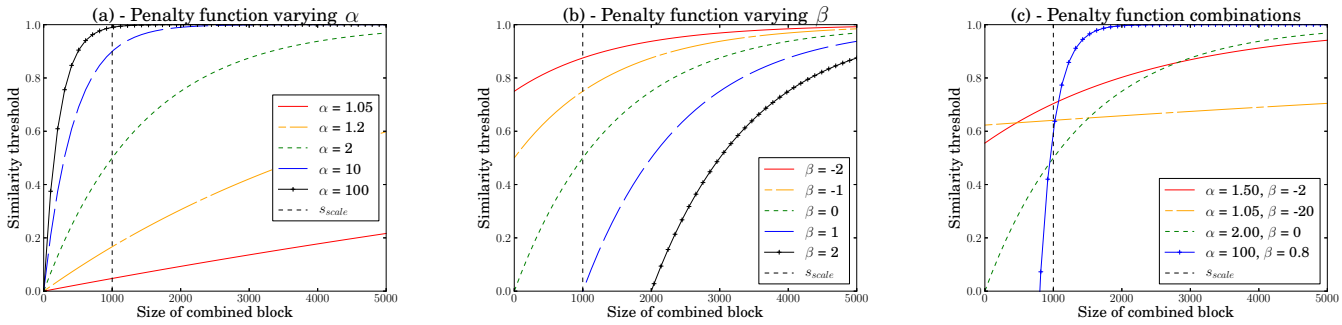

Figure 2: Penalty function example configurations where the vertical dotted line in each plot is s_{scale}.

lecting blocking keys, such as those developed by Kejriwal and Miranker [13] and Ramadan and Christen [17].

In the worst case, the time complexity of the similarity-based approach is $O(|\mathbf{R}|^3 \log(|\mathbf{R}|))$, while the size-based approach is $O(|\mathbf{R}|^3)$. For the similarity-based approach, Q can contain $O(|\mathbf{R}|^2)$ blocks (line 7) and during the loop (lines 8 - 15) we have to perform $O(|\mathbf{R}|)$ insertions into Q of time complexity $O(\log(|\mathbf{R}|))$ (line 13). For the size-based approach the size of Q is at most $O(|\mathbf{R}|)$ (line 5) but calculating $Argmax(\varsigma(\mathbf{b}_i, \mathbf{b}_k))$ (line 8) is potentially $O(|\mathbf{R}|^2)$ so we end up with an overall complexity of $O(|\mathbf{R}|^3)$.

In practice the similarity-based approach is significantly slower than the size-based approach. In addition the running time of both approaches is much more dependent on the number of unique BKVs generated by the blocking keys in K rather than the size of $\mathbf{R}$. This is because we create one block for each BKV during clustering so the running time of the similarity-based approach becomes $O(|\mathbf{B}|^3 log(|\mathbf{B}|))$ and the size-based approach becomes $O(|\mathbf{B}|^3)$. In the worst case, each record generates a unique BKV and we end up with the asymptotic complexity above. Phonetic encodings such as Soundex and Double Metaphone [7], which have hard limits on the maximum number of unique BKVs they can create, can be particularly effective in this regard. Similarly, selecting just the first one or two characters from an attribute also restricts the maximum number of blocks that can be created at each step. In addition, some optimisation techniques such as pre-calculating and caching similarity values can be performed to improve the efficiency of both techniques.

