Neighbourhood Blocking for Record Linkage

Daniel Elias
New South Wales Data Analytics Centre
Email: daniel.elias@treasury.nsw.gov.au

***Abstract*—This paper describes Neighbourhood Blocking – a novel method for the indexing step in the record linkage process. This method combines advantages of Standard Blocking and Sorted Neighbourhood Indexing with some slight generalizations that allow meaningful treatment of missing values and a limited number of blocking field mismatches. Comparison of the Cartesian product of the blocks is avoided by imposing matching criteria restrictions which enable a recursive implementation. Numerical experiments and tests on benchmark datasets are reported in which Neighbourhood Blocking is compared to Standard Blocking and Sorted Neighbourhood Indexing. Under the conditions tested, Neighbourhood Blocking is found to frequently produce superior index quality, often at the expense of increased runtime. Scale testing indicates that index production speeds for Neighbourhood Blocking and Standard Blocking are similar when the database size is sufficiently large.**

Keywords*—*algorithm, blocking, conflation, coreference resolution, data integration, data linkage, data matching, deduplication, disambiguation, duplicate detection, entity resolution, identity resolution, indexing, list washing, merge/purge, name resolution, neighbourhood blocking, object identification, object identity problem, propensity score matching, record linkage, record matching, record resolution, reference reconciliation, sorted neighbourhood indexing

I. Motivation

A. Overview

The primary motivation for combining aspects of Standard Blocking and Sorted Neighbourhood Indexing is to produce a single inedxing method with the following features:

- Inclusion of all pairs of records closer to one another than some threshold distance
- Use of multiple fields to assess record similarity

As outlined in sections I-B and I-C, Standard Blocking lacks the first of these features and and Sorted Neighbourhood Indexing lacks the second. This suggests that a hybrid method might combine them. As Theorem IV.1 shows, Neighbourhood Blocking achieves this.

B. An issue with Standard Blocking

Standard Blocking is similar to a database join in that it matches record pairs where the values of certain specified fields ("blocking keys") are equal. A strength of this approach compared to Sorted Neighbourhood Indexing is that it simultaneously constrains the differences between values in multiple fields.

However, one disadvantage of Standard Blocking is that the only selection criterion for record pairs is whether or not they are in the same block. Any meaningful notions of block proximity or position *within* blocks (eg: when the blocks are discretized versions of continuous variables) are ignored. Take for example, the eight points illustrated in Figure 1. Ideally, an indexing method that includes pairs of points that are far apart (for example, pair AD) should also include all pairs of points in the central cluster (ie: DE, EF and DF).

Depending on the positions of block boundaries, this will happen in some cases but not others. Consider the case illustrated in Figure 2 where a single block boundary divides the points into two horizontal blocks. Points A, B, C, D and E are grouped together in the lower block and points F, G and H in the upper one. Critically, however, the block boundary separates point F from points D and E meaning that the relatively close pairs DF and EF are omitted, while relatively distant pairs such as AC and GH are included.

The same problem occurs (with different pairs of points) with the vertical blocking illustrated in

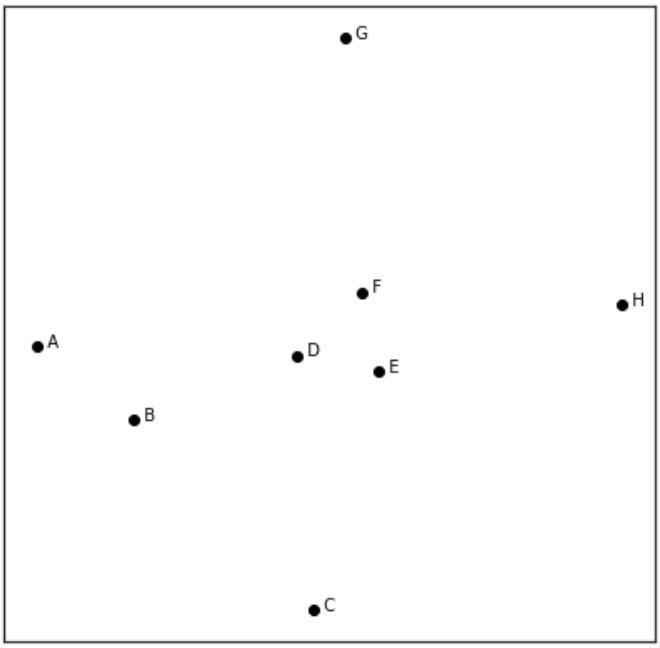

Fig. 1: Example points for indexing illustrations

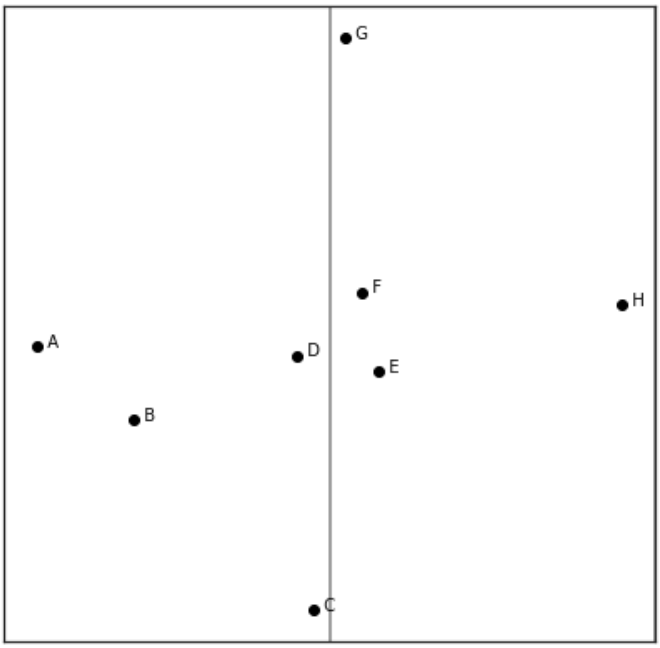

Fig. 3: Vertical blocks

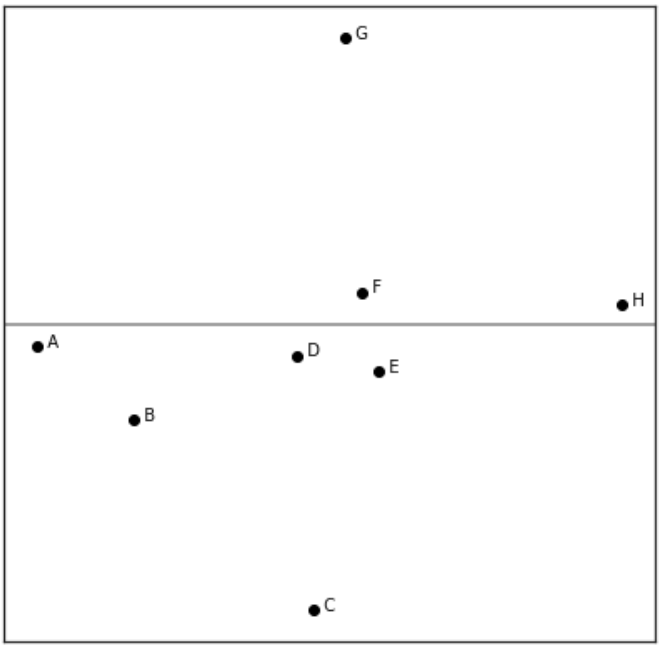

Fig. 2: Horizontal blocks

Figure 3. In this case, the block boundary separates point D from points E and F.

Since the horizontal and vertical block boundaries in Figures 2 and 3 are problematic for different sets of point pairs, it may be tempting to simply take the union of the indexes produced using the horizontal and vertical blocks. This doesn't solve the problem because in addition to including many distant pairs of points, pair DF is still excluded.

Extending the blocking to both variables also doesn't solve the problem. In fact in this case, it exacerbates it. As Figure 4 illustrates, all the points in the central cluster are separated by block boundaries, meaning that none of the closest pairs in this (albeit contrived) dataset will be included in the index.

The problems illustrated in the examples above also apply in non-trivial datasets. Figure 5 shows the relationship between recall and "total boundary size" for various configurations of Standard Blocking applied to the simulated dataset used in Section VI-B (all fields are numeric and differences between BKVs for true matches are normally distributed). Here, "recall" represents the proportion of true matches present in the index, and "total boundary size" is the total volume of key space hyperplanes corresponding to block boundaries. One striking feature of Figure 5 is that at the moderate to high levels of recall shown (the region of interest

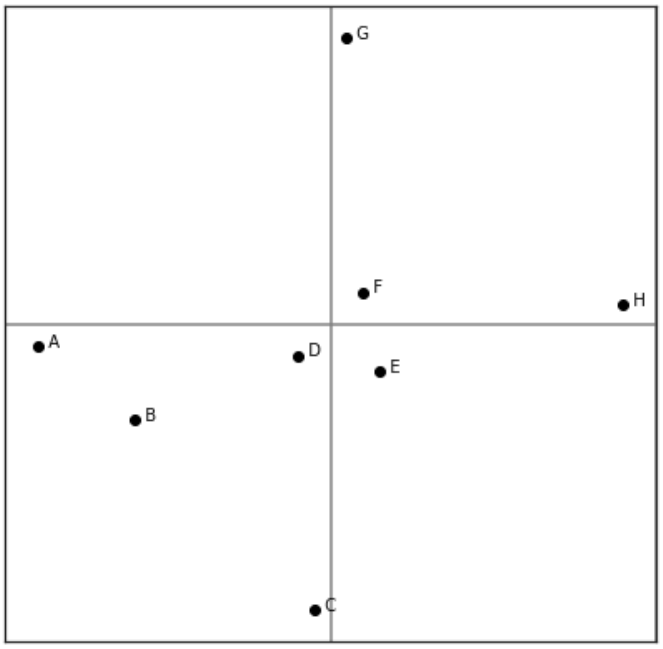

Fig. 4: Bivariate blocks

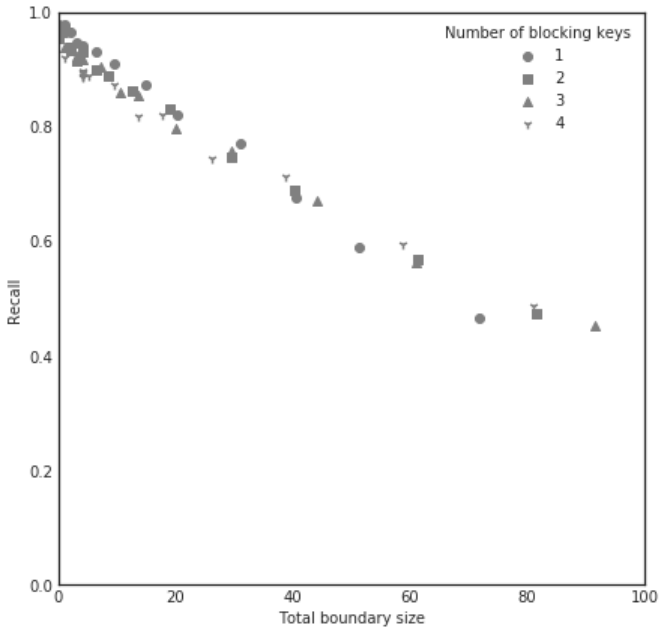

Fig. 5: Standard Blocking: Recall vs Boundary Size (simulated)

in indexing), recall is almost entirely determined by total boundary size and the relationship is approximately linear (linear regression of recall against boundary size in the region shown produces an R^2 of 0.97). This indicates that when the boundaries are far apart, there is an approximately constant "cost" of adding more block boundary. An intuitive explanation of this is that the choice of blocking keys determines both the size of the boundaries and their orientation. However, when boundaries are far apart (compared to the distance between points in matching pairs), each unit of boundary has an equal chance of dividing some matching points, regardless of its orientation.

C. An issue with Sorted Neighbourhood Indexing

Sorted Neighbourhood Indexing produces an ordering of the distinct combinations of values in certain fields ("sorting keys") and returns all record pairs whose combinations of sorting key values are closer than a specified distance in that ordering. Since the "windows" used to select records for pairing can overlap , the problem (described above) of nearby pairs straddling block boundaries doesn't apply in Sorted Neighbourhood Indexing.

However, one weakness of Sorted Neighbourhood Indexing is that it uses only a single ordering of the records, causing it to typically include more distant record pairs than, say, Standard Blocking with multiple blocking keys.

To illustrate this, consider once again the set of points in Figure 1. Any ordering of the points corresponds to a (one-dimensional) route that visits all the points. For example, sorting by the horizontal ordinate and then the vertical one produces a route that is monotonically non-decreasing in the horizontal ordinate and varying widely in the vertical one. This is illustrated in Figure 6. Taking a window size of 3 produces point groups ABD, BDC, DCG, CGF, GFE and FEH. The Sorted Neighbourhood Index (ie: all intra-group pairs of records) therefore includes distant pairs such as DC, GF and CG while excluding near ones such as DF and DE. CG is an example of a more distant pair than any that would be included by bivariate Standard Block Indexing as illustrated in Figure 4.

The same problem (once again, involving different points) occurs when sorting first by the vertical ordinate as illustrated in Figure 7. In this case,

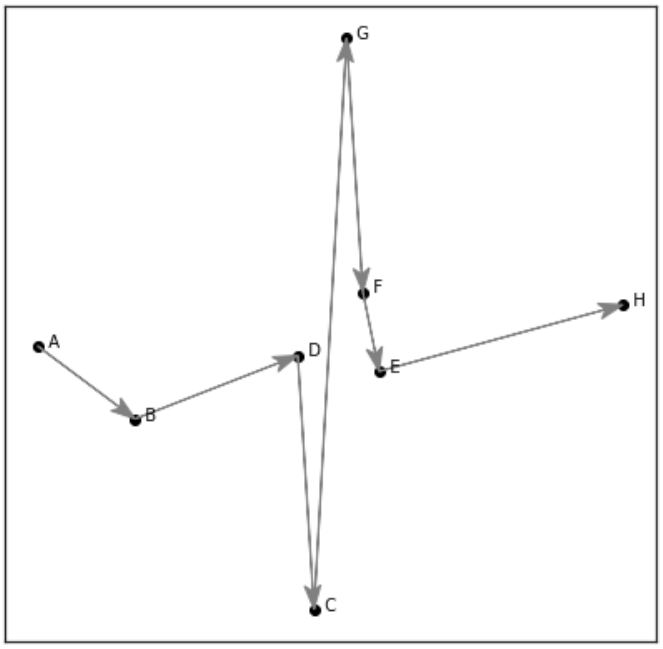

Fig. 6: Sorted Neighbourhood: route with sorting by horizontal variable

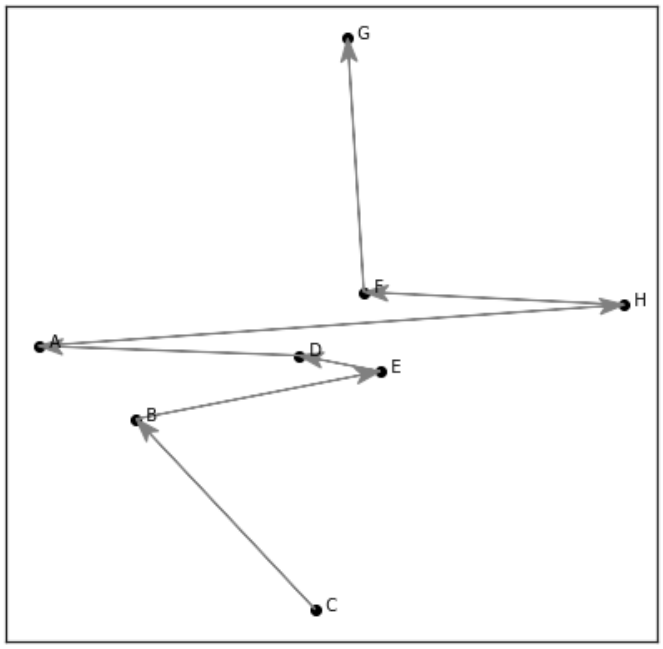

Fig. 7: Sorted Neighbourhood: route with sorting by vertical variable

distant pairs such as AH, HF and FG are included, while close ones such as DF and EF are excluded. AH is a more distant pair than any intra-block pair in Figure 4.

