Big Data Research ••• (••••) •••–•••

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking

George Papadakis [a,*], George Papastefanatos [b], Themis Palpanas [c], Manolis Koubarakis [a]

[a] University of Athens, Greece
[b] Athena Research Center, Greece
[c] Paris Descartes University, France

ARTICLE INFO

Article history:
Received 15 March 2016
Accepted 29 August 2016
Available online xxxx

Keywords:
Entity Resolution
Redundancy-positive blocking
Meta-blocking

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 to raise precision at a limited cost in recall. At the core of this procedure lies the blocking graph, where the nodes correspond to entities and the edges connect the comparable pairs. There are several configurations for Meta-blocking, but no hints on best practices. In general, the node-centric approaches are more robust and suitable for a series of applications, but suffer from low precision, due to the large number of unnecessary comparisons they retain.

In this work, we present three novel methods for node-centric Meta-blocking that significantly improve precision. We also introduce a pre-processing method that restricts the size of the blocking graph by removing a large number of noisy edges. As a result, it reduces the overhead time of Meta-blocking by 2 to 5 times, while increasing precision by up to an order of magnitude for a minor cost in recall. The same technique can be applied as graph-free Meta-blocking, enabling for the first time Entity Resolution over very large datasets even on commodity hardware. We evaluate our approaches through an extensive experimental study over 19 voluminous, established datasets. The outcomes indicate best practices for the configuration of Meta-blocking and verify that our techniques reduce the resolution time of state-of-the-art methods by up to an order of magnitude.

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. Exhaustive solutions to this task suffer from low efficiency, due to their inherently quadratic complexity: every entity profile has to be compared with all others. This problem is accentuated by the continuously larger size of datasets that are now available on the Web. For example, the *LOD-Stats*[1] Web application recorded around a billion triples for Linked Open Data in December, 2011, which had grown to more than 100 billion triples by March, 2016. As a result, ER typically scales to large data collections through approximate techniques, which sacrifice recall to a controllable extent in order to enhance precision and time efficiency.

The most popular among these techniques is *blocking* [1–3]. It groups similar entities into clusters (called *blocks*) so that comparisons are executed only between the entities within each block [4,5]. Typically, blocking methods for Big Data have to overcome high levels of noise not only in attribute values, but also in attribute names, due to the unprecedented schema heterogeneity. For instance, Google Base[2] alone encompasses 100,000 distinct schemata that correspond to 10,000 entity types [6]. Most blocking methods deal with these high levels of noise through *redundancy* [1,7]: they place every entity profile into multiple blocks so as to reduce the likelihood of missed matches.

The simplest method of this type is Token Blocking [9,2]. It disregards schema information and semantics, creating a separate block for every token that appears in the attribute values of at least two entities. To illustrate its functionality, consider the entity profiles in Fig. 1(a), where p_1 and p_2 match with p_3 and p_4, respectively; Token Blocking clusters them in the blocks of Fig. 1(b), which place both pairs of duplicates in at least one common block

* Corresponding author.
E-mail addresses: gpapadis@di.uoa.gr (G. Papadakis), gpapas@imis.athena-innovation.gr (G. Papastefanatos), themis@mi.parisdescartes.fr (T. Palpanas), koubarak@di.uoa.gr (M. Koubarakis).

[1] http://stats.lod2.eu.

[2] http://www.google.com/base.

http://dx.doi.org/10.1016/j.bdr.2016.08.002
2214-5796/

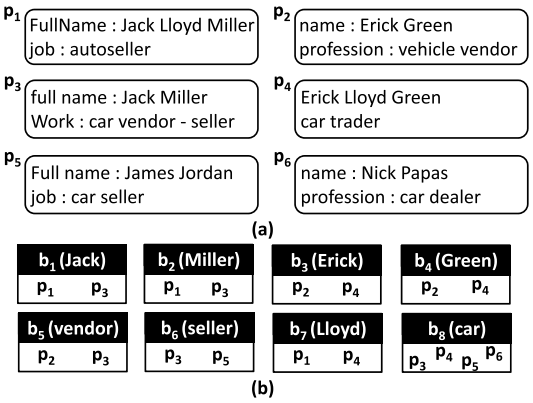

Fig. 1. (a) A set of entity profiles, and (b) the blocks of Token Blocking.

at the cost of 13 comparisons, in total. The resulting computational cost is high, given that the brute-force approach executes 15 comparisons.

This is a general trait of block collections that involve redundancy: in their effort to achieve high recall, they produce a large number of unnecessary comparisons. These come in two forms: the *redundant* ones repeatedly compare the same entity profiles across different blocks, while the *superfluous* ones compare non-matching entities. 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 Fig. 1(b) involve 3 redundant and 8 superfluous out of the 13 comparisons.

Current state-of-the-art. To mitigate this phenomenon, methods such as Comparison Propagation [10] and Iterative Blocking [11] aim to process an existing block collection in the optimal way (see Section 2 for more details). Among these methods, Meta-blocking achieves the best balance between precision and recall, being one of the few techniques to scale well to millions of entities [7,8]. In essence, it 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. This procedure operates in 2 steps.

First, it transforms B into the blocking graph G_B, which contains a node n_i for every entity p_i in B and an edge $e_{i,j}$ for every pair of *co-occurring entities* p_i and p_j (i.e., entities sharing at least one block). Fig. 2(a) depicts the graph for the blocks in Fig. 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 adjacent entities are matching, based on the blocks they have in common. For instance, the edges in Fig. 2(a) are weighted with the Jaccard similarity of the lists of blocks containing their adjacent entities. The edges with low weights correspond to *superfluous comparisons* and are pruned. A possible approach is to discard all edges with a weight lower than the overall mean one (1/4). This yields the pruned graph in Fig. 2(b).

Pruning algorithms of this type are called *edge-centric*, because they iterate over the edges of the blocking graph and retain the globally best ones. Higher recall is achieved by the *node-centric* pruning algorithms, which iterate over the nodes of the blocking graph and retain the locally best edges. These are the edges with the highest weights in each neighborhood and correspond to the most likely matches for each entity. In contrast, the edge-centric algorithms do not guarantee to include every entity in the restructured blocks. Their recall is lower than the node-centric algorithms by 20%, on average, when compared under the same settings [7].

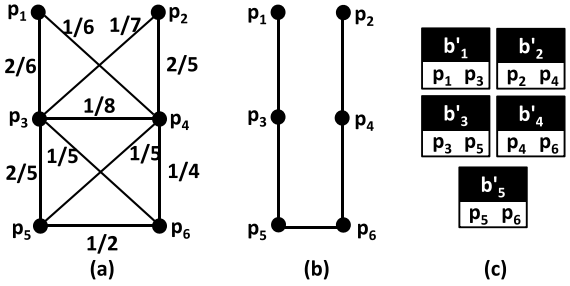

Fig. 2. (a) A blocking graph extracted from the blocks in Fig. 1(b), (b) one of the possible edge-centric pruned blocking graphs, and (c) the new blocks derived from it.

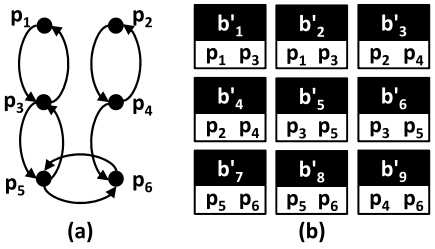

Fig. 3. (a) One of the possible node-centric pruned blocking graphs for the graph in Fig. 2(a). For clarity, the retained edges are directed and outgoing, since they might be preserved in the neighborhoods of both adjacent entities. (b) The new blocks derived from the pruned graph.

To illustrate the functionality of node-centric approaches, consider the pruned blocking graph in Fig. 3(a); for each node in Fig. 2(a), it has retained the adjacent edges that exceed the average weight of the neighborhood. Regardless of the type of the pruning algorithm, the restructured block collection B' is formed by creating a new block for every retained edge – as depicted in Figs. 2(c) and 3(b). In both cases, B' maintains the original recall, while reducing the number of executed comparisons to 5 and 9, respectively.

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

(i) Though more robust to recall, the node-centric pruning algorithms exhibit low efficiency, because they retain a considerable portion of redundant and superfluous comparisons. In most cases, their precision is lower than the edge-centric ones by 50% [7]. This is also illustrated in our example, where the restructured blocks of Fig. 3(b) contain 4 redundant comparisons in b'_2, b'_4, b'_6 and b'_8 and 3 superfluous in b'_5, b'_7 and b'_9; the edge-centric counterpart in Fig. 2(c) retains just 3 superfluous comparisons.

(ii) The processing of voluminous datasets involves a significant overhead. The corresponding blocking graphs comprise millions nodes that are strongly connected with billions edges. Inevitably, the pruning of such graphs is very time-consuming, leaving plenty of room for improving its efficiency (see Section 5.6).

(iii) Meta-blocking is difficult to configure. There are five different weighting schemes that can be combined with four pruning algorithms, thus yielding 20 *pruning schemes*, in total (see Section 3 for more details). As yet, there are no guidelines on how to choose the best configuration for the application at hand and the available resources.

Proposed solution. In this paper, we describe novel techniques for overcoming the weaknesses of Meta-blocking.

First, we propose three new node-centric pruning algorithms that achieve significantly higher precision than the existing ones. The most conservative approach, *Redundancy Pruning*, produces restructured blocks with no redundant comparisons and prunes up to 50% more comparisons. It achieves the same recall as the existing techniques, but its precision is almost the double. The other

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

two methods exploit generic properties of the blocking graph to prune at least 50% more comparisons than the existing techniques. *Graph Partitioning* applies only to bipartite blocking graphs, considering exclusively one of the two partitions in its processing. *Reciprocal Pruning* applies to any blocking graph, retaining only the edges that are important for both adjacent entities. Their recall is slightly lower than the baselines, but their precision is higher by up to an order of magnitude.

Second, we introduce *Block Filtering*, which removes every entity from the blocks that are the least important for it. This approach can be used in two ways: *(i)* In conjunction with graph-based pruning schemes, it acts as a pre-processing technique that shrinks the blocking graph, by discarding more than 50% of its unnecessary edges. Thus, it enhances the scalability of all graph-based Meta-blocking techniques to a significant extent. *(ii)* As a standalone, graph-free Meta-blocking method that involves significantly lower space and time complexities (i.e., overhead). Its configuration is straightforward, it scales to voluminous datasets even with commodity hardware, and it requires up to 2 orders of magnitude less time than the state-of-the-art method.

Finally, we address the problem of a-priori selecting the best pruning scheme, depending on the application at hand and the available resources. We analytically compare the performance of all Meta-blocking methods over 12 real and 7 synthetic established benchmarks, which range from few thousands to several million entities. Our experimental results provide insights into the effect of weighting schemes on each pruning algorithm and identify the pruning schemes that consistently exhibit the best balance between recall, precision and run-time for the main types of ER applications. Our thorough experiments also verify that our techniques outperform the best relevant methods in the literature as well as the best Meta-blocking techniques to a significant extent.

Contributions & paper organization. In summary, we make the following contributions:

- We present three new node-centric pruning algorithms that significantly improve the precision of existing ones from 30% to 800% at a small cost in recall.
- We introduce a graph-free technique that minimizes the overhead of Meta-blocking by cleaning the blocking graph from most of its noisy edges. With its help, ER scales to large datasets even with limited resources and the resolution time improves almost by an order of magnitude.
- We experimentally verify the superior performance of our new methods through an extensive study over 19 voluminous datasets with different characteristics. Its outcomes provide insights into the best configuration for Meta-blocking, depending on the resources 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: in Section 2, we delve into the most relevant works in the literature, while in Section 3, we formally define the task of Meta-blocking, elaborating on its main notions. Section 4 introduces our novel techniques, and Section 5 presents our thorough experimental evaluation. We conclude the paper in Section 6 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 [4,5]. A large part of the proposed techniques are approximate, with blocking being the most popular among them [1]. Some blocking methods produce disjoint blocks (e.g., Standard Blocking [12]), but most of them yield overlapping blocks with redundant comparisons. In this way, they achieve high recall in the context of noisy data. Depending on the interpretation of redundancy, blocking methods are distinguished into three categories [7]:

1. The *redundancy-positive* methods ensure that the more blocks two entities share, the more likely they are to be matching. In this category fall the Suffix Arrays [13], Q-grams Blocking [14], MFIBlocks [3], Attribute Clustering [2] and Token Blocking [9].
2. The *redundancy-negative* methods ensure that the most similar entities share just one block. In Canopy Clustering [15], for instance, the entities that are highly similar to the current seed are removed from the pool of candidate matches and are exclusively placed in its block.
3. The *redundancy-neutral* methods produce overlapping blocks, but the number of common blocks between two entities is irrelevant to their likelihood of matching. As such, consider the single-pass Sorted Neighborhood [16]: all pairs of entities co-occur in the same number of blocks, which is equal to the size of the sliding window.

Meta-blocking operates exclusively on top of redundancy-positive blocking methods, which are the most popular ones and have been proven to scale to large datasets [17,2,7].

Another line of research focuses on developing methods that optimize the processing of an existing block collection. In this category falls Meta-blocking, whose pruning schemes can be either unsupervised [7] or supervised [18]. The latter achieve higher accuracy than the former, due to the composite pruning rules that are learned by a classifier trained over a set of labeled edges. In practice, though, their utility is limited, because 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 techniques in this work.

In the same category falls Block Purging [9,2], which aims for discarding oversized blocks that are dominated by redundant and superfluous comparisons. It sets an upper limit on the comparisons that can be contained in a valid block and purges those blocks that exceed it. Comparison Propagation [10] discards all redundant comparisons by enumerating the input blocks and building an inverted index that points from entity ids to block ids. Then, it marks a comparison $c_{i,j}$ as redundant if the id of the current block is larger than the least common block id of the entities p_i and p_j. Iterative Blocking [11] 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.

Compared to Meta-blocking, the functionality of Block Purging is coarser and less accurate, because it targets entire blocks, instead of individual comparisons. Yet, Block Purging is frequently used as a pre-processing step for Meta-blocking [7,18]. The other two methods are competitive to Meta-blocking, because they target individual redundant comparisons. However, Meta-blocking goes beyond them, because it targets superfluous comparisons, as well. Its superior performance has been experimentally verified in [7].

