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

George Papadakis[$], George Papastefanatos[#], Themis Palpanas[◇], Manolis Koubarakis[$]

[◇] Paris Descartes University, France themis@mi.parisdescartes.fr
[#]IMIS, Research Center "Athena", Greece gpapas@imis.athena-innovation.gr
[$]Dep. of Informatics & Telecommunications, Uni. Athens, Greece {gpapadis, koubarak}@di.uoa.gr

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 to 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-Stats*[1] 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 Base[2] 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].

[1]`http://stats.lod2.eu`
[2]`http://www.google.com/base`

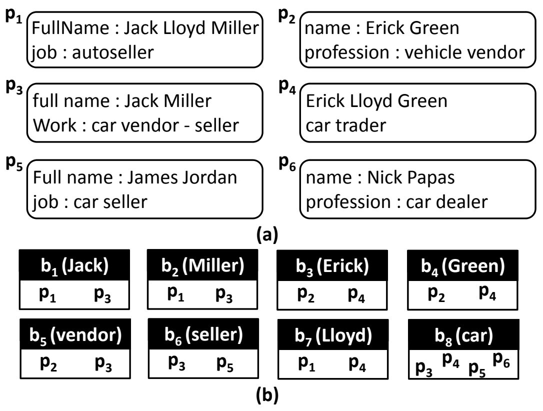

Figure 1: (a) A set of entity profiles, and (b) the corresponding blocks produced by Token Blocking.

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_4 contain one redundant comparison each, which are repeated in b_1 and b_3, respectively; all other blocks entail superfluous comparisons between non-matching entity profiles, except for the redundant comparison p_3-p_5 in b_8 (it is repeated in b_6). 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-

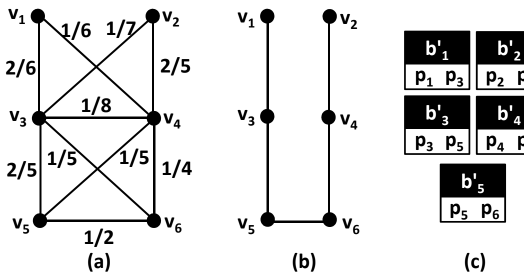

Figure 2: (a) A blocking graph extracted from the blocks in Figure 1(b), (b) one of the possible edge-centric pruned blocking graphs, and (c) the new blocks derived from it.

sion and recall [22, 23], and is the focus of this work.

Meta-blocking restructures a block collection B into a new one B' that contains a significantly lower number of unnecessary comparisons, while detecting almost the same number of duplicates. It operates in two steps [7, 22, 23]: first, it transforms B into the blocking graph G_B, which contains a vertex v_i for every entity profile p_i in B, and an edge $e_{i,j}$ for every pair of *co-occurring profiles* p_i and p_j (i.e., entity profiles sharing at least one block). Figure 2(a) depicts the graph for the blocks in Figure 1(b). As no parallel edges are constructed, every pair of entities is compared at most once, thus eliminating all *redundant* comparisons.

Second, it annotates every edge with a weight analogous to the likelihood that the incident entities are matching. For instance, the edges in Figure 2(a) are weighted with the Jaccard similarity of the lists of blocks associated with their incident entity profiles. The lower the weight of an edge, the more likely it is to connect non-matching entities. Therefore, Meta-blocking discards most *superfluous* comparisons by pruning the edges with low weights. A possible approach is to discard all edges with a weight lower than the overall mean weight (1/4). This yields the pruned graph in Figure 2(b). The restructured block collection B' is formed by creating a new block for every retained edge – as depicted in Figure 2(c). Note that B' maintains the original recall, while reducing the comparisons from 13 to just 5.

Open issues. Despite the significant enhancements in efficiency, Meta-blocking suffers from two drawbacks:

(i) There is plenty of room for raising its precision, especially for the configurations that are more robust to recall. The reason is that they retain a considerable portion of redundant and superfluous comparisons. This is illustrated in our example, where the restructured blocks of Figure 2(c) contain 3 superfluous comparisons in b'_3, b'_4 and b'_5.

(ii) The processing of voluminous datasets involves a significant overhead. The corresponding blocking graphs comprise millions of nodes that are strongly connected with billions of edges. Inevitably, the pruning of such graphs is very time-consuming; for example, a graph with 3.3 million nodes and 35.8 billion edges requires 16 hours, on average, on commodity hardware (see Section 6.3).

Proposed Solution. In this paper, we describe novel techniques for overcoming both weaknesses identified above.

First, we speed up Meta-blocking in two ways:

(i) 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.[3]

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-

[3]See `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 Purging [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-p_j in block b_k 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* ($p_i \equiv p_j$) 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 $c_{i,j}$. 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) 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) corresponds to recall, assessing the portion of existing duplicates that can be detected in B. More formally, $PC = |D(B)|/|D(E)|$. 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 ($OTime$) measures the time required for extracting B either from the input entity collection E or from another block collection B'.

