Active Learning based Similarity Filtering for Efficient and Effective Record Linkage*

Charini Nanayakkara, Peter Christen, and Thilina Ranbaduge

School of Computing, The Australian National University,
Canberra, ACT 2600, Australia. charini.nanayakkara@anu.edu.au

Abstract. The limited analytical value of using individual databases on their own increasingly requires the integration of large and complex databases for advanced data analytics. Linking personal medical records with travel and immigration data, for example, will allow the effective management of pandemics such as the current COVID-19 outbreak by tracking potentially infected individuals and their contacts. One major challenge for accurate linkage of large databases is the quadratic or even higher computational complexities of many advanced linkage algorithms. In this paper we present a novel approach that, based on the expected number of true matches between two databases, applies active learning to remove compared record pairs that are likely non-matches before a computationally expensive classification or clustering algorithm is employed to classify record pairs. Unlike blocking and indexing techniques that are used to reduce the number of record pairs to be compared, using recursive binning on a data dimension such as time or space, our approach removes likely non-matching record pairs in each bin after their comparison. Experiments on two real-world databases show that similarity filtering can substantially reduce run time and improve precision, at the costs of a small reduction in recall, of the final linkage results.

Keywords: Entity resolution · Efficiency enhancement · Binning

1 Introduction

Record linkage, as outlined in Fig. 1, is the process of identifying pairs of records that correspond to the same entity in one or across two or more databases [3]. Due to the quadratic time complexity of comparing every possible pair of records across two databases to be linked, the comparison step in record linkage is often preceded by a *blocking* or *indexing* step [16], where similar records are grouped into blocks such that only pairs of records within a block are compared. Additional *meta-blocking* [7] methods can be applied to further reduce the number of record pairs that need to be compared by analysing records within and across blocks to prevent redundant and superfluous record pair comparisons [16].

The pairwise comparison step of record linkage then consists of the calculation of similarities between two records, generally using string comparison functions applied on attributes such as names and addresses [3]. A *similarity graph*

* This work was partially funded by the ARC under grant DP160101934.

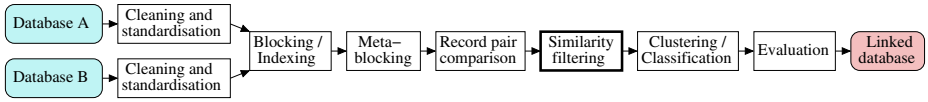

Fig. 1. The steps of the record linkage process, with our contribution highlighted.

can then be generated where nodes correspond to records and edges to the calculated similarities between them. However, even with blocking, indexing, and meta-blocking applied, many of these similarities will be low, and furthermore they do not correspond to true matches [3, 16]. In the classification step all compared record pairs are then classified as *matches* (records assumed to correspond to the same entity) or *non-matches* (records assumed to correspond to different entities) using a decision model that can be as simple as a similarity threshold, that can take match and error probabilities into account, or that uses training data to learn a supervised classification model [3]. Given training data are often not available in real-world linkage applications, unsupervised techniques need to be used that exploit the structure of a similarity graph [2, 6, 10, 20].

While blocking, indexing, and meta-blocking can significantly reduce the number of record pairs that need to be compared in the comparison step, the similarity graph generated from pair-wise comparisons can still be very large. For example, using a Min-hash LSH [13] based blocking on the data sets used in the evaluation in Sect. 4 that contain 17,613 and 3,007,153 records, resulted in similarity graphs containing over 4 and 34 million edges, respectively. Such large graphs are commonly required to ensure that the vast majority of true matches are included in order to obtain a high recall of the final linkage results [3].

Large similarity graphs can, however, challenge any algorithm used to classify record pairs because these graphs are likely very imbalanced and contain many non-matching record pairs. The size of these graphs can also result in the classification step to become the bottleneck of the record linkage process [6].

In our work we remove record pairs from a similarity graph that are unlikely true matches before this graph is being used for clustering or classification. We assume that for a given linkage problem an approximate number of expected true matches can be obtained from a domain expert. For example, when linking product databases from two online stores (where one-to-one links are expected), then the number of true matches is limited by the number of records in the smaller of two databases being linked. On the other hand, when linking birth records of families, then the known distribution of family sizes in a population can be used to estimate an expected number of true matches [19].

We develop an active learning process where we bin the record pairs in a similarity graph according to a suitable data dimension, such as time or space. For example, work on temporal linkage [11] has shown that people will move over time and possibly even change their names, resulting in lower similarities for true matches. Similarly, if people move longer distances then a larger number of their address details will change (such as state or even country). Our approach recursively splits a similarity graph into bins, where we then obtain, via active

learning, information from a domain expert about the distribution of matches and non-matches in these bins. We finally select a desired number of record pairs with the highest similarities from each bin, resulting in a much reduced similarity graph that still has a high recall of true matches, and that facilitates accurate clustering or classification with substantially reduced run times.