3. Preliminaries

In this section, we first elaborate on the main notions of Entity Resolution and its evaluation metrics. Then, we delve into the functionality of Meta-blocking and introduce the weighting schemes and the pruning algorithms that have been proposed in the literature. Finally, we distinguish the applications of ER into two main

[3] See http://sourceforge.net/projects/erframework.

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

categories and explain which pruning algorithms are suitable for each of them.

Entity Resolution At the core of ER lies the notion of *entity profile*, p. As such, we define a uniquely identified collection of name-value pairs that describe a real-world object. A set of entity profiles is called *entity collection* (E). The goal of ER is to identify the different profiles that describe the same real-world object. Two such profiles, p_i and p_j, are called *duplicates* and are said to be *matching*, a condition denoted by $p_i \equiv p_j$. A non-redundant comparison between two duplicate entities is called *matching comparison*.

Depending on the input entity collection(s), we identify two general ER tasks [1,2,7]: *(i) Dirty ER* takes as input a single entity collection that contains 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 its goal is to identify the matching entities between them. In the context of Databases, the former task is called *Deduplication* and the latter *Record Linkage* [1]. Our methods apply uniformly to both Clean–Clean and Dirty ER, except for Graph Partitioning, which is only compatible with Clean–Clean ER.

Blocking improves the run-time of both ER tasks by grouping similar entities into blocks so that comparisons are limited between co-occurring entities. Placing an entity into a block is called *block assignment*. An individual block is symbolized by b, with $|b|$ denoting its *size* (i.e., number of entities) and $||b||$ denoting its *cardinality* (i.e., number of comparisons). In the case of Dirty ER, we have $||b|| = |b| \times (|b| - 1)/2$, while for Clean–Clean ER, we have $||b|| = |b^1| \times |b^2|$, where b^1 and b^2 are the two inner blocks of b that exclusively comprise entities from E_1 and E_2, respectively. A set of blocks is called *block collection* (B), with $|B|$ denoting its size (i.e., number of blocks) and $||B||$ its cardinality (i.e., total number of comparisons): $||B|| = \sum_{b_i \in B} ||b_i||$.

Performance Evaluation Metrics To assess the performance of a blocking method, we follow the best practice in the literature, treating entity matching as an orthogonal task [1,3,2]. We assume that two duplicate entities can be detected using any of the available methods as long as they co-occur in at least one block. $D(B)$ represents the set of co-occurring duplicate entities, whereas $D(E)$ denotes the set of duplicates that are contained in the input entity collection E. With their help, the following measures for block collections are defined [1,3,7]:

(i) Pairs Quality (PQ) assesses precision, expressing the portion of comparisons that correspond to 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 matching entities, PQ offers a pessimistic estimation of precision. More formally:

$$PQ = \frac{|D(B)|}{||B||},$$

where $|D(B)|$ stands for the size of $D(B)$. PQ takes values in the interval $[0, 1]$, with higher values indicating higher precision.

(ii) Pairs Completeness (PC) assesses recall, expressing the portion of existing pairs of duplicates that can be detected in B. More formally:

$$PC = \frac{|D(B)|}{|D(E)|},$$

where $|D(E)|$ stands for the size of $D(E)$. 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 performance of ER exclusively depends on the accuracy of the selected 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 $|D(B)|$), increasing PC, but reducing PQ. Hence, a blocking method is successful if it achieves a good balance between precision and recall.

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)	

Fig. 4. All configurations for the two main parameters of Meta-blocking: the weighting scheme and the pruning algorithm. Every configuration of the one parameter is compatible with all configurations of the other.

Meta-blocking The redundancy-positive block collections place every entity 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 [7]. More formally, Meta-blocking aims to transform a block collection B into a new one B' such that $PC(B') \approx PC(B)$ and $PQ(B') \gg PQ(B)$.

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

Definition 1. Given a block collection B, its *blocking graph* is an undirected graph $G_B = \{V_B, E_B, W_B\}$, where V_B is the set of its nodes such that $\forall p_i \in B \ \exists v_i \in V_B$, $E_B \subseteq V_B \times V_B$ is the set of undirected edges between all pairs of co-occurring entities in B, and W_B is the set of edge weights that take values in the interval $[0, 1]$ such that $\forall e_{i,j} \in E_B \ \exists w_{i,j} \in W_B$.

The goal of Meta-blocking is to prune the edges of the blocking graph in a way that retains the matching entities. The functionality of this procedure 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 entities. The two parameters are independent in the sense that every configuration of the one can be combined with every configuration of the other. Fig. 4 lists their values, which yield 20 configurations, in total. We explain their functionality below.

Weighting schemes for Meta-blocking. Five schemes have been proposed for weighting the edges of the blocking graph. The higher the weight they assign to an edge, the more likely it is to connect matching entities. For their formal definitions, we use the following notation: $B_i \subseteq B$ 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. All weights are normalized to the interval $[0, 1]$. The weighting schemes are the following [7]:

(i) Aggregate Reciprocal Comparisons Scheme (ARCS) captures the intuition that the smaller the blocks two entities share, the more likely they are to be matching. Thus, its weights are derived from the following formula:

$$ARCS(e_{i,j}) = \sum_{b_k \in B_{i,j}} \frac{1}{||b_k||}.$$

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

(ii) Common Blocks Scheme (CBS) expresses the fundamental property of redundancy-positive block collections that two entities are more likely to match, when they share many blocks. Thus, the weight of each edge is equal to the number of blocks the adjacent entities have in common:

$$CBS(e_{i,j}) = |B_{i,j}|.$$

(iii) Enhanced Common Blocks Scheme (ECBS) improves CBS by discounting the effect of the entities that are placed in a large number of blocks:

$$ECBS(e_{i,j}) = CBS(e_{i,j}) \cdot \log \frac{|B|}{|B_i|} \cdot \log \frac{|B|}{|B_j|}.$$

(iv) Jaccard Scheme (JS) estimates the portion of blocks shared by two entities:

$$JS(e_{i,j}) = \frac{|B_{i,j}|}{|B_i| + |B_j| - |B_{i,j}|}.$$

(v) Enhanced Jaccard Scheme (EJS) improves JS by discounting the effect of entities involved in too many non-redundant comparisons (i.e., high node degree):

$$EJS(e_{i,j}) = JS(e_{i,j}) \cdot \log \frac{|E_B|}{|v_i|} \cdot \log \frac{|E_B|}{|v_j|}.$$

Pruning schemes. Meta-blocking discards part of the edges of the blocking graph using an edge- or a node-centric pruning algorithm in combination with 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, we distinguish the pruning criteria into *weight thresholds*, which specify the minimum weight of the retained edges, and *cardinality thresholds*, which determine the maximum number of retained edges.

Every combination of a pruning algorithm and a pruning criterion is called *pruning scheme*. The following four pruning schemes were proposed in [7] and were experimentally verified to achieve a good balance between *PC* and *PQ*:

(i) Weighted Edge Pruning (WEP) comprises an edge-centric pruning algorithm that uses a global weight threshold. In essence, it discards all edges that do not exceed the average edge weight of the entire blocking graph.

(ii) Cardinality Edge Pruning (CEP) consists of an edge-centric algorithm that is coupled with a global cardinality threshold. It retains the top-K edges of the entire blocking graph, with K defined as:

$$K = \lfloor \frac{BC(B) \times |E|}{2} \rfloor,$$

where $BC(B)$ stands for the *Blocking Cardinality* of B, i.e., the average number of blocks associated with every entity in B: $BC(B) = \sum_{b_i \in B} |b_i|/|E|$ [2].

(iii) Weighted Node Pruning (WNP) comprises a node-centric pruning algorithm that employs a local weight threshold. For each node, it retains those adjacent entities that exceed the average edge weight of its neighborhood.

(iv) Cardinality Node Pruning (CNP) consists of a node-centric pruning algorithm that uses a global cardinality threshold. For each node, it retains the top-k edges of its neighborhood, where k is defined as:

$$k = \lfloor BC(B) - 1 \rfloor.$$

In short, WEP and WNP constitute *weight-based* pruning schemes, discarding the edges that do not exceed their weight threshold. Their configuration typically yields conservative thresholds, performing a shallow pruning that retains high recall [7]. On the other hand, CEP and CNP constitute *cardinality-based* pruning schemes; they rank the edges of the blocking graph in descending order of weight and retain the top-N weighted ones, where N is their cardinality threshold. For example, CEP could retain the 4 top-weighted edges of the graph in Fig. 2(a), resulting in the pruned graph of Fig. 2(b) and yielding the blocks of Fig. 2(c). Most commonly, though, CEP and CNP perform deep pruning, achieving higher precision than WEP and WNP at the cost of lower recall [7].

Applications of Entity Resolution Based on their requirements with respect to *PC* and *PQ*, the applications of ER can be distinguished into two categories:

(i) The *efficiency-intensive* applications aim to minimize the response time of ER, while detecting the vast majority of the duplicates. In other words, their goal is to maximize precision for a recall that exceeds 0.80. To this category belong real-time applications or applications with limited temporal resources, such as Pay-as-you-go ER [19], entity-centric search [20] and crowdsourcing ER [21]. Ideally, their goal is to identify a new pair of duplicate entities with every executed comparison.

(ii) The *effectiveness-intensive* applications afford a higher response time in order to maximize recall. At a minimum, *PC* should not fall below 0.95. Most such applications correspond to off-line batch processes like data cleaning in data warehouses, which practically call for almost perfect recall [22]. Yet, higher precision is pursued even in off-line applications in order to ensure that they scale well to voluminous datasets.

Meta-blocking facilitates both categories of ER applications through two types of pruning algorithms. The weight-based ones (WEP and WNP) accommodate the effectiveness-intensive applications and the cardinality-based algorithms (CEP and CNP) the efficiency-intensive applications.

4. Proposed approach

In this section, we present our four new methods that lead to scalable Meta-blocking. The first three constitute novel node-centric pruning algorithms that achieve higher precision than the existing ones. Their operation relies on the structure of the directed pruned blocking graph and aims to reduce the noisy, unnecessary edges that are retained for each node. They are competitive to each other and can only be used interchangeably.

Graph Partitioning exploits the bipartite blocking graphs that are formed in the case of Clean–Clean ER and the knowledge that every pair of duplicate entities is split among the two partitions. Thus, it suffices to retain the best edges only for the nodes of the one partition.

Redundancy Pruning and *Reciprocal Pruning* apply to both Clean–Clean and Dirty ER, as they merely consider the reciprocal links between two nodes. Both methods actually discard one of the reciprocal links in each case. They only differ in their treatment of the remaining edges, which do not have a reciprocal link. The former method retains these edges, while the latter discards them.

The fourth method, *Block Filtering*, is compatible with all graph-based Meta-blocking methods, operating as a pre-processing step that reduces the size of their blocking graph. This is accomplished by removing every entity from the least important of the associated blocks. Block Filtering can also be used independently, as a graph-free Meta-blocking technique.

We now elaborate on the functionality of each method, illustrating it through examples. We also analyze its scope, its configuration and its complexity.

4.1. Graph Partitioning

In the case of Clean–Clean ER, Meta-blocking creates a bipartite blocking graph, as the nodes of every entity collection are only

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

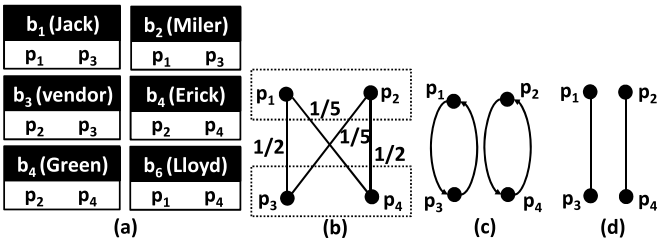

Fig. 5. (a) The blocks produced by Token Blocking for the entities of Fig. 1(a) in the case of Clean–Clean ER with $E_1 = \{p_1, p_2\}$ and $E_2 = \{p_3, p_4, p_5, p_6\}$, (b) the corresponding bipartite blocking graph, (c) the pruned blocking graph produced by the original node-centric pruning, and (d) the pruned blocking graph produced by Graph Partitioning.

connected with nodes from the other collection [7]. Graph Partitioning exploits this bipartite structure and produces restructured blocks with no redundant comparisons and a smaller portion of superfluous ones; it selects one of the node partitions, called *seed partition*, and creates new blocks only for its entities. In this way, the pruned blocking graph is undirected and contains no parallel edges (i.e., no redundant comparisons). Compared to the existing pruning algorithms, it is expected to contain approximately $\frac{|V_B|-|V_B^x|}{|V_B|}$% less comparisons, where V_B^x denotes the set of nodes of the seed partition.[4] Yet, it should retain most of the co-occurring duplicates, since every pair of matching entities is represented in the seed partition.

To illustrate its functionality, assume that the entities in Fig. 1(a) are divided in two individually clean (i.e., duplicate-free) entity collections: $E_1 = \{p_1, p_2\}$ and $E_2 = \{p_3, p_4, p_5, p_6\}$. In this case, Token Blocking yields the blocks in Fig. 5(a), which differ from those in Fig. 1(b) in that they contain entities from both E_1 and E_2. The corresponding bipartite blocking graph is depicted in Fig. 5(b). The upper partition comprises p_1 and p_2 and the lower one p_3 and p_4. Again, the edges are weighted with the Jaccard similarity of the block lists associated with the adjacent entities.

The original approach to node-centric pruning retains for every node one of its two edges, which is denoted as an outgoing edge in Fig. 5(c). In total, 4 edges/comparisons are retained, of which 2 are redundant. Graph Partitioning selects one of the node partitions, the upper one for example, and retains the best edges only for its nodes. The resulting undirected pruned blocking graph is presented in Fig. 5(d). It contains just 2 edges, with no redundant or superfluous comparisons.

Scope & functionality. This method exclusively applies to Clean–Clean ER. Theoretically, it is applicable to Dirty ER, as well, through *Graph Bipartization*, i.e., the task of identifying a minimum set of vertices that when deleted from a graph, make it bipartite. In practice, though, this approach does not scale well to the large blocking graphs produced by Meta-blocking, as Graph Bipartization belongs to the NP-complete complexity class [23]. Further, the relevant techniques cannot maintain the original levels of recall, as they cannot guarantee that most pairs of duplicate entities are split among the two partitions. This necessary condition holds by definition in the case of Clean–Clean ER.

Graph Partitioning leads to two new pruning algorithms: *Partition Weighted Node Pruning* (Partition WNP) and *Partition Cardinality Node Pruning* (Partition CNP). Their pseudocode is illustrated in Algorithms 1 and 2, respectively.