As with Standard Blocking, taking the union of the Sorted Neighbourhood Indexes using the two different sort orders doesn't always solve the problem of excluded nearby pairs. In this case, pair DF is excluded from the union of the two indexes.

D. Other issues

a) Treatment of Missing Values: Neither Standard Blocking nor Sorted Neighbourhood Indexing provide any meaningful treatment of missing values. When these methods are used in practice, missing values are typically either imputed or excluded (by omitting either rows or columns from the tables being linked). However, in reality missing values are just that – missing. The fact that a value is missing from a particular record neither confirms nor contradicts that record's pairing with any other record. Therefore, some allowance for limited wildcard matching of missing values is often desirable.

b) Allowance for Limited Field Mismatches: Records that refer to the same entity often contain differences in field values that would place them far apart in the sorting order. For example, consider the records shown in Table I. These appear to refer to two individuals - Catherine and Timothy Bourke, each of whom is referred to by a pair of records. In each pair of records, there are three fields which either agree exactly or would be nearby in the field's sorting order. However, they aren't the same three fields in both cases. In situations like this, a record pair selection rule like "any three of these five fields approximately agree" would be desirable.

Indexes of this type can be constructed using intersections and unions of Standard Blocking or Sorted Neighbourhood Indexes. However, this can involve many operations. In this case, an index of three-out-of-five *exact* matches is produced by the union of $\binom{5}{3}$ Standard Blocking indexes (ie: 10 of them), each corresponding to a particular choice of three out of the five fields. To modify this index to use three-out-of-five *proximity* matches, each Standard Blocking index can be replaced with the intersection of three Sorted Neighbourhood Indexes.

This can quickly become unwieldy. An algorithm that provides this type of of matching more directly can be of value in situations like this one.

E. Organization of This Paper

Section II provides an introduction to the entire record linkage process and indicates the role of indexing within it. Section III outlines existing work on indexing methods and describes Standard Blocking and Sorted Neighbourhood Indexing. Section IV describes Neighbourhood Blocking in detail and compares its properties to other indexing methods. Sections V and VI report comparative performance of Neighbourhood Blocking and other indexing methods on benchmark datasets and randomly generated datasets respectively. A discussion and conclusion follow in Sections VII and VIII.

In summary, this paper's contributions are:

- Proposal and description of Neighbourhood Blocking
- Description of a recursive implementation which avoids comparison of the Cartesian product of the records
- Theorems relating to:
    - Inclusion of all pairs of records that are closer to one another than a specific Euclidean distance.
    - Conditions under which Neighbourhood Blocking is an unambiguous superset of Standard Blocking
    - Neighbourhood Blocking Index size relative to that of Standard Blocking
- Tests comparing Neighbourhood Blocking, Standard Blocking and Sorted Neighbourhood Indexing with respect to:
    - index quality in both benchmark datasets and randomly generated ones
    - scalability properties in randomly generated datasets

II. Background - Record Linkage

A. Overview

Many applications such as data integration, deduplication and fraud detection require the identification of distinct records referring to the same entity without the aid of unambiguous identifying fields. For example, many census datasets lack a field suitable for unambiguously identifying individual people. In such cases, the task of resolving object identity must rely on a combination of other field values where each field provides a partial and imperfect indication of object identity. In the case of census data, these fields might include name, address, date of birth, country of birth and other personal details. These non-key fields typically have a many-to-many relationship with object identity. To take address as an example, several people might live at the same address, and the same person may have different addresses at different times (or even a choice of homes at the same time). There can also be multiple representations of the same piece of information due to variations in the use of abbreviation, preferred names (eg: Betty rather than Elizabeth), spelling of numbers etc. Additionaly, some differences are simply due to errors.

The combination of these factors has made record linkage a field of ongoing research since at least the 1940's. The term "record linkage" was used by [1] who colourfully calls references to the same person in multiple datasets "pages in the book of life" which, through identification of their coreference can be "bound into a single volume". A formal probabilistic approach assuming independence of matches in different fields was proposed by [2], and has since been recognized as equivalent to Naive Bayes Classification [3].

The identification of records referring to the same entity has been pursued separately in several disciplines. Consequently, many names are now used to refer to it. These include: conflation, coreference resolution, data linkage, data matching, deduplication, disambiguation, entity resolution, name resolution, object identification, propensity score matching, record linkage, reference reconciliation and several others. In this paper, it will be referred to generally as "record linkage" and as "deduplicaion" when the records to be matched are in the same table.

The current state of the art in record linkage is described in [4] [5] [6] [7] [8]. In addition to outlining theory and methods, [8] also reviews available tools.

Row	Given name	Surname	Address	Birth Date	SSID
1	Catherine	Bourke	42 Black Stump Cres	15-Mar-1958	3984257
2	Cathy	Smythe	42 Black Stump Cres	15-Mar-1958	398425
3	Timothy	Bourke	42 Black Stump Cres	06-Dec-1959	3939872
4	Timothy	Bourk	110 Beachfront Drive	06-Dec-1995	3939872

TABLE I: Example Records for Deduplication

The steps in the record linkage process described by [4] are:

1) *Preprocessing* – extraction of normalized representations and other features from the source dataset(s)
2) *Indexing* – Selection of record pairs for further consideration as possible matches
3) *Comparison* – Feature extraction from the selected record pairs
4) *Classification* – Binary classification of record pairs into matches and non-matches
5) *Evaluation* – Assessment of the quality of the record linkage produced

In deduplication, *merging* is often added as a final step. This involves producing a single record from each group of records that refer to the same entity.

These steps are described in more detail in sections II-B to II-F below.

The typical strategy in record linkage is to isolate the task of mapping records to entities and therefore produce a "single version of the truth" for subsequent use in querying or other applications. It should be noted that this simplification comes at the cost of discarding any alternative versions of the truth suggested by the original dataset. [9] outlines an alternative to this strategy where potential repetition in the original dataset is retained, and uncertainty about record-entity mappings (eg: probability that the entity satisfies an SQL WHERE clause) is included in query results.

In addition, there is significant variation between different record linkage methods, some of which make the steps described here non-separable. For example, [10] describes progressive blocking which involves integrating the indexing step with the comparison and classification steps in order to prioritize the processing of record pairs in regions near where the greatest density of record pairs classified as matches have been found so far.

B. Preprocessing

This step produces a normalized representation of the dataset. It can be thought of as feature extraction from individual records (rather than from pairs of records as in the Comparison step). Operations might include:

- parsing and datatype conversion
- splitting dates, names, addresses etc into components such as day, month, year, title, first name, surname etc.
- normalization of capitalization and abbreviations
- computation of topic model weights for description fields

C. Indexing

Given the normalized representation of the individual records produced in the Preprocessing step, the Indexing step selects record pairs for consideration as possible matches. In small datasets, it may be practical to select all possible record pairs. Where the dataset involved is too large for this, some method more scalable than the pairwise comparison in the Comparison step must be used to discount a large proportion of the possible record pairs. These scalable methods typically involve sorting and/or grouping (Neighbourhood Blocking uses both of these).

For example, consider the deduplication of the dataset illustrated in Table II. A rapid, high-recall indexing step with only 0.5% precision would reduce the number of record pairs to be considered in detail from approximately 5 billion to 1 million - a factor of $\frac{1}{5{,}000}$.

Item	Count
Total records	100,000
Duplicate records	5,000
Total record pairs	4,999,950,000
True matching record pairs	5,000

TABLE II: Example dataset for deduplication

The indexing step involves several competing objectives:

- Scalability - sufficient to accommodate the source dataset
- Size reduction - elimination of a large proportion of possible record pairs. This proportion is called the "reduction ratio".
- Recall - proportion of true matching pairs retained

In practice, sufficient scalability for the dataset in question acts as a constraint on the choice of method and parameters. Given that constraint, an application-specific tradeoff is made between reduction ratio and recall. As will be shown in section VI, there are cases where Neighbourhood Blocking provides simultaneous improvement in both reduction ratio and recall (at the expense of increased runtime) when compared to Standard Blocking or Sorted Neighbourhood Indexing.

Common techniques for selecting pairs of similar records are listed in Table III. These are described more fully in Section III.

D. Comparison

This step produces one or more similarity or distance measures for each pair of records identified in the Indexing step. It can be thought of as feature extraction for *record pairs* (as opposed to individual records as in the Preprocessing step).

Common pairwise similarity or distance measures include:

- binary indicators of exact matches for particular fields
- binary indicators of membership of the same set of synonyms / alternative name forms (eg: William, Bill) / soundex codes etc
- numeric differences (eg: for a person's height or weight)
- String similarity or difference measures (eg: various types of edit distance, Q-gram-based methods). Chapter 5 in [4] outlines several of these.

E. Classification

This step produces the final classification of record pairs into matches and non-matches. There are three sub-tasks involved with this.

First, there is the analysis of *individual record pairs* which typically involves producing a single similarity score or probability for each pair of records under consideration. [11] provides an overview of distance measurement techniques. The similarity score can be computed using a manually specified formula (eg: the sum of individual similarity scores for different fields) or through machine learning techniques. Classification models can be used in cases where a sufficiently representative sample of known matching pairs is available. In the absence of this, accurate classification can sometimes be produced using clustering as reported in [12].

Second, the classification can be refined by applying constraints that apply *across record pairs*. In deduplication, the constraint of transitive closure can be used. That is: for three records (call them A, B and C), if pairs (A, B) and (B, C) are matches then pair (A, C) must also be a match. [13] describes scalable methods of incorporating this constraint. In linkage tasks where both tables are presumed not to contain any duplicate records, a uniqueness constraint can be used. That is: no two records in the same table can be matched with the same record in the other table.

Finally, there will typically be some record pairs for which the classification is borderline or inconclusive. Depending on the application, the relatively high expense of *clerical review* (ie: manual classification) might be justified for these record pairs.

F. Evaluation

Record linkage is ultimately a binary classification task where the class of interest (ie: record

Indexing technique	Record pair selection criterion
Standard blocking	Exact match in one or more fields
Sorted neighbourhood	Proximity in a sorting order
Qgram-based, suffix-based	Exact match in any slight variations of a field's value
Canopy clustering; many : 1 mappings	cluster or group membership

TABLE III: Common indexing techniques

pairs that are matches) is extremely rare. For example, deduplication of the dataset described in Table II, non-matching record pairs are approximately a million times more common than matching ones. Consequently, metrics like precision, sensitivity and specificity which focus on the class of interest are typically more useful in evaluating linkage quality than overall accuracy.

G. Merging

In deduplication, groups of records relating to the same entity are identified. It is then typically desirable to merge information from each such record group in to a single record. This is often considered to be a separate task to record linkage, but section 6.12 in [4] provides some detail on it.

H. Treatment of missing values

Real datasets typically contain missing values which can complicate all the steps in the linkage process. There are three main possibilities for coping with missing values:

- Use methods robust to missing values at *every* step in the process
- Omit *rows* containing missing values from consideration
- Omit *columns* containing missing values from those steps involving methods not robust to missing values.

As will be explained in Section IV, Neighbourhood Blocking provides meaningful treatment of missing values.

III. Related Work

A. Overview

The purpose of indexing in record linkage is to produce pairs of records for further consideration in the Comparison and Classification steps. The need for this was identified in [2] where the indexing technique now known as Standard Blocking is also described. Indexing for record linkage is a field of active research, and several approaches described in this section have significant similarities to Neighbourhood Blocking.

B. Full Index

The simplest way of selecting record pairs for further consideration is simply to select all possible pairs. This is known as "Full Indexing" and produces indexes of the sizes indicated in Table IV. The full index size can be manageable in the case of smaller datasets. For example, [14] focuses on the Comparison phase of the record linkage process (ie: comparisons of pairs of records rather than the selection of pairs for consideration). Numerical deduplication experiments are performed there on datasets with 2,000 rows and therefore approximately 2,000,000 record pairs in a full index.

Task	Full Index Pair Count
Linkage (n rows to m rows)	nm
Deduplication (n rows)	$\frac{n^2-n}{2}$

TABLE IV: Full Index Sizes

C. Standard Blocking

Standard Blocking, described by [2], produces all pairs of records which have exact matches in all fields designated as "blocking keys" (there can be multiple blocking keys). This can be implemented using the following steps:

1) Produce an inner join of the table(s) using the blocking keys (for deduplication, there is only one table - join it to itself). Retain only the two columns containing the record

identifiers for the left and right tables in the join.

2) If there is only one source table (ie: the index is for deduplication rather than linkage), discard all rows where the left row identifier is greater than or equal to the right row identifier.
3) The remaining pairs of row identifiers are the index.

D. Sorted Neighbourhood Indexing

This method, described in [15] selects all pairs of records that are within a fixed distance of one another in a (single) sorted list. Unlike Standard Blocking, Sorted Neighbourhood Indexing produces the union of full indexes of *overlapping* groups of records. Since these record groups overlap, Sorted Neighbourhood Indexing avoids the block boundary problem described in Section I-B. [16] describes an efficient method for implementing an online version of Sorted Neighbourhood Indexing using a tree-based approach to maintain the sort order as new records are added.

Sorted Neighbourhood Indexing requires two parameters:

- a list of sorting keys (there could be just one), and
- a "window width" which must be an odd positive integer

These steps can be used to produce a Sorted Neighbourhood Index:

1) Produce a single table of all distinct combinations of sorting keys in the dataset (ie: if there are two tables, take the union of the sorting key combinations in both of them). Call this the "Key Combination Table"
2) Sort the Key Combination Table.
3) For each record in the source table(s) find which row in the Key Combination Table has the same combination of sorting keys. Store these row numbers in a new "Rank" column in the source table(s)
4) Join the source table(s) (if there's only one then join it to itself). The join condition is that the absolute difference in Rank does not exceed $\frac{w-1}{2}$ where w is the window width. Retain only the two columns containing the record identifiers for the left and right tables in the join.
5) If there is only one source table (ie: the index is for deduplication rather than linkage), discard all rows where the left row identifier is greater than or equal to the right row identifier.
6) The remaining pairs of row identifiers are the index.

E. Mappings and value modifications

These methods map individual blocking key values to one or more alternative versions.

Many-to-one mappings (such as Soundex which maps strings to sound codes) are a way of coarsening blocking (thereby increasing the number of record pairs included).

Many-to-many mappings (such as string modifications using q-grams or suffix arrays) result in blocking where each record is effectively a member of multiple blocks, thereby reducing the effects of block boundaries. Some such methods are described in chapter four of [4]. Many-to-many mappings can often result in very large indexes. [17] describes an approach to address this by "pruning" the size of the mapping. This involves including only relatively rare BKV variants, greatly reducing the size of the index, but retaining those record pairs involving the coincidence of unusual variants.

