Unsupervised Blocking Key Selection for Real-Time Entity Resolution

Banda Ramadan(✉) and Peter Christen

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

Abstract. Real-time entity resolution (ER) is the process of matching query records in sub-second time with records in a database that represent the same real-world entity. Indexing is a major step in the ER process, aimed at reducing the search space by bringing similar records closer to each other using a blocking key criterion. Selecting these keys is crucial for the effectiveness and efficiency of the real-time ER process. Traditional indexing techniques require domain knowledge for optimal key selection. However, to make the ER process less dependent on human domain knowledge, automatic selection of optimal blocking keys is required. In this paper we propose an unsupervised learning technique that automatically selects optimal blocking keys for building indexes that can be used in real-time ER. We specifically learn multiple keys to be used with multi-pass sorted neighbourhood, one of the most efficient and widely used indexing techniques for ER. We evaluate the proposed approach using three real-world data sets, and compare it with an existing automatic blocking key selection technique. The results show that our approach learns optimal blocking/sorting keys that are suitable for real-time ER. The learnt keys significantly increase the efficiency of query matching while maintaining the quality of matching results.

Keywords: Record linkage · Unsupervised learning · Automatic blocking · Key selection · Sorted neighbourhood indexing

1 Introduction

Massive amounts of data are being collected by most business and government organisations. Given that many of these organisations rely on information in their day-to-day operations, the quality of the collected data has a direct impact on the quality of the produced outcomes [4]. Data validation and cleaning are often employed to improve data quality [4]. One important practice in data cleaning is entity resolution (ER), which is the task of identifying records that refer to the same real-world entity.

This research was funded by the Australian Research Council (ARC), Veda, and Funnelback Pty. Ltd., under Linkage Project LP100200079.

The ER process encompasses several steps [4]: *data preprocessing*, which cleans and standardizes the data to be used; *indexing* or (*blocking*), which reduces the search space; *record comparison*, which compares candidate records in detail using a set of similarity matching functions [8]; *classification*, where pairs or groups of candidate records are classified into matches (records that are assumed to refer to the same entity) and non-matches (records that are assumed to refer to different entities); and finally, *evaluation*, where the ER process is evaluated with regard to matching accuracy and completeness [4]. Since many services in both the private and public sectors are moving online, organisations increasingly require real-time ER (with sub-second response times) on query records that need to be matched with existing entity databases [7,15].

Indexing is a vital step in the ER process especially for large databases as it reduces the number of candidate records to be compared in detail to find matching records. This can be achieved by two main approaches. The first is to partition a database to be matched into several blocks according to a *blocking key* criterion, where only records that are inserted into the same block are compared with each other [9]. The second approach is to sort the records in a database according to a *sorting key* criterion that brings similar records close to each other, so that only records that are close to each other will be compared [11].

A good indexing technique should group similar records into one block or close to each other in the index [5]. This depends mainly on the blocking/sorting key used to partition/sort the records in a database. An optimal key needs to find all the true matching records, while keeping to a minimum the number of true non-matching records. However, an optimal key for one domain will likely not work for another domain [5].

Moreover, an optimal key for batch ER might not be suitable for real-time ER, because for real-time ER we need to have small block sizes to achieve fast query matching. Selecting an optimal key needs expert knowledge of the nature of the data and the requirements of the domain. To the best of our knowledge, no existing learning technique for indexing considers real-time ER. Therefore, there is a need for novel techniques that learn optimal keys for different real-time ER domains without the need for manual intervention.

Contribution: In this paper, we propose a general learning technique that automatically selects optimal keys for building indexes to be used in real-time ER in order to find matches in a database effectively and efficiently. Our approach can be used with different indexing techniques. We demonstrate how this automatic key selection can be used with an existing sorted neighbourhood-based real-time indexing technique [19]. We learn more than one key to be used with multi-pass sorting or blocking techniques. We evaluate the proposed technique on three real-world databases and compare it with an existing technique [12].

