Scaling Entity Resolution to Large, Heterogeneous Data with Enhanced Meta-blocking

George Papadakis, George Papastefanatos, Themis Palpanas, Manolis Koubarakis

ABSTRACT
Entity Resolution constitutes a quadratic task that typically scales to large entity collections through blocking. The resulting blocks can be restructured by Meta-blocking in order to significantly increase precision at a limited cost in recall. Yet, its processing can be time-consuming, while its precision remains poor for configurations with high recall. In this work, we propose new meta-blocking methods that improve precision by up to an order of magnitude at a negligible cost in recall. We also introduce two efficiency techniques that, when combined, reduce the overhead time of Meta-blocking by more than an order of magnitude. We evaluate our approaches through an extensive experimental study over 6 real-world, heterogeneous datasets. The outcomes indicate that our new algorithms outperform all meta-blocking techniques as well as the state-of-the-art methods for block processing in all respects.

1. INTRODUCTION
A common task in the context of Web Data is Entity Resolution (ER), i.e., the identification of different entity profiles that pertain to the same real-world object. ER suffers from low efficiency, due to its inherently quadratic complexity: every entity profile has to be compared with all others. This problem is accentuated by the continuously increasing volume of heterogeneous Web Data; LOD-Stats1 recorded around 1 billion triples for Linked Open Data in December, 2011, which had grown to 85 billion by September, 2015. Typically, ER scales to these volumes of data through blocking [4].

The goal of blocking is to boost precision and time efficiency at a controllable cost in recall [4, 5, 21]. To this end, it groups similar profiles into clusters (called blocks) so that it suffices to compare the profiles within each block [7, 8]. Blocking methods for Web Data are confronted with high levels of noise, not only in attribute values, but also in attribute names. In fact, they involve an unprecedented schema heterogeneity: Google Base2 alone encompasses 100,000 distinct schemata that correspond to 10,000 entity types [17]. Most blocking methods deal with these high levels of schema heterogeneity through a schema-agnostic functionality that completely disregards schema information and semantics [5]. They also rely on redundancy, placing every entity profile into multiple blocks so as to reduce the likelihood of missed matches [4, 22].

The simplest method of this type is Token Blocking [21]. In essence, it splits the attribute values of every entity profile into tokens based on whitespace; then, it creates a separate block for every token that appears in at least two profiles. To illustrate its functionality, consider the entity profiles in Figure 1(a), where \( p_1 \) and \( p_2 \) match with \( p_3 \) and \( p_4 \), respectively; Token Blocking clusters them in the blocks of Figure 1(b). Despite the schema heterogeneity and the noisy values, both pairs of duplicates co-occur in at least one block. Yet, the total cost is 13 comparisons, which is rather high, given that the brute-force approach executes 15 comparisons.

This is a general trait of redundancy-based blocking methods: in their effort to achieve high recall in the context of noisy and heterogeneous data, they produce a large number of unnecessary comparisons. These come in two forms [22, 23]: the redundant ones repeatedly compare the same entity profiles across different blocks, while the superfluous ones compare non-matching profiles. In our example, \( b_2 \) and \( b_3 \) contain one redundant comparison each, which are repeated in \( b_1 \) and \( b_4 \), respectively; all other blocks entail superfluous comparisons between non-matching entity profiles, except for the redundant comparison \( p_1 p_3 \) in \( b_6 \) (it is repeated in \( b_8 \)). In total, the blocks of Figure 1(b) involve 3 redundant and 8 superfluous out of the 13 comparisons.

Block Processing. To improve the quality of redundancy-based blocks, methods such as Meta-blocking [5, 7, 22], Comparison Propagation [21] and Iterative Blocking [27] aim to process them in the optimal way (see Section 2 for more details). Among these methods, Meta-blocking achieves the best balance between preci-

1http://stats.lod2.eu
2http://www.google.com/base
We introduce Block Filtering, which intelligently removes profiles from blocks, in which their presence is unnecessary. This acts as a pre-processing technique that shrinks the blocking graph, discarding more than half of its unnecessary edges, on average. As a result, the running time is also reduced to half, on average.

(ii) We accelerate the creation and the pruning of the blocking graph by minimizing the computational cost for edge weighting, which is the bottleneck of Meta-blocking. Our approach reduces its running time by 30% to 70%.

In combination, these two techniques restrict drastically the overhead of Meta-blocking even on commodity hardware. For example, the blocking graph mentioned earlier is now processed within just 3 hours, instead of 16.

Second, we enhance the precision of Meta-blocking in two ways:
(i) We redefine two pruning algorithms so that they produce restructured blocks with no redundant comparisons. On average, they save 30% more comparisons for the same recall.
(ii) We introduce two new pruning algorithms that rely on a generic property of the blocking graph: the reciprocal links. That is, our algorithms retain only the edges that are important for both incident profiles. Their recall is slightly lower than the existing techniques, but precision raises by up to an order of magnitude.

We analytically examine the performance of our methods using 6 real-world established benchmarks, which range from few thousands to several million entities. Our experimental results designate that our algorithms consistently exhibit the best balance between recall, precision and run-time for the main types of ER applications among all meta-blocking techniques. They also outperform the best relevant methods in the literature to a significant extent.

Contributions & Paper Organization. In summary, we make the following contributions:

- We improve the running time of Meta-blocking by an order of magnitude in two complementary ways: by cleaning the blocking graph from most of its noisy edges, and by accelerating the estimation of edge weights.
- We present four new pruning algorithms that raise precision by 30% to 100% at a small (if any) cost in recall.
- We experimentally verify the superior performance of our new methods through an extensive study over 6 datasets with different characteristics. In this way, our experimental results provide insights into the best configuration for Meta-blocking, depending on the data and the application at hand. The code and the data of our experiments are publicly available for any interested researcher.

The rest of the paper is structured as follows: Section 2 delves into the most relevant works in the literature, while Section 3 elaborates on the main notions of Meta-blocking. In Section 4, we introduce two methods for minimizing its running time, and in Section 5, we present new pruning algorithms that boost the precision of Meta-blocking at no or limited cost in recall. Section 6 presents our thorough experimental evaluation, while Section 7 concludes the paper along with directions for future work.

2. RELATED WORK

Entity Resolution has been the focus of numerous works that aim to tame its quadratic complexity and scale it to large volumes of data [4, 8]. Blocking is the most popular among the proposed approximate techniques [5, 7]. Some blocking methods produce disjoint blocks, such as Standard Blocking [9]. Their majority, though, yields overlapping blocks with redundant comparisons in an effort to achieve high recall in the context of noisy and heterogeneous data [4]. Depending on the interpretation of redundancy, blocking methods are distinguished into three categories [22]:

(i) The redundancy-positive methods ensure that the more blocks two entity profiles share, the more likely they are to be matching. In this category fall the Suffix Arrays [1], Q-grams Blocking [12], Attribute Clustering [21] and Token Blocking [21].
(ii) The redundancy-negative methods ensure that the most similar entity profiles share just one block. In Canopy Clustering [19], for instance, the entity profiles that are highly similar to the cur-

---

3See http://sourceforge.net/projects/erframework for both the code and the datasets.
rent seed are removed from the pool of candidate matches and are exclusively placed in its block.

(iii) The redundancy-neutral methods yield overlapping blocks, but the number of common blocks between two profiles is irrelevant to their likelihood of matching. As such, consider the single-pass Sorted Neighborhood [13]: all pairs of profiles co-occur in the same number of blocks, which is equal to the size of the sliding window.

Another line of research focuses on developing techniques that optimize the processing of an existing block collection, called block processing methods. In this category falls Meta-blocking [7], which operates exclusively on top of redundancy-positive blocking methods [20, 21]. Its pruning can be either unsupervised [22] or supervised [23]. The latter achieves higher accuracy than the former, due to the composite pruning rules that are learned by a classifier trained over a set of labelled edges. In practice, though, its utility is limited, as there is no effective and efficient way for extracting the required training set from the input blocks. For this reason, we exclusively consider unsupervised Meta-blocking in this work.

Other prominent block processing methods are the following:

(i) Block Purgation [21] aims for discarding oversized blocks that are dominated by redundant and superfluous comparisons. It automatically sets an upper limit on the comparisons that can be contained in a valid block and purges those blocks that exceed it. Its functionality is coarser and, thus, less accurate than Meta-blocking, because it targets entire blocks instead of individual comparisons. However, it is complementary to Meta-blocking and is frequently used as a pre-processing step [22, 23].