5. PENALTY FUNCTION

If two blocks have similar BKVs, then it may be preferable to merge them even if they are large, and use the next blocking key in K to split them and enforce the size restrictions. We now present a penalty function that replaces the hard size restrictions (s_{min} and s_{max}) on merging blocks in our approaches with a sliding scale, that combines block size and block similarity to determine whether or not to merge two blocks. The penalty function $\mathbf{\Phi}$ is as follows:

$$\mathbf{\Phi}(\mathbf{b}_i, \mathbf{b}_j) = 1 - \alpha^{-\left(\frac{|\mathbf{b}_i|+|\mathbf{b}_j|}{s_{scale}} - \beta\right)} \text{ for } \alpha \geq 1 \text{ and } \beta \in \mathbb{R}.$$

Two blocks $\mathbf{b}_i$ and $\mathbf{b}_j$ will be merged if they satisfy the inequality $\varsigma(\mathbf{b}_i, \mathbf{b}_j) \geq \mathbf{\Phi}(\mathbf{b}_i, \mathbf{b}_j)$. As the combined block size gets larger, the similarity threshold required for merging also increases, and vice versa.

The penalty function involves two parameters, α and β, which together with s_{scale} (related to s_{min} and s_{max}), produce the desired merging behaviour.

- α determines the trade-off between similarity and size. Higher values of α produce a stricter similarity threshold as the block size increases. In the limit as $\alpha \to \infty$, $\mathbf{\Phi}$ becomes a hard size restriction. In this case block similarity does not affect the merging decisions.

- β constrains the clustering by enforcing either a minimum block size ($\beta > 0$) or a minimum similarity threshold ($\beta < 0$). If $\beta = 0$ then there are no size or similarity restrictions on the merging. We note that β can only create one of these two types of restrictions for a given clustering problem, since there may be no solution if both a minimum block size and a minimum similarity threshold are specified.

- s_{scale} is a scaling parameter that is useful for computational reasons. In practice, including s_{scale} removes the need to repeatedly calculate extremely large exponents of numbers very close to 1 when computing $\mathbf{\Phi}(\mathbf{b}_i, \mathbf{b}_j)$. Eliminating s_{scale} by changing α and β gives a mathematically identical function, but with α extremely close to 1 in practice. Including s_{scale} improves computational performance and prevents machine precision from influencing results. We explain how to set s_{scale} below.

We next provide the idea behind the penalty function with reference to the examples in Figure 2. In each example s_{scale} is set to 1,000 (the vertical dashed line). Consider the case where $\alpha = 2$ and $\beta = 0$ represented by the curved dashed line from $(0, 0)$ to the top right corner in each example. Before merging two blocks $\mathbf{b}_i$ and $\mathbf{b}_j$ where $|\mathbf{b}_i| + |\mathbf{b}_j| = s_{scale}$ (i.e. $|\mathbf{b}_i| + |\mathbf{b}_j| = 1{,}000$), the similarity between the blocks must be at least $1 - \frac{1}{2^1} = 0.5$. Before merging two blocks with a combined size of $2 * s_{scale}$, the similarity must be at least $1 - \frac{1}{2^2} = 0.75$. A size of $3 * s_{scale}$ requires similarity greater than 0.875, and so on.

The value of α determines the rate at which the required similarity approaches 1.0, with higher values approaching more quickly than lower values as shown in Figure 2(a). Changing the value of β has the effect of moving the curve to the left or right as shown in Figure 2(b). For example, $\beta = -1$ and $\alpha = 2$ set a minimum similarity for merging to be $1 - \frac{1}{2^1} = 0.5$. If $\beta = 1$ and $\alpha = 2$, then blocks will be merged regardless of similarity until the combined size

