The VLDB Journal (2021) 30:537–557
https://doi.org/10.1007/s00778-021-00656-7

REGULAR PAPER

Efficient and effective ER with progressive blocking

Sainyam Galhotra[1] · Donatella Firmani[2] · Barna Saha[3] · Divesh Srivastava[4]

Received: 26 May 2020 / Revised: 13 November 2020 / Accepted: 6 February 2021 / Published online: 13 March 2021

Abstract

Blocking is a mechanism to improve the efficiency of entity resolution (ER) which aims to quickly prune out all non-matching record pairs. However, depending on the distributions of entity cluster sizes, existing techniques can be either (a) too aggressive, such that they help scale but can adversely affect the ER effectiveness, or (b) too permissive, potentially harming ER efficiency. In this paper, we propose a new methodology of *progressive blocking* (`pBlocking`) to enable both efficient and effective ER, which works seamlessly across different entity cluster size distributions. `pBlocking` is based on the insight that the effectiveness–efficiency trade-off is revealed only when the output of ER starts to be available. Hence, `pBlocking` leverages partial ER output in a feedback loop to refine the blocking result in a data-driven fashion. Specifically, we bootstrap `pBlocking` with traditional blocking methods and progressively improve the building and scoring of blocks until we get the desired trade-off, leveraging a limited amount of ER results as a guidance at every round. We formally prove that `pBlocking` converges efficiently ($O(n \log^2 n)$ time complexity, where n is the total number of records). Our experiments show that incorporating partial ER output in a feedback loop can improve the efficiency and effectiveness of blocking by 5× and 60%, respectively, improving the overall F-score of the entire ER process up to 60%.

Keywords Entity resolution · Blocking · Data integration

1 Introduction

Entity resolution (ER) is the problem of identifying which records in a data set refer to the same real-world entity [7]. ER technologies are key for solving complex tasks (e.g., building a knowledge graph) but comparing all the record pairs to decide which pairs match is often infeasible. For this reason, the first step of ER selects sub-quadratic number of record pairs to compare in the subsequent steps. To this end, a commonly used approach is *blocking* [26]. Blocking groups similar records into *blocks* and then selects pairs from the 'cleanest' blocks—i.e., those with fewer non-matching pairs—for further comparisons. The literature is rich with methods for building and processing blocks [26], but depending on the data set at hand, different techniques can either leave too many matching pairs outside, leading to incomplete ER results and low effectiveness, or include too many non-matching pairs, leading to low efficiency.

`pBlocking` We propose a new *progressive* blocking technique that overcomes the above limitations by short-circuiting the two operations—blocking and pair comparisons—that are traditionally solved sequentially. Our method starts with an aggressive blocking step, which is efficient but not very effective. Then, it computes a limited amount of ER results on a subset of pairs selected by the aggressive blocking and sends these partial (matching and non-matching) results from the ER phase back to the blocking phase, creating a 'loop,' to improve blocking effectiveness. In this way, blocking can progressively self-regulate and adapt to the properties of each dataset, with no configuration effort. We illustrate our blocking method, that we call `pBlocking`, in the following example.

✉ Sainyam Galhotra
sainyam@cs.umass.edu

Donatella Firmani
donatella.firmani@uniroma3.it

Barna Saha
barnas@berkeley.edu

Divesh Srivastava
divesh@att.com

[1] University of Massachusetts Amherst, Amherst, USA

[2] Roma Tre University, Rome, Italy

[3] UC Berkeley, Berkeley, USA

[4] AT&T Chief Data Office, Bedminster, NJ, USA

Table 1 Sample records (we omit schema information) referring to four distinct entities

r_1^{c6}: 'chevy corvette c6'	r_2^{c6}: 'chevy corvette c6 navigation'
r_3^{c6}: 'chevrolet corvette c6'	r_1^{z6}: 'corvette z6 navigation'
r_1^{ma}: 'chevy malibu navigation'	r_2^{ma}: 'chevrolet chevy malibu'
r_3^{ma}: 'chevrolet malibu'	r_1^{ci}: 'citroen c6 navigation'

r_i^e represents the i-th record referring to entity e. Records in the first two rows refer to a Chevrolet Corvette C6 ($c6$) and a Z6 ($z6$). Records in the last two rows to a Chevrolet Malibu (ma) and a Citröen C6 (ci) (same model name as Corvette C6 but different car)

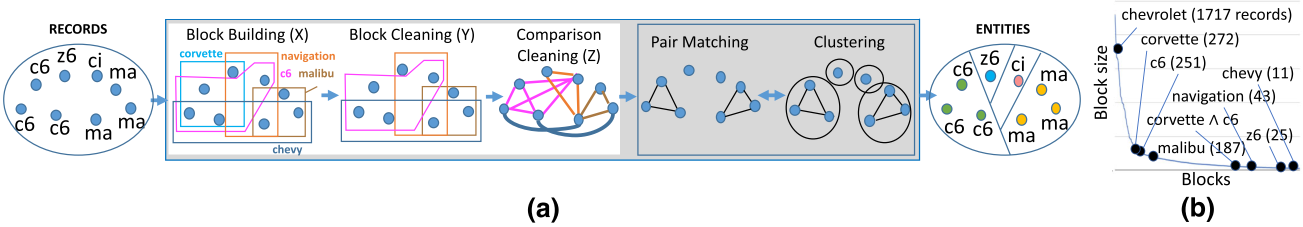


Fig. 1 **a** Illustration of a standard blocking pipeline. Block building, block cleaning and comparison cleaning sub-tasks are highlighted in white. The downstream ER algorithm is shown in gray. Description of each record is reported in Table 1. **b** Block size distribution (standard blocking) for the real `cars` dataset used in our experiments

Example 1 Consider the records in Table 1 from the `cars` dataset used in our experiments, and a standard schema-agnostic blocking strategy $\mathcal{S}$ such as [21]. As shown in Fig. 1a, we consider three blocking sub-tasks [26]. First, during *block building* $\mathcal{S}$ creates a separate block for each text token (we only show the blocks 'corvette,' 'navigation,' 'malibu,' 'c6' and 'chevy'). Then, during *block cleaning*, $\mathcal{S}$ uses a threshold to prune out all the blocks of large size. Depending on the threshold value (using the block sizes in the entire `cars` dataset, shown in Fig. 1b), we can have any of the following extreme behaviors. (Note that no intermediate setting of the threshold can yield a sparse set of candidates that is at the same time complete.)

– *Aggressive* blocking: $\mathcal{S}$ prunes every block except the smallest one ('chevy') and returns (r_1^{c6}, r_2^{c6}), (r_1^{c6}, r_1^{ma}), (r_2^{c6}, r_1^{ma}) and (r_1^{ma}, r_2^{ma}), missing r_3^{c6} and r_3^{ma}.
– *Permissive* blocking: $\mathcal{S}$ prunes only the largest block ('chevrolet') and returns many non-matching pairs.

Finally, during *comparison cleaning*, $\mathcal{S}$ can use another threshold to further prune out pairs sharing few blocks, e.g., by using *meta-blocking* [23]. As in block cleaning, different threshold values can yield aggressive or permissive behaviors. Note that matching pairs such as (r_2^{c6}, r_3^{c6}) share the same number of blocks ('corvette' and 'c6') as non-matching pairs such as (r_2^{c6}, r_1^{z6}) ('corvette' and 'navigation'). (Even worse, the block corresponding 'c6' is larger than 'navigation'.)

`pBlocking` can solve these problems in a few rounds: the first round does aggressive blocking, the second round does more effective blocking by making targeted updates accordingly to partial ER results, and so on. Examples of such updates to the blocking result are discussed below.

1. Creation of new blocks that help *inclusion* of (r_1^{c6}, r_3^{c6}), (r_2^{c6}, r_3^{c6}): `pBlocking` creates a *new* block 'corvette ∧ c6' with records present in both blocks 'corvette' and 'c6'. This block is much smaller than its two constituents and has only Corvette C6 cars.
2. Adaptive cleaning to help *inclusion* of (r_1^{ma}, r_3^{ma}), (r_2^{ma}, r_3^{ma}): `pBlocking` can discourage pruning of block 'malibu' that contains Chevrolet Malibu cars, even if it is a large block;
3. Adaptive cleaning to help *exclusion* of non-matching pairs: `pBlocking` can encourage pruning of block 'navigation' that contains no matching pairs, even if it is a small block.

After a few rounds of updates like the above, `pBlocking` returns all the matching pairs with very few non-matching pairs. Note that after the last round, the ER output can be computed on the resulting pairs as in the traditional setting. Updates of type (1) are performed via a new *block intersection* algorithm, while (2) and (3) are performed by a new *block scoring* method. By construction, when the blocking scores converge, the entire blocking result also converges.

Our contributions. The main contribution of this paper is a new blocking methodology with both high efficiency and effectiveness in a variety of application scenarios. Since `pBlocking` can in principle start off using any blocking strategy, it represents not only a new approach but also a way to ‘boost’ traditional ones. pBlocking works seamlessly across different entity cluster size distributions such as:

- *small entity clusters* where using block intersection, pBlocking can recover entities such as Corvette C6 consisting of few records sharing large and dirty blocks.
- *large entity clusters* where using block scoring, pBlocking can recover entities such as Chevrolet Malibu consisting of many records sharing large and clean blocks.

We prove theoretically and show empirically that, with a few rounds and a limited amount of partial ER results, our progressive blocking method can provide a significant boost in blocking effectiveness without penalizing efficiency. Specifically, we (i) demonstrate fast convergence with low space and time complexity ($O(n \log^2 n)$, where n is the number of records) of pBlocking; (ii) report experiments achieving up to 60% increase in recall when compared to state-of-the-art blocking [5], and up to 5x boost in efficiency. Finally, we observe that pBlocking can yield up to 70% increase on the F-score of the final ER result, thus confirming substantial benefits of our approach.

Outline. The rest of this paper is organized as follows. Sections 2 and 3 provide preliminary discussions and a high-level description of the pBlocking approach. Sections 4 and 5 explain our block intersection and block scoring methods, respectively. Section 6 provides theoretical analysis of pBlocking’s effectiveness, and Sect. 7 provides extensive experimental results and key takeaways. Section 8 discusses the related work, and we conclude in Sect. 9.

2 Blocking preliminaries

Table 2 summarizes the main symbols used throughout the paper. Let V be the input set of records, with $|V| = n$. Consider an (unknown) graph $\mathcal{C} = (V, E^+)$, where $(u, v) \in E^+$ means that u and v represent the same entity. $\mathcal{C}$ is transitively closed, that is, each of its connected components $C \subseteq V$ is a clique representing a distinct entity. We call each clique a *cluster* of V and refer to the partition induced by $\mathcal{C}$ as the ER *ground truth*.

Definition 1 (*Pair recall*) Given a set of matching record pairs $A' \subseteq V \times V$, pair recall is the fraction of pairs $(u, v) \in E^+$ that can be either (i) matched directly, because $(u, v) \in A'$, or (ii) indirectly inferred from other pairs $(u, w_0), (w_0, w_1), \ldots, (w_c, v) \in A'$ by connectivity.

A formal definition of the blocking task follows.

Problem 1 (*Blocking Task*) Given a set of records V, group records into possibly overlapping blocks $\mathcal{B} \equiv \{B_1, B_2, \ldots\}$, $B_i \subseteq V$ and compute a graph $P = (V, A')$, where $A' \subseteq A$, $A \equiv \{(u, v) : \exists B_i \in \mathcal{B} \text{ s.t. } u \in B_i \wedge v \in B_i\}$, such that A' is sparse $(|A'| << \binom{n}{2})$ and A' has high pair recall. We refer to P as the *blocking graph*.

Table 2 Notation table

V	Collection of records
$\mathcal{C}$	Collection of clusters
B	Block: A subset of records, $B \subseteq V$
$p_m(u, v)$	Similarity between u and v
$P = (V, A')$	Blocking graph, $A' \subset V \times V$
ϕ	Feedback frequency
$p(B)$	Probability score of a block B
$u(B)$	Uniformity score of block B
$H(B)$	Entropy of block B
$\mathcal{H}$	Block hierarchy
G_t	Random geometric graph
γ	Fraction of nodes used for scoring blocks
μ_g	Expected similarity of a matching edge
μ_r	Expected similarity of a non-matching edge

The blocking graph P is the final product of blocking and contains all the pairs that can be considered for pair matching. The efficiency and effectiveness of the blocking method is measured as pair recall (PR) of (the set of edges in) P and the number of edges in it for a certain PR, respectively. Blocking methods consist of three sub-tasks as defined by [26]: block building, block cleaning and comparison cleaning. In the following, we describe each of these steps and the corresponding methods in the literature.