(ii) Resolution Time ($RTime$) is equal to $OTime$ 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	Pruning Algorithms
1) Aggregate Reciprocal Comparisons (ARCS)	1) Cardinality Edge Pruning (CEP)
2) Common Blocks (CBS)	2) Cardinality Node Pruning (CNP)
3) Enhanced Common Blocks (ECBS)	3) Weighted Edge Pruning (WEP)
4) Jaccard Similarity (JS)	4) Weighted Node Pruning (WNP)
5) Enhanced Jaccard Similarity (EJS)	

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

Aggregate Reciprocal Comparisons Scheme	$ARCS(p_i, p_j, B) = \sum_{b_k \in B_{ij}} \frac{1}{\|\|b_k\|\|}$
Common Blocks Scheme	$CBS(p_i, p_j, B) = \|B_{ij}\|$
Enhanced Common Blocks Scheme	$ECBS(p_i, p_j, B) = CBS(p_i, p_j, B) \cdot \log \frac{\|B\|}{\|B_i\|} \cdot \log \frac{\|B\|}{\|B_j\|}$
Jaccard Scheme	$JS(p_i, p_j, B) = \frac{\|B_{ij}\|}{\|B_i\| + \|B_j\| - \|B_{ij}\|}$
Enhanced Jaccard Scheme	$EJS(p_i, p_j, B) = JS(p_i, p_j, B) \cdot \log \frac{\|V_B\|}{\|v_i\|} \cdot \log \frac{\|V_B\|}{\|v_j\|}$

Figure 4: The formal definition of the five weighting schemes. $B_i \subseteq B$ denotes the set of blocks containing p_i, $B_{i,j} \subseteq B$ 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.

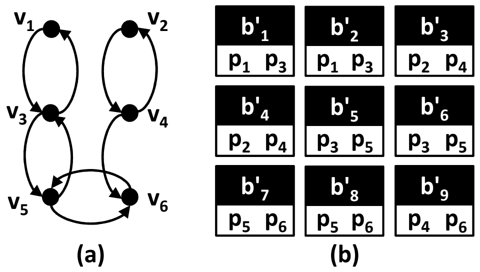

Figure 5: (a) One of the possible node-centric pruned blocking graphs for the graph in Figure 2(a). (b) The new blocks derived from the pruned graph.

Every combination of a pruning algorithm with a pruning criterion is called *pruning 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 $K = \lfloor \sum_{b \in B} |b|/2 \rfloor$.

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

(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) combines 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 crowdsourcing 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).

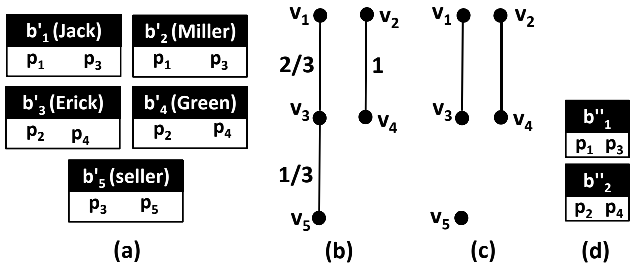

Figure 6: (a) The block collection produced by applying Block Filtering to the blocks in Figure 1(b), (b) the corresponding blocking graph, (c) the pruned blocking graph produced by WEP, and (d) the corresponding restructured block collection.

4. TIME EFFICIENCY IMPROVEMENTS

We now propose two methods for accelerating the processing of Meta-blocking, minimizing its *OTime*: *(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 restructures a block collection by removing profiles from blocks, in which their presence is not necessary. The *importance* of a block b_k for an individual entity profile $p_i \in b_k$ 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_8 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'_2 and another one in b'_4, and 1 superfluous in block b'_5. 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., $O(|B|\cdot\log|B|)$. 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  counter[] ← {}; // count blocks per profile
3  orderBlocks(B); // sort in descending importance
4  maxBlocks[] ← getThresholds(B); // limit per profile
5  foreach $b_k \in B$ do        // check all blocks
6      foreach $p_i \in b_k$ do // check all profiles
7          if counter[i] > maxBlocks[i] then
8              $b_k \leftarrow b_k \setminus p_i$; // remove profile
9          else
10             counter[i]++; // increment counter
11     if $|b_k| > 1$ then       // retain blocks with
12         $B' \leftarrow B' \cup b_k$; // at least 2 profiles
13 return B';
```

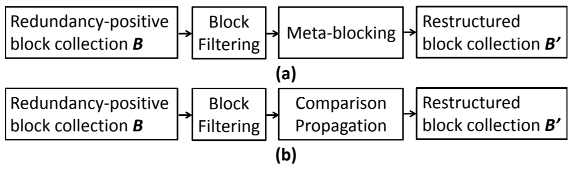
Redundancy-positive block collection B
Block Filtering
Meta-blocking
Restructured block collection B'
(a)
Redundancy-positive block collection B
Block Filtering
Comparison Propagation
Restructured block collection B'
(b)

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  $EI \leftarrow$ buildEntityIndex( $B$ );
3  foreach $b_k \in B$ do        // check all blocks
4      foreach $c_{i,j} \in b_k$ do // check all comparisons
5          $B_i \leftarrow EI$.getBlockList ( $p_i$ );
6          $B_j \leftarrow EI$.getBlockList ( $p_j$ );
7          $commonBlocks \leftarrow 0$;
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 $commonBlocks = 0$ then // $1^{st}$ common id
13                     if $m \neq k$ then     // it violates LeCoBI
14                         break to next comparison;
15                 $commonBlocks$++;
16         $w_{i,j} \leftarrow$ calculateWeight ( $commonBlocks$, $B_i$, $B_j$ );
17         $W \leftarrow W \cup \{w_{i,j}\}$;
18 return $W$;
```