(ii) Comparison Propagation [21] discards all redundant comparisons from a block collection without any impact on recall. In a small scale, this can be accomplished directly, using a central data structure \( H \) that hashes all executed comparisons; then, a comparison is executed only if it is not contained in \( H \). Yet, in the scale of billions of comparisons, Comparison Propagation can only be accomplished indirectly: the input blocks are enumerated according to their processing order and the Entity Index is built. This is an inverted index that points from entity ids to block ids. Then, a comparison \( p_i \) in block \( b_i \) is executed (i.e., non-redundant) only if it satisfies the Least Common Block Index condition (LeCoBI for short). That is, if the id \( k \) of the current block \( b_k \) equals the least common block id of the profiles \( p_i \) and \( p_j \). Comparison Propagation is competitive to Meta-blocking, but targets only redundant comparisons. We compare their performance in Section 6.4.

(ii) Iterative Blocking [27] propagates all identified duplicates to the subsequently processed blocks so as to save repeated comparisons and to detect more duplicates. Hence, it improves both precision and recall. It is competitive to Meta-blocking, too, but it targets exclusively redundant comparisons between matching profiles. We employ it as our second baseline method in Section 6.4.

3. PRELIMINARIES

Entity Resolution. An entity profile, \( p \), is defined as a uniquely identified collection of name-value pairs that describe a real-world object. A set of profiles is called entity collection, \( E \). Given \( E \), the goal of ER is to identify all profiles that describe the same real-world object; two such profiles, \( p_i \) and \( p_j \), are called duplicates (\( p_irighteq p_j \)) and their comparison is called matching. The set of all duplicates in the input entity collection \( E \) is denoted by \( D(E) \), with \( |D(E)| \) symbolizing its size (i.e., the number of existing duplicates).

Depending on the input entity collection(s), we identify two ER tasks [4, 21, 22]: (i) Dirty ER takes as input a single entity collection with duplicates and produces as output a set of equivalence clusters. (ii) Clean-Clean ER receives two duplicate-free, but overlapping entity collections, \( E_1 \) and \( E_2 \), and identifies the matching entity profiles between them. In the context of Databases, the former task is called Deduplication and the latter Record Linkage [4].

Blocking improves the run-time of both ER tasks by grouping similar entity profiles into blocks so that comparisons are limited between co-occurring profiles. Placing an entity profile into a block is called block assignment. Two profiles, \( p_i \) and \( p_j \), assigned to the same block are called co-occurring and their comparison is denoted by \( c_{ij} \). An individual block is symbolized by \( b \), with \( |b| \) denoting its size (i.e., number of profiles) and \( |\{b\}| \) denoting its cardinality (i.e., number of comparisons). A set of blocks \( B \) is called block collection, with \( |B| \) denoting its size (i.e., number of blocks) and \( |\{B\}| \) its cardinality (i.e., total number of comparisons): \( |\{B\}| = \sum_{b \in B} |\{b\}| \).

Performance Measures. To assess the effectiveness of a blocking method, we follow the best practice in the literature, which treats entity matching as an orthogonal task [4, 5, 7]. We assume that two duplicate profiles can be detected using any of the available matching methods as long as they co-occur in at least one block. \( D(B) \) stands for the set of co-occurring duplicate profiles and \( |D(B)| \) for its size (i.e., the number of detected duplicates).

In this context, the following measures are typically used for estimating the effectiveness of a block collection \( B \) that is extracted from the input entity collection \( E \) [4, 5, 22]:

(i) Pairs Quality (\( PQ(B) \)) corresponds to precision, assessing the portion of comparisons that involve a non-redundant pair of duplicates. In other words, it considers as true positives the matching comparisons and as false positives the superfluous and the redundant ones (given that some of the redundant comparisons involve duplicate profiles, \( PQ \) offers a pessimistic estimation of precision). More formally, \( PQ = |D(B)|/|\{B\}| \). \( PQ \) takes values in the interval \([0, 1] \), with higher values indicating higher precision for \( B \).

(ii) Pairs Completeness (\( PC(B) \)) corresponds to recall, assessing the portion of existing duplicates that can be detected in \( B \). More formally, \( PC = |D(B)|/|\{E\}| \). \( PC \) is defined in the interval \([0, 1] \), with higher values indicating higher recall.

The goal of blocking is to maximize both \( PC \) and \( PQ \) so that the overall effectiveness of ER exclusively depends on the accuracy of the entity matching method. This requires that \( |D(B)| \) is maximized, while \( |\{B\}| \) is minimized. However, there is a clear trade-off between \( PC \) and \( PQ \): the more comparisons are executed (higher \( |\{B\}| \)), the more duplicates are detected (higher \( |D(B)| \)), thus increasing \( PC \); given, though, that \( |\{B\}| \) increases quadratically for a linear increase in \( |D(B)| \) [10, 11], \( PQ \) is reduced. Hence, a blocking method is effective if it achieves a good balance between \( PC \) and \( PQ \).

To assess the time efficiency of a block collection \( B \), we use two measures [22, 23]:

(i) Overhead Time (\( OT(B) \)) measures the time required for extracting \( B \) either from the input entity collection \( E \) or from another block collection \( B' \).

(ii) Resolution Time (\( RT(B) \)) is equal to \( OT \) plus the time required to apply an entity matching method to all comparisons in the restructured blocks. As such, we use the Jaccard similarity of all tokens in the values of two entity profiles for entity matching – this approach does not affect the relative efficiency of the examined methods and is merely used for demonstration purposes.

For both measures, the lower their value, the more efficient is the corresponding block collection.

Meta-blocking. The redundancy-positive block collections place every entity profile into multiple blocks, emphasizing recall at the cost of very low precision. Meta-blocking aims for improving this balance by restructuring a redundancy-positive block collection \( B \) into a new one \( B' \) that contains a small part of the original unnecessary comparisons, while retaining practically the same recall [22]. More formally, \( PC(B') \approx PC(B) \) and \( PQ(B') \gg PQ(B) \).
Weighting Schemes
1) Aggregate Reciprocal Comparisons (ARCS)
2) Common Blocks (CBS)
3) Enhanced Common Blocks (ECBS)
4) Jaccard Similarity (JS)
5) Enhanced Jaccard Similarity (EJS)

Pruning Algorithms
1) Cardinality Edge Pruning (CEP)
2) Cardinality Node Pruning (CNP)
3) Weighted Edge Pruning (WEP)
4) Weighted Node Pruning (WNP)

Figure 3: All configurations for the two parameters of Meta-blocking: the weighting scheme and the pruning algorithm.

| Aggregate Reciprocal Comparisons Scheme | ARCS(p_i, p_j, B) = $\sum_{k=\ell}^{B} \frac{1}{|\bar{B}_k|}$ |
|----------------------------------------|--------------------------------------------------|
| Common Blocks Scheme                   | CBS(p_i, p_j, B) = $|B_i| - |B_j|$ |
| Enhanced Common Blocks Scheme          | ECBS(p_i, p_j, B) = CBS(p_i, p_j, B) - $|B_i|/|B_j| |B_j|/|B_i|$ |
| Jaccard Scheme                         | JS(p_i, p_j, B) = $|B_i|/|B_i| + |B_j|/|B_j| - |B_i|/|B_j|$ |
| Enhanced Jaccard Scheme                | EJS(p_i, p_j, B) = JS(p_i, p_j, B) - $|B_i|/|B_i| \cdot \log |B_j|\cdot \log |B_i|/|B_j|$ |

Figure 4: The formal definition of the five weighting schemes. $B \subseteq B$ denotes the set of blocks containing $p_i, B_{ij} \subseteq B$ the set of blocks shared by $p_i$ and $p_j$, and $v_i$ the degree of node $v_i$.

Central to this procedure is the blocking graph $G_B$, which captures the co-occurrences of profiles within the blocks of $B$. Its nodes correspond to the profiles in $B$, while its undirected edges connect the co-occurring profiles. The number of edges in $G_B$ is called graph size ($|E_B|$) and the number of nodes graph order ($|V_B|$).

Meta-blocking prunes the edges of the blocking graph in a way that leaves the matching profiles connected. Its functionality is configured by two parameters: (i) the scheme that assigns weights to the edges, and (ii) the pruning algorithm that discards the edges that are unlikely to connect duplicate profiles. The two parameters are independent in the sense that every configuration of the one is compatible with any configuration of the other (see Figure 3).