F. Other Methods

Since indexing is applied to the entire source dataset, it typically has a greater need for scalability than the Comparison and Classification steps that follow it. Consequently, indexing algorithms typically require field comparison methods with properties that enhance algorithm scalability (eg: methods that rely on sorting require ordinal field comparisons).

Most indexing algorithms use these scalability-friendly comparison methods exclusively. However, some algorithms use them *in addition to* other comparison methods which lack the properties that support algorithm scalability.

Canopy Clustering is one such method. It is an approximate clustering method specifically suited to large datasets and distance functions that are slow to compute. It initially estimates distances between records using some faster distance function and then uses the slow distance function to revise distances below a threshold value. The steps involved in Canopy Clustering are:

1) Begin with no records allocated to clusters.
2) Choose an unallocated record at random. This will be the "centroid" of a new cluster. Use a fast comparison technique (eg: Jaccard similarity of q-gram sets) to identify other unallocated records that are similar to it (this involves comparison of the centroid record to all unallocated records). Then (only on those records selected) use *the slow comparison technique* to compute their distances to the centroid record. Allocate the centroid record and any others that were found to be sufficiently close to it to the new cluster.
3) As long as any records remain unallocated, continue repeating the previous step.

Another such method is Progressive Blocking which is described in [10]. This is a method of prioritizing the Comparison and Classification of record pairs with the aim of processing those more likely to be matches first. To do this, it relies on *integration with the Comparison and Classification tasks* in order to obtain feedback from them on the density of true matches found so far in different regions in the record pair space.

IV. Neighbourhood Blocking

A. Intuition

Neighbourhood Blocking can be thought of as an extension of Standard Blocking to include proximity matching. Equally, it can be thought of an extension of Sorted Neighbourhood Indexing to require proximity with respect to multiple sorting keys. Like Standard Blocking, it partitions the dataset into groups of records or "blocks" corresponding to distinct combinations of Blocking Key Values (BKVs – values in nominated fields). In addition to pairing each record with all other records in its own block, Neighbourhood Blocking introduces the notion of a "neighbourhood" of blocks that are "close" to a central block. Here "closeness" is determined by positions of BKVs in their respective sort orders. The block neighbourhood is a multi-dimensional analogue of the record "window" in Sorted Neighbourhood Indexing.

Standard Blocking and Sorted Neighbourhood Indexing share the property that there is a sorting order of the records such that: if a record pair is in the index, the records in it are close to one another in that sorting order. This property can be used to compute the index without comparing the Cartesian product of the blocks. In Neighbourhood Blocking, there is no single sorting order with this property since block proximity depends on multiple sorting orders (ie: one for each blocking key). Therefore, an alternative method of avoiding comparison of the Cartesian product of the blocks is required.

To achieve this, all blocking key values are mapped to integers. This preserves the information required to compute the index (ie: equality and rank difference relationships between BKVs) but adds the ability to perform arithmetic transformations. Specifically, "coarsening" transformations of the form $x \mapsto \lfloor \frac{x}{a} \rfloor$ (where $a > 1$) are especially useful because they reduce the number of distinct BKVs while weakly preserving their order. This allows pairs of matching blocks to be found recursively by applying the Neighbourhood Blocking algorithm to a coarsened representation of its own blocks. This is described in section IV-D2.

Since all (non-missing) values are converted to integers, only value comparison methods that are applicable to integers can be used. This rules out methods like q-grams and suffix arrays, but it does allow two other useful record matching criteria:

- wildcard matching of missing values, and
- allowance for complete mismatches in a limited number of blocking keys

Use of these two additional record matching criteria loosely corresponds to modifying the dimensionality of the block neighbourhoods.

B. Terms and Abbreviations

Key	A field used for grouping records into Blocks (blocking key) or for sorting them (sort key)
BKV	Blocking Key Value - one of the values contained in a blocking key
Block	A group of all records with a particular combination of BKVs

C. Functional Description

Neighbourhood Blocking requires the following parameters:

- One or more blocking keys
- For each blocking key, a rank distance limit for Proximity matching (see below).
- A maximum number of wildcard (null value) matches allowed for a record match.
- A maximum number of field mismatches allowed for a record match.

As with Standard Blocking, the dataset is partitioned into non-overlapping blocks, each corresponding to a distinct combination of blocking key values (BKVs).

Two BKV matching criteria are used:

- *Wildcard*: The BKV is null in one or both records
- *Proximity*: The rankings of non-null BKVs of the two records (in a sorted list of that blocking key's distinct non-null values) differ by no more than the rank distance limit specified for that blocking key. A special case of Proximity is *Equality* which is where the BKVs are equal and not null.

Both of these criteria can be evaluated from absolute differences in BKV rankings (assuming that arithmetic operations involving Null return Null).

The record pairs included in the index are all those satisfying *both* of the following conditions:

- The values of at least a prescribed minimum number of blocking keys must match by the Wildcard or Proximity criteria.
- The number of these blocking keys matching by the Wildcard criterion must not exceed a prescribed maximum (specified as a parameter of the blocking process)

D. Algorithm Description

1) Outline of Steps: Neighbourhood Blocking can be implemented in the following steps:

a) Normalize BKVs: Replace all non-null BKVs with integers representing their rank (ie: each non-null BKV is replaced with the count of *distinct* non-null values in the same column which appear earlier than it in a sorted list). This enables the recursive implementation discussed in Section IV-D2.

b) Atomic blocking: Produce a (single) master table of block BKV combinations by taking all distinct combinations of BKVs in the table(s) being indexed. Assign a distinct block ID to each row in this table.

c) Produce a linkage index of candidate block pairs: This is an index of the pairs of blocks on which the matching conditions will be checked in the next step. The simplest approach is to use a Full Index (ie: check the matching criteria for all pairs of blocks). However, as discussed in Section IV-D2, there are cases when the overwhelming majority of candidate block pairs (which is quadratic in the number of blocks) can be eliminated by recursively indexing tables representing coarsened blocks.

d) Identify pairs of matching blocks: Compare the blocks in each of the block pairs identified in the previous step and determine which ones satisfy the record matching conditions. Put their block IDs into a link table (ie: each row is a pair of block IDs).

e) Translate block pairs to record pairs: Use a sequence of database-style joins to determine the pairs of row IDs corresponding to row pairs satisfying all the matching conditions. This sequence is:

1) For each row in the table(s) being indexed, add a column containing the ID of the block containing that record. This can be done by joining by the blocking keys to the block definition table described in sectio IV-D1b.

2) Join these tables on these block IDs via the block link table which was produced in step IV-D1d.

f) For deduplication, filter the record pairs: In the case of deduplication of a single table (as opposed to linkage of two tables), filter the list of record id pairs to only include unique pairs (regardless of order) of distinct record IDs. This filtered list of record ID pairs is the final index.

2) Recursive Implementation: There is a complication in step IV-D1c. Neighbourhood Blocking involves the combination of each atomic block with multiple others. The general nature of both the Proximity criterion and the Wildcard criterion mean that there is no single ordering for the blocks such that satisfaction of all the block matching criteria corresponds to proximity in the ordering. Consequently, sorting-based methods for avoiding comparison of the Cartesian product of blocks (which can be used in Standard Blocking or Sorted Neighbourhood Indexing) are inapplicable to Neighbourhood Blocking.

However, a method that can be used by Neighbourhood Blocking to avoid comparing the Cartesian product of its blocks is Neighbourhood Blocking itself (applied to a coarsened representation of the blocks). This works as follows:

- A Full Index of the blocks is only produced if the blocking is maximally coarse (ie: each blocking key has only one non-null value. This limits the maximum number of distinct BKV combinations to 2^n (including null values) where n is the number of blocking keys.
- When the blocking is not maximally coarse:
  1) For each of the table(s) being indexed, produce a dataset describing the *blocks* present in that table. That is: a subset of the rows from the block table (described in section IV-D1b) corresponding to the distinct BKV combinations in the table being linked. Each of these "block datasets" is keyed by block ID and contains columns corresponding to the (current) BKV ranks (produced in step IV-D1a).
  2) Coarsen the BKV ranks in the tables produced in the previous step by replacing each non-null value x with $\left\lfloor \frac{x}{a} \right\rfloor$ where $a > 1$.
  3) Produce a Neighbourhood Blocking index on these coarsened block description tables, using the same parameters (blocking keys, rank distance limits, wildcard limit and mismatch limit) that are being used for indexing the records.

At each stage of the coarsening described above, the number of distinct non-null values in each blocking key halves (subject to a minimum of one non-null value). This means that:

- the size of the indexing calculations for successive coarsenings decreases exponentially, and
- the number of coarsenings required to achieve maximal coarseness is logarithmic in the maximum number of distinct values in any blocking key

E. Comparison to other methods

Standard Blocking can be thought of as a particular case of Neighbourhood Blocking where no field mismatches are allowed, all rank distance limits are zero, and no wildcard matches or field mismatches are allowed.

Similarly, there are configurations of Sorted Neighbourhood Indexing which are special cases of Neighbourhood Blocking where only one blocking key is used and there are no wildcard matches or field mismatches.

Strictly speaking, since Neighbourhood Blocking effectively involves a many-to-many mapping of BKVs, it bears some similarity to mapping-based techniques like q-grams or suffix arrays (described in section III-E). However, in Neighbourhood Blocking, inexact BKV matches are restricted to those that can be determined from sorting order or null values alone. As is shown in Section IV-D, this means that the inexact matches can be determined using arithmetic operations rather than lookup tables.

Neighbourhood Blocking is distinct from Progressive Blocking as outlined in [10]. The primary differences are that it is entirely separable from the Comparison and Classification steps, uses multiple sorting orders to determine block proximity and allows for wildcard matching and field mismatches.

A comparison of some key features of Neighbourhood Blocking and some of its counterparts is summarized in Table V.

Feature	Standard	Sorted N'hood	Progressive	Neighbourhood
multiple block keys	✓		✓	✓
multiple orderings	N/A			✓
block combination / overlap		✓	✓	✓
separable from Comparison	✓	✓		✓
nulls as wildcards				✓
limited non-matches				✓

TABLE V: Comparison of Index Algorithm Features

F. Properties

The Proximity matching criterion allows the inclusion of record pairs which straddle block boundaries. By Theorem IV.1, where there is a notion of position within blocks (making the notion of "close pairs straddling block boundaries" meaningful), Neighbourhood Blocking includes all record pairs closer than a specific "inclusion distance", regardless of the specific locations of block boundaries.

Theorem IV.1 (Inclusion distance)**.** *A Neighbourhood Blocking index using blocking keys that are discretized versions of continuous variables will include all pairs of records whose non-discretized Euclidean distance is less than the product of (a) the length of the unit of discretization, and (b) the lowest of its rank distance limits. This is true regardless of the locations of block boundaries.*

Proof: Let the differences in the values of each of the non-discretized blocking keys be $\delta_1, \delta_2 \cdots \delta_n$ (where n is the number of blocking keys). Let the unit of discretization be u and let the minimum of the rank distance limits be r_{min}. By assumption, the Euclidean distance between the records is less than $r_{min}u$. In other words:

$$\sqrt{\sum_j \delta_j^2} < r_{min}u \Rightarrow \sum_j \delta_j^2 < (r_{min}u)^2 \quad (1)$$

If the record pair is *not* included in the index by the Proximity criterion, the following condition must be true.

$$\exists j : \delta_j > r_{min}u \quad (2)$$

which implies:

$$\sum_j \delta_j^2 > (r_{min}u)^2 \quad (3)$$

since

$$\forall j : \delta_j^2 > 0 \quad (4)$$

Since (1) is a contradiction of (3), any pair of records separated by a Euclidean distance less than $r_{min}u$ must be in the index. ■

Clearly, a Neighbourhood Blocking Index is a superset of a Standard Blocking index which uses the same keys. However, by Theorem IV.2, it is also a superset of a Standard Blocking Index where the granularity of any or all of the blocking keys is coarsened by combining groups of $1 + r_j$ adjacent values where r_j is the j[th] blocking key's rank distance limit.

Theorem IV.2 (Superset of Standard Blocking)**.** *If:*

1) X_N *is a Neighbourhood Blocking index with keys* $k_1, k_2 \cdots k_n$ *and corresponding rank distance limits of* $r_1, r_2 \cdots r_n$
2) *Each* k'_j *(*$j \in \{1 \cdots n\}$*) is a* $(1 + r_j)$*:1 mapping of* k_j *such that each distinct value of* k'_j *corresponds to* $(1 + r_j)$ *consecutive sorted values of* k_j
3) X_S *is a Standard Blocking Index with keys* $k'_1, k'_2 \cdots k'_n$

Then: $X_N \supseteq X_S$

Proof: X_S comprises the union of Full Indexes on each of its blocks. Therefore, it suffices to show that the Full Index of each such block is included in X_N.

Consider any X_S block. All records in it contain identical values of $k'_1, k'_2 \cdots k'_n$. In the BKV matching for X_N, assumption 2 implies that the

values for each of $k_1, k_2 \cdots k_n$ must match by the Proximity criterion. Thus, all record pairs in the X_S block are included in X_N. ■

Theorem IV.3 relates to an idealized database where:

- all keys have sufficiently many distinct values for edge effects to be negligible,
- records are uniformly distributed throughout the key space, and
- no keys have any null values

Under these idealized conditions, it is shown that the size of a Neighbourhood Blocking index is larger than a Standard Blocking Index using the same keys by a factor of $\prod_j(1 + 2r_j)$, where r_j is the rank distance limit for the j[th] blocking key. By Corollary IV.3.1, for the same idealized database and where all blocking keys are also sorting keys, a Neighbourhood Blocking Index has the same reduction ratio as a Standard Blocking Index with each key coarsened by a factor of $1 + 2r_j$.

