blocking: An R Package for Blocking of Records for Record Linkage and Deduplication

by Maciej Beręsewicz and Adam Struzik

Abstract Entity resolution (probabilistic record linkage, deduplication) is essential for estimation based on multiple sources. It aims to link records without common identifiers that refer to the same entity (e.g., person, company). Without identifiers, researchers must specify which records to compare to calculate matching probability and reduce computational complexity. Traditional deterministic blocking uses common variables like names or dates of birth, but assumes error-free, complete data. To address this limitation, we developed the R package **blocking**, which uses approximate nearest neighbour search and graph algorithms to reduce number of comparisons. This paper presents the package design, functionalities, and two case studies.

1 Introduction

1.1 Blocking for record linkage

Entity resolution (probabilistic record linkage, deduplication) is essential for estimation based on multiple sources (cf. Fellegi and Sunter (1969), Binette and Steorts (2022)). The goal is to link records without common identifiers that refer to the same entity (e.g., person, company, job position). This situation is frequently observed in administrative records, particularly for foreign-born populations. For instance, the Social Insurance Institution register in Poland at the end of 2023 included 1.206 million records referring to approximately 1.105 million individuals, of which about 10% had missing information in the personal identifier (PESEL) and about 50% had missing address details. Note that the exact number of individuals is certainly lower than 1.105 million, as the 10% with missing identifiers may include duplicates (cf. Beręsewicz (2025)).

This drives the need to link records without identifiers, which often requires certain assumptions about how to reduce the large number of possible comparisons, as it is not feasible to compare all pairs of records in large datasets (e.g., the aforementioned example would require over 6 billion comparisons). Consequently, researchers aim to reduce the number of comparisons in various ways prior to the record linkage/deduplication stage. The rationale is twofold: computational resource constraints and clerical review workload.

Reducing the number of comparisons is accomplished through blocking, a method that limits possible comparisons by assuming that certain variables must match exactly or that combinations of variables should match above a specified threshold. For instance, a standard approach assumes that sex or birth year must match exactly, whilst other record characteristics may vary. Another method employs phonetic algorithms such as Soundex (cf. Wright (1960)) or its adaptations for non-English languages (cf. 1) to block records that sound similar but are spelt differently (e.g., Smith and Smyth, or Anna and Ania). Furthermore, with the growing popularity of large language models (both closed and open-source), one may consider using embeddings (Mikolov et al., 2013) to identify nearest neighbours and treat these as potential comparison pairs. For a comprehensive review of blocking methods, see Steorts et al. (2014) or Papadakis et al. (2020). Section 1.2 discusses existing R packages that implement blocking methods.

Reducing the number of pairs has inherent costs: missed comparisons lead to increased false positive rates (FPR) and false negative rates (FNR) in linkage studies. To assess these errors, a subset of true pairs should be provided, or simulation studies of proposed methods should be conducted. Alternatively, one may consider approaches proposed by Dasylva and

Goussanou (2021) and Dasylva and Goussanou (2022), who demonstrated how to estimate FPR and FNR without access to an audit sample.

1.2 Existing software and our contribution

The R ecosystem offers several packages that implement various blocking techniques, which we have grouped according to the following classification:

- **Deterministic blocking**:
  - **reclin2** (van der Laan (2024), van der Laan (2022)) allows pairing records using `pair_blocking()` with a prespecified list of columns in a `data.frame`, and the `pair_minsim()` function, which allows specifying the minimal similarity score (e.g., 1 out of 3 variable values must match exactly).
  - **RecordLinkage** (Sariyar and Borg (2025), Sariyar and Borg (2010)) allows specifying blocking variables in the `blockfld` parameter of either `compare.dedup()` or `compare.linkage()` functions as a vector (either character or numeric).
  - **fastLink** (Enamorado et al. (2023), Enamorado et al. (2019)) implements various blocking methods via the `blockData()` function, including exact matching, window matching (e.g., no more than 2-year difference between birth years), and k-means clustering. Notably, `fastLink` returns datasets split into separate lists, whilst `reclin2` and `RecordLinkage` packages create a single dataset.
- **Phonetic blocking**:
  - **RecordLinkage** allows direct specification of phonetic comparison via the `phonetic` argument in `compare.dedup()` or `compare.linkage()` functions using the `soundex()` function. However, this is used for string comparison rather than blocking.
  - Additionally, **stringdist** (van der Loo, 2014) implements the Soundex algorithm, whilst **phonics** (Howard, II (2021),
    1) implements various phonetic algorithms that can be applied prior to the blocking procedure (e.g., to create a new column).
- **Probabilistic blocking**:
  - **klsh** (Steorts, 2020) is the only R package that implements probabilistic blocking using the k-means variant of locality sensitive hashing. The main `klsh()` function implements this approach, and the resulting object is a list containing row identifiers for the prespecified number of blocks (via the `num.blocks` argument).