In more detail, five schemes have been proposed for weighting the edges of the blocking graph [22]. Their formal definitions are presented in Figure 4. They all normalize their weights to [0, 1] so that the higher values correspond to edges that are more likely to connect matching profiles. The rationale behind each scheme is the following: ARCS captures the intuition that the smaller the blocks two profiles share, the more likely they are to be matching; CBS expresses the fundamental property of redundancy-positive block collections that two profiles are more likely to match, when they share many blocks; ECBS improves CBS by discounting the effect of the profiles that are placed in a large number of blocks; JS estimates the portion of blocks shared by two profiles; EJS improves JS by discounting the effect of profiles involved in too many non-redundant comparisons (i.e., they have a high node degree).

Based on these weighting schemes, Meta-blocking discards part of the edges of the blocking graph using an edge- or a node-centric pruning algorithm. The former iterates over the edges of the blocking graph and retains the globally best ones, as in Figure 2(b); the latter iterates over the nodes of the blocking graph and retains the locally best edges. An example of node-centric pruning appears in Figure 5(a); for each node in Figure 2(a), it has retained the incident edges that exceed the average weight of the neighborhood. For clarity, the retained edges are directed and outgoing, since they might be preserved in the neighborhoods of both incident profiles. Again, every retained edge forms a new block, yielding the restructured block collection in Figure 5(b).

Every pruning algorithm relies on a pruning criterion. Depending on its scope, this can be either a global criterion, which applies to the entire blocking graph, or a local one, which applies to an individual node neighborhood. With respect to its functionality, the pruning criterion can be a weight threshold, which specifies the minimum weight of the retained edges, or a cardinality threshold, which determines the maximum number of retained edges.

Every combination of a pruning algorithm with a pruning criterion is called prunning scheme. The following four pruning schemes were proposed in [22] and were experimentally verified to achieve a good balance between PC and PQ:

(i) Cardinality Edge Pruning (CEP) couples the edge-centric pruning with a global cardinality threshold, retaining the top-$K$ edges of the entire blocking graph, where $K = \sum_{b \subseteq B} |b|/2$.

(ii) Cardinality Node Pruning (CNP) couples the node-centric pruning with a global cardinality threshold. For each node, it retains the top-$K$ edges of its neighborhood, with $K = \sum_{b \subseteq B} |b|/|E| - 1$.

(iii) Weighted Edge Pruning (WEP) couples edge-centric pruning with a global weight threshold equal to the average edge weight of the entire blocking graph.

(iv) Weighted Node Pruning (WNP) couples the node-centric pruning with a local weight threshold equal to the average edge weight of every node neighborhood.

The weight-based schemes, WEP and WNP, discard the edges that do not exceed their weight threshold and typically perform a shallow pruning that retains high recall [22]. The cardinality-based schemes, CEP and CNP, rank the edges of the blocking graph in descending order of weight and retain a specific number of the top ones. For example, if CEP retained the 4 top-weighted edges of the graph in Figure 2(a), it would produce the pruned graph of Figure 2(b), too. Usually, CEP and CNP perform deeper pruning than WEP and WNP, trading higher precision for lower recall [22].

Applications of Entity Resolution. Based on their performance requirements, we distinguish ER applications into two categories:

(i) The efficiency-intensive applications aim to minimize the response time of ER, while detecting the vast majority of the duplicates. More formally, their goal is to maximize precision ($PQ$) for a recall ($PC$) that exceeds 0.80. To this category belong real-time applications or applications with limited temporal resources, such as Pay-as-you-go ER [26], entity-centric search [25] and crowd-sourcing ER [6]. Ideally, their goal is to identify a new pair of duplicate entities with every executed comparison.

(ii) The effectiveness-intensive applications can afford a higher response time in order to maximize recall. At a minimum, recall ($PC$) should not fall below 0.95. Most of these applications correspond to off-line batch processes like data cleaning in data warehouses, which practically call for almost perfect recall [2]. Yet, higher precision ($PQ$) is pursued even in off-line applications so as to ensure that they scale to voluminous datasets.

Meta-blocking accommodates the effectiveness-intensive applications through the weight-based pruning schemes (WEP, WNP) and the efficiency-intensive applications through the cardinality-based schemes (CEP, CNP).
4. TIME EFFICIENCY IMPROVEMENTS

We now propose two methods for accelerating the processing of Meta-blocking, minimizing its $O\text{Time}$: (i) Block Filtering, which operates as a pre-processing step that reduces the size of the blocking graph, and (ii) Optimized Edge Weighting, which minimizes the computational cost for the weighting of individual edges.

4.1 Block Filtering

This approach is based on the idea that each block has a different importance for every entity profile it contains. For example, a block with thousands of profiles is usually superfluous for most of them, but it may contain a couple of matching entity profiles that do not co-occur in another block; for them, this particular block is indispensable. Based on this principle, Block Filtering structures a block collection by removing profiles from blocks, in which their presence is not necessary. The importance of a block $b_i$ for an individual entity profile $p_i \in b_i$ is implicitly determined by the maximum number of blocks $p_i$ participates in.

Continuing our example with the blocks in Figure 1(b), assume that their importance is inversely proportional to their id; that is, $b_1$ and $b_2$ are the most and the least important blocks, respectively. A possible approach to Block Filtering would be to remove every entity profile from the least important of its blocks, i.e., the one with the largest block id. The resulting block collection appears in Figure 6(a). We can see that Block Filtering reduces the 15 original comparisons to just 5. Yet, there is room for further improvements, due to the presence of 2 redundant comparisons, one in $b_1'$ and another one in $b_2'$, and 1 superfluous in block $b_4'$. Using the JS weighting scheme, the graph corresponding to these blocks is presented in Figure 6(b) and the pruned graph produced by WEP appears in Figure 6(c). In the end, we get the 2 matching comparisons in Figure 6(d). This is a significant improvement over the 5 comparisons in Figure 2(c), which were produced by applying the same pruning scheme directly to the blocks of Figure 1(b).

In more detail, the functionality of Block Filtering is outlined in Algorithm 1. First, it orders the blocks of the input collection $B$ in descending order of importance (Line 3). Then, it determines the maximum number of blocks per entity profile (Line 4). This requires an iteration over all blocks in order to count the block assignments per entity profile. Subsequently, it iterates over all blocks in the specified order (Line 5) and over all profiles in each block (Line 6). The profiles that have more block assignments than their threshold are discarded, while the rest are retained in the current block (Lines 7-10). In the end, the current block is retained only if it still contains at least two entity profiles (Lines 11-12).

The time complexity of this procedure is dominated by the sorting of blocks, i.e., $O(|B| \log |B|)$. Its space complexity is linear with respect to the size of the input, $O(|E|)$, because it maintains a threshold and a counter for every entity profile.

Algorithm 1: Block Filtering.

Input: $B$ the input block collection
Output: $B'$ the restructured block collection

1. $B' \leftarrow \{\}$
2. $\text{counter}[i] \leftarrow \{\}$: // count blocks per profile
3. $\text{orderBlocks}(B)$: // sort in descending importance
4. $\maxBlocks[i] \leftarrow \text{getThresholds}(B)$: // limit per profile
5. $\text{foreach } b_i \in B \text{ do} $ // check all blocks
6. $\text{foreach } p_i \in b_i \text{ do} $ // check all profiles
7. if $\text{counter}[i] > \maxBlocks[i]$ then
8. $b_i \leftarrow b_i \setminus p_i$: // remove profile
9. else
10. $\text{counter}[i]++$: // increment counter
11. if $|b_i| > 1$ then // retain blocks with
12. $B' \leftarrow B' \cup b_i$: // at least 2 profiles
13. return $B'$.

Figure 7: (a) Using Block Filtering for pre-processing the blocking graph of Meta-blocking, and (b) using Block Filtering as a graph-free Meta-blocking method.

The performance of Block Filtering is determined by two factors: (i) The criterion that specifies the importance of a block $b_i$. This can be defined in various ways and ideally should be different for every profile in $b_i$. For higher efficiency, though, we use a criterion that is common for all profiles in $b_i$. It is also generic, applying to any block collection, independently of the underlying ER task or the schema heterogeneity. This criterion is the cardinality of $b_i$, $|b_i|$, presuming that the less comparisons a block contains, the more important it is for its entities. Thus, Block Filtering sorts a block collection from the smallest block to the largest one.