2 Related Work

The earliest proposed indexing approach is standard blocking [9], which inserts records into blocks according to a *blocking key* criterion. This criterion is usually

based on one or more attribute values. Only records within the same block are compared with each other. This approach has the disadvantage of assigning records into the wrong block in case of errors in the attributes used as blocking keys (i.e. dirty data). To prevent this from occurring, iterative blocking [24] can be applied where multiple blocking keys are used and each record can be inserted into more than one block.

The sorted neighbourhood method (SNM) [11] arranges all records in the database(s) to be matched into a sorted array using a *sorting key*. Then a fixed-size widow is used to scan over the sorted records comparing only records within the window at any step. The main drawback of this method is its sensitivity to errors and variations at the beginning of the attribute values that are used as sorting keys, which can significantly affect the quality of the matching results [5]. This drawback is handled by performing a multi-pass approach [11] where different sorting keys are used in each pass to improve the matching quality of the approach. Various other indexing techniques that are based on either one of the above main approaches have been proposed [1,13,15,17,20,21]. However, for all of these techniques blocking or sorting keys need to be defined manually by an expert who has domain and application knowledge.

Various automatic techniques were proposed that allow learning optimal blocking/sorting keys based on supervised learning which requires the use of gold standard data for training. Bilenko et al. [2] proposed an approach that deals with the learning process as an approximation problem that is based on the red-blue set cover problem. Michelson et al. [18] proposed a related approach for learning which attributes are more suitable as blocking keys, and which similarity measures should be used for comparing these attributes.

Another supervised approach was recently proposed by Vogel and Naumann [23]. The authors use unigrams of attribute values (i.e. a combination of single characters from different attributes) as blocking keys. Both accuracy and efficiency of the generated blocks are used to learn the set of optimal blocking keys. They also improved their approach by taking the length of attribute values into consideration when generating the unigrams to be used as keys. All of the above automatic approaches require labeled training data. However, such labeled data is not always available and is usually expensive to generate.

To overcome this problem, several un-supervised automatic blocking key selection techniques have been developed [3,10,12,16]. Ma et al. [16] has proposed an approach that is based on type semantics, where the authors consider the type of entities when learning the blocking keys for data from the web. Another unsupervised learning approach was proposed by Kejriwal and Miranker [12] where the authors automatically generate a weakly labeled data set. This labeled data is then used as a training set to learn the optimal blocking keys using the Fisher discrimination criterion [12]. Giang [10] on the other hand proposed a technique that learns the blocking keys in context of the classifier function that is used in the classification step of the ER process. The classifier is used to generate labeled data. The authors then use the Probably Approximately Correct (PAC) approach to learn the blocking keys.

All mentioned unsupervised approaches focus mainly on the quality of the generated blocks and do not consider the block sizes when selecting blocking keys. However, a blocking key that can be used with real-time ER must also ensure that the sizes of the generated blocks are small enough to be able to resolve queries in real-time. In this paper, we propose an automatic blocking key selection technique that considers the coverage of a key, the maximum size of the generated blocks, as well as the distribution of the size of the generated blocks. Our aim is to learn blocking keys that are suitable for real-time ER.

3 Preliminaries and Overview

We use the following notation to present our approach. We assume a database $R = \{r_1, r_2, ..., r_{|R|}\}$, where each $r_i \in R$ contains several attributes $A = (a_1, a_2, ..., a_{|A|})$. We denote the attribute value a_j in r_i with $r_i.a_j$, where $1 \leq i \leq |R|$ and $1 \leq j \leq |A|$.

A *blocking key* (BK), denoted as $k_{j,l} = \langle a_j, f_l \rangle$, is a pair consisting of an attribute $a_j \in A$ and a blocking function $f_l \in F$, with F being the set of candidate blocking functions. Examples of such functions include exact value (isExact), same first character (sameFirst1), or same last three characters (sameLast3). The blocking function f_l is applied on attribute a_j, and the resulting value for a record r_i is called a blocking key value (BKV) and denoted as $k_{j,l}(r_i) = f_l(r_i, a_j)$. We denote the set of all candidate BKs with K. We assume the functions in F are manually selected by domain and ER experts, but for future work we aim to investigate techniques to automatically identify suitable blocking functions based on the content of a database. Our optimal key selection approach will identify the best BKs for real-time ER based on three criteria, as described in Sect. 4.