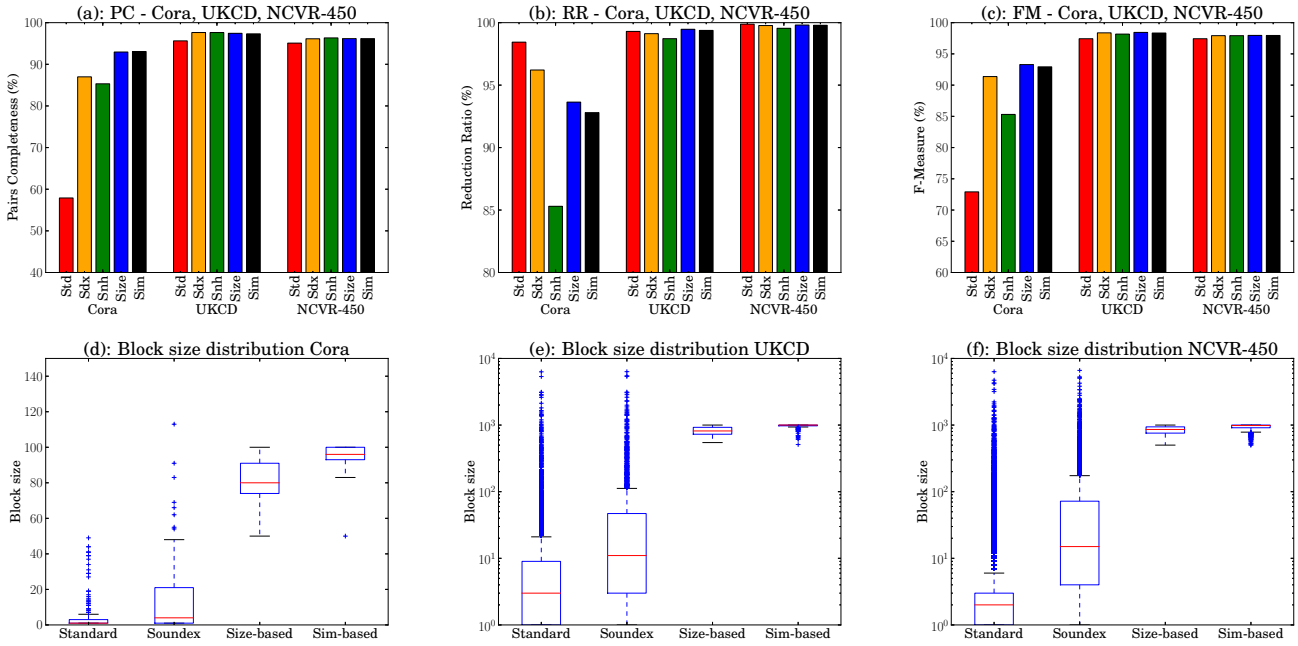

Figure 3: 3(a) Pairs Completeness, 3(b) Reduction Ratio, and 3(c) F-Measure results for standard blocking (Std), Soundex encoding (Sdx), sorted neighbourhood (Snh), our proposed size-based approach (Size), and our proposed similarity-based approach (Sim). Block size distributions for 3(d) Cora, 3(e) UKCD, and 3(f) NCVR-450. We do not include block size distributions for the sorted neighbourhood approach since this technique uses a window of constant size.

is at least 1,000 (equal to s_{scale}). By combining different values of α and β we can obtain a wide variety of merging conditions as shown in Figure 2(c).

We now explain how best to choose the values of α, β and s_{scale} in order to achieve the desired merging behaviour. If minimum block size is not critical, the default we use on a data set is $s_{scale} = 0.5 * s_{max}$, $\alpha = 2$ and $\beta = 0$. This sets a similarity threshold of 0.75 to merge blocks with combined size greater than s_{max} and prevents blocks with very low similarity from being merged regardless of size. If minimum block size is important, then the default parameters we use are $s_{scale} = s_{min}$, $\alpha = (2 * s_{max})/(s_{max} - s_{min})$ and $\beta = 1$. This causes blocks to be merged regardless of similarity up to a combined size of s_{min}, and sets a similarity threshold of 0.75 to merge blocks with a combined size larger than s_{max}. In both cases, with some knowledge of the data, the value of α can be scaled to increase or decrease the similarity threshold of 0.75 as desired.