(ii) The filtering ratio ($r$) that determines the maximum number of block assignments per profile. It is defined in the interval $[0, 1]$ and expresses the portion of blocks that are retained for each profile. For example, $r=0.5$ means that each profile remains in the first half of its associated blocks, after sorting them in ascending cardinality. We experimentally fine-tune this parameter in Section 6.2.

Instead of a local threshold per entity profile, we could apply the same global threshold to all profiles. Preliminary experiments, though, demonstrated that this approach exhibits low performance, as the number of blocks associated with every profile varies largely, depending on the quantity and the quality of information it contains. This is particularly true for Clean-Clean ER, where $E_1$ and $E_2$ usually differ largely in their characteristics. Hence, it is difficult to identify the break-even point for a global threshold that achieves a good balance between recall and precision for all profiles.

Finally, it is worth noting that Block Filtering can be used in two fundamentally different ways, which are compared in Section 6.4.

(i) As a pre-processing method that prunes the blocking graph before applying Meta-blocking – see Figure 7(a).

(ii) As a graph-free Meta-blocking method that is combined only with Comparison Propagation – see Figure 7(b).

The latter workflow skips the blocking graph, operating on the level of individual profiles instead of profile pairs. Thus, it is expected to be significantly faster than all graph-based algorithms. If it achieves higher precision, as well, it outperforms the graph-based workflow in all respects, rendering the blocking graph unnecessary.
Algorithm 2: Original Edge Weighting.

Input: \( B \) the input block collection
Output: \( W \) the set of edge weights

1. \( W \leftarrow \{\}; \)
2. \( EI \leftarrow \text{returnEntityIndex} (B); \)
3. foreach \( b_i \in B \) do // check all blocks
   4.   foreach \( c_{ij} \in b_i \) do // check all comparisons
      5.     \( B_i \leftarrow EI.\text{getBlockList} (p_i); \)
      6.     \( B_j \leftarrow EI.\text{getBlockList} (p_j); \)
      7.   \( \text{commonBlocks} \leftarrow \{\}; \)
      8.   foreach \( m \in B_i \) do
         9.     foreach \( n \in B_j \) do
            10.    if \( m < n \) then break; // repeat until
               11.    if \( n < m \) then continue; // finding common id
               12.    if \( \text{commonBlocks} = \{\} \) then // 1\textsuperscript{st} common id
                  13.        if \( m = k \) then // it violates LeCoBI
                     14.            \( \text{commonBlocks}++; \)
                     15.            \( w_{i,j} \leftarrow \text{calculateWeight} (\text{commonBlocks}, B_i, B_j); \)
                  16.    \( W \leftarrow W \cup \{w_{i,j}\}; \)
               17.    return \( W; \)

4.2 Optimized Edge Weighting

A complementary way of speeding up Meta-blocking is to accelerate its bottleneck, i.e., the estimation of edge weights. Intuitively, we want to minimize the computational cost of the procedure that derives the weight from every individual edge. Before we explain in detail our solution, we give some background, by describing how the existing Edge Weighting algorithm operates.

The blocking graph cannot be materialized in memory in the scale of million nodes and billion edges. Instead, it is implemented implicitly. The key idea is that every edge \( e_{ij} \) in the blocking graph \( G_B \) corresponds to a non-redundant comparison \( c_{ij} \) in the block collection \( B \). In other words, a comparison \( c_{ij} \in b_i \in B \) defines an edge \( e_{ij} \) in \( G_B \) as long as it satisfies the LeCoBI condition (see Section 2). The condition is checked with the help of the Entity Index during the core process that derives the blocks shared by \( p_i \) and \( p_j \).

In more detail, the original implementation of Edge Weighting is outlined in Algorithm 2. Note that \( B \) stands for the block list of \( p_i \), i.e., the set of block ids associated with \( p_i \), sorted from the smallest to the largest one. The core process appears in Lines 7-15 and relies on the established process of Information Retrieval for intersecting the posting lists of two terms while answering a keyword query [18]: it iterates over the blocks lists of two co-occurring profiles in parallel, incrementing the counter of common blocks for every id they share (Line 15). This process is terminated in case the first common block id does not coincide with the id of the current block \( b_i \), thus indicating a redundant comparison (Lines 12-14).

Our observation is that since this procedure is repeated for every comparison in \( B \), a more efficient implementation would significantly reduce the run time of Meta-blocking. To this end, we develop a filtering technique inspired by similarity joins [14].

Prefix Filtering [3, 14] is a prominent method, which prunes dissimilar pairs of strings with the help of the minimum similarity threshold \( t \) that is determined \( a\text{-}\text{priori} \); \( t \) can be defined with respect to various similarity metrics that are essentially equivalent, due to a set of transformations [14]. Without loss of generality, we assume in the following that \( t \) is normalized in \([0, 1]\), just like the edge weights, and that it expresses a Jaccard similarity threshold.

Adapted to edge weighting, Prefix Filtering represents every profile \( p_i \) by the \([1 - t]|B_i|\) smallest blocks of \( B_i \). The idea is that pairs having disjoint representations cannot exceed the similarity threshold \( t \). For example, for \( t=0.8 \) an edge \( e_{ij} \) could be pruned using \( 1/5 \) of \( B_i \) and \( B_j \), thus speeding up the nested loops in Lines 8-9 of Algorithm 2. Yet, there are 3 problems with this approach:

(i) For the weight-based algorithms, the pruning criterion \( t \) can only be determined \( a\text{-}\text{posteriori} \) – after averaging all edge weights in the entire graph (WEP), or in a node neighborhood (WNP). As a result, the optimizations of Prefix Filtering apply only to the pruning phase of WEP and WNP and not to the initial construction of the blocking graph.

(ii) For the cardinality-based algorithms CEP and CNP, \( t \) equals the minimum edge weight in the sorted stack with the top-weighted edges. Thus, its value is continuously modified and cannot be used for \( a\text{-}\text{priori} \) building optimized entity representations.

(iii) Preliminary experiments demonstrated that \( t \) invariably takes very low values, below 0.1, for all combinations of pruning algorithms and weighting schemes. These low thresholds force all versions of Prefix Filtering to consider the entire block lists \( B_i \) and \( B_j \) as entity representations, thus ruining their optimizations.

For these reasons, we propose a novel implementation that is independent of the similarity threshold \( t \). Our approach is outlined in Algorithm 3. Instead of iterating over all comparisons in \( B \), it iterates over all input profiles in \( E \) (Line 3). The core procedure in Lines 6-12 works as follows: for every profile \( p_i \), it iterates over all co-occurring profiles in the associated blocks and records their frequency in an array. At the end of the process, \( \text{commonBlocks}[j] \) indicates the number of blocks shared by \( p_i \) and \( p_j \). This information is then used in Lines 13-16 for estimating the weight \( w_{i,j} \). This method is reminiscent of ScanCount [16]. Note that the array \( f\text{lags} \) helps us to avoid reallocating memory for \( \text{commonBlocks} \) in every iteration, a procedure that would be costly, due to its size, \(|E| [16]; \text{neighbors} \) is a hash set that stores the unique profiles that co-occur with \( p_i \), gathering the distinct neighbors of node \( n_i \) in the blocking graph without evaluating the LeCoBI condition.

4.3 Discussion

The average time complexity of Algorithm 2 is \( O(|B|BPE[|B|]) \), where \( BPE(B) = \sum_{i \in B} |B_i|/|E| \) is the average number of blocks associated with every profile in \( B \); \( 2BPE \) corresponds to the average computational cost of the nested loops in Lines 8-9, while \(|B| \) stems from the nested loops in Lines 3-4, which iterate over all comparisons in \( B \).
Block Filtering improves this time complexity in two ways:
(i) It reduces $|B|$ by discarding a large part of the redundant and superfluous comparisons in $B$.
(ii) It reduces $2 \cdot BPE$ to $r \cdot BPE$ by removing every profile from $(1-r)100\%$ of its associated blocks, where $r$ is the filtering ratio. The computational cost of Algorithm 3 is determined by two procedures that yield an average time complexity of $O(|B| + |v||E|):
(i) The three nested loops in Lines 3-7. For every block $b$, these loops iterate over $|b|-1$ of its entity profiles (i.e., over all profiles except $p_j$) for $|b|$ times – once for each entity profile. Therefore, the process in Lines 8-12 is repeated $|b| \cdot (|b|-1) \cdot |b||B|$ times and the overall complexity of the three nested loops is $O(|B|)$.
(ii) The loop in Lines 13-16. Its cost is analogous to the average node degree $|v|$, i.e., the average number of neighbors per profile. It is repeated for every profile and, thus, its overall cost is $O(|v||E|)$.
Comparing the two algorithms, we observe that the optimized implementation minimizes the computational cost of the process that is applied to each comparison: instead of intersecting the associated block lists, it merely updates 2-3 cells in 2 arrays and adds an entity id in the set of neighboring profiles. The former process involves two nested loops with an average cost of $O(2 \cdot BC)$, while the latter processes have a constant complexity, $O(1)$. Note that Algorithm 3 incorporates the loop in Lines 13-16, which has complexity of $O(|v||E|)$. In practice, though, this is considerably lower than both $O(2 \cdot BPE \cdot |B|)$ and $O(|B|)$, as we show in Section 6.3.