Block building ($\mathcal{BB}$) takes as input V and returns a block collection $\mathcal{B}$, by assigning each record in V to possibly multiple blocks. The popular *standard blocking* [21] strategy creates a separate block B_t for each token t in the records and assigns to B_t all records that contain the token t. In order to tolerate spelling errors, *q-grams blocking* [11] considers character-level q-grams instead of entire tokens. Other strategies include *canopy clustering* [18] and *sorted neighborhood* [13]. Canopy clustering iteratively selects a random seed record r and creates a new block B_r (or a canopy) with all the records that have a high similarity with r according to a given similarity function (e.g., using a subset of features [18]). We can use different similarity functions to build different sets of canopies. Sorted neighborhood sorts all the records according to multiple sort orders (e.g., each according to a different attribute [13]), and then, it slides a window w of tokens over each ordering, every time creating a new block B_w. Blocks have the same number of distinct tokens, but the number of records in a block can vary significantly. Multiple block building strategies can be employed at the same time to generate the collection of blocks $\mathcal{B}$.

Block cleaning ($\mathcal{BC}$) takes as input the block collection $\mathcal{B}$ and returns a subset $\mathcal{B}' \subseteq \mathcal{B}$ by pruning blocks that may contain too many non-matching record pairs. Block cleaning is typically performed by assigning each block a $score : \mathcal{B} \rightarrow \mathbb{R}$ with a block scoring procedure and then pruning blocks with low score. Traditional scoring strategies include functions of block size such as TF-IDF [7,22].

Comparison cleaning ($\mathcal{CC}$) takes as input the set A of all intra-block record pairs in the block collection $\mathcal{B}'$ (which is a subset of the intra-block record pairs in $\mathcal{B}$) and returns a graph $P = (V, A')$, with $A' \subseteq A$, by pruning pairs that are likely to be non-matching. Comparison cleaning is typically performed by assigning each pair a $weight : A \rightarrow \mathbb{R}$ and then pruning pairs with low weight. Weighting strategies include *meta-blocking* [23] possibly with active learning [5,31]. In classic meta-blocking, $weight(u, v)$ corresponds to the number of blocks in which u and v co-occur, based on the assumption that that more blocks a record pair shares, the more likely it is to be matching.[1] The recent `BLOSS` strategy [5] employs active learning on top of the pairs generated by meta-blocking and learns a classifier using features extracted from the blocking graph for further pruning.

We denote with $\mathcal{B}(X, Y, Z)$ a blocking strategy that uses the methods X, Y, and Z, respectively, for block building, block cleaning and comparison cleaning. The strategy used in our `cars` example (Example 1) can be denoted as $\mathcal{B}$(*standard blocking, TF-IDF, meta-blocking*).

After blocking. Typical ER algorithms include *pair matching* and entity *clustering* operations. Such operations label as 'matching' the pairs referring to the same entity and 'non-matching' otherwise and typically require the use of a classifier [20] or a crowd [36]. Clustering consists of building a possibly noisy clustering $\mathcal{C}'$ according to labels and can be done with a variety of techniques, including robust variants of connected components [33] and random graphs [9]. This noisy clustering is the final product of ER.

3 Overview of pBlocking

Analogous to traditional blocking methods, `pBlocking` takes as input a collection V of records and returns a blocking graph P. A high-level view of the methods introduced in `pBlocking` for each of the main blocking sub-tasks of Sect. 2 is provided below. Such methods, unlike previous ones, can leverage a feedback of partial ER results.

Algorithm 1 Our blocking method `pBlocking`

```
Require: Records $V$, methods $X$, $Y$, $Z$ for each blocking step, method $W$ for pair
    matching and clustering. Default: $X$=standard blocking, $Y$= TF-IDF, $Z$=meta-
    blocking, $W$=Eager.
Ensure: Blocking graph $P$
$\mathcal{C}' \leftarrow \emptyset$
$B \leftarrow$ build the first level of block hierarchy with method $X$
$scores \leftarrow$ initialize block scores using method $Y$
$P \leftarrow$ block cleaning and comparison cleaning with method $Z$
$P_{new} \leftarrow \emptyset$
for round=2; round $\leq 1/\phi \wedge P \neq P_{new}$; round++ do
    while ER progress is less than $\phi$ do
        $\mathcal{C}' \leftarrow$ Execute an incremental step of method $W$ for pair matching and clus-
        tering on $P$
    $score \leftarrow$ update the block scores according to $\mathcal{C}'$   //Feedback
    $B \leftarrow$ update the block hierarchy based on $score$
    $P \leftarrow P_{new}$
    $P_{new} \leftarrow$ block cleaning and comparison cleaning with $Z$
return $P_{new}$
```

Block building in `pBlocking` constructs new blocks arranged in the form of a *hierarchy*. First level blocks are initialized with blocks generated by a traditional method (e.g., standard blocking, sorted neighborhood, canopy clustering or q-gram blocking). Subsequent levels contain intersections of the blocks in previous levels. `pBlocking` can use feedback from the partial ER output to build intersections such as 'corvette ∧ c6' that can lead to new, cleaner blocks, and avoid bad intersections such as 'corvette ∧ chevrolet' that would not improve the fraction of matching pairs in P (Chevrolet Corvette C6 and Z6 are different entities). We discuss block intersection in Sect. 4.

Block cleaning in `pBlocking` prunes dirty blocks based on feedback-based scores. First round scores are initialized with a traditional method (e.g., TF-IDF). Then, scores are refined based on feedback by combining two quantities: the fraction $p(B)$ of matching pairs in a block B, and the block uniformity $u(B)$, which captures the distribution of entities within the block ($u(B)$ is the inverse of *perplexity* [17]). Since the goal of blocking phase is to identify blocks that have a higher fraction of matching pairs and fewer entity clusters, we combine the above values as $score(B) = p(B) \cdot u(B)$. `pBlocking` can use feedback from the partial ER output to estimate $p(B)$ and $u(B)$, yielding high scores for clean blocks such as 'malibu' (high $p(B)$ and high $u(B)$) and low scores for dirtier blocks such as 'navigation' (low $p(B)$ and low $u(B)$), and 'c6' (low $u(B)$). We discuss block scoring in Sect. 5.

Finally, *comparison cleaning* in `pBlocking` is implemented with a traditional method such as meta-blocking.

Workflow. Algorithm 1 describes the `pBlocking` workflow and how the introduced blocking methods can be used. We denote with `pBlocking`(X, Y, Z) a progressive blocking strategy that uses the methods X, Y and Z, respectively, for building the first level of the block hierarchy, initializing the block scores, and performing comparison cleaning

[1] This assumption holds for block building methods such as standard blocking, q-grams blocking and sorted neighborhood with multiple orderings [13] and extends naturally to canopy clustering by using multiple similarity functions.

as described in Algorithm 1. In our cars example, we use pBlocking(*standard blocking, TF-IDF, meta-blocking*).

We first initialize the set of clusters $\mathcal{C}'$, the block hierarchy and the block scores (**lines 1–3**). The next step (**line 4**) consists of computing the first version of the blocking graph P according to the selected method for comparison cleaning (e.g., meta-blocking). The graph P is then progressively updated, round after round (**lines 6–12**). In order to activate the feedback mechanism, pBlocking needs to interact with an ER algorithm W for pair matching and clustering operations (**line 7–8**). Algorithm W is executed over P until it makes a *progress* of ϕ with $\phi \in [0, 1]$, that is, until $\phi \cdot n \log^2 n$ record pairs have been processed since the previous round.[2] At that point, the algorithm W is interrupted, $\mathcal{C}'$ is updated (**line 8**) and sent as feedback to all of pBlocking's components. Based on such feedback, we update the function $score(B) = p(B) \cdot u(B)$ (**line 9**) and construct new blocks in the form of a hierarchy (**line 10**). Blocks with high score are used to enumerate the most promising record pairs and generate the updated blocking graph P_{new} (**lines 11-12**). When either the maximum number of rounds $\frac{1}{\phi}$ has been reached (setting $\phi = 1$ is the same as switching off the feedback) or the blocking result converges ($P = P_{new}$), pBlocking terminates by returning P.

We present a formal analysis of the effectiveness of pBlocking in Sect. 6. We refer to Sect. 7 for experiments. Due to its robustness to different choices of the pair matching algorithm W, we do not include W in pBlocking's parameters (differently from X, Y, Z). Natural choices for W include *progressive* ER strategies that can process P in an online fashion and compute $\mathcal{C}'$ incrementally [20,34,35]. However, traditional algorithms, such as [7] can be used as well by adding *incremental ER* techniques [12,37] on top.

3.1 Computational complexity

For efficiency, it is crucial to ensure that the total time and space taken to compute P is close to linear in n. Since every round of pBlocking comes with its own time and space overhead, we first describe how to bound the complexity of every round and then discuss how to set the parameter ϕ in Algorithm 1 (and thus the maximum number of rounds) so as to bound the complexity of the entire workflow.

Round Complexity. pBlocking implements the following strategies to decrease overhead of each round.

Efficient block cleaning. We compute the block scores by sampling $\Theta(\log n)$ records from each of the top $O(n)$ high score blocks computed in the previous round.

Efficient comparison cleaning. For simplicity, we build P by enumerating at most $\Theta(n \log^2 n)$ intra-block pairs by processing blocks in non-increasing block score.

Based on the above discussion, we have Lemma 1.

Lemma 1 *A single round of* pBlocking*(X, Y, Z), such as* pBlocking*(*standard blocking, TF-IDF, meta-blocking*) has $O(n \log^2 n)$ space and time complexity.*

Proof We first show that the total feedback is limited to $O(n \log^2 n)$ space complexity, even though it considers all transitively inferred matching and non-matching edges, which can be $\Omega(n \log^2 n)$. For the matching pairs, we store all the records with an entity id such that any pair of records that have been resolved share the same id. This requires $O(n)$ space in the worst case and captures all the matching edges that have been identified in the ER output. For the non-matching pairs, we store a non-matching edge between their entity ids. Since the maximum number of pairs returned by pBlocking is limited to $O(n \log^2 n)$, the total number of pairs compared in each round and thus the number of non-matching edges stored is also $O(n \log^2 n)$. Then, we analyze the complexity of using feedback for the $\mathcal{BB}$ and $\mathcal{BC}$ tasks. Since the maximum number of blocks considered in any round for the scoring component is $O(n)$ and the scoring mechanism samples $O(\log^2 n)$ pairs from each block, the total number of edges enumerated for block scoring and building is $O(n \log^2 n)$. Since the maximum number of pairs for inclusion in the graph H is also $O(n \log^2 n)$, a single round of pBlocking outputs H in $O(n \log^2 n)$ total work. □

Workflow complexity. As discussed in Sect. 6, ϕ can be set to a small constant fraction. Thus, along with Lemma 1, this guarantees an $O(n \log^2 n)$ complexity for the entire workflow. Experimentally a smaller ϕ value yields higher final recall; thus, as a default we set $\phi = 0.01$, yielding a maximum of 100 rounds. Although such a ϕ value gets the best trade-off between effectiveness and efficiency in our experiments, we also observe that slight variations of its setting do not affect the performance much (Sect. 7), demonstrating the robustness of pBlocking.

4 Block building

One of the major challenges of block building ($\mathcal{BB}$) is that when generating candidate pairs that capture matches it can also generate a number of non-matching pairs. This phenomenon is highly prevalent in datasets with very few matching pairs. To overcome this challenge, our *block building by intersection* algorithm takes a collection of blocks $B_1, \ldots, B_m$ built by traditional methods for $\mathcal{BB}$ and creates smaller clean blocks out of large dirty ones, thus contributing to the recall of the blocking graph without adding extra

[2] For algorithms such as [35], progress can be defined as a fraction $\phi \cdot n$ of processed *records* since the previous round.

Algorithm 2 Block Layers Creation

```
Require: Set of records $V$, depth $d$
Ensure: Layer set $\{L_1, \ldots, L_d\}$
for $i = 1; i \leq d; i++$ do
    $L_i \leftarrow \phi$
processed $\leftarrow \phi$
for $v \in V$ do
    blockLst$\leftarrow$ getBlocks(v)
    for $i = 2; i$ <d; $i++$ do
        for $\mathcal{B} = \{B_j : B_j \in$ blockLst$\}, |\mathcal{B}| = i$ do
            $B' = \cap_{B_j \in \mathcal{B}} B_j$
            if $B' \notin$ processed then
                $L_i$.append($B'$)
                processed.append($B'$)
        blockLst$\leftarrow L_i$
```

Algorithm 3 Layer Cleaning

```
Require: Layer set $\{L_1, \ldots, L_d\}$
Ensure: Cleaned Layer set $\{L_1, \ldots, L_d\}$
for $i = 2; i < d; i++$ do
    for block $\in L_i$ do
        parentLst $\leftarrow$ getParents(block)
        if $\prod_{p \in parentLst} score(p) < score(block)$ or $\prod_{p \in parentLst} \frac{|L_{i-1}[p]|}{n} < \frac{|L_i[block]|}{n}$ then
            continue
        else
            $L_i$.remove(block)
```

non-matching pairs. An *intersection block hierarchy* $\mathcal{H}$ is constructed as follows. Let the first layer be $B_1, \ldots, B_m$. Then blocks in layer L consist of the intersection of L distinct blocks in the first layer.