To incorporate the penalty function, both Algorithm 1 and 2 have to be slightly modified. In Algorithm 1, we replace the size restrictions on $\mathbf{b}_i$ and $\mathbf{b}_j$ in lines 4 - 6 with the penalty function condition, and the same for $\mathbf{b}_k$ and $\mathbf{b}_{ij}$ in lines 11 and 12. In Algorithm 2, all blocks are inserted into Q in line 5, not just blocks with size less than s_{min}. Similarly $\mathbf{b}_{ij}$ is always reinserted into Q in line 11, regardless of size. Additionally, in line 8, we replace the size restriction on $\mathbf{b}_k$ with the penalty function condition on $\mathbf{b}_i$ and $\mathbf{b}_k$.

6. EXPERIMENTAL EVALUATION

We have evaluated our approaches on three data sets. (1) Cora: This is a public bibliographic data set of scientific papers that has previously been used to evaluate ER techniques [20]. This data set contains 1,295 records and truth data is available. (2) UKCD: This data set consists of census data for 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 individual records of 32,000 households. A portion of this data (nearly 5,000 records) has been manually linked by domain experts. Fu et al. [9] have used this data set for household based group linkage where the task is to link households across time. (3) NCVR: This data set consists of voter registration data for the state of North Carolina in the USA [4].[1] It contains 8.2 million records consisting of the full name, address, age and other personal information of voters registered in the state. For most of our experiments we make use of a subset of this data set containing 447,898 records, named NCVR-450. We use the full data set to test the scalability of our approaches.

To evaluate our approaches we compared performance with standard blocking [8], Soundex encoding [7], and sorted neighbourhood based indexing [11]. For evaluation measures we used pairs completeness and reduction ratio [2] and a combination of the two measures similar to F-Measure: Pairs Completeness (PC) = $\frac{s_M}{n_M}$, Reduction Ratio (RR) = $1 - \frac{s_M + s_N}{n_M + n_N}$ and the combined F-Measure (FM) = $\frac{2 * PC * RR}{PC + RR}$, where n_M, n_N, s_M, s_N correspond to the total number of matched pairs, the total number of non-matched pairs, the number of true matched candidate record pairs and the number of true non-matched candidate pairs, respectively.

[1] ftp://alt.ncsbe.gov/data/

Cora										
s_{min} - s_{max}		20 - 50			20 - 100			50 - 100		
Block similarity measure ς		Single	Average	Complete	Single	Average	Complete	Single	Average	Complete
Size-based	PC	83.45	84.19	80.90	**92.95**	92.24	81.95	**92.95**	91.55	85.90
	RR	96.86	**97.03**	97.01	96.20	96.35	96.50	93.64	93.61	93.63
	FM	89.66	90.16	88.23	**94.55**	94.25	88.63	93.29	92.57	89.60
Similarity-based	PC	87.77	88.27	85.52	92.95	92.95	92.95	**93.07**	92.97	92.95
	RR	96.51	96.61	**96.71**	95.64	96.09	96.14	92.80	93.28	93.55
	FM	91.93	92.25	90.77	94.28	94.49	**94.52**	92.93	93.12	93.25
UKCD										
s_{min} - s_{max}		50 - 100			100 - 200			500 - 1,000		
Block similarity measure ς		Single	Average	Complete	Single	Average	Complete	Single	Average	Complete
Size-based	PC	89.64	88.72	87.49	93.65	93.32	91.57	**97.44**	96.77	95.92
	RR	**99.95**	**99.95**	**99.95**	99.89	99.89	99.90	99.47	99.48	99.48
	FM	94.51	94.00	93.31	96.67	96.49	95.55	**98.44**	98.11	97.67
Similarity-based	PC	90.43	90.24	89.33	93.76	93.82	93.18	97.32	97.27	**97.42**
	RR	99.94	99.94	**99.95**	99.88	99.89	99.89	99.38	99.44	99.45
	FM	94.95	94.84	94.34	96.72	96.76	96.42	98.34	98.34	**98.42**
NCVR-450										
s_{min} - s_{max}		500 - 1,000			2,500 - 5,000			5,000 - 10,000		
Block similarity measure ς		Single	Average	Complete	Single	Average	Complete	Single	Average	Complete
Size-based	PC	96.17	96.25	96.15	96.49	96.53	96.48	96.63	**96.64**	96.63
	RR	99.81	**99.82**	**99.82**	99.07	99.08	99.09	98.19	98.16	98.19
	FM	97.96	**98.00**	97.95	97.76	97.79	97.77	97.40	97.39	97.40
Similarity-based	PC	96.17	96.35	96.32	96.50	96.57	96.55	96.67	**96.68**	96.66
	RR	99.79	99.80	**99.81**	98.96	99.01	99.07	98.00	98.03	98.05
	FM	97.95	**98.04**	98.03	97.71	97.77	97.79	97.33	97.35	97.35