5. PRECISION IMPROVEMENTS
We now introduce two new pruning algorithms that significantly enhance the effectiveness of Meta-blocking, increasing precision for similar levels of recall: (i) Redefined Node-centric Pruning, which removes all redundant comparisons from the restructured blocks of CNP and WNP, and (ii) Reciprocal Pruning, which inhibits the most promising matches from the reciprocal links of the directed pruned blocking graphs of CNP and WNP.
There are several reasons why we exclusively focus on improving the precision of CNP and WNP:
(i) Node-centric algorithms are quite robust to recall, since they retain the most likely matches for each node and, thus, guarantee to include every profile in the restructured blocks. Edge-centric algorithms do not provide such guarantees, because they retain the overall best edges from the entire graph.
(ii) Node-centric algorithms are more flexible, in the sense that their performance can be improved in generic, algorithmic ways. Instead, edge-centric algorithms improve their performance only with threshold fine-tuning. This approach, though, is application-specific, as it is biased by the characteristics of the block collection at hand. Another serious limitation is that some superfluous comparisons cannot be pruned without discarding part of the matching ones. For instance, the superfluous edge $e_{5,4}$ in Figure 2(a) has a higher weight than both matching edges $e_{1,3}$ and $e_{5,4}$.
(iii) There is more room for improvement in node-centric algorithms, because they exhibit lower precision than the edge-centric ones. They process every profile independently of the others and they often retain the same edge for both incident profiles, thus yielding redundant comparisons. They also retain more superfluous comparisons than the edge-centric algorithms in most cases [22]. As an example, consider Clean-Clean ER: every profile from both

This seems similar to the effect of Prefix Filtering, but there are fundamental differences: (i) Prefix Filtering does not reduce the number of executed comparisons; it just accelerates their pruning.
(ii) Prefix Filtering relies on a similarity threshold for pairs of profiles, while the filtering ratio $r$ pertains to individual profiles.

![Figure 8: (a) The undirected pruned blocking graph corresponding to the directed one Figure 5(a), and (b) the corresponding block collection.](image)

entity collections retains its connections with several incident nodes, even though only one of them is matching with it.
Nevertheless, our experiments in Section 6.4 demonstrate that our new node-centric algorithms outperform the edge-centric ones, as well. They also cover both efficiency- and effectiveness-intensive ER applications, enhancing both CNP and WNP.

5.1 Redefined Node-centric Pruning
We can enhance the precision of both CNP and WNP without any impact on recall by discarding all the redundant comparisons they retain. Assuming that the blocking graph is materialized, a straightforward approach is to convert the directed pruned graph into an undirected one by connecting every pair of neighboring nodes with a single undirected edge – even if they are reciprocally linked. In the extreme case where every retained edge has a reciprocal link, this saves 50% more comparisons and doubles precision.
As an example, the directed pruned graph in Figure 5(a) can be transformed into the undirected pruned graph in Figure 8(a); the resulting blocks, which are depicted in Figure 8(b), reduce the retained comparisons from 9 to 5, while maintaining the same recall as the blocks in Figure 5(b): $p_1$-$p_3$ co-occur in $b'_1$, and $p_2$-$p_4$ in $b'_2$. Yet, it is impossible to materialize the blocking graph in memory in the scale of billions of edges. Instead, the graph is implicitly implemented as explained in Section 4.2. In this context, a straightforward solution for improving CNP and WNP is to apply Comparison Propagation to their output. This approach, though, entails a significant overhead, as it evaluates the LeGoBi condition for every retained comparison; on average, its total cost is $O(2 \cdot BPE \cdot |B|)$.
The best solution is to redefine CNP and WNP so that Comparison Propagation is integrated into their functionality. The new implementations are outlined in Algorithms 4 and 5, respectively. In both cases, the processing consists of two phases:
(i) The first phase involves a node-centric functionality that goes through the nodes of the blocking graph and derives the pruning criterion from their neighborhood. A central data structure stores the top-$k$ nearest neighbors (CNP) or the weight threshold (WNP) per node neighborhood. In total, this phase iterates twice over every edge of the blocking graph – once for each incident node.
(ii) The second phase operates in an edge-centric fashion that goes through all edges, retaining those that satisfy the pruning criterion for at least one of the incident nodes. Thus, every edge is retained at most once, even if it is important for both incident nodes. In more detail, Redefined CNP iterates over all nodes of the blocking graph to extract their neighborhood and calculate the corresponding cardinality threshold $k$ (Lines 2-4 in Algorithm 4). Then it iterates over the edges of the current neighborhood and places the top-$k$ weighted ones in a sorted stack (Lines 5-8). In the second phase, it iterates over all edges and retains those contained in the sorted stack of either of the incident profiles (Lines 10-13).
Similarly, Redefined WNP first iterates over all nodes of the blocking graph to extract their neighborhood and to estimate the
Algorithm 4: Redefined Cardinality Node Pruning.

Input: (i) $G_B^e$ the blocking graph, and (ii) $\omega$ the function defining the local cardinality thresholds.

Output: $G_{\text{out}}^o$ the pruned blocking graph

1. $\text{SortedStacks}[i] \leftarrow \{\}$ // sorted stack per node
2. $\text{foreach} v_i \in V_G \text{ do} // \text{for every node}$
3. $G_v \leftarrow \text{getNeighborhood}(v_i, G_B)$
4. $k \leftarrow \text{cr}(G_v) / \text{get local cardinality threshold}$
5. $\text{foreach} e_{i,j} \in E_G \text{ do} // \text{add every adjacent edge}$
6. $\text{SortedStacks}[i].\text{push}(e_{i,j}) // \text{in sorted stack}$
7. if $k < \text{SortedStacks}[i].\text{size}()$ then
8. $\text{SortedStacks}[i].\text{pop}(); // \text{remove last edge}$
9. $E_{\text{out}}^o \leftarrow \{\} // \text{set of retained edges}$
10. $\text{foreach} e_{i,j} \in E_B \text{ do} // \text{for every edge}$
11. if $e_{i,j} \in \text{SortedStacks}[i]$ then
12. $\text{OR} e_{i,j} \in \text{SortedStacks}[j] \text{ then} // \text{retain if in}$
13. $E_{\text{out}}^o \leftarrow E_{\text{out}}^o \cup \{e_{i,j}\} // \text{top-k for either node}$
14. return $G_{\text{out}}^o = (V_B, E_{\text{out}}^o, WS)$;

Algorithm 5: Redefined Weighted Node Pruning.

Input: (i) $G_B^e$ the blocking graph, and (ii) $\omega$ the function defining the local weight thresholds.

Output: $G_{\text{out}}^o$ the pruned blocking graph

1. $\text{weights}[i] \leftarrow \{\} // \text{thresholds per node}$
2. $\text{foreach} v_i \in V_G \text{ do} // \text{for every node}$
3. $G_v \leftarrow \text{getNeighborhood}(v_i, G_B)$
4. $\text{weight[i]} \leftarrow \omega(G_v) // \text{get local threshold}$
5. $E_{\text{out}}^o \leftarrow \{\} // \text{set of retained edges}$
6. $\text{foreach} e_{i,j} \in E_B \text{ do} // \text{for every edge}$
7. if $\text{weight[i]} \leq \text{weight[i]}, \text{weight}$
8. $\text{OR} \text{weight[j]} \leq \text{weight[i]}, \text{weight then} // \text{retain if}$
9. $E_{\text{out}}^o \leftarrow E_{\text{out}}^o \cup \{e_{i,j}\} // \text{exceeds either threshold}$
10. return $G_{\text{out}}^o = (V_B, E_{\text{out}}^o, WS)$;

Figure 9: (a) The pruned blocking graph produced by applying Reciprocal Pruning to the graph in Figure 5(a), and (b) the restructured blocks.

identical to Redefined CNP and Redefined WNP, respectively. The only difference is that they use conjunctive conditions instead of disjunctive ones: the operator OR in Lines 11-12 and 7-8 of Algorithms 4 and 5, respectively, is replaced by the operator AND. Both algorithms use the same pruning criteria as Redefined methods, while sharing the same average time and space complexities: $O(|E_B|)$ and $O(BPE|V_B|+|B'|)$, respectively.