In practice, the situation is more complicated, as missing data may be present in blocking/matching variables (such as birth dates) or typos may occur in names and surnames. Therefore, we developed **blocking**, which leverages approximate nearest neighbour (ANN) algorithms and graphs to create numerous small blocks that can be used in subsequent analysis (this approach is similar to micro-clustering; cf. Johndrow et al. (2018)). The basic workflow of the **blocking** package consists of the following steps:

1. Create shingles of the input character vectors using **tokenizers** (Mullen et al., 2018) and **text2vec** (Selivanov et al., 2023) packages, or provide a matrix of vectors (e.g., embeddings via **ragnar**, Kalinowski and Falbel (2025)) representing the input character vectors.
2. Search for nearest neighbours using ANNs algorithms implemented in **rnndescent** (Melville, 2024b), **RcppHNSW** (Melville, 2024a), **mlpack** (Curtin et al. (2023), Singh Parihar et al. (2025)), and **RcppAnnoy** (Eddelbuettel, 2024).
3. Create final blocks using **igraph** (Csárdi et al. (2025),Csardi and Nepusz (2006)).

This is the only package in the R ecosystem that readily applies modern ANNs algorithms to reduce the number of comparisons and significantly accelerate record linkage and deduplication tasks. Additionally, we have developed the `pair_ann()` function for seamless integration with the **reclin2** package, as described in one of the package vignettes.

1.3 Outline of article

This paper is structured as follows. Section 2 provides a description of the main functionalities of the **blocking** package and how results can be assessed. Section 3 presents two case studies: probabilistic record linkage and deduplication. These examples demonstrate how our package can improve the entity resolution pipeline and integrate with existing R packages.

2 Blocking of records using `blocking()` function

2.1 The main function

The main functionality is available via the `blocking()` function, which contains the following key arguments:

- `x, y` – reference vectors, where `y = NULL` indicates that deduplication is applied;
- `representation` – whether `x` and `y` should be represented as shingles or vectors (e.g., provided by the user via the `model` argument);
- `ann` – which ANNs algorithm should be applied (by default, we use the **rnndescent** package as it supports sparse matrices);
- `distance` – which distance metric should be applied (default is `cosine` distance);
- `graph` – whether a plot of the graph showing connected records should be returned (default `FALSE`);
- `true_blocks` – if a subset of true blocks is available, it can be provided here so that quality measures, presented in the next section, are returned;
- `n_threads` – number of threads used for computation;
- `control_txt` – controls provided via `controls_txt()` specifying how `x, y` are processed;
- `control_ann` – controls provided via `controls_ann()` allowing users to fine-tune the ANNs algorithm (see documentation for the `controls_ann()` function and `control_*` functions with names referring to specific algorithms, e.g., `control_nnd()` for the NND algorithm).

This function returns an object of class `blocking` containing the following elements:

- `result` – a `data.table` with indices (rows) of x, y, block, and distance between points;
- `method` – name of the ANNs algorithm used;
- `deduplication` – information about whether deduplication was applied;
- `representation` – information about whether shingles or vectors were used;
- `metrics` – quality assessment metrics, if `true_blocks` is provided;
- `confusion` – confusion matrix, if `true_blocks` is provided;
- `colnames` – variable names (`colnames`) used for search;
- `graph` – an `igraph` class object.

2.2 Assessment of results

The package implements several measures that can be used to assess results. The first is the *reduction ratio* (RR), which indicates the reduction in comparison pairs within the given blocks. It has a value between $[0, 1]$, where 1 indicates perfect reduction whilst values close to 0 indicate poor reduction. The RR indicator for deduplication has the following form:

$$\mathrm{RR}_{\mathrm{dedup}} = 1 - \frac{\sum_{i=1}^{k} \binom{|B_i|}{2}}{\binom{n}{2}},$$

where k is the total number of blocks, n is the total number of records in the dataset, and $|B_i|$ is the number of records in the i-th block. $\sum_{i=1}^{k} \binom{|B_i|}{2}$ is the number of comparisons after blocking, whilst $\binom{n}{2}$ is the total number of possible comparisons without blocking. For record linkage, the reduction ratio is defined as follows:

$$\mathrm{RR}_{\mathrm{reclin}} = 1 - \frac{\sum_{i=1}^{k} |B_{i,x}| \cdot |B_{i,y}|}{m \cdot n},$$

where m and n are the sizes of datasets X and Y, and k is the total number of blocks. The term $|B_{i,x}|$ is the number of unique records from dataset X in the i-th block, whilst $|B_{i,y}|$ is the number of unique records from dataset Y in the i-th block. The expression $\sum_{i=1}^{k} |B_{i,x}| \cdot |B_{i,y}|$ represents the number of comparisons after blocking.

