Chapter 4
A Review of Unsupervised and Semi-supervised Blocking Methods for Record Linkage

Kevin O'Hare, Anna Jurek-Loughrey, and Cassio de Campos

Abstract Record linkage, referred to also as entity resolution, is a process of identifying records representing the same real-world entity (e.g. a person) across varied data sources. To reduce the computational complexity associated with record comparisons, a task referred to as blocking is commonly performed prior to the linkage process. The blocking task involves partitioning records into blocks of records and treating records from different blocks as not related to the same entity. Following this, record linkage methods are applied within each block significantly reducing the number of record comparisons. Most of the existing blocking techniques require some degree of parameter selection in order to optimise the performance for a particular dataset (e.g. attributes and blocking functions used for splitting records into blocks). Optimal parameters can be selected manually but this is expensive in terms of time and cost and assumes a domain expert to be available. Automatic supervised blocking techniques have been proposed; however, they require a set of labelled data in which the matching status of each record is known. In the majority of real-world scenarios, we do not have any information regarding the matching status of records obtained from multiple sources. Therefore, there is a demand for blocking techniques that sufficiently reduce the number of record comparisons with little to no human input or labelled data required. Given the importance of the problem, recent research efforts have seen the development of novel unsupervised and semi-supervised blocking techniques. In this chapter, we review existing blocking techniques and discuss their advantages and disadvantages. We detail other research areas that have recently arose and discuss other unresolved issues that are still to be addressed.

K. O'Hare (✉) · A. Jurek-Loughrey
School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Belfast, UK
e-mail: kohare08@qub.ac.uk; a.jurek@qub.ac.uk

C. de Campos
Buys Ballotgebouw, Utrecht University, Utrecht, The Netherlands
e-mail: c.decampos@uu.nl

Deepak P, A. Jurek-Loughrey (eds.), *Linking and Mining Heterogeneous and Multi-view Data*, Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-030-01872-6_4

4.1 Record Linkage

Record Linkage (RL) is a process of identifying and linking pairs of records from multiple data sources (e.g. databases) representing the same real-world entity (referred to as matching records). An overview of a general RL process is demonstrated in Fig. 4.1.

Records are first standardised in order to remove inconsistencies between otherwise matching values. This may consist of removing non-alphanumeric characters or replacing multiple spelling variations of the same word with a single common spelling (e.g. "$John$", "$Johnny$", "Jon" $\rightarrow$ "$John$"). Following this, the standardised records are divided into groups (blocks) in such a manner that records within each group have a high chance of being linked in the subsequent linkage process. This is necessary to reduce the computational cost of linkage as record pair comparison is computationally expensive and grows in number exponentially with dataset sizes. Figure 4.2 depicts a simple example of a blocking process in which records are placed into blocks according to their *Surname* attribute value. In this case, the blocking process reduces the comparison space from 45 comparisons $\left(\text{i.e. } \frac{n(n-1)}{2} \text{ where } n = \text{number of records}\right)$ to only 8 comparisons, 2 of which are of matching record pairs (highlighted in green).

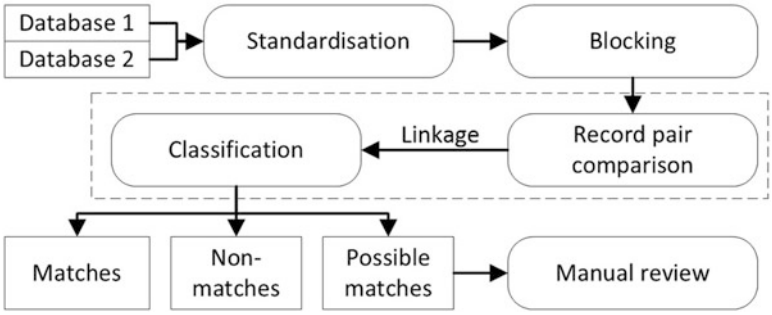

Fig. 4.1 General overview of RL process

	First Name	Surname	Postcode	Age
R1	Paul	Smyth	BT5 6AE	32
R2	Robert	Paulson	BT4 3DC	56
R3	Susan	Rivers	BT4 3DC	62
R4	Joanne	Rivers	BT2 3RD	84
R5	Neil	Smith	BT6 7ED	51
R6	Paul	Smith	BT6 5AE	32
R7	Bob	Paulson	BT4 3DC	56
R8	Joan	Rivers	BT2 3RD	83
R9	Joan	Smith	BT2 3RD	83
R10	Adam	Smyth	BT4 7AR	29

"Smyth"

R1	Paul	Smyth	BT5 6AE	32
R10	Adam	Smyth	BT4 7AR	29

"Paulson"

R2	Robert	Paulson	BT4 3DC	56
R7	Bob	Paulson	BT4 3DC	56

"Rivers"

R3	Susan	Rivers	BT4 3DC	62
R4	Joanne	Rivers	BT2 3RD	84
R8	Joan	Rivers	BT2 3RD	83

"Smith"

R5	Neil	Smith	BT6 7ED	51
R6	Paul	Smith	BT6 5AE	32
R9	Joan	Smith	BT2 3RD	83

Fig. 4.2 Example of a simple blocking scheme

Following blocking, linkage is performed exclusively on the record pairs within the same block. The blocked record pairs are compared and assigned a similarity value which informs whether they should be considered as a match, non-match or possible match requiring further manual review.

A good blocking method places as many matching record pairs, and as few non-matching record pairs into the same blocks allowing for an efficient subsequent linkage phase. Blocking methods are commonly evaluated with labelled data (with known matching status of each record pair) using evaluation metrics such as *Reduction Ratio* (RR), *Pairs Completeness* (PC) and a harmonic mean $F_{RR,PC}$ of RR and PC [21].

Definition 1 (Reduction Ratio) For two datasets, A and B, Reduction Ratio is defined as:

$$RR = 1 - \frac{N}{|A| \times |B|}, \tag{4.1}$$

where $|A| \times |B|$ is the total number of unique possible among both datasets, and $N \leq (|A| \times |B|)$ is the number of record pairs formed by a blocking method.

RR indicates how much the comparison space is reduced after the blocking phase. In the example from Fig. 4.2, the potential comparison space is reduced from 45 record pairs to 8, which equates to $RR = 1 - \frac{8}{45} = 0.8\dot{2}$.

Definition 2 (Pairs Completeness) Pairs Completeness is defined as:

$$PC = \frac{N_m}{|M|}, \tag{4.2}$$

where $N_m \leq |M|$ is the number of matching record pairs contained within the reduced comparison space after blocking, and $|M|$ is the number of matches within the entire dataset.

PC is the ratio of matching record pairs found within the formed blocks. In Fig. 4.2, there are five record pairs that are considered matches (i.e. $\{R_1, R_6\}$, $\{R_2, R_7\}$, $\{R_4, R_8\}$, $\{R_4, R_9\}$ and $\{R_8, R_9\}$) which equates to $PC = \frac{2}{5} = 0.4$.

One can notice that there is a trade-off between RR and PC. Comparing all record pairs (placing all the records in the same block) minimises RR but maximises PC, whereas performing no comparisons at all (placing each record in an individual block) maximises RR and minimises PC. Ideally, one looks for a blocking method that achieves an optimal degree of both RR and PC. A commonly applied evaluation metric, which balances the trade-off between RR and PC, is the harmonic mean of RR and PC.

Definition 3 (Harmonic Mean of RR and PC) For a given RR and PC, the harmonic mean is defined as:

$$F_{RR,PC} = \frac{2 \cdot RR \cdot PC}{RR + PC}. \tag{4.3}$$

A perfect blocking method optimises both RR and PC so that only matching record pairs are compared. To perfectly partition record pairs in such a manner is a non-trivial task. Many different automatic blocking methods have been proposed in order to address it.

Supervised approaches use labelled training data to generate highly proficient blocking methods. However, in the majority of real-world scenarios we do not have any information regarding the matching status of records obtained from multiple sources. Therefore, there is a significant demand for blocking approaches that can achieve good proficiency with little or no need for labelled data.

In the following sections, we review existing unsupervised and semi-supervised blocking methods discussing their advantages and disadvantages.

4.2 Blocking Approaches

In this section, we discuss different categories of blocking algorithms, providing examples and details on how they work as well as what advantages and disadvantages they have.

4.2.1 Standard Key-Based Blocking

Standard blocking uses a set of blocking keys to determine which records should be placed in the same block. Consider a dataset of records $R = r_1, \ldots, r_n$ where each record is represented by attributes from a scheme $A = a_1, \ldots, a_m$. Accordingly, we can represent a record r_i as $[r_{i,1}, \ldots, r_{i,m}]$, where $r_{i,j}$ is the value that the ith record takes for the jth attribute. A blocking key is defined as follows:

Definition 4 (Blocking Key) A blocking key is an $\langle a_j, h \rangle$ combination where $a_j \in A$ is an attribute and h is an indexing function. For each $r_i \in R$, h takes $r_{i,j}$ as an input and provides a set of values, referred to as blocking key values (BKV), as an output.

BKVs determine into which block(s) records are placed. Each unique BKV refers to a specific block. Due to the complexity of datasets (which may contain a variety of issues such as missing values, typographical errors, acronyms or initialisations), a single blocking key is rarely likely to capture all matching record pairs efficiently. Multiple blocking keys therefore tend to be used in the form of a blocking scheme.