Example 2 Consider our `cars` example in Sect. 1, and the blocks corresponding to tokens 'corvette' and 'c6,' namely $B_{\texttt{corvette}}$, and $B_{\texttt{c6}}$. A sample block in the second level of $\mathcal{H}$ is $B_{corvette,c6} = B_{corvette} \cap B_{c6}$. When we build the new block, we only include records containing the two tokens 'corvette' and 'c6' (possibly non-consecutively), thus obtaining a cleaner block than the original ones.

Refined blocks. We refer to the newly created block as a *refined* block, and to the intersecting blocks as *parent* blocks. Not all the refined blocks are useful. We need one of the following correlation based conditions to hold to decide if a refined block $B_{i,j}$ must be kept in $\mathcal{H}$.

- $score(B_{i,j}) > score(B_i) \cdot score(B_j)$, that is the score of the refined block is higher than the combined score of the parent blocks.
- The existence of a randomly chosen record r in blocks B_i and B_j is positively correlated, i.e., $Pr[r \in B_{i,j}] = |B_{i,j}|/n > Pr(r \in B_i) \cdot Pr(r \in B_j)$, which simplifies to $|B_{i,j}| > \frac{|B_i||B_j|}{n}$. For example, the number of common records in blocks corresponding to tokens 'c6' and 'corvette' is much higher than the common records in blocks corresponding to 'navigation' and 'c6'.

Suppose the maximum depth of the hierarchy is d which is a constant. The construction of refined blocks can take $O(n^d)$ time if the number of blocks considered in the first layer is $O(n)$. For efficiency, we iterate over the records (linear scan) and for each record r, we consider all pairs of blocks that contain r as candidates to generate blocks in the different levels of the hierarchy. The following lemma bounds the total number of refined blocks across the hierarchy.

Lemma 2 *The number of blocks present in $\mathcal{H}$ is $O(n)$ if each record r is present in a constant number of blocks.*

Proof Our algorithm considers each record $u \in V$ and generates intersection blocks by performing conjunction of blocks that contain the record u. Suppose the record u is present in γ_u blocks in the first layer. Then the maximum number of blocks present in $\mathcal{H}$ that contain u is $\sum_{i=1}^{d} \binom{\gamma_u}{i}$. Assuming γ_u is a constant, the maximum number of blocks in the hierarchy is $n \sum_{i=1}^{d} \binom{\gamma_u}{i} = O(n)$. □

Refinement algorithm. We are now ready to describe `pBlocking`'s intersection method for building the block hierarchy. Our method has two steps:

- (Alg. 2) The first step creates all possible blocks considering the intersection search space.
- (Alg. 3) The cleaning phase removes the blocks that do not satisfy the correlation criterion described above.

Algorithm 2 describes the creation step, which iterates over all the records in the corpus and creates all possible blocks per record. The list of all blocks to which a record belongs is constructed (denoted by blockLst), and the new blocks are added in different layers. The layer of the new block depends on the number of intersecting blocks that constitute the new block. Then, the cleaning step in Algorithm 3 iterates over the different layers and keeps only the blocks that satisfy the score or size requirements. For a block in layer q, `getParents()` identifies the two blocks which are in layer $(q-1)$ whose conjunction generates the block being considered. If these parents have been removed during the cleaning phase, then their parents are considered and the process is continued recursively until we end up at the ancestors present in the list of blocks.

Block Layers Creation (Alg. 2) constructs all blocks in the form of a hierarchy and Layer Cleaning (Alg. 3) deactivates the blocks that do not satisfy the correlation requirements. Since the result of Block Layers Creation does not change in different `pBlocking` iterations, decoupling the creation component from the cleaning component (which changes dynamically) allows for more efficient computation.

Time complexity. Assuming the depth of the hierarchy is a constant, Algorithms 2 and 3 operate in time linear in the number of records n. Block refinement takes 3 minutes for a data set with $1M$ records in our experiments.

5 Block cleaning

Let $A' \subset V \times V$ be the pairs selected by blocking phase at a given point (we recall that A' is the edge set of the blocking graph $P = (V, A')$) and each considered pair $(u, v) \in A'$ has a similarity value denoted by $p_m(u, v)$. A block $B \subseteq V$ refers to a subset of records. Using this notation, we discuss the different methods for scoring blocks and how the scores converge with feedback for effective ER performance.

Block scoring. Block scoring helps to distinguish informative blocks based on their ability to capture records from a single cluster. By selecting pairs within informative blocks, downstream ER operations can focus on records pairs that have high probability of being a match. The most common mechanism used in the literature is TF-IDF, and it assigns block scores inversely proportional to the block size prioritizing smaller blocks over larger ones. If the data set has small clusters, such a simple method can work well. However, if the data set has a skewed cluster size distribution, some large blocks are just uninformative (and are rightfully less preferred by TF-IDF), but others can represent a large cluster and thus should stand out in the scoring. Distinguishing these blocks before pair matching can be difficult, but `pBlocking` provides a way to leverage the feedback.

Specifically, the scoring algorithm of `pBlocking` prioritizes blocks having (a) high fraction of matching pairs measured as matching probability within a block and (b) fewer number of clusters (especially larger clusters) measured as uniformity (a function of entropy of the cluster distribution within a given block B). Lower entropy and hence lower diversity values indicate the representativeness of B toward a particular cluster as opposed to higher entropy values which refer to the presence of many fragmented clusters.

More formally, the matching probability score identifies the probability that a randomly chosen pair $(u, v) \mid u, v \in B$ refers to the same entity and is defined as follows.

Definition 2 (*Matching Probability score $p(B)$*) The value $p(B)$ is defined as the fraction of matching pairs within a block B.

The block uniformity, $u(B)$, captures perplexity of cluster distribution within B measured in terms of its entropy.

Definition 3 (*Cluster Entropy $H(B)$*) The cluster entropy of a block, $H(B)$, refers to the entropy of the cluster distribution when restricted to the records present in block B. Mathematically, $H(B) = -\sum_{C \in \mathcal{C}} p_C \log p_C$, where $p_C = |C \cap B|/|B|$ refers to the probability that a randomly chosen node from B belongs to cluster C.

Using $H(B)$, block uniformity score is defined as follows.

Definition 4 (*Block Uniformity $u(B)$*) The block uniformity $u(B) = e^{-H(B)}$ is the inverse of perplexity [17] of the cluster distribution within the block where perplexity refers to the exponential of cluster distribution entropy.

Example 3 Suppose that we know that a block B contains records of two clusters C_1 and C_2, and thus, we can compute the uniformity of B exactly. If the two clusters are perfectly balanced in B, i.e., $|C_1 \cap B| = 0.5 \cdot |B|$ and $|C_2 \cap B| = 0.5 \cdot |B|$, the entropy is $H(B) = -0.5 \log 0.5 - 0.5 \log 0.5 \approx 0.69$ and thus $u(B) = e^{-H(B)} = 0.5$. If there is some skew, e.g., $|C_1 \cap B| = 0.7 \cdot |B|$ and $|C_2 \cap B| = 0.3 \cdot |B|$, then the entropy is lower $H(B) = -0.7 \log 0.7 - 0.3 \log 0.3 \approx 0.61$ and the uniformity is higher $u(B) \approx 0.54$. In the extreme case where $C_1 \cap B = B$ and $C_2 \cap B = \emptyset$, $H(B) = 0$ and $u(B) = 1$.

Note that when resolving two duplicate-free datasets where all clusters are of size 2 (also known as Record Linkage) the entropy increases with block size, thus block uniformity yields comparable results to traditional TF-IDF.

Since the goal of block scoring is to identify blocks that have high matching probability and high uniformity, we multiply the two values to get a final estimate of the block score.

Definition 5 (*Block Score, $score(B)$*) The score of a block B, $score(B)$, is defined as the product of matching probability score and uniformity score of B. That is, $score(B) = p(B)u(B)$.

Next, we describe the algorithm to estimate these components of block score. The exact value of matching probability and block uniformity requires complete ER results. However, `pBlocking` estimates these scores initially with the similarity estimates of every pair of records and refines these scores with additional feedback from partial ER results.

Matching probability score. The matching probability score is estimated as the average matching similarity of pairs of records within the block, i.e.:

$$p(B) = \frac{\sum_{u,v \in B} p_m(u, v)}{\binom{|B|}{2}}$$

where $p_m(u, v)$ is estimated as follows:

- for pairs declared as matches, we set $p_m(u, v) = 1$;
- for pairs declared as non-matches, we set $p_m(u, v) = 0$;
- for unlabeled pairs, we use the p_m values computed by common similarity metrics (e.g., via jaccard similarity or the similarity-to-probability mapping as in [28]).

Block uniformity estimation. Estimating uniformity score requires the cluster size distribution in B, which is harder to infer from the prior similarity values. We next describe a mechanism to estimate entropy $H(B)$ needed to compute the uniformity score. We consider each record $u \in B$ and consider the cluster C_u that contains u. We are interested in computing $\frac{|C_u \cap B|}{|B|}$ in order to compute entropy $H(B)$. Instead, we compute the expected size of $|C_u \cap B|$ as $E_u = E[|C_u \cap B|] = \sum_{v \in B} p_m(u, v)$ based on p_m values of edges incident on u. We compute the expected cluster size for every record $u \in B$ and sort them in non-increasing order. Let L be the sorted list. Let the first record in the sorted list L, that is, the node with highest expected cluster size in B be u. On expectation u has E_u records in B that belong to C_u. All these records must have similar expected cluster sizes as well. We put u and the next $\lfloor E_u \rfloor$ records from L to a set S_U, assuming that they belong to the same cluster C_u. We recurse on $L \backslash S_U$ until a partition $\{S_U, S_V, \ldots\}$ of the block is generated. The size of each partition can be thought of as a rough estimate of the true cluster distribution in B and is used to calculate the entropy.

Example 4 Consider a block B, with $|B| = 10$. Let $[u_1, u_2 \ldots u_{10}]$ be the corresponding list L of records sorted in non-increasing E_{u_i} values. If $E_{u_1} = \sum_{i \in 2 \ldots 10} p_m(u_1, u_i) = 6.6$ we set $S_{U1} = \{u_1 \ldots u_{1+\lfloor E_{u_1} \rfloor}\} = \{u_1 \ldots u_7\}$ and then consider the next node in L which is u_8. If $E_{u_8} = \sum_{i \in 9,10} p_m(u_8, u_i) = 2$ we set $S_{U8} = \{u_8 \ldots u_{8+\lfloor E_{u_8} \rfloor}\} = \{u_8 \ldots u_{10}\}$ and then finish. As $|S_{U1}| = 0.7 \cdot |B|$ and $|S_{U8}| = 0.3 \cdot |B|$, we estimate $u(B) = e^{-0.7 \log 0.7 - 0.3 \log 0.3} \approx 0.54$.

The value returned by this mechanism is generally an underestimate of the true entropy $H(B)$, but in practice it can approach $H(B)$ quickly with increasing feedback data and turns out to be very efficient. Section 6.2 discusses this convergence rate in different application scenarios.

Efficient block cleaning. Traditional scoring strategies such as TF-IDF are based on block size computation and thus operate in linear time. Computing our $score(B)$ values requires instead to process intra-block pairs and thus yields potentially quadratic computation. Hence, we sample $\Theta(\log n)$ records from each block for its score computation. This strategy operates in $\Theta(\log^2 n)$ time and takes less than 1 minute for a data set with $1M$ records in our experiments. Our sampling strategy gives an approximation within a factor of $(1 + \epsilon)$ of the matching probability scores estimated using all the records within each block (Lemma 7).

6 Analysis of pBlocking

In this section, we present a theoretical analysis of the effectiveness of `pBlocking`. We first analyze the pair recall of blocking in the absence of feedback by considering a natural generative model for block creation. Next we analyze the effect of feedback on block scoring and final recall.

6.1 Pair recall without feedback

We start by giving the following basic lemma below.

Lemma 3 *The blocking graph $P = (V, A')$ contains a spanning tree for each clique C of $\mathcal{C} = (V, E^+)$ iff the pair recall is 1.*

Proof If A' contains a spanning tree for each clique C, then any pair $(u, v) \in A' \cap E^+$ contributes directly to the recall. All pairs of records (u, v) that refer to the same entity, $(u, v) \in E^+$ and are not present in A', $(u, v) \notin A'$ can be inferred from the edges in the spanning tree using transitivity, ensuring pair recall = 1. For the converse, let us assume that $\exists\, C \in \mathcal{C}$ such that A' does not contain any spanning tree over the matching edges. This implies that C is split into multiple components (say C_1, C_2) when restricted to $A' \cap E^+$ edges. In this case, the collection of matching edges joining these components, $\{(x, y), \forall x \in C_1, y \in C_2\}$ cannot be inferred as none of these edges are processed by the mentioned ER operations, yielding pair recall of P less than 1. □

Our probabilistic model for block creation is motivated by the standard blocking [21], sorted neighborhood [13] and canopy clustering [18] algorithms which aim to generate blocks that capture high similarity candidate pairs. This model of block generation is closely related to random geometric graphs [29] which were proposed by Gilbert in 1961 and have been used widely to analyze spatial graphs.