A *block* $b \in B_{j,l}$ is a set of records R_b where all $r_i \in R_b$ have the same BKV: $R_b = \{r_i \in R : k_{j,l}(r_i) = f_l(r_i, a_j)\}$. $B_{j,l}$ is the set of all blocks generated by a BK $k_{j,l}$ on all records in R.

Current approaches for learning blocking/sorting keys [3,10,12,16] do not learn keys that are suitable for real-time ER. In real-time ER, the selected BK(s) should generate block sizes within a controllable range to make sure that the number of detailed comparisons needed to match a query record (denoted as q) is within an allocated time. Also, keys that generate blocks of similar sizes are more suitable for real-time ER than keys that generate blocks of different sizes, as the time required to resolve different query records will be the same [6].

In our work, we aim to learn a list of optimal blocking/sorting keys, $O \subset K$, to be used with multi-pass indexing techniques [19] to perform ER and deliver high quality matching results in real-time. Following [12], our approach does not require existing training data sets to learn these optimal keys.

The overall framework of the proposed approach contains the following steps, as illustrated in Fig. 1. In step (1) we generate positive and negative training data sets (R_P and R_N) to be used in the learning process [12], as detailed below. In step (2) the set of candidate blocking keys K is generated. The proposed learning

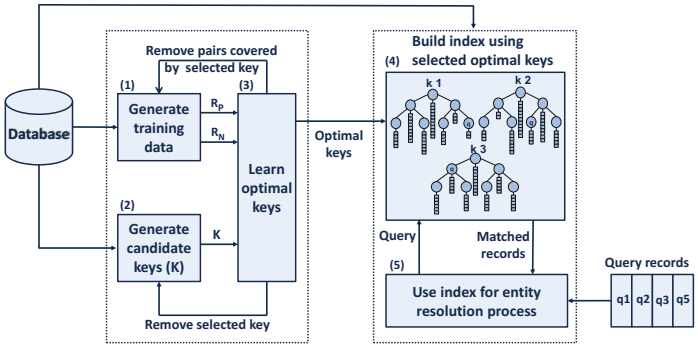

Fig. 1. Framework of the proposed approach

algorithm, described in Sect. 4, is employed in step (3) using the generated training data sets R_P and R_N to select a set of optimal blocking keys $O \subset K$. The selected optimal keys O are used in step (4) to index (block) all records from the database R. Any real-time indexing technique can be used for this step [19]. Finally, in step (5), the built index is used for matching query records q with records within the index in real-time.

3.1 Generating Training Data Sets

As in most practical applications of ER no training data sets (gold standard data) are available, such data can be generated using classification functions as in [10,12]. In this step we use Kejriwal's [12] approach to generate weakly labeled training data sets using a TF-IDF weighting scheme to calculate the similarity between record pairs $(r_x, r_y) \in R$ as follows.

A lower and upper thresholds $0 < l < u < 1$ are used to generate the training data sets. Record pairs (r_x, r_y) that have a TF-IDF similarity value $sim(r_x, r_y)$ below l are labeled as negative matches, and all pairs that have a TF-IDF value above u are labeled as positive matches. We generate a positive training set $R_P \subset R$ where the similarity between record pairs is greater than or equal to the upper threshold u: $R_P = \{r_x, r_y \in R : sim(r_x, r_y) \geq u\}$; and a negative training set $R_N \subset R$ where the similarity between record pairs is less than or equal to the lower threshold l: $R_N = \{r_x, r_y \in R : sim(r_x, r_y) \leq l\}$ with $R_P \cap R_N = \emptyset$.