Definition 5 (Blocking Schemes) Given a set of individual blocking keys, $K = k_1, \ldots, k_{k'}$, a blocking scheme is a combination of keys that are used collectively.

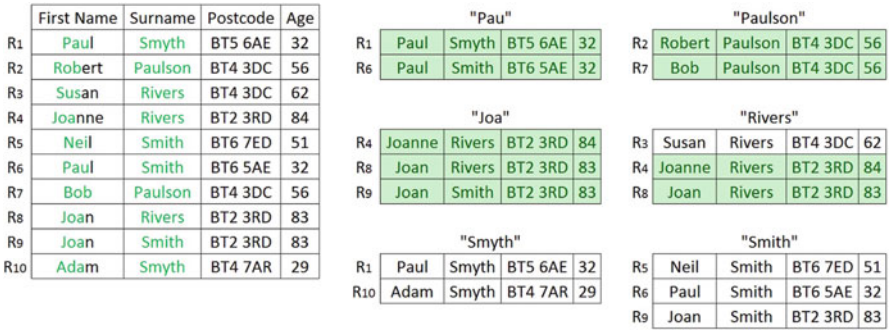

Fig. 4.3 Example of a disjunctive blocking scheme. For clarity, only those blocks that contain more than one record, and thus form at least one record pair, are presented

This combination can be disjunctive, $\langle k_i \rangle \cup \ldots \cup \langle k_j \rangle$, conjunctive, $\langle k_i \rangle \cap \ldots \cap \langle k_j \rangle$, or a disjunction of conjunctions (Disjunctive Normal Form, DNF), $\langle \langle k_i \rangle \cap \ldots \cap \langle k_j \rangle \rangle \cup \ldots \cup \langle \langle k_{i'} \rangle \cap \ldots \cap \langle k_{j'} \rangle \rangle$.

In Fig. 4.3, a disjunctive blocking scheme is applied (i.e. $\langle$*First Name, 1st 3 characters*$\rangle \cup \langle$*Surname, Exact Match*$\rangle$). This disjunctive blocking scheme captures more of the matching record pairs (highlighted in green) than that of Fig. 4.2 but at a cost of generating additional record comparisons.

Effective blocking schemes may be created manually by a domain expert [13, 16] or can be automatically learned using a blocking scheme learning (BSL) algorithm [6, 34, 47]. BSL algorithms tend to follow a common general approach in which individual blocking keys are first evaluated and then ranked according to a criteria (e.g. ratio of matches to non-matches). The top-ranked individual keys continue to form conjunctions until a satisfaction criteria is met (e.g. maximum number of record pairs formed, and maximum conjunction length). The top-ranked keys and conjunctions are then selected as a blocking scheme.

BSL algorithms have been demonstrated to be very effective [6, 34, 47] but often require labelled data in order to indicate the "best" keys and conjunctions in a supervised manner [6, 47]. The problem of labelled data requirement for evaluating blocking keys was addressed in [34]. In this work, an unsupervised BSL approach, which automatically labels its own data from the target dataset, was proposed. Records are first placed into blocks according to their commonly shared tokens (i.e. individual words). Following this, a window of predetermined size is slid over the records within each block. Record pairs within the window are then compared using the (log) Term Frequency-Inverse Document Frequency (TF-IDF) measure (Eq. (4.4)). The Log TF-IDF measure is formally defined as:

$$sim(t_1, t_2) = \sum_{q \in t_1 \cap t_2} w(t_1, q) \cdot w(t_2, q), \tag{4.4}$$

where

$$w(t,q) = \frac{w'(t,q)}{\sqrt{\sum_{q \in t} w'(t,q)^2}}, \tag{4.5}$$

and

$$w'(t,q) = \log(tf_{t,q} + 1) \cdot \log\left(\frac{|R|}{df_q} + 1\right) \tag{4.6}$$

where (t_1, t_2) represents a record pair, $w(t,q)$ is the normalised TF-IDF weight of a term q in a record t, $tf_{t,q}$ represents the term frequency of q in t, $|R|$ is the total number of records in the dataset R and df_q is the document frequency of the term q in the cohort. Predefined quantities of the most and least similar record pairs are labelled as positives and negatives, respectively. All blocking keys are applied to the labelled record pairs creating binary feature vectors (i.e. 1 or 0 if the record pair agree or disagree by the respective key). Keys that cover above a predetermined proportion of negative vectors are omitted from further consideration. If the disjunction of all permitted keys is unable to cover a predetermined proportion of the positive vectors, then a message is returned explaining that a blocking scheme cannot be learned under the current parameters. Otherwise, the keys are ranked by Fisher Score [15] and applied in descending order until the predetermined proportion of positive vectors is covered. Keys that cover new positive vectors are iteratively added to the blocking scheme. For DNF schemes, a similar approach is taken except the list of permitted keys is supplemented by conjunctions of keys restricted to a predetermined maximum conjunction length. Furthermore, only those conjunctions that have Fisher Score equal to or greater than the average Fisher Score of the permitted keys are added. The authors note that their work is the only one to cast blocking scheme learning as a feature selection problem. Kejriwal and Miranker [34] evaluate their unsupervised BSL approach against the supervised baseline approach of Bilenko et al. [6] using RR, PC and $F_{RR,PC}$. In [6], blocking keys that cover too many negatives pairs, and negative pairs covered by too many blocking keys are excluded from consideration. The remaining keys (and their conjunctions if applicable) are then ranked by their ratio of covered positives to covered negatives and applied iteratively as either a disjunctive or DNF blocking scheme. In [34], experimental evaluations indicate that their novel unsupervised BSL algorithm is seen to at least equal the performance of the supervised baseline in most cases. The authors document great success for both the automatic labelling and BSL approach for each dataset but it is worth noting that these experiments were only carried out with a small number of small datasets, the largest of which only containing 1295 records. Whether this holds for substantially larger datasets of higher dimensions is unknown. A further potential issue is that the automatic labelling approach requires comparing all record pairs of a dataset that share any

common tokens (i.e. individual words). For substantially large datasets of high dimension, this may be non-trivial to carry out.

q-Gram indexing [4, 9, 54, 56, 69] is a popular older unsupervised blocking approach that extends upon standard blocking. It is commonly used as a benchmark for other approaches to compare against as it is easy to comprehend and implement. It was first generalised in [20] and experimentally evaluated using Bi-Grams (i.e. $q = 2$) in [10]. The basic concept is that a defined blocking key is used to generate BKVs for different records of a dataset, as per the standard blocking approach, which are then split into lists of q-Grams (i.e. substrings of length q). Permutations of sub-lists are then constructed according to a threshold value from $0.0 \rightarrow 1.0$. The threshold value is multiplied by the length of the entire q-Gram list, and the rounded answer dictates the length of the permutations of q-Gram sub-lists to be constructed. The different permutations of sub-lists are then concatenated to form different blocking keys. Inverted index blocks are then created for each key with the records placed accordingly. Figure 4.4 depicts Bi-Gram indexing being applied to the same records of Fig. 4.2. In this case, the DNF blocking key $\langle$*First Name, 1st 3 characters*$\rangle \cap \langle$*Postcode, Last 3 characters*$\rangle$ is used to generate the individual BKVs. These BKVs are then split into Bi-Gram lists, and a threshold of 0.6 is used to further generate the sub-lists. As each Bi-Gram list consists of 5 Bi-Grams, this results in $5 \times 0.6 = 3$ sub-lists of Bi-grams per record. In this case, the comparison space is reduced to only four unique record pair comparisons, all of which are of matching record pairs (i.e. $\{R_4, R_8\}$, $\{R_4, R_9\}$, $\{R_8, R_9\}$ and $\{R_2, R_7\}$). q-Gram indexing of a dataset results in a maximum of $O(\frac{n^2}{b})$ comparisons where b is the number of indices or blocks created. As such, the efficiency and complexity

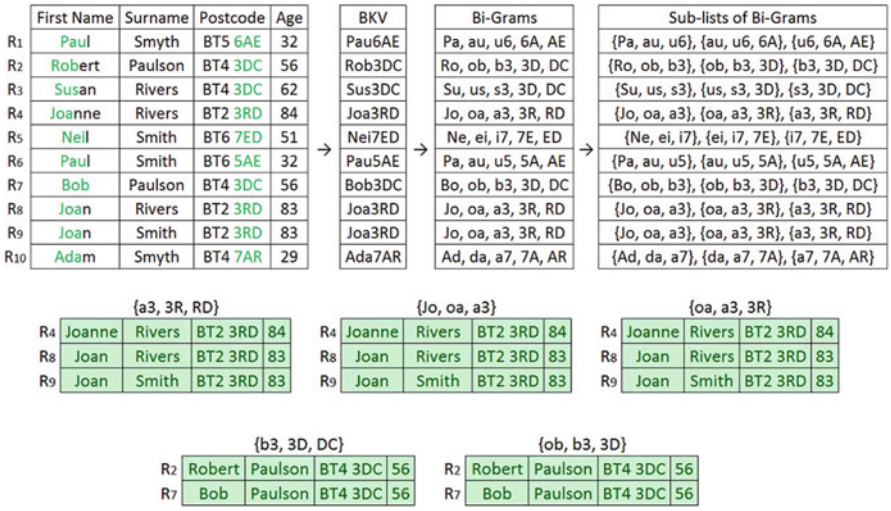

Fig. 4.4 Example of q-Gram indexing where $q = 2$ (i.e. Bi-Gram indexing). For clarity, only those blocks that contain more than one record, and thus form at least one record pair, are presented

of q-Grams indexing is very much dependent upon the value of q used. Smaller q values tend to better accommodate typographical errors between near-similar BKVs of matching records but at a cost of generating more indices which adds to the complexity. Conversely, larger values of q reduce complexity in that fewer indices are created but at a potential cost to PC. Additionally, the q-Gram sub-list generation time tends to be dominated by the recursive generation of sub-lists for longer BKVs meaning that BKV length should also be considered when selecting q [9]. By allowing records to be inserted into multiple blocks, q-Gram indexing allows for fuzzy-matching which is a desirable quality in a blocking technique.