Theorem IV.3 (Index size relative to Standard Blocking)**.** *In datasets where each block contains the same number of records and each sorting key has the same number of distinct values, the index sizes for Standard Blocking and Neighbourhood Blocking are related by:*

$$\lim_{d,v\to\infty} \frac{\mid X_N \mid}{\mid X_S \mid} = \prod_j (1 + 2r_j) \tag{5}$$

where:

X_S	*is the set of record pairs from Standard Blocking*
X_N	*is the set of record pairs from Neighbourhood Blocking where no field mismatches or wildcards are allowed.*
r_j	*is the rank distance limit used in Neighbourhood Blocking for the j[th] blocking key*
d	*is the number of records per block,*
v	*is the number of distinct values of each blocking key*

Proof: The ratio of the index sizes can be itemized into contributions from interior and non-interior blocks. Here, "interior block" means a block whose neighbourhood is not limited by maximum or minimum values of any of the sorting keys. The itemization of the index size ratio is expressed in (6).

$$\frac{\mid X_N \mid}{\mid X_S \mid} = p_I R_I + (1 - p_I) R_N \tag{6}$$

where:

p_I	proportion of blocks that are interior blocks
R_I	ratio of number of pairs contributed by interior blocks
R_N	ratio of number of pairs contributed by non-interior blocks

The number of record pairs contributed to the Standard Blocking Index by every block (interior or not) is given by (7) for a deduplication index and by (8) for a linkage index.

$$\frac{d(d-1)}{2} \tag{7}$$

$$d^2 \tag{8}$$

In Neighbourhood Blocking, each record is paired with the records in its own block and also any other block whose BKV ranks differ by no more than r_j for each j. A record in an interior block can therefore be paired with records in a total of $\prod_j(1+2r_j)$ blocks (since the paired record's ranking in the j[th] key's sort order can differ by any of: $-r_j \cdots 0 \cdots r_j$). Therefore, the number of record pairs contributed to the Neighbourhood Blocking Index by each *interior* block is given by (9) for a deduplication index and by (10) for a linkage index.

$$\frac{d(\left(\prod_j(1 + 2r_j)\right) d - 1)}{2} \tag{9}$$

$$d^2 \prod_j (1 + 2r_j) \tag{10}$$

R_I is given by (9) divided by (7) in the case of a deduplication index and by (10) divided by (8) in the case of a linkage index. In both these cases:

$$\lim_{d\to\infty} R_I = \prod_j (1 + 2r_j) \tag{11}$$

Let n be the number of blocking keys. The total number of blocks is v^n, and the number of interior blocks is $\prod_j(v - 2r_j)$ which is no smaller than

$(v - 2r_{max})^n$ where $r_{max} = \sup_j r_j$ Therefore, the proportion of blocks that are interior blocks satisfies:

$$p_I \geq \left(1 - \frac{2r_{max}}{v}\right)^n \tag{12}$$

implying that:

$$\lim_{v \to \infty} p_I = 1 \tag{13}$$

Combining (13) and (11) with the recognition that R_N must be finite completes the proof. ■

Corollary IV.3.1. *In the limiting case described in Theorem IV.3, Neighbourhood Blocking produces the same index size as Standard Blocking with each blocking key coarsened by a factor of* $(1+2r_j)$ *where* r_j *is the* j^{th} *blocking key's rank distance limit.*

Proof: Let the total number of records in the dataset be N. This is related to d, v and n by (14).

$$N = dv^n \Rightarrow d = Nv^{-n} \tag{14}$$

The size of a Standard Blocking index is therefore given by (15) for a deduplication index and by (16) for a Linkage index.

$$\frac{N(Nv^{-n} - 1)}{2} \tag{15}$$

$$N^2 v^{-n} \tag{16}$$

If the blocks are coalesced into larger ones with a "side length" of $(1 + 2r_j)$ old blocks in the direction of each (j^{th}) blocking key, v^n in (14), (15) and (16) is effectively replaced with a smaller number $\frac{v^n}{\prod_j (1+2r_j)}$. Therefore, for both deduplication and linkage indexes in the limiting case as $N \to \infty$ the ratio of the size of a Standard Blocking Index with the larger blocks to that of one with the smaller blocks tends to:

$$\prod_j (1 + 2r_j) \tag{17}$$

which is the same ratio given in (5) ■

G. Application to earlier examples

Figure 8 illustrates how the application of Neighbourhood Blocking addresses the two issues outlined in Section I. Namely, application of proximity matching in multiple directions and inclusion of close pairs of points that straddle block boundaries. In Figure 8, the block size is half that used in Figure 4 and the rank distance limit is 1 for all keys. By Theorem IV.2, the Neighbourhood Blocking shown in Figure 8 includes all the pairings produced by the Standard Blocking shown in Figure 4. The shading in Figure 8 indicates blocks containing points that are paired with point D, the darker shaded block matching by Equality, and the lighter shaded ones matching by Proximity. Although this blocking contains boundaries in the same positions as those in Figure 4 (and therefore the central cluster is still divided by block boundaries), these do not prevent the inclusion of point pairs from the central cluster in the index since all these pairs match by the Proximity criterion.

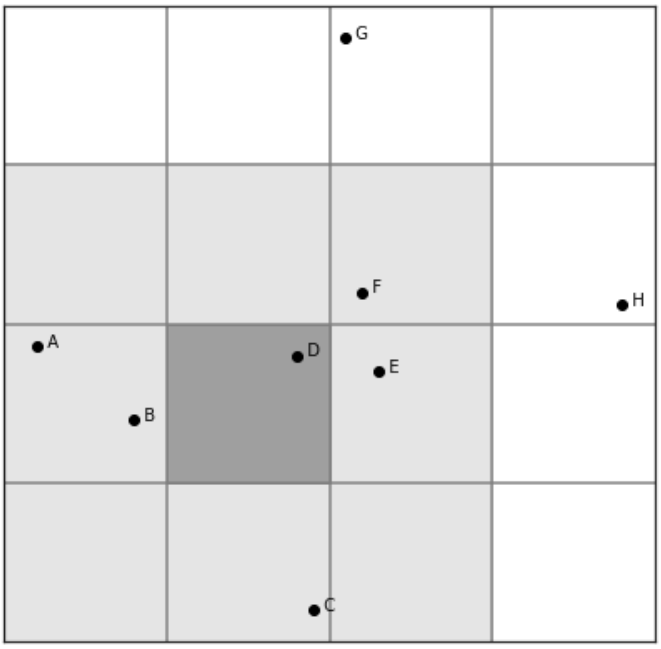

Fig. 8: Neighbourhood Blocking (coarse): pairings for point D

From the perspective of inclusion of the same record pairs, the Neighbourhood Blocking illustrated in Figure 8 is comparable to the Standard Block-

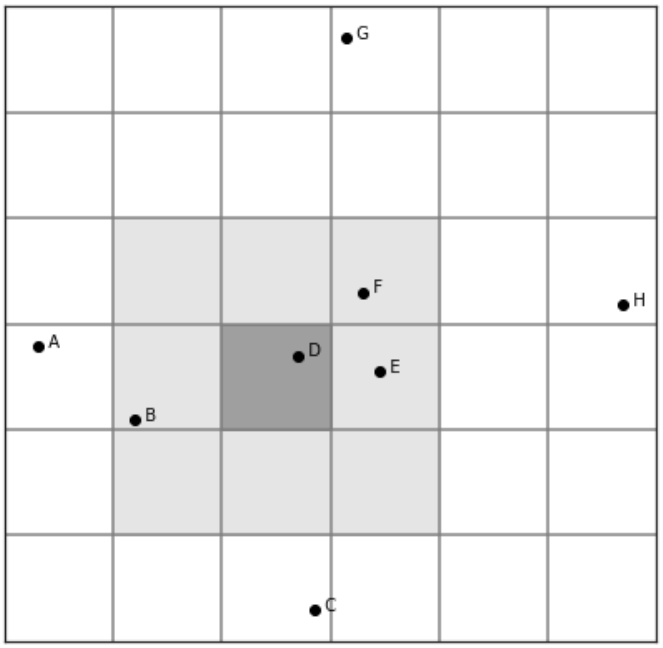

Fig. 9: Neighbourhood Blocking: pairings for point D

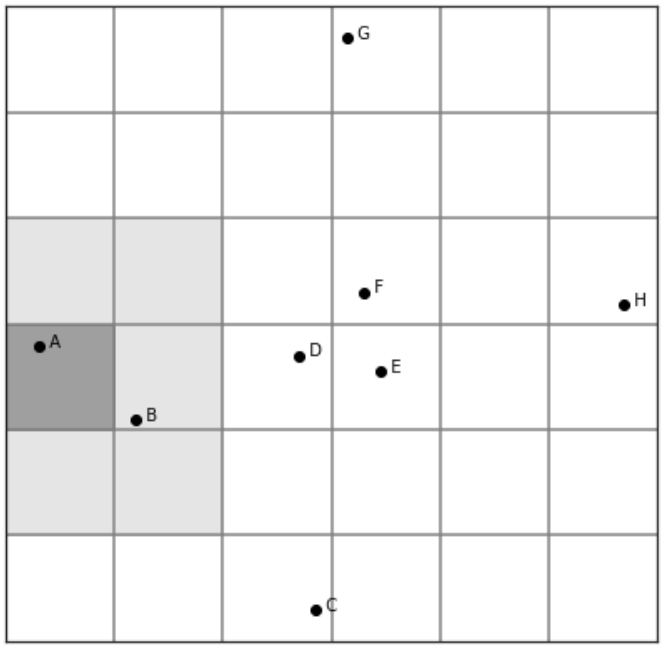

Fig. 10: Neighbourhood Blocking: pairings for point A

ing shown in Figure 4. However, including all the record pairs in the Standard Blocking Index *plus extra ones* means that the Neighbourhood Blocking in Figure 8 has a lower reduction ratio than the Standard Blocking in Figure 4. As Corollary IV.3.1 indicates, the reduction ratios in a large dataset are comparable (under idealized conditions) when the block side length used in Neighbourhood Blocking (with $r_j = 1 \forall j$) is around $\frac{1}{3}$ that used in Standard Blocking. This is illustrated in Figures 9, 10 and 11. Comparison of figures 8 and 9 highlights that although the reduction in block size introduces new boundaries, this does not cause the omission of close point pairs as happens in Standard Blocking. Instead, the pairs removed from the index are distant ones like AD and CD. Although a new block boundary is introduced dividing pair AB, this close pair is still included in the index as shown in Figure 10. Indeed, by Theorem IV.1 that the index includes all point pairs that are closer than the width of the blocks, regardless of the specific positions of the block boundaries.

Comparison of Figure 11 with Figures 9 and 10 illustrates that (unlike Standard Blocking) Neighbourhood Blocking does not exhibit transitive closure. Transitive closure can be violated when there is a sequence of points whose end points are too far apart for inclusion in the index, but each successive pair of points in the sequence is close enough for inclusion. In this example, pairs AB and BD are included as illustrated in Figure 11, but as Figures 9 and 10 show, pair AD is excluded.

H. Wildcard Matching of Missing Values

Figure 12 illustrates the treatment of missing values in Neighbourhood Blocking. This includes the same points as in earlier examples, with the addition of two new ones:

- P which has a null value for the vertical (y) ordinate, and
- Q which has null values for both ordinates (x and y)

The area highlighting in Figure 12 indicates the matching criteria for point D when up to one (null value) wildcard match is allowed. Wildcard matches are indicated by the hatched areas in the bars containing null values. Since up to one wildcard

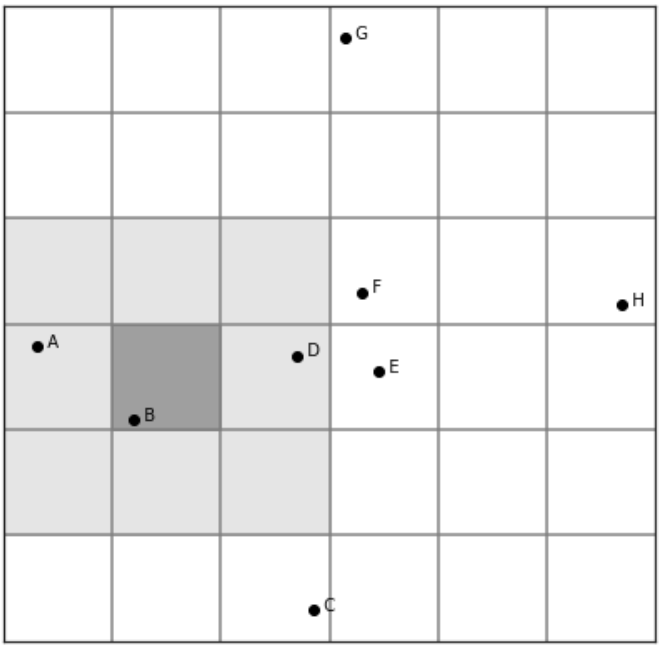

Fig. 11: Neighbourhood Blocking: pairings for point B

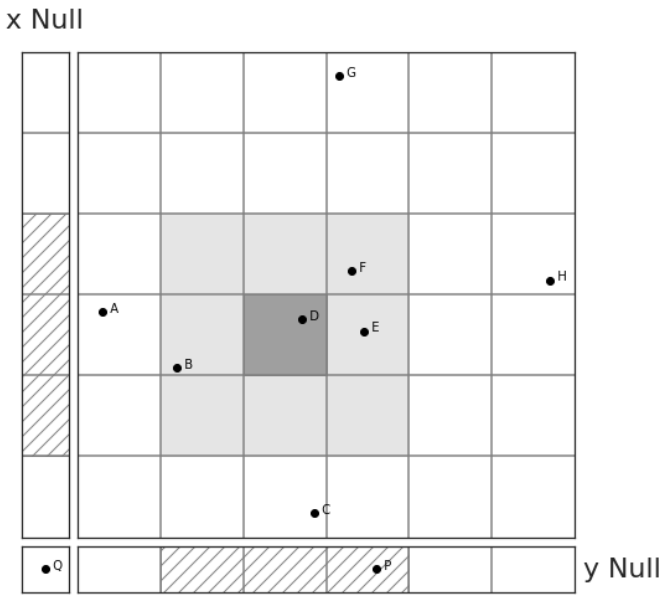

Fig. 12: Neighbourhood Blocking: pairings for point D allowing one missing value

match is allowed, pair PD will be included in the index but point Q (which has two null values) will not.

Figure 12 also illustrates the rationale for allowing null or missing values to be treated as wildcards and also for parameterizing a limit to how much this is done (if at all). What is known about point P is that its horizontal ordinate does not preclude a match with point D. Its vertical ordinate is missing so it neither precludes nor implies a match with point D. Which way should this ambiguity be resolved? Treating its missing vertical ordinate as wild results in it being paired with many other points (C, D, E, F and G in this case). Allowing two wildcard matches would cause point Q to match all other points, making the "cost" (decrease in reduction ratio) of including point Q even greater. However, as will be seen in Section VI, there are cases where allowing wildcard treatment of null values delivers a significant improvement in recall with little deterioration in reduction ratio.

I. Allowance for Non-Matches

Figure 13 illustrates the effect of allowing non-matches in one of the fields. In this figure, the blocking is twice as granular as that in Figures 9 to 11. Allowing a mismatch in either of the two fields would cause points in the cross-hatched regions to be paired with point D. As can be seen, this feature has the potential to dramatically increase the index size. In this example, it causes point D to be paired with all other points.

V. Application to Benchmark Datasets

Index quality comparisons between Neighbourhood Blocking, Standard Blocking and Sorted Neighbourhood Indexing were made on several benchmark datasets which are summarized in Table VI.

a) FEBRL Datasets: These datasets are supplied with the Python "recordlinkage" package and focus on resolving identities of people. These datasets are ready for indexing and have little need for preprocessing.

b) DBG Leipzig Datasets: These datasets are available for download from the University of Leipzig Database Group at:

Source	Dataset	Entity Type	Column Count(s)	Row Count(s)	True Matches	Calculated Fields
recordlinkage	FEBRL1	Person	13	1,000	500	Date components
	FEBRL2	Person	13	5,000	1,934	Date components
	FEBRL3	Person	13	5,000	6,538	Date components
	FEBRL4	Person	13; 13	5,000; 5,000	5,000	Date components
DBG Leipzig	Amazon-Google Products	Product	11; 11	1,363; 3,226	1,300	parsed codes; topic modelling
	ABT-Buy	Product	10; 11	1,081; 1,092	1,097	parsed codes; topic modelling
	DBLP-ACM	Publication	4; 4	2,616; 2,294	2,224	
	DBLP-Scholar	Publication	4; 4	2,616; 64,263	5,347	

TABLE VI: Sample Datasets

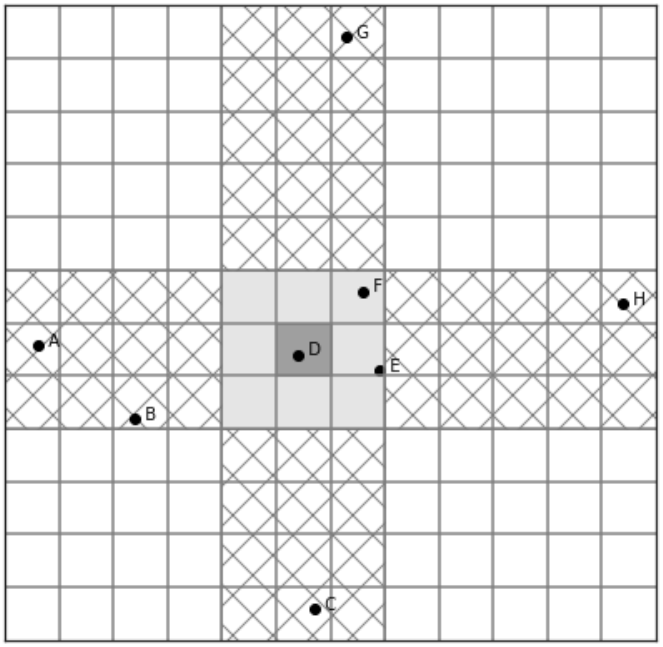

Fig. 13: Neighbourhood Blocking: pairings for point D allowing one non-match

https://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution

These datasets are less ready for indexing than the FEBRL ones and require some preprocessing.