Algorithm 3: Optimized Edge Weighting.

```
Input: $B$ the input block collection, $E$ the input entity collection
Output: $W$ the set of edge weights
1  $W \leftarrow \{\}$; $commonBlocks[] \leftarrow \{\}$; $flags[] \leftarrow \{\}$;
2  $EI \leftarrow$ buildEntityIndex( $B$ );
3  foreach $p_i \in E$ do // check all profiles
4      $B_i \leftarrow EI$.getBlockList ( $p_i$ );
5      $neighbors \leftarrow \{\}$;  // set of co-occurring profiles
6      foreach $b_k \in B_i$ do // check all associated blocks
7          foreach $p_j(\neq p_i) \in b_k$ do // co-occurring profile
8              if $flags[j] \neq i$ then
9                  $flags$[j] = $i$;
10                 $commonBlocks$[j] = 0;
11                 $neighbors \leftarrow neighbors \cup \{p_j\}$;
12             $commonBlocks$[j]++;
13     foreach $p_j \in neighbors$ do
14         $B_j \leftarrow EI$.getBlockList ( $p_j$ );
15         $w_{i,j} \leftarrow$ calculateWeight ( $commonBlocks$[j], $B_i$, $B_j$ );
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 $e_{i,j}$ in the blocking graph G_B corresponds to a non-redundant comparison $c_{i,j}$ in the block collection B. In other words, a comparison $c_{i,j}$ in $b_k \in B$ defines an edge $e_{i,j}$ 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_i 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_k, 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-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 $\lfloor(1-t)\cdot|B_i|\rfloor$+1 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 $e_{i,j}$ 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-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-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, $commonBlocks[j]$ 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 $flags$ helps us to avoid reallocating memory for $commonBlocks$ in every iteration, a procedure that would be costly, due to its size, $|E|$ [16]; $neighbors$ 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 $O(2\cdot BPE\cdot||B||)$, where $BPE(B)=\sum_{b\in B}|b|/|E|$ is the average number of blocks associated with every profile in B; $2\cdot BPE$ 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 $2 \cdot r \cdot BPE$ by removing every profile from $(1-r) \cdot 100\%$ of its associated blocks, where r is the filtering ratio.[4]

The computational cost of Algorithm 3 is determined by two procedures that yield an average time complexity of $O(||B|| + |\bar{v}| \cdot |E|)$:

(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_i) for $|b|$ times – once for each entity profile. Therefore, the process in Lines 8-12 is repeated $|b| \cdot (|b|-1) = ||b||$ 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 $|\bar{v}|$, i.e., the average number of neighbors per profile. It is repeated for every profile and, thus, its overall cost is $O(|\bar{v}| \cdot |E|)$.

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 $O(|\bar{v}| \cdot |E|)$. 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 infers 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 $e_{5,6}$ in Figure 2(a) has a higher weight than both matching edges $e_{1,3}$ and $e_{2,4}$.

(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 entity collections retains its connections with several incident nodes, even though only one of them is matching with it.

[4]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.

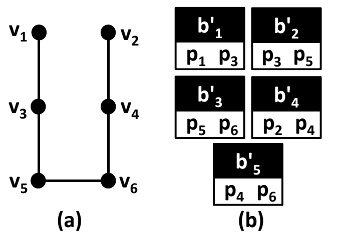

Figure 8: (a) The undirected pruned blocking graph corresponding to the directed one Figure 5(a), and (b) the corresponding block collection.

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'_4.

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 LeCoBI 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) $G_B^{in}$ the blocking graph, and
       (ii) $ct$ the function defining the local cardinality thresholds.
Output: $G_B^{out}$ the pruned blocking graph
1  $SortedStacks[] \leftarrow \{\}$; // sorted stack per node
2  foreach $v_i \in V_B$ do // for every node
3      $G_{v_i} \leftarrow$ getNeighborhood( $v_i, G_B$ );
4      $k \leftarrow ct( G_{v_i} )$; // get local cardinality threshold
5      foreach $e_{i,j} \in E_{v_i}$ do // add every adjacent edge
6          $SortedStacks[i]$.push( $e_{i,j}$ ); // in sorted stack
7          if $k < SortedStacks[i].size()$ then
8              $SortedStacks[i]$.pop(); // remove last edge
9  $E_B^{out} \leftarrow \{\}$; // the set of retained edges
10 foreach $e_{i,j} \in E_B$ do // for every edge
11     if $e_{i,j} \in SortedStacks[i]$
12     OR $e_{i,j} \in SortedStacks[j]$ then // retain if in
13         $E_B^{out} \leftarrow E_B^{out} \cup \{ e_{i,j} \}$; // top-k for either node
14 return $G_B^{out} = \{V_B, E_B^{out}, WS\}$;
```

Algorithm 5: Redefined Weighted Node Pruning.

```
Input: (i) $G_B^{in}$ the blocking graph, and
       (ii) $wt$ the function defining the local weight thresholds.
Output: $G_B^{out}$ the pruned blocking graph
1  weights[] $\leftarrow \{\}$; // thresholds per node
2  foreach $v_i \in V_B$ do // for every node
3      $G_{v_i} \leftarrow$ getNeighborhood( $v_i, G_B$ );
4      $weights[i] \leftarrow wt( G_{v_i} )$; // get local threshold
5  $E_B^{out} \leftarrow \{\}$; // the set of retained edges
6  foreach $e_{i,j} \in E_B$ do // for every edge
7      if $weights[i] \leq e_{i,j}.weight$
8      OR $weights[j] \leq e_{i,j}.weight$ then // retain if it
9          $E_B^{out} \leftarrow E_B^{out} \cup \{ e_{i,j} \}$; // exceeds either threshold
10 return $G_B^{out} = \{V_B, E_B^{out}, WS\}$;
```

corresponding weight threshold (Lines 2-4 in Algorithm 5). Then, it iterates once over all edges and retains those exceeding the weight thresholds of either of the incident nodes (Lines 6-9).