In [4], Bi-Gram indexing is shown to be superior to the standard blocking and sorted neighbour (discussed in Sect. 4.2.3) approaches. However, in [26] q-Gram indexing was compared to a number of other blocking approaches and was found to be among the worst performing in terms of computational runtime. In [9], it is shown that it is possible to estimate the overhead cost of q-Gram indexing. In the same paper, the authors note that although q-Gram indexing tends to have higher PC than standard blocking and sorted neighbour approaches, it also has poorer RR leading to a more time-consuming linkage process. In [63], the authors state that weighting every Bi-Gram equally negatively affects performance. In [54, 56], an extended variation of q-Gram indexing is described which according to the authors improves the standard variation by generating indexing keys with higher discriminability. In doing so, they attempt to detect all of the duplicates that standard q-Gram indexing would detect but in fewer comparisons. They achieve this by concatenating different combinations of at least $L = max(1, [k \cdot T])$ sequential q-Grams of a BKV where L is derived from user-defined parameters $T \in [0, 1]$ and k, which is the number of q-Grams in the original BKV.

Another extension to standard blocking is that of Suffix-Array-based indexing [1, 7, 54, 56] in which records are blocked according to a list of suffixes of minimum length l_m derived from the respective BKV of a record by a defined blocking key. As one can imagine, the effectiveness of Suffix-Array-based indexing is dependent upon the set value of l_m. Smaller l_m values result in larger lists of suffix values meaning that each record is placed in more blocks, improving PC at a cost to RR. On the other hand, large l_m values result in shorter lists of more distinct suffix values, improving RR but at a potential negative cost to PC. If l_m is equal to the size of the BKV itself, then the approach is no different than standard blocking. Very common suffixes are often excluded from consideration as their inclusion results in extremely large blocks. In [7], Suffix-Array-based indexing is shown to have some of the lowest PC results among a number of blocking approaches for a number of synthetically generated evaluation datasets. An extended variation of Suffix-Array-based indexing is defined in [54, 56] in which all substrings of length l_m of a BKV are considered rather than just the suffixes. In [54], variations of both Suffix-Array-based indexing and its extended variation are defined in which all tokens of all values of all entities are used rather than BKVs generated from a defined blocking key.

4.2.2 Schema-Agnostic Blocking Approaches

A schema provides information regarding the attributes of a dataset (e.g. the number of attributes, column names or data types). Often, prior knowledge of a dataset schema is necessary in order to select and use the most discriminative attributes that are resistant to error or missing values and in cases of heterogeneous data sources are common to each dataset. With schema-agnostic blocking methods, no such prior information is needed. In this subsection, we detail commonly used schema-agnostic blocking approaches as well as some variations that improve upon their original implementation.

A commonly used schema-agnostic blocking approach is *Token-Blocking* [34, 36, 37, 51, 52] in which individual records are split into bags of all possible tokens (individual words) and grouped by their commonly shared tokens. Figure 4.5 depicts *Token-Blocking* applied to the same records of Fig. 4.2. *Token-Blocking* is favoured by many for a number of reasons. The basic concept is very easy to comprehend and implement and is applicable to both homogeneous and heterogeneous data sources. Matching record pairs are highly likely to share at least one common token therefore this approach tends to result in high PC in all datasets. A downside of *Token-Blocking* is that many non-matching record pairs may also contain at least one common token, which results in low RR. A variation of *Token-Blocking* improves upon this by grouping records by common token sequences of length n rather than individual tokens. This improves RR significantly but at a potential negative cost to PC. A recent research trend referred to as Meta-blocking (discussed in Sect. 4.2.6) has been used to improve the RR of *Token-Blocking* with little to no negative effect to PC.

Token-Blocking can be further improved by stipulating that the common tokens between records must be present in the same attribute column(s) in order for the records to be placed in the same block. For homogeneous cases, this is easy to implement as every record of a single dataset inherently shares the same schema. For heterogeneous cases, only those attributes that are common to each of the multiple data sources may be used. With no knowledge about the schemas, it can be difficult to tell which (if any) attributes are common to the multiple data sources. A further issue for heterogeneous cases is that attributes that refer to the same real-world values may have different column headings between data sources (i.e. "*First Name*", "*1st Name*", "*Name-1*", "*Birth Name*" or "*Given Name*"). Therefore, multiple data sources may erroneously be declared as sharing no common attributes when they in fact do but under different headings. In order to allow for improved *Token-Blocking* for heterogeneous cases, one would need to identify the common attributes without referencing the schema. Papadakis et al. [51] achieve this by combining *Token-Blocking* with attribute clustering. Attribute clustering is a form of schema matching in which a similarity value is assigned to attribute column pairs between datasets. Highly similar attribute columns are clustered together and thought of as referring to the same value. *Token-Blocking* is then performed in a

way that only records that share a common token by clustered attributes are grouped together. In the experimental evaluations of [51], attribute clustering combined with *Token-Blocking* was seen to obtain near-equal PC to that of just *Token-Blocking* but in substantially less comparisons in almost every case.

4.2.3 Sorted Neighbourhood

Sorted Neighbourhood [8, 22, 56, 74] is a simple sort and merge style unsupervised blocking method. Records are first sorted in lexicographical order according to their respective values from a user-defined sorting key (i.e. a concatenation of substring values from an attribute or number of attributes). A window of fixed size is then passed over the ordered sorting key values. Only records that fit within the window at any given time are considered in the linkage phase. As no single sorting key is likely to perfectly order the matches of a dataset adjacently, multiple sorting keys tend to be used in independent passes as part of a *Multi-Pass Sorted Neighbourhood* approach [4, 48]. Figure 4.6 demonstrates *Sorted Neighbourhood* applied to the same records of Fig. 4.2 using windows of size 3. The sorting key value in this case consists of the first letter of the *First Name* attribute value combined with the last two characters of $Postcode$ and the entire of Age (highlighted in green).

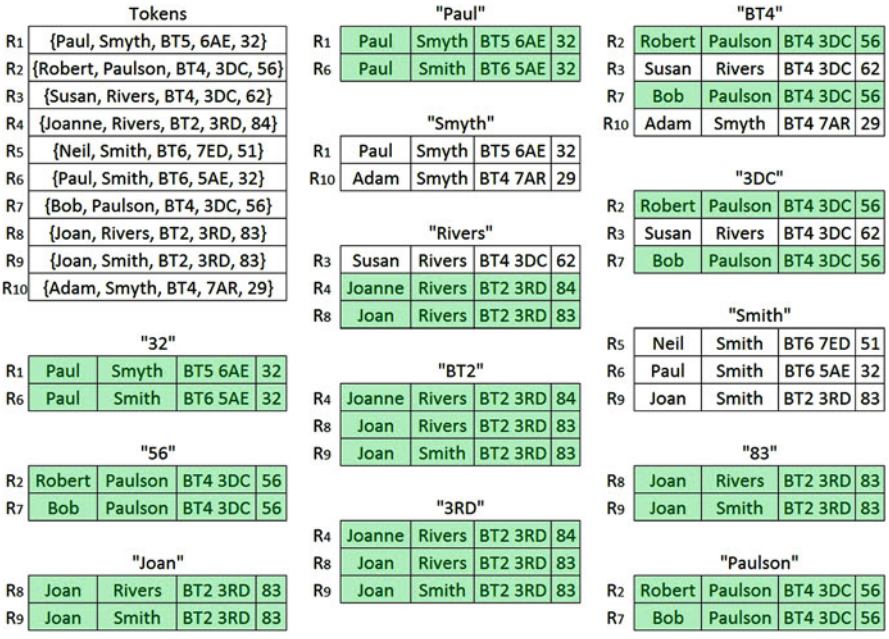

Fig. 4.5 Example of *Token-Blocking*. For clarity, only those blocks that contain more than one record, and thus form at least one record pair, are presented

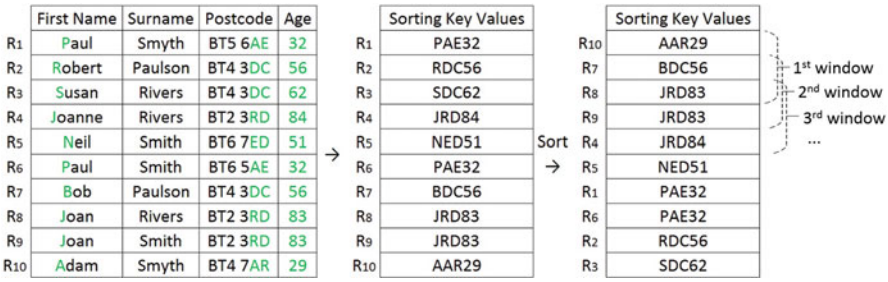

Fig. 4.6 Example of sorted neighbour

Sorted Neighbourhood is an easy to comprehend and implement blocking approach that dates back over 20 years, making it a staple go-to blocking method in the world of *RL*. In [26], *Sorted Neighbourhood* was seen to generate fewer false negatives than standard blocking and outperformed all other baseline approaches in terms of computational runtime on average. However, in the same paper *Sorted Neighbourhood* was also seen to generate more unnecessary comparison pairs than some of the other approaches. Another limitation of *Sorted Neighbourhood* is its dependency upon parameter choices such as window size and the sorting key(s). A small window size ensures a low number of comparisons but increases the risk of overlooking matching record pairs that do not happen to be positioned near one another in an ordered list. Conversely, a large window size increases the number of matching record pairs formed, but also the number of non-matching record pairs. Attributes with low diversity are poor contributors to a sorting key as many records could have the same attribute value (i.e. gender). If the number of records with the same sorting key value is substantially larger than the window size, many matching record pairs could be overlooked. In cases of sensitive data, a schema may not be available making the manual definition of a sorting key difficult. An additional downside of the *Sorted Neighbourhood* approach is the computational demand in ordering lists of sorting key values. For very large datasets, the ordering may become a computational bottleneck.