Table 2: Effects of parameter settings on PC, RR and F-Measure for Cora, UKCD, and NCVR-450, showing different configurations of s_{min}, s_{max} and the three different block similarity measures (ς) single link, average link, and complete link. The best value(s) in each row is shown in bold.

We do not explicitly model block quality. However, since merging blocks can only improve improve PC, we merge blocks until s_{max} is reached, regardless of block quality. If higher quality blocks are preferred over larger blocks, this can be achieved by using the penalty function, where a minimum similarity threshold will prevent blocks with a low likelihood of containing true matches from being merged, regardless of block size.

All our experiments were performed on a server with 6-core 64-bit Intel Xeon 2.4 GHz CPUs, 128 GBytes of memory and running Ubuntu 14.04. All programs were written in Python 3. For similarity functions we used Jaro-Winkler for single proper name attributes (i.e. first name or last name) and q-gram based Jaccard similarity for other string attributes with $q = 2$ [2].

Each of our experiments uses a single list of blocking keys. As with many blocking techniques, the overall results could be improved by combining the blocks generated from multiple lists of blocking keys, with a corresponding reduction in s_{max} so as to maintain any efficiency requirements. In future work we plan to investigate the automatic selection of blocking keys to reduce the need for domain expertise.

The experimental results on the Cora, UKCD, and NCVR-450 data sets are shown in Figure 3(a) - 3(c). For Cora we set $s_{min} = 50$, $s_{max} = 100$ and $K = \langle\langle Title, \mathsf{Ext}\rangle, \langle Author, \mathsf{Ext}\rangle\rangle$. For UKCD we set $s_{min} = 500$, $s_{max} = 1,000$ and $K = \langle\langle Surname, \mathsf{Ext}\rangle, \langle First\ Name, \mathsf{Ext}\rangle, \langle Birth\ Parish, \mathsf{Ext}\rangle\rangle$. For NCVR-450 we set $s_{min} = 500$, $s_{max} = 1,000$ and $K = \langle\langle Surname, \mathsf{F2}\rangle, \langle First\ Name, \mathsf{F2}\rangle\rangle$.

On all three data sets, we achieve equal or better F-Measure values than the three baseline approaches. This indicates that our approaches achieve comparable blocking quality to other common blocking techniques. However, the main focus of our approaches was to satisfy the block size restrictions while achieving high quality blocking. We also show the distribution of block sizes generated by our approaches in Figure 3(d) - 3(f). As can be seen from the results, both our approaches produce blocks in the required size range, 500 - 1,000 records for UKCD and NCVR-450, and 50 - 100 records for Cora. While the size-based approach tends to distribute the block sizes throughout the interval $[s_{min}, s_{max}]$, the similarity-based approach tends to generate the majority of blocks with size close to s_{max}. This means it creates fewer blocks overall and makes it appropriate for parallel ER applications.

We tested different parameter settings for our approaches to examine how sensitive they are to changing s_{min}, s_{max}, and the block similarity measure ς, and the results are shown in Table 2. In most cases, the choice of block similarity measure ς has minimal effect on the results. However, complete link did not work well with the size-based approach, particularly on the Cora data set. Changing s_{min} and s_{max} affects the trade-off between PC and RR as expected.