In more detail, Partition WNP iterates over the nodes of the seed partition of the blocking graph (Line 2 in Algorithm 1). For each

Algorithm 1: Partition Weighted Node Pruning.

Input: (i) G_B^{in} the blocking graph,
(ii) x the seed partition of the bipartite graph, and
(iii) wt the function defining the local weight thresholds.
Output: G_B^{out} the pruned blocking graph

1 $E_B^{out} \leftarrow \{\}$; // the set of retained edges
2 **foreach** $v_i \in V_B^x$ **do** // for each node in seed partit.
3 $G_{v_i} \leftarrow$ getNeighborhood(v_i, G_B);
4 $t_{v_i} \leftarrow wt(G_{v_i})$; // get local weight threshold
5 **foreach** $e_{i,j} \in E_{v_i}$ **do** // for every adjacent edge
6 **if** $t_{v_i} \leq e_{i,j}.weight$ **then** // retain it, if its
7 $E_B^{out} \leftarrow E_B^{out} \cup \{ e_{i,j} \}$; // weight exceeds t_{v_i}
8 **return** $G_B^{out} = \{V_B^x, E_B^{out}, WS\}$;

Algorithm 2: Partition Cardinality Node Pruning.

Input: (i) G_B^{in} the blocking graph,
(ii) x the seed partition of the bipartite graph, and
(iii) ct the function defining the local cardinality thresholds.
Output: G_B^{out} the pruned blocking graph

1 $E_B^{out} \leftarrow \{\}$; // the set of retained edges
2 **foreach** $v_i \in V_B^x$ **do** // for every node in seed partit.
3 $SortedStack_{v_i} \leftarrow \{\}$;
4 $G_{v_i} \leftarrow$ getNeighborhood(v_i, G_B);
5 $k \leftarrow ct(G_{v_i})$; // get local cardinality threshold
6 **foreach** $e_{i,j} \in E_{v_i}$ **do** // add every adjacent edge
7 $SortedStack_{v_i}$.push($e_{i,j}$); // in sorted stack
8 **if** $k < SortedStack_{v_i}.size()$ **then**
9 $SortedStack_{v_i}$.pop(); // remove last edge
10 $E_B^{out} \leftarrow E_B^{out} \cup SortedStack_{v_i}$; //retain top-k edges
11 **return** $G_B^{out} = \{V_B^x, E_B^{out}, WS\}$;

node, it extracts its neighborhood (Line 3) and estimates the current weight threshold, t_{v_i} (Line 4). Then, it retains those edges of the neighborhood that have a weight higher than or equal to t_{v_i} (Lines 5–7).

Partition CNP iterates over the nodes of the seed partition (Line 2 in Algorithm 2), extracts the corresponding neighborhood (Line 4) and computes its cardinality threshold, k (Line 5). Then, it adds all edges of the neighborhood in a sorted stack and selects the top k ones (Lines 6–9), which are retained in the pruned blocking graph (Line 10).

Configuration. For both pruning algorithms, we employ established pruning criteria [7]: Partition WNP uses local weight thresholds that are equal to the average weight of each node neighborhood, while Partition CNP sets the global cardinality threshold k equal to $k = \lfloor BC(B) - 1 \rfloor$.

Hence, the only parameter that determines the performance of Graph Partitioning is the criterion that selects the seed partition of the blocking graph. We specify as seed the partition with the lowest order (i.e., number of nodes) for both pruning algorithms; in case of a tie, one of the partitions is randomly selected. This criterion ensures the maximum efficiency gains, while being generic enough to apply to any blocking graph. It is also efficient, requiring a single iteration over the nodes of the graph, i.e., $O(|V_B|)$.

Complexity. The worst-case performance of Graph Partitioning corresponds to a complete blocking graph, which yields a time complexity of $O(|V_B^x| \cdot |E_B|)$. In practice, though, the blocking graph is sparse, with Partition WNP iterating over its edges twice: once for specifying the local weight thresholds and once for pruning the edges. Partition CNP performs a single iteration over E_B, due to the global threshold it employs. Hence, both algorithms account for a linear time complexity with respect to the graph size, i.e., $O(|E_B|)$.

Their space complexity is dominated by the storage requirements of the blocking graph, i.e., $O(|V_B| + |E_B|)$.

[4] This means that in the worst case, where both partitions have the same order (i.e., number of nodes), Graph Partitioning prunes 50% more comparisons than the existing pruning algorithms. In all other cases, its pruning is even deeper.

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

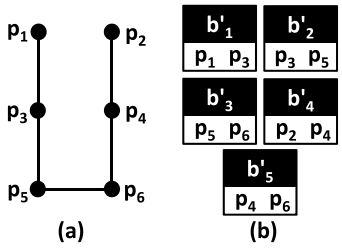

Fig. 6. (a) The undirected blocking graph produced by applying Redundancy Pruning to the graph in Fig. 3(a), and (b) the corresponding block collection.

4.2. Redundancy Pruning

This method focuses exclusively on removing all redundant comparisons from the restructured blocks produced by the existing node-centric pruning algorithms. This means that it does not target superfluous comparisons and, thus, it has no impact on the recall of WNP and CNP. In essence, it operates on top of the directed pruned blocking graph they produce, retaining an undirected edge for every pair of adjacent nodes – even if they are connected by two directed edges. In the extreme case where every retained edge has a reciprocal link, Redundancy Pruning saves 50% more comparisons, thus doubling precision.

As an example, Redundancy Pruning transforms the directed pruned graph in Fig. 3(a) into the undirected pruned graph in Fig. 6(a). The resulting block collection is depicted in Fig. 6(b); it maintains the same recall as the blocks in Fig. 3(b), but contains 4 less comparisons.

Scope & functionality. Redundancy Pruning is compatible with both Dirty and Clean–Clean ER and yields two new node-centric pruning algorithms: *Redundancy Weighted Node Pruning* (Redundancy WNP) and *Redundancy Cardinality Node Pruning* (Redundancy CNP).

A naive implementation for these methods is to apply Comparison Propagation to the output of WNP and CNP, respectively. However, this approach entails a significant overhead, especially in the case of large restructured block collections. For better performance, we integrate Comparison Propagation into the functionality of WNP and CNP. The corresponding approaches are presented in Algorithms 3 and 4, respectively.

In both cases, the processing consists of two phases. The first one involves a node-centric functionality, which iterates over the nodes of the blocking graph and calculates the pruning criterion from their neighborhood. Unlike the original pruning algorithms, there is a central data structure that stores the weight threshold of each neighborhood or the top-k nearest neighbors per node. The second phase operates in an edge-centric fashion: it iterates over the edges of the blocking graph and retains those that satisfy the pruning criterion for at least one of the adjacent nodes. In this way, every edge is retained at most once, even if it is important for both adjacent entities.

More specifically, Redundancy WNP first iterates over all nodes of the blocking graph to extract their neighborhood and estimate the corresponding weight threshold (Lines 2–4 in Algorithm 3). Then, it iterates over all edges and retains those exceeding the weight thresholds of either of the adjacent nodes (Lines 6–9).

Similarly, Redundancy CNP iterates over all nodes of the blocking graph in order 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 adjacent entities (Lines 10–13).

Configuration. Both Redundancy WNP and Redundancy CNP use the established pruning criteria [7]. The former uses local weight thresholds that are equal to the average weight of each node neighborhood, while the latter sets k equal to $k = \lfloor BC(B) - 1 \rfloor$.

Algorithm 3: Redundancy 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[] ← {}; // thresholds per node
2 **foreach** $v_i \in V_B$ **do** // for every node
3 G_{v_i} ← 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
9 $E_B^{out} \leftarrow E_B^{out} \cup \{ e_{i,j} \}$; // exceeds either threshold
10 **return** $G_B^{out} = \{V_B, E_B^{out}, WS\}$;

Algorithm 4: Redundancy 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*[] ← {}; // sorted stack per node
2 **foreach** $v_i \in V_B$ **do** // for every node
3 G_{v_i} ← 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 among
13 $E_B^{out} \leftarrow E_B^{out} \cup \{ e_{i,j} \}$; // the top-k ones for either node
14 **return** $G_B^{out} = \{V_B, E_B^{out}, WS\}$;

Complexity. The time complexity of Redundancy Pruning 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. Its space complexity is $O(|V_B| + |E_B|)$.

4.3. Reciprocal Pruning

This approach enhances the efficiency of the existing node-centric pruning algorithms by exploiting one of their disadvantages, namely the redundant comparisons that are contained in their restructured block collections. It actually turns it into their advantage, based on the idea that the retained redundant comparisons indicate pairs of entities that have high chances of matching. For example, the comparisons corresponding to the edges $e_{1,3}^{\rightarrow}$ and $e_{3,1}^{\rightarrow}$ in Fig. 3(a) indicate that p_1 is highly likely to match with p_3 and vice versa, thus reinforcing the likelihood that the two entities are duplicates.

Reciprocal Pruning builds on this pattern in order to improve the performance of both WNP or CNP. In essence, it applies them to the input block collection and retains only their redundant comparisons. That is, it retains one comparison for every pair of entities that are reciprocally connected in the directed pruned blocking graph; entities that are connected with a single edge, are not compared in the restructured block collection. In this way, the restructured block collection retains fewer superfluous comparisons, while producing no redundant ones. In the extreme case where all

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

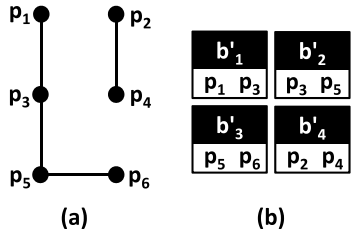

Fig. 7. (a) The pruned blocking graph produced by applying Reciprocal Pruning to the graph in Fig. 3(a), and (b) the restructured blocks.

retained edges have a reciprocal link, it prunes 50% more comparisons than the existing methods. In all other cases, the efficiency gains are even greater. Provided that the blocking graph captures strong patterns of co-occurrence, Reciprocal Pruning is also expected to maintain a large portion of the detected duplicates.

Continuing our example, Reciprocal Pruning operates on top of the directed pruned blocking graph in Fig. 3(a) to create the new, undirected pruned blocking graph in Fig. 7(a). Apparently, the latter graph contains an undirected edge for every pair of reciprocal directed edges in the former graph. Fig. 7(b) presents the restructured blocks that are produced by Reciprocal Pruning. They contain just 4 comparisons, of which 2 are matching and 2 superfluous (in b'_2 and b'_3). Compared to the blocks in Fig. 3(b), the overall efficiency is significantly enhanced at no cost in recall. Compared to Redundancy Pruning in Fig. 6(b), this approach prunes one more comparison.

Scope & functionality. Reciprocal Pruning is compatible with both Dirty and Clean–Clean ER. It yields two new node-centric pruning algorithms: *Reciprocal Weight Node Pruning* (Reciprocal WNP) and *Reciprocal Cardinality Node Pruning* (Reciprocal CNP). We do not present an outline of their functionality, as it is almost identical to Redundancy Pruning. The only difference is that they identify the retained edges using conjunctive instead of disjunctive conditions: the operator OR in Lines 7–8 and 11–12 of Algorithms 3 and 4, respectively, is replaced by the operator AND.

Configuration. Reciprocal WNP and CNP use the same established pruning criteria as the above techniques.

Complexity. The time complexity of Reciprocal Pruning is $O(|E_B|)$ in the case of a sparse graph, while its space complexity is $O(|V_B| + |E_B|)$.

4.4. Block Filtering

This approach is based on the idea that each block has a different importance for every entity it contains. For example, an oversized block is usually superfluous for most of its entities, but it may contain a couple of matching entities that do not appear in another block; for these entities, this particular block is indispensable, as they do not co-occur elsewhere.

Building upon this principle, Block Filtering restructures a block collection by removing entities from blocks, in which their presence is not necessary. The *importance* of a block for a given entity is implicitly determined by the maximum number of blocks that this entity should participate in.

As an example, consider the blocks in Fig. 1(b), assuming that their importance for their entities decreases with the increase of their index (i.e., 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 from the least important of its blocks, i.e., the one with the largest block index. The resulting block collection is presented in Fig. 8(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 in blocks b'_2 and b'_4 and 1 superfluous in

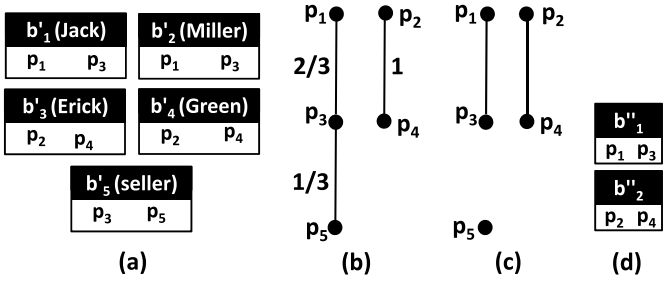

Fig. 8. (a) The restructured block collection that is produced by applying Block Filtering to the blocks in Fig. 1(b), (b) the corresponding blocking graph, (c) the pruned blocking graph that is produced by dropping the edges with a weight lower than the average, and (d) the corresponding restructured block collection.

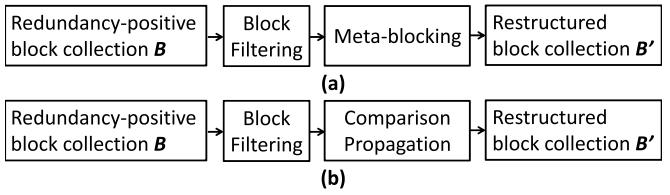

Fig. 9. (a) Using Block Filtering for pre-processing the blocking graph of graph-based Meta-blocking methods, and (b) using Block Filtering for graph-free Meta-blocking.

block b'_5. Applying Meta-blocking to these blocks yields the blocking graph of Fig. 8(b), which could be transformed into the graph of Fig. 8(c) with edge-centric pruning (i.e., by discarding the edges with a weight lower than the average). In this way, we get the two matching comparisons in Fig. 8(d). This is a significant improvement over the 4 comparisons in Fig. 2(c), which were produced by applying the same pruning scheme to the original block collection.