Variants of the *Sorted Neighbourhood* approach have been developed that attempt to further improve upon its efficiency and effectiveness by overcoming some of these disadvantages. One popular variant uses dynamic window sizing [14, 33, 41, 56, 61, 68, 69, 73, 75] to overcome the issues associated with using windows of a fixed size. Sorting key values are ordered lexicographically, as per the original approach, and each value is compared to those preceding it in the ordered list until a predetermined distance threshold is exceeded. Sorting key values that fall within the distance threshold are considered "close", as are their respective records. Records associated with "close" sorting key values collectively form record pairs for linkage. Dynamically sized windows are, in practice, often much smaller than a fixed window size, but increase in size when necessary.

In [58, 61], the dynamic window sizing variant of *Sorted Neighbourhood* is used to form index trees by which RL queries can be made in sub-second time. This type

of RL process is referred to as Real-Time RL and is discussed in greater detail in Sect. 4.2.7.

In [36], the authors note that *Sorted Neighbourhood* assumes that a sorting key can be applied to all records of a dataset (or datasets) as they all share the same attributes (i.e. schema). This is not always the case, such as with Resource Description Framework (RDF) data. Therefore, the authors adapt *Sorted Neighbourhood* to make it applicable. They achieve this by defining sorting keys based on the properties in the input RDF graphs, the subject Uniform Resource Identifiers (URI) in the respective triples or both of them. This is discussed in greater detail in Sect. 4.2.8.

In [57], a variation of *Sorted Neighbourhood* is proposed in which the ordered sorting key values of a dataset have a window of size 2 passed over them. Window sizes are dynamically and iteratively increased if any duplicates are detected within them. By iteratively extending window size in this manner, windows only ever become as large as necessary (i.e. until no further matches are found).

In [29], a variation of *Sorted Neighbourhood* referred to as *Sorted Neighbourhood on Encrypted Fields (SNEF)* is introduced which has the advantage of being applicable to encrypted fields. Rather than being a blocking method in its own right, it is an implementation that improves the quality of the record pairs that are generated by an independent blocking method (a concept referred to as *Meta-Blocking* which is explained in greater detail in Sect. 4.2.6). SNEF assigns each blocked record a score according to a rank function which is based on the results of the used blocking process. Records within each block are ranked accordingly, and a window of fixed size is passed over them. The intuition being that records within blocks that have similar rank scores are indicated as being especially similar. Records that fit within this window form record pairs for comparison as per the general *Sorted Neighbourhood* approach.

To better scale *Sorted Neighbourhood* for large datasets, it has been combined with the *MapReduce* framework [39, 40, 44, 46]. During the *Mapping* stage of the MapReduce framework, the workload of finding sorting key values for different records is partitioned so that it can be ran in parallel by many computers, rather than sequentially by a single computer. During the subsequent *Reduce* stage, the records are partitioned to each computer by sorting key values and then reduced into pairs that share the same sorting key value.

Collectively, these variations overcome many of the disadvantages of *Sorted Neighbourhood*, but parameter selection and the need for a domain expert tend to remain an issue. In many cases, there is still a reliance upon a domain expert to define the, potentially multiple, sorting keys that are needed. In [5], the automatic learning of sorting keys was explored but assumes the existence of labelled data. Additionally, although the dynamic window sizing approaches avoid selection of a fixed window size, there is still a need to select a distance threshold value by which the windows change size dynamically. This entails many of the same issues associated with selecting a fixed window size, albeit to a much lesser extent.

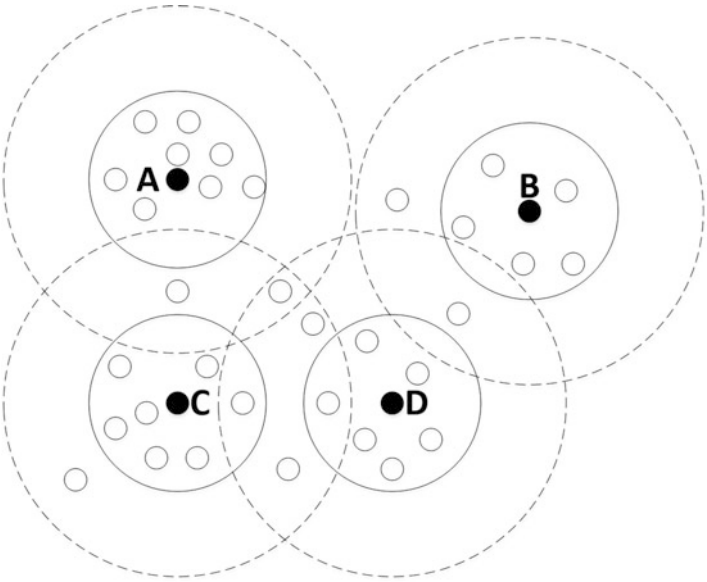

Fig. 4.7 Example of canopy clustering

4.2.4 Clustering-Based Blocking Methods

Clustering is a process by which records are divided into groups so that each record is similar to the records within the same cluster (similar according to a given similarity function) and dissimilar to records from different clusters. Overlapping clusters may be constructed if records are permitted to be assigned to more than one cluster. Adapted versions of the general clustering algorithm have been used as an alternative to blocking or to aid part of the blocking process.

Canopy Clustering [3, 6, 32, 35, 45] is an adapted version of the general clustering algorithm, that forms overlapping clusters of similar records. In *Canopy Clustering*, a record is chosen (potentially at random) as a centroid, and all other records within a *loose* distance threshold of it form a canopy. The centroid of a canopy and all records within a $tight$ distance threshold of it form an inner circle of especially similar records, that are excluded from becoming the centroid of any new canopy. Additional (non-excluded) records are chosen as centroids, and new canopies are formed in the same manner. This is best illustrated in Fig. 4.7 by centroids A, B, C and D having an inner circle of $tight$ distance threshold records and outer circle of $loose$ distance threshold records (i.e. $threshold_{loose} > threshold_{tight}$). Note that the $tight$ records of the different canopies never overlap. This ensures welcome overlapping of diverse canopies that cover many records collectively.

A disadvantage of *Canopy Clustering* is that its efficiency and effectiveness are very much reliant upon appropriate selection of parameters, namely the

loose and *tight* distance threshold values and the number of centroids. In [4], *Canopy Clustering* is shown to outperform standard blocking and sorted neighbourhood when appropriate parameters are selected; however, selection of such parameters can be difficult. Overly, loose distance threshold values may cause very large clusters to be generated resulting in poor RR and exceptionally long linkage runtime. On the other hand, selecting overly strict distance threshold values can result in very small clusters to be generated, which may fail to capture many of the matches within a dataset resulting in poor PC. *Canopy Clustering* complexity is dependent upon the selected number of centroids, as every other record must be compared to each of the centroids during the *Canopy Clustering* process. A smaller number of centroids therefore increase the efficiency of the clustering process, but at a potential cost to RR as large clusters may be generated. Conversely, a larger number of centroids may improve PC, especially if overlapping clusters are permitted, but at a cost to clustering efficiency. Therefore, poor parameter selection may result in the formation of a small number of large clusters (poor RR) containing most matching pairs (good PC), or a large number of small clusters (good RR) containing few matching pairs (poor PC). Determining which records to use as centroids is equally important. If multiple records referring to the same real-world entity are used as centroids of different clusters, then many matching record pairs may be overlooked, especially if clusters are not allowed to overlap.

In [18], a clustering approach based on the string similarity between BKVs is used to iteratively form blocks with regulated size. Between each iteration, a block quality and block size trade-off is used to determine which overly small blocks should be merged in order to improve *PC* and which overly large blocks should be further partitioned in order to improve *RR*. The authors acknowledge that although their BSL approach takes longer than that of the baselines, they achieve superior RR because of the regulated block sizes. They also detail that in their evaluations they manually determined the blocking keys, and the order in which they should further partition the overly large blocks. Automatic learning of blocking keys was left for future work.