We tested the penalty function and the results are shown in Figure 4. For Cora we set $s_{scale} = 50$ and $s_{max} = 100$, and for UKCD and NCVR-450 we set $s_{scale} = 500$ and $s_{max} = 1,000$. When $\beta = 0$ (no minimum block size or minimum similarity threshold), the penalty function generally achieves the best combination of PC and RR values,

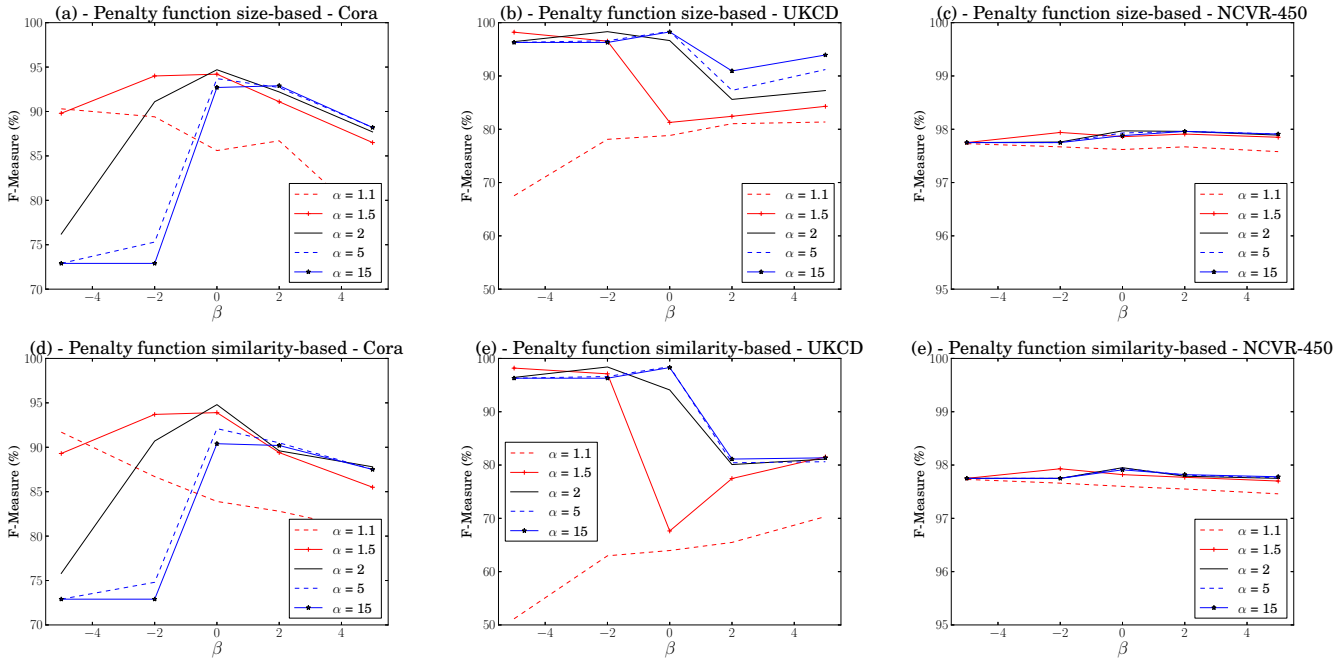

Figure 4: Penalty function results for Cora - 4(a) and 4(d), UKCD - 4(b) and 4(e), and NCVR-450 - 4(c) and 4(f). For each data set we display how different combinations of α and β affect the F-Measure values.