Both R_P and R_N are then used to generate a set of *blocking key vectors*, V_P and V_N respectively, by applying all keys $k_{j,l} \in K$ on the record pairs in R_P and R_N [12]. Each record pair is converted into a vector of Boolean values (i.e 0 or 1 bits), with one value for each candidate key $k_{j,l} \in K$. If a record pair (r_x, r_y) in R_P or R_N for a certain candidate key $k_{j,l}$ results in having the same key value, i.e. $k_{j,l}(r_x) = k_{j,l}(r_y)$, then the corresponding element in the pair's vector is set

to 1 and the pair is said to be *covered* by this key. Otherwise, the corresponding vector element is set to 0, and the pair is said to be *uncovered* by that key.

The generated set of key vectors V_P and V_N are then used in our key selection algorithm to learn the optimal keys, as described in the following section. Alternatively, if a truth training set is available, step (1) of our framework is not required. The rest of the steps of our framework are described in more detail in the following sections.

4 Optimal Key Selection

The indexing step of real-time ER should bring similar records close to each other while maintaining small block sizes to be able to match query records in real-time. The BKs used in the indexing step have an impact on the quality and efficiency of query matching. To make sure that the keys we select are suitable for real-time ER we use three criteria:

- ***Key coverage:*** The coverage C of a key $k_{j,l}$ that is applied on record pairs (r_x, r_y) in database R is defined as the number of record pairs that evaluate to the same key value: $C_{k_{j,l}} = |\{r_x, r_y \in R : k_{j,l}(r_x) = k_{j,l}(r_y)\}|$. A key with a high coverage value leads to grouping a high number of true positive matches into the blocks generated using that key while having a minimal number of negative matches in the blocks. We use the blocking key vectors V_P and V_N (described in Sect. 3.1) to measure the coverage of a BK by calculating its Fisher score [12] as follows:

$$C_k = \frac{|V_P|(\mu_{p,k} - \mu_k)^2 + |V_N|(\mu_{n,k} - \mu_k)^2}{|V_P|\sigma_{p,k}^2 + |V_N|\sigma_{n,k}^2} \tag{1}$$

  where $\mu_{p,k}$ and $\mu_{n,k}$ are the mean of all bits generated from evaluating the key $k \in K$ on all pairs in V_P and V_N respectively, $\sigma_{p,k}^2$ and $\sigma_{n,k}^2$ are the variance of corresponding bits of k in V_P and V_N respectively, and μ_k is the mean of the corresponding bits of key k in $V_P \cup V_N$. Note that a key will have a high Fisher score if it has high coverage in V_P and low coverage in V_N. The aim of using this measure is to select keys that produce high quality blocks (with mostly true matches, and only few negative matches grouped within the generated blocks).
- ***Block Size:*** The size of a block b is the number of records that are inserted into that block and it is denoted as $S_b = |R_b|$. In this criterion we use two measures: the maximum block size denoted as $S_{b_{(max)}} = max\{S_b : b \in B\}$, and the average block size denoted as $S_{b_{(ave)}} = ave\{S_b : b \in B\}$. The aim of using the size criterion is to control the number of candidate records that are required to be compared with a query record within a desired time range.
- ***Distribution of blocks:*** For the set of blocks B generated from applying a key k on all records in database R, the distribution of k is measured by calculating the variance V_k of the sizes of all blocks in B (which reflects how far the generated block sizes are spread) using:

$$V_k = \frac{\sum_{b=1}^{|B|}(S_b - \mu_S)^2}{|B|} \tag{2}$$

where S_b is the size of a block $b \in B$ and μ_S is the mean of all block sizes in B. A variance value that is equal to 0 means that all generated blocks have exactly the same size. For real-time ER it is better to generate blocks of similar sizes where the time required to match a query record is similar for different query records. Therefore, a BK is more suitable for real-time ER if its variance of the sizes of the generated blocks is close to 0.

Generating Candidate Keys: The candidate key list is the list of all keys which we select our optimal keys from. The candidate BKs can differ based on the domain, the used indexing technique, and the databases to be matched. Because we are evaluating our key selection algorithm using a sorted neighbourhood indexing technique (as will be discussed in Sect. 5), we generate a list of candidate BKs K that capture the beginning of attribute values (i.e. isExact, sameFirst1, sameFirst2, sameFirst3, sameFirst4, sameFirst5, and concatenatedIsExact). To generate a set of K blocking keys we apply all blocking functions in F on all $r_i.a_j \in R$. The generated set of blocking keys K is given to the proposed learning algorithm along with V_P and V_N to select optimal keys as follows.