2 Related Work

The need for linking databases has ensured research for over fifty years and led to a diverse range of methods being developed [4, 5, 14]. Research into how to improve the scalability of record linkage has concentrated on blocking and indexing, and more recently meta-blocking [16]. The former aim to limit comparisons to only record pairs that are likely true matches, while meta-blocking aims to prevent redundant record pair comparisons, or superfluous comparisons between pairs already classified as non-matches [7]. Any good blocking, indexing, and meta-blocking method needs to be able to group true matches into the same block while records of different entities are grouped into different blocks [3].

Even though blocking, indexing, and meta-blocking help to improve the efficiency of the comparison step, the classification step may still be inefficient due to the presence of a large number of record pairs where many of them are likely not matches. Apart from the parallelisation of linkage algorithms [7], limited research has so far investigated how to improve the efficiency of the classification step without compromising the final linkage quality.

For many real-world record linkage applications, obtaining complete ground truth data (all true matching record pairs) is challenging due to large database sizes [3]. Even though crowdsourcing has been explored for record linkage [23] to mitigate the lack of ground truth data, allowing the public to classify record pairs is not applicable in many domains due to privacy concerns [4]. Active learning approaches, where a small number of selected record pairs are manually classified by trusted domain experts, have therefore been adopted for record linkage to generate ground truth data suitable to train supervised classifiers [17, 18, 24], or to generate high quality blocking results [21]. Active learning based on domain expertise, while being able to generate high quality ground truth data, can however only generate small numbers of labelled record pairs. Therefore, approaches that consider a limited labelling budget are crucial [24]. To the best of our knowledge, our approach is a first to explore how active learning can be employed to conduct filtering of record pairs after their comparison to improve the overall efficiency and effectiveness of the linkage process.

3 Active Learning based Similarity Filtering

We now describe our record pair similarity filtering approach based on domain knowledge. Domain experts often have a good understanding about what the number of true matches in their databases might be, depending upon the linkage situation (such as one-to-one or many-to-many links) and application [3].

As shown in Fig. 1, similarity filtering is an additional step applied between the comparison and classification steps in the record linkage process [3]. The aim of filtering is to improve effectiveness and run time of the classification step by reducing the number of non-matching record pairs (represented by their similarities) that are given to a classification or clustering algorithm [2, 6, 10, 20].

3.1 Problem Definition

Without loss of generality, we assume two databases, $\mathbf{D}_A$ and $\mathbf{D}_B$, are to be linked. A blocking method [16] has generated a set of candidate record pairs (r_i, r_j), with $r_i \in \mathbf{D}_A$ and $r_j \in \mathbf{D}_B$. These pairs have been compared using comparison functions, such as approximate string comparators [3], applied on a set of attributes, A, that generally includes names, addresses, and so on. Each compared record pair is represented by a similarity vector, $\mathbf{s}_{i,j}$, where $|\mathbf{s}_{i,j}|$ corresponds to the number of compared attributes, $|A|$. Assuming all similarities are in $[0, 1]$ (with a similarity of 0 for totally different values and 1 for an exact match), an overall normalised similarity for a pair can be calculated as $s_{i,j} = \sum_{k=1}^{|A|} \mathbf{s}_{i,j}[k]/|A|$. Each record pair is either a true match or a true non-match, where we assume the true match status is unknown for all pairs. We denote the sets of true matches and non-matches by M and N, respectively.

We also assume each record pair has a distance, $d_{i,j}$, in a specific data dimension, such as time and/or space. For example, records about people often contain addresses, and using geocoding [12] these can be used to calculate geographical distances between records. Similarly, for records that contain timestamps (such as publication records, birth, marriage or death certificates, or census records) temporal distances can be calculated between record pairs [11].

The set of compared record pairs can be represented as an undirected similarity graph, $G = (V, E)$, where each node (vertex) in V represents a record, r_i or r_j, and an edge $(r_i, r_j) \in E$ connects two records r_i and r_j if their overall similarity $s_{i,j}$ is at least a certain similarity threshold t, $s_{i,j} \geq t$, with $0 \leq t \leq 1$. The problem we aim to solve can now be defined as follows.

Definition 1 (Similarity Graph Filtering). *Given a similarity graph $G = (V, E)$, a budget β_t of the number of manual classifications of record pairs that can be conducted by an oracle, the expected number of true matches m in G, and a multiplier ϵ for the number of links to select. The aim of similarity filtering is to select a subset of record pairs $(r_i, r_j) \in E$ into a similarity graph $G_s = (V_s, E_s)$, with $V_s \subseteq V$ and $E_s \subset E$, based on manual classification of β_t record pairs in E, such that the number of matches in E_s is maximised while $|E_s| = m \cdot \epsilon$.*

Our similarity filtering approach is based on the assumption that record pairs that have a higher similarity are generally more likely to be true matches, i.e. $P((r_a, r_b) \in M | s_{a,b}) > P((r_e, r_f) \in M | s_{e,f})$ if $s_{a,b} > s_{e,f}$. While this assumption does not necessarily hold for every record pair in G, it is a common assumption used in record linkage [1, 22]. We also assume that the distances, $d_{i,j}$, of record pairs affect the values in their corresponding similarity vectors, $\mathbf{s}_{i,j}$, as is illustrated in Fig. 2. For example, the further people move the more details in their