A. Reproducibility

The tests reported in this section and in section VI can be reproduced using code located at:

https://github.com/dan-elias/neighbourhood_blocking

This repository contains:

- A Python implementation of Neighbourhood Blocking
- A Makefile for installing dependencies and downloading and preparing benchmark datasets
- Scripts for performing the tests
- Jupyter notebooks for viewing results
- A README file containing instructions

B. Index Quality Metrics

In this section and in section VI, index quality is assessed using two measures: recall and reduction ratio. These reflect two separate objectives of the indexing process. Namely:

- retention of true matches
- reduction of index size

These measures are given by:

$$\text{recall} = \frac{\mid X \cap T \mid}{\mid T \mid}$$

$$\text{reduction ratio} = 1 - \frac{\mid X \mid}{\mid \Omega \mid}$$

where:

X	set of record pairs in the index,
T	set of record pairs that are true matches
Ω	set of all possible record pairs

Precision is not used. Although it is an important measure for assessing the *final* record pairings (produced by the Classification step), it is typically sacrificed in the Indexing step in favour of scalability.

C. Methodology

After the calculated fields described in Table VI were added to the datasets, a number of indexes were calculated for each dataset using each of the indexing methods. These were based on valid combinations of the parameters listed in Table VII. These parameters were used in each of the indexing methods as described in sections V-C1 to V-C3. For each index produced, a point representing its recall and reduction ratio was computed. These were grouped by indexing method and the "frontier points" among them were identified as those that:

1) are on the convex hull surrounding all points for the indexing method, and
2) do not have *both* lower recall and lower reduction ratio than any other point

Variable	Values	
	bounds	distribution
Number of blocking keys	1 .. number of columns	linear
Half window	1 .. $\frac{1}{4}$ number of rows	geometric
Non-match Limit	0, 1	

TABLE VII: Indexing Parameters for Benchmark Datasets

1) Sorted Neighbourhood Indexing: Sorted Neighbourhood Indexes were produced for all combinations of database column (used as the sort key) and Half Window. The window width used was $1 + 2r$ where r is the Half Window.

2) Standard Blocking: The only parameter used to determine Standard Blocking configurations was the number of blocking keys. When the number of blocking keys was the same as the number of fields available, blocking was done on all fields. Otherwise, a sample of indexes was produced using multiple combinations of the required fields.

3) Neighbourhood Blocking: Neighbourhood Blocking indexes were computed for all combinations of one or two blocking keys. The rank distance limit was set to the Half Window.

D. Results

Index quality frontiers for the tests performed are shown in Figures 14 to 21. Each figure relates to a dataset. Each curve within these figures relates to an indexing method. The points on the curve represent the frontier of combinations of recall and reduction ratio achieved with the corresponding indexing method and dataset.

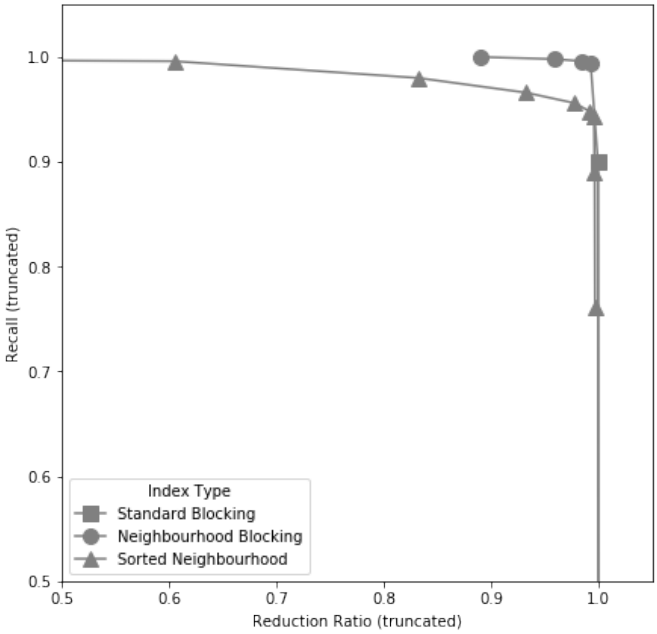

Fig. 14: Index Quality Frontiers - FEBRL1

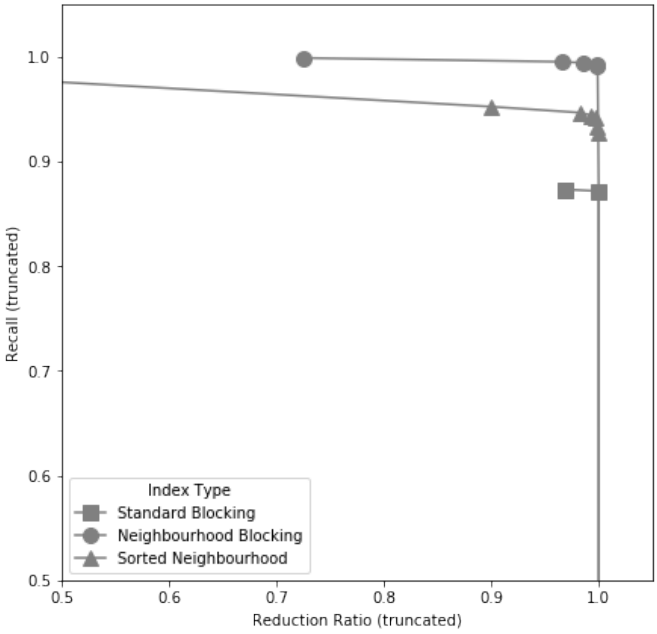

Fig. 15: Index Quality Frontiers - FEBRL2

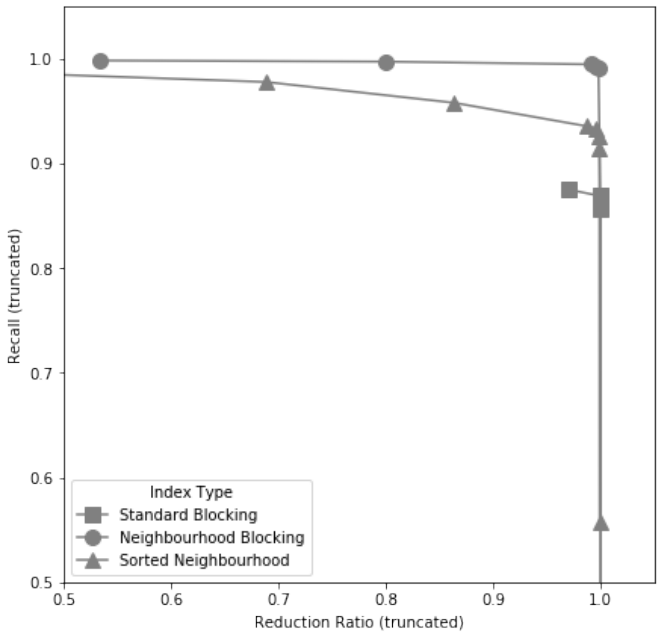

Fig. 16: Index Quality Frontiers - FEBRL3

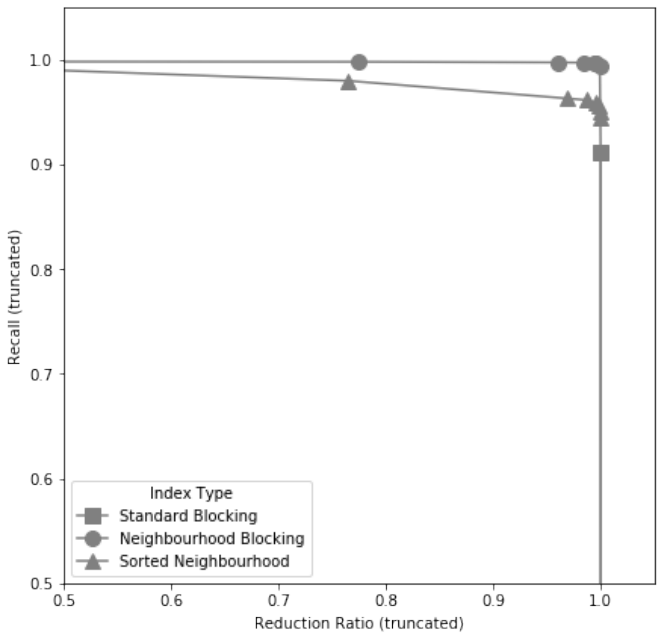

Fig. 17: Index Quality Frontiers - FEBRL4

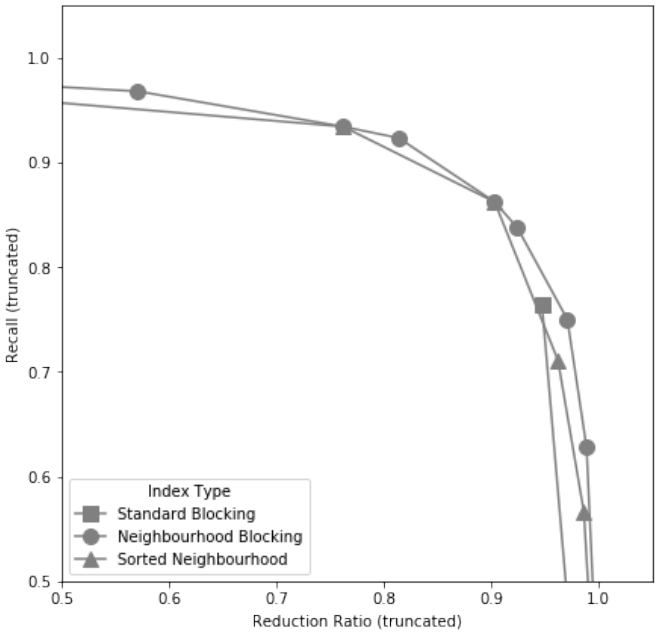

Fig. 18: Index Quality Frontiers - Abt-Buy

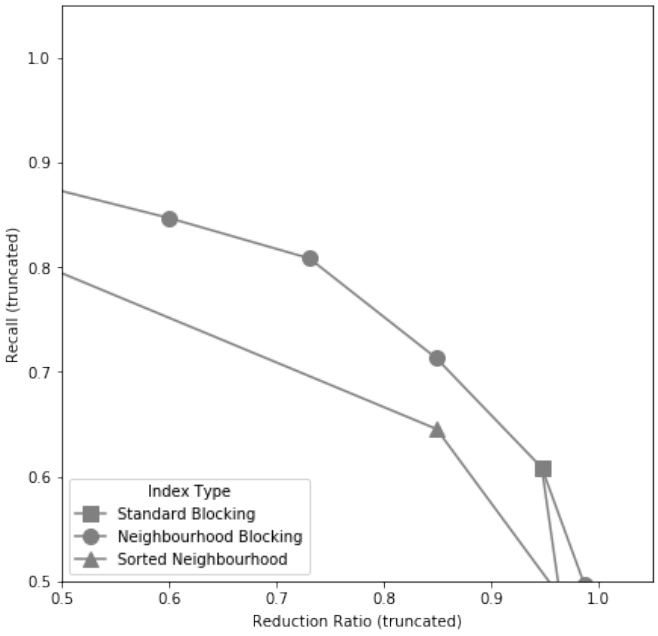

Fig. 19: Index Quality Frontiers - Amazon-GoogleProducts

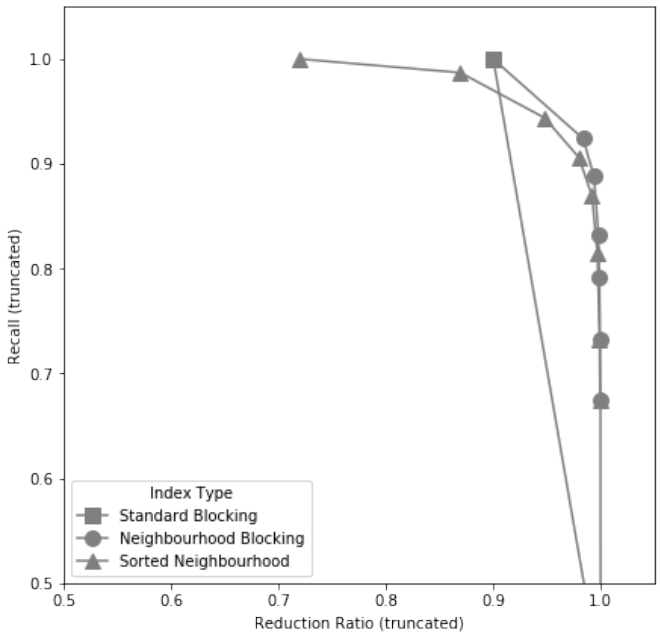

Fig. 20: Index Quality Frontiers - DBLP-ACM

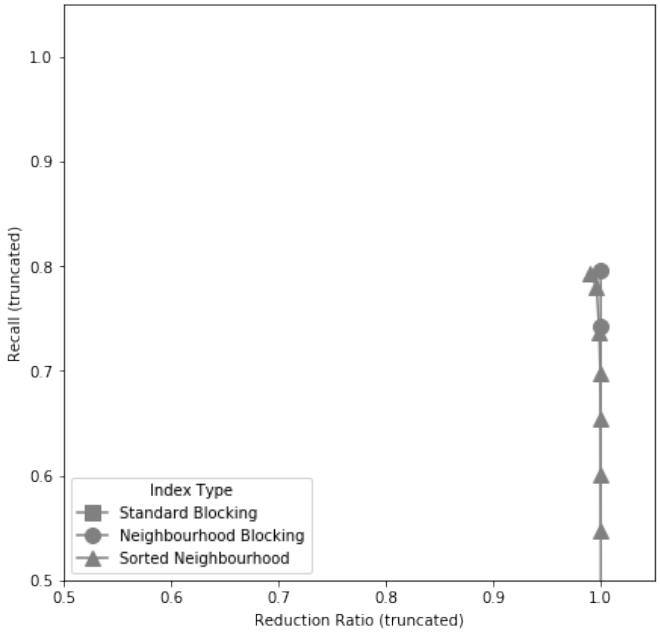

Fig. 21: Index Quality Frontiers - DBLP-Scholar

The results for all the FEBRL datasets (Figures 14 to 17) are broadly similar. Index quality is moderate to high and the ranking of index quality frontiers is the same. In all cases, the ranking is:

1) Neighbourhood Blocking with a near-perfect result
2) Sorted Neighbourhood Indexing
3) Standard Blocking

In the DBG Leipzig datasets(Figures 18 to 21), all three indexing methods produce far lower index quality than those achieved in the FEBRL datasets. [18] reports using string similarity measures in the indexing step to achieve higher index quality in these datasets.