For both algorithms, the function *getNeighborhood* in Line 3 implements the Lines 4-16 of Algorithm 3. Note also that both algorithms use the same configuration as their original implementations: $k = \lfloor \sum_{b \in B} |b|/|E| - 1 \rfloor$ for Redefined CNP and the average weight of each node neighborhood for Redefined WNP. Their time complexity is $O(|V_B| \cdot |E_B|)$ in the worst-case of a complete blocking graph, and $O(|E_B|)$ in the case of a sparse graph, which typically appears in practice. Their space complexity is dominated by the requirements of Entity Index and the number of retained comparisons, i.e., $O(BPE \cdot |V_B| + ||B'||)$, on average.

5.2 Reciprocal Node-centric Pruning

This approach treats the redundant comparisons retained by CNP and WNP as strong indications for profile pairs with high chances of matching. As explained above, these comparisons correspond to reciprocal links in the blocking graph. For example, the edges $e_{\vec{1,3}}$ and $e_{\vec{3,1}}$ in Figure 5(a) indicate that p_1 is highly likely to match with p_3 and vice versa, thus reinforcing the likelihood that the two profiles are duplicates. Based on this rationale, Reciprocal Pruning retains one comparison for every pair of profiles that are reciprocally connected in the directed pruned blocking graph of the original node-centric algorithms; profiles that are connected with a single edge, are not compared in the restructured block collection.

In our example, Reciprocal Pruning converts the directed pruned blocking graph in Figure 5(a) into the undirected pruned blocking graph in Figure 9(a). The corresponding restructured blocks in Figure 9(b) contain just 4 comparisons, one less than the blocks in Figure 8(b). Compared to the blocks in Figure 5(b), the overall efficiency is significantly enhanced at no cost in recall.

In general, Reciprocal Pruning yields restructured blocks with no redundant comparisons and less superfluous ones than both the original and the redefined node-centric pruning. In the worst case, all pairs of nodes are reciprocally linked and Reciprocal Pruning coincides with Redefined Node-centric Pruning. In all other cases, its precision is much higher, while its impact on recall depends on the strength of co-occurrence patterns in the blocking graph.

Reciprocal Pruning yields two new node-centric algorithms: Reciprocal CNP and Reciprocal WNP. Their functionality is almost

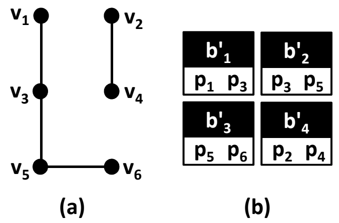

Figure 9: (a) The pruned blocking graph produced by applying Reciprocal Pruning to the graph in Figure 5(a), and (b) the restructured blocks.

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 $O(BPE \cdot |V_B| + ||B'||)$, 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, $|\bar{p}|$ for the mean number of name-value pairs

	Original Block Collections						After Block Filtering					
	D_{1C}	**D_{2C}**	**D_{3C}**	**D_{1D}**	**D_{2D}**	**D_{3D}**	**D_{1C}**	**D_{2C}**	**D_{3C}**	**D_{1D}**	**D_{2D}**	**D_{3D}**
$\lvert B\rvert$	6,877	40,732	1,239,424	44,097	76,327	1,499,534	6,838	40,708	1,239,066	44,096	76,317	1,499,267
$\lvert\lvert B\rvert\rvert$	$1.92\cdot10^6$	$8.11\cdot10^7$	$4.23\cdot10^{10}$	$9.49\cdot10^7$	$5.03\cdot10^8$	$8.00\cdot10^{10}$	$6.98\cdot10^5$	$2.77\cdot10^7$	$1.30\cdot10^{10}$	$2.38\cdot10^7$	$1.37\cdot10^8$	$2.31\cdot10^{10}$
BPE	4.65	28.17	17.56	10.67	32.86	14.79	3.63	22.54	14.05	8.54	26.29	11.83
$PC(B)$	0.994	0.980	0.999	0.997	0.981	0.999	0.990	0.976	0.998	0.994	0.976	0.997
$PQ(B)$	$1.19\cdot10^{-3}$	$2.76\cdot10^{-4}$	$2.11\cdot10^{-5}$	$2.43\cdot10^{-5}$	$4.46\cdot10^{-5}$	$1.12\cdot10^{-5}$	$3.28\cdot10^{-3}$	$8.06\cdot10^{-4}$	$6.86\cdot10^{-5}$	$9.62\cdot10^{-5}$	$1.62\cdot10^{-4}$	$3.86\cdot10^{-5}$
RR	0.988	0.873	0.984	0.953	0.610	0.986	0.637	0.659	0.693	0.749	0.727	0.711
$\lvert V_B\rvert$	61,399	50,720	3,331,647	63,855	50,765	3,333,356	60,464	50,720	3,331,641	63,855	50,765	3,333,355
$\lvert E_B\rvert$	$1.83\cdot10^6$	$6.75\cdot10^7$	$3.58\cdot10^{10}$	$7.98\cdot10^7$	$2.70\cdot10^8$	$6.65\cdot10^{10}$	$6.69\cdot10^5$	$2.52\cdot10^7$	$1.14\cdot10^{10}$	$2.11\cdot10^7$	$9.76\cdot10^7$	$2.00\cdot10^{10}$
$OTime(B)$	2.1 sec	5.6 sec	4 min	2.2 sec	5.7 sec	5 min	2.3 sec	6.4 sec	5 min	2.5 sec	6.5 sec	6 min
$RTime(B)$	19 sec	65 min	~350 hrs	13 min	574 min	~660 hrs	9 sec	24 min	~110 hrs	3 min	174 min	~190 hrs

(a) (b)

Table 1: Technical characteristics of (a) the original block collections, and (b) the ones restructured by Block Filtering with r=0.80.