Definition 6 (*random geometric graphs*) Let S^t refer to the surface of a t-dimensional unit sphere, $S^t \equiv \{x \in \mathbb{R}^{t+1} \mid ||x||_2 = 1\}$. A random geometric graph $G_t(V, E)$ of n vertices V, has parameters $t \in \mathbb{Z}^+$ and a real number $r \in [0, 2]$. It assigns each vertex $i \in V$ to a point chosen independently and uniformly at random within S^t and any pair of vertices $i, j \in V$ are connected if the distance between their respective points is less than r.

Now, we define the probabilistic block generation model.

Definition 7 (*probabilistic block generation*) The block generation model places the records $u \in V$ independently and uniformly at random within S^t. Every record u constructs a ball of volume $(\alpha \log n/n)$ with u as the center, where α is a given parameter and all points within the ball are referred to as block B_u.

The set of points present within a ball B_u can be seen as high similarity points that would have been chosen as blocking candidates in the absence of feedback. Our probabilistic block generation model constructs n blocks, one for each node and every pair of records that co-occur in a block $B_u, u \in V$, has an edge in the blocking graph $P_g(V, E)$ (subscript g to emphasize generative model). Next we analyze pair recall of $P_g(V, E)$.

Notation. Let $d(u, v)$ refer to the distance between records u and v and r_ϵ refer to the radius of an ϵ-volume ball[3] in t dimensions. Under these assumptions, we first show that the expected number of edges in the blocking graph P_g is at least $\frac{\alpha(n-1)\log n}{2}$ and then that $P_g(V, E)$ has recall $<< 1$.

Lemma 4 *The blocking graph $P_g(V, E)$ contains at least $\alpha\frac{(n-1)\log n}{2}$ candidate pairs on expectation.*

Proof Each record $u \in V$ constructs a spherical ball of volume $\alpha \log n/n$, with u as the center and all points within the ball are added as neighbors of u in the blocking graph. Hence, the number of expected neighbors of u within the ball is $\alpha(n-1)\log n/n$. There are a total of n such blocks (one ball per record) and each of the candidate pairs (u, v) is counted twice (once for the block B_u and once for the block B_v). Hence there are a total of $\frac{\alpha(n-1)\log n}{2}$ such candidate pairs. Notice that this analysis ignores the candidate pairs (u, v) which are more than $r_{\alpha \log n/n}$ from each other but are connected in the blocking graph. This would happen if they are present together in another block centered at $w \in V\backslash\{u, v\}$, that is $\exists w \mid d(u, w) \leq r_{\alpha \log n/n}$ and $d(v, w) \leq r_{\alpha \log n/n}$. This shows that the total number of candidate pairs in the blocking graph is atleast $\frac{\alpha(n-1)\log n}{2}$. □

Additionally, $P_g(V, E)$ has the following property:

Lemma 5 *A blocking graph P_g is a subgraph of a random geometric graph G_t with $r = 2r_{\alpha \log n/n}$*

Proof Following the construction of blocking graph, if the distance between any pair of vertices $u, v \in V$ is less than or equal to $r_{c \log n/n}$, then $(u, v) \in E$. Similarly, any pair of nodes $u, v \in V$ such that $d(u, v) > 2r_{c \log n/n}$, then $(u, v) \notin E$. However, if $r_{c \log n/n} < d(u, v) \leq 2r_{c \log n/n}$, the pair $(u, v) \in H_g$ only if $\exists w \in V$ such that $d(u, w) \leq r_{c \log n/n}$ and $d(v, w) \leq r_{c \log n/n}$. This shows that the blocking graph H_g is a subgraph of a random geometric graph where a pair of vertices (u,v) is connected only if the distance $d(u, v) \leq 2r_{c \log n/n}$ is connected. □

This means that if G_t has suboptimal recall then P_g also has poor recall and hence, we analyze the recall of G_t with $r = 2r_{\alpha \log n/n}$. Lemma 3 shows that the blocking graph will achieve recall $= 1$ only if it contains a spanning tree of each cluster. Hence, we analyze the formation of spanning trees in $G'_t = G_t(V, E \cap E^+)$ that refers to G_t restricted to matching edges. We show the following result,

Lemma 6 *The graph G_t restricted to matching edges in the ground truth, E^+ splits a cluster C, where $|C| = o(n/\alpha)$ into multiple components.*

Proof Using the connectivity result from [29], a random geometric graph G_t of n nodes is disconnected if the expected degree of the nodes is $< \log n$. Additionally, it splits the graph G_t into many smaller clusters. Therefore, a cluster $C \in V$ is disconnected in $G'_t = G_t(V, E \cap E^+)$ if the degree of each vertex is $< \log |C|$.

The expected degree of a record $u \in C$, restricted to G'_t is $O(|C|(\frac{\alpha \log n}{n})) = o(\log n)$ if $|C| = o(n/\alpha)$. Hence, the expected degree of each node within a cluster C is $o(\log |C|)$, leading to formation of disconnected components within C. □

Theorem 1 *A blocking graph $P_g(V, E)$, generated according to the probabilistic block model has recall < 1 unless all clusters have size $\Theta(n)$ assuming α is a constant.*

Proof Lemma 6 shows that the cluster C of size $< n/\alpha$ is split into various disconnected components when restricted to matching edges. Hence, the blocking graph P_g does not form a spanning tree of C and will have recall less than 1 (Lemma 3). Since the cluster C is broken into many small clusters, the drop in recall is also significant. □

Remark. The analysis extends when considering less noisy data such as when only a constant fraction of records are placed randomly on the unit sphere, and the remaining records are grouped together according to the cluster identity they belong to. Our analysis exposes the lack of robustness of performing blocking without feedback.

6.2 Pair recall with feedback

In this section, we analyze the pair recall of blocking when employed with `pBlocking`. For this analysis, we consider the noisy edge similarity model $p_m(u, v)$ that builds on the edge noise model studied in prior work on ER [8].

Definition 8 (*Noisy edge model*) Noisy edge model defines the similarity of a pair of records with parameters $\theta \in (0, 1)$, $\beta = \Theta(\log n)$ and $\beta' = \Theta(\log n)$. A matching edge $(u, v) \in E^+$ has a similarity distributed uniformly at random within $[\theta, 1]$ with probability $1 - \frac{\beta}{n}$ and remaining edges are distributed uniformly within $[0, \theta)$. A non-matching edge has similar distribution on similarity values with β' instead of β.

When $\beta << \beta'$, the matching probability score of a block with higher fraction of matching edges is much higher than the one with fewer matching edges and `pBlocking` algorithm will consider blocks in the correct ordering even in the absence of feedback. However, it is most challenging when non-matching edges are generated with a distribution similar to matching edges, that is β and β' are close. We define a random variable $X(u, v)$ to refer to the edge similarity distributed according to the noisy edge model. Following this

[3] $\epsilon = O(r^t_\epsilon)$.

notion, let μ_g and μ_r denote the expected similarity of a matching and non-matching edge, respectively.

$$\mu_g = (1 - \beta/n)\,\frac{1+\theta}{2} + \frac{\beta}{n}\frac{\theta}{2}$$

and μ_r has the same value with β' instead of β.

We show that the feedback-based block score initialized with TF-IDF weights is able to achieve perfect recall with a feedback of $\Theta(n \log^2 n)$ pairs assuming that the ER phase makes no mistakes on the pairs that it processes, helping to ensure the correctness of partially inferred entities. Additionally, the feedback from the ER phase is distributed randomly across edges within a block. We also discuss the extension when feedback is biased toward pairs from large entity clusters and high similarity pairs. In those scenarios, `pBlocking`'s scoring mechanism converges quicker leveraging the larger feedback due to transitivity.

Effect of Sampling. First, we show that sampling $\Theta(\log n)$ records from a block gives approximation within a factor of $(1+\epsilon)$ of the matching probability score computed using all the records.

Lemma 7 *For a block B with $|B| > c \log n$, the matching probability score of B estimated by sampling $\Theta(\log n/\epsilon^2)$ records randomly is within $[(1-\epsilon), (1+\epsilon)]$ factor of $p(B)$ with a probability of $1 - o(1)$, where $p(B)$ is the score using all $|B|$ records.*

Proof Consider a block B with more than $c \log n$ records. Let $X(u, v)$ denote the edge similarity of a pair (u, v) according to the noisy edge model. The matching probability score of B on considering the complete block is $\frac{1}{\binom{|B|}{2}} \sum_{u,v\in B} X(u, v)$. The expected score of the block (μ_B) is

$$\frac{1}{\binom{|B|}{2}} E\left[\sum_{u,v\in B} X(u,v)\right] = \frac{1}{\binom{|B|}{2}} \sum_{\substack{u,v\in B,\\ (u,v)\in E^+}} E[X(u,v)] + \frac{1}{\binom{|B|}{2}} \sum_{\substack{u,v\in B,\\ (u,v)\in E^-}} E[X(u,v)] = (1-\alpha)\mu_g + \alpha\mu_r$$

where α is the fraction of non-matching pairs in the block B.

For a sample of $S = c \log n/\epsilon'^2$ records, the expected probability score (μ_S) is $(1-\alpha)\mu_g + \alpha\mu_r$, where $\epsilon' = \epsilon/(2+\epsilon)$

$$\frac{1}{\binom{c\log n}{2}} E[\sum_{u,v\in S} X(u,v)] = \frac{1}{\binom{c\log n}{2}} \sum_{\substack{u,v\in S,\\ (u,v)\in E^+}} E[X(u,v)] + \frac{1}{\binom{c\log n}{2}} \sum_{\substack{u,v\in S,\\ (u,v)\in E^-}} E[X(u,v)] = (1-\alpha)\mu_g + \alpha\mu_r$$

Using Hoeffding's inequality [14],

$$Pr\left[\frac{1}{\binom{c\log n}{2}} \sum_{u,v\in S} X(u,v) \le (1-\epsilon')\mu_S\right] \le e^{-2\epsilon'^2\mu_S^2\binom{c\log n}{2}} \le e^{-2\log n} = \frac{1}{n^2}$$

Using the same argument, we can show that $Pr\left[(1-\epsilon')\mu_S \le \frac{1}{\binom{c\log n}{2}} \sum_{u,v\in S} X(u,v) \le (1+\epsilon')\mu_S\right] \ge 1 - \frac{2}{n^2}$ This shows that the calculated probability score on the samples S is within a factor of $(1-\epsilon')$ and $(1+\epsilon')$ of the expected score with a probability of $1-o(1)$. The probability score of B on considering all records is also within a factor of $(1-\epsilon')$ and $(1+\epsilon')$ of the expected value μ_S. Therefore, the estimated score on sampling guarantees approximation within a factor of $\frac{1+\epsilon'}{1-\epsilon'} = 1 + 2\epsilon'/(1-\epsilon') = 1+\epsilon$ with a high probability. □

The above lemma can extend to block uniformity because p_m values are used analogously for expected cluster sizes. In Lemma 8, we show how to set the constant within the Θ notation based on level of noise in the p_m values.

To prove the convergence of `pBlocking`, we first estimate the lower and upper bound of matching probability scores of a block B in the presence of feedback and show that a feedback of $\Theta(\log^2 n)$ is enough to rank blocks with larger fraction of matching pairs higher than the blocks with fewer matching pairs. Our analysis first considers the blocks containing more than $\gamma \log n$ records (where γ is a large constant say 12) and we analyze the smaller blocks separately.

Convergence for large blocks. First, we evaluate the converged block scores with a feedback F and evaluate the condition that the block scores are in the correct order. For this analysis, we consider the fraction of matching edges for block score computation but similar lemmas extend for the uniformity score calculation.

Lemma 8 *For all blocks B, with more than $\gamma \log n$ records, the matching probability score of B, $p(B)$ after a feedback* *of $F = O(\log^2 n)$ randomly chosen pairs is at most $(1-\alpha)|F|/\binom{\gamma \log n}{2} + 1.5p'(1 - |F|/\binom{\gamma \log n}{2})$ with a probability of $1 - 1/n^3$, where α is the fraction of non-matching pairs in B, γ is a constant and $p' = \mu_g(1-\alpha) + \mu_r\alpha$.*

Proof For block scoring, `pBlocking` considers a sample of $S = \gamma \log n$ records (where γ is a large constant) and considers the sample ensuring that feedback $F \subseteq S \times S$ belongs to this sample. The total number of matching edges which have been identified with feedback over randomly chosen pairs is $(1-\alpha)|F|$. Let $X(u, v)$ be a random variable that refers to the similarity of the pair (u, v) and $\mu(u, v)$ to its expected value. For $S = \gamma \log n$, the expected similarity of non-feedback edges within C is

$$\sum_{\substack{u,v \in S,\\ (u,v) \notin F}} \mu(u, v) = \sum_{(u,v)\in E^+} E[X(u, v)] + \sum_{(u,v)\notin E^+} E[X(u, v)]$$
$$= \sum_{(u,v)\in E^+} \mu_g + \sum_{(u,v)\notin E^+} \mu_r$$
$$= \left(\binom{\gamma \log n}{2} - |F|\right)\left(\mu_g(1-\alpha) + \mu_r\alpha\right)$$

We use the Hoeffding inequality to bound the total similarity, $\sum X(u, v)$ of T edges which do not have feedback where $T = \left(\binom{\gamma \log n}{2} - |F|\right) = \gamma'\binom{\log n}{2}$, for some constant γ'.

$$\sum_{u,v\in S,(u,v)\notin F} X(u, v) \leq (1+\delta) \sum_{u,v\in S,(u,v)\notin F} \mu(u, v)$$

with a probability of $1 - e^{-2\delta^2\mu_T^2/|T|}$ which can be simplified as $1 - e^{-\delta^2\mu_T}$, since $\mu_r, \mu_g > 1/2$ Hence, the probability of success simplifies to $> 1 - 1/n^3$ after substituting $\delta = 0.5$. Therefore, the similarity score of the block B is atmost $\left(\frac{|F|}{\binom{\gamma \log n}{2}}(1-\alpha) + 1.5p'(1 - |F|/\binom{\gamma \log n}{2})\right)$ with a high probability. □

Similarly, we prove a lower bound on block score.