In the case of the Amazon-GoogleProducts dataset (Figure 19), Neighbourhood Blocking did produce a noticeable improvement over the other methods, but as with all the DBG Leipzing datasets, the absolute quality of all the indexes suggests either the use of a Full Index or a more general value comparison method (as used by [18]).

VI. Numerical Experiments

In addition to the tests on benchmark datasets reported in Section V, numerical experiments were conducted to compare Neighbourhood Blocking with Standard Blocking and Sorted Neighbourhood Indexing. Two sets of experiments were performed: one focusing on index quality, the other on scalablity.

The scalability test also included Full Indexing (ie: production of all possible record pairs). This is a limiting case of all the other methods in that it is completely indiscriminate with respect to which record pairs are included.

A. Dataset Generation

Test datasets were generated using the following steps:

1) Specify:
   - r number of rows
   - c number of columns
   - d number of occurrences of each distinct entity
   - ϵ standard deviation of perturbations
   - p proportion null values
2) Produce an array of r rows $\times$ c columns. For each notional group of d consecutive rows (ie: groups of rows sharing the same value of

$\left\lfloor \frac{j}{d} \right\rfloor$ where j is the row index) populate the first row with uniformly distributed random numbers in the range $[0, 1]$ and copy these values to all rows in the group.

3) Perturb all values in the array by adding normally distributed perturbations drawn from $N(0, \epsilon)$
4) Overwrite the proportion p of all values in the table with Null

For the indexing methods requiring discrete keys (ie: Standard Blocking and Neighbourhood Blocking), the base dataset is discretized by computing:

$$\left\lfloor \frac{x}{u} \right\rfloor$$

where:

x is the value in base dataset, and
u is the unit of discretization $u > 0$

B. Index Quality Comparison

This experiment compared the quality of indexes produced by Neighbourhood Blocking, Standard Blocking and Sorted Neighbourhood Indexing. The measures used were recall and reduction ratio as described in Section V-B. The tradeoff between these for each indexing method was produced by varying parameters which affect the size of the index produced. For the datasets tested, it was found that Neighbourhood Blocking produced a tradeoff between these quality measures that was superior overall to those produced by the other methods.

1) Methodology: The parameters of the dataset used for the index quality comparison are summarized in Table VIII.

Item	Value
Columns	5
Distinct entities	500
Perturbed duplicates	1
Null values	1%

TABLE VIII: Dataset Parameters - Index Quality Test

Deduplication indexes were produced using various configurations of Standard Blocking, Sorted Neighbourhood Indexing and Neighbourhood Blocking.

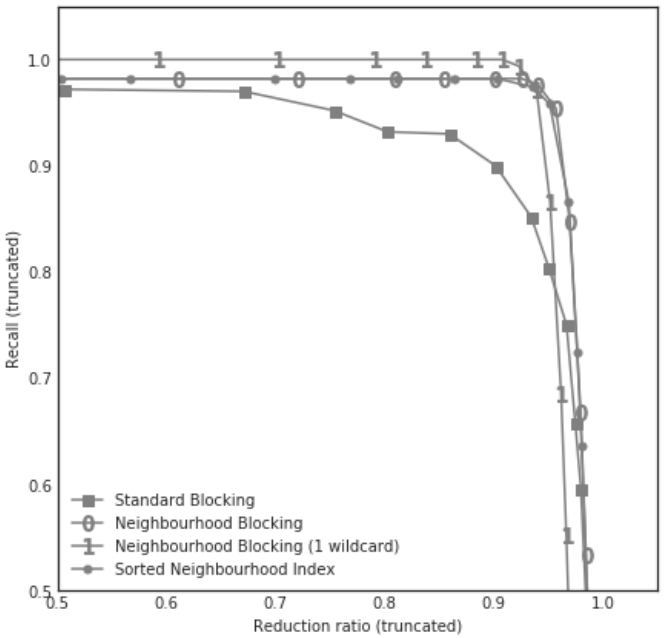

Fig. 22: Recall - Reduction Ratio Tradeoff - One Blocking Key

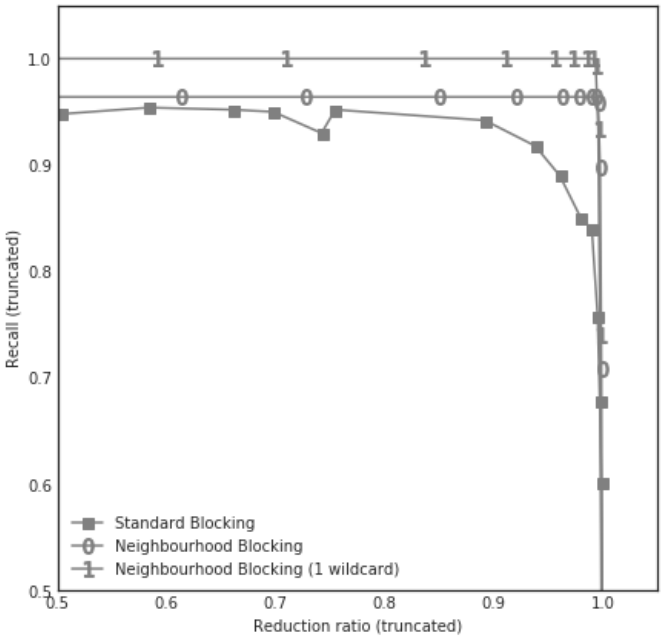

Fig. 23: Recall - Reduction Ratio Tradeoff - Two Blocking Keys

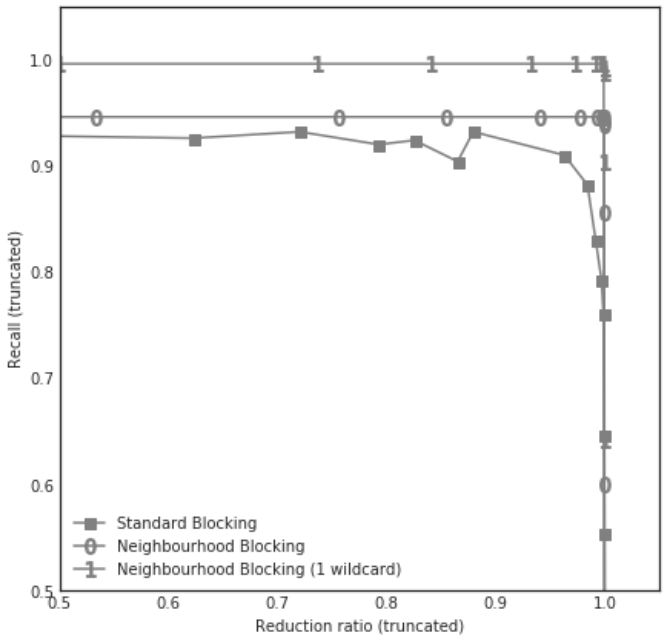

Fig. 24: Recall - Reduction Ratio Tradeoff - Three Blocking Keys

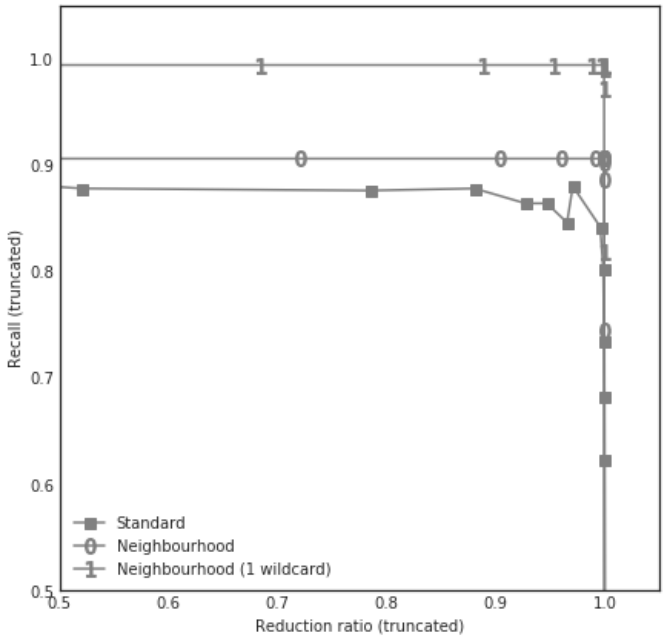

Fig. 25: Recall - Reduction Ratio Tradeoff - Five Blocking Keys

Records containing missing values were excluded from Standard Blocking and Sorted Neighbourhood Indexes.

All Neighbourhood Blocking configurations used a rank difference limit of 1 for all blocking keys.

For each basic index configuration, indexes of various sizes were produced by varying an appropriate "sizing" parameter. The relationship between recall (of the known true matches) and the index reduction ratio was observed. The index configurations which were compared together with their sizing parameters are summarized in Table IX.

2) Results: Figures 22 to 25 show relationships between recall and reduction ratio for indexes using 1, 2, 3 and 5 keys respectively. Figure 22 includes Sorted Neighbourhood Indexing because it is comparable to Neighbourhood Blocking with one sorting key. Note that in all these figures, both axes are truncated so as to begin at 0.5 instead of zero. This is because a purely random selection of half the possible record pairs would have a reduction ratio and *expected* recall of 0.5. The region of interest is therefore confined to that where *both* reduction ratio and recall exceed 0.5.

Sections VI-B2a to VI-B2e discuss several observations that can be made from these results.

a) Recall Ceiling: In all the experiments performed, each indexing method has a "recall ceiling" reflecting a limit on the recall that can be produced due to the nature of the indexing method. For each number of blocking keys (ie: in each of Figures 22 to 25), the ranking of the indexing methods in terms of their recall ceilings is the same. Specifically:

1) It is highest for Neighbourhood Blocking with wildcard matching
2) The next highest is Neighbourhood Blocking without wildcard matching. In the case of one blocking key (Figure 22), this is tied with Sorted Neighbourhood Indexing.
3) It is lowest for Standard Blocking.

This ordering is intuitive. The records in each matching pair lie close to one another in the key space, except for those records with missing values. When considering indexing configurations sufficiently indiscriminate to include half of all pos-

Index type	Configuration	Sizing
Sorted Neighbourhood		window size
Standard Blocking	number of keys	discretization granularity
Neighbourhood Blocking	number of keys; wildcard matches	discretization granularity

TABLE IX: Index configurations compared

sible record pairs, Neighbourhood Blocking with wildcard matching could potentially include all true matches, including those involving missing values (as is the case in these examples).

Where some records involved in true matches have missing BKVs (as is the case here), the exclusion of wildcard matching prevents any records containing missing BKVs from being included in the index. In all the indexing methods tested, this reduces the recall ceiling.

The recall ceiling for Standard Blocking is further reduced by its exclusion of record pairs that straddle of block boundaries. In order to achieve reduction ratios in the region of interest (ie: greater than 50%), Standard Blocking must include at least one block boundary.

b) Maximum Reduction Ratio: Almost all the indexing configurations used can achieve reduction ratios approaching 100% (ie: they can be made sufficiently restrictive to make the index very small). The only exception to this is in Figure 22 where the tradeoff curve for Neighbourhood Blocking with one wildcard match is bounded on the right at less than 100%. This is intuitive because if there is only one blocking key, any record with a null value in that key would have null values in *all* blocking keys (since there is only one key). The Wildcard criterion therefore matches any record with a missing value in that key to *all other records*. Preventing this from happening is a large part of the motivation for parameterizing the maximum number of wildcard matches.

As an aside, results for wildcard match limits greater than one are not shown in this report because in the dataset used, records with multiple missing BKVs were extremely rare. Consequently, increasing the wildcard match limit made almost no difference to the indexes produced.

c) Importance of Missing values vs Dimensionality: The impact of wildcard matching on the recall ceiling is indicated by the gap between the near-horizontal lines at the top of each of Figures 22 to 25. This is smallest in Figure 22 (where only one blocking key was used), increases steadily as the number of keys is increased and is largest in Figure 25 (where five blocking keys were used). This reflects the fact that all the keys used contain some missing values, and as the number of keys increases, so does the number of records that have *at least one* missing BKV. As mentioned in section VI-B2b, the test dataset contains almost no instances of multiple missing BKVs in the same record, so even with multiple keys, allowance for one wildcard match is still sufficient to allow near-100% recall.

d) Quality Tradeoff vs Dimensionality: In Figures 22 to 25, the "roundedness" of the top-right corner in each curve reflects the degree to which the corresponding indexing method exhibits a tradeoff between the two quality measures calculated (ie: reduction ratio and recall). To the extent that this top-right corner is rounded (as in Figure 22), a deterioration in reduction ratio must be tolerated in order to reach the indexing method's recall ceiling.

The indexing methods tested with varying dimensionality are Neighbourhood Blocking and Standard Blocking. For each of these, the tradeoff between the quality measures is larger when fewer blocking keys are used (although, as noted in section VI-B2a, blocking methods adversely impacted by missing values also have higher recall ceilings when there are fewer keys).

This is as would be expected because the parameter used to control the size of the index is the granularity of the blocking key(s). Blocks of the same volume (to produce the same index size) produced by partitioning fewer dimensions will have more extreme shape (long and thin). When the thickness of the blocks becomes comparable to the

distances between matching pairs of points, further reducing the block thickness (to increase reduction ratio) would have a large impact on recall.

e) Quality Tradeoff vs Indexing Method: As mentioned in Section VI-B2d, the points on the frontier (upwards and rightwards) in Figures 22 to 25 represent the best tradeoff between reduction ratio and recall within the population of indexes produced. In all the cases tested, these frontiers are produced by Neighbourhood Blocking. In the case of one blocking key, part of the frontier is produced by including Wildcard matching, and part by excluding it. The part formed by excluding Wildcard matching is also produced by Sorted Neighbourhood Indexing.

C. Scalability Comparison

This experiment compared the runtime used by each of the indexing methods when they are applied to datasets of varying size and composition. Indexing configurations varied by window size (or granularity of blocking key discretization), degree of wildcard matching and number of blocking keys. Databases varied by overall size and by the number of rows representing each distinct entity.

It was found that runtime for Neighbourhood Blocking depends most strongly on two factors:

- How sparsely populated the blocks are (ie: the ratio of the number of records in the database to the number of distinct combinations of BKVs)
- The size of the final index produced

When database sparsity is low:

- runtime for all three indexing methods is approximately linear in the size of the index produced, and
- the rates of index production for Standard and Neighbourhood blocking are similar

1) Methodology: The production of deduplication indexes was parameterized using the following variables:

a) Method: The indexing method used (Full Index, Standard Blocking, Neighbourhood Blocking or Sorted Neighbourhood)

b) Rows: The number of rows in the table to be deduplicated

c) Blocking Keys: The number of blocking keys to use.

d) Entity Instances: Each distinct entity in the dataset had this number of rows representing it (with values slightly perturbed). The number of distinct entities is therefore the (rounded) quotient of the number of rows and the number of instances per entity.

e) Granularity: The main parameter determining how discriminative the indexing method is. For methods based on blocking, this is the number of distinct values allowed by the discretization of each blocking key. For Sorted Neighbourhood Indexing, granularity is the (rounded) ratio of the number of rows to the window size. In addition, negative values of granularity were used to directly specify half window size (which is only applicable to Sorted Neighbourhood Indexing).

The values that these variables were allowed to take are summarized in Table X.