Scope & functionality. This is a generic method that applies to any block collection, covering both Clean–Clean or Dirty ER. It can be used in two fundamentally different ways, merely by adjusting its threshold to the desired level of pruning (see Section 5.4): *(i)* As a pre-processing method that performs shallow pruning to the edges of the blocking graph before applying a graph-based Meta-blocking technique. The corresponding workflow is depicted in Fig. 9(a). *(ii)* As an independent, graph-free Meta-blocking method that performs deep pruning. In this case, the restructured blocks contain redundant comparisons, which are removed by Comparison Propagation, increasing precision at no cost in recall. Fig. 9(b) shows the corresponding workflow.

The functionality of Block Filtering is outlined in Algorithm 5. 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 (Line 4). This requires an iteration over all blocks in order to count the block assignments of each entity. Subsequently, it iterates over all blocks in the specified order (Line 5) and over all entities in each block (Line 6). The entities that have less block assignments than their threshold are retained, while the rest are discarded from the current block (Lines 7–10). Finally, the block is retained only if it contains at least two entities (Lines 11–12).

Configuration. The functionality of Block Filtering is determined by two parameters:

(i) The criterion that specifies the importance of each block. As such, we employ the cardinality of blocks ($||b||$), assuming that the smaller a block is, the more important it is for its entities. Thus, Block Filtering sorts a block collection in ascending order of its cardinality, i.e., from the smallest block to the largest one in terms of comparisons.

(ii) The *ratio* (r) that determines the maximum number of block assignments per entity. It is defined in the interval $[0, 1]$ and ex-

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

Algorithm 5: Block Filtering.

```
   Input: B the input block collection
   Output: B' the restructured block collection
1  B' ← {};
2  counter[] ← {};
3  orderBlocks(B); // sort in descending importance
4  maxBlocks[] ← getThresholds(B); // limit per entity
5  foreach b_i ∈ B do          // check all blocks
6      foreach p_i ∈ b_i do // check all entities
7          if counter[i] < maxBlocks[i] then
8              b_i ← b_i \ p_i; // remove entity
9          else
10             counter[i]++; // increment counter
11     if |b_i| > 1 then          // retain blocks with
12         B' ← B' ∪ b_i; // at least 2 entities
13 return B';
```

presses the portion of blocks that are retained for each entity. For example, $r = 0.5$ means that each entity remains in the first half of its associated blocks, after sorting them in ascending cardinality. We experimentally fine-tune this parameter in Section 5.4.

Instead of a local threshold per entity, we could apply the same global threshold to all entities. Preliminary experiments, though, demonstrated that this approach is impractical in many cases. The reason is that the number of blocks associated with every entity varies significantly, depending on the quantity and the quality of information in the corresponding entity profiles. The smaller the global threshold is, the higher the efficiency gains are, but more information is then removed from entities with large profiles (i.e., many block assignments), potentially reducing recall. The opposite is true for large global thresholds. 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 entities. This phenomenon is particularly intense for Clean–Clean ER, where it is common for the input entity collections to differ largely in their characteristics.

Complexity. Block Filtering iterates twice over all entities in the blocks of the input collection B. Thus, its time complexity is equal to $O(\sum_{b_i \in B} |b_i|)$. Given, though, that every block contains a fixed number of entities that is independent of the size of the input entity collection, its time complexity is equal to $O(|B|)$. Apparently, this linear complexity scales very well to voluminous sets of blocks.

The space complexity of Block Filtering is dominated by the threshold and the counter it maintains for every entity (Line 2). Thus, it is linear to the size of the input entity collection, $O(|E|)$.

5. Evaluation

In this section, we delve into the performance characteristics of our methods through a thorough experimental study. We begin with a presentation of the datasets and the relevant metrics in Section 5.1. We then discuss the performance of the existing Meta-blocking techniques in Section 5.2. We examine the three new methods for node-centric pruning in Section 5.3. In Section 5.4, we fine-tune Block Filtering and demonstrate its beneficial effect on the input blocks, their blocking graph and the main pruning algorithms. We discuss the relative performance of the main pruning schemes in Section 5.5, selecting the best one for each ER sub-task and for each type of ER applications. We analytically compare the selected graph-based techniques with graph-free Meta-blocking and Iterative Blocking in Section 5.6.

5.1. Experimental set-up

We implemented our approaches in Java 1.7 and tested them on a server with Intel i7-4930K (3.40 GHz) and 32 GB RAM, running Debian 7.

Datasets. We examine the performance of our methods over 6 real and 7 synthetic datasets. They constitute established benchmarks [24,9,10,2,7,25] with significant variations in their size and characteristics. The real datasets 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. The synthetic datasets pertain to Dirty ER and are mainly used for scalability analysis [1,3]. In total, we have 19 block collections that lay the ground for a thorough experimental evaluation of our Meta-blocking techniques. All datasets are publicly available.[5]

To present the technical characteristics of the original datasets and the resulting block collections, we use the following notation: $|E|$ stands for the number of entities they contain, NVP for the total number of name-value pairs in their entity profiles, $|D(E)|$ for the number of existing duplicates, BFC for the comparisons executed by the brute-force approach, $|B|$ for the number of blocks in B, $||B||$ for the total cardinality of B, $PC(B)$ for the original recall of B (before Meta-blocking), $PQ(B)$ for the original precision, $|V_B|$ for the order of the blocking graph and $|E_B|$ for its size. Finally, $RT(B)$ denotes the resolution time of B, i.e., the time required for applying an entity matching method to all pairs of entities in B. As such, we use the Jaccard similarity of all tokens in the values of two entity profiles – regardless of the associated attribute names.

Tables 1(a) and (b) present the technical characteristics of the real datasets for Clean–Clean and Dirty ER, respectively. R_{1C} contains product entities from the on-line retailers Abt.com and Buy.com; they have been matched based on their Universal Product Code (UPC). R_{2C} contains bibliographic data stemming from the DBLP[6] and the ACM digital library,[7] while R_{3C} matches DBLP publications with query results from Google Scholar[8]; in both cases, the ground-truth was determined manually. R_{4C} contains product entities from Amazon.com that have been matched with query results from Google Base through their UPC. R_{5C} matches movies from IMDB[9] and DBPedia[10] based on their ImdbId. Finally, R_{6C} involves entities from two snapshots of the English Wikipedia[11] Infoboxes, which have been automatically matched through their URL. For Dirty ER, the datasets R_{xD} with $x \in [1, 6]$ were derived by merging the entity profiles of the individually clean collections forming R_{xC}.

Table 1(c) presents the technical characteristics of the synthetic datasets. They have been generated by *FEBRL* [26] in two stages: first, a series of original (i.e., duplicate-free) entities were produced based on frequency tables of person names and address values. Then, duplicate entities were generated by applying random modifications, such as character or word insertions and deletions. The resulting datasets contain 60% original entities and 40% duplicate ones, with up to nine matches per original entity.

From all datasets, we extracted a redundancy-positive block collection by applying Token Blocking [9] in conjunction with Block Purging [2]. The resulting blocks exhibit nearly perfect recall in all cases, as their *PC* consistently exceeds 0.98. Most of them involve at least 75% less comparisons than the brute-force approach, which

[5] See http://sourceforge.net/projects/erframework.
[6] http://www.dblp.org.
[7] http://dl.acm.org.
[8] http://scholar.google.gr.
[9] http://www.imdb.com.
[10] http://dbpedia.org.
[11] http://en.wikipedia.org.

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

Table 1
Technical characteristics of the datasets used in our experimental study.

	Entity characteristics				Block characteristics					Blocking graph	
	$\|E\|$	NVP	$\|D(E)\|$	BFC	$\|B\|$	$\|\|B\|\|$	$PC(B)$	$PQ(B)$	$RT(B)$	$\|V_B\|$	$\|E_B\|$
R_{1C}	1,076–1,076	2,568–2,308	1,076	$1.16 \cdot 10^6$	2,020	$1.61 \cdot 10^5$	0.983	0.007	4 s	2,152	$1.17 \cdot 10^5$
R_{2C}	2,616–2,294	10,464–9,162	2,224	$6.00 \cdot 10^6$	6,819	$3.75 \cdot 10^5$	0.999	0.006	6 s	4,910	$2.87 \cdot 10^5$
R_{3C}	2,516–61,353	10,064–198,001	2,308	$1.54 \cdot 10^8$	6,877	$1.92 \cdot 10^6$	0.994	0.001	22 s	63,869	$1.83 \cdot 10^6$
R_{4C}	1,354–3,039	5,302–9,110	1,104	$4.11 \cdot 10^6$	6,676	$2.74 \cdot 10^6$	0.999	$4.03 \cdot 10^{-4}$	8 min	4,393	$1.37 \cdot 10^6$
R_{5C}	27,615–23,182	155,436–816,009	22,863	$6.40 \cdot 10^8$	40,732	$8.11 \cdot 10^7$	0.980	$2.76 \cdot 10^{-4}$	1.6 hrs	50,797	$6.75 \cdot 10^7$
R_{6C}	$1.19 \cdot 10^6$–$2.16 \cdot 10^6$	$1.69 \cdot 10^7$–$3.50 \cdot 10^7$	892,586	$2.58 \cdot 10^{12}$	$1.24 \cdot 10^6$	$4.23 \cdot 10^{10}$	0.999	$2.11 \cdot 10^{-5}$	342 hrs	$3.33 \cdot 10^6$	$3.58 \cdot 10^{10}$
(a) Real, Clean–Clean ER datasets											
R_{1D}	2,152	4,876	1,076	$2.31 \cdot 10^6$	3,752	$2.08 \cdot 10^6$	0.991	0.001	6 s	2,151	$9.71 \cdot 10^5$
R_{2D}	4,910	19,626	2,224	$1.21 \cdot 10^7$	7,571	$3.34 \cdot 10^6$	1.000	0.001	7 s	4,910	$2.72 \cdot 10^6$
R_{3D}	63,869	208,065	2,308	$2.04 \cdot 10^9$	44,138	$3.13 \cdot 10^8$	0.999	$7.37 \cdot 10^{-6}$	8 min	63,864	$2.62 \cdot 10^8$
R_{4D}	4,393	14,412	1,104	$9.65 \cdot 10^6$	10,067	$2.79 \cdot 10^7$	1.000	$3.96 \cdot 10^{-5}$	32 min	4,393	$7.21 \cdot 10^6$
R_{5D}	50,797	971,445	22,863	$1.29 \cdot 10^9$	76,344	$8.38 \cdot 10^8$	0.982	$2.68 \cdot 10^{-5}$	8 hrs	50,769	$3.80 \cdot 10^8$
R_{6D}	$3.35 \cdot 10^6$	$5.19 \cdot 10^7$	892,586	$5.63 \cdot 10^{12}$	$1.50 \cdot 10^6$	$8.00 \cdot 10^{10}$	0.999	$1.12 \cdot 10^{-5}$	646 hrs	$3.33 \cdot 10^6$	$6.65 \cdot 10^{10}$
(b) Real, Dirty ER datasets											
S_1	10,000	106,108	8,705	$5.00 \cdot 10^7$	11,076	$4.06 \cdot 10^6$	0.997	0.002	1 min	10,000	$3.90 \cdot 10^6$
S_2	50,000	530,854	43,071	$1.25 \cdot 10^9$	40,683	$9.58 \cdot 10^7$	0.998	$4.49 \cdot 10^{-4}$	21 min	50,000	$9.28 \cdot 10^7$
S_3	100,000	$1.06 \cdot 10^6$	85,497	$5.00 \cdot 10^9$	72,720	$3.79 \cdot 10^8$	0.997	$2.25 \cdot 10^{-4}$	76 min	100,000	$3.68 \cdot 10^8$
S_4	200,000	$2.12 \cdot 10^6$	172,403	$2.00 \cdot 10^{10}$	123,759	$1.52 \cdot 10^9$	0.997	$1.13 \cdot 10^{-4}$	4.5 hrs	200,000	$1.48 \cdot 10^9$
S_5	300,000	$3.18 \cdot 10^6$	257,034	$4.50 \cdot 10^{10}$	166,208	$3.42 \cdot 10^9$	0.997	$7.50 \cdot 10^{-5}$	10 hrs	300,000	$3.32 \cdot 10^9$
S_6	$1.00 \cdot 10^6$	$1.06 \cdot 10^7$	857,538	$5.00 \cdot 10^{11}$	442,101	$2.21 \cdot 10^{10}$	0.996	$3.87 \cdot 10^{-5}$	55 hrs	$1.00 \cdot 10^6$	$2.12 \cdot 10^{10}$
S_7	$2.00 \cdot 10^6$	$2.12 \cdot 10^7$	$1.72 \cdot 10^6$	$2.00 \cdot 10^{12}$	863,631	$8.84 \cdot 10^{10}$	0.996	$1.93 \cdot 10^{-5}$	221 hrs	$2.00 \cdot 10^6$	$8.67 \cdot 10^{10}$
(c) Synthetic, Dirty ER datasets											

compares all possible pairs of entities. Still, their precision is significantly lower than 0.01 across all datasets. 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.

Measures. To assess the impact of Meta-blocking on *blocking effectiveness*, we use two measures: *(i)* the absolute recall of the block collections, as expressed through Pairs Completeness (*PC*), and *(ii)* their relative recall, as expressed through *Relative Pairs Completeness* (*RPC*). The latter measure captures the ratio between the recall of the restructured block collection B' and the original one, B:

$$RPC(B, B') = PC(B')/PC(B).$$

RPC takes positive values, with higher ones indicating better recall for B'. Those exceeding 1 indicate that the restructured block collection B' has a higher recall than B.

To assess the impact of Meta-blocking on *blocking efficiency*, we use the total cardinality of the block collections ($||B||$) and their precision, i.e., Pairs Quality (*PQ*). We also employ the following established measures [26,7]:

- The *Reduction Ratio* (*RR*) captures the relative decrease in the cardinality of the restructured block collection B' in comparison with the original one B:

$$RR(B, B') = 1 - ||B'||/||B||.$$

Provided that $||B'|| < ||B||$, *RR* takes values in the interval $[0, 1]$, with higher ones indicating better efficiency.

- The *Overhead Time* (*OTime*) measures the time taken by a Meta-blocking method to restructure the input block collection. The lower its value is, the more efficient is the corresponding method.
- The *Resolution Time* (*RTime*) is equal to *OTime* plus the time required to apply an entity matching method to all comparisons in the restructured block collection. As the entity matching method, we use the Jaccard similarity of all tokens in the values of two entity profiles. Note that this similarity metric is merely used for demonstration purposes, as it does not affect the detection of real matches. The lower $RTime(B)$ is, the more efficient is the block collection B.

5.2. Existing Meta-blocking techniques

Table 2 presents the performance of the existing pruning algorithms, averaged across all weighting schemes.