Lemma 9 *For all blocks B with $|B| \geq \gamma \log n$, the matching probability score after a feedback $F = O(\log^2 n)$ record pairs in B is at least $(1-\alpha)|F|/\binom{\gamma \log n}{2} + 0.5p'(1 - |F|/\binom{\gamma \log n}{2})$ with a probability of $1 - 1/n^3$, where $p' = \mu_g(1-\alpha) + \mu_r\alpha$ and γ is a constant.*

Now, we analyze different scenarios of edge noise to understand the trade-off between required feedback and noise.

Lemma 10 *For every pair of blocks, B_c, B_d with more than $\gamma \log n$ records, the matching probability score estimate of B_c with $1-\alpha$ fraction of matching edges is greater than the score of B_d with $1-\beta$ (with $\alpha < \beta$) fraction of matching edges with a probability of $1 - \frac{2}{n}$ if $\left((1-\alpha)\mu_g + \alpha\mu_r\right) > 3\left((1-\beta)\mu_g + \beta\mu_r\right)$ even in the absence of feedback.*

Proof Using Lemmas 8 and 9, we can evaluate the condition that $score(B_c) > score(B_d)$ with a probability of $1 - \frac{2}{n^3}$, in the absence of feedback. In order to guarantee this for all blocks, we perform a union bound over $\Theta(n^2)$ pairs of blocks, guaranteeing the success rate to $1 - o(1)$. □

The previous lemma shows a scenario where the noise is not high and the prior based estimation of matching probability scores give a correct ordering of blocks. Now, we consider the more challenging noisy scenario and show that $\Theta(\log^2 n)$ feedback per block is enough for correct ordering.

Lemma 11 *For every pairs of blocks, B_c, B_d with more than $\gamma \log n$ records, the matching probability score estimate of B_c with $1-\alpha$ fraction of matching edges is greater than the score of B_d with $1-\beta$ (where $\alpha < \beta$) fraction of matching edges with a probability of $1 - \frac{2}{n}$ whenever the ER phase provides overall feedback of $\Theta(n \log^2 n)$ randomly chosen edges.*

Proof Using Lemma 9, $score(B_c) \geq |F|/\binom{\gamma \log n}{2}(1-\alpha) + 0.5(\mu_g(1-\alpha) + \alpha\mu_r)(1 - |F|/\binom{\gamma \log n}{2})$ and using Lemma 8, $score(B_d) \leq |F|/\binom{\gamma \log n}{2}(1-\beta) + 1.5(\mu_g(1-\beta) + \beta\mu_r)(1 - |F|/\binom{\gamma \log n}{2})$ with a probability of $1 - \frac{2}{n^3}$. Hence, $score(B_c) > score(B_d)$ holds if $F = c\log^2 n$, where c is a large constant. With a union bound over $\binom{n}{2}$ pairs of blocks, the score of any block B_c (with higher fraction of matches) is higher than that of any block B_d (with lower fraction of matches) with a probability of $1 - \frac{2}{n}$. The total feedback to ensure $\Theta(\log^2 n)$ feedback on each block is $\Theta(n \log^2 n)$ as we consider $\Theta(n)$ blocks for scoring. □

Convergence for small blocks. The above analysis does not extend to blocks of size less than $\gamma \log n$. However, all these blocks are ranked higher than the large blocks by TF-IDF. Hence, when `pBlocking` is initialized, the initial set of candidates generated will consider all these blocks before any of the larger blocks. In the worst case, there can be δn such blocks, for some constant δ because our approach constructs a constant number of blocks per record (say δ). Thus, the maximum number of candidates considered from small blocks is $\delta n\binom{\gamma \log n}{2}$ and all these candidates are considered in the first iteration of `pBlocking`. Following the discussion on small and large blocks, we prove the main result of the convergence of `pBlocking`.

Theorem 2 *`pBlocking` pipeline achieves perfect recall with a feedback of $O(n \log^2 n)$ spread randomly across blocks.*

Proof For blocks with more than $\gamma \log n$ records, Lemmas 10 and 11 show that a block with higher fraction of matching pairs is ranked higher than a block with fewer matching pairs, if provided with a feedback of $\Theta(n \log^2 n)$. Blocks with less than $\gamma \log n$ records have not been considered above but in the worst case, these blocks generate $O(n \log^2 n)$ candidates as the maximum number of blocks considered is $\Theta(n)$. This ensures that a feedback of $\Theta(n \log^2 n)$ is sufficient to ensure the stated result. □

Discussion. Lemma 11 considers the convergence of block scores when the feedback is provided randomly over $\Theta(\log^2 n)$ edges within a block. If the feedback is biased toward $\Theta(\log^2 n)$ non-matching edges, the scores of noisier blocks will drop quicker and pBlocking will converge faster. Similarly, if the ER algorithm queries pairs with higher similarity (e.g., edge ordering [36]) or grows clusters by processing nodes (e.g., node ordering [35]), providing larger feedback due to transitivity, this will only facilitate the growth (reduction) in score of blocks with higher (lower) fraction of matching pairs leading to faster convergence.

Finally, for the presented analysis, we assumed that oracle answers are correct. Nonetheless, (i) for small amount of oracle errors ($\sim$ 5%), we can leverage methods such as [9,33] to correct them, and (ii) in more challenging applications with up to 20% erroneous answers, we show experimentally (see Sect. 7) that pBlocking keeps converging, only at a slightly slower rate and demonstrates robustness.

7 Experiments

In this section, we empirically demonstrate the ability of pBlocking to boost the efficiency and effectiveness of blocking and thus to improve the performance of ER. We also demonstrate the fast convergence of pBlocking thus confirming our theoretical analysis in Sect. 6, and the robustness of pBlocking in different scenarios, including errors in ER results. This section is structured as follows.

- *Section* 7.2. We compare the efficiency and effectiveness of pBlocking to prior work showing higher pair recall and faster running time in all the data sets.
- *Section* 7.3. We analyze pBlocking when used in conjunction with different ER methods showing higher *F-score* (up to 60%) irrespective of the method of choice.
- *Section* 7.4. We study the dynamic performance of pBlocking and show its ability to converge monotonically to high effectiveness without compromising on efficiency in different scenarios including errors in ER results.

7.1 Setup

Before showing results, we describe our experimental setup and the methods considered in our experiments.

Experimental setup. We implemented the algorithms in Java and machine learning tools in Python. The code runs on a server with 500GB RAM and 64 cores. We consider six real-world data sets (see Table 3) of various sizes and diverse cluster distributions. All the datasets are publicly available and come with their own manually curated ground truth. We use publicly available pre-trained deep learning models[4] to generate text descriptions of the image data (cars). febrl1 and febrl2 were constructed with uniform and zipfian distributions of cluster sizes. For more details about these parameters, please refer to [3]. For implementing the hierarchy we observed that we can trim at a depth of 10 without any significant drop in the performance. The implementation of blocking strategies is adapted from [26][5].

Blocking methods. We consider 10 strategies for the blocking sub-tasks described in Sect. 2 and combine such strategies into 20 different pipelines. We study such pipelines with and without our pBlocking approach on top.

- $\mathcal{BB}$) We consider five methods for Block Building ($\mathcal{BB}$) and follow the suggestions of [27] for their configuration. Standard blocking [21] (StBl) generates a new block for each text token in the dataset. Q-grams blocking [11] (QGBL) generates a new block for each 3-gram of characters. Sorted neighborhood [13] (SoNE) sorts the tokens for each attribute and generates a new block for every sliding window of size 3 over these sort orders. Dynamic Blocking [19] (DyBl) generates a new block for each token and constructs a hierarchy containing intersections of these large blocks. All blocks of size more than 20 are considered for hierarchy construction[6] Canopy clustering [18] (CaCl) generates a new block for each cluster of high similarity records (calculated as unweighted Jaccard similarity). We construct multiple instances of canopies (blocks), one for each attribute (i.e., based on the similarity of record pairs with respect to that attribute) and one based on all attributes together.
- $\mathcal{BC}$) We consider 2 traditional block scoring methods for Block Cleaning ($\mathcal{BC}$), dubbed TF-IDF [30] and uniform scoring (Unif). For comparison purposes, we process blocks in non-increasing score order until the

[4] https://cloud.google.com/vision, https://www.ibm.com/watson/services/visual-recognition/

[5] http://sourceforge.net/projects/erframework/

[6] This threshold on block size was shown to have best blocking quality in [19].

Table 3 Number of nodes n (i.e., records), number of clusters k (i.e., entities), size of the largest cluster $|C_1|$, the total number of matches in the data set $|E^+|$ and the reference to the paper where they appeared first

Dataset	n		k	$\lvert C_1 \rvert$	$\lvert E^+ \rvert$	Ref.	Description
songs	1M	1M	0.99M	2	146K	[6]	Self-join of songs with very few matches
citations	1.8M	2.5M	3.8M	2	558K	[6]	Bibliographic records from DBLP and CiteSeer
products	2554	22K	23.5K	2	1154	[10]	A collection of products from retail companies website
cora	1.9K		191	236	62.9K	[40]	Title, author, venue, and date of scientific papers
cars	16.5K		48	1799	5.9M	Partially from [16]	Descriptions of cars with make and model
camera	29.7K		26K	91	102K	[4]	A collection of cameras from over 25 retail companies
febrl1	100M		99.5M	2	500K	[3]	A collection of hospital patients data, including name, address and phone number, that we produced using the data set generator of the Febrl system
febrl2	100M		50M	100	2500M	[3]	

number of intra-block pairs equals to a parameter M and then prune the remaining blocks. We set default M to 500 million for febrl and 10 million for all other datasets.[7]

$\mathcal{CC}$) We consider 2 popular methods for Comparison Cleaning ($\mathcal{CC}$), dubbed meta-blocking [23] (MB) and BLOSS [5], and follow the suggestions of [23] for their configuration. Weights of record pairs are set to their Jaccard similarity weighted with the block scores from the $\mathcal{BC}$ sub-task. We consider the top 100 high weight pairs for each record and prune the remaining record pairs.

We recall that variants of our approach are denoted as pBlocking(,,) while traditional blocking pipelines without feedback are denoted as $\mathcal{B}$(,,) where the parameters correspond to techniques for $\mathcal{BB}$, $\mathcal{BC}$ and $\mathcal{CC}$ sub-tasks, respectively. Default methods are StBl for $\mathcal{BB}$, TF-IDF for $\mathcal{BC}$ and MB for $\mathcal{CC}$. Default ϕ for pBlocking is 0.01.

Pair matching and Clustering methods. We consider the following 3 strategies that leverage the notion of an *oracle* to answer pairwise queries of the form 'does u match with v?' (a) Edge [36] with default parameter setting. (b) Eager [9], the state-of-the-art technique to solve ER in the presence of erroneous oracle answers. (c) Node is the ER mechanism derived from [35] and was proposed as an improvement over Edge. The Eager algorithm handles noise for data sets with matching pairs much larger than n and performs similar to Edge for data sets that have fewer matching pairs [8], so we use it as default. Each of these techniques recalculate the prioritization of the updated set of blocked pairs in each feedback round. We implement the abstract oracle tool with a classifier using scikit learn[8] in Python. We consider two variants, random forests (default) and a neural network. The random forest classifier is trained with default settings of scikit learn. The neural network is implemented with a 3-layer convolutional neural network followed by two fully connected layers. We used word2vec word embeddings for each token in the records. In structured data sets, we extract similarity features for each attribute as in [6]. For cars, we use the text descriptions to calculate text-based features along with image-based features. Given the unstructured nature of text descriptions for some data sets, we extracted POS tags using Spacy[9]. All the considered classifiers are trained offline with less than $1,000$ labeled pairs, containing a similar amount of matching and non-matching pairs. These labeled record pairs are the ones provided by the respective source for citations, songs, products and camera (the

[7] We note that setting a score threshold rather than a limit on the number of pairs would not take into account different scores distributions fairly.