4.2.5 Locality-Sensitive Hashing

Locality-Sensitive Hashing (LSH)-based blocking approaches [11, 19, 25, 31, 38, 42, 50, 64, 67, 71] use several functions selected from a family of hash functions to convert records into hash key values of a fixed size and placed into buckets of similar records according to their respective hash key values. By stipulating that the family of hash functions are locality-sensitive to a distance metric d, one ensures that the probability of generating the same hash key value by a hash function is dependent upon the distance between both records. Therefore, similar records are likely to generate the same hash key value by a particular hash function. In this section, we describe LSH when used with Jaccard Similarity, but other variants of LSH may be adapted for use with other distance metrics.

Definition 6 (Jaccard Similarity) The Jaccard Similarity of two sets of elements, A and B, is defined as the ratio of the size of their intersection to the size of their union (i.e. $\frac{A \cap B}{A \cup B}$)

Records are first divided into sets of substrings of length k referred to as k-shingles, a similar process to that of *q-Gram Indexing* discussed in Sect. 4.2.2. Just like with *q-Gram Indexing*, a desirable k value is one small enough so that typographical errors between near-similar record pairs are accounted for, but also large enough so that the generated shingle sets are of reasonable size so complexity does not become an issue.

For large datasets, storing all shingle sets of the respective records would be inefficient as there would be many repeated elements of length k. Instead, the shingle sets are used to form a sparse Boolean matrix in which the rows represent the universal shingle set (with each element thought of as a hash function, $h()$) and the columns represent each record. 1s and 0s are used to represent if the respective record of a column does or does not contain the respective shingle element of each row. Therefore, highly similar columns contain 1's for many of the same hash functions; which also means that they, and their respective records, have high Jaccard Similarity. In Fig. 4.8, we see a Boolean matrix constructed for the same records as that of Fig. 4.2 when using shingles of size $k = 2$ (i.e. Bi-Grams). For clarity, only a small sample of the universal 2-shingle set for this case is presented.

	First Name	Surname	Postcode	Age	2-shingles
R1	Paul	Smyth	BT5 6AE	32	{Pa, au, ul, Sm, my, yt, th, BT, T5, 5_, _6, 6A, AE, 32}
R2	Robert	Paulson	BT4 3DC	56	{Ro, ob, be, er, rt, Pa, au, ul, ls, so, on, BT, T4, 4_, _3, 3D, DC, 56}
R3	Susan	Rivers	BT4 3DC	62	{Su, us, sa, an, Ri, iv, ve, er, rs, BT, T4, 4_, _3, 3D, DC, 62}
R4	Joanne	Rivers	BT2 3RD	84	{Jo, oa, an, nn, ne, Ri, iv, ve, er, rs, BT, T2, 2_, _3, 3R, RD, 84}
R5	Neil	Smith	BT6 7ED	51	{Ne, ei, il, Sm, mi, it, th, BT, T6, 6_, _7, 7E, ED, 51}
R6	Paul	Smith	BT6 5AE	32	{Pa, au, ul, Sm, mi, it, th, BT, T6, 6_, _5, 5A, AE, 32}
R7	Bob	Paulson	BT4 3DC	56	{Bo, ob, Pa, au, ul, ls, so, on, BT, T4, 4_, _3, 3D, DC, 56}
R8	Joan	Rivers	BT2 3RD	83	{Jo, oa, an, Ri, iv, ve, er, rs, BT, T2, 2_, _3, 3R, RD, 83}
R9	Joan	Smith	BT2 3RD	83	{Jo, oa, an, Sm, mi, it, th, BT, T2, 2_, _3, 3R, RD, 83}
R10	Adam	Smyth	BT4 7AR	29	{Ad, da, am, Sm, my, yt, th, BT, T4, 4_, _7, 7A, AR, 29}

	Records									
2-shingles	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
32	1	0	0	0	0	1	0	0	0	0
56	0	1	0	0	0	0	1	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
3D	0	1	1	0	0	0	1	0	0	0
an	0	0	1	1	0	0	0	1	1	0
Sm	1	0	0	0	1	1	0	0	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
th	1	0	0	0	1	1	0	0	1	1
yt	1	0	0	0	0	0	0	0	0	1

Fig. 4.8 Example of a Boolean matrix

Different hash functions from a Boolean matrix can be combined to form hash table functions by which records can be placed into hash tables. A single hash table function cannot be expected to perfectly partition a dataset into hash tables. Therefore, multiple hash table functions are often required in order to achieve good blocking results.

LSH is attractive as a blocking approach for a number of reasons. It can have time complexity as low as $O(n)$, making it highly scalable for even large and highly dimensional datasets. LSH may also be leveraged over similarity spaces (e.g. semantic similarity, and textual similarity) to improve block quality by removing repeated or redundant pairs without negatively affecting accuracy [71]. As individual records are converted into Boolean vectors the approach lends itself to privacy-preserving record linkage. With only alphanumeric characters (i.e. $A \rightarrow Z$ and $0 \rightarrow 9$) and spaces, the maximum number of unique k-shingles is limited to $(26 + 10 + 1)^k$. However, in reality the actual number is often much smaller as many k-shingles are unlikely to occur in real-world datasets.

Manual parameter setting is often needed when using LSH (e.g. k for k-shingling, and L for the number of hash table functions). The authors of [38] note that despite LSH's fast and effective performance it suffers from a number of drawbacks that make it unsuitable for large-scale record linkage. In particular, they highlight the memory requirement during the hash table generation process and the high runtime for datasets with many duplicates. Other papers have modified or adapted LSH as part of more complex blocking approaches that aim to overcome some of these drawbacks. A simple Hamming distance-based LSH variation is described in [19, 31, 42]. In this variation, values in randomly and uniformly chosen ith positions of the universal shingle set are used to construct hash table functions. In the same papers, the authors describe a Euclidean distance-based variation of LSH, in which vectors of records are projected onto a Euclidean space allowing for nearby points to be grouped together. This concept of mapping entries to a multidimensional vector space has been well documented in older works such as FastMap [17], StringMap [27], MetricMap [70] and SparseMap [24].

In the proceeding subsection, we give special attention to a particular variation of LSH that has been adopted in a considerable number of recent research efforts.

4.2.5.1 LSH with MinHashing

$MinHashing$ allows for the formation of a much smaller representation of a Boolean matrix, that still allows for a comparable estimation of the Jaccard similarity of two records by their respective columns. The $MinHash$ function, $h(C)$, takes the value of the first row in which 1 appears in a column C. Applied to the Boolean matrix of Table 4.1, we get $\{1, 3, 1, 1\}$ for columns $\{C_1, C_2, C_3, C_4\}$, respectively (note 1 appears in the top row of columns C_1, C_3 and C_4 but in the third row for column C_2). If the rows of a Boolean matrix are randomly permuted ($Orderings$ in Table 4.1), the probability that a row's hash function assigns two columns the same $MinHash$ value is approximately the same as their Jaccard

Table 4.1 Example of a Boolean matrix converted into a signature matrix

	Boolean matrix				Orderings								
Shingles	C_1	C_2	C_3	C_4	O_1	O_2	O_3						
s_1	1	0	1	1	2	2	4						
s_2	1	0	1	1	3	10	6			Signature matrix			
s_3	0	1	0	0	6	4	3			C_1	C_2	C_3	C_4
s_4	1	0	1	1	5	3	7		O_1	1	4	1	1
s_5	0	1	0	0	4	7	5	→	O_2	2	1	1	2
s_6	1	0	1	1	1	5	1		O_3	1	3	1	1
s_7	1	0	1	1	7	8	2						
s_8	0	1	1	0	8	1	9						
s_9	0	1	0	0	9	9	10						
s_{10}	0	1	0	0	10	6	8						

Similarity. $MinHash$ signatures are generated for each record by concatenating the individual $MinHash$ values for each record across multiple random row order permutations. In other words, the number of permutations dictates the length of the $MinHash$ signature value.

The Jaccard Similarity of a pair of $MinHash$ signatures (fraction of $MinHash$ values in which they agree) is approximately equal to that of the Jaccard similarity of their respective columns. Similar $MinHash$ signatures are therefore indicative of similar record pairs. The higher the number of permutations used to create the $MinHash$ signatures, the lower the expected error. In Table 4.1, C_1 and C_4 have identical $MinHash$ signatures, and looking at the Boolean matrix we see that this is also the case for their Boolean vectors. Blocking only the pairs with exactly matching signatures would be naive as even near-similar record pairs may have different $MinHash$ values for some hash functions (e.g. $\{C_1, C_3\}$ and $\{C_3, C_4\}$). Instead, the columns of the signature matrix are separated into l bands of m $MinHash$ values each. Columns containing the same $MinHash$ vector in each band are then blocked together as they match for that part of the signature.

MinHash LSH has an advantage over standard LSH in that different hash table functions no longer require definition. Instead, the $MinHash$ function is used iteratively upon a Boolean matrix with its rows permuted between each iteration. l and m values require definition for banding the signature matrix. However, these are arguably easier parameters to set with than the large number of hash table functions that potentially exist as part of the basic LSH approach. This is especially true for cases in which the universal shingle set is of considerable size.