Variable	Values		
	number of values	bounds	distribution
Method	Neighbourhood Blocking (no wildcards, no proximity) Neighbourhood Blocking (no wildcards) Neighbourhood Blocking (1 wildcard) Neighbourhood Blocking (2 wildcards) Sorted Neighbourhood Indexing Standard Blocking Full Index		
Rows	16	10 ... 1,000,000	geometric
Blocking keys	10	1 ... 10	linear
Entity instances	5	1 ... 100	geometric
Granularity	10	1 ... 1,000	geometric
	10	-100 ... -1	geometric

TABLE X: Scalability test - database and indexing parameter values

Except in the "no proximity" case, all configurations of Neighbourhood Blocking used rank difference limits of 1 for all blocking keys.

Indexing was performed and timed for each combination of values for these variables, except where these were invalid, not meaningful for the index type, or were estimated to exceed resource or time constraints. The following rules determined which combinations of parameter values were used:

- The tests with fewer rows were run first. Limits were placed on the index size and resource consumption. When an indexing configuration caused these to be exceeded, similar configurations with more rows were omitted. The conditions relating to scale limits used were:
    - Index size exceeds 10,000,000 pairs
    - Runtime exceeds 30 seconds
    - Memory usage exceeds 7GB
- A Full Index was only computed once for each distinct number of rows
- A Sorted Neighbourhood Index was only computed when the number of blocking keys was 1, and only once for each distinct combination of granularity and number of rows.
- Extreme or uninformative combinations were excluded, such as granularity exceeding the number of rows, or the number of wildcard matches exceeding the number of blocking keys.
- Negative granularity was only used with Sorted Neighbourhood Indexing (this was used to specify the half-window)

2) Results:

a) Timing vs Index Length: Index timings are shown in Figures 26 to 32. Each of these figures relates to a particular indexing configuration and shows the relationship between index size and runtime. The scales on the axes are consistent across these figures, and ordinary Least Squares lines of best fit are included. Details of these lines of best fit are shown in the upper portion of Table XI (ie: the rows indicating no filtering).

b) Stable relationship between index size and timing: Figures 26 to 28 have broadly the same shape and their lines of best fit are almost identical. This indicates that although the indexing algorithm parameters significantly impact index size, they appear to have little impact on *the relationship between index size and runtime*.

c) Linear lower bounds on indexing time: Another observation that can be made about all of Figures 26 to 32 is that runtime in all cases is subject

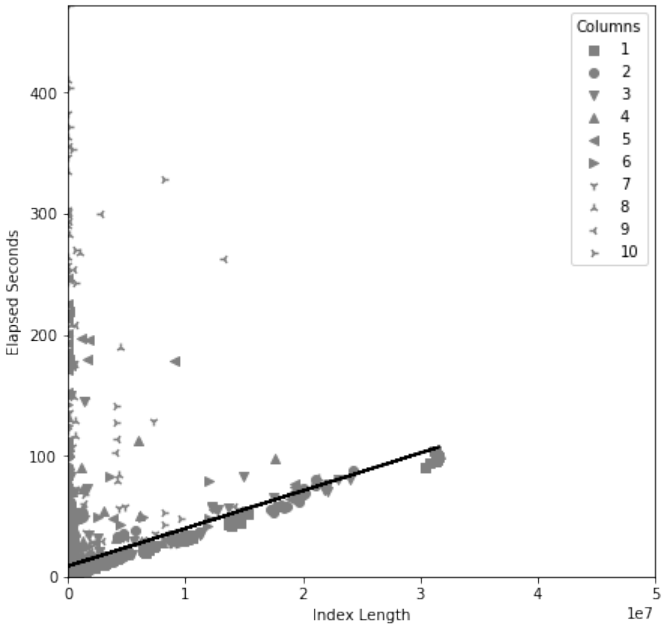

Fig. 26: Index Timings: Neighbourhood Blocking (No wildcards)

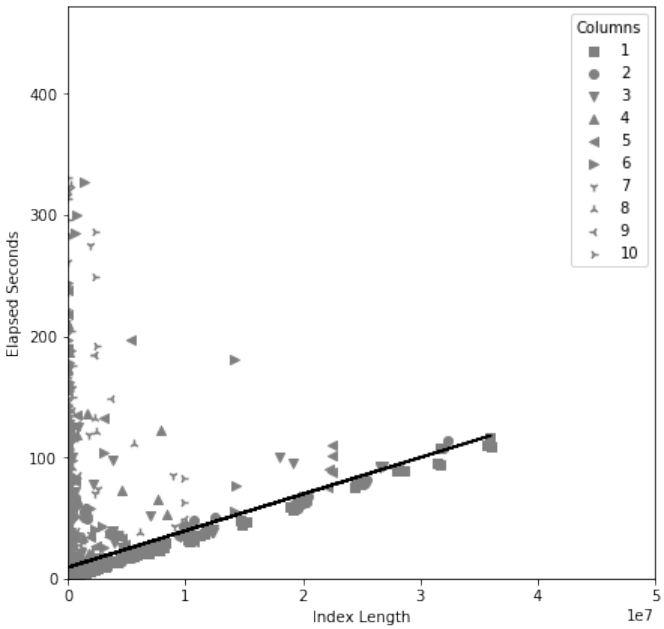

Fig. 27: Index Timings: Neighbourhood Blocking (1 wildcard)

Method	Filtering	Intercept	Slope	R^2
Neighbourhood Blocking - No wildcards or adjacency		4.92	3.18	0.33
Neighbourhood Blocking - no wildcards		8.68	3.12	0.08
Neighbourhood Blocking - 1 wildcard		9.23	3.02	0.09
Neighbourhood Blocking - 2 wildcards		10.27	3.15	0.07
Full		1.19	1.08	0.99
Sorted Neighbourhood		1.22	10.34	0.96
Standard Blocking		1.55	3.23	0.82
Neighbourhood Blocking	Non-sparse	0.26	3.20	0.99
Standard Blocking	Non-sparse	0.23	3.13	0.99

TABLE XI: Indexing times - lines of best fit by method

to a lower bound which is approximately linear in the index size.

d) Impact of block sparsity: The relationships shown in Figures 26 to 28 look almost like a tilted "V" in that the great majority of observations are close to either the vertical axis or the line representing the lower bound of the observed timings. Points close to the vertical axis indicate a very long time was taken to produce a very small index. Figures 33 and 34 provide an insight into why this is so. These are scatterplots of indexing speed against block sparsity. In these figures, the different configurations of Neighbourhood Blocking are not distinguished from one another due to their similarity as noted in section VI-C2b. The variables on the axes of Figures 33 and 34 are:

Index Production Rate: The ratio of the index size to the time taken to produce it. For a particular index timing, this corresponds to the slope of a line between the origin and a point in Figures 26 to 32. Index production rate is shown on the vertical axis of Figures 33 and 34.

Log_{10} (possible blocks / rows) "possible blocks" here means the number of possible distinct combinations of BKVs. That is: the product of the numbers of distinct valid values of the blocking keys. The ratio of this number to the number of rows in the dataset is monotonically related to the expected number of rows in each block and is therefore a measure of how sparsely the blocks are populated with records. This variable is shown on the horizontal axis of Figures 33 and 34.

Figure 33 is a scatterplot of index production rate against block sparsity for all indexes produced using Neighbourhood Blocking, Standard Blocking and Sorted Neighbourhood Indexing. It shows that the overall relationships between index production rates and sparsity are very similar for Standard and Neighbourhood Blocking, and that these are distinct from the corresponding relationship for Sorted Neighbourhood Indexing. Figure 34 is the same plot except that it omits points relating to indexes which took less than a second to compute.

Some observations can be made about Figure 34:

- The production rate for Sorted Neighbourhood Indexing doesn't depend on block sparsity. This is as expected, since blocks are irrelevant to Sorted Neighbourhood Indexing.
- The range of production rates observed is strongly related to sparsity for both Standard and Neighbourhood Blocking.
- The relationships between production rate and sparsity for Standard Blocking and Neighbourhood Blocking are very similar. The points relating to Neighbourhood Blocking (grey circles) form a sigmoid shape band with a narrow range of the highest production rates at the lowest sparsity, and a much wider range of generally lower production rates at high sparsity. The region occupied by the points relating to Standard Blocking (black squares) is almost identical to that relating to Neighbourhood Blocking.

A key implication here is that when each block contains many rows, index production speeds for Standard Blocking and Neighbourhood Blocking are similar. This is intuitive, since the primary difference between the two methods is that Neighbourhood Blocking includes the task of identifying matches

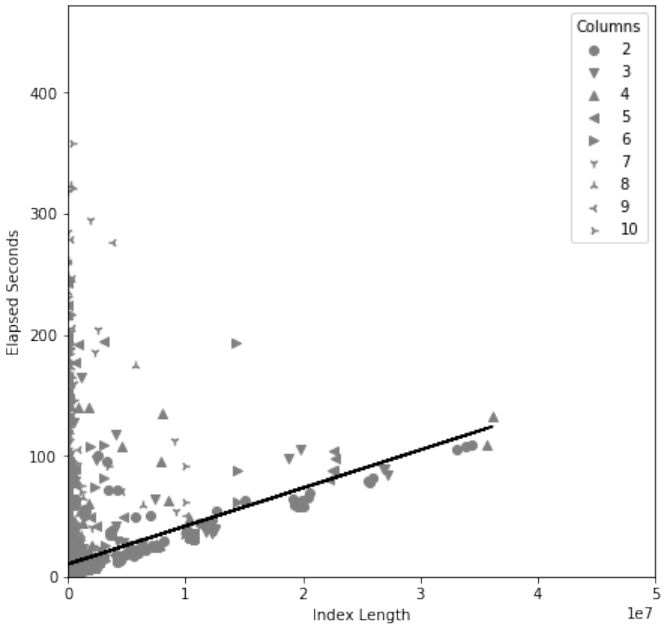

Fig. 28: Index Timings: Neighbourhood Blocking (2 wildcards)

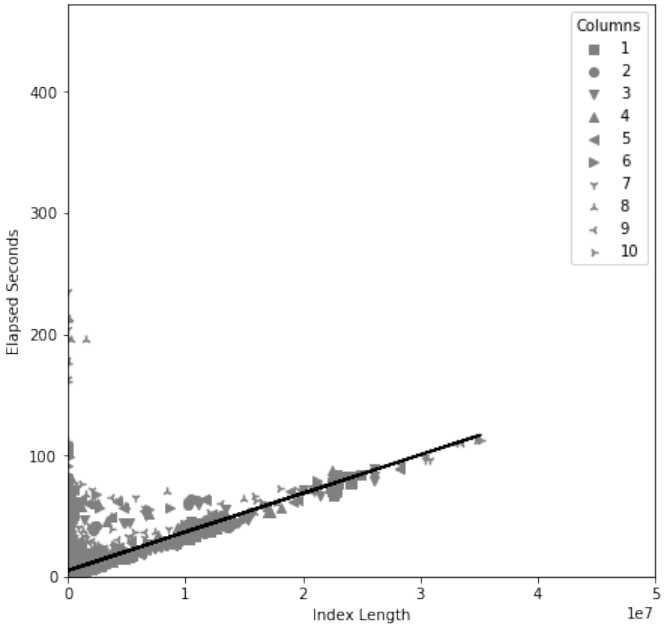

Fig. 29: Index Timings: Neighbourhood Blocking (No wildcards, no sorting keys)

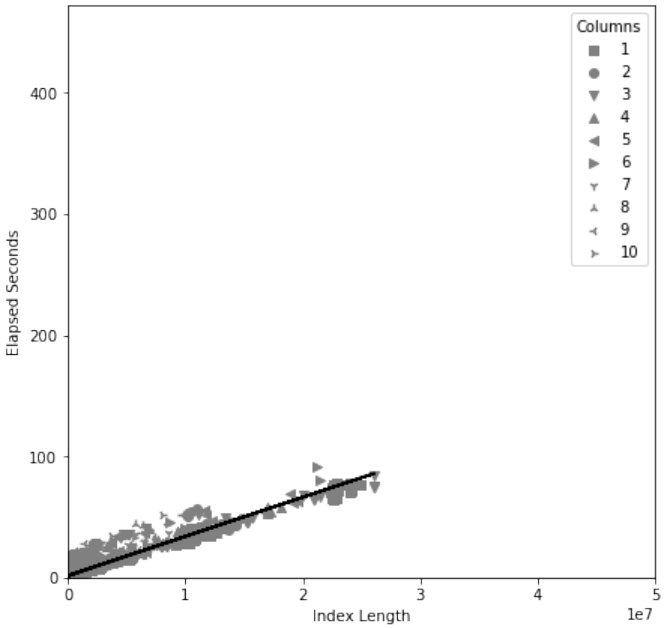

Fig. 30: Index Timings: Standard Blocking (all configurations)