[8] https://scikit-learn.org/stable/

[9] https://spacy.io/

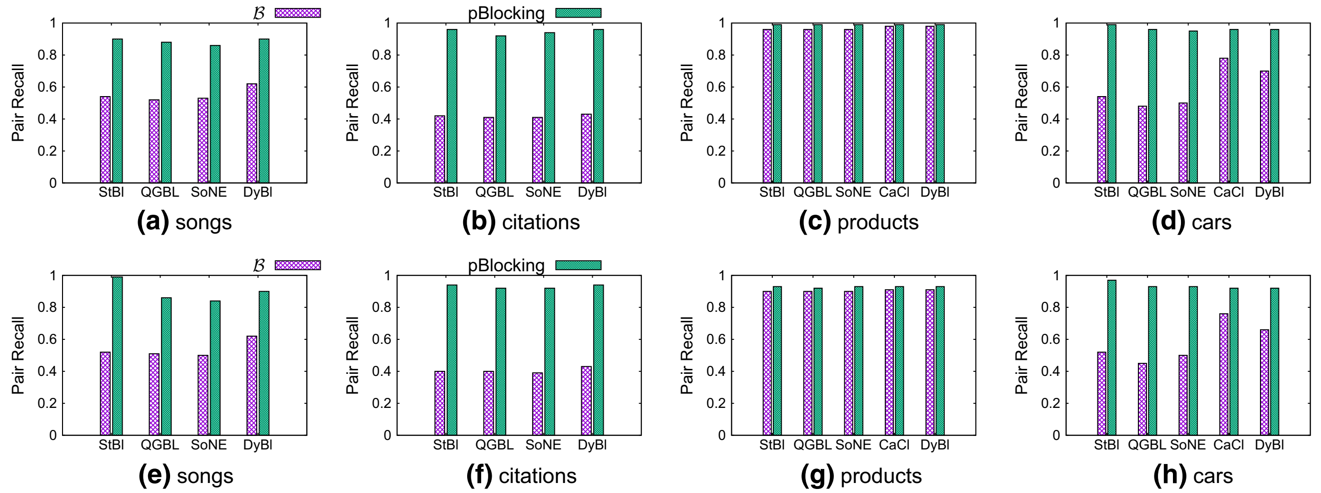


Fig. 2 Pair recall of $\mathcal{B}$ and pBlocking with TF-IDF for $\mathcal{BC}$ and varying $\mathcal{BB}$ and $\mathcal{CC}$. **a–d** use MB and (e-h) use BLOSS for $\mathcal{CC}$. CaCl did not finish within 24 hrs on songs and citations data set

papers mentioned in Table 3, column 'ref.'). For cars and cora, we perform active learning (following the guidelines of [6]) to identify a small set of labeled examples for training, which are excluded from the evaluation of blocking quality.

7.2 Benefits of progressive blocking

In this experiment, we evaluate the empirical benefit of pBlocking compared to previous blocking strategies.

Blocking effectiveness. Figures 2 and 4 compare the pair recall (PR) of pBlocking and of a traditional blocking pipeline $\mathcal{B}$ for different choices of the block building and comparison cleaning techniques. We use default block cleaning technique with TF-IDF and default M value. pBlocking achieves more than 0.90 recall for all data sets and with all the block building strategies, demonstrating its robustness to different cluster distributions and properties of the data. Conversely, most of the considered block building strategies (StBl, QGBL, SoNE and DyBl) have significantly lower recall even when used together with BLOSS for selecting the pairs wisely. QGBL and SoNE help to improve recall in data sets with spelling errors, but due to very few spelling mistakes in our data sets, StBl has slightly higher recall. DyBl creates blocks of moderate size that are expected to capture matching pairs. This technique performs better than StBl but the constructed smaller blocks contain a lot of non-matching pairs that affect pair recall.

In terms of the data sets, the no-feedback blocking approach $\mathcal{B}$ has varied behavior. products and camera yield the best performance due to the presence of relatively cleaner blocks that help to easily identify matching pairs even without feedback. songs and citations have higher noise in records, and cars has a skewed distribution of clusters, thereby making it harder for previous techniques. Even though cars and febrl2 have low noise, large blocks that contain majority of the records referring to same entity are partitioned by DyBl and ranked lower by TF-IDF weighting of blocks. Across all datasets and blocking strategies, the comparison between pBlocking and $\mathcal{B}$ is statistically significant ($p < 0.01$) using the student's paired t-test. For this analysis, we do not consider cora (the smallest data set) as it has less than 2M pairs, and hence, all techniques achieve perfect recall.

Figure 3 performs the same comparison with the pipelines initialized using Unif weights in place of TF-IDF. Since, all blocks are assigned equal weight, we consider the block cleaning threshold of 100 along with default value of M. pBlocking performs substantially better than $\mathcal{B}$ for different settings of block building techniques across various datasets. With comparison to TF-IDF weighting scheme, Unif performs slightly worse but the difference is not substantial. The no-feedback pipeline $\mathcal{B}$ has varied performance across different data sets with the best performance on products and cora while poorest performance on citations and songs. We observed similar behavior for cora, camera and febrl datasets.

This experiment demonstrates that pBlocking helps to improve the pair recall of all blocking techniques (for same set of parameters) and datasets. Note that increasing the block cleaning threshold M improves pair recall further but worsens the efficiency of the pipeline. As an example, [24] enumerates more than 10^{10} candidates for million scale datasets (where maximum possible candidates $\approx 10^{12}$), as opposed to 10M candidates in Fig. 2. As reported in [24], the pipeline with 10^{10} candidates requires more than 14.5 hours per dataset to achieve 0.82. For a fair comparison of blocking efficiency, we compare the pair recall within a time budget

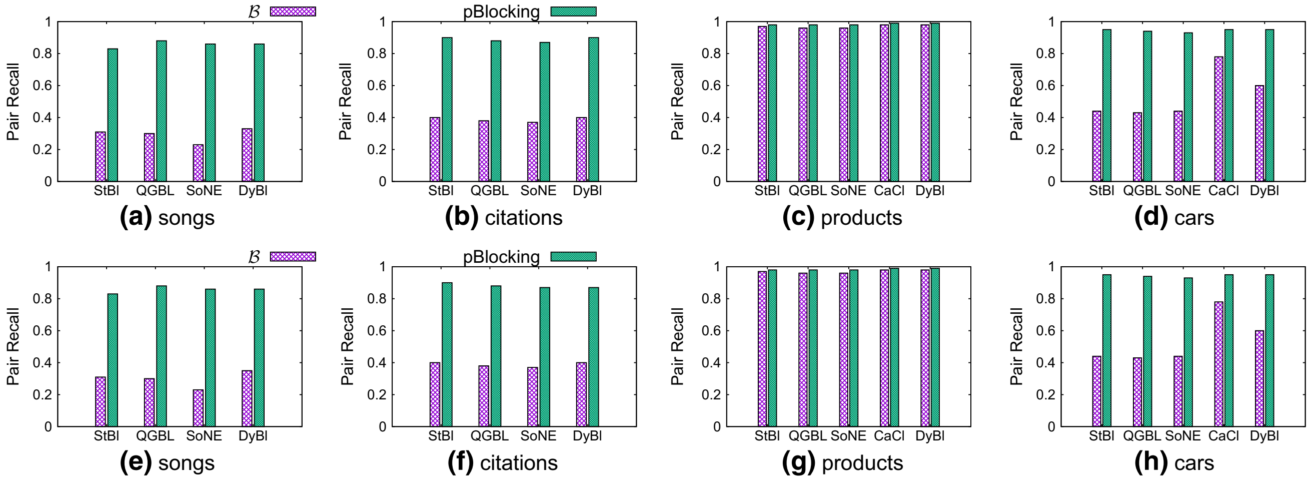


Fig. 3 Pair recall of $\mathcal{B}$ and `pBlocking` with `Unif` for $\mathcal{BC}$ and varying $\mathcal{BB}$ and $\mathcal{CC}$. **a–d** use `MB` and **e–h** use `BLOSS` for $\mathcal{CC}$. `CaCl` did not finish within 24 h on `songs` and `citations` data set

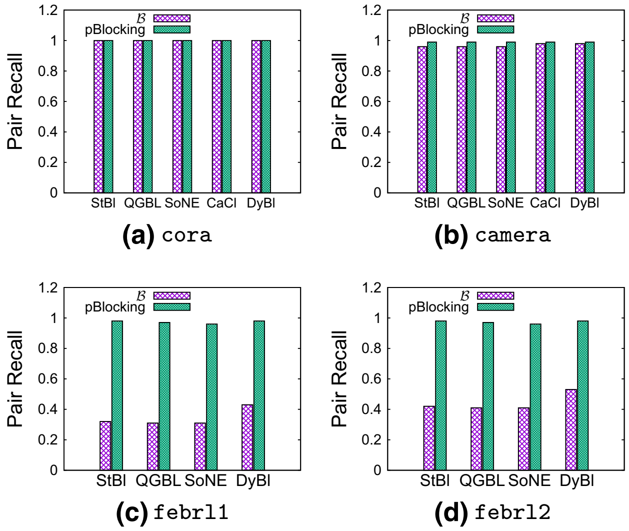


Fig. 4 Pair recall of $\mathcal{B}$ and `pBlocking` with varying $\mathcal{BB}$, `TF-IDF` for $\mathcal{BC}$ and `MB` for $\mathcal{CC}$. `CaCl` did not finish within 24 h on `febrl` datasets. We observed similar results with `BLOSS` for $\mathcal{CC}$

of 1hr and time taken to achieve 0.95 pair recall in 'Blocking efficiency' paragraph and Table 4.

Multiple blocking methods. Figure 5 demonstrates the effectiveness of considering feedback in pipelines where multiple block building procedures are used to initialize the pipeline. $\mathcal{B}$ has lower than 0.6 pair recall even when we consider different combinations of block building strategies. Using `DyBl` along with `QGBL` achieves the highest pair recall among the considered combinations, due to the ability of `DyBl` to construct smaller blocks that capture matching record pairs. However, `pBlocking` achieves more than 0.90 pair recall for all combinations of block building strategies.

Blocking efficiency. In this experiment, we consider two different settings to compare (i) the time required to achieve more than 0.95 pair recall (ii) the pair recall when the pipeline is allowed to run for a fixed amount of time (1 hour). We run each technique for various values of M and choose the best value that satisfies the required constraints. In the case of fixed budget of running time = 1hour, we run `pBlocking`'s feedback loop for the most iterations that allow the pipeline to process all records in the required time limit.

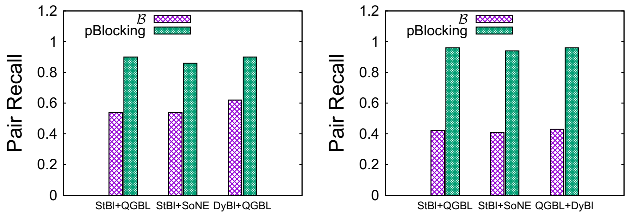
Table 4 compares the total time required to achieve 0.95 pair recall for each dataset ('Blocking' column denotes the time taken to perform blocking and 'ER' column denotes the time taken in pair matching and clustering phases of the pipeline). The time taken by the blocking component of the pipeline is higher for `pBlocking` as compared to $\mathcal{B}$ due to the extra effort spent in incorporating feedback, constructing new blocks and ranking based on their quality. However, `pBlocking`'s blocking component is highly effective and substantially reduces the time taken to process the candidates generated by the blocking phase to identify matches. Overall, `pBlocking` provides more than 3 times reduction in running time for most large scale datasets in this setting. In terms of total number of pairs enumerated, `pBlocking` considers around M=10 million to achieve 0.95 recall for `citations` as opposed to more than 200 million for $\mathcal{B}$. We observed similar results for other block building (`SoNE`, `QGBL`, `CaCl` and `DyBl`) and cleaning strategies with a difference that `DyBl` runs for `febrl` datasets in around 16 hrs.