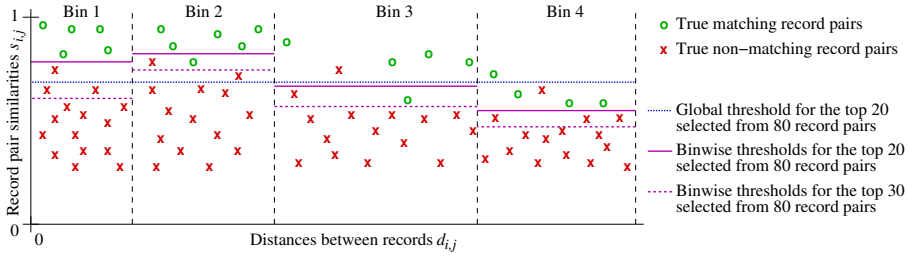

Fig. 2. Filtering of record pairs (links) with the highest similarities. Compared to using all 80 links, with $m = 20$ and $\epsilon = 1$, the filtered similarity graph contains a much smaller number of true non-matches at the cost of losing only few true matches. If the top 20 links are chosen globally (no binning), then the recall of the filtered graph is only 0.8 (4 out of 20 true matches are missed), whereas when links are chosen locally using bin specific thresholds, then recall would be 0.9 (only 2 true matches are removed by the filtering process). If we set $\epsilon = 1.5$ and select 30 record pairs then recall will be 1.

addresses will likely change. While a local move will result in a changed street address only, a move further away can also lead to changed city, zipcode, and even state values. As we discuss next, we employ a binning based active learning approach to identify different similarity thresholds for filtering on different subsets (bins) of record pairs in E using the distances $d_{i,j}$ of record pairs.

3.2 Binning based Filtering

After initialising the main data structures, in line 3 of Algo. 1 we generate the first bin $\mathbf{b}_1$ with the full similarity graph G, and set the level of this bin to $\mathbf{b}_1.l = 1$. The budget β_1, of how many record pairs are manually classified (labelled) by the human oracle in this first bin is calculated with the $CalcBudget()$ function. Due to the recursive process of splitting a bin into two in each iteration, we allocate a labelling budget that depends on the level of a bin. With a total budget of β_t, for a bin at level l we allocate a budget of $\beta_l = \beta_t/(2^{2l-1})$, such that a budget of $\beta_t/2^l$ is allocated across all bins at level l. For example, with $\beta_t = 1,000$, we will manually label $\beta_1 = 500$ record pairs in $\mathbf{b}_1$ (with level $l = 1$), $\beta_2 = 125$ in each of the two bins at level $l = 2$, $\beta_3 = 31$ in each of the four bins at level $l = 3$, and so on. Note that the set of manually labelled (classified) record pairs in a bin $\mathbf{b}$, denoted by $\mathbf{b}.\mathbf{c}$, is propagated from a parent bin to its two child bins in the recursive bin splitting process.

In line 5 of Algo 1 we calculate the optimal similarity threshold $\mathbf{b}_1.t$ corresponding to the $m \cdot \epsilon$ record pairs with the highest similarities in $\mathbf{b}_1$. In line 6 the oracle then manually classifies β_1 record pairs in bin $\mathbf{b}_1$ as $\mathbf{b}_1.\mathbf{c}$ using the $GetOracleLabels()$ function. This function conducts labelling such that both the child bins of $\mathbf{b}_1$ inherit labelled record pairs from $\mathbf{b}_1$ based on the binning method γ (which we describe below). The function selects record pairs for labelling that are close to the bin threshold $\mathbf{b}_1.t$, with $\beta_1/2$ pairs selected above

Algorithm 1: ***Binning based similarity graph filtering using active learning***

Input:
- G: Pairwise similarity graph
- β_t: Total budget (maximum number of record pairs the oracle can manually classify)
- β_m: Minimum number of manual classifications a bin must contain
- m: Expected number of true matches
- ϵ: Multiplier for number of record pairs (links) to select
- γ: Binning method (either equal width or equal depth)

Output:
- G_s: Pairwise similarity graph containing $m \cdot \epsilon$ selected links