Another way to assess blocking is to examine the confusion matrix at the *block* level, i.e., blocking results are compared with ground-truth *blocks* in a pairwise manner (e.g., one true positive pair occurs when both records from the comparison pair belong to the same predicted *block* and to the same ground-truth *block* in the evaluation `data.frame`). The values in this table are defined as follows:

- True positive (TP): record pairs correctly matched in the same block.
- False positive (FP): record pairs identified as matches that are not true matches in the same block.
- True negative (TN): record pairs correctly identified as non-matches (different blocks).
- False negative (FN): record pairs identified as non-matches that are true matches in the same block.

Metrics calculated based on this confusion matrix are presented in Table 1.

Table 1: Evaluation Metrics

Metric	Formula	Metric	Formula
Recall	$\frac{TP}{TP+FN}$	Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	$\frac{TP}{TP+FP}$	Specificity	$\frac{TN}{TN+FP}$
F1 Score	$2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}}$	False Positive Rate	$\frac{FP}{FP+TN}$
False Negative Rate	$\frac{FN}{FN+TP}$		

3 Case studies

3.1 An example of blocking for record linkage

Let us first load the required packages.

```
library("blocking")
library("data.table")
library("reclin2")
```

We demonstrate the use of the `blocking()` function for record linkage on the `foreigners` dataset included in the package. This fictional representation of the foreign population in Poland was generated based on publicly available information, preserving the distributions from administrative registers. It contains 110,000 rows with 100,000 entities (thus containing 10,000 duplicates). Each row represents one record, with the following columns: `fname` – first name, `sname` – second name, `surname` – surname, `date` – date of birth, `region` – region (county), `country` – country, and `true_id` – a person identifier.

Next, we load the data and examine the first six records.

```
data("foreigners")
head(foreigners)
```

```
#>     fname  sname    surname       date region country true_id
#>    <char> <char>     <char>     <char> <char>  <char>   <num>
#> 1:   emin            imanov 1998/02/05            031       0
#> 2: nurlan        suleymanli 2000/08/01            031       1
#> 3:   amio        maharrsmov 1939/03/08            031       2
#> 4:   amik        maharramof 1939/03/08            031       2
#> 5:   amil        maharramov 1993/03/08            031       2
#> 6:  gadir        jahangirov 1991/08/29            031       3
```

In the next step, we split the dataset into two separate `data.frames`: one containing the first appearance of each entity in the `foreigners` dataset, and the other containing subsequent appearances. We then add row identifiers (`x` and `y`).

```
foreigners_1 <- foreigners[!duplicated(foreigners$true_id), ]
foreigners_1[, x := 1:.N]
foreigners_2 <- foreigners[duplicated(foreigners$true_id), ]
foreigners_2[, y := 1:.N]
```

Now, in both datasets we remove separators from the date column and create a new character column that concatenates information from all columns (excluding `true_id`) in each row. Information stored in the `txt` column will be used for blocking records in the `blocking()` function.

```
foreigners_1[, txt := paste0(fname, sname, surname, gsub("/", "", date),
                             region, country)]
foreigners_2[, txt := paste0(fname, sname, surname, gsub("/", "", date),
                             region, country)]
head(foreigners_1[, .(true_id, txt)])
```

```
#>    true_id                          txt
#>      <num>                       <char>
#> 1:       0         eminimanov19980205031
#> 2:       1   nurlansuleymanli20000801031
#> 3:       2     amiomaharrsmov19390308031
#> 4:       3    gadirjahangirov19910829031
#> 5:       4 zaurbayramova1996100601261031
#> 6:       5       asifmammadov19970726031
```

The default algorithm is the Nearest Neighbour Descent Method (Dong et al., 2011) implemented in the **rnndescent** package. Note that a default parameter of the blocking() function is seed = 2023, which sets the random seed.

```
result_reclin <- blocking(x = foreigners_1$txt,
                          y = foreigners_2$txt)
```

Now, we can examine the results by printing the `result_reclin` object. In this example, we have created 6,470 blocks based on 1,232 columns (2-character shingles). Blocks are small, as we have 3,920 blocks of 2 elements, 1,599 blocks of 3 elements, . . . , 2 blocks of 7 elements.

```
result_reclin
```

```
#> ========================================================
#> Blocking based on the nnd method.
#> Number of blocks: 6470.
#> Number of columns used for blocking: 1232.
#> Reduction ratio: 0.9999.
#> ========================================================
#> Distribution of the size of the blocks:
#>    2    3    4    5    6    7
#> 3920 1599  928   19    2    2
```

To access the result, one should use `result_reclin$result`. The resulting data.table has four columns (as presented below):