Learning Optimal Keys: Our learning algorithm (see Algorithm 1) automatically selects the list of optimal keys O based on the three criteria discussed earlier (key coverage, generated block sizes, and distribution of block sizes) to ensure that the selected keys can be used with real-time ER to provide matching results efficiently. We start the algorithm by initialising the valid key list (K_v) to be empty. This list is then filled with keys that cover less than n_m pairs in V_N, where n_m is the maximum allowed number of covered vectors from negative key vectors (lines 1-4). Then, for each key in the valid key list K_v, if the key has a maximum block size $S_{b_{max}}$ that is greater than the maximum allowed block size s_m, it is removed from K_v (lines 5-8). In lines 9-13, for all keys k left in K_v, we calculate an overall score SC_k to determine which keys should be added to the optimal key list O based on the following equation:

$$SC_k = \alpha \cdot (1 - C_k) + \beta \cdot S_{b_{(ave)_k}} + (1 - \alpha - \beta) \cdot V_k \tag{3}$$

where C_k is the coverage of k (as calculated in Equation 1), $S_{b_{(ave)_k}}$ and V_k are the average block size and the variance between the block sizes, respectively. We assume that the blocks are generated by applying the blocking key k on all records in R. The aim is to select a set of blocking keys that have high coverage, low average block size, and low variance between block sizes (note that keys with large maximum block size $S_{b(max)}$ were removed earlier from K_v in line 8). The parameters α and β are used to control the weights of the three criteria based on the domain and application area. Each weight parameter is a value between 0 and 1 where the total of all weights is equal to 1. Regardless of the weight parameters used, the lower the overall score for a key is, the more this key is suited for real-time ER.

Algorithm 1. ***LearnOptimalBK***$(V_P, V_N, K, n_m, s_m, L)$

Input:
- Positive key vectors: V_P
- Negative key vectors: V_N
- Candidate blocking keys: K
- Maximum allowed covered vectors from negative key vectors: n_m
- Maximum allowed block size: s_m
- Number of blocking keys to be selected: L

Output:
- List of optimal blocking keys: O

```
0:  while l ≤ L do
1:      K_v := [ ], O := [ ], scores := [ ]          // Initialise valid keys, optimal keys,
                                                     // and scores to be empty
2:      for k ∈ K do
3:          if k covers pairs in V_N that are < n_m then
4:              K_v.add(k)                           // Add k to the valid key list K_v
5:      for k ∈ K_v do
6:          S_b(max) := GetMaxBlockSize(k)           // Get the maximum block size for k
7:          if S_b(max) > s_m then                   // Remove keys with large block size
8:              K_v.remove(k)
9:      for k ∈ K_v do
10:         V_k := GetVariance(k)                    // Get the variance for k
11:         S_b(ave) := GetAveBlockSize(k)           // Get the average block size for k
12:         SC_k := CalcScore(V_P, V_N, S_b(ave), V_k)  // Calculate overall score for k
13:         scores.add(SC_k)                         // Add this key's score to the list of scores
14:     Sort scores ascending
15:     o := scores[0]                               // This optimal key has the lowest score value
16:     O.add(o)                                     // Add this optimal key to optimal key list
17:     V_c := getCoveredPairs(k, V_P)               // Get pairs from V_P that are covered by k
18:     V_P.remove(V_c)                              // Remove all pairs covered by k from V_P
19:     K.remove(o)                                  // Remove the selected optimal key from K
20:     l := l + 1
21: Return O                                         // Return the optimal key list
```

After calculating the overall score SC_k for all keys in K_v, these scores are sorted in an ascending order, since a lower overall score is better (line 14). The first key in the overall score list is then added to the optimal key list O (lines 15-16). In lines $17 - 18$, all positive vectors that are covered by the selected optimal key are removed from V_P, and the optimal key is also removed from K in line 19. This process continues until the required number of optimal keys L is reached or until there are no positive vectors left in V_P. The selected optimal keys O are evaluated by performing the ER process on database R using the keys selected in the indexing step as described next.