1: $\mathbf{B} = [\,]$ // Initialise an empty list to store the final bins
2: $\mathbf{Q} = [\,]$ // Initialise a queue to hold bins to be processed further
3: $\mathbf{b}_1 = InitBin(G)$; $\mathbf{b}_1.l = 1$ // Initialise first bin with all links in $\mathbf{G}$ and set bin level to 1
4: $\beta_1 = CalcBudget(\beta_t, 1)$ // Get budget for the first bin
5: $\mathbf{b}_1.t = GetTopPairsThresh(\mathbf{b}_1, m \cdot \epsilon)$ // Get the threshold for the top $m \cdot \epsilon$ links
6: $\mathbf{b}_1.\mathbf{c} = GetOracleLabels(\mathbf{b}_1, \beta_1, \gamma)$ // Manual Classification of β_1 links in bin $\mathbf{b}_1$
7: $\mathbf{b}_1.s = CalcScore(\mathbf{b}_1)$ // Calculate the score for bin $\mathbf{b}_1$
8: $\mathbf{Q}.add(\mathbf{b}_1)$ // Add the first bin to the queue
9: **while** ($\mathbf{Q} \neq []$) **do**: // Process queue sorted by bin scores
10: $\mathbf{b}^p = \mathbf{Q}.pop()$ // Get the next (parent) bin to process based on its score
11: $\mathbf{b}^l, \mathbf{b}^r = SplitBin(\mathbf{b}^p, \gamma)$ // Split parent bin into two based on binning method γ
12: $\beta_c = CalcBudget(\beta_t, \mathbf{b}^p.l + 1)$ // Get the budget for the two child bins
13: **if** $((|\mathbf{b}^l.\mathbf{c}| + \beta_c) \geq \beta_m)$ **and** $((|\mathbf{b}^r.\mathbf{c}| + \beta_c) \geq \beta_m)$ **then**: // Enough labels in both bins
14: $\mathbf{b}^l.\mathbf{c} = \mathbf{b}^l.\mathbf{c} \cup GetOracleLabels(\mathbf{b}^l, \beta_c, \gamma)$; $\mathbf{b}^r.\mathbf{c} = \mathbf{b}^r.\mathbf{c} \cup GetOracleLabels(\mathbf{b}^r, \beta_c, \gamma)$
15: $t^l, t^r = CalcBestThresh(\mathbf{b}^l, \mathbf{b}^r)$ // Run Algorithm 2 to get optimal bin thresholds
16: **if** $(t^l == \mathbf{b}^p.t)$ **and** $(t^r == \mathbf{b}^p.t)$ **then**: // Best thresholds are same as for parent bin
17: $\mathbf{B}.add(\mathbf{b}^p)$; **go to line 9** // Add parent bin to final bin list and process next bin
18: $\mathbf{b}^l.t = t^l$; $\mathbf{b}^l.s = CalcScore(\mathbf{b}^l)$; $\mathbf{Q}.add(\mathbf{b}^l)$ // Add both child bins to queue
19: $\mathbf{b}^r.t = t^r$; $\mathbf{b}^r.s = CalcScore(\mathbf{b}^r)$; $\mathbf{Q}.add(\mathbf{b}^r)$
20: **else if** $(|\mathbf{b}^l.\mathbf{c}| + \beta_c) \geq \beta_m)$ **then**: // Only the left child bin has enough labels
21: $\mathbf{b}^l.\mathbf{c} = \mathbf{b}^l.\mathbf{c} \cup GetOracleLabels(\mathbf{b}^l, \beta_c, \gamma)$; $\mathbf{b}^l.s = CalcScore(\mathbf{b}^l)$; $\mathbf{Q}.add(\mathbf{b}^l)$
22: $\mathbf{B}.add(\mathbf{b}^r)$ // Add right child bin to final bin list
23: **else if** $(|\mathbf{b}^r.\mathbf{c}| + \beta_c) \geq \beta_m)$ **then**: // Only the right child bin has enough labels
24: $\mathbf{b}^r.\mathbf{c} = \mathbf{b}^r.\mathbf{c} \cup GetOracleLabels(\mathbf{b}^r, \beta_c, \gamma)$; $\mathbf{b}^r.s = CalcScore(\mathbf{b}^r)$; $\mathbf{Q}.add(\mathbf{b}^r)$
25: $\mathbf{B}.add(\mathbf{b}^l)$ // Add left child bin to final bin list
26: **else**: $\mathbf{B}.add(\mathbf{b}^p)$ // Add parent bin to the final bin list
27: $G_s = (V_s = \emptyset, E_s = \emptyset)$ // Initialise empty similarity graph of selected links
28: **for** $\mathbf{b} \in \mathbf{B}$ **do**: // Iterate through the bins in the final bin list
29: $G_s.insert(GetLinks(\mathbf{b}, \mathbf{b}.t))$ // Generate the final similarity graph with selected links
30: **return** G_s // Return the final similarity graph

and $\beta_1/2$ pairs below the threshold. This helps to effectively shift the bin threshold depending upon the manual labels obtained, as we discuss below. We then calculate the score $\mathbf{b}_1.s$ of bin $\mathbf{b}_1$ in line 7, where we describe four score functions in Section 3.4. These scores are used to order the queue $\mathbf{Q}$ and determine which bin to process next in the iterative phase of our approach.

We iteratively process bins in $\mathbf{Q}$ starting in line 9 as long as the queue is not empty. In line 10 we select the next (parent) bin, $\mathbf{b}^p$, with the highest score, which we then split into two child bins, $\mathbf{b}^l$ and $\mathbf{b}^r$, using the binning method γ. The function $SplitBin()$ performs either equal width or equal depth binning [8] on the parent bin $\mathbf{b}^p$ as specified by γ, using the distances $d_{i,j}$ of each record pair in $\mathbf{b}^p$. $SplitBin()$ also increases the level of the child bins as $\mathbf{b}^l.l = \mathbf{b}^p.l + 1$ and $\mathbf{b}^r.l = \mathbf{b}^p.l + 1$, propagates the optimal threshold ($\mathbf{b}^l.t = \mathbf{b}^p.t$ and $\mathbf{b}^r.t = \mathbf{b}^p.t$), and splits the set of manual classifications in $\mathbf{b}^p$ according to the binning method such that $\mathbf{b}^l.\mathbf{c} \cup \mathbf{b}^r.\mathbf{c} = \mathbf{b}^p.\mathbf{c}$.