Starting with the cardinality-based algorithms, we observe that CEP reduces the executed comparisons from a whole order of magnitude, for the smallest datasets, up to 3 orders for the largest ones. 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 below 0.80 for four out of the six datasets. The reason is that Dirty ER is more difficult than Clean–Clean ER, involving much larger blocking graphs with more noisy edges between non-matching entities.

Compared to CEP, CNP is more robust to recall. For Clean–Clean ER, its *PC* consistently exceeds 0.90, whereas for Dirty ER, its *PC* drops below 0.80 only for a single dataset. This robustness stems from its node-centric functionality, which retains the best edges per node, instead of the globally best ones. It results, however, in lower efficiency than CEP, as CNP retains twice as many comparisons, on average. This leads to a lower precision than CEP, as well.

For the weight-based algorithms, we observe that WEP consistently maintains a recall high enough to satisfy the requirements of effectiveness-intensive ER applications ($PC > 0.95$). The only exception is the smallest dataset for Clean–Clean ER, namely R_{1C}. At the same time, WEP reduces the executed comparisons by a whole order of magnitude for most datasets and enhances precision to a similar extent.

As expected, its node-centric counterpart, WNP, is more robust to recall. Its *PC* consistently exceeds 0.96 and outperforms WEP across all datasets. On average, though, it executes twice as many comparisons, thus exhibiting significantly lower precision than WEP.

In summary, our experiments verify that the cardinality-based algorithms are more suitable for efficiency-intensive ER applications, as they excel in precision. In contrast, the weight-based

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

Table 2
Average performance of the existing pruning algorithms across all weighting schemes over the real datasets.

	Clean–Clean ER						Dirty ER					
	R_{1C}	R_{2C}	R_{3C}	R_{4C}	R_{5C}	R_{6C}	R_{1D}	R_{2D}	R_{3D}	R_{4D}	R_{5D}	R_{6D}
$\|\|B'\|\|$	$1.38 \cdot 10^4$	$2.73 \cdot 10^4$	$1.43 \cdot 10^5$	$7.54 \cdot 10^4$	$7.14 \cdot 10^5$	$2.63 \cdot 10^7$	$2.37 \cdot 10^4$	$3.45 \cdot 10^4$	$4.04 \cdot 10^5$	$1.08 \cdot 10^5$	$8.86 \cdot 10^5$	$2.47 \cdot 10^7$
$PC(B')$	0.901	0.987	0.967	0.799	0.865	0.724	0.614	0.985	0.814	0.493	0.471	0.535
$PQ(B')$	0.070	0.081	0.016	0.012	0.028	0.025	0.028	0.063	0.005	0.005	0.012	0.019
(a) Cardinality Edge Pruning (CEP)												
$\|\|B'\|\|$	$2.56 \cdot 10^4$	$5.36 \cdot 10^4$	$2.34 \cdot 10^5$	$1.48 \cdot 10^5$	$1.41 \cdot 10^6$	$4.95 \cdot 10^7$	$4.72 \cdot 10^4$	$6.87 \cdot 10^4$	$7.66 \cdot 10^5$	$2.15 \cdot 10^5$	$1.72 \cdot 10^6$	$4.63 \cdot 10^7$
$PC(B')$	0.944	0.997	0.974	0.919	0.947	0.974	0.867	0.995	0.956	0.772	0.870	0.954
$PQ(B')$	0.040	0.041	0.010	0.007	0.015	0.018	0.020	0.032	0.003	0.004	0.012	0.018
(b) Cardinality Node Pruning (CNP)												
$\|\|B'\|\|$	$2.62 \cdot 10^4$	$5.26 \cdot 10^4$	$4.32 \cdot 10^5$	$4.39 \cdot 10^5$	$1.48 \cdot 10^7$	$6.64 \cdot 10^9$	$2.40 \cdot 10^5$	$5.37 \cdot 10^5$	$4.78 \cdot 10^7$	$2.17 \cdot 10^6$	$1.03 \cdot 10^8$	$1.19 \cdot 10^{10}$
$PC(B')$	0.933	0.990	0.977	0.964	0.963	0.977	0.952	0.998	0.994	0.952	0.955	0.973
$PQ(B')$	0.043	0.049	0.011	0.003	$2.62 \cdot 10^{-3}$	$6.66 \cdot 10^{-4}$	0.006	0.007	$1.16 \cdot 10^{-4}$	$8.56 \cdot 10^{-4}$	$5.94 \cdot 10^{-4}$	$3.54 \cdot 10^{-4}$
(c) Weighted Edge Pruning (WEP)												
$\|\|B'\|\|$	$5.50 \cdot 10^4$	$1.11 \cdot 10^5$	$1.11 \cdot 10^6$	$8.32 \cdot 10^5$	$2.81 \cdot 10^7$	$1.60 \cdot 10^{10}$	$5.39 \cdot 10^5$	$1.04 \cdot 10^6$	$9.95 \cdot 10^7$	$4.50 \cdot 10^6$	$2.26 \cdot 10^8$	$3.00 \cdot 10^{10}$
$PC(B')$	0.961	0.997	0.988	0.988	0.972	0.997	0.976	0.999	0.996	0.982	0.973	0.995
$PQ(B')$	0.020	0.022	0.002	0.002	$1.14 \cdot 10^{-3}$	$1.44 \cdot 10^{-4}$	0.003	0.003	$4.79 \cdot 10^{-5}$	$3.91 \cdot 10^{-4}$	$2.42 \cdot 10^{-4}$	$7.63 \cdot 10^{-5}$
(d) Weighted Node Pruning (WNP)												

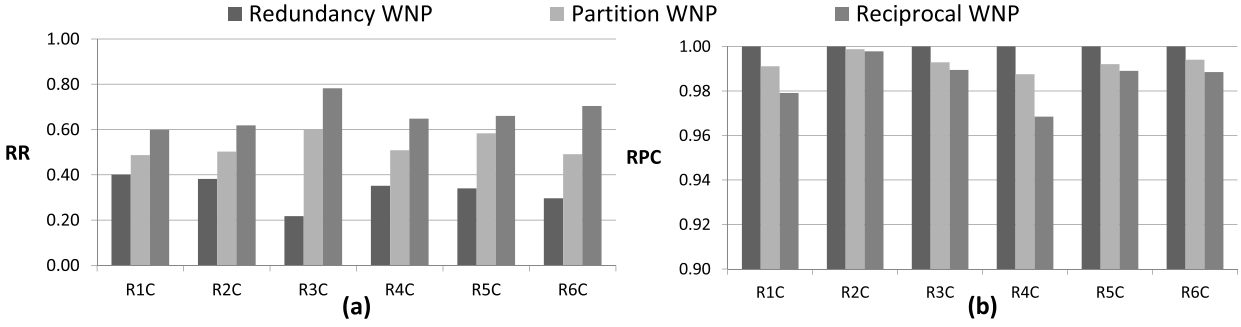

Fig. 10. (a) *RR* and (b) *RPC*, averaged across all weighting schemes, for the new methods for WNP over the real datasets for Clean–Clean ER.

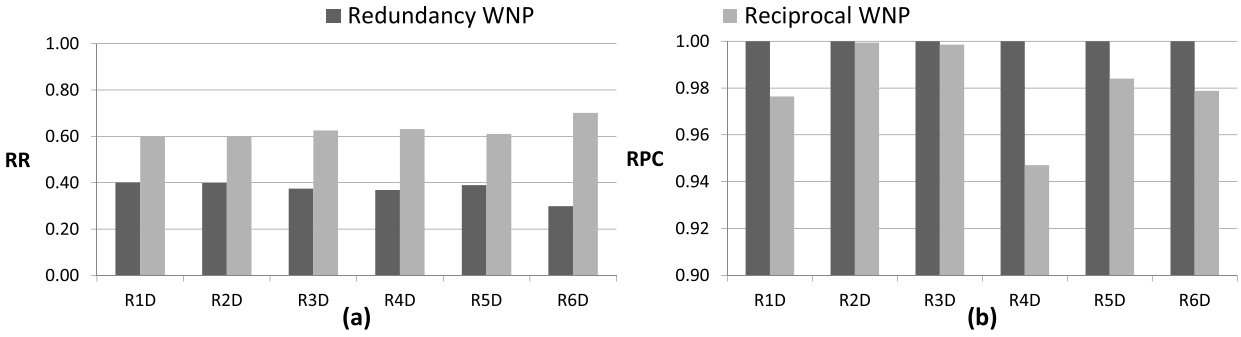

Fig. 11. (a) *RR* and (b) *RPC*, averaged across all weighting schemes, for the new methods for WNP over the real datasets for Dirty ER.

algorithms excel in recall and are more suitable for effectiveness-intensive applications. In both cases, the node-centric algorithms trade higher recall for lower precision. These patterns verify previous findings about the relative performance of pruning algorithms [7].

5.3. New node-centric pruning algorithms

We now examine our novel methods for node-centric pruning: Graph Partitioning, Redundancy Pruning and Reciprocal Pruning. We combined them with the five weighting schemes and applied the resulting pruning schemes to all real datasets. In each case, we consider *RR* and *RPC* with respect to the original node-centric pruning algorithm, averaged across all weighting schemes. From these two metrics, we can infer the effect on precision, as well: for an *RPC* close to 1, precision increases by 1.5 times for $RR = 0.33$, by 2 times for $RR = 0.50$, by 3 times for $RR = 0.67$ and by 4 times for $RR = 0.75$. We first examine the weight-based pruning algorithms and then the cardinality-based ones.

New methods for Weighted Node Pruning (WNP) The outcomes of our experimental analysis over the Clean–Clean and the Dirty ER datasets are presented in Figs. 10 and 11, respectively. Starting with Clean–Clean ER, we observe that the new methods can be distinguished in two categories: the first one entails Redundancy WNP, which maintains the original recall ($RPC = 1$), while conveying significant enhancements in efficiency. On average, across all weighting schemes and datasets, this method reduces total cardinality by 33% and increases precision by 1.5 times.

The second category entails the two other methods, which sacrifice recall in order to enhance efficiency to a larger extent. Parti-

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

Table 3
Performance of the best weight-based, node-centric pruning algorithms over the real datasets, averaged across all weighting schemes.

	Clean–Clean ER						Dirty ER					
	R_{1C}	R_{2C}	R_{3C}	R_{4C}	R_{5C}	R_{6C}	R_{1D}	R_{2D}	R_{3D}	R_{4D}	R_{5D}	R_{6D}
$\|\|B'\|\|$	$2.82 \cdot 10^4$	$5.52 \cdot 10^4$	$4.48 \cdot 10^5$	$4.09 \cdot 10^5$	$1.17 \cdot 10^7$	$8.16 \cdot 10^9$	$2.16 \cdot 10^5$	$4.13 \cdot 10^5$	$3.73 \cdot 10^7$	$1.66 \cdot 10^6$	$8.81 \cdot 10^7$	$8.96 \cdot 10^9$
$PC(B')$	0.953	0.996	0.981	0.976	0.964	0.991	0.953	0.999	0.994	0.930	0.956	0.980
$PQ(B')$	0.040	0.045	0.010	0.003	$2.54 \cdot 10^{-3}$	$2.99 \cdot 10^{-4}$	0.007	0.007	$1.33 \cdot 10^{-4}$	$1.17 \cdot 10^{-3}$	$6.75 \cdot 10^{-4}$	$3.15 \cdot 10^{-4}$
	(a) Partition WNP						(b) Reciprocal WNP					

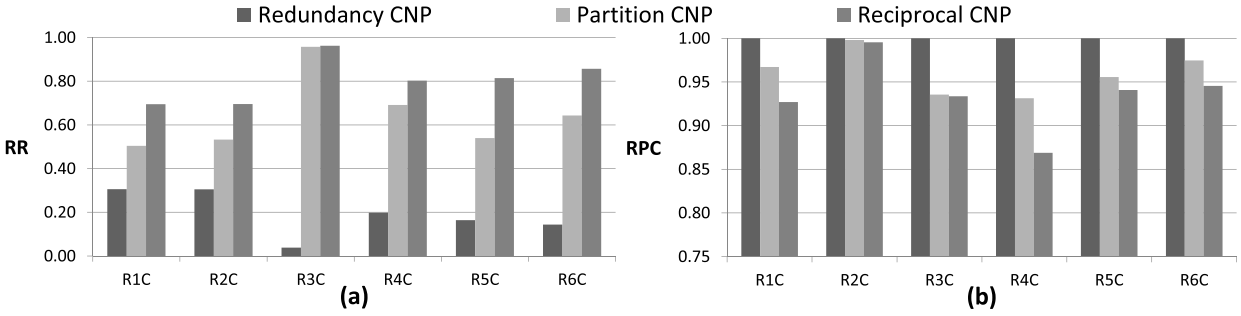

Fig. 12. (a) *RR* and (b) *RPC*, averaged across all weighting schemes, for the new methods for CNP over the real datasets for Clean–Clean ER.

tion WNP performs a conservative pruning that is robust to recall, decreasing *PC* by less than 1% ($RPC = 0.993$, on average). Yet, its pruning is so accurate that it manages to reduce total cardinality by 53% and to increase the original precision by 2.4 times. Reciprocal WNP performs a deeper pruning that consistently achieves the highest *RR* at the cost of the lowest *RPC*. On average, it reduces total cardinality by 67% for $RPC = 0.985$, while enhancing precision by 3.3 times.

In the case of Dirty ER, we observe similar patterns for the two methods that are applicable. Redundancy WNP maintains the original recall and increases efficiency to a significant extent. On average, its *RR* amounts to 0.373, rising precision by 1.6 times. Reciprocal WNP performs a drastic pruning that retains 63% less comparisons at the cost of slightly lower recall. On average, $RPC = 0.981$, while precision rises by 2.7 times.

Best approach. Ideally, the best method for WNP yields the highest increase in precision, while inducing the lowest reduction in recall. Our experimental outcomes indicate that there is no such clear winner among the three new pruning algorithms. Instead, there is a clear trade-off between recall and precision, with small reductions in *RPC* yielding large increases in *RR* and, thus, precision. This applies to both Clean–Clean and Dirty ER. Therefore, for each ER task, we seek the pruning algorithm that consistently exhibits the best balance between precision and recall in the sense that it maximizes *RR* for a *PC* that consistently exceeds 0.95 – the recall threshold of effectiveness-intensive applications.