	|E|	**|D(E)|**	**|N|**	**|P|**	**$\lvert\bar{p}\rvert$**	**||E||**	**RT(E)**
D_{1C}	2,516	2,308	4	$1.01\cdot10^4$	4.0	$1.54\cdot10^8$	26 min
	61,353		4	$1.98\cdot10^5$	3.2		
D_{2C}	27,615	22,863	4	$1.55\cdot10^5$	5.6	$6.40\cdot10^8$	533 min
	23,182		7	$8.16\cdot10^5$	35.2		
D_{3C}	1,190,733	892,579	30,688	$1.69\cdot10^7$	14.2	$2.58\cdot10^{12}$	~21,000 hrs
	2,164,040		52,489	$3.50\cdot10^7$	16.2		

(a) Entity Collections for Clean-Clean ER

D_{1D}	63,869	2,308	4	$2.08\cdot10^5$	3.3	$2.04\cdot10^9$	272 min
D_{2D}	50,797	22,863	10	$9.71\cdot10^5$	19.1	$1.29\cdot10^9$	1,505 min
D_{3D}	3,354,773	892,579	58,590	$5.19\cdot10^7$	15.5	$5.63\cdot10^{12}$	~47,000 hrs

(b) Entity Collections for Dirty ER

Table 2: Technical characteristics of the entity collections. For D_{3C} and D_{3D}, $RT(E)$ was estimated from the average time required for comparing two of its entity profiles: 0.03 msec.

per profile, $||E||$ for the number of comparisons executed by the brute-force approach and $RT(E)$ for its resolution time; in line with $RTime(B)$, $RT(E)$ is computed using the Jaccard similarity of all tokens in the values of two profiles as the entity matching method.

Tables 2(a) and (b) present the technical characteristics of the real entity collections for Clean-Clean and Dirty ER, respectively. D_{1C} contains bibliographic data from DBLP (`www.dblp.org`) and Google Scholar (`http://scholar.google.gr`) that were matched manually [15, 24]. D_{2C} matches movies from IMDB (`imdb.com`) and DBPedia (`http://dbpedia.org`) based on their ImdbId [22, 23]. D_{3C} involves profiles from two snapshots of English Wikipedia (`http://en.wikipedia.org`) Infoboxes, which were automatically matched based on their URL [22, 23]. For Dirty ER, the datasets D_{xD} with $x \in [1, 6]$ were derived by merging the profiles of the individually clean collections that make up D_{xC}, as explained above.

Measures. To assess the *effectiveness* of a restructured block collection B', we use four established measures [4, 5, 22]: *(i)* its cardinality $||B'||$, i.e., total number of comparisons, *(ii)* its recall $PC(B')$, *(iii)* its precision $PQ(B')$, and *(iv)* its *Reduction Ratio* (RR), which expresses the relative decrease in its cardinality in comparison with the original block collection, B: $RR(B, B') = 1 - ||B'||/||B||$. The last three measures take values in the interval [0, 1], with higher values, indicating better effectiveness; the opposite is true for $||B'||$, as effectiveness is inversely proportional to its value (cf. Section 3).

To assess the *time efficiency* of a restructured block collection B', we use the two measures that were defined in Section 3: *(i)* its *Overhead Time* $OTime(B')$, which measures the time required by Meta-blocking to derive it from the input block collection, and *(ii)* its *Resolution Time* $RTime(B')$, which adds to $OTime(B')$ the time taken to apply an entity matching method to all comparisons in B'.

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

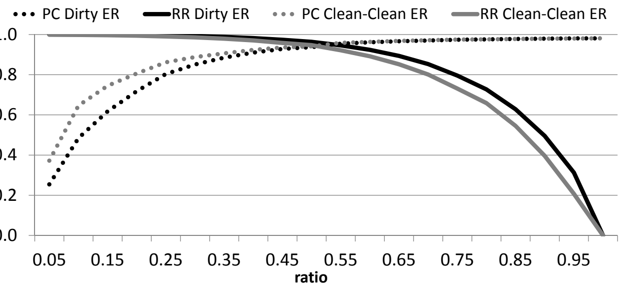

Figure 10: The effect of Block Filtering's ratio r on the blocks of D_{5C} and D_{5D} with respect to RR and PC.

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 $|V_B|$ and $|E_B|$ 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-$||B||/||E||$.

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 $OTime$. 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 tackles 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 D_{5C} and D_{5D} – 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

	Original Block Collections						After Block Filtering					
	D_{1C}	D_{2C}	D_{3C}	D_{1D}	D_{2D}	D_{3D}	D_{1C}	D_{2C}	D_{3C}	D_{1D}	D_{2D}	D_{3D}
$\|\|B'\|\|$	$1.43 \cdot 10^5$	$7.14 \cdot 10^5$	$2.63 \cdot 10^7$	$3.41 \cdot 10^5$	$8.34 \cdot 10^5$	$2.47 \cdot 10^7$	$1.10 \cdot 10^5$	$5.72 \cdot 10^5$	$2.11 \cdot 10^7$	$2.73 \cdot 10^5$	$6.67 \cdot 10^5$	$1.97 \cdot 10^7$
$PC(B')$	0.966	0.860	0.724	0.765	0.522	0.535	0.948	0.871	0.748	0.823	0.641	0.566
$PQ(B')$	0.016	0.028	0.025	0.005	0.014	0.019	0.020	0.035	0.032	0.007	0.022	0.026
$OT(B')$	395 ms	49 sec	9.4 hrs	22 sec	16 min	17.2 hrs	142 ms	12 sec	2.0 hrs	5 sec	4 min	3.7 hrs

(a) Cardinality Edge Pruning (CEP)