In line 12 we calculate the oracle budget β_c for the child bins based on their level, and in line 13 we check if both child bins will contain enough manual classifications (based on their allocated budgets as well as the labels inherited from their parent). The reason for checking if a bin can have at least β_m labels is to avoid underfitting (where not enough manual labels are available in a bin to calculate an optional similarity threshold). If both bins can have β_m labels, then in line 14 we obtain new manual classifications ($\mathbf{b}^l.\mathbf{c}$ and $\mathbf{b}^r.\mathbf{c}$) for them, and in line 15 we calculate the new optimal similarity thresholds for the child bins using the function $CalcBestThres()$, as we describe in Sect 3.3. If it turns out that the optimal threshold of the parent, $\mathbf{b}^p.t$ cannot be improved (in line 16) because the distribution of the similarities of links in both child bins is homogeneous (highly similar), then we add the parent bin $\mathbf{b}^p$ to the final list of bins $\mathbf{B}$ in line 17, and go back to line 9 to process the next bin in $\mathbf{Q}$.

Otherwise, in lines 18 and 19, for each child bin $\mathbf{b}^l$ and $\mathbf{b}^r$, the threshold is set to its calculated optimal value, its bin score is calculated, and then both child bins are added to the queue $\mathbf{Q}$. On the other hand, if only one of the two child bins can have at least β_m labels, in lines 20 to 25 we obtain manual classifications for that bin, update the remaining budget and the score of that bin, and add it to $\mathbf{Q}$, while the other child bin (the one not having enough labels) is added to the final list of bins $\mathbf{B}$. If neither child bin can have at least β_m labels then in line 26 we add the parent bin $\mathbf{b}^p$ to the final list of bins $\mathbf{B}$.

Subsequent to processing all bins in $\mathbf{Q}$, we generate the filtered similarity graph of selected links (record pairs), G_s, in lines 27 to 29 by looping over all bins in $\mathbf{b} \in \mathbf{B}$, and adding all record pairs with a pairwise similarity of at least the bin threshold $\mathbf{b}.t$ into the graph G_s.

3.3 Calculating Optimal Bin Similarity Thresholds

We now describe the functionality of the $CalcBestThresh()$ function (used in line 15 in Algo. 1), as outlined in Algo. 2. The input to Algo. 2 is a bin pair $\mathbf{b}^l$ and $\mathbf{b}^r$, and the function calculates a pair of optimal thresholds, t^l and t^r, which minimise the total number of false negatives across both bins. The algorithm starts with obtaining the lists of false negatives, $\mathbf{fn}_l$ and $\mathbf{fn}_r$, in the two bins, where true matching records pairs (as manually classified by the oracle) that have a similarity below the thresholds $\mathbf{b}^l.t$ and $\mathbf{b}^r.t$ are considered as false negatives. We assume that record pairs in a bin are sorted based on their similarities. In lines 2 and 3, we then calculate the initial total number of false negatives, f_t, and initialise a list $\mathbf{S}$ with a tuple made of f_t and the initial thresholds.

The loop starting in line 4 (with $\oplus$ representing list concatenation) then shifts thresholds for each false negative record pair fn in both child bins, where the function $ShiftThresh()$ sets the threshold of one of the bins ($\mathbf{b}^l$ or $\mathbf{b}^r$) to the similarity value of fn. The threshold of the other child bin is adjusted such that the total number of record pairs with a similarity greater than the thresholds is unchanged. This ensures that we select $m \cdot \epsilon$ links at any time, despite the changing thresholds. The new thresholds t^l and t^r are returned by $ShiftThresh()$, and we then obtain the lists of false negatives $\mathbf{fn}_l$ and $\mathbf{fn}_r$ for

Algorithm 2: ***Calculate optimal bin similarity thresholds, function*** $CalcBestThres()$