The last two columns of Table 4 compare the pair recall of the generated candidates when the technique is allowed to run for 1 hour. `pBlocking` achieves better pair recall as compared to $\mathcal{B}$ across all datasets. The gain in recall is higher for larger datasets. The performance of `pBlocking` for `cars` is lower than that of `pBlocking` in Fig. 2d because the feed-

Table 4 Running time comparison of $\mathcal{B}$ and pBlocking with StBl and DyBl for $\mathcal{BB}$, TF-IDF for $\mathcal{BC}$ and MB for $\mathcal{CC}$

Dataset	StBl							
	0.95 Pair recall						Time budget: 1 h	
	pBlocking(StBl,TF-IDF,MB)			$\mathcal{B}$(StBl,TF-IDF,MB)			Pair recall	
	Blocking	ER	Total	Blocking	ER	Total	pBlocking	$\mathcal{B}$
songs	4.5 min	24.5 min	29 min	3 min	180 min	3 h 3 min	0.96	0.78
citations	12 min	43 min	55 min	Did not finish in 24 h			0.97	0.64
cars	3 h 20 min	50 min	4 h 10 min	25 min	11 h 30 min	11 h 55 min	0.78	0.54
febrl1	55 min	2 h 35 min	3 h 30 min	Did not finish in 24 h			0.64	0.21
febrl2	95 min	4 h 15 min	5 h 50 min	Did not finish in 24 h			0.34	0.15
products	35 s	5 min 55 s	6 min 30 s	27 s	5 min 46 s	6 min 13 s	0.99	0.98
camera	42 s	11 min 38 s	12 min 20 s	33 s	12 min 30 s	13 min 3 s	0.97	0.96
cora	30 s	4 min 50 s	5 min 20 s	27 s	4 min 48 s	5 min 15 s	1	1
	DyBl							
	pBlocking(DyBl,TF-IDF,MB)			$\mathcal{B}$(DyBl,TF-IDF,MB)			Pair recall	
	Blocking	ER	Total	Blocking	ER	Total	pBlocking	$\mathcal{B}$
songs	6.5 min	24.5 min	31 min	5 min	180 min	3 h 5 min	0.96	0.84
citations	15 min	43 min	58 min	15 min	10 h	10 h 15 min	0.97	0.67
cars	3 h 30 min	50 min	4 h 20 min	30 min	11 h 25 min	11 h 55 min	0.78	0.64
febrl1	58 min	2 h 35 min	3 h 33 min	1 h 7 min	15 h	16 h 7 min	0.64	0.21
febrl2	100 min	4 h 15 min	5 h 55 min	1 h 7 min	15 h 20 min	16 h 27 min	0.34	0.15
products	38 s	5 min 55 s	6 min 33 s	32 s	5 min 46 s	6 min 18 s	0.99	0.98
camera	45 s	11 min 38 s	12 min 23 s	37 s	12 min 30 s	13 min 7 s	0.97	0.96
cora	36 s	4 min 50 s	5 min 26 s	32 s	4 min 48 s	5 min 20 s	1	1

The 'blocking' column denotes the time taken to perform blocking and 'ER' denotes the time taken to identify matches over blocked pairs



(a) songs **(b)** citations

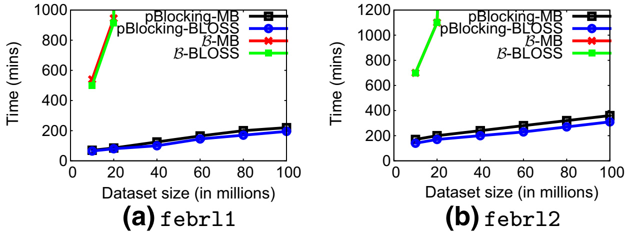
Fig. 5 Pair recall of $\mathcal{B}$ and pBlocking with combination of two block building strategies and TF-IDF for $\mathcal{BC}$ and MB for $\mathcal{CC}$

back loop does not converge completely in 1hr. The pipeline runs for 8 rounds of feedback in this duration. This is consistent with the performance of pBlocking in Fig. fig:carsa, where the feedback is turned off after 10 iterations. The performance of pBlocking and $\mathcal{B}$ is similar for small datasets of low noise like products, cora and camera as opposed to songs, citations and cars.

Scalability. Figure 6 compares the time taken by pBlocking on different subsamples of febrl dataset to reach 0.95 pair recall[10]. The time taken by pBlocking increases linearly with increase in dataset size and the pipeline identifies a majority of the matching records in less than 6 hrs. Since the number of matching pairs in the ground truth increases linearly with dataset size and low noise in records, the size of blocking graph and the time taken by the pair matching and clustering components scales linearly. The time taken by BLOSS is slightly lower than the time taken by MB because BLOSS processes the meta-blocking graph to further prune out non-matching record pairs. This optimization increases the time taken by blocking phase of the pipeline but significantly reduces the number of pairs compared by the pair matching phase, thereby improving the overall efficiency. On the other hand, $\mathcal{B}$ does not run for more than 20M records in less than 24 hrs. This experiment demonstrates scalability of pBlocking to achieve high recall over large scale datasets in a reasonable time.

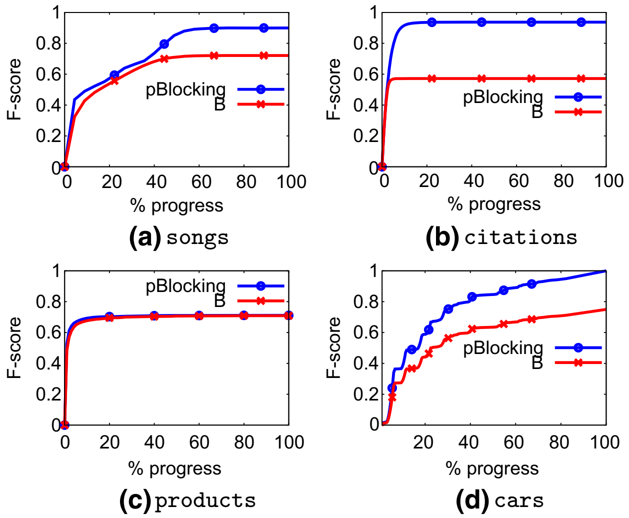
Progressive behavior. Figure 7 compares the F-score of different pipelines with respect to the progress of ER phase. F-score of the entities identified by pBlocking grows faster

[10] Each subsample was generated by using Febrl dataset generator with smaller value of n, the number of records.



(a) febrl1 **(b)** febrl2

Fig. 6 Time taken by pBlocking and $\mathcal{B}$ with StBl for $\mathcal{BB}$ and TF-IDF for $\mathcal{BC}$ for varying dataset size



(a) songs **(b)** citations **(c)** products **(d)** cars

Fig. 7 Comparison of F-score of $\mathcal{B}$(DyBl,TF-IDF,MB) and pBlocking(DyBl,TF-IDF,MB) with respect to ER progress

Table 5 (a) Pair recall of pBlocking on varying ER strategies. (b) Comparison of the final F-score of the Eager method

(a)

Dataset	$\mathcal{B}$	pBlocking		
		Edge	Node	Eager
songs	0.53	0.9	0.9	0.9
citations	0.42	0.90	0.87	0.95
cars	0.54	0.98	0.99	0.98
febrl1	0.32	0.98	0.98	0.98
febrl2	0.41	0.97	0.99	0.98
products	0.95	0.98	0.98	0.98
camera	0.92	0.97	0.97	0.97
cora	1	1	1	1

(b)

Dataset	$\mathcal{B}$	pBlocking
songs	0.65	**0.92**
citations	0.56	**0.92**
cars	0.64	**0.94**
febrl1	0.48	**0.98**
febrl2	0.58	**0.98**
products	0.71	**0.72**
camera	0.92	**0.95**
cora	0.99	0.99

The blocking graph is computed with pBlocking(StBl, TF-IDF, MB) and $\mathcal{B}$(StBl, TF-IDF, MB) (both with default settings)

Bold values indicate better quality (higher F-score)

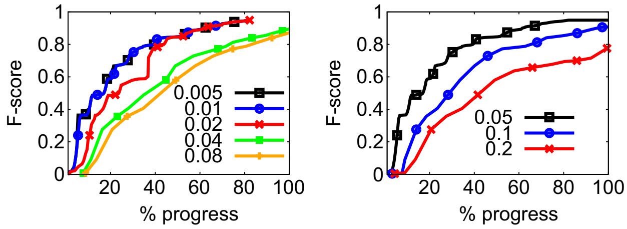
than $\mathcal{B}$, demonstrating its effectiveness to maintain better progressive behavior. pBlocking achieves more than 0.9 F-score across all datasets but $\mathcal{B}$ converges at a lower F-score due to the loss in pair recall of the blocking phase. In terms of datasets, pBlocking and $\mathcal{B}$ achieve similar progressive F-score throughout the ER progress for products dataset. products has around 0.72 final F-score due to low precision of pair matching and clustering phase. We observed similar behavior for other blocking pipelines.

7.3 Robustness of progressive blocking

In this section, we evaluate the performance of pBlocking with varying strategies for pair matching and clustering in Algorithm 1 (referred to as W in the pseudo-code). For this analysis, we use the default setting for M as in Fig. 2.

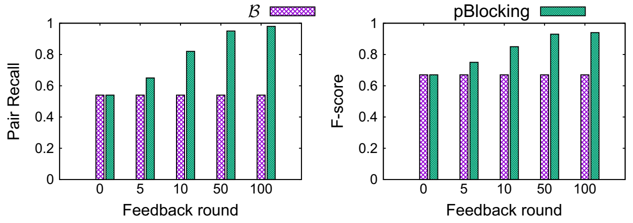
Varying ER methods. We recall that pBlocking can be used in conjunction with a variety of techniques for pair matching and clustering. Table 5a compares the pair recall of the blocking graph, when using the different progressive ER methods mentioned in Sect. 7.1. The final pair recall of pBlocking is more than 0.90 in all data sets and matching algorithms except citations for node ER and more than 0.85 in all cases. This observation confirms our theoretical analysis in Sect. 6.2, demonstrating that the feedback loop can improve the blocking, irrespective of the ER algorithm under consideration (which is a desirable property for a blocking algorithm). The above comparison of ER performance considers the algorithms with a default choice of random forest classifier as the oracle. We observed that the feedback from the ER phase when using a neural network classifier contains slightly more errors but the blocking phase with pBlocking shows similar recall. We provide more discussion on ER errors in Sect. 7.4.

Benefit on the final ER result. Table 5b compares the F-score of the final ER results when blocking is performed with and without pBlocking. In this experiment, we use the state-of-the-art algorithm, Eager as the pair matching algorithm with default parameter values. Final F-score achieved with feedback is more than 0.9 for all data sets except products. For songs, citations and cars, the F-score of pBlocking is 1.5 times more than that of traditional blocking pipeline without feedback, thus demonstrating the effects of better effectiveness and efficiency of blocking.



(a) Feedback Frequency **(b)** Oracle error

Fig. 8 Progressive behavior of `pBlocking` with varying feedback frequency and errors in the feedback (`cars`)



(a) Pair Recall comparison **(b)** F-score comparison

Fig. 9 Effect of feedback loop in `cars` dataset

7.4 Progressive behavior

This section studies the performance of `pBlocking` dynamically, in terms of (i) effect of feedback frequency ϕ, (ii) effect of error on convergence, and (iii) convergence of the blocking result in the maximum number of rounds.

Feedback frequency. The ϕ parameter represents the fraction of newly processed record pairs after which feedback is sent from the partial ER results back to the blocking phase. Therefore, the parameter ϕ can control the maximum number of rounds of `pBlocking` and how often the blocking graph is updated. In order to describe the effect of varying ϕ, Fig. 8a shows the F-score of ER results as a function of the percentage of rounds completed, that we refer to as the *blocking progress*. In the figure, different curves correspond to different feedback frequencies, including the default one (in blue). This plot shows that by updating the blocking graph more frequently (and thus increasing the number of rounds), the F-score increases faster when ϕ is reduced from 0.08 to 0.01. The plot also shows that the F-score corresponding to smaller values of ϕ (up to 0.01) is consistently higher or equal as compared to the F-score corresponding to larger values of ϕ. Given that the running time of the pipeline increases with more frequent updates (smaller values of ϕ), there appears to be limited value in decreasing ϕ below 0.01, thus justifying our choice for its default setting.

Effect of ER errors. As in the previous experiment, Fig. 8b shows the effect of synthetic error in the ER results by varying the fraction of erroneous oracle answers. To this end, we corrupted the oracle answers randomly so as to get the desired amount of noise. We note that even when 1 out of 5 answers are wrong, the final F-score is almost 0.8, growing monotonically from the beginning to the end at the cost of a few extra pairs compared. `pBlocking` converges slower with higher error but the error does not accumulate, and it performs much better than any other baseline. Additionally, we observed that even with 20% error, the pair recall of `pBlocking` is as high as 0.98 even though the F-score is close to 0.8 due to mistakes made by pair matching and clustering phase. This confirms that `pBlocking` is robust to errors in ER results and maintains high effectiveness to produce ER results with high F-score.

Score Convergence. Figure 9a compares the pair recall (PR) of the blocking phase of `pBlocking(StBl, TF-IDF,MB)` after every round of feedback with the recall of $\mathcal{B}$(`StBl`,`TF-IDF`,`MB`). Both $\mathcal{B}$ and `pBlocking` start with PR value close to 0.52, and `pBlocking` consistently improves with more feedback achieving PR close to 0.9 in less than 18 rounds. This shows the convergence of `pBlocking`'s score assignment strategy to achieve high PR values even with minimal feedback. Figure 9b compares the final F-score achieved by our method if the feedback loop is stopped after a few rounds. It shows that `pBlocking` achieves more than 0.8 F-score even when stopped after 10 rounds of feedback. This experiment validates that the convergence of block scoring leads to the convergence of the entire ER workflow.

7.5 Key takeaways

The empirical analysis in the previous sections has demonstrated `pBlocking`'s benefit on final F-score and its ability to boost effectiveness of blocking techniques across all data sets without compromising on efficiency. The key takeaways from our analysis are summarized below.

- `pBlocking` improves pair recall irrespective of the technique used for block building, block cleaning or comparison cleaning (Fig. 2), thus demonstrating its flexibility.
- Feedback-based scoring helps in particular to boost blocking efficiency and effectiveness for noisy datasets with many matching pairs (i.e., containing large clusters) such as `cars`, by enabling accurate selection of cleanest blocks.
- The block intersection algorithm helps in particular with data sets having fewer matching pairs (i.e., with mainly small clusters) such as `citations` and `songs`, by providing a way to build small focused blocks with high fraction of matching pairs. Block intersection can also help in data sets like `products` and `camera`, but the

benefit is not as high as that in songs, because many records in such data sets have unique identifiers (e.g., product model IDs) and thus initial blocks are reasonably clean.