MinHash LSH has been used in a considerable number of recent research papers [11, 31, 64, 71] due to it being fast, effective and relatively easy to implement. In [64], the authors note that due to the semi-structured nature of data of the Web, some data source schemas can potentially consist of thousands of attributes. They acknowledge that the comparison of all attribute pairs for the purpose of scheme matching is impracticable and, therefore, propose a *MinHash LSH*-based

preprocess to reduce this comparison space significantly. The set of attributes is represented as a matrix where each column is a vector of the attributes determined by a weight function. By iteratively permuting the rows of the matrix and concatenating the different $MinHash$ values for each column, $MinHash$ signatures can be constructed for the attributes represented by each column. The $MinHash$ signatures of each attribute are then partitioned into l bands of size m. Only the attributes of columns that agree by at least one $MinHash$ signature band are then passed onto a subsequent attribute-match induction algorithm, thus reducing the comparison space significantly. In [11, 71], *MinHash LSH* is combined with semantic constraints in order to better pair records based on both their textual and conceptual similarity. In adding an additional filter by which record pairs must agree, the comparison space formed by *MinHash LSH* is further reduced with little to no negative impact upon PC. This improves RL quality overall. In [31], LSH was combined with a homomorphic matching technique in order to allow for privacy-preserving record linkage. The authors of this paper also state that in their experimental evaluation, a Hamming distance variation of the LSH approach was able to achieve superior results than that of the popular *MinHash LSH* variation. In [38], the authors extend *MinHash LSH* to form an iterative match-merge LSH approach that reuses a single hash table, greatly reducing the hash table generation time. As records are placed into their respective $MinHash$ buckets, they are checked against any already placed records. If the newly placed record contains some or all of an already placed record, a merge or replacement occurs accordingly. This process iteratively continues (for all records) until a termination condition is met; that is, the reduction ratio of the ith iteration is below a predetermined threshold. Every match-merge step that is performed results in both a smaller record set and hash table. This is reflected in the respective experimental evaluations in which this extended variation was observed to operate up to 5.6 times faster than standard LSH whilst maintaining equivalent accuracy.

4.2.6 Meta-blocking and Progressive-Blocking

Meta-Blocking methods [12, 30, 49, 51, 53, 55, 64] aim to improve the efficiency of how linkage is performed following a blocking process. This may be achieved by restructuring the blocks prior to linkage (*Meta-Blocking*), or managing how they are processed during linkage (*Progressive-Blocking*), rather than by simply performing all record pair comparisons arbitrarily.

Meta-Blocking improves the linkage efficiency of a block collection by prioritising the most promising blocks or record pairs. In [51], the authors observe that a record paired with many other records within a block collection is unlikely to match with any of them during linkage. To omit such record pairs from the linkage process would therefore be useful as the number of comparisons could be reduced with negligible cost to recall. *Meta-Blocking* approaches commonly accomplish this by building a graph to represent the entity-to-block relationships of a block collection.

In such a graph, nodes represent the individual records of a block collection, and edges between records indicate co-occurrence of these records in at least one block. Information from this graph can then be leveraged to indicate the most similar record pairs within the blocks. In [51, 53, 55, 64], *Meta-Blocking* has been used to further improve the RL efficiency of the unsupervised and schema-agnostic $token$ blocking approach by only performing linkage upon blocks or record pairs indicated as most promising. The linkage process then continues until a cost/gain approximation indicates finding further matches to be too costly. In [64], the authors combine attribute clustering, $token$ blocking and *Meta-Blocking* to form an unsupervised blocking approach they refer to as the "BLAST Approach". The blocking and *Meta-Blocking* stages are evaluated with Pairs Quality (PQ) (Eq. (4.7)) used as a surrogate for precision alongside PC and $F_{PQ,PC}$, where tp is the number of true positives (matches correctly classified as matches) and fp is the number of false positives (non-matches incorrectly classified as matches).

$$PQ = \frac{|tp|}{|tp| + |fp|}. \tag{4.7}$$

In their experimental evaluations, PQ was seen to greatly improve, whilst maintaining high PC. Their approach was observed to perform equal to or better than other blocking and *Meta-Blocking* approaches.

Progressive-Blocking techniques aim to learn and adapt as they progress so as to improve the likelihood of detecting additional duplicates with minimal additional effort. A progressive *Sorted Neighbourhood* approach [57] was described earlier in Sect. 4.2.3 in which small windows that contain duplicates are extended in size in the hope of detecting further duplicates. Iterative blocking [23, 52, 55, 72] is another progressive approach in which any record pair classified as a match during linkage is merged and distributed to all other blocks containing either record, replacing the original record in the process. This implementation likely improves PC, because what would have been previously overlooked matches now have a higher chance of being detected due to the transitive relation of matching record pairs. Efficiency also tends to improve in a progressive convergent manner as the repeated merging, distribution and replacement of individual records with merged records in earlier blocks saves the overall processing time of subsequent blocks.

4.2.7 Blocking for Real-Time RL

The approaches presented so far in this chapter describe unsupervised and semi-supervised blocking methods for batch RL of structured relational datasets. In batch RL, the general RL process from Fig. 4.1 is enacted in its entirety upon a dataset or datasets. Ideally, placing most matching record pairs in blocks with few non-matching pairs so that the computational demand of the subsequent linkage process is reduced. Real-Time RL, on the other hand, involves a query record

being matched against records from possibly multiple, typically disparate, large data sources. For batch RL, runtime can be expected to be considerable if the datasets are of substantial size or complexity, but in real-time RL queries are ideally processed in sub-second runtime. In this section, we detail some real-time RL blocking techniques that help achieve this goal.

In [43], an LSH blocking approach (Sect. 4.2.5) is made scalable to large-scale dynamic datasets and query-able by combining it with a dynamic sorting tree. In this approach, the records of large LSH blocks are sorted, and a window is passed over them in order to return the nearest neighbours of a query record. The authors note that as *MinHash LSH* involves a random permutation of rows when forming LSH blocks, a predefined fixed sorting key may result in an entire block being returned as a query result. To overcome this issue, the authors define a scoring function by which one or several attributes that contain many unique and well-distributed values may be selected as a sorting key. In their experimental evaluations, the authors demonstrate that combining LSH with dynamic sorting trees results in much lower query times than that of LSH alone. They further state that their approach can be effective and efficiently used for large-scale real-time RL cases. This is particularly true for noisy data, as LSH operates on a k-Gram level.

In [58], a forest-based sorted neighbourhood indexing technique is proposed that aims to facilitate real-time RL by using multiple distinct index trees with different sorting keys. In this approach, an initial $build$ phase constructs index trees using records from an existing database. A subsequent $query$ phase then allows for the built index to be queried and all relevant entries to be retrieved with the index updated in the process. The tree data structure consists of braided trees, referred to as AVL trees [62], that combine a height balanced binary tree property with a double-linked list. Every node in the tree is linked to its alphabetically sorted predecessor and successor nodes, with a list of identifiers of all records that have the respective nodes key value as their sorting key value. The nodes of the tree are sorted alphabetically by these sorting key values where a sorting key value is generated for each record in the database, and a record identifier is inserted into the tree based on the sorting key value. Record identifiers of records with matching sorting key values are listed together to the same node as a list. This allows for sorting key value searching to be reduced from $O(\log(n))$, as per array-based indexing, to $O(\log(k))$ given that there are k different nodes in a tree, n records in a dataset and $k < n$. Multiple different sorting key values are used during the build phase to construct multiple trees with records inserted into every tree where applicable as a node. A query record is inserted into all of the respective nodes of the different trees according to the sorting key used to generate them. For a single tree, all candidate records of all nodes within a window to the node containing the query record are considered candidate matches to the query record. This repeats for all trees forming a collective set of candidate matches to the query record. In their experimental evaluations, the authors experiment with both fixed and adaptive window size. The authors state that by using multiple trees blocking is noticeably improved over use of single trees with only a minor increase to the average insertion and query time. In the worst case, the achieved times are still considerably fast, that

is ∼1 ms insertion time and ∼15 ms query time, respectively, when using three trees. In their experimental evaluations, they found that their approach was able to perform over one order of magnitude faster than that of a similar baseline approach [60]. The same authors of [58] improve upon this work in another paper [59] by combining it with automatic learning of keys used for constructing the multiple trees. They achieve this by generating weakly labelled training data using the approach of [34] which was described earlier in Sect. 4.2.1. With their weakly labelled data, they evaluate individual keys and select those deemed most appropriate for real-time RL according to a scoring function that considers three factors: key coverage, generated block sizes and distribution of block sizes. The ideal keys are those that have high coverage, low average block size and low variance between block sizes. In their experimental evaluations, they evaluate the real-time RL approach of their previous paper using keys selected by this technique, against those selected by the blocking key selection technique of [34]. Note that in the baseline paper optimal keys were selected for standard blocking not real-time RL. They find that their selection technique chooses significantly better keys for real-time RL than the baseline in terms of query time, with comparable recall for two of the three evaluation datasets and only a 5% decrease over the baseline for the third dataset.

4.2.8 Blocking for RDF Data

Resource Description Framework (RDF) is a series of specifications for conceptually describing or modelling information among Web resources. Web information may be represented as an RDF triple. That is, a statement in the form of a *Subject–Predicate–Object* expression where a $Subject$ is connected to an $Object$ by a $Predicate$ (a verb) that details the type of relationship between the two. $Predicates$ may take multiple values unlike attributes in relational RL [2]. Representing Web information resources in this manner allows for their blocking and interlinking among multiple knowledge bases. Much recent work has focused on the blocking and linking of RDF data, a process commonly referred to as instance matching or link discovery.