Input:
- $\mathbf{b}^l, \mathbf{b}^r$: Left and right child bins

Output:
- t^l, t^r: Optimal bin threshold pair

1: $\mathbf{fn}_l = GetFalseNeg(\mathbf{b}^l, \mathbf{b}^l.t)$; $\mathbf{fn}_r = GetFalseNeg(\mathbf{b}^r, \mathbf{b}^r.t)$ // Get list of false negatives
2: $f_t = |\mathbf{fn}_l| + |\mathbf{fn}_r|$ // Get the initial total false negative count
3: $\mathbf{S} = [(f_t, \mathbf{b}^l.t, \mathbf{b}^r.t)]$ // Initialise a list with bin thresholds and total false negative count
4: **for** $fn \in \mathbf{fn}_l \oplus \mathbf{fn}_r$ **do**: // Iterate through list of all false negative record pairs
5: $t^l, t^r = ShiftThresh(\mathbf{b}^l, \mathbf{b}^r, fn)$ // Shift thresholds in child bins
6: $\mathbf{fn}_l = GetFalseNeg(\mathbf{b}^l, t^l)$; $\mathbf{fn}_r = GetFalseNeg(\mathbf{b}^r, t^r)$ // Get new false negatives lists
7: **if** $(|\mathbf{fn}_l| > f_t)$ **or** $(|\mathbf{fn}_r| > f_t)$ **then**: // One bin exceeds the false negative total
8: **break** // Stop shifting threshold in a given direction
9: **else**: $\mathbf{S}.add((|\mathbf{fn}_l| + |\mathbf{fn}_r|, t^l, t^r))$: // Add thresholds and total false negative count
10: $t^l, t^r = GetMinFalseNegThres(\mathbf{S})$ // Get thresholds with minimum total false negatives
11: **return** t^l, t^r // Return the optimal thresholds

t^l and t^r. In lines 7 and 8 we check if at least one of the bins has more false negatives than the original total false negative count, f_t, and if so we end further shifting of thresholds because no more improvement can be gained (a threshold combination that results in one of the bins having more false negatives compared to the original cannot be improved). If the condition in line 7 is not met, in line 9 we add the new total false negative count $|\mathbf{fn}_l| + |\mathbf{fn}_r|$ together with the new threshold pair t^l and t^r to the list $\mathbf{S}$. In line 10, we finally obtain the optimal bin threshold pair t^l and t^r that has a minimum total false negative count, and in line 11 we return this threshold pair.

3.4 Bin Scoring Functions

An important aspect of our recursive binning approach is the ordering of the queue $\mathbf{Q}$ based on the bin scores, $\mathbf{b}.s$, which determine how bins are being processed. Our aim is to calculate an optimal threshold for each bin such that the total number of false negatives is minimised before the budget is used up. We describe four variations of the function $CalcScore()$. In all variations we only consider the record pairs manually classified by the oracle in a given bin, $\mathbf{b}.\mathbf{c}$.

1. **False negative count** ($\mathbf{score_{fn}}$): With this approach we calculate the number of false negative record pairs contained in a bin $\mathbf{b}$, where a false negative is a pair that has been classified as a true match by the oracle and that has a similarity below the bin threshold $\mathbf{b}.t$. Using this scoring function means bins that contain more false negatives will be at the top of the queue $\mathbf{Q}$, and processed first.
2. **Bin recall** ($\mathbf{score_r}$): With this approach we calculate the recall of bin $\mathbf{b}$ as the proportion of manually classified true matches with a similarity above $\mathbf{b}.t$ over all manually classified true matches in $\mathbf{b}$. With this scoring function we process bins in $\mathbf{Q}$ such that those bin with lowest recall are processed first. This allows us to further explore bins that have fewer true positives and adjust their thresholds to improve their recall.
3. **Normalised false negative count** ($\mathbf{score_{nfn}}$): This approach is similar to the $\mathbf{score_{fn}}$ approach, except that we divide the score $\mathbf{score_{nfn}}$ by the bin size $|\mathbf{b}|$, to find the bins with the largest proportion of false negatives.

4. **Adjusted bin recall** ($\mathbf{score_{ar}}$): This approach is similar to the $\mathbf{score_r}$ function except that we adjust the original $\mathbf{score_r}$ value by dividing it by the bin size $|\mathbf{b}|$. With this approach, larger bins that have a lower bin recall value will be processed first, whereas with the $\mathbf{score_r}$ approach we order bins independent of their sizes.

We next evaluate our active learning based similarity filtering approach.

4 Experimental Evaluation

The aim of our experiments is to evaluate how applying similarity filtering before classification, as shown in Fig. 1, can improve the overall record linkage process by reducing run time and memory consumption, while at the same time improving or at least retaining linkage quality as obtained without filtering.

We evaluated our novel filtering technique using two real world data sets for which ground truth data are available. The Isle of Skye (IoS) data set [19] contains 17,614 birth records from Scotland from 1861 to 1901, where the aim is to link (cluster) all birth records (siblings) by the same mother. The North Carolina voter (NCVR) data set (see: https://dl.ncsbe.gov) contains records with personal details (such as names and addresses) of US voters from the years 2011 to 2020, from where we selected around 3 million records of voters who were represented by multiple records across several years and where at least one (likely several) of their name and/or address values changed over time (as a voter moved and/or changed their name). The number of true matches for IoS is $m = 40{,}891$ while for NCVR it is $m = 6{,}978{,}001$. For the data dimensions used for binning, we calculated time distances as the number of days between two birth records in IoS and the number of months between two voter records in NCVR, while we calculated geographical (space) distances using address geocoding [12] for IoS and the distances between zipcodes for NCVR, respectively. We will make our programs and similarity graphs available to allow repeatability.

As evaluation metrics we use precision and recall as commonly used to evaluate record linkage algorithms [9], where recall is the proportion of true matches that were correctly included in a filtered similarity graph, while precision is the proportion of true matches in a filtered similarity graph. For the final clustering results, recall measures the proportion of correctly classified true matches and precision the proportion of true matches in the set of classified matches.