8 Related work

Blocking has been used to scale entity resolution (ER) for a very long time. However, all techniques in the literature have considered blocking as a preprocessing step and suffered from the trade-off between effectiveness and efficiency/scalability. We divide the related work into two parts: advanced blocking methods which we improve upon and progressive ER methods which can be used to generate a limited amount of matching/non-matching pairs to send as a feedback to our blocking computation.

Advanced blocking methods. There are many blocking methods in the literature with different internal functionalities and solving different blocking sub-tasks. In this paper, we considered four representative block building strategies, namely standard blocking [21], canopy clustering [18], sorted neighborhood [13] and q-grams blocking [11]. It is well known that such techniques can yield a fairly dense blocking graph when used alone. We refer the reader to [26] for an extensive survey of various blocking techniques and their shortcomings. Such block building strategies can be used as the method X in our Algorithm 1.

One of the prior blocking techniques, dynamic blocking [19] considers conjunctions of large blocks to construct a hierarchy of smaller co-occurring sub-blocks. This approach assumes a priori knowledge of the attributes that are used to whittle down oversized blocks to an acceptable size and was primarily designed for datasets with small clusters (e.g., of size 2), where smaller blocks are correlated with matching pairs. On the other hand, pBlocking uses the block scores as a guidance to construct the hierarchy and rank the blocks. Following the score-based hierarchy construction procedure, pBlocking does not partition large blocks that contain a lot of matching pairs and partitions all blocks that contain fewer matching pairs irrespective of their size.

Recent works have proposed advanced methods that can be used in combination with the mentioned block building techniques by focusing on the comparison cleaning sub-task (thus improving on efficiency). The first technique in this space is *meta-blocking* [23]. Meta-blocking aims to extract the most similar pairs of records by leveraging block-to-record relationships and can be very efficient in reducing the number of unnecessary pairs produced by traditional blocking techniques, but it is not always easy to configure. To this end, follow-up works such Blast [31] use 'loose' schema information to distinguish promising pairs, while [2] and SNB [25] rely on a sample of labeled pairs for learning accurate blocking functions and classification models, respectively. Finally, the most recent strategy BLOSS [5] uses *active learning* to select such a sample and configure the meta-blocking. The goal of traditional meta-blocking [23] and its follow-up techniques like BLOSS [5] prune out low-similarity candidates from the blocking graph generated using various block building strategies discussed above. Their performance is highly dependent on the effectiveness of block building techniques and the quality of blocking graph. On the other hand, pBlocking constructs meaningful blocks that effectively capture majority of the matching pairs and scores each block based on their quality to generate fewer non-matching pairs in the blocking graph. Meta-blocking techniques compute the blocking graph statically, prior to ER and thus can be used as the Z method in our Algorithm 1. In Fig. 2, we compare with classic meta-blocking and BLOSS, as the latter shows its superiority over Blast and SNB.

Progressive ER. Many applications need to resolve data sets efficiently but do not require the ER result to be complete. Recent literature described methods to compute the best possible partial solution. Such techniques include *pay-as-you-go* ER [38] that use 'hints' on records that are likely to refer to the same entity and more generally *progressive* ER such as the schema-agnostic method in [32] and the strategies in [1,28] that consider a limit on the execution time. In our discussion, we considered oracle-based techniques, namely Edge [36], Node [35] and Eager [9]. Edge processes record pairs in non-increasing probability of referring to same entity. In each round of pBlocking pipeline, it reorders the set of newly blocked pairs based on their probability. On the other hand, Node sorts records in non-increasing order of expected size of their clusters and processes sequentially to identify entities. In each iteration, it queries the considered record with the set of clusters formed by records processed in previous iterations. Edge and Node were not designed to optimize for progressiveness of ER. Eager optimizes for progressive F-score of the resolved entities by calculating a benefit metric for all unprocessed records (by leveraging the hybrid algorithm in [8]). The benefit metric of a record v is shown to be a robust estimate of the marginal gain in recall if v is processed. Additionally, it corrects oracle errors with an expander graph based error correction toolkit [9]. Differently from other progressive techniques, oracle-based methods consider a limit on the number of pairs that are examined by the oracle for matching/non-matching response. Such techniques were originally designed for dealing with the crowd but they can also be used with a variety of classifiers due to their flexibility. All these techniques naturally work in combination with pBlocking by sending as feedback their partial results and using the updated set of blocked pairs to regenerate ranking of pairwise comparisons to resolve entities.

Other ER methods. In addition to the above methods, we mention works on ER architectures that can help users to debug and tune parameters for the different components of ER [6,10,15,27]. Specifically, the approaches in [6,10] show how to leverage the crowd in this setting. All of these techniques are orthogonal to the scope of our work, and we do not consider them in our analysis. The previous work in [39] proposes to greedily merge records as they are matched by ER, while processing the blocks one at a time. Each merged record (containing tokens from the component records) is added to the unprocessed blocks, permitting its participation in the subsequent matching and merging by their iterative algorithm. Limitations of processing blocks one at a time has been shown in more recent blocking works [23].

9 Conclusions and future work

We have proposed a new blocking algorithm, pBlocking that progressively updates the relative scores of blocks and constructs new blocks by leveraging a novel feedback mechanism from partial ER results. Most of the techniques in the literature perform blocking as a preprocessing step to prune out redundant non-matching record pairs. However, these techniques are sensitive to the distribution wof cluster sizes and the amount of noise in the data set and thus are either highly efficient with poor recall or have high recall with poor efficiency. pBlocking can boost the effectiveness and efficiency of blocking across all data sets by jump-starting blocking with any of the standard techniques and then using new robust feedback-based methods for solving blocking sub-tasks in a data-driven way. To the best of our knowledge, pBlocking is the first framework where blocking and pair matching components of ER can help each other and produce high quality results in synergy.

Limitations and future work. One limitation of pBlocking is due to the initial set of seed blocks it considers to construct new blocks that prune out non-matching pairs. Any record pair that does not share any of the seed blocks would never be identified as a candidate even after running pBlocking. We believe that considering feedback from partial ER results can be helpful to explore other blocking strategies and is an interesting problem for future work.

Acknowledgements This work is supported partly by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.

References

1. Altowim, Y., Kalashnikov, D.V., Mehrotra, S.: Progressive approach to relational entity resolution. PVLDB **7**(11), 999–1010 (2014)
2. Bilenko, M., Kamath, B., Mooney, R.J.: Adaptive blocking: learning to scale up record linkage. In: ICDM (2006)
3. Christen, P., Churches, T., Hegland, M.: Febrl-a parallel open source data linkage system. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, pp. 638–647 (2004)
4. Crescenzi, V., Angelis, A. D., Firmani, D., Mazzei, M., Merialdo, P., Piai, F., Srivastava, D.: Alaska: a flexible benchmark for data integration tasks (2021)
5. dal Bianco, G., Gonçalves, M.A., Duarte, D.: Bloss: effective meta-blocking with almost no effort. Inf. Syst. **75**, 75–89 (2018)
6. Das, S., Paul Suganthan, G.C., Doan, A., Naughton, J.F., Krishnan, G., Deep, R., Arcaute, E., Raghavendra, V., Park, Y.: Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In: SIGMOD (2017)
7. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
8. Firmani, D., Saha, B., Srivastava, D.: Online entity resolution using an oracle. PVLDB **9**(5), 384–395 (2016)
9. Galhotra, S., Firmani, D., Saha, B., Srivastava, D.: Robust entity resolution using random graphs. In: SIGMOD (2018)
10. Gokhale, C., Das, S., Doan, A., Naughton, J.F., Rampalli, N., Shavlik, J., Zhu, X.: Corleone: hands-off crowdsourcing for entity matching. In: SIGMOD (2014)
11. Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate string joins in a database (almost) for free. VLDB **1**, 491–500 (2001)
12. Gruenheid, A., Dong, X.L., Srivastava, D.: Incremental record linkage. PVLDB **7**(9), 697–708 (2014)
13. Hernández, M.A., Stolfo, S.J.: The merge/purge problem for large databases. ACM Sigmod Rec. **24**, 127–138 (1995)
14. Hoeffding, W.: Probability inequalities for sums of bounded random variables. In: Hoeffding, W. (ed.) The Collected Works of Wassily Hoeffding, pp. 409–426. Springer, Berlin (1994)
15. Konda, P., Das, S., Paul Suganthan, G.C., Doan, A., Ardalan, A., Ballard, J.R., Li, H., Panahi, F., Zhang, H., Naughton, J., et al.: Magellan: toward building entity matching management systems. PVLDB **9**(12), 1197–1208 (2016)
16. Krause, J., Stark, M., Deng, J., Fei-Fei, L.: 3d object representations for fine-grained categorization. In: 4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13) (2013)
17. Manning, C.D., Manning, C.D., Schütze, H.: Foundations of statistical natural language processing (1999)
18. McCallum, A., Nigam, K., Ungar, L.H.: Efficient clustering of high-dimensional data sets with application to reference matching. In: Proceedings of ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 169–178 (2000)
19. McNeill, N., Kardes, H., Borthwick, A.: Dynamic record blocking: efficient linking of massive databases in mapreduce. Citeseer (2012)
20. Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., Raghavendra, V.: Deep learning for entity matching: a design space exploration. In: SIGMOD (2018)
21. Papadakis, G., Alexiou, G., Papastefanatos, G., Koutrika, G.: Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. PVLDB **9**(4), 312–323 (2015)
22. Papadakis, G., Ioannou, E., Palpanas, T., Niederee, C., Nejdl, W.: A blocking framework for entity resolution in highly heterogeneous information spaces. IEEE Trans. Knowl. Data Eng. **25**(12), 2665–2682 (2012)

23. Papadakis, G., Koutrika, G., Palpanas, T., Nejdl, W.: Meta-blocking: taking entity resolutionto the next level. TKDE **26**, 1946–1960 (2014)
24. Papadakis, G., Mandilaras, G., Gagliardelli, L., Simonini, G., Thanos, E., Giannakopoulos, G., Bergamaschi, S., Palpanas, T., Koubarakis, M.: Three-dimensional entity resolution with JedAI. Inf. Sys. **93**, 101565 (2020)
25. Papadakis, G., Papastefanatos, G., Koutrika, G.: Supervised meta-blocking. PVLDB **7**(14), 1929–1940 (2014)
26. Papadakis, G., Svirsky, J., Gal, A., Palpanas, T.: Comparative analysis of approximate blocking techniques for entity resolution. PVLDB **9**(9), 684–695 (2016)
27. Papadakis, G., Tsekouras, L., Thanos, E., Giannakopoulos, G., Palpanas, T., Koubarakis, M.: The return of JedAI: end-to-end entity resolution for structured and semi-structured data. PVLDB **11**(12), 1950–1953 (2018)
28. Papenbrock, T., Heise, A., Naumann, F.: Progressive duplicate detection. TKDE **27**(5), 1316–1329 (2015)
29. Penrose, M., et al.: Random Geometric Graphs, vol. 5. Oxford University Press, Oxford (2003)
30. Schütze, H., Manning, C.D., Raghavan, P.: Introduction to information retrieval. In: Proceedings of the International Communication of Association for Computing Machinery Conference, pp. 260 (2008)
31. Simonini, G., Bergamaschi, S., Jagadish, H.: Blast: a loosely schema-aware meta-blocking approach for entity resolution. PVLDB **9**(12), 1173–1184 (2016)
32. Simonini, G., Papadakis, G., Palpanas, T., Bergamaschi, S.: Schema-agnostic progressive entity resolution. IEEE Trans. Knowl. Data Eng. **31**(6), 1208–1221 (2018)
33. Verroios, V., Garcia-Molina, H.: Entity resolution with crowd errors. In: ICDE, pp. 219–230 (2015)
34. Verroios, V., Garcia-Molina, H., Papakonstantinou, Y.: Waldo: an adaptive human interface for crowd entity resolution. In: SIGMOD (2017)
35. Vesdapunt, N., Bellare, K., Dalvi, N.: Crowdsourcing algorithms for entity resolution. PVLDB **7**(12), 1071–1082 (2014)
36. Wang, J., Li, G., Kraska, T., Franklin, M. J., Feng, J.: Leveraging transitive relations for crowdsourced joins. In: SIGMOD (2013)
37. Whang, S.E., Garcia-Molina, H.: Incremental entity resolution on rules and data. VLDB J. **23**(1), 77–102 (2014)
38. Whang, S.E., Marmaros, D., Garcia-Molina, H.: Pay-as-you-go entity resolution. TKDE **25**(5), 1111–1124 (2013)
39. Whang, S.E., Menestrina, D., Koutrika, G., Theobald, M., Garcia-Molina, H.: Entity resolution with iterative blocking. In: SIGMOD (2009)
40. www.cs.umass.edu/mccallum/data/cora-refs.tar.gz