In the case of Clean–Clean ER, this condition is satisfied by Partition WNP. Redundancy WNP achieves much lower *RR* and precision, while Reciprocal WNP is less robust to recall, reducing it below 0.95 for several weighting schemes.

In the case of Dirty ER, Redundancy WNP yields much lower *RR* and precision than Reciprocal WNP. The latter violates the recall constraint only over R_{4D}, where *RPC* drops to 0.947 and its recall is 0.93, on average. However, this dataset contains the highest levels of noise: it is the only dataset, where the blocks contain more comparisons than the brute-force approach, while the edges of the blocking graph connect 75% of all pairs of nodes (this can be derived by dividing $|E_B|$ with BFC from Table 1). Even in this context, the combination of Reciprocal WNP with ECBS achieves a recall equal to 0.957, whereas the *PC* of the other weighting schemes exceeds 0.92 in all cases. Thus, we can deduce that Reciprocal WNP offers the best balance between recall and precision for Dirty ER.

The average performance of Partition and Reciprocal WNP over the real datasets for Clean–Clean and Dirty ER, respectively, is presented in Table 3. Comparing it with Table 2(d), we observe that Partition WNP increases the precision of WNP by at least two times across all datasets, while maintaining *PC* above the recall threshold. Compared to the performance of WEP in Table 2(c), it consistently exhibits higher recall, which is combined with higher precision over R_{4C} and R_{5C}, as well.

In the case of Dirty ER, Reciprocal WNP satisfies the recall constraint across all datasets, except for R_{4D}, while increasing precision by 2 to 3 times. Its precision is consistently higher even in comparison with WEP. Given that its recall is also higher across all datasets (except for R_{4D}), it outperforms WEP in all aspects.

New methods for Cardinality Node Pruning (CNP) The performance of the new methods for CNP, averaged across all weighting schemes, is presented in Figs. 12 and 13 for the Clean–Clean and the Dirty ER datasets, respectively.

Starting with Clean–Clean ER, we observe that there are two extremes in the performance of the new pruning algorithms. On the one hand, Redundancy CNP maintains the original recall, while conveying a moderate increase in efficiency. On average, across all weighting schemes and datasets, it retains 19% less comparisons, which increases *PQ* by 1.2 times. On the other hand, Reciprocal CNP exhibits the highest precision across all datasets at the cost of the largest reductions in recall. On average, it retains 81% less comparisons for $RPC = 0.934$ and increases precision by 8.1 times. Partition CNP lies in the middle of these two extremes; on average, $RR = 0.645$ and $RPC = 0.961$, while precision increases by 5.6 times.

In the case of Dirty ER, Figs. 13(a) and (b) depict a similar performance for Redundancy CNP. Its *RR* amounts to 0.235, rising precision by 1.3 times, on average, across all datasets and weighting schemes. Reciprocal CNP exhibits similar levels of *RR* as before, reducing the total cardinality by 77%. Yet, its impact on recall is much larger and its *RPC* drops to 0.805, on average. This means that the absolute *PC* falls below 0.8 for the datasets where CNP already had a low recall, namely R_{1D}, R_{4D} and R_{5D}. Apparently, this should be attributed to the more challenging conditions of Dirty ER.

Best approach. Similar to WNP, none of the new methods for CNP outperforms all others with respect to both recall and precision. Instead, the trade-off between *RR* and *RPC* is prevalent: the

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

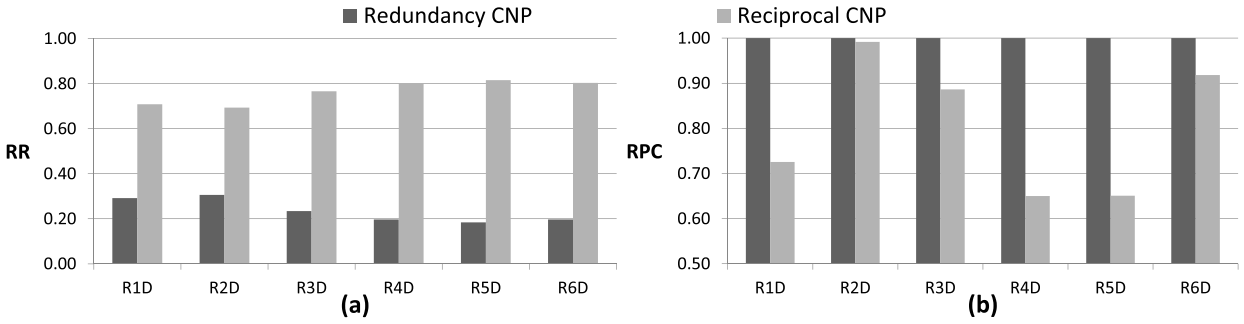

Fig. 13. (a) *RR* and (b) *RPC*, averaged across all weighting schemes, for the new methods for CNP over the real datasets for Dirty ER.

Table 4
Performance of the best cardinality-based, node-centric pruning algorithms over the real datasets, averaged across all weighting schemes.

	Clean–Clean ER						Dirty ER					
	R_{1C}	R_{2C}	R_{3C}	R_{4C}	R_{5C}	R_{6C}	R_{1D}	R_{2D}	R_{3D}	R_{4D}	R_{5D}	R_{6D}
$\|\|B'\|\|$	$7.82 \cdot 10^3$	$1.63 \cdot 10^4$	$8.87 \cdot 10^3$	$2.93 \cdot 10^4$	$2.62 \cdot 10^5$	$7.10 \cdot 10^6$	$3.34 \cdot 10^4$	$4.77 \cdot 10^4$	$5.87 \cdot 10^5$	$1.73 \cdot 10^5$	$1.40 \cdot 10^6$	$3.72 \cdot 10^7$
$PC(B')$	0.876	0.993	0.910	0.791	0.891	0.921	0.867	0.995	0.956	0.772	0.870	0.954
$PQ(B')$	0.121	0.135	0.237	0.032	0.079	0.119	0.028	0.046	0.004	0.005	0.014	0.023
	(a) Reciprocal CNP						(b) Redundancy CNP					

higher the former gets for one method, the lower is its *RPC* and vice versa. In this context, the best approach is the one that maximizes *RR* for a recall that consistently satisfies the requirements of efficiency-intensive applications ($PC \geq 0.80$).

For Clean–Clean ER, the best method in these terms is Reciprocal CNP. Its absolute performance is presented in Table 4(a). Despite the huge increase in precision, its recall remains above 0.8 across all datasets, except for R_{4C}, where it amounts to 0.791. This exception is caused by the poor performance of a single weighting scheme, namely CBS, for which $PC = 0.687$. For the other weighting schemes, *PC* remains well over the recall threshold. Compared to CEP in Table 2(a), Reciprocal CNP consistently achieves higher precision. For half the datasets, it outperforms CEP with respect to recall, as well.

In the case of Dirty ER, only Redundancy CNP satisfies the recall constraint of efficiency-intensive applications on a consistent basis. Its absolute performance is presented in Table 4(b). Similar to original CNP, its recall violates the relevant threshold only for the noisiest dataset, namely R_{4D}. Even in this case, it maintains a recall well over 0.8 for two of the weighting schemes, namely ARCS and ECBS. Compared to CEP, it consistently achieves a higher recall, while its precision is higher for four out of the six datasets.

Discussion Our experimental analysis indicates that the best methods for cardinality- and weight-based node-centric pruning are Reciprocal CNP and Partition WNP, respectively, in the case of Clean–Clean ER. For Dirty ER, Redundancy CNP and Reciprocal WNP should be preferred, respectively. These four methods yield significantly higher precision than the original pruning algorithms for CNP and WNP, while achieving similar recall on a consistent basis. In many cases, their performance is superior to their edge-centric counterparts, as well.

5.4. Block Filtering

To evaluate the performance of Block Filtering, we first fine-tune its parameter and examine the quality of the resulting restructured blocks. Then, we investigate its impact on the major pruning algorithms.

Fine-tuning the ratio As explained above, the *ratio* (r) of Block Filtering takes a value in the interval $[0, 1]$ that determines the portion of the most important blocks that are retained for each entity. To examine its effect on the functionality of Block Filtering, we measured the performance of the restructured block collections using all values of r in the interval $[0.05, 1]$ with a step of 0.05. Fig. 14 presents the resulting performance in terms of *RR* and *RPC* in comparison with the original blocks of R_{5C} and R_{5D}. The other datasets exhibit similar patterns and are omitted for brevity.

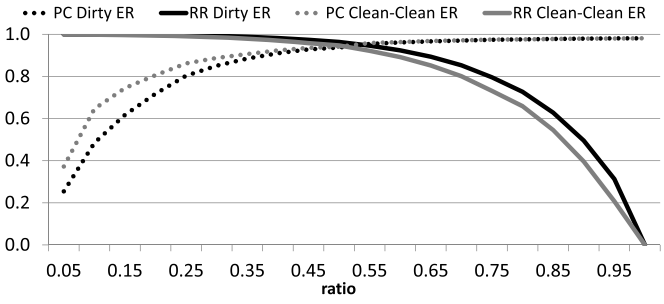

Fig. 14. The effect of parameter *ratio* on the performance of the blocks derived by Block Filtering from R_{5C} and R_{5D} in terms of *RR* and *RPC*.

We observe that there is a clear trade-off between *RR* and *RPC*: the smaller the value of r, the less blocks are retained for each entity and the lower is the total cardinality of the restructured blocks $||B'||$, thus increasing *RR*. Inevitably, this reduces the number of detected duplicates and decreases *PC*. The opposite is true for large values of r. Further, Block Filtering exhibits a robust performance with respect to r, with small variations in its value leading to small differences in *RR* and *RPC*.

To use Block Filtering as a pre-processing method, its ratio should be set to a value that increases precision at a low cost in recall. We quantify this constraint by requiring that r maintains *RPC* above 0.99, while maximizing *RR* and, thus, *PQ*; $r = 0.80$ satisfies this constraint across all real datasets.

Using this configuration, we estimated the performance of the restructured block collections and the corresponding blocking graphs over all real datasets. Table 5(a) presents it in absolute terms and Table 5(b) in relation to the original block collections. The latter employs two new metrics in order to measure the impact of Block Filtering on the blocking graph structure:

(i) Reduction in Nodes (*RN*) denotes the relative reduction in the order of the blocking graph and is formally defined as:

Table 5
(a) Technical characteristics of the restructured block collections of the real datasets after applying Block Filtering with $r = 0.80$, and (b) their relative performance with respect to the original block collections in Tables 1(a) and (b).

	Clean–Clean ER						Dirty ER					
	R_{1C}	R_{2C}	R_{3C}	R_{4C}	R_{5C}	R_{6C}	R_{1D}	R_{2D}	R_{3D}	R_{4D}	R_{5D}	R_{6D}
$\|B'\|$	2,007	6,817	6,863	6,649	40,724	1,239,356	3,750	7,570	44,138	10,063	76,343	1,499,462
$\|\|B'\|\|$	$8.52 \cdot 10^4$	$1.63 \cdot 10^5$	$8.84 \cdot 10^5$	$1.16 \cdot 10^6$	$3.12 \cdot 10^7$	$1.49 \cdot 10^{10}$	$6.08 \cdot 10^5$	$6.68 \cdot 10^5$	$5.91 \cdot 10^7$	$6.00 \cdot 10^6$	$1.49 \cdot 10^8$	$2.68 \cdot 10^{10}$
$PC(B')$	0.975	0.998	0.992	0.996	0.978	0.999	0.982	0.999	0.997	0.999	0.977	0.998
$PQ(B')$	0.012	0.014	0.003	0.001	0.001	$5.99 \cdot 10^{-5}$	0.002	0.003	$3.90 \cdot 10^{-5}$	$1.84 \cdot 10^{-4}$	$1.50 \cdot 10^{-4}$	$3.32 \cdot 10^{-5}$
$\|V_{B'}\|$	2,149	4,907	60,935	4,391	50,720	3,331,647	2,151	4,910	63,864	4,393	50,769	3,333,356
$\|E_{B'}\|$	$6.53 \cdot 10^4$	$1.17 \cdot 10^5$	$8.48 \cdot 10^5$	$7.67 \cdot 10^5$	$2.84 \cdot 10^7$	$1.31 \cdot 10^{10}$	$3.44 \cdot 10^5$	$5.14 \cdot 10^5$	$5.06 \cdot 10^7$	$2.96 \cdot 10^6$	$1.07 \cdot 10^8$	$2.33 \cdot 10^{10}$

(a)

$RR(B, B')$	0.472	0.565	0.540	0.575	0.615	0.648	0.708	0.800	0.811	0.785	0.822	0.664
$RPC(B, B')$	0.991	0.999	0.998	0.997	0.997	1.000	0.992	0.999	0.998	0.999	0.995	0.999
$RN(B, B')$	0.000	0.000	0.005	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
$RE(B, B')$	0.440	0.593	0.537	0.440	0.579	0.633	0.645	0.811	0.806	0.590	0.719	0.649

(b)

$$RN(B, B') = 1 - |V_{B'}|/|V_B|,$$

where $|V_B|$ and $|V_{B'}|$ denote the order of the blocking graphs derived from the original and the restructured block collections (after applying Block Filtering), respectively.

(ii) Reduction in Edges (*RE*) stands for the relative reduction in the size of the blocking graph and is estimated as:

$$RE(B, B') = 1 - |E_{B'}|/|E_B|.$$

Both measures are defined in the interval $[0, 1]$, with higher values corresponding to deeper pruning.

We observe that the number of blocks is almost the same as in Table 1. Yet, their total cardinality is reduced to such an extent that *RR* exceeds 0.50 for most Clean–Clean ER datasets. In the case of Dirty ER, *RR* takes even higher values, exceeding 0.70 across all datasets.[12] Given that Block Filtering reduces recall to a negligible extent ($RPC > 0.99$), precision rises between 2 and 3 times for Clean–Clean ER and between 3 and 6 times for Dirty ER. Regarding the blocking graph, there is no significant reduction on its nodes, since $RN(B, B')$ is practically 0 across all datasets. In contrast, its edges are reduced to a considerable extent, with $RE(B, B')$ approaching *RR* in most cases.