6. EXPERIMENTAL EVALUATION

We now examine the performance of our techniques through a series of experiments. We present their setup in Section 6.1 and in Section 6.2, we fine-tune Block Filtering, assessing its impact on blocking effectiveness. Its impact on time efficiency is measured in Section 6.3 together with that of Optimized Edge Weighting. Section 6.4 assess the effectiveness of our new pruning algorithms and compares them to state-of-the-art block processing methods.

6.1 Setup

We implemented our approaches in Java 8 and tested them on a desktop machine with Intel i7-3770 (3.40GHz) and 16GB RAM, running Lubuntu 15.04 (kernel version 3.19.0). We repeated all time measurements 10 times and report the average value so as to minimize the effect of external parameters.

Datasets. In our experiments, we use 3 real-world entity collections. They are established benchmarks [15, 21, 24] with significant variations in their size and characteristics. They pertain to Clean-Clean ER, but are used for Dirty ER, as well; we simply merge their clean entity collections into a single one that contains duplicates in itself. In total, we have 6 block collections that lay the ground for a thorough experimental evaluation of our techniques.

To present the technical characteristics of the entity collections, we use the following notation: $|E|$ stands for the number of profiles they contain, $|D(E)|$ for the number of existing duplicates, $|N|$ for the number of distinct attribute names, $|P|$ for the total number of name-value pairs, $|p|$ for the mean number of name-value pairs.
Table 1: Technical characteristics of the (a) original block collections, and (b) the ones restructured by Block Filtering with r=0.80.

(a) Original Block Collections

<table>
<thead>
<tr>
<th>Block Collections</th>
<th>D1C</th>
<th>D2C</th>
<th>D3C</th>
<th>D1D</th>
<th>D2D</th>
<th>D3D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original Block Collections</td>
<td>6.687</td>
<td>40.732</td>
<td>1,239,424</td>
<td>44,097</td>
<td>76,327</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.92·10⁶</td>
<td>8.11·10⁷</td>
<td>4.23·10¹⁰</td>
<td>9.49·10⁹</td>
<td>5.03·10⁹</td>
</tr>
<tr>
<td></td>
<td>BPE</td>
<td>4.65</td>
<td>28.17</td>
<td>17.56</td>
<td>10.67</td>
<td>32.86</td>
</tr>
<tr>
<td></td>
<td>PC(B)</td>
<td>0.994</td>
<td>0.980</td>
<td>0.999</td>
<td>0.997</td>
<td>0.981</td>
</tr>
<tr>
<td></td>
<td>P(Q(B))</td>
<td>1.19·10⁻³</td>
<td>2.76·10⁻⁴</td>
<td>2.11·10⁻⁵</td>
<td>4.46·10⁻⁵</td>
<td>1.12·10⁻⁵</td>
</tr>
<tr>
<td></td>
<td>RR</td>
<td>0.988</td>
<td>0.873</td>
<td>0.984</td>
<td>0.953</td>
<td>0.610</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61,399</td>
<td>50,720</td>
<td>3,331,647</td>
<td>63,855</td>
<td>50,765</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.83·10⁻⁵</td>
<td>6.75·10⁻⁷</td>
<td>3.56·10⁻¹⁰</td>
<td>7.98·10⁻⁸</td>
<td>2.70·10⁻⁸</td>
</tr>
<tr>
<td></td>
<td>OT time(B)</td>
<td>2.1 sec</td>
<td>5.6 sec</td>
<td>4 min</td>
<td>2.2 sec</td>
<td>5.7 sec</td>
</tr>
<tr>
<td></td>
<td>RT time(B)</td>
<td>19 sec</td>
<td>65 min</td>
<td>~350 hrs</td>
<td>13 min</td>
<td>57 min</td>
</tr>
</tbody>
</table>

(b) After Block Filtering

<table>
<thead>
<tr>
<th>Block Collections</th>
<th>D1C</th>
<th>D2C</th>
<th>D3C</th>
<th>D1D</th>
<th>D2D</th>
<th>D3D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>After Block Filtering</td>
<td>6.858</td>
<td>40.078</td>
<td>1,239,066</td>
<td>44,096</td>
<td>76,317</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.98·10⁵</td>
<td>2.77·10¹</td>
<td>1.30·10⁻¹⁰</td>
<td>2.38·10⁻¹³</td>
<td>1.37·10⁻¹³</td>
</tr>
<tr>
<td></td>
<td>BPE</td>
<td>3.63</td>
<td>22.54</td>
<td>14.05</td>
<td>8.54</td>
<td>26.29</td>
</tr>
<tr>
<td></td>
<td>PC(B)</td>
<td>0.990</td>
<td>0.976</td>
<td>0.998</td>
<td>0.994</td>
<td>0.976</td>
</tr>
<tr>
<td></td>
<td>P(Q(B))</td>
<td>3.28·10⁻³</td>
<td>8.06·10⁻⁴</td>
<td>6.86·10⁻⁵</td>
<td>9.62·10⁻⁵</td>
<td>1.62·10⁻⁴</td>
</tr>
<tr>
<td></td>
<td>RR</td>
<td>0.637</td>
<td>0.659</td>
<td>0.693</td>
<td>0.749</td>
<td>0.619</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60,464</td>
<td>50,720</td>
<td>3,331,641</td>
<td>63,855</td>
<td>50,765</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.69·10⁻⁵</td>
<td>2.52·10⁻⁷</td>
<td>1.14·10⁻¹⁰</td>
<td>2.11·10⁻⁸</td>
<td>9.76·10⁻¹⁰</td>
</tr>
<tr>
<td></td>
<td>OT time(B)</td>
<td>2.3 sec</td>
<td>6.4 sec</td>
<td>5 min</td>
<td>2.5 sec</td>
<td>6.5 sec</td>
</tr>
<tr>
<td></td>
<td>RT time(B)</td>
<td>9 sec</td>
<td>24 min</td>
<td>~110 hrs</td>
<td>3 sec</td>
<td>17 min</td>
</tr>
</tbody>
</table>

Figure 10: The effect of Block Filtering’s ratio r on the blocks of D1C and D2D with respect to RR and PC.

**Table 2:** Technical characteristics of the entity collections.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Entity Collections for Clean-Clean ER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1C</td>
<td>2.516</td>
<td>2,308</td>
<td>4</td>
<td>1.01·10⁸</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>D2C</td>
<td>27,615</td>
<td>22,863</td>
<td>4</td>
<td>1.55·10⁷</td>
<td>5.6</td>
<td></td>
</tr>
<tr>
<td>D3C</td>
<td>1,190,733</td>
<td>2,164,040</td>
<td>8</td>
<td>7.16·10¹⁰</td>
<td>35.2</td>
<td></td>
</tr>
<tr>
<td>(b) Entity Collections for Dirty ER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1D</td>
<td>63,869</td>
<td>2,308</td>
<td>4</td>
<td>2.08·10⁻¹⁰</td>
<td>3.3</td>
<td>2.04·10⁻¹⁰</td>
</tr>
<tr>
<td>D2D</td>
<td>50,797</td>
<td>22,863</td>
<td>10</td>
<td>9.71·10⁻⁹</td>
<td>19.1</td>
<td>1.29·10⁻⁹</td>
</tr>
<tr>
<td>D3D</td>
<td>3,354,773</td>
<td>892,579</td>
<td>58,590</td>
<td>5.19·10⁻¹⁹</td>
<td>15.5</td>
<td>5.63·10⁻¹²</td>
</tr>
</tbody>
</table>

Note: We observed that all block collections exhibit nearly perfect recall, as their PC consistently exceeds 0.98. They also convey significant gains in efficiency, executing an order of magnitude less comparisons than the brute-force approach (RR>0.9) in most cases. They reduce the resolution time to a similar extent, due to their very low OT time. Still, their precision is significantly lower than 0.01 in all cases. This means that on average, more than 100 comparisons have to be executed in order to identify a new pair of duplicates. The corresponding blocking graphs vary significantly in size, ranging from tens of thousands edges to tens of billions, whereas their order ranges from few thousands nodes to few millions.