- `x` – reference dataset (i.e., `foreigners_1`) – this may not contain all units of `foreigners_1`;
- `y` – query (each row of `foreigners_2`) – this will contain all units of `foreigners_2`;
- `block` – the block identifier;
- `dist` – distance between pairs.

```
head(result_reclin$result)
```

```
#>        x     y block      dist
#>    <int> <int> <num>     <num>
#> 1:     3     1     1 0.2216882
#> 2:     3     2     1 0.2122737
#> 3:    21     3     2 0.1172652
#> 4:    57     4     3 0.1863238
#> 5:    57     5     3 0.1379310
#> 6:    61     6     4 0.2307692
```

Let's examine the first block. Clearly, there are typos in the `fname` and `surname`. Nevertheless, all records refer to the same entity (as denoted by `true_id`).

```
rbind(foreigners_1[3, 1:7], foreigners_2[1:2, 1:7])
```

```
#>     fname  sname    surname       date region country true_id
#>    <char> <char>     <char>     <char> <char>  <char>   <num>
#> 1:   amio        maharrsmov 1939/03/08            031       2
#> 2:   amik        maharramof 1939/03/08            031       2
#> 3:   amil        maharramov 1993/03/08            031       2
```

Now we use the `true_id` column to evaluate our approach.

```
matches <- merge(x = foreigners_1[, .(x, true_id)],
                 y = foreigners_2[, .(y, true_id)],
                 by = "true_id")
matches[, block := rleid(x)]
head(matches)
```

```
#> Key: <true_id>
#>    true_id     x     y block
#>      <num> <int> <int> <int>
#> 1:       2     3     1     1
#> 2:       2     3     2     1
#> 3:      20    21     3     2
#> 4:      56    57     4     3
#> 5:      56    57     5     3
#> 6:      60    61     6     4
```

We have 10,000 matched pairs, which can be used in the true_blocks argument of the `blocking()` function to specify the true block assignments. We obtain quality metrics for the assessment of record linkage.

```
res_reclin <- blocking(x = foreigners_1$txt,
                       y = foreigners_2$txt,
                       true_blocks = matches[, .(x, y, block)])

res_reclin
```

```
#> ========================================================
#> Blocking based on the nnd method.
#> Number of blocks: 6470.
#> Number of columns used for blocking: 1232.
#> Reduction ratio: 0.9999.
#> ========================================================
#> Distribution of the size of the blocks:
#>    2    3    4    5    6    7
#> 3920 1599  928   19    2    2
#> ========================================================
#> Evaluation metrics (standard):
#>      recall   precision         fpr         fnr    accuracy specificity
#>     96.7532     78.6700      0.0038      3.2468     99.9957     99.9962
#>    f1_score
#>     86.7795
```

For example, our approach results in a 3.25% FNR. To improve this, we can increase the epsilon parameter of the NND method from 0.1 to 0.5. To do so, we configure the `control_ann` parameter in the `blocking()` function using the `controls_ann()` and `control_nnd()` functions.

```
res_reclin2 <- blocking(x = foreigners_1$txt,
                        y = foreigners_2$txt,
                        true_blocks = matches[, .(x, y, block)],
                  control_ann = controls_ann(nnd = control_nnd(epsilon = 0.5)))

res_reclin2
```

```
#> ========================================================
#> Blocking based on the nnd method.
#> Number of blocks: 6470.
#> Number of columns used for blocking: 1232.
#> Reduction ratio: 0.9999.
#> ========================================================
#> Distribution of the size of the blocks:
#>    2    3    4    5    6    7
#> 3920 1599  928   19    2    2
#> ========================================================
#> Evaluation metrics (standard):
#>      recall   precision         fpr         fnr    accuracy specificity
#>     96.7532     78.6700      0.0038      3.2468     99.9957     99.9962
#>    f1_score
#>     86.7795
```

That decreases the FNR to 3.25%.

Now, to use the result in the record linkage process, we add this information to both datasets and specify it in the appropriate argument of a given function. Below, we present an example using the `reclin2` package with a simple score.

```
foreigners_1[res_reclin2$result, on = "x", block:= i.block]
foreigners_2[res_reclin2$result, on = "y", block:= i.block]

pair_blocking(x = foreigners_1,
              y = foreigners_2, on = "block") |>
  compare_pairs(on = c("fname", "surname", "date"),
                default_comparator = cmp_jarowinkler()) |>
  score_simple("score", on = c("fname", "surname", "date")) |>
  head(n= 4)

#>   First data set:  100 000 records
#>   Second data set: 10 000 records
#>   Total number of pairs: 4 pairs
#>   Blocking on: 'block'
#>
#>       .x    .y     fname   surname      date    score
#>    <int> <int>     <num>     <num>     <num>    <num>
#> 1:     3     1 0.8333333 0.8666667 1.0000000 2.700000
#> 2:     3     2 0.8333333 0.9333333 0.9666667 2.733333
#> 3:    21     3 0.8333333 0.9166667 1.0000000 2.750000
#> 4:    57     4 0.9259259 0.9259259 0.9666667 2.818519
```