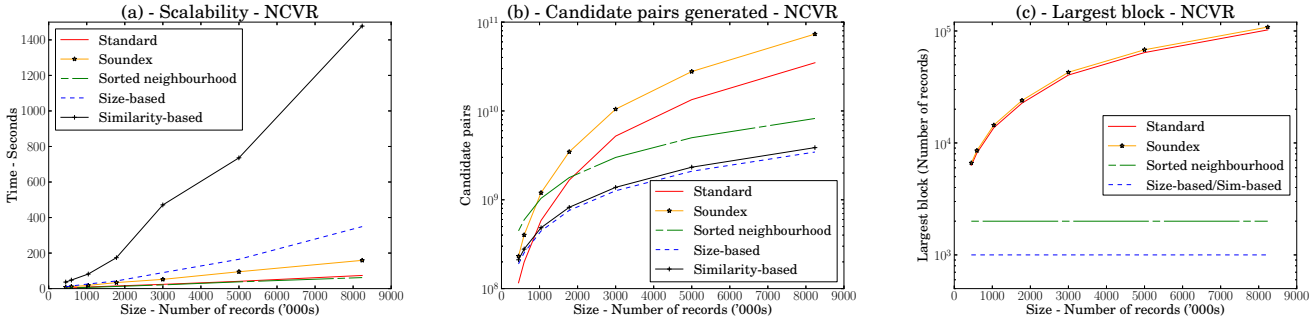

Figure 5: The scalability of the different approaches on the full NCVR data set.

the exception being for low values of α where the similarity threshold is very low, even for large blocks which results in poor RR values. High values of α and negative values of β mean the similarity threshold to perform any merging is high. This essentially negates the clustering steps of the algorithms, which results in poor PC values for data sets with lower data quality. High values of α in combination with positive values of β produce generally balanced blocks. We note that for the UKCD data set, setting $\alpha = 1.1$ performs very poorly. It repeatedly merges many blocks in each iteration of the algorithm and either runs out of blocking keys (resulting in poor RR values), or has to use attributes that have poor data quality (resulting in poor PC values). For the NCVR-450 data set, the penalty function produces very similar results regardless of the settings for α and β. The merging of blocks has less impact on the NCVR-450 data set, since it is relatively clean so merges do not increase PC values substantially, and also large enough that it requires many merges to reduce RR values significantly.

We also tested the scalability of our approaches using subsets of different sizes of the entire NCVR data set. We set $s_{min} = 500$, $s_{max} = 1,000$ and $K = \langle\langle Surname, \mathsf{F2}\rangle, \langle First\ Name, \mathsf{F2}\rangle\rangle$ and the results are shown in Figure 5(a). As can be seen, even though the asymptotic complexity of each approach is cubic or worse, because functions such as F2 or Sdx generate a limited number of BKVs the scalability is still nearly linear in practice. However, in the future we plan to optimise both approaches to improve their scalability.

We compared the total number of candidate pairs generated as well as the largest block generated by the different approaches and the results are shown in Figure 5(b) and Figure 5(c). Controlling the maximum block size ensures that the total number of candidate pairs increases linearly with the size of the data set which means that once the data set becomes large, our techniques generate fewer candidate pairs than the traditional and Soundex based approaches. As a result, even though our approaches increase the time required for blocking compared to the baseline approachs,

in general this will be more than made up for by a reduction in the time required to perform the matching.

In addition, the worst case block size is also controlled by our approaches. This means that if the blocking is being performed as a pre-processing step for an ER technique with scalability worse than quadratic, such as Markov logic networks [20], or privacy-preserving record linkage [22], then the time saving will be even greater than that indicated by the reduction in the number of candidate pairs. Controlling the worst-case block size means that our techniques are suitable for real-time ER, where operational requirements limit the number of comparisons that can be performed [18].

Finally, we investigated the impact of the sample size in the similarity calculations on the NCVR-450 data set using Soundex encodings. Even with a sample size of 1, the clustering still produced similar results to the complete calculation and the reduction in F-Measure was less than 0.1% in all cases. As a result, we conclude that the sample size does not significantly affect the performance.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have developed two novel recursive clustering approaches which can generate blocks for ER within a given size range. We have also proposed a penalty function which allows us to control the trade-off between block size and block quality, and fine tune either approach. We have evaluated our approaches on three data sets. Our experimental results show that both our techniques perform well in comparison to the baseline approaches and create blocks in the required size range.