5 Experimental Evaluation

In our experiments, we use three data sets (see Table 1). The OZ data set contains personal information that is generated by randomly selecting records from an Australian telephone directory (a clean data set). Duplicates are added to this data set by randomly modifying attribute values based on typing, scanning and OCR errors, or phonetic variations [22]. Both the Cora[1] and DBLP/ACM [14] data sets contain bibliographic information and are commonly used in ER research.

[1] Available from: http://secondstring.sourceforge.net

Table 1. Data sets used in our experiments

Data set	Type	Number of records	Number of entities
OZ	Real-world (modified)	34,588	30,292
Cora	Real-world	1,295	112
DBLP2/ACM	Real-world	2,616/2,294	2,686

We use the blocking key selection approach proposed in [12] as a baseline. The authors in [12] propose a Fisher Disjunctive algorithm (FDJ) that uses the Fisher discrimination criterion to select optimal blocking keys. Unlike our approach (that considers key coverage, block sizes, and blocks distribution), this approach only considers key coverage when selecting optimal blocking keys. We compare our approach with the baseline approach using recall (the fraction of relevant instances that are retrieved) to measure the quality of the compared approaches, and query time (the time required to resolve a single query record) to measure efficiency. In addition, we generate various statistics about the number of candidate records required by both approaches to resolve a query record.

For generating the training data sets (described in Sect. 3.1) we used a lower threshold $l = 0.1$ and an upper threshold $u = 0.7$ to weakly label record pairs into positive and negative pairs. The generated training data sets are then used in our learning algorithm as described in Sect. 4. To learn the optimal blocking keys we used $n_{max} = 100$ for the maximum allowed number of covered vectors from V_N, we used a maximum block size of $sm = 100$, and for the weight parameters we used $\alpha = 0.2$ and $\beta = 0.4$. Weights and thresholds used are selected based on an experimental investigation of using different values. We aim to investigate learning these values to produce blocks with high quality and small size in our future work.

To conduct the evaluation we use the keys selected by our approach and the keys selected by the FDJ approach to build indexes that can be used to resolve query records in real-time. A real-time forest-based dynamic sorted neighbourhood index (F-DySNI) is used for this purpose [19]. The index consists of multiple tree data structures where each tree is built using a different sorting key. The F-DySNI has two phases: a build phase where index trees are built using records from an existing entity database, and a query phase where the built index can be queried by retrieving candidate records for a query record from all index trees, and the index is updated by inserting the query record.

When a query record arrives it is inserted into all trees in the index. Then, in each tree, a window of size w is used to generate a set of candidate records from the tree node that contains the query record and the neighbouring tree nodes that fall within the window w. The query is then compared (using an approximating string similarity function [4]) with the generated candidate records. Candidate records with similarities above a specific threshold are considered to be matches. We use 50% of the records in each data set to build the indexes, and the remaining records are used as query records. For generating the candidate records we use a window of size $w = 2$ (the same window size used in [19]).

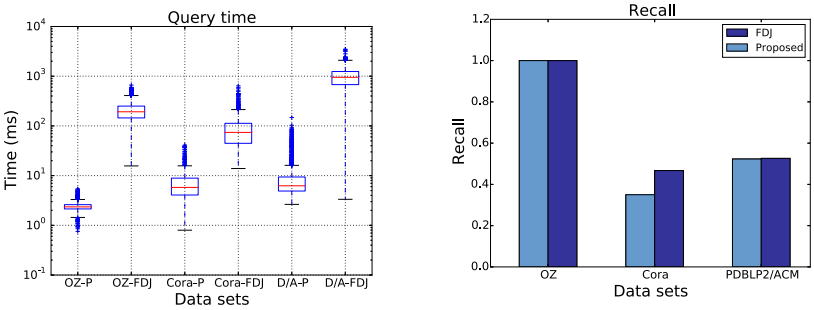

Fig. 2. Time and recall measures for the different data sets generated using the keys selected by our approach and the Fisher Disjunctive (FDJ) approach proposed in [12]. D/A in the left plot refers to DBLP/ACM and P refers to the proposed approach.