3.2 An example of blocking for deduplication

In this section, we demonstrate a deduplication application using the `blocking()` function on the `RLdata500` dataset from the **RecordLinkage** package. Note that the dataset is included in the `blocking` package. It contains artificial personal data, and fifty records have been duplicated with randomly generated errors. Each row represents one record, with the following columns: `fname_c1` – first name, `fname_c2` – second name, `lname_c1` – last name, `lname_c2` – last name (second component), `by`, `bm`, `bd` – year, month, and day of birth, `rec_id` – record ID, and `ent_id` – entity ID.

```
data("RLdata500")
head(RLdata500)

#>    fname_c1 fname_c2 lname_c1 lname_c2    by    bm    bd rec_id ent_id
#>      <char>   <char>   <char>   <char> <int> <int> <int>  <int>  <int>
#> 1:  CARSTEN             MEIER           1949     7    22      1     34
#> 2:     GERD             BAUER           1968     7    27      2     51
#> 3:   ROBERT          HARTMANN           1930     4    30      3    115
#> 4:   STEFAN             WOLFF           1957     9     2      4    189
#> 5:     RALF           KRUEGER           1966     1    13      5     72
#> 6:  JUERGEN             FRANKE           1929     7     4      6    142
```

For the purpose of this example, we create a new column (`id_count`) that indicates how many times a given unit occurs, and then add leading zeros to the `bm` and `bd` columns. Finally, we create a new string column that concatenates information from all columns (excluding `rec_id`, `ent_id`, and `id_count`), as presented below.

```
RLdata500[, id_count :=.N, ent_id]
RLdata500[, txt:=tolower(paste0(fname_c1,fname_c2,lname_c1,lname_c2,by,
                                sprintf("%02d", bm),sprintf("%02d", bd)))]
head(RLdata500[, .(rec_id, id_count, txt)])

#>    rec_id id_count                    txt
#>     <int>    <int>                 <char>
```

```
#> 1:      1        1   carstenmeier19490722
#> 2:      2        2      gerdbauer19680727
#> 3:      3        1 roberthartmann19300430
#> 4:      4        1    stefanwolff19570902
#> 5:      5        1    ralfkrueger19660113
#> 6:      6        1  juergenfranke19290704
```

As in the previous example, we use the `txt` column in the `blocking()` function. This time, we set `ann = "hnsw"` to use the Hierarchical Navigable Small World (HNSW; Malkov and Yashunin (2018)) algorithm from the **RcppHNSW** package.

```
res_dedup <- blocking(x = RLdata500$txt,
                      ann = "hnsw",
                      verbose = 1)
```

```
#> ===== creating tokens =====
#> ===== starting search (hnsw, x, y: 500, 500, t: 429) =====
#> ===== creating graph =====
```

The results are as follows. This time, the HNSW algorithm provided blocks varying from 2 to 17 units.

```
res_dedup
```

```
#> ============================================================
#> Blocking based on the hnsw method.
#> Number of blocks: 133.
#> Number of columns used for blocking: 429.
#> Reduction ratio: 0.9916.
#> ============================================================
#> Distribution of the size of the blocks:
#>  2  3  4  5  6  7  8  9 10 11 12 17
#> 46 35 23  8  6  6  2  3  1  1  1  1
```

Next, we create a long `data.table` with information on blocks and units from the original dataset. We add the block information to the final dataset. We can check in how many blocks the same entities (`ent_id`) are observed. In our example, all identical entities are in the same blocks.

```
df_block_melted <- melt(res_dedup$result, id.vars = c("block", "dist"))
df_block_melted_rec_block <- unique(df_block_melted[, .(rec_id=value, block)])
RLdata500[df_block_melted_rec_block, on = "rec_id", block_id := i.block]
RLdata500[, .(uniq_blocks = uniqueN(block_id)), .(ent_id)][, .N, uniq_blocks]
```

```
#>    uniq_blocks     N
#>          <int> <int>
#> 1:           1   450
```

Additionally, we visualise the result based on whether a block contains matches or not.

Finally, we compare the evaluation metrics across all ANN algorithms supported by the `blocking()` function, i.e., NND, HNSW, Annoy (from the **RcppAnnoy** package), Locality-Sensitive Hashing (LSH, from the **mlpack** package), and k-Nearest Neighbours (kNN – denoted as `"kd"`, from the **mlpack** package). We use the `rec_id` and `ent_id` columns from the `RLdata500` dataset to specify the true blocks and then calculate evaluation metrics for all algorithms.