In [32], a two-step blocking scheme learner for scalable linkage discovery is presented. With the proposed approach, an optional unsupervised dataset mapping is first performed between a pair of dataset collections (arbitrary mixes of RDF and tabular datasets). Matrices are generated using the dot product of the normalised term (individual token) frequency vectors of two datasets. Pairs between collections are then mapped according to a confidence function of the max Hungarian algorithm [28] applied to a respective matrix. Following this, Mapped pairs proceed to a second step which learns a link-discovery blocking scheme. RDF datasets are reconciled with the tabular datasets by representing them as property tables. From this point onwards, the problem is reduced to the more common problem of learning effective blocking schemes for tabular datasets with differing schemas. The authors use the same BSL approach as that of their earlier work in [34]

(discussed in Sect. 4.2.1) and incorporate bagging (random sampling) in case there are an insufficient number of training examples. In cases where there are no training examples at all, they suggest using the automatic training set generation algorithm presented in the same earlier paper. In their experimental evaluations, perfect mapping was achieved in all three test sets. Furthermore, their approach outperformed the baseline regardless of whether it was completely unsupervised or provided with a perfectly labelled training set. As this approach incorporates much of [34], it has many of the same advantages and disadvantages that were discussed in Sect. 4.2.1.

In [65, 66], an approach is presented that selects candidate pairs by discriminating keys chosen using domain-independent unsupervised learning. The same authors of both papers present this work for RDF, but state in [66] that it generalises to structured datasets too. Starting with a set of triples, all data-type $Predicates$ and instances are extracted. For each $Predicates$, all $Object$ values are retrieved, and three metrics ($Discriminability$, $Coverage$ and $F_{Dis,Cov}$) are computed. $Discriminability$ gives an indication of the diversity of a $Predicate$ and is defined as the ratio of its number of unique respective $Object$ values to its total number of respective $Object$ values. Any $Predicates$ with $Discriminability$ less than a pre-defined threshold value are immediately omitted from consideration as many instances have the same $Object$ values on this $Predicate$ which indicates poor RR. $Coverage$ is the ratio of the number of instances a key covers to that of the total number of instances, indicating the $Coverage$ of a $Predicate$. $F_{Dis,Cov}$ is the harmonic mean of $Discriminability$ and $Coverage$ and is used to indicate how well a $Predicate$ achieves a balance of being highly discriminative whilst covering a large number of instances. The $Predicate$ with the highest $F_{Dis,Cov}$ value is selected as the candidate selection key, provided that it is higher than a predetermined threshold value. If no suitable $Predicate$ is found, then conjunctions of $Predicates$ are iteratively explored until one is. Triples are indexed according to the selected key and when selecting candidate pairs only those with *n-Gram* similarity above a pre-defined threshold value are returned. In both papers, the proposed method outperforms the baselines but the authors admit to a number of potential problems. Firstly, they cannot be sure what effect values such as phone numbers may have on the size of the returned candidate sets. This is due to how their approach retrieves candidate pairs based on common *n-Gram*'s, and is never evaluated using numerical data. Given data primarily describing instances in the same geographic area, one can expect many phone numbers to share the same prefix; therefore, oversized candidate sets would be expected. Another issue they state is that as their algorithm looks for exact matching on query tokens when looking up similar instances within the index that not all co-referent instances for a given instance may be retried in poor quality datasets. They leave addressing this issue with fuzzy matching as an area for future research.

4.3 Conclusion

In this chapter, we discussed different categories of unsupervised and semi-supervised blocking methods giving examples of each and stating their advantages and disadvantages. A number of simple blocking approaches were initially presented that have been used for many years now. These simple approaches have been shown to perform especially well when a domain expert with intrinsic knowledge of the target data is at hand as they can effectively set the necessary parameters.

It was also discussed that under different circumstances (e.g. sensitive data, lack of domain expert, RDF data and RL queries) the simple blocking approaches may no longer be suitable therefore requiring more advanced approaches specific to these circumstances. For each circumstance, it was explained how different advanced blocking approaches came to be and are applicable where simple blocking approaches are not. The ultimate goal of unsupervised blocking research is for the development of unsupervised blocking approaches that are completely free of parameters. The approaches presented in this chapter may be unsupervised but operating completely parameter-free persists as a goal to be achieved. Areas of research that have arose as a result of blocking research were also discussed, namely *Meta-Blocking*, *Progressive Blocking* approaches and blocking approaches for RDF data. As datasets become increasingly larger and we look to the Web for their storage, one can predict that these particular areas of research will become increasingly relevant. With such exponential growth, the development of unsupervised blocking approaches for exceptionally large-scale data is expected to continue as a relevant topic of research.

References

1. Aizawa, A., Oyama, K.: A fast linkage detection scheme for multi-source information integration. In: Proceedings of International Workshop on Challenges in Web Information Retrieval and Integration, WIRI'05, pp. 30–39. IEEE, Piscataway (2005)
2. Atencia, M., David, J., Scharffe, F.: Keys and pseudo-keys detection for web datasets cleansing and interlinking. In: International Conference on Knowledge Engineering and Knowledge Management, pp. 144–153. Springer, Berlin (2012)
3. Babu, B.V., Santoshi, K.J.: Unsupervised detection of duplicates in user query results using blocking. In: International Journal of Computer Science and Information Technologies, **5**(3), 3514–3520. IJCSIT (2014)
4. Baxter, R., Christen, P., Churches, T., et al.: A comparison of fast blocking methods for record linkage. In: ACM SIGKDD, vol. 3, pp. 25–27. Citeseer (2003)
5. Bertolazzi, P., De Santis, L., Scannapieco, M.: Automatic record matching in cooperative information systems. In: Proceedings of the International Workshop on Data Quality in Cooperative Information Systems (DQCIS), p. 9 (2003)
6. Bilenko, M., Kamath, B., Mooney, R.J.: Adaptive blocking: learning to scale up record linkage. In: Sixth International Conference on Data Mining, ICDM'06, pp. 87–96. IEEE, Piscataway (2006)

7. Christen, P.: Improving data linkage and deduplication quality through nearest-neighbour based blocking. In: Department of Computer Science, The Australian National University. ACM (2007)
8. Christen, P.: Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer, Berlin (2012)
9. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans. Knowl. Data Eng. **24**(9), 1537–1555 (2012)
10. Christen, P., Churches, T., et al.: Febrl-freely extensible biomedical record linkage. In: Department of Computer Science, Australian National University (2002)
11. Cui, M.: Towards a scalable and robust entity resolution-approximate blocking with semantic constraints. In: COMP8740: Artificial Intelligence Project, Australian National University (2014)
12. dal Bianco, G., Gonçalves, M.A., Duarte, D.: Bloss: effective meta-blocking with almost no effort. Inf. Syst. **75**, 75–89 (2018)
13. Bilenko, M.: Learnable similarity functions and their application to clustering and record linkage. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence, pp. 981–982 (2004)
14. Draisbach, U., Naumann, F., Szott, S., Wonneberg, O.: Adaptive windows for duplicate detection. In: 2012 IEEE 28th International Conference on Data Engineering (ICDE), pp. 1073–1083. IEEE, Piscataway (2012)
15. Duda, R.O., Hart, P.E., Stork, D.G., et al.: Pattern Classification, 2nd edn, p. 55. Wiley, New York (2001)
16. Elfeky, M.G., Verykios, V.S., Elmagarmid, A.K.: Tailor: a record linkage toolbox. In: Proceedings of 18th International Conference on Data Engineering, pp. 17–28. IEEE, Piscataway (2002)
17. Faloutsos, C., Lin, K.I.: FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets, vol. 24. ACM, New York (1995)
18. Fisher, J., Christen, P., Wang, Q., Rahm, E.: A clustering-based framework to control block sizes for entity resolution. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 279–288. ACM, New York (2015)
19. Gionis, A., Indyk, P., Motwani, R., et al.: Similarity search in high dimensions via hashing. In: VLDB'99 Proceedings of the 25th International Conference on Very Large Data Bases, vol. 99, pp. 518–529 (1999)
20. Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D., et al.: Approximate string joins in a database (almost) for free. In: Proceeding VLDB '01 Proceedings of the 27th International Conference on Very Large Data Bases, vol. 1, pp. 491–500 (2001)
21. Guillet, F., Hamilton, H.J.: Quality Measures in Data Mining, vol. 43. Springer, Berlin (2007)
22. Hernández, M.A., Stolfo, S.J.: Real-world data is dirty: data cleansing and the merge/purge problem. Data Min. Knowl. Disc. **2**(1), 9–37 (1998)
23. Herschel, M., Naumann, F., Szott, S., Taubert, M.: Scalable iterative graph duplicate detection. IEEE Trans. Knowl. Data Eng. **24**(11), 2094–2108 (2012)
24. Hristescu, G., Farach-Colton, M.: Cluster-preserving embedding of proteins. Technical Report 99-50, Computer Science Department, Rutgers University (1999)
25. Ioannou, E., Papapetrou, O., Skoutas, D., Nejdl, W.: Efficient semantic-aware detection of near duplicate resources. In: Extended Semantic Web Conference, pp. 136–150. Springer, Berlin (2010)
26. Isele, R.: Learning expressive linkage rules for entity matching using genetic programming. Ph.D. thesis (2013)
27. Jin, L., Li, C., Mehrotra, S.: Efficient record linkage in large data sets. In: Proceedings of the Eighth International Conference on Database Systems for Advanced Applications, DASFAA '03, p. 137. IEEE Computer Society, Washington (2003). http://dl.acm.org/citation.cfm?id=789081.789250