	D_{1C}	D_{2C}	D_{3C}	D_{1D}	D_{2D}	D_{3D}	D_{1C}	D_{2C}	D_{3C}	D_{1D}	D_{2D}	D_{3D}
$\|\|B'\|\|$	$2.34 \cdot 10^5$	$1.41 \cdot 10^6$	$4.95 \cdot 10^7$	$6.38 \cdot 10^5$	$1.62 \cdot 10^6$	$4.63 \cdot 10^7$	$1.69 \cdot 10^5$	$1.10 \cdot 10^6$	$3.95 \cdot 10^7$	$5.10 \cdot 10^5$	$1.31 \cdot 10^6$	$3.63 \cdot 10^7$
$PC(B')$	0.975	0.946	0.973	0.949	0.880	0.951	0.955	0.942	0.965	0.936	0.888	0.942
$PQ(B')$	0.010	0.015	0.018	0.003	0.012	0.018	0.013	0.020	0.022	0.004	0.016	0.023
$OT(B')$	899 ms	83 sec	18.6 hrs	58 sec	17 min	35.2 hrs	310 ms	25 sec	4.7 hrs	13 sec	6 min	8.6 hrs

(b) Cardinality Node Pruning (CNP)

	D_{1C}	D_{2C}	D_{3C}	D_{1D}	D_{2D}	D_{3D}	D_{1C}	D_{2C}	D_{3C}	D_{1D}	D_{2D}	D_{3D}
$\|\|B'\|\|$	$4.32 \cdot 10^5$	$1.48 \cdot 10^7$	$6.64 \cdot 10^9$	$1.38 \cdot 10^7$	$7.81 \cdot 10^7$	$1.19 \cdot 10^{10}$	$1.63 \cdot 10^5$	$5.50 \cdot 10^6$	$2.11 \cdot 10^9$	$3.99 \cdot 10^6$	$2.26 \cdot 10^7$	$4.06 \cdot 10^9$
$PC(B')$	0.977	0.963	0.977	0.987	0.970	0.973	0.953	0.947	0.967	0.964	0.944	0.965
$PQ(B')$	$1.08 \cdot 10^{-2}$	$2.62 \cdot 10^{-3}$	$6.66 \cdot 10^{-4}$	$2.99 \cdot 10^{-4}$	$7.30 \cdot 10^{-4}$	$3.54 \cdot 10^{-4}$	$2.92 \cdot 10^{-2}$	$6.54 \cdot 10^{-3}$	$1.49 \cdot 10^{-3}$	$9.51 \cdot 10^{-4}$	$1.62 \cdot 10^{-3}$	$7.27 \cdot 10^{-4}$
$OT(B')$	588 ms	92 sec	17.1 hrs	40 sec	26 min	31.7 hrs	193 ms	23 sec	3.7 hrs	8 sec	7 min	6.9 hrs

(c) Weighted Edge Pruning (WEP)

	D_{1C}	D_{2C}	D_{3C}	D_{1D}	D_{2D}	D_{3D}	D_{1C}	D_{2C}	D_{3C}	D_{1D}	D_{2D}	D_{3D}
$\|\|B'\|\|$	$1.11 \cdot 10^6$	$2.81 \cdot 10^7$	$1.60 \cdot 10^{10}$	$3.41 \cdot 10^7$	$1.54 \cdot 10^8$	$3.00 \cdot 10^{10}$	$4.64 \cdot 10^5$	$1.05 \cdot 10^7$	$5.38 \cdot 10^9$	$9.84 \cdot 10^6$	$5.29 \cdot 10^7$	$9.49 \cdot 10^9$
$PC(B')$	0.988	0.972	0.997	0.993	0.971	0.995	0.979	0.964	0.997	0.979	0.959	0.992
$PQ(B')$	$2.32 \cdot 10^{-3}$	$1.14 \cdot 10^{-3}$	$1.44 \cdot 10^{-4}$	$1.13 \cdot 10^{-4}$	$3.11 \cdot 10^{-4}$	$7.63 \cdot 10^{-5}$	$5.13 \cdot 10^{-3}$	$3.01 \cdot 10^{-3}$	$3.56 \cdot 10^{-4}$	$3.42 \cdot 10^{-4}$	$6.79 \cdot 10^{-4}$	$1.94 \cdot 10^{-4}$
$OT(B')$	862 ms	85 sec	18.6 hrs	55 sec	16 min	35.0 hrs	303 ms	24 sec	4.7 hrs	13 sec	5 min	9.0 hrs

(d) Weighted Node Pruning (WNP)

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

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 $|E_B|$, while its size $|V_B|$ 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. Their original performance appears in the left part, while the right part presents their performance after Block Filtering.

Original Performance. We observe that CEP reduces the executed comparisons by 1 to 3 orders of magnitude for the smallest and the largest datasets, respectively. It increases precision (*PQ*) to a similar extent at the cost of much lower recall in most of the cases. This applies particularly to Dirty ER, where *PC* drops consistently below 0.80, the minimum acceptable recall of efficiency-intensive ER applications. The reason is that Dirty ER is more difficult than Clean-Clean ER, involving much larger blocking graphs with many more noisy edges between non-matching entity profiles.

CNP is more robust to recall than CEP, as its *PC* lies well over 0.80 across all datasets. Its robustness stems from its node-centric functionality, which retains the best edges per node, instead of the globally best ones. This comes, though, at the cost of a much higher computational cost: its overhead time is larger than that of CEP by 44%, on average. Further, CNP retains almost twice as many comparisons and yields a slightly lower precision than CEP.

For WEP, we observe that its recall consistently exceeds 0.95, the minimum acceptable *PC* of effectiveness-intensive ER applications. At the same time, it executes almost an order of magnitude less comparisons than the original blocks in Table 1(a) and enhances *PQ* to a similar extent. These patterns apply to all datasets.