6 Results and Discussions

The aim of the experiments is to investigate if the optimal keys selected by our learning approach are suitable for real-time ER. Results in Fig. 2 illustrate the query time required to resolve a single query in milliseconds (ms) and recall values for the three data sets. The results show that the blocking keys selected by our approach improve the efficiency of query matching significantly. The selected keys using the proposed approach achieved an average query time of 2, 8, and 9 ms for the OZ, Cora, and DBLP-ACM respectively, while the selected keys using the baseline achieved an average query time of 206, 175, and 938 ms for OZ, Cora and DBKP-ACM respectively. This significant improvement in query time is achieved while maintaining recall for the OZ and DBLP-ACM but with a 5% decrease in recall value for Cora. Note that in our experiments the weight ($\alpha = 0.2$) that we use for the quality (i.e. key coverage) is half of the weight ($\beta = 0.4, \gamma = 0.4$) that we use for the block size and the distribution.

The results in Table 2 show various statistical measures for the number of candidate records generated using the proposed and the FDJ approaches using the F-DySNI with a window of size $w = 2$. It is clear from the table that the proposed approach has decreased the number of candidate records greatly which is the reason behind the significant decrease in query times.

Fig. 3 illustrates the distribution of the block sizes generated using the keys selected by the proposed and the FDJ approaches on the OZ data set (the other two data sets were too small to clearly show how the blocks are distributed). The results show that the keys selected by our approach lead to having block sizes that do not exceed the maximum allowed block size. It also shows that the generated blocks using our approach have similar sizes. In contrast, the keys selected by the FDJ approach lead to blocks of various sizes. These results are also supported by Table 2 where the standard deviation values (which measure the amount of variation from the average) of the number of candidate records required using our selected keys are small (compared to the values of the baseline) for the three data sets. This means that the block sizes tend to be close to

Table 2. Statistics for the number of candidate records generated using F-DySNI with three trees and a window $w = 2$. The keys selected by our learning approach and by the FDJ approach are used to build the trees in the index.

	OZ		Cora		DBLP-ACM	
	Proposed	FDJ	Proposed	FDJ	Proposed	FDJ
Average	10	2,041	28	274	30	2,643
Median	10	1,936	26	268	17	2,853
St.deviation	2	800	11	107	30	1,170
Minimum	6	162	10	54	10	13
Maximum	18	6,134	70	583	235	4,389

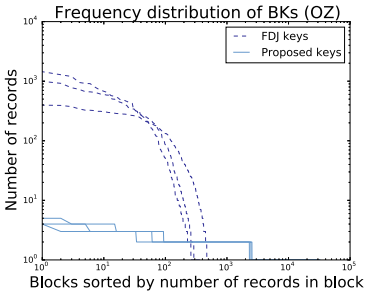

Fig. 3. Frequency distribution of the sizes of the blocks generated using the blocking keys selected by the proposed approach and the FDJ approach on the OZ data set

the average block size. We can conclude that the blocking keys selected by our proposed approach are suitable for use with real-time ER.

7 Conclusion and Future Work

We proposed an unsupervised blocking key selection algorithm that automatically selects optimal blocking keys for building indexes that can be used with real-time ER. We specifically learnt multiple keys to be used with multi-pass sorted neighbourhood indexing. We evaluated our approach using three real-world data sets and compared it with an existing automatic blocking key selection technique. The results show that our approach can learn keys that are suitable for real-time ER. The keys selected by our approach reduced query times significantly while maintaining matching quality. For future work we aim to investigate how we can automatically identify candidate blocking functions based on the content of the database. We also aim to investigate learning the weights that are used in our key selection algorithm to produce blocks with high quality and small size. Additionally, we plan to compare our proposed approach with other existing blocking key selection approaches.