As illustrated in Fig. 2, as baseline we explore a simple filtering approach using a global threshold for selecting the m record pairs (links) with the highest similarity, assuming m was provided by a domain expert. We then investigate our active learning based filtering approach, where we explore if the binning of record pairs can help improve the quality of the filtered similarity graph.

In a set of initial experiments we found that equal depth binning [8] always produced better results than equal width binning on both data dimensions time and space, and that a value of $\beta_m = 25$ for the minimum number of manual classifications per bin always gave the best results. We therefore use these parameter settings in all our experiments. We then set the total budget as

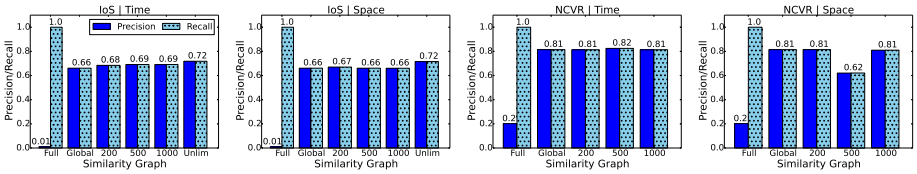

Fig. 3. Precision and recall results of the full similarity graph (unfiltered) compared with the quality of the graph filtered with a global threshold (top m links), as well as binwise thresholds for different total budgets and against different data dimensions.

$\beta_t = [200, 500, 1000]$ (as well as to *unlimited* for the IoS data set), to investigate how different budgets influence the quality of the generated filtered similarity graphs. All of the four scoring functions discussed in Sect. 3.4 produced very similar results, with the false negative count score function, $\mathbf{score_{fn}}$, obtaining slightly better results, and we therefore used this function in all our experiments.

In Fig. 3 we show the precision and recall results obtained for the original full similarity graph, as compared with the filtered graphs for the global threshold, as well as results for binning when using different total budgets. For all filtered graphs, both precision and recall values are the same because we limit the number of record pairs in the filtered graph to m, the number of true matches as estimated by a domain expert. We can see that the precision of the filtered graphs are far better compared to the original graph, since our filtering approach was able to remove a large proportion of the true non-matching record pairs.

Our aim of capturing more true matches with binning, compared to using a global threshold, has been successful mostly for the IoS data set with the time data dimension with a 4.5% maximum improvement even with a limited budget. The maximum recall improvement we can obtain with a fully supervised method (unlimited budget) is nine percent and therefore obtaining an improvement of 4.5% with a limited total budget of $\beta_t \leq 1000$ shows that our approach of using active learning can lead to improved similarity filtering. The fact that recall can be improved using bin-wise thresholds indicates that linked records in different bins do have different similarity distributions of true matching record pairs.

Our binning approach has worked well for the IoS data set on the time dimension, since for birth record pairs we find patterns such as no true matches between zero to nine months (an impossible age gap for siblings). For experiments on the space dimension on IoS, and time dimension on NCVR, the maximum recall improvement obtained is less, which indicates less distinct differences in the bins generated on these data dimensions. With NCVR in the space dimension, recall drops with $\beta_t = 500$. This indicates that certain numbers of manual classifications can lead to wrong binning due to incorrectly set classification thresholds that result in wrong selection of record pairs in some bins. Therefore, clear patterns across a data dimension are needed for our approach to work.

Table 1 shows the percentage difference in time, precision, and recall obtained with three clustering algorithms on the full similarity graphs compared to run-

Table 1. Percentage changes for time (T), precision (P), and recall (R), of clustering the filtered similarity graphs (using the time data dimension) compared to clustering the full graphs. Significant reductions in time and graph sizes ($|G|$=4 M and $|G_s|$=41 K for **IoS**, while $|G|$=34.5 M and $|G_s|$=6.9M for **NCVR**) and improvements in precision can be seen, at the costs of some reductions in recall.

		Global threshold T / P / R	$b_{tot} = 200$ T / P / R	$b_{tot} = 500$ T / P/ R	$b_{tot} = 1,000$ T / P / R
IoS	Robust [15]	-71/129/-12	-70/122/-10	-69/122/-9	-69/122/-9
	Star [6]	-98/71/-17	-98/70/-16	-98/71/-15	-98/71/-15
	Conn [6]	-100/8154/-19	-99/6519/-17	-99/5996/-17	-99/5996/-17
NCVR	Star [6]	-65/12/-16	-65/12/-16	-63/16/-14	-66/13/-16
	Conn [6]	-97/1348/-7	-97/1302/-8	-97/1379/-8	-97/1155/-8

ning these algorithms on the filtered graphs. We used three algorithms that have been used for record linkage; robust graph clustering [15], star clustering [6], and simple connected components clustering [6]. Robust graph clustering results are not shown for NCVR because we could not run this algorithm on the full NCVR graph in reasonable time. This highlights the advantage of similarity filtering to reduce the run times of computationally expensive algorithms used for linking large databases. For most experiments, precision has improved considerably with small losses in recall, while the time taken to run clustering was reduced quite significantly as well. The reduction in recall is justifiable especially for the IoS data set, where the results show the reduction in time and the improvement in precision to be more than five fold the reduction in recall. The percentage reductions in recall are slightly reduced when bin-wise thresholds are applied on the IoS data set compared to using a global threshold. Such improvements were obtained while reducing the similarity graph size from four million record pairs to only around forty thousand pairs for IoS, and a reduction from over 34 million to less than seven million record pairs for NCVR, as shown in the table.