We compare our package with the `klsh()` function from the **klsh** package, configured to create 10 blocks (denoted as `klsh_10`) and 100 blocks (denoted as `klsh_100`), respectively.

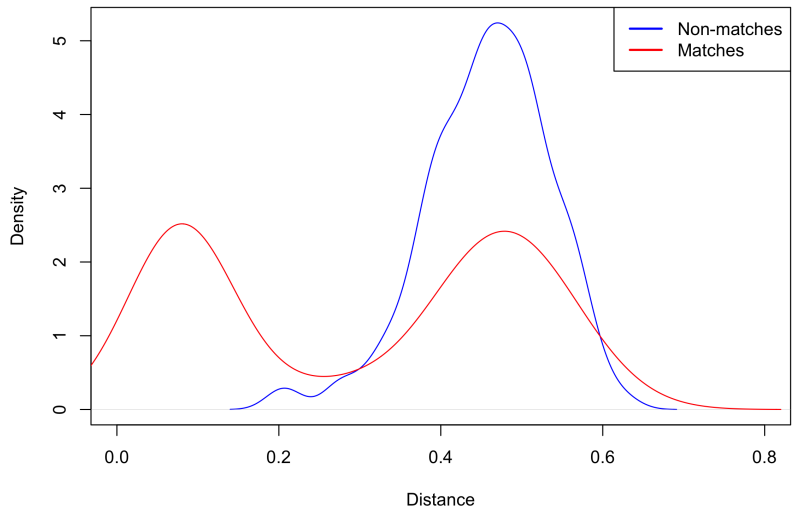

Figure 1: Distribution of distances between true matches and non-matches within blocks

In both settings, we use 20 random projections and 2-character shingles. The results are presented in Table 2.

```
set.seed(2025)
true_blocks <- RLdata500[, c("rec_id", "ent_id"), with = FALSE]
setnames(true_blocks, old = c("rec_id", "ent_id"), c("x", "block"))
eval_metrics <- list()
ann <- c("nnd", "hnsw", "annoy", "lsh", "kd")
for (algorithm in ann) {
  eval_metrics[[algorithm]] <- blocking(
    x = RLdata500$txt,
    ann = algorithm,
    true_blocks = true_blocks)$metrics
}

blocks_klsh_10 <- klsh::klsh(
  r.set = RLdata500[, c("fname_c1", "fname_c2", "lname_c1",
                        "lname_c2", "by", "bm", "bd")],
  p = 20,
  num.blocks = 10,
  k = 2)

klsh_10_metrics <- klsh::confusion.from.blocking(
  blocking = blocks_klsh_10,
  true_ids = RLdata500$ent_id)[-1]

klsh_10_metrics$f1_score <- with(klsh_10_metrics,
                                 2*precision*recall/(precision + recall))

eval_metrics$klsh_10 <- unlist(klsh_10_metrics)

blocks_klsh_100 <- klsh::klsh(
  r.set = RLdata500[, c("fname_c1", "fname_c2", "lname_c1",
                        "lname_c2", "by", "bm", "bd")],
  p = 20,
  num.blocks = 100,
```

Table 2: Comparison of various approximate nearest neighbour algorithms implemented in the **blocking** and the **klsh** package for creation of blocks for deduplication

	recall	precision	fpr	fnr	accuracy	specificity	f1_score
nnd	100	5.17	0.74	0	99.26	99.26	9.83
hnsw	100	4.76	0.80	0	99.20	99.20	9.08
annoy	100	4.80	0.79	0	99.21	99.21	9.17
lsh	98	1.04	3.74	2	96.26	96.26	2.06
kd	100	5.19	0.73	0	99.27	99.27	9.87
klsh_10	84	0.33	10.13	16	89.87	89.87	0.66
klsh_100	90	3.72	0.94	10	99.06	99.06	7.14

```
  k = 2)

klsh_100_metrics <- klsh::confusion.from.blocking(
  blocking = blocks_klsh_100,
  true_ids = RLdata500$ent_id)[-1]

klsh_100_metrics$f1_score <- with(klsh_100_metrics,
                                  2*precision*recall/(precision + recall))

eval_metrics$klsh_100 <- unlist(klsh_100_metrics)

round(do.call(rbind, eval_metrics) * 100, 2) |>
  kable(digits=2)
```

The results demonstrate a clear performance hierarchy among the ANN algorithms implemented in the `blocking` package, with traditional tree-based methods (NND, HNSW, Annoy, and kNN) achieving perfect recall (100%) whilst maintaining excellent precision and F1 scores around 5-10%. Notably, these methods exhibit minimal FPR (0.73-0.80%) and maintain high specificity (99.20-99.27%), indicating their effectiveness in creating tight, accurate blocks.