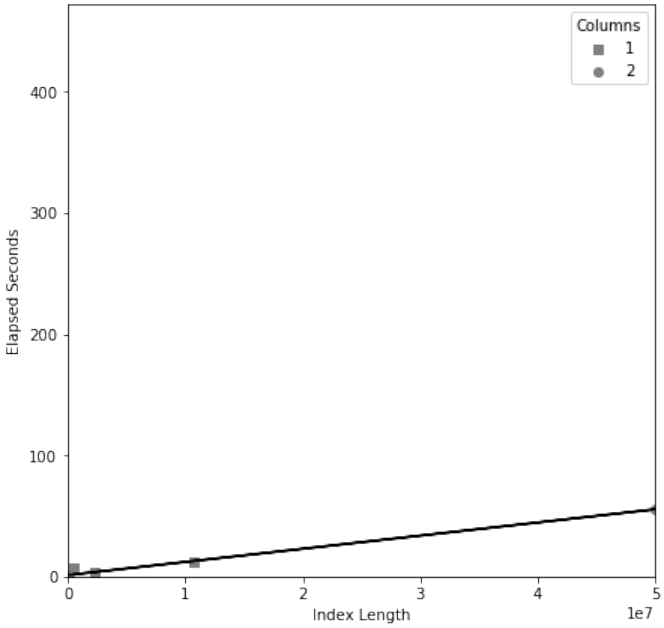

Fig. 31: Index Timings: Full Index

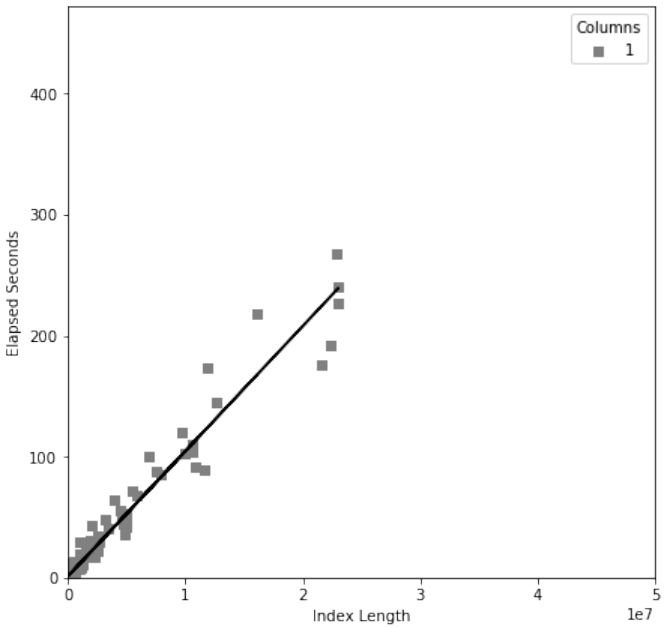

Fig. 32: Index runtime vs size - Sorted Neighbourhood

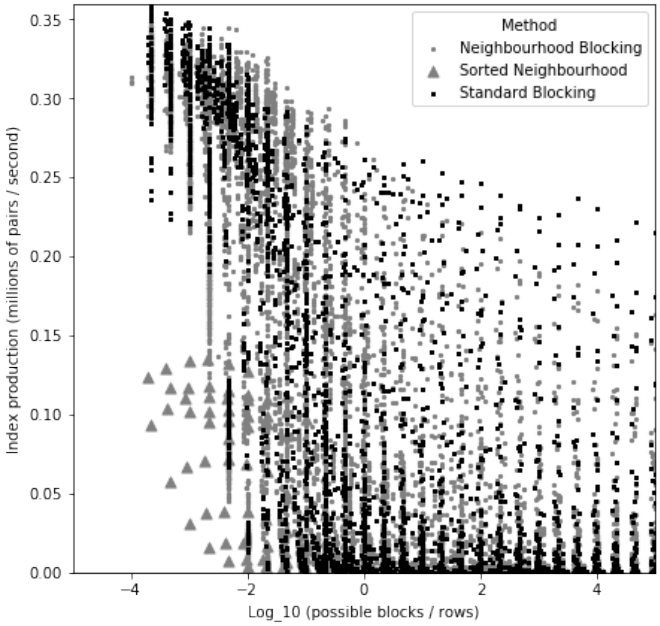

Fig. 33: Index production rates

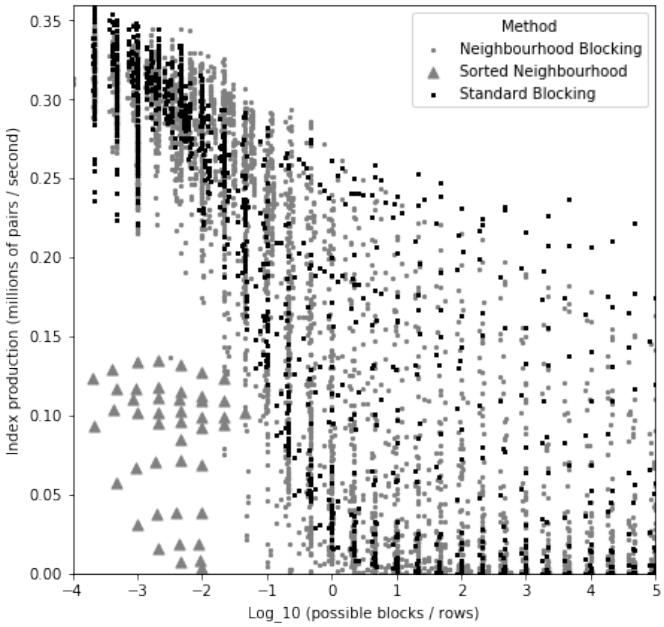

Fig. 34: Index production rates for timings exceeding 1 second

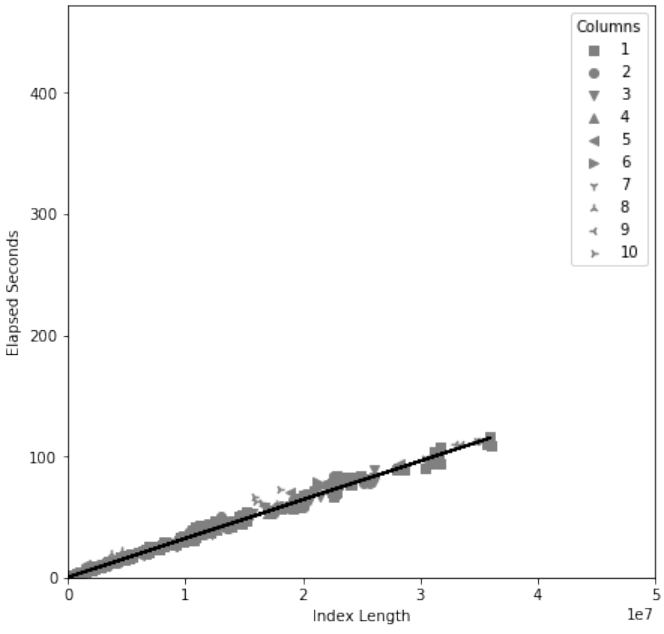

Fig. 35: Index Timings: Neighbourhood Blocking (All configurations, low sparsity)

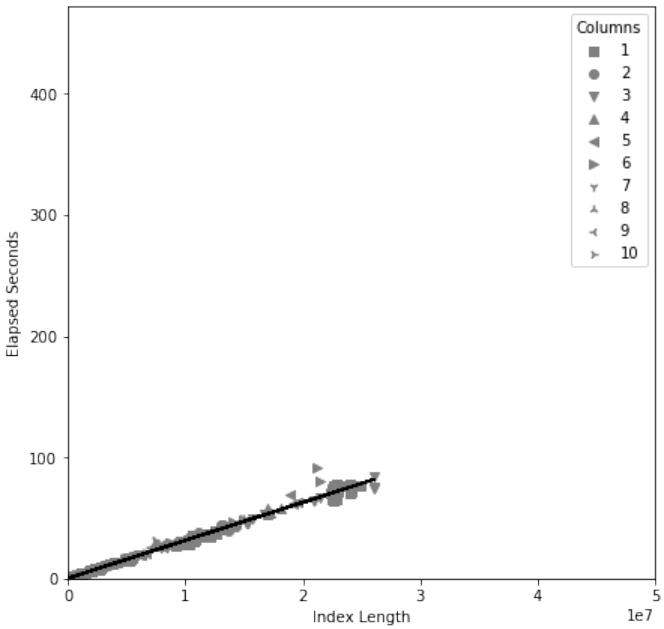

Fig. 36: Index Timings: Standard Blocking (low sparsity)

between blocks. The time taken to do this is independent of the number of records contained in the blocks.

Figures 35 and 36 illustrate the effect of controlling for sparsity. These graphs are the same as Figures 26 to 30 except that they omit cases involving datasets where there are fewer than 100 records per possible combination of BKVs (also, timings for all Neighbourhood Blocking configurations are combined in Figure 35). In both cases, excluding sparse datasets makes the relationship between index size and runtime almost completely linear (R^2 increases to 99%). Details of the lines of best fit in Figures 35 and 36 are shown in the lower portion of Table XI. This is a key observation: *Index production rates for Standard and Neighbourhood Blocking converge in sufficiently dense databases*.

VII. Discussion

A. Applicability

The impact of block sparsity on index production rate mentioned in section VI-C2d means that compared to Standard Blocking, Neighbourhood Blocking *on the same dataset to produce an index of the same size* is more likely to be slower than faster. This is because for the indexes to be the same size, the blocks used in Neighbourhood Blocking must contain fewer records per block (ie: they must be sparser) than the blocks used in Standard Blocking.

As noted in Section IV-E, Standard Blocking and Sorted Neighbourhood Indexing are both special cases of Neighbourhood Blocking. Therefore, Neighbourhood Blocking can only offer an improvement over these simpler methods in cases that can make use of its incremental features, namely:

- Wildcard matching of missing values
- Proximity matching using multiple sorting orders
- Allowance for a limited number of field mismatches

These apply when the dataset contains missing values, or when there are multiple blocking keys with meaningful sort orders in which block proximity provides an imperfect indication of record matches. The FEBRL benchmark datasets and the random datasets used in Section VI satisfy these conditions.

Because Neighbourhood Blocking is generally slower than these similar, simpler indexing methods, its use is only justified in cases where it produces indexes of superior quality (if this allows the index to be smaller, time might even be saved in the subsequent Comparison and Classification steps).

B. Index Size

Indexes produced using Neighbourhood Blocking can be much larger than those produced using Standard Blocking *with the same blocking keys*. This can happen when field mismatches are allowed, when missing values are such that they produce a large number of wildcard matches (as discussed in section VI-B2b) or when proximity matching is allowed in even a moderate number of blocking keys (by Theorem IV.3). This can represent an opportunity to either use more blocking keys or more granular ones. For example, granularity can be chosen arbitrarily for keys that are discretized versions of continuous variables. By Theorem IV.2, when rank distance limits are increased from zero to one, key

granularity can be *at least* doubled without loss of recall but, it may be desirable to increase it much more. As discussed in section IV-G, Neighbourhood Blocking doesn't exclude nearby record pairs that straddle block boundaries. Therefore, the Standard Blocking's "cost of boundary size" illustrated in Figure 5 doesn't apply to Neighbourhood Blocking. The unit of discretization in such keys can therefore be reduced to something resembling the maximum distance between truly matching records since (by Theorem IV.1) all pairs of records no further apart than the block size are included in the index.

C. Index Quality

Allowing field mismatches or adding of wildcard or proximity matching to Standard Blocking unambiguously improves recall through the inclusion of more record pairs. Similarly, changing from a single record ordering (as is used in Sorted Neighbourhood Indexing) to multiple sort orderings reduces the inclusion of distance pairs like CG and AH in Figures 6 and 7. Furthermore, as discussed in section IV-G, Neighbourhood Blocking does not imply transitive closure meaning distant pairs of points can be excluded from the index even if there is a sequence of points between them where each successive pair of points is included in the index.

These features indicate situations where either recall or reduction ratio can be improved by the use of Neighbourhood Blocking instead of Standard Blocking or Sorted Neighbourhood Indexing. However, a key observation in Sections V and VI-B is that there are cases where Neighbourhood Blocking delivers *simultaneous* improvements in both recall and reduction ratio.

D. Scalability

As noted in Section VII-A Neighbourhood Blocking is typically slower than Standard Blocking at producing an index *of the same size on the same data*. However, this is largely because the blocks used in Neighbourhood Blocking need to more sparsely populated by records if the indexes are to be the same size. As reported in Section VI-C2d, this sparsity effect impacts both Standard Blocking and Neighbourhood Blocking. Its effect on the speed of both indexes is similar in magnitude and occurs at similar levels of sparseness. Adding rows to the database without increasing the number of distinct BKVs decreases sparseness. This improves index production rates for both Standard and Neighbourhood Blocking up to a limit. This indicates that in sufficiently large datasets, index production speeds are similar for Standard Blocking and Neighbourhood Blocking, even if they are very different in smaller datasets. It should be noted that, depending on the number of sorting keys and the number of distinct BKVs that each of these has, "Sufficiently large" in this context could far exceed the size of practical databases.

VIII. Conclusion

Neighbourhood Blocking shares many features with Standard Blocking and Sorted Neighbourhood Indexing. It combines their respective advantages of multidimensionality and proximity matching. In addition, it adds wildcard matching of missing values and allowance for limited field mismatches. These are found to result in superior index quality both in benchmark datasets and randomly generated ones.

Scalability tests indicate similar index production speeds for Neighbourhood Blocking and Standard Blocking in sufficiently large datasets, but that the database size at which this happens could be extremely large. At smaller database sizes, Standard Blocking can be significantly faster.

The relative usefulness of Neighbourhood Blocking therefore depends on it producing a sufficient improvement in index quality to justify its typically slower speed. Factors increasing the likelihood of this include: a significant number of missing values; multiple attribute fields in which sorting provides an imperfect indication of similarity; continuous-valued keys; multiple records in most blocks.

IX. Further Work

This work could be extended by making a progressive version of Neighbourhood Blocking. This would be similar to Progressive Blocking as described by [10], except that additional match types would be allowed and block proximity would be determined by multiple sorting orders.

Another possible extension is an implementation enhancement to reduce the impact of block sparsity on runtime. A prominent possibility for addressing this is to introduce a heuristic rule to choose the degree of key coarsening to apply at each step of the recursive implementation described in Section IV-D2. The implementation tested here doubles key coarseness at each stage, but larger multiples can be used as well.

ACKNOWLEDGMENT

I thank Josiah Poon at the University of Sydney for reviews of drafts and for many valuable discussions.

REFERENCES

[1] Halbert L. Dunn. "Record Linkage". In: *American Journal of Public Health and the Nation's Health* 39 (1946), pp. 1412–1416.

[2] Ivan P. Fellegi and Alan B. Sunter. "A Theory for Record Linkage". In: *Journal of the American Statistical Association* 64 (1969), pp. 1183–1210.

[3] D. Randall Wilson. "Beyond Probabilistic Record Linkage: Using Neural Networks and Complex Features to Improve Genealogical Record Linkage". In: *Proceedings of International Joint Conference on Neural Network*. San Jose, California, USA: IEEE, 2011.

[4] Peter Christen. *Data Matching*. Springer, 2012. ISBN: 978-3-642-31164-2.

[5] Peter Christen. "A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication". In: *IEEE Transactions on Knowledge and Data Engineering* 24 (2012), pp. 1537–1555.

[6] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. "Duplicate Record Detection: A Survey". In: *IEEE Transactions on Knowledge and Data Engineering* 19.1 (2007), pp. 1–16. ISSN: 1041-4347. DOI: 10.1109/TKDE.2007.250581.

[7] Thomas N. Herzog, Fritz J. Scheuren, and William E. Winkler. *Data Quality and Record Linkage Techniques*. Springer, 2007. ISBN: 978-0-387-69502-0.

[8] Federico Maggi. "A Survey of Probabilistic Record Matching Models, Techniques and Tools". PhD thesis. DEI, Politecnico di Milano, 2008.

[9] Periklis Andritsos, Ariel Fuxman, and Rene J. Miller. "Clean Answers over Dirty Databases: A Probabilistic Approach". In: *Proceedings of the 22nd International Conference on Data Engineering (ICDE06)*. Atlanta, GA, USA: IEEE, 2006.

[10] T. Papenbrock, A. Heise, and F. Naumann. "Progressive Duplicate Detection". In: *IEEE Transactions on Knowledge and Data Engineering* 27.5 (2015), pp. 1316–1329. ISSN: 1041-4347. DOI: 10.1109/TKDE.2014.2359666.

[11] K. Abou-Moustafa. "What Is the Distance Between Objects in a Data Set?: A Brief Review of Distance and Similarity Measures for Data Analysis". In: *IEEE Pulse* 7.2 (2016), pp. 41–47. ISSN: 2154-2287. DOI: 10.1109/MPUL.2015.2513727.

[12] Shaun J. Grannis, J. Marc Overhage, Siu Hui, et al. "Analysis of a Probabilistic Record Linkage Technique without Human Review". In: *AMIA 2003 Symposium Proceedings*. Washington, DC, USA: AMIA, 2003.

[13] Steven Euijong Whang and Hector Garcia-Molina. "Joint entity resolution on multiple datasets". In: *The VLDB Journal* 22 (2013), pp. 776–795.

[14] Simon K. POON, Josiah POON, Mary K. LAMb, et al. "An Ensemble Approach for Record Matching in Data Linkage". In: *Studies in health technology and informatics* 227 (2016), pp. 113–119.

[15] Mauricio A. Hernandez and Salvatore J. Stolfo. "The Merge/Purge Problem for Large Databases". In: *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*. San Jose, California, USA: ACM, 1995.

[16] Bada Ramadan, Huizhi Liang Peter Christen, and Ross W. Gayler. "Dynamic Sorted Neighborhood Indexing for Real-Time Entity Resolution". In: *Journal of Data and Information Quality* 6 (2015).

[17] Khairul Nizam B. and Shahrul Azman M.N. "Efficient identity matching using static pruning q-gram indexing approach". In: *Decision Support Systems* 73 (2015), pp. 97–108.

[18] H. Kopcke, A. Thor, and E. Rahm. "Learning-Based Approaches for Matching Web Data Entities". In: *IEEE Internet Computing* 14.4 (2010), pp. 23–31. ISSN: 1089-7801. DOI: 10.1109/MIC.2010.58.