Finally, WNP saves 60% of the brute-force comparisons, on average, retaining twice as many comparisons as WEP. Its recall remains well over 0.95 across all datasets, exceeding that of WEP to a minor extent. As a result, its precision is half that of WEP, while its overhead time is slightly higher.

In summary, these experiments verify previous findings about the relative performance of pruning schemes [22]: the cardinality-based ones excel in precision, being more suitable for efficiency-intensive ER applications, while the weight-based schemes excel in recall, being more suitable for effectiveness-intensive applications. In both cases, the node-centric algorithms trade higher recall for lower precision and higher overhead.

Block Filtering. Examining the effect of Block Filtering in the right part of Table 3, we observe two patterns:

(i) Its impact on overhead time depends on the dataset at hand, rather than the pruning scheme applied on top of it. In fact, *OTime* is reduced by 65% (D_{1C}) to 78% (D_{1D}), on average, across all pruning schemes. This is close to *RR* and the reduction in the order of the blocking graph $|E_B|$, but higher than them, because Block Filtering additionally reduces *BPE* by 20%.

(ii) Its impact on blocking effectiveness depends on the type of the pruning criterion used by Meta-blocking. For cardinality thresholds, Block Filtering conveys a moderate decrease in the retained comparisons, with $||B'||$ dropping by 20%, on average. The reason is that both CEP and CNP use thresholds that are proportional to *BPE*, which is reduced by (1-r)·100%. At the same time, their recall is either reduced to a minor extent (<2%), or increases by up to 10%. The latter case appears in half the datasets and indicates that Block Filtering cleans the blocking graph from noisy edges, enabling CEP and CNP to identify more duplicates with fewer retained comparisons.

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						Reciprocal Node-centric Pruning					
	D_{1C}	**D_{2C}**	**D_{3C}**	**D_{1D}**	**D_{2D}**	**D_{3D}**	**D_{1C}**	**D_{2C}**	**D_{3C}**	**D_{1D}**	**D_{2D}**	**D_{3D}**
$\|\|B'\|\|$	$1.63 \cdot 10^5$	$8.52 \cdot 10^5$	$3.36 \cdot 10^7$	$3.91 \cdot 10^5$	$1.02 \cdot 10^6$	$2.92 \cdot 10^7$	$6.54 \cdot 10^3$	$2.50 \cdot 10^5$	$5.88 \cdot 10^6$	$1.19 \cdot 10^5$	$2.86 \cdot 10^5$	$7.12 \cdot 10^6$
$PC(B')$	0.955	0.942	0.965	0.936	0.888	0.942	0.880	0.886	0.912	0.847	0.650	0.868
$PQ(B')$	0.014	0.025	0.026	0.006	0.020	0.029	0.312	0.084	0.142	0.017	0.057	0.111
$OT(B')$	339 ms	5 sec	2.1 hrs	5 sec	23 sec	4.9 hrs	329 ms	5 sec	2.1 hrs	5 sec	23 sec	4.9 hrs

(a) Redefined Cardinality Node Pruning **(b) Reciprocal Cardinality Node Pruning**

	D_{1C}	D_{2C}	D_{3C}	D_{1D}	D_{2D}	D_{3D}	D_{1C}	D_{2C}	D_{3C}	D_{1D}	D_{2D}	D_{3D}
$\|\|B'\|\|$	$3.72 \cdot 10^5$	$7.52 \cdot 10^6$	$3.96 \cdot 10^9$	$7.23 \cdot 10^6$	$3.26 \cdot 10^7$	$6.96 \cdot 10^9$	$9.30 \cdot 10^4$	$2.96 \cdot 10^6$	$1.41 \cdot 10^9$	$2.61 \cdot 10^6$	$2.02 \cdot 10^7$	$2.54 \cdot 10^9$
$PC(B')$	0.979	0.964	0.994	0.979	0.959	0.992	0.953	0.949	0.977	0.964	0.924	0.971
$PQ(B')$	$6.53 \cdot 10^{-3}$	$4.79 \cdot 10^{-3}$	$5.01 \cdot 10^{-4}$	$4.91 \cdot 10^{-4}$	$1.09 \cdot 10^{-3}$	$2.74 \cdot 10^{-4}$	$3.16 \cdot 10^{-2}$	$8.78 \cdot 10^{-3}$	$1.47 \cdot 10^{-3}$	$1.14 \cdot 10^{-3}$	$1.78 \cdot 10^{-3}$	$7.88 \cdot 10^{-4}$
$OT(B')$	582 ms	10 sec	5.6 hrs	9 sec	45 sec	10.7 hrs	576 ms	10 sec	5.4 hrs	9 sec	45 sec	10.5 hrs

(c) Redefined Weighted Node Pruning **(d) Reciprocal Weighted Node Pruning**

Table 4: Performance of the new pruning schemes on top of Block Filtering over all datasets, averaged across all weighting schemes.