In the future, we intend to extend the current work in several directions. First, we hope to investigate the possibility of automatically selecting the blocking keys using techniques similar to Kejriwal and Miranker [13]. We also aim to investigate optimisations to the algorithms and the use of different clustering techniques, to improve the quality of the results and the scalability of our approaches.

Acknowledgements

This work was partially funded by the Australian Research Council, Veda, and Funnelback Pty. Ltd., under Linkage Project LP100200079.

8. REFERENCES

[1] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM TKDD*, 1(1), 2007.

[2] P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer, 2012.

[3] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE TKDE*, 24(9), 2012.

[4] P. Christen. Preparation of a real temporal voter data set for record linkage and duplicate detection research. Technical report, Australian National University, 2014.

[5] A. Das Sarma, A. Jain, A. Machanavajjhala, and P. Bohannon. An automatic blocking mechanism for large-scale de-duplication tasks. In *ACM CIKM*, pages 1055–1064, 2012.

[6] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *ACM SIGMOD*, pages 85–96, 2005.

[7] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE TKDE*, 19(1):1–16, 2007.

[8] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *JASA*, 64(328):1183–1210, 1969.

[9] Z. Fu, P. Christen, and J. Zhou. A graph matching method for historical census household linkage. In *PAKDD, Springer LNAI vol. 8443*, pages 485–496. 2014.

[10] N. Ganganath, C.-T. Cheng, and C. Tse. Data clustering with cluster size constraints using a modified k-means algorithm. In *CyberC*, pages 158–161, Oct 2014.

[11] M. A. Hernandez and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Springer DMKD*, 2(1):9–37, 1998.

[12] D. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM TODS*, 31(2):716–767, 2006.

[13] M. Kejriwal and D. P. Miranker. An unsupervised algorithm for learning blocking schemes. In *IEEE ICDM*, pages 340–349, 2013.

[14] H. Kim and D. Lee. Parallel linkage. In *ACM CIKM*, pages 283–292, 2007.

[15] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Elsevier DKE*, 69(2):197–210, 2010.

[16] M. Malinen and P. Fränti. Balanced k-means for clustering. In *SSSPR, Springer LNCS vol. 8621*, pages 32–41. 2014.

[17] B. Ramadan and P. Christen. Unsupervised blocking key selection for real-time entity resolution. In *PAKDD, Springer LNAI vol. 9078*, pages 574–585, 2015.

[18] B. Ramadan, P. Christen, and H. Liang. Dynamic sorted neighborhood indexing for real-time entity resolution. In *ADC, Springer LNCS vol. 8506*, pages 1–12. 2014.

[19] D. Rebollo-Monedero, M. Solé, J. Nin, and J. Forné. A modification of the k-means method for quasi-unsupervised learning. *Elsevier KBS*, 37(0):176 – 185, 2013.

[20] P. Singla and P. Domingos. Entity resolution with Markov logic. In *IEEE ICDM*, pages 572–582, 2006.

[21] D. Vatsalan and P. Christen. Sorted nearest neighborhood clustering for efficient private blocking. In *PAKDD*, volume 7819 of *LNCS*, pages 341–352. Springer, 2013.

[22] D. Vatsalan, P. Christen, and V. S. Verykios. A taxonomy of privacy-preserving record linkage techniques. *Elsevier IS*, 38(6):946–969, 2013.

[23] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *ACM SIGMOD*, pages 219–232, 2009.

[24] R. Xu and I. Wunsch, D. Survey of clustering algorithms. *IEEE TNN*, 16(3):645–678, May 2005.

[25] S. Zhu, D. Wang, and T. Li. Data clustering with size constraints. *Elsevier KBS*, 23(8):883–889, 2010.