In contrast, the LSH-based methods show more variable performance: the `mlpack` LSH implementation achieves 98% recall but suffers from higher FPR (3.74%) and, importantly, FNR (2%), whilst the `klsh` package results reveal a trade-off between block granularity and performance – `klsh_10` with only 10 blocks shows poor recall (84%), high FPR (10.13%), and FNR (16%), whereas `klsh_100` with 100 blocks recovers much of the performance (90% recall, 0.94% FPR, but high FNR of 10%). This indicates that the `klsh`, particularly these implementations of the LSH approach, misses a large number of true matches.

These findings suggest that modern ANN algorithms like NND, HNSW, and Annoy provide superior blocking performance for entity resolution tasks, offering both computational efficiency and high-quality results that minimise both missed matches and false linkages.

4 Summary

In this paper, we have demonstrated the basic use cases of the **blocking** package. We believe that the software will be useful for researchers working in various fields where integration of multiple sources is an important aspect. This is certainly of interest in the field of official statistics, where register-based statistics rely on high-quality linkage of administrative datasets, or medical studies, where assessment of health statistics relies on correct linkage of medical history with treatment outcomes or mortality records.

Furthermore, for users interested in integration with the **reclin2** package, we refer to the documentation of the `pair_ann()` function and the vignette entitled `"Integration with existing packages"`, which provides case studies demonstrating how the **blocking** package can be included in existing record linkage/deduplication pipelines.

5 Acknowledgements

Work on this package is supported by the National Science Centre, OPUS 20 grant no. 2020/39/B/HS4/00941. We also thank participants of the uRos 2024 conference for valuable comments and discussion.

We have also developed a Python version of the package, `BlockingPy`, which is available through PyPI. It has a similar structure but offers more ANN algorithms (e.g., FAISS) and enables the use of embeddings. For more details, see: Strojny, T., & Beręsewicz, M. (2025). BlockingPy: Approximate nearest neighbours for blocking of records for entity resolution. arXiv preprint arXiv:2504.04266.

References

M. Beręsewicz. Estimation of the number of foreigners in poland [szacunek liczby cudzoziemców w polsce]. Presentation at: Resident, But Who? – Measuring the Unmeasurable [Mieszkaniec, czyli kto? – zmierzyć niemierzalne], March 2025. URL https://dsp.stat.gov.pl/dsp2025/slides/sesja_3_problemy_pomiaru/2__beresewiczm.pdf. March 24-25, 2025. [p1]

O. Binette and R. C. Steorts. (almost) all of entity resolution. *Science Advances*, 8(12):eabi8021, 2022. doi: https://doi.org/10.1126/sciadv.abi8021. URL https://www.science.org/doi/abs/10.1126/sciadv.abi8021. [p1]

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL https://igraph.org. [p2]

G. Csárdi, T. Nepusz, V. Traag, S. Horvát, F. Zanini, D. Noom, and K. Müller. *igraph: Network Analysis and Visualization in R*, 2025. URL https://CRAN.R-project.org/package=igraph. R package version 2.1.4. [p2]

R. R. Curtin, M. Edel, O. Shrit, S. Agrawal, S. Basak, J. J. Balamuta, R. Birmingham, K. Dutt, D. Eddelbuettel, R. Garg, S. Jaiswal, A. Kaushik, S. Kim, A. Mukherjee, N. G. Sai, N. Sharma, Y. S. Parihar, R. Swain, and C. Sanderson. mlpack 4: a fast, header-only c++ machine learning library. *Journal of Open Source Software*, 8(82), 2023. doi: https://doi.org/10.21105/joss.05026. [p2]

A. Dasylva and A. Goussanou. Estimating the false negatives due to blocking in record linkage. *Survey Methodology*, 47(2):299–312, 2021. URL https://www150.statcan.gc.ca/n1/pub/12-001-x/2021002/article/00002-eng.pdf. [p1]

A. Dasylva and A. Goussanou. On the consistent estimation of linkage errors without training data. *Japanese Journal of Statistics and Data Science*, 5(1):181–216, 2022. doi: https://doi.org/10.1007/s42081-022-00153-3. [p2]

W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 577–586, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306324. doi: https://doi.org/10.1145/1963405.1963487. URL https://doi.org/10.1145/1963405.1963487. [p5]

D. Eddelbuettel. *RcppAnnoy: 'Rcpp' Bindings for 'Annoy', a Library for Approximate Nearest Neighbors*, 2024. URL https://CRAN.R-project.org/package=RcppAnnoy. R package version 0.0.22. [p2]

T. Enamorado, B. Fifield, and K. Imai. Using a probabilistic model to assist merging of large-scale administrative records. *American Political Science Review*, 113(2):353–371, 2019. doi: https://doi.org/10.1017/S0003055418000783. [p2]