	D_{1C}	**D_{2C}**	**D_{3C}**	**D_{1D}**	**D_{2D}**	**D_{3D}**
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, over 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 D_{1C}, where *BPE* takes the lowest value across all datasets (3.63), and by 92% for D_{2D}, 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 $O(2 \cdot BPE)$ 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 D_{1C} and D_{2D}, 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 Redfined 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 D_{2D}, where *PC* drops below 0.80 for all weighting schemes. Given that the corresponding Clean-Clean ER dataset (D_{2C}) exhibits much higher *PC*, the poor recall for D_{2D} 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 D_{1C} and D_{2C}. *(ii)* Reciprocal CNP typically outperforms CEP in all respects of effectiveness. On average, it executes 67% less comparisons, while increasing recall by 8%. D_{1C} is the exception that proves this rule: Reciprocal CNP saves more than an order of magnitude more comparisons, but CEP achieves slightly higher recall. *(ii)* 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 $PC \geq 0.8$. 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 trades 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 D_{2C}, 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

	R_{1D}	**R_{2D}**	**R_{3D}**	**R_{1C}**	**R_{2C}**	**R_{3C}**
$\|\|B'\|\|$	$4.22 \cdot 10^4$	$7.47 \cdot 10^5$	$4.86 \cdot 10^8$	$4.53 \cdot 10^5$	$2.10 \cdot 10^6$	$2.12 \cdot 10^9$
$PC(B')$	0.870	0.862	0.963	0.862	0.804	0.965
$PQ(B')$	0.048	0.026	0.002	0.004	0.009	$4.07 \cdot 10^{-4}$
$OTime(B')$	24 msec	86 msec	1 min	62 msec	150 msec	2 min

(a) Efficiency-intensive Graph-free Meta-blocking (*r*=0.25)

	R_{1D}	**R_{2D}**	**R_{3D}**	**R_{1C}**	**R_{2C}**	**R_{3C}**
$\|\|B'\|\|$	$2.21 \cdot 10^5$	$6.16 \cdot 10^6$	$8.52 \cdot 10^9$	$4.73 \cdot 10^6$	$2.36 \cdot 10^7$	$3.55 \cdot 10^{10}$
$PC(B')$	0.973	0.959	0.998	0.980	0.954	0.965
$PQ(B')$	$1.02 \cdot 10^{-2}$	$3.56 \cdot 10^{-3}$	$1.05 \cdot 10^{-4}$	$4.78 \cdot 10^{-4}$	$9.24 \cdot 10^{-4}$	$2.51 \cdot 10^{-5}$
$OTime(B')$	30 msec	221 msec	6 min	146 msec	650 msec	25 min

(b) Effectiveness-intensive Graph-free Meta-blocking (*r*=0.55)

	R_{1D}	**R_{2D}**	**R_{3D}**	**R_{1C}**	**R_{2C}**	**R_{3C}**
$\|\|B'\|\|$	$1.76 \cdot 10^6$	$1.32 \cdot 10^7$	$2.34 \cdot 10^{10}$	$9.07 \cdot 10^7$	$4.08 \cdot 10^8$	$4.81 \cdot 10^{10}$
$PC(B')$	0.994	0.980	0.999	0.997	0.981	0.999
$PQ(B')$	$1.31 \cdot 10^{-3}$	$1.70 \cdot 10^{-3}$	$3.81 \cdot 10^{-5}$	$2.54 \cdot 10^{-5}$	$5.49 \cdot 10^{-5}$	$1.85 \cdot 10^{-5}$
$OTime(B')$	76 msec	2 sec	1.9 hrs	5 sec	1 min	11.1 hrs

(c) Iterative Blocking

Table 6: Performance of the baseline methods over all datasets.

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 *PQ*, 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 above 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

[1] A. N. Aizawa and K. Oyama. A fast linkage detection scheme for multi-source information integration. In *WIRI*, pages 30–39, 2005.

[2] Y. Altowim, D. V. Kalashnikov, and S. Mehrotra. Progressive approach to relational entity resolution. *PVLDB*, 7(11):999–1010, 2014.

[3] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, page 5, 2006.

[4] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng.*, 24(9):1537–1555, 2012.

[5] V. Christophides, V. Efthymiou, and K. Stefanidis. *Entity Resolution in the Web of Data*. Morgan & Claypool Publishers, 2015.

[6] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. Large-scale linked data integration using probabilistic reasoning and crowdsourcing. *VLDB J.*, 22(5):665–687, 2013.

[7] X. L. Dong and D. Srivastava. *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2015.

[8] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[9] I. Fellegi and A. Sunter. A theory for record linkage. *Journal of American Statistical Association*, pages 1183–1210, 1969.

[10] L. Getoor and A. Machanavajjhala. Entity resolution: Theory, practice &amp; open challenges. *PVLDB*, 5(12):2018–2019, 2012.

[11] L. Getoor and A. Machanavajjhala. Entity resolution for big data. In *KDD*, 2013.

[12] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

[13] M. Hernández and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

[14] Y. Jiang, G. Li, J. Feng, and W. Li. String similarity joins: An experimental evaluation. *PVLDB*, 7(8):625–636, 2014.

[15] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493, 2010.

[16] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, pages 257–266, 2008.

[17] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007.

[18] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[19] A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[20] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl. Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data. In *WSDM*, pages 53–62, 2012.

[21] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Trans. Knowl. Data Eng.*, 25(12):2665–2682, 2013.

[22] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl. Meta-blocking: Taking entity resolution to the next level. *IEEE Trans. Knowl. Data Eng.*, 2014.

[23] G. Papadakis, G. Papastefanatos, and G. Koutrika. Supervised meta-blocking. *PVLDB*, 7(14):1929–1940, 2014.

[24] A. Thor and E. Rahm. Moma - a mapping-based object matching system. In *CIDR*, pages 247–258, 2007.

[25] M. J. Welch, A. Sane, and C. Drome. Fast and accurate incremental entity resolution relative to an entity knowledge base. In *CIKM*, pages 2667–2670, 2012.

[26] S. E. Whang, D. Marmaros, and H. Garcia-Molina. Pay-as-you-go entity resolution. *IEEE Trans. Knowl. Data Eng.*, 25(5):1111–1124, 2013.

[27] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, pages 219–232, 2009.