28. Jonker, R., Volgenant, T.: Improving the Hungarian assignment algorithm. Oper. Res. Lett. **5**(4), 171–175 (1986)
29. Karakasidis, A., Verykios, V.S.: A sorted neighborhood approach to multidimensional privacy preserving blocking. In: IEEE 12th International Conference on Data Mining Workshops (ICDMW), pp. 937–944. IEEE, Piscataway (2012)
30. Karakasidis, A., Koloniari, G., Verykios, V.S.: Scalable blocking for privacy preserving record linkage. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 527–536. ACM, New York (2015)
31. Karapiperis, D., Verykios, V.S.: An LSH-based blocking approach with a homomorphic matching technique for privacy-preserving record linkage. IEEE Trans. Knowl. Data Eng. **27**(4), 909–921 (2015)
32. Kejriwal, M., Miranker, D.P.: A two-step blocking scheme learner for scalable link discovery. In: CEUR Workshop Proceedings. Vol 1317, pp. 49–60 (2014)
33. Kejriwal, M., Miranker, D.P.: N-way heterogeneous blocking. Tech. rep., TR-14-06 (2013)
34. Kejriwal, M., Miranker, D.P.: An unsupervised algorithm for learning blocking schemes. In: IEEE 13th International Conference on Data Mining (ICDM), pp. 340–349. IEEE, Piscataway (2013)
35. Kejriwal, M., Miranker, D.P.: On linking heterogeneous dataset collections. In: International Semantic Web Conference (Posters & Demos), pp. 217–220. Citeseer (2014)
36. Kejriwal, M., Miranker, D.P.: Sorted neighborhood for schema-free RDF data. In: International Semantic Web Conference. pp. 217–229. Springer, Berlin (2015)
37. Kejriwal, M., Miranker, D.P.: An unsupervised instance matcher for schema-free RDF data. Web Semant. Sci. Serv. Agents World Wide Web **35**, 102–123 (2015)
38. Kim, H.S., Lee, D.: Harra: fast iterative hashed record linkage for large-scale data collections. In: Proceedings of the 13th International Conference on Extending Database Technology, pp. 525–536. ACM, New York (2010)
39. Kolb, L., Thor, A., Rahm, E.: Parallel sorted neighborhood blocking with MapReduce (2010). arXiv:1010.3053
40. Kolb, L., Thor, A., Rahm, E.: Multi-pass sorted neighborhood blocking with MapReduce. Comput. Sci. Res. Dev. **27**(1), 45–63 (2012)
41. Lehti, P., Fankhauser, P.: Unsupervised duplicate detection using sample non-duplicates. Lect. Notes Comput. Sci. **4244**, 136 (2006)
42. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Cambridge University Press, Cambridge (2014)
43. Liang, H., Wang, Y., Christen, P., Gayler, R.: Noise-tolerant approximate blocking for dynamic real-time entity resolution. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 449–460. Springer, Berlin (2014)
44. Ma, K., Yang, B.: Parallel NoSQL entity resolution approach with MapReduce. In: International Conference on Intelligent Networking and Collaborative Systems (INCOS), pp. 384–389. IEEE, Piscataway (2015)
45. McCallum, A., Nigam, K., Ungar, L.H.: Efficient clustering of high-dimensional data sets with application to reference matching. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 169–178. ACM, New York (2000)
46. Mestre, D.G., Pires, C.E., Nascimento, D.C.: Adaptive sorted neighborhood blocking for entity matching with MapReduce. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 981–987. ACM, New York (2015)
47. Michelson, M., Knoblock, C.A.: Learning blocking schemes for record linkage. In: AAAI'06 Proceedings of the 21st National Conference on Artificial Intelligence, pp. 440–445 (2006)
48. Naumann, F., Herschel, M.: An introduction to duplicate detection. Synth. Lect. Data Manage. **2**(1), 1–87 (2010)
49. Papadakis, G., Palpanas, T.: Blocking for large-scale entity resolution: challenges, algorithms, and practical examples. In: IEEE 32nd International Conference on Data Engineering (ICDE), pp. 1436–1439. IEEE, Piscataway (2016)

50. Papadakis, G., Ioannou, E., Niederée, C., Palpanas, T., Nejdl, W.: Eliminating the redundancy in blocking-based entity resolution methods. In: Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries, pp. 85–94. ACM, New York (2011)
51. Papadakis, G., Ioannou, E., Palpanas, T., Niederee, C., Nejdl, W.: A blocking framework for entity resolution in highly heterogeneous information spaces. IEEE Trans. Knowl. Data Eng. **25**(12), 2665–2682 (2013)
52. Papadakis, G., Koutrika, G., Palpanas, T., Nejdl, W.: Meta-blocking: taking entity resolution to the next level. IEEE Trans. Knowl. Data Eng. **26**(8), 1946–1960 (2014)
53. Papadakis, G., Papastefanatos, G., Koutrika, G.: Supervised meta-blocking. Proc. VLDB Endowment **7**(14), 1929–1940 (2014)
54. Papadakis, G., Alexiou, G., Papastefanatos, G., Koutrika, G.: Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. Proc. VLDB Endowment **9**(4), 312–323 (2015)
55. Papadakis, G., Papastefanatos, G., Palpanas, T., Koubarakis, M.: Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. In: 19th International Conference on Extending Database Technology, pp. 221–232 (2016)
56. Papadakis, G., Svirsky, J., Gal, A., Palpanas, T.: Comparative analysis of approximate blocking techniques for entity resolution. Proc. VLDB Endowment **9**(9), 684–695 (2016)
57. Papenbrock, T., Heise, A., Naumann, F.: Progressive duplicate detection. IEEE Trans. Knowl. Data Eng. **27**(5), 1316–1329 (2015)
58. Ramadan, B., Christen, P.: Forest-based dynamic sorted neighborhood indexing for real-time entity resolution. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, pp. 1787–1790. ACM, New York (2014)
59. Ramadan, B., Christen, P.: Unsupervised blocking key selection for real-time entity resolution. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 574–585. Springer, Berlin (2015)
60. Ramadan, B., Christen, P., Liang, H., Gayler, R.W., Hawking, D.: Dynamic similarity-aware inverted indexing for real-time entity resolution. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 47–58. Springer, Berlin (2013)
61. Ramadan, B., Christen, P., Liang, H., Gayler, R.W.: Dynamic sorted neighborhood indexing for real-time entity resolution. J. Data Inf. Qual. **6**(4), 15 (2015)
62. Rice, S.V.: Braided AVL trees for efficient event sets and ranked sets in the SIMSCRIPT III simulation programming language. In: Western MultiConference on Computer Simulation, San Diego, pp. 150–155 (2007)
63. Shu, L., Chen, A., Xiong, M., Meng, W.: Efficient spectral neighborhood blocking for entity resolution. In: IEEE 27th International Conference on Data Engineering (ICDE), pp. 1067–1078. IEEE, Piscataway (2011)
64. Simonini, G., Bergamaschi, S., Jagadish, H.: Blast: a loosely schema-aware meta-blocking approach for entity resolution. Proc. VLDB Endowment **9**(12), 1173–1184 (2016)
65. Song, D., Heflin, J.: Scaling data linkage generation with domain-independent candidate selection. In: Proceedings of the 10th International Semantic Web Conference (ISWC) (2013)
66. Song, D., Heflin, J.: Automatically generating data linkages using a domain-independent candidate selection approach. In: International Semantic Web Conference, pp. 649–664. Springer, Berlin (2011)
67. Steorts, R.C., Ventura, S.L., Sadinle, M., Fienberg, S.E.: A comparison of blocking methods for record linkage. In: International Conference on Privacy in Statistical Databases, pp. 253–268. Springer, Berlin (2014)
68. Tamilselvi, J.J., Gifta, C.B.: Handling duplicate data in data warehouse for data mining. Int. J. Comput. Appl. **15**(4), 1–9 (2011)
69. Tamilselvi, J., Saravanan, V.: Token-based method of blocking records for large data warehouse. Adv. Inf. Mining **2**(2), 5–10 (2010)

70. Wang, J.T.L., Wang, X., Lin, K.I., Shasha, D., Shapiro, B.A., Zhang, K.: Evaluating a class of distance-mapping algorithms for data mining and clustering. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 307–311. ACM, New York (1999)
71. Wang, Q., Cui, M., Liang, H.: Semantic-aware blocking for entity resolution. IEEE Trans. Knowl. Data Eng. **28**(1), 166–180 (2016)
72. Whang, S.E., Menestrina, D., Koutrika, G., Theobald, M., Garcia-Molina, H.: Entity resolution with iterative blocking. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, pp. 219–232. ACM, New York (2009)
73. Whang, S.E., Marmaros, D., Garcia-Molina, H.: Pay-as-you-go entity resolution. IEEE Trans. Knowl. Data Eng. **25**(5), 1111–1124 (2013)
74. Winkler, W.E.: Overview of record linkage and current research directions. Bureau of the Census. Citeseer (2006)
75. Yan, S., Lee, D., Kan, M.Y., Giles, L.C.: Adaptive sorted neighborhood methods for efficient record linkage. In: Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries, pp. 185–194. ACM, New York (2007)