T. Enamorado, B. Fifield, and K. Imai. *fastLink: Fast Probabilistic Record Linkage with Missing Data*, 2023. URL https://CRAN.R-project.org/package=fastLink. R package version 0.6.1. [p2]

I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American statistical association*, 64(328):1183–1210, 1969. doi: https://doi.org/10.1080/01621459.1969.10501049. [p1]

J. P. Howard, II. *phonics: Phonetic Spelling Algorithms in R*, 2021. URL https://jameshoward.us/phonics-in-r/. R package version 1.3.10. [p2]

J. Johndrow, K. Lum, and D. Dunson. Theoretical limits of microclustering for record linkage. *Biometrika*, 105(2):431–446, 2018. doi: https://doi.org/10.1093/biomet/asy003. [p2]

T. Kalinowski and D. Falbel. *ragnar: Retrieval-Augmented Generation (RAG) Workflows*, 2025. URL https://CRAN.R-project.org/package=ragnar. R package version 0.2.0. [p2]

Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018. doi: https://doi.org/10.1109/tpami.2018.2889473. [p9]

J. Melville. *RcppHNSW: 'Rcpp' Bindings for 'hnswlib', a Library for Approximate Nearest Neighbors*, 2024a. URL https://CRAN.R-project.org/package=RcppHNSW. R package version 0.6.0. [p2]

J. Melville. *rnndescent: Nearest Neighbor Descent Method for Approximate Nearest Neighbors*, 2024b. URL https://CRAN.R-project.org/package=rnndescent. R package version 0.1.6. [p2]

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. URL https://arxiv.org/abs/1301.3781. [p1]

L. A. Mullen, K. Benoit, O. Keyes, D. Selivanov, and J. Arnold. Fast, consistent tokenization of natural language text. *Journal of Open Source Software*, 3:655, 2018. doi: https://doi.org/10.21105/joss.00655. URL https://doi.org/10.21105/joss.00655. [p2]

G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Comput. Surv.*, 53(2), Mar. 2020. ISSN 0360-0300. doi: https://doi.org/10.1145/3377455. URL https://doi.org/10.1145/3377455. [p1]

M. Sariyar and A. Borg. The RecordLinkage Package: Detecting Errors in Data. *The R Journal*, 2(2):61–67, 2010. doi: https://doi.org/10.32614/RJ-2010-017. URL https://doi.org/10.32614/RJ-2010-017. [p2]

M. Sariyar and A. Borg. *RecordLinkage: Record Linkage Functions for Linking and Deduplicating Data Sets*, 2025. URL https://CRAN.R-project.org/package=RecordLinkage. R package version 0.4-12.5. [p2]

D. Selivanov, M. Bickel, and Q. Wang. *text2vec: Modern Text Mining Framework for R*, 2023. URL https://CRAN.R-project.org/package=text2vec. R package version 0.6.4. [p2]

Y. Singh Parihar, R. Curtin, D. Eddelbuettel, and J. Balamuta. *mlpack: 'Rcpp' Integration for the 'mlpack' Library*, 2025. URL https://CRAN.R-project.org/package=mlpack. R package version 4.6.2. [p2]

R. Steorts. *klsh: Blocking for Record Linkage*, 2020. URL https://CRAN.R-project.org/package=klsh. R package version 0.1.0. [p2]

R. C. Steorts, S. L. Ventura, M. Sadinle, and S. E. Fienberg. A comparison of blocking methods for record linkage. In J. Domingo-Ferrer, editor, *Privacy in Statistical Databases*, pages 253–268, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11257-2. doi: https://doi.org/10.1007/978-3-319-11257-2_20. [p1]

D. J. van der Laan. reclin2: a toolkit for record linkage and deduplication. *R Journal*, 14(2), 2022. URL https://journal.r-project.org/articles/RJ-2022-038/. [p2]

J. van der Laan. *reclin2: Record Linkage Toolkit*, 2024. URL https://CRAN.R-project.org/package=reclin2. R package version 0.5.0. [p2]

M. van der Loo. The stringdist package for approximate string matching. *The R Journal*, 6: 111–122, 2014. URL https://CRAN.R-project.org/package=stringdist. [p2]

M. A. Wright. Mechanizing a large index. *The Computer Journal*, 3(2):76, 1960. doi: https://doi.org/10.1093/comjnl/3.2.76. [p1]

Maciej Beręsewicz
University of Economics and BusinessStatisical Office in Poznań
Department of Statistics
Centre for the Methodology of Population Studies
https://maciejberesewicz.com
ORCiD: 0000-0002-8281-4301
maciej.beresewicz@poznan.pl

Adam Struzik
Adam Mickiewicz UniversityStatisical Office in Poznań
Faculty of Mathematics and Computer Science
Centre for Urban Statistics
adastr5@st.amu.edu.pl