References

1. Aizawa, A., Oyama, K.: A fast linkage detection scheme for multi-source information integration. In: WIRI, Tokyo (2005)

2. Bilenko, M., Kamath, B., Mooney, R.J.: Adaptive blocking: learning to scale up record linkage. In: IEEE ICDM, Hong Kong (2006)
3. Cao, Y., Chen, Z., Zhu, J., Yue, P., Lin, C.Y., Yu, Y.: Leveraging unlabeled data to scale blocking for record linkage. In: IJCAI, Barcelona (2011)
4. Christen, P.: Data Matching. Springer (2012)
5. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE Transactions on Knowledge and Data Engineering **24**(9) (2012)
6. Das Sarma, A., Jain, A., Machanavajjhala, A., Bohannon, P.: An automatic blocking mechanism for large-scale de-duplication tasks. In: ACM CIKM, Hawaii (2012)
7. Dong, X.L., Srivastava, D.: Big data integration. In: IEEE ICDE, Brisbane (2013)
8. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering **19**(1) (2007)
9. Fellegi, I., Sunter, A.: A theory for record linkage. Journal of the American Statistical Association **64**(328) (1969)
10. Giang, P.H.: A machine learning approach to create blocking criteria for record linkage. Health Care Management Science (2014)
11. Hernandez, M.A., Stolfo, S.J.: The merge/purge problem for large databases. In: ACM SIGMOD, San Jose (1995)
12. Kejriwal, M., Miranker, D.P.: An unsupervised algorithm for learning blocking schemes. In: IEEE ICDM, Dallas (2013)
13. Kim, H., Lee, D.: HARRA: fast iterative hashed record linkage for large-scale data collections. In: ICDT, Lausanne, Switzerland (2010)
14. Köpcke, H., Thor, A., Rahm, E.: Evaluation of entity resolution approaches on real-world match problems. VLDB Endowment **3**(1–2) (2010)
15. Liang, H., Wang, Y., Christen, P., Gayler, R.: Noise-tolerant approximate blocking for dynamic real-time entity resolution. In: Tseng, V.S., Ho, T.B., Zhou, Z.-H., Chen, A.L.P., Kao, H.-Y. (eds.) PAKDD 2014, Part II. LNCS (LNAI), vol. 8444, pp. 449–460. Springer, Heidelberg (2014)
16. Ma, Y., Tran, T.: Typimatch: type-specific unsupervised learning of keys and key values for heterogeneous web data integration. In: ACM WSDM, Rome (2013)
17. McCallum, A., Nigam, K., Ungar, L.: Efficient clustering of high-dimensional data sets with application to reference matching. In: ACM SIGKDD, Boston (2000)
18. Michelson, M., Knoblock, C.A.: Learning blocking schemes for record linkage. In: AAAI, Boston (2006)
19. Ramadan, B., Christen, P.: Forest-based dynamic sorted neighborhood indexing for real-time entity resolution. In: ACM CIKM, Shanghai (2014)
20. Ramadan, B., Christen, P., Liang, H.: Dynamic sorted neighborhood indexing for real-time entity resolution. In: Wang, H., Sharaf, M.A. (eds.) ADC 2014. LNCS, vol. 8506, pp. 1–12. Springer, Heidelberg (2014)
21. Ramadan, B., Christen, P., Liang, H., Gayler, R.W., Hawking, D.: Dynamic similarity-aware inverted indexing for real-time entity resolution. In: Li, J., Cao, L., Wang, C., Tan, K.C., Liu, B., Pei, J., Tseng, V.S. (eds.) PAKDD 2013 Workshops. LNCS (LNAI), vol. 7867, pp. 47–58. Springer, Heidelberg (2013)
22. Tran, K.N., Vatsalan, D., Christen, P.: Geco: an online personal data generator and corruptor. In: ACM CIKM, New York (2013)
23. Vogel, T., Naumann, F.: Automatic blocking key selection for duplicate detection based on unigram combinations. In: VLDB Workshops, Istanbul (2012)
24. Whang, S.E., Menestrina, D., Koutrika, G., Theobald, M., Garcia-Molina, H.: Entity resolution with iterative blocking. In: ACM SIGMOD, Providence (2009)