Effect on the main pruning algorithms We now examine the performance of the main pruning algorithms on top of Block Filtering using the usual measures, *RR* and *RPC*. This time they are defined as: $RR(B_{or}, B_{bf}) = 1 - ||B_{bf}||/||B_{or}||$, and $RPC(B_{or}, B_{bf}) = PC(B_{bf})/PC(B_{or})$, where B_{or} stands for the blocks produced by applying the pruning algorithm to the original block collection that is described in Tables 1(a) or (b), and B_{bf} denotes the blocks produced by applying the same pruning algorithm to the restructured blocks of Block Filtering (Table 5(a)). Fig. 15 presents the outcomes over the Clean–Clean and the Dirty ER datasets. The left diagrams pertains to *RR* and the right ones to *RPC*, with the two measures averaged across all weighting schemes.

These diagrams demonstrate that the effect of Block Filtering is determined by the type of the pruning threshold. For cardinality-based thresholds, it conveys a moderate decrease in total cardinality, with *RR* fluctuating between 0.15 and 0.25 across all datasets for CEP and CNP variants. The reason is that these thresholds do not depend on the structure of the blocking graph, but are proportional to the average blocks per entity in B. Given that the ratio of Block Filtering determines the number of retained blocks per entity, these thresholds are reduced by at most 20% for $r = 0.8$. The impact of Block Filtering on the recall of these thresholds is either minor, with $RPC \gg 0.95$, or beneficial, with $RPC > 1$. The latter case appears in half the datasets and indicates that more duplicates are identified by the same pruning scheme when applied to the restructured blocks, because Block Filtering cleans them from noisy edges.

Quite different is the impact on the weight-based pruning thresholds. In this case, Block Filtering reduces the number of retained comparisons to a large extent; *RR* is analogous to $RE(B, B')$ in Table 5(b) and consistently exceeds 0.45 and 0.59 for the Clean–Clean and the Dirty ER datasets, respectively. This applies equally to WEP and WNP variants, because they all use the average edge weight as pruning threshold and, thus, they depend directly on the size of the blocking graph. With respect to recall, *RPC* never exceeds 1, but fluctuates between 0.95 and 0.99. This indicates an affordable reduction in recall, which is caused by two factors: first, Block Filtering discards some matching edges itself, and second, it reduces the extend of co-occurrence for a small part of the matching entities, thus lowering the weight of their edges.

The absolute performance of the main pruning algorithms is presented in Table 6. We observe that CEP consistently achieves higher precision and retains a lower number of comparisons than its original performance in Table 2. On average, across all datasets and weighting schemes, its *RR* is 0.178 for both Clean–Clean and Dirty ER, whereas its precision rises by 1.2 and 1.4 times, respectively. Most notably, its recall increases for the 3 largest datasets of Clean–Clean ER and for four datasets of Dirty ER. In these 7 cases, the restructured blocks of Block Filtering outperform the original ones with respect to precision and recall, due to the lower portion of noisy edges.

Similar patterns are exhibited by the methods for CNP, when comparing their new performance in Table 6(b) with the original one in Table 4. The total cardinality of the restructured blocks is reduced by 16% and 20%, on average, for Clean–Clean and Dirty ER datasets, respectively. At the same time, their precision increases by 1.2 and 1.3 times, respectively. Most importantly, their *PC* over R_{4C} and R_{4D} now exceeds 0.8 and, thus, both algorithms satisfy the recall requirements of efficiency-intensive applications across all datasets.

A different behavior is exhibited by the weight-based pruning algorithms. For WEP, the lower the original *PC* in Table 2(c) is, the lower is *RPC* in Figs. 15(b) and (d). As a result, WEP now violates the recall threshold of effectiveness-intensive applications for two Clean–Clean and two Dirty ER datasets – originally, this applied to just one dataset for each task. This is accompanied by significant

[12] To understand this discrepancy, consider a block that contains 5 entities from E_1 and 5 from E_2; it involves $(5 \cdot 5 =)25$ comparisons in the case of Clean–Clean ER and $(10 \cdot 9/2 =)45$ in the case of Dirty ER. If Block Filtering removes one of the entities, we have 20 ($RR = 0.25$) and 27 ($RR = 0.40$) comparisons for Clean–Clean and Dirty ER, respectively.

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

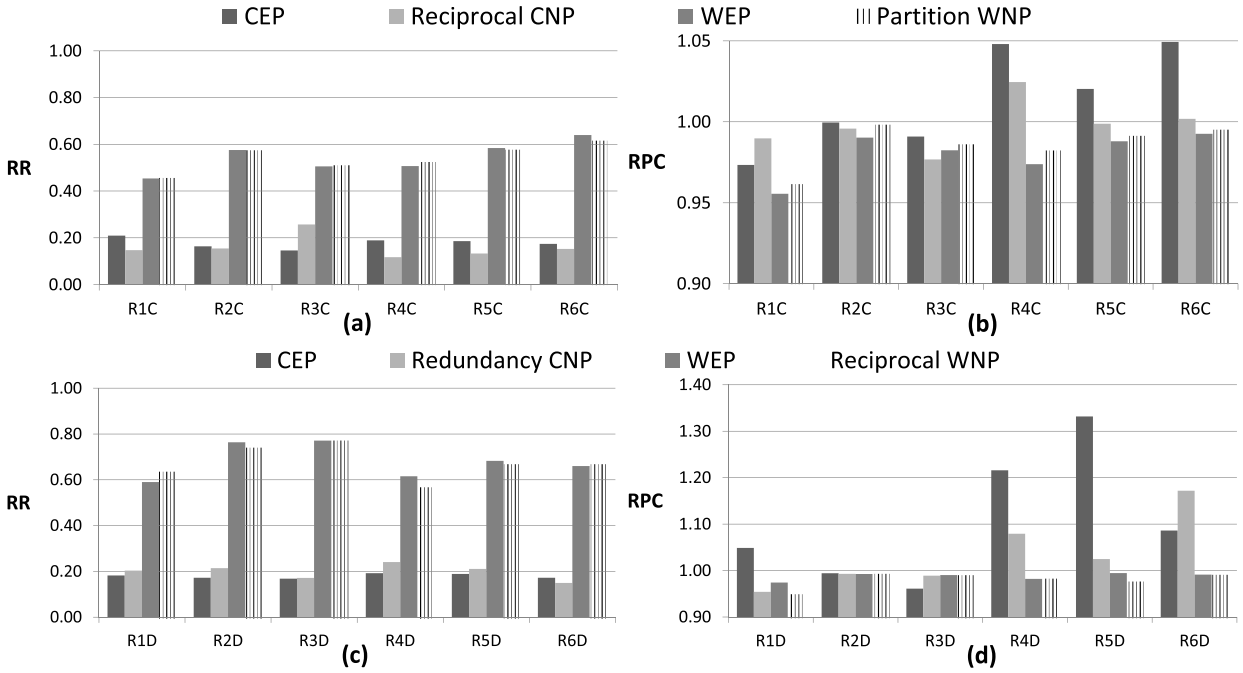

Fig. 15. The effect of Block Filtering on the performance of the main pruning algorithms over the real datasets for Clean–Clean ER (a–b) and Dirty ER (c–d) with respect to *RR* and *RPC*, averaged across all weighting schemes.

Table 6
Performance of the main pruning algorithms, averaged across all weighting schemes, on top of Block Filtering over the real datasets.

	Clean–Clean ER						Dirty ER					
	R_{1C}	R_{2C}	R_{3C}	R_{4C}	R_{5C}	R_{6C}	R_{1D}	R_{2D}	R_{3D}	R_{4D}	R_{5D}	R_{6D}
$\|\|B''\|\|$	$1.09 \cdot 10^4$	$2.28 \cdot 10^4$	$1.22 \cdot 10^5$	$6.11 \cdot 10^4$	$5.82 \cdot 10^5$	$2.18 \cdot 10^7$	$1.94 \cdot 10^4$	$2.86 \cdot 10^4$	$3.36 \cdot 10^5$	$8.73 \cdot 10^4$	$7.19 \cdot 10^5$	$2.04 \cdot 10^7$
$PC(B'')$	0.878	0.987	0.959	0.838	0.882	0.760	0.644	0.979	0.782	0.599	0.628	0.581
$PQ(B'')$	0.087	0.096	0.018	0.015	0.035	0.031	0.036	0.076	0.005	0.008	0.020	0.025
	(a) Cardinality Edge Pruning											
$\|\|B''\|\|$	$6.67 \cdot 10^3$	$1.38 \cdot 10^4$	$6.59 \cdot 10^3$	$2.58 \cdot 10^4$	$2.31 \cdot 10^5$	$6.02 \cdot 10^6$	$2.66 \cdot 10^4$	$3.75 \cdot 10^4$	$4.86 \cdot 10^5$	$1.31 \cdot 10^5$	$1.11 \cdot 10^6$	$3.16 \cdot 10^7$
$PC(B'')$	0.867	0.989	0.889	0.810	0.890	0.922	0.828	0.988	0.945	0.833	0.892	0.952
$PQ(B'')$	0.140	0.159	0.312	0.036	0.096	0.138	0.033	0.059	0.005	0.007	0.019	0.027
	(b-i) Reciprocal CNP						(b-ii) Redundancy CNP					
$\|\|B''\|\|$	$1.43 \cdot 10^4$	$2.24 \cdot 10^4$	$2.13 \cdot 10^5$	$2.16 \cdot 10^5$	$6.17 \cdot 10^6$	$2.40 \cdot 10^9$	$9.81 \cdot 10^4$	$1.27 \cdot 10^5$	$1.10 \cdot 10^7$	$8.33 \cdot 10^5$	$3.27 \cdot 10^7$	$4.06 \cdot 10^9$
$PC(B'')$	0.891	0.980	0.960	0.938	0.951	0.969	0.928	0.991	0.985	0.935	0.947	0.965
$PQ(B'')$	0.075	0.106	0.026	0.006	0.006	0.001	0.013	0.021	$3.61 \cdot 10^{-4}$	$1.70 \cdot 10^{-3}$	$1.27 \cdot 10^{-3}$	$7.04 \cdot 10^{-4}$
	(c) Weighted Edge Pruning											
$\|\|B''\|\|$	$1.54 \cdot 10^4$	$2.35 \cdot 10^4$	$2.19 \cdot 10^5$	$1.94 \cdot 10^5$	$4.95 \cdot 10^6$	$3.13 \cdot 10^9$	$7.82 \cdot 10^4$	$1.06 \cdot 10^5$	$8.42 \cdot 10^6$	$7.13 \cdot 10^5$	$2.90 \cdot 10^7$	$2.95 \cdot 10^9$
$PC(B'')$	0.916	0.994	0.967	0.958	0.956	0.986	0.907	0.994	0.987	0.912	0.935	0.973
$PQ(B'')$	0.069	0.098	0.022	0.006	0.006	0.001	0.015	0.025	$4.25 \cdot 10^{-4}$	$2.08 \cdot 10^{-3}$	$1.40 \cdot 10^{-3}$	$7.27 \cdot 10^{-4}$
	(d-i) Partition WNP						(d-ii) Reciprocal WNP					

increase in efficiency: *PQ* rises by 2.1 and 2.5 times, respectively, while $RR = 0.612$, on average.

Similar patterns are observed when comparing the new performance of WNP variants with their original one in Table 3. Again, the reduction in recall is higher for the datasets with lower original *PC*. Yet, Partition WNP violates the recall threshold only for the smallest dataset of Clean–Clean ER. Its robustness to recall allows it to reduce the total cardinality by 54% and to increase precision by 2.1 times, on average. For Dirty ER, the precision of Reciprocal WNP increases to a larger extent than all other pruning algorithms: on average, $RR = 0.678$ and *PQ* rises by 2.5 times. On the flip side, its average recall falls below 0.95 for three datasets. In every case, though, at least one of the pruning schemes maintains the recall higher than this threshold.

Discussion Our experiments demonstrate that Block Filtering trades significantly higher precision for slightly lower recall. In this way, it enhances the performance not only of the block collections, but of the main pruning algorithms, as well. Therefore, it constitutes an indispensable pre-processing step for graph-based Meta-blocking, as depicted in Fig. 9(a).

5.5. Best configurations for Meta-blocking

In this section, we identify the best configurations of Meta-blocking on top of Block Filtering for Clean–Clean and Dirty ER over the effectiveness- and the efficiency-intensive applications. We begin with selecting the best pruning algorithms and continue with choosing the best weighting schemes.

Best Pruning Algorithms Looking at the numbers of Table 6, we observe that there is a straightforward choice for efficiency-intensive applications over Dirty ER datasets. Redundancy CNP consistently satisfies their requirements on recall ($PC > 0.80$), whereas CEP

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

falls short of recall across all datasets – except for R_{2D}. Therefore, the former algorithm poses the best option in this case.

In the case of Clean–Clean ER, Reciprocal CNP consistently exceeds the recall threshold of efficiency-intensive applications, while CEP violates it for the largest dataset, R_{6C}. The former algorithm also achieves the highest recall for half of the datasets. It also retains 61% less comparisons, on average, and outperforms CEP with respect to precision across all datasets. Thus, we can safely conclude that Reciprocal CNP offers the best choice for efficiency-intensive Clean–Clean ER applications.

For the effectiveness-intensive applications over Clean–Clean ER, Partition WNP consistently achieves higher recall than WEP. It violates the recall constraint ($PC > 0.95$) only for the smallest dataset, while WEP does so for R_{1C} and R_{4C}. The reason is that the node-centric functionality of Partition WNP is more robust to recall and retains more comparisons than WEP across most datasets. There are two exceptions, R_{4C} and R_{5C}, where it outperforms WEP with respect to total cardinality $||B'||$ and precision, as well. On the whole, Partition WNP exhibits the best performance when requiring high effectiveness over Clean–Clean ER.

In the case of Dirty ER, there is no clear winner. WEP and Reciprocal WNP satisfy the recall requirements of the effectiveness-intensive applications for the same datasets: R_{2D}, R_{3D} and R_{6D}. For these datasets, the latter retains 22% less comparisons and achieves higher PC, thus outperforming WEP with respect to both recall and precision. For the other three datasets, Reciprocal WNP achieves higher precision, but lower recall than WEP. Both algorithms satisfy the recall constraint in combination with few of the weighting schemes. Based on this observation, we select between them by comparing the relative performance of their most robust weighting schemes. For both algorithms, only ECBS maintains a PC higher than 0.95 across the five largest datasets. In combination with Reciprocal WNP, it exhibits higher precision in four out of the five datasets. Thus, we opt for Reciprocal WNP as the best algorithm in this case.