**6.2 Block Collections**

**Original Blocks:** From all datasets, we extracted a redundancy-positive block collection by applying Token Blocking [21]. We also applied Block Purging [21] in order to discard those blocks that contained more than half of the input entity profiles. The technical characteristics of the resulting block collections are presented in Table 1(a). Remember that [V| and [E| stand for the order and the size of the corresponding blocking graph, respectively. RR has been computed with respect to [E| in Table 2, i.e., RR=1−|E||/|E|.

We observe that all block collections exhibit nearly perfect recall, as their PC consistently exceeds 0.98. They also convey significant gains in efficiency, executing an order of magnitude less comparisons than the brute-force approach (RR>0.9) in most cases. They reduce the resolution time to a similar extent, due to their very low OT time. Still, their precision is significantly lower than 0.01 in all cases. This means that on average, more than 100 comparisons have to be executed in order to identify a new pair of duplicates. The corresponding blocking graphs vary significantly in size, ranging from tens of thousands edges to tens of billions, whereas their order ranges from few thousands nodes to few millions.

Note that we experimented with additional redundancy-positive blocking methods, such as Q-grams Blocking. All of them involved a schema-agnostic functionality that tracks effectively the schema heterogeneity. They all produced blocks with similar characteristics as Token Blocking and are omitted for brevity. In general, the outcomes of our experiments are independent of the schema-agnostic, redundancy-positive methods that yield the input blocks.

**Block Filtering:** Before using Block Filtering, we have to fine-tune its filtering ratio r, which determines the portion of the most important blocks that are retained for each profile. To examine its effect, we measured the performance of the restructured blocks using all values of r in [0.05, 1] with a step of 0.05. We consider two evaluation measures: recall (PC) and reduction ratio (RR).

Figure 10 presents the evolution of both measures over the original blocks of D1C and D2D — the other datasets exhibit similar patterns and are omitted for brevity. We observe that there is a clear trade-off between RR and PC: the smaller the value of r, the less blocks are retained for each profile and the lower is the total cardinality of the restructured blocks |B||, thus increasing RR; this reduces the number of detected duplicates, thus decreasing PC. The opposite is true for large values of r. Most importantly, Block Filtering exhibits a robust performance with respect to r, with small
variations in its value leading to small differences in RR and PC.

To use Block Filtering as a pre-processing method, we should set its ratio to a value that increases precision at a low cost in recall. We quantify this constraint by requiring that r decreases PC by less than 0.5%, while maximizing RR and, thus, PQ. The ratio that satisfies this constraint across all datasets is r = 0.80. Table 1(b) presents the characteristics of the restructured block collections corresponding to this configuration. Note that RR has been computed with respect to the cardinality of the original blocks.

We observe that the number of blocks is almost the same as in Table 1(a). Yet, their total cardinality is reduced by 64% to 75%, while recall is reduced by less than 0.5% in most cases. As a result, PQ rises from 2.7 to 4.0 times, but still remains far below 0.01. There is an insignificant increase in OTime and, thus, RTime decreases to the same extent as RR. The same applies to the order of the blocking graph |E|, while its size |V| remains almost the same. Finally, BPE is reduced by (1-r)-100% = 20% across all datasets.

6.3 Time Efficiency Improvements

Table 3 presents the performance of the four existing pruning schemes, averaged across all weighting schemes, before and after Block Filtering.

<table>
<thead>
<tr>
<th>Original Block Collections</th>
<th>After Block Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1C</td>
<td>D2C</td>
</tr>
<tr>
<td>1.43·10^6</td>
<td>7.14·10^5</td>
</tr>
<tr>
<td>0.966</td>
<td>0.860</td>
</tr>
<tr>
<td>0.016</td>
<td>0.028</td>
</tr>
<tr>
<td>395 ms</td>
<td>49 sec</td>
</tr>
</tbody>
</table>

(a) Cardinality Edge Pruning (CEP)

<table>
<thead>
<tr>
<th>Original Block Collections</th>
<th>After Block Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1C</td>
<td>D2C</td>
</tr>
<tr>
<td>2.34·10^6</td>
<td>1.41·10^6</td>
</tr>
<tr>
<td>0.975</td>
<td>0.946</td>
</tr>
<tr>
<td>0.010</td>
<td>0.015</td>
</tr>
<tr>
<td>899 ms</td>
<td>83 sec</td>
</tr>
</tbody>
</table>

(b) Cardinality Node Pruning (CNP)

<table>
<thead>
<tr>
<th>Original Block Collections</th>
<th>After Block Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1C</td>
<td>D2C</td>
</tr>
<tr>
<td>4.32·10^6</td>
<td>1.48·10^6</td>
</tr>
<tr>
<td>0.977</td>
<td>0.963</td>
</tr>
<tr>
<td>1.08·10^-2</td>
<td>2.62·10^-3</td>
</tr>
<tr>
<td>OTD</td>
<td>588 ms</td>
</tr>
</tbody>
</table>

(c) Weighted Edge Pruning (WEP)

<table>
<thead>
<tr>
<th>Original Block Collections</th>
<th>After Block Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1C</td>
<td>D2C</td>
</tr>
<tr>
<td>1.11·10^6</td>
<td>2.81·10^6</td>
</tr>
<tr>
<td>0.987</td>
<td>0.972</td>
</tr>
<tr>
<td>2.32·10^-2</td>
<td>1.14·10^-3</td>
</tr>
<tr>
<td>OTD</td>
<td>862 ms</td>
</tr>
</tbody>
</table>

(d) Weighted Node Pruning (WNP)

Table 3: Performance of the existing pruning schemes, averaged across all weighting schemes, before and after Block Filtering.

For weight thresholds, Block Filtering reduces the number of retained comparisons to a large extent: |B'| drops by 62% to 71% for both WEP and WNP. The reason is that their pruning criteria depends directly on the size and the structure of the blocking graph. At the same time, their recall gets lower by less than 3% in all cases, an affordable reduction that is caused by two factors: (i) Block Filtering discards some matching edges itself, and (ii) Block Filtering reduces the extent of co-occurrence for some matching entity profiles, thus lowering the weight of their edges.
Redefined Node-centric Pruning

| ||B′|| | D1C | D2C | D3C | D1D | D2D | D3D |
|---|---|---|---|---|---|---|---|
| PC(B) | 0.955 | 0.942 | 0.965 | 0.936 | 0.888 | 0.942 |
| PQ(B) | 0.014 | 0.025 | 0.026 | 0.006 | 0.020 | 0.029 |
| OT(B) | 339 ms | 5 sec | 2.1 hrs | 5 sec | 23 sec | 4.9 hrs |

Reciprocal Node-centric Pruning

| ||B′|| | D1C | D2C | D3C | D1D | D2D | D3D |
|---|---|---|---|---|---|---|---|
| PC(B) | 0.880 | 0.886 | 0.912 | 0.847 | 0.650 | 0.868 |
| PQ(B) | 0.312 | 0.084 | 0.142 | 0.017 | 0.057 | 0.111 |
| OT(B) | 329 ms | 5 sec | 2.1 hrs | 5 sec | 23 sec | 4.9 hrs |

<table>
<thead>
<tr>
<th>(a) Redefined Cardinality Node Pruning</th>
<th>(b) Reciprocal Cardinality Node Pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Redefined CNP</td>
<td>1.63·10^6</td>
</tr>
<tr>
<td>Reciprocal CNP</td>
<td>6.54·10^4</td>
</tr>
</tbody>
</table>

| CEP | 117 ms | 4 sec | 1.5 hrs | 3 sec | 14 sec | 1.8 hrs |
| CNP | 246 ms | 6 sec | 2.2 hrs | 6 sec | 25 sec | 4.3 hrs |
| WEP | 150 ms | 6 sec | 2.7 hrs | 5 sec | 25 sec | 3.8 hrs |
| WNP | 257 ms | 8 sec | 4.4 hrs | 7 sec | 33 sec | 7.5 hrs |

Table 5: OTime of Optimized Edge Weighting for each pruning scheme, averaged across all weighting schemes, the datasets in Table 1(b), i.e., after Block Filtering.