5 Conclusions and Future Work

Record linkage is increasingly challenged by database sizes and the lack of ground truth data available in linkage applications. While blocking, indexing, and more recently meta-blocking, aim to reduce the number of record pairs that need to be compared, here we have presented a novel similarity filtering approach that removes compared pairs of records that have low similarities and are therefore unlikely true matches. Combining recursive binning of record pairs with active learning, we identify thresholds in bins that result in a substantially filtered set of record pairs while maintaining high recall of these pairs. Experiments on two real-world data sets have shown that even with a small manual labelling budget we can obtain filtered record pairs of high quality. As future work we aim to improve our method of how to select suitable record pairs for manual labelling, and we plan to incorporate the manually labelled matches and non-matches into the final clustering process using constraint clustering approaches.

References

1. Arasu, A., Götz, M., Kaushik, R.: On active learning of record matching packages. In: ACM SIGMOD. pp. 783–794. Indianapolis (2010)
2. Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. ACM TKDD **1**(1) (2007)
3. Christen, P.: Data Matching – Concepts and techniques for record linkage, entity resolution, and duplicate detection. Springer, Heidelberg (2012)
4. Christen, P., Ranbaduge, T., Schnell, R.: Linking Sensitive Data. Springer, Heidelberg (2020)
5. Dong, X.L., Srivastava, D.: Big data integration. Synthesis Lectures on Data Management, Morgan and Claypool Publishers (2015)
6. Draisbach, U., Christen, P., Naumann, F.: Transforming pairwise duplicates to entity clusters for high-quality duplicate detection. ACM JDIQ **12**(1), 1–30 (2019)
7. Efthymiou, V., Papadakis, G., Papastefanatos, G., Stefanidis, K., Palpanas, T.: Parallel meta-blocking for scaling entity resolution over big heterogeneous data. Information Systems **65**, 137–157 (2017)
8. Han, J., Kamber, M., Pei, J.: Data mining: Concepts and Techniques. Morgan Kaufmann, 3 edn. (2012)
9. Hand, D.J., Christen, P.: A note on using the F-measure for evaluating record linkage algorithms. Statistics and Computing **28**(3) (2018)
10. Hassanzadeh, O., Chiang, F., Lee, H., Miller, R.: Framework for evaluating clustering algorithms in duplicate detection. VLDB **2**(1) (2009)
11. Hu, Y., Wang, Q., Vatsalan, D., Christen, P.: Improving temporal record linkage using regression classification. In: PAKDD. pp. 561–573. Jeju, South Korea (2017)
12. Kirielle, N., Christen, P., Ranbaduge, T.: Outlier detection based accurate geocoding of historical addresses. In: AusDM. pp. 41–53. Springer, Adelaide (2019)
13. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of massive datasets. CUP (2014)
14. Mudgal, S., Li, H., Rekatsinas, T., Doan, A., et al.: Deep learning for entity matching: A design space exploration. In: ACM SIGMOD. pp. 19–34. Houston (2018)
15. Nanayakkara, C., Christen, P., Ranbaduge, T.: Robust temporal graph clustering for group record linkage. In: PAKDD. pp. 526–538. Macau (2019)
16. Papadakis, G., Skoutas, D., Thanos, E., Palpanas, T.: Blocking and filtering techniques for entity resolution: A survey. ACM Computing Surveys **53**(2), 1–42 (2020)
17. Primpeli, A., Bizer, C., Keuper, M.: Unsupervised bootstrapping of active learning for entity resolution. In: ESWC. pp. 215–231. Crete (2020)
18. Qian, K., Popa, L., Sen, P.: Active learning for large-scale entity resolution. In: ACM CIKM. p. 1379–1388. Singapore (2017)
19. Reid, A., Davies, R., Garrett, E.: Nineteenth-century Scottish demography from linked censuses and civil registers: A 'sets of related individuals' approach. History and Computing **14**(1–2), 61–86 (2002)
20. Saeedi, A., Peukert, E., Rahm, E.: Using link features for entity clustering in knowledge graphs. In: ESWC. pp. 576–592. Crete (2018)
21. Shao, J., Wang, Q.: Active blocking scheme learning for entity resolution. In: PAKDD. pp. 350–362. Melbourne (2018)
22. Tao, Y.: Entity matching with active monotone classification. In: ACM PODS. pp. 49–62. Houston (2018)
23. Vesdapunt, N., Bellare, K., Dalvi, N.: Crowdsourcing algorithms for entity resolution. PVLDB **7**(12), 1071–1082 (2014)
24. Wang, Q., Vatsalan, D., Christen, P.: Efficient interactive training selection for large-scale entity resolution. In: PAKDD. pp. 562–573. Ho Chi Minh City (2015)