We can conclude that Block Filtering enhances the scalability of Meta-blocking to a significant extent, accelerating the processing of all pruning schemes almost by 4 times, on average. It also achieves much higher precision, while its impact on recall is either negligible or beneficial. Thus, it constitutes an indispensable pre-processing step for Meta-blocking. For this reason, the following experiments are carried out on top of Block Filtering.

**Optimized Edge Weighting.** Table 5 presents the overhead time of the four pruning schemes when combined with Block Filtering and Optimized Edge Weighting (cf. Algorithm 3). Comparing it with OTime in the right part of Table 3, we observe significant enhancements in efficiency. Again, they depend on the dataset at hand, rather than the pruning scheme. In fact, the higher the BPE of a dataset after Block Filtering, the larger is the reduction in overhead time: OTime is reduced by 19% for D1C, where BPE takes the lowest value across all datasets (3.63), and by 92% for D3D, where BPE takes the highest one (26.29). The reason is that Optimized Edge Weighting minimizes the computational cost of the process that is applied to every comparison by Original Edge Weighting (cf. Algorithm 2) from O(2·BPE) to O(1).

Also interesting is the comparison between OTime in Table 5 and OTime in the left part of Table 3, i.e., before applying Block Filtering to the input blocks. On average, across all datasets and pruning schemes, OTime is reduced by 87%, which is almost an order of magnitude. Again, the lowest (72%) and the highest (98%) average reductions correspond to D1C and D3C, respectively, which exhibit the minimum and the maximum BPE before Block Filtering. We can conclude, therefore, that the two efficiency optimizations are complementary and indispensable for scalable Meta-blocking.

### 6.4 Precision Improvements

To estimate the performance of Redefined and Reciprocal Node-centric Pruning, we applied the four pruning schemes they yield to the datasets in Table 1(b). Their performance appears in Table 4.

**Cardinality-based Pruning.** Starting with Redefined CNP, we observe that it maintains the recall of the original CNP, while conveying a moderate increase in efficiency. On average, across all weighting schemes and datasets, it retains 18% less comparisons, increasing PQ by 1.2 times. With respect to OTime, there is no clear winner, as the implementation of both algorithms is highly similar, relying on Optimized Edge Weighting. Hence, the original CNP is just 2% faster, on average, because it does not store all retained edges per node in memory.

For Reciprocal CNP, OTime is practically identical with that of Redefined CNP, as they only differ in a single operator. Yet, Reciprocal CNP consistently achieves the highest precision among all versions of CNP at the cost of the lowest recall. On average, it retains 82% and 78% less comparisons than CNP and Redefined CNP, respectively, while increasing precision by 7.9 and 6.9 times, respectively; it also decreases recall by 11%, but exceeds the minimum acceptable PC for efficiency-intensive applications (0.80) to a significant extent in most cases. The only exception is D3D, where PC drops below 0.80 for all weighting schemes. Given that the corresponding Clean-Clean ER dataset (D3C) exhibits much higher PC, the poor recall for D3D is attributed to highly similar (i.e., noisy) entity profiles in one of the duplicate-free entity collections.

It is worth comparing at this point the new pruning schemes with their edge-centric counterpart: CEP coupled with Block Filtering (see right part of Table 3). We observe three patterns: (i) On average, CEP keeps 33% less comparisons than Redefined CNP, but lowers recall by 18%. Its recall is actually so low in most datasets that Redefined CNP achieves higher precision in all cases except D1C and D2C. (ii) Reciprocal CNP typically outperforms CEP in all respects of effectiveness. On average, it executes 67% less comparisons, while increasing recall by 8%. D1C is the exception that proves this rule: Reciprocal CNP saves more than an order of magnitude more comparisons, but CEP achieves slightly higher recall. (iii) The overhead time of CEP is lower by 44%, on average, than both algorithms, because it iterates once instead of twice over all edges in the blocking graph.

On the whole, we can conclude that Reciprocal CNP offers the best choice for efficiency-intensive applications, as it consistently achieves the highest precision among all cardinality-based pruning schemes with PC≥0.8. However, in datasets with high levels of noise, where many entity profiles share the same information, Redefined CNP should be preferred; it is more robust to recall than CEP and Reciprocal CNP, while saving 30% more comparisons than CNP. In any case, Block Filtering is indispensable.

**Weight-based Pruning.** As expected, Redefined WNP maintains the same recall as the original implementation of WNP, while conveying major enhancements in efficiency. On average, across all weighting schemes and datasets, it reduces the retained comparisons by 28% and increases precision by 1.5 times. It is also faster than WNP by 7%, but in practice, the method with the lowest OTime varies across the datasets.

Reciprocal WNP performs a deeper pruning that consistently trims a lower number of executed comparisons for a lower recall. On average, it reduces the total cardinality of WNP by 72% and the recall by 2%. Its mean PC drops slightly below 0.95 in D2C, but this does not apply to all weighting schemes; two of them exceed the minimum acceptable recall of effectiveness-intensive applications even for this dataset. As a result, Reciprocal WNP enhances the precision of WNP by 3.9 times. Its overhead is slightly lower than...
Redefined WNP, because it retains less comparisons in memory. Thus, it is faster than WNP by 9%, on average.

Compared to WEP, Redefined WNP exhibits lower precision, but is more robust to recall, maintaining PC well above 0.95 under all circumstances. In contrast, WEP violates this constraint in all datasets for at least one weighting scheme. Reciprocal WNP scores higher precision than WEP, while being more robust to recall, as well. For this reason, it is the optimal choice for effectiveness-intensive applications. Again, for datasets with high levels of noise, Redefined WNP should be preferred.

Baseline Methods. We now compare our techniques with the state-of-the-art block processing method Iterative Blocking [27] and with Graph-free Meta-blocking (see end of Section 4.1). The functionality of the former was optimized by ordering the blocks from the smallest to the largest cardinality; its functionality was further optimized for Clean-Clean ER by assuming the ideal case where two matching entities are not compared to other co-occurring entities after their detection. The configuration of Graph-free Meta-blocking was also optimized by setting \( r \) to the smallest values in \([0.05, 1.0]\) with a step of 0.05 that ensures a recall higher than 0.80 and 0.95 across all real datasets; this resulted in \( r = 0.25 \) and \( r = 0.55 \) for efficiency- and effectiveness-intensive applications, respectively. Table 6 presents the performance of the two methods.

Juxtaposing Efficiency-intensive Graph-free Meta-blocking and Reciprocal CNP, we notice a clear trade-off between precision and recall. The latter approach emphasizes PC, retaining 85% less comparisons at the cost of 5% lower PC, on average. The only advantage of Graph-free Meta-blocking is its minimal overhead: its lightweight functionality is able to process datasets with millions of entities within few minutes even on commodity hardware. Similar patterns arise when comparing Reciprocal WNP with Effectiveness-intensive Graph-free Meta-blocking: the former executes 58% less comparisons at the cost of a 2% decrease in recall, thus achieving higher precision. This behaviour can be explained by the fine-grained functionality of Reciprocal Pruning: unlike Graph-free Meta-blocking, which operates on the level of individual entities, it considers pairwise comparisons, thus being more accurate in their pruning.

Compared to Iterative Blocking, Reciprocal WNP retains less comparisons by a whole order of magnitude at the cost of slightly lower recall. Thus, it achieves significantly higher precision. The overhead of Iterative Blocking is significantly lower in most cases, but it does not scale well to large datasets, even though Iterative Blocking does not involve a blocking graph. The reason is that it goes through the input block collection several times, updating the representation of the duplicate entities in all blocks that contain them, whenever a new match is detected.

7. CONCLUSIONS

In this paper, we introduced two techniques for boosting the efficiency of Meta-blocking along with two techniques for enhancing its effectiveness. Our thorough experimental analysis verified that in combination, our methods go well beyond the existing Meta-blocking techniques in all respects and simplify its configuration, depending on the data and the application at hand. For efficiency-intensive ER applications, Reciprocal CNP processes a large heterogeneous dataset with 3 millions entities and 80 billion comparisons within 2 hours even on commodity hardware; it also manages to retain recall and precision above 0.8 and 0.1, respectively. For effectiveness-intensive ER applications, Reciprocal WNP processes the same voluminous dataset within 5 hours on commodity hardware, while retaining recall > 0.95 and precision close to 0.01. In both cases, Block Filtering and Optimized Edge Weighting are indispensable. In the future, we plan to adapt our techniques for Enhanced Meta-blocking to Incremental Entity Resolution.

References