In summary, our experiments advocate that two node-centric, cardinality-based pruning algorithms are the most appropriate for efficiency-intensive applications: Reciprocal CNP for Clean–Clean ER and Redundancy CNP for Dirty ER. In the case of effectiveness-intensive applications, two weight-based, node-centric algorithms perform the best pruning: Partition WNP for Clean–Clean ER and Reciprocal WNP for Dirty ER.

Best weighting schemes We now identify the best weighting scheme for the selected pruning algorithms. For each of them, we consider the performance of its combination with every weighting scheme in terms of PC and PQ. The outcomes of our experiments over the relevant real datasets are presented in Figs. 16(a) to (d).

Starting with Reciprocal CNP, we observe in Fig. 16 (a-ii) that CBS consistently achieves the highest precision across all datasets. However, its recall falls to 0.756 over R_{4C}, because of its unstable functionality; given that most pairs of entities share one or two blocks, CBS produces many ties and the corresponding edges are randomly ordered in the sorted stack of each node. When the cardinality threshold does not cover all ties, the impact on recall may be significant. Therefore, this scheme should not be applied to cardinality-based pruning algorithms. Among the other weighting schemes, ARCS consistently achieves the lowest recall, which falls below 0.8 over R_{4C}. Thus, it should be avoided, as well. The remaining three weighting schemes exhibit similar performance, with a recall well over 0.8 across all datasets. In the absence of a clear winner, we select JS as the best scheme, because it achieves the highest precision for half the datasets.

In the case of Redundancy CNP, we observe a totally different behavior, due to the different functionality of the algorithm. JS achieves the lowest recall in most cases, violating the recall constraint over R_{1D} and R_{4D}. EJS follows it in close distance and, thus, both schemes are rejected. The rest of the weighting schemes exceed the recall threshold across all datasets. Among them, ARCS exhibits the highest precision in four cases and lies close to the maximum PQ in the other two. Thus, we mark it as the best choice for Redundancy CNP over Dirty ER.

For Partition WNP, there is a clear trade-off between recall and precision among the weighting schemes: for each dataset, the scheme with the lowest recall achieves the highest precision, and vice versa. For R_{1C}, all weighting schemes violate the recall constraint. CBS and ARCS violate it one and two more times, respectively, and are rejected. Among the remaining weighting schemes, there is a clear alignment: we have ECBS $<$ JS $<$ EJS in terms of recall and vice versa for precision. Given that they all overcome the recall threshold across the five largest datasets, ECBS offers the best balance between PC and PQ.

For Reciprocal WNP, we have explained that ECBS is the only weighting scheme satisfying the recall requirements of effectiveness-intensive applications for the 5 largest datasets.

In summary, our experimental analysis indicates that for efficiency-intensive applications over Clean–Clean ER, we should apply Reciprocal CNP in conjunction with JS. For Dirty ER, the combination of Redundancy CNP with ARCS yields the best balance between recall and precision. The best configurations for the effectiveness-intensive applications is the combination of ECBS with Partition and Reciprocal WNP for Clean–Clean and Dirty ER, respectively. Collectively, these schemes are called *Enhanced Meta-blocking* in the following.

5.6. Graph-free Meta-blocking and baseline methods

We now compare Enhanced Meta-blocking with the current state-of-the-art. We also consider the performance of *Graph-free Meta-blocking*, which combines Block Filtering with Comparison Propagation, as in Fig. 9(b). We first examine the efficiency-intensive methods and then the effectiveness-intensive ones. We use the real datasets for examining the analytical performance of each method and the synthetic ones for performing a scalability analysis. We repeated the time measurements 10 times and considered the average values of *OTime* and *RTime* in order to minimize the effect of external parameters. For block collections with more than 10^9 comparisons, *RTime* was approximated using the average time required for comparing two entity profiles – 0.029 milliseconds for R_{1C}/R_{1D} and 0.009 milliseconds for S_6/S_7.

Efficiency-intensive methods We adapted Graph-free Meta-blocking to efficiency-intensive applications by identifying the ratio of Block Filtering that maximizes precision for a recall higher than 0.80 across all real datasets. This ratio is $r = 0.28$. As a baseline method, we employ the original CNP in combination with the same weighting schemes as Enhanced Meta-blocking, i.e., JS for Clean–Clean ER and ARCS for Dirty ER (these particular pruning schemes exhibit high performance in the experiments of [7], as well). The performance of these methods over the real datasets is presented in Table 7.

Starting with Graph-free Meta-blocking, we observe that it exhibits lower recall than CNP over Clean–Clean ER, but retains less comparisons and achieves higher precision. On average, it reduces PC by 6.9% and $||B'||$ by 64%, thus increasing PQ by 3.4 times. However, it executes an order of magnitude more comparisons for the largest dataset (R_{6C}), rising PC by 0.7% and decreasing PQ by 5.8 times. Similarly, it retains 47% more comparisons, on average, across the four largest datasets of Dirty ER. This brings about no significant change in recall, while precision decreases by 4.3 times. Yet, it consistently exhibits a far better time efficiency: on aver-

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

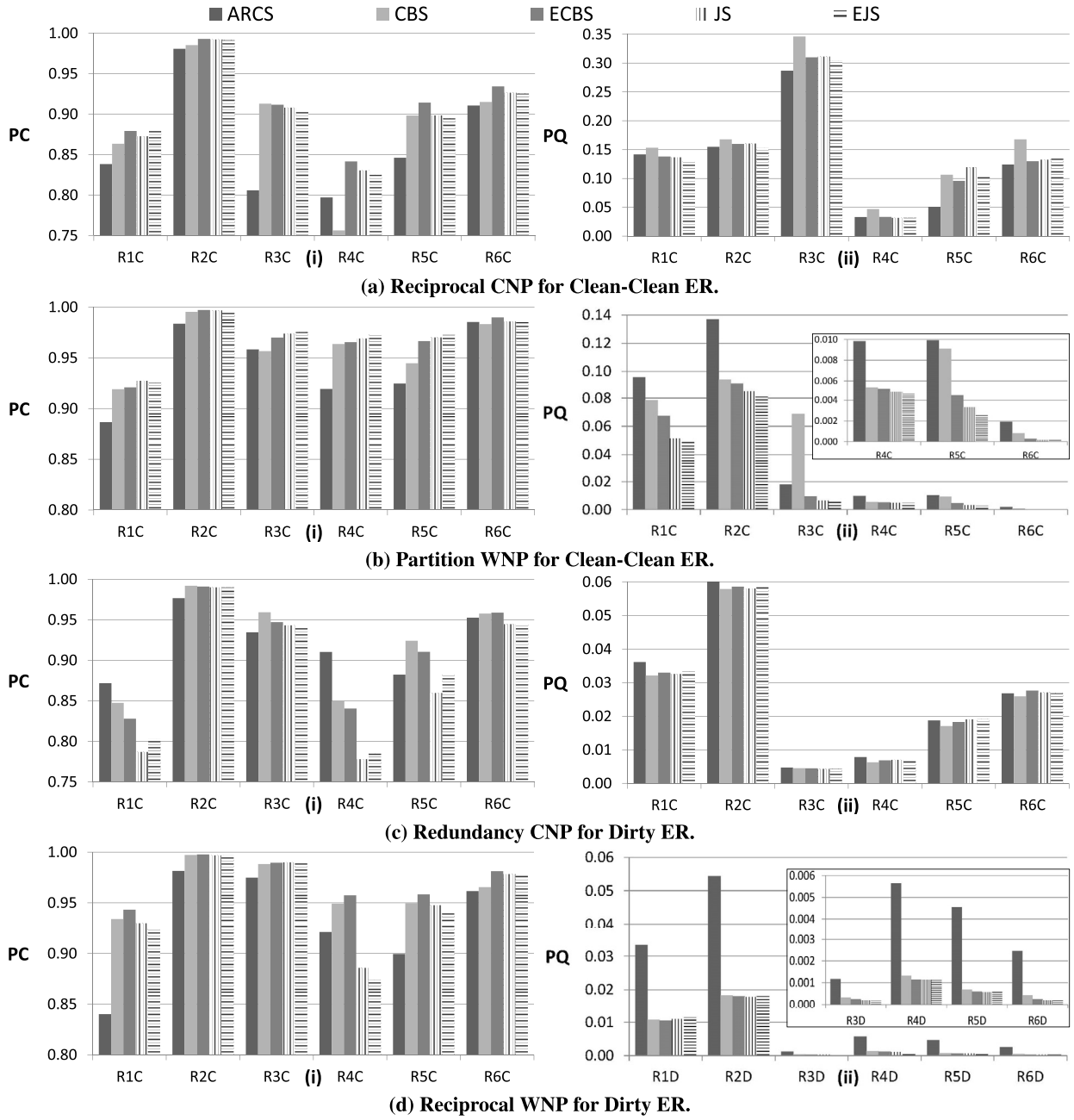

Fig. 16. Performance of all weighting schemes with respect to recall (*PC*) and precision (*PQ*) in combination with the best pruning algorithms across the relevant real datasets. The sub-figures in (b-ii) and (d-ii) zoom into smaller scales to highlight precision over the largest datasets.

age, it is faster than CNP by 3 orders of magnitude with respect to *OTime* and by 7 times with respect to *RTime*.

Comparing Enhanced Meta-blocking to CNP, we observe that it consistently trades lower recall for higher precision. For Clean–Clean ER, Reciprocal CNP sacrifices recall by 5.3% in order to prune 83% more comparisons and increase precision by 9.6 times, on average. For Dirty ER, Redundancy CNP sacrifices recall by 1.4% in order to prune 40% more comparisons and increase precision by 1.6 times. Nevertheless, Enhanced Meta-blocking satisfies the recall constraint in all cases. In terms of time efficiency, its average *OTime* is lower than CNP by 4 times, mainly due to the pre-processing of Block Filtering; its average *RTime* is lower by 5 times, due to the deeper pruning it performs.

Comparing Graph-free with Enhanced Meta-blocking, we observe negligible differences in recall. The latter, though, achieves significantly higher precision in most of the cases. On average, the *PQ* of Reciprocal CNP is higher by 12.3 times (Clean–Clean ER) and that of Redundancy CNP by 5.4 times (Dirty ER); there is a single exception in each case, where the graph-free approach achieves the highest *PQ*: R_{2C} and R_{2D}, respectively. Enhanced Meta-blocking also scales much better to the 4 largest datasets of each ER task, executing 80% and 68% less comparisons for Clean–Clean and Dirty ER, respectively.

The superiority of Enhanced Meta-blocking should be attributed to the finer granularity of its functionality: it operates on the level of comparisons (i.e., pairs of entities), whereas Block Filtering operates on the level of blocks and of individual entities. As a result, the pruning of the graph-based approach is more accurate at the cost of lower efficiency: on average, its *OTime* is higher by 2 orders of magnitude than Graph-free Meta-blocking. Its *RTime* is higher by 2.7 times over the Dirty ER datasets, despite the fewer comparisons it executes. This difference is caused

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

Table 7
Technical characteristics of the restructured block collections of the real datasets for the best efficiency-intensive methods: (a) stand-alone Block Filtering with $r = 0.28$, (b) Block Filtering with $r = 0.80$ in conjunction with Reciprocal CNP (JS) for Clean–Clean ER and Redundancy CNP (ARCS) for Dirty ER, and (c) CNP in combination with JS and ARCS for Clean–Clean and Dirty ER, respectively.

	Clean–Clean ER						Dirty ER					
	R_{1C}	R_{2C}	R_{3C}	R_{4C}	R_{5C}	R_{6C}	R_{1D}	R_{2D}	R_{3D}	R_{4D}	R_{5D}	R_{6D}
$\|\|B'\|\|$	$6.55 \cdot 10^3$	$8.32 \cdot 10^3$	$5.85 \cdot 10^4$	$6.28 \cdot 10^4$	$1.00 \cdot 10^6$	$2.92 \cdot 10^8$	$2.61 \cdot 10^4$	$2.40 \cdot 10^4$	$1.33 \cdot 10^6$	$2.17 \cdot 10^5$	$3.79 \cdot 10^6$	$5.68 \cdot 10^8$
$PC(B')$	0.806	0.962	0.912	0.866	0.884	0.977	0.818	0.968	0.948	0.901	0.865	0.970
$PQ(B')$	0.132	0.257	0.036	0.015	0.020	0.003	0.034	0.090	0.002	0.005	0.005	0.002
OTime	1 ms	3 ms	13 ms	16 ms	178 ms	43 s	5 ms	5 ms	169 ms	111 ms	1 s	62 s
RTime	102 ms	92 ms	593 ms	8 s	50 s	2.4 hrs	462 ms	262 ms	13 s	21 s	4 min	4.6 hrs
(a) Graph-free Meta-blocking												
$\|\|B'\|\|$	$6.86 \cdot 10^3$	$1.37 \cdot 10^4$	$6.73 \cdot 10^3$	$2.88 \cdot 10^4$	$2.33 \cdot 10^5$	$6.23 \cdot 10^6$	$2.59 \cdot 10^4$	$3.61 \cdot 10^4$	$4.56 \cdot 10^5$	$1.28 \cdot 10^5$	$1.08 \cdot 10^6$	$3.17 \cdot 10^7$
$PC(B')$	0.873	0.992	0.908	0.831	0.898	0.927	0.871	0.977	0.935	0.910	0.883	0.951
$PQ(B')$	0.137	0.161	0.312	0.032	0.088	0.133	0.036	0.060	0.005	0.008	0.019	0.027
OTime	23 ms	29 ms	158 ms	1 s	11 s	1.9 hrs	515 ms	477 ms	55 s	12 s	8 min	11.9 hrs
RTime	112 ms	169 ms	223 ms	2 s	18 s	2.0 hrs	1 s	1 s	60 s	24 s	9 min	12.1 hrs
(b) Enhanced Meta-blocking												
$\|\|B'\|\|$	$2.56 \cdot 10^4$	$5.36 \cdot 10^4$	$2.34 \cdot 10^5$	$1.48 \cdot 10^5$	$1.41 \cdot 10^6$	$4.95 \cdot 10^7$	$4.72 \cdot 10^4$	$6.87 \cdot 10^4$	$7.66 \cdot 10^5$	$2.15 \cdot 10^5$	$1.72 \cdot 10^6$	$4.63 \cdot 10^7$
$PC(B')$	0.952	0.998	0.976	0.890	0.944	0.970	0.894	0.987	0.940	0.929	0.899	0.955
$PQ(B')$	0.040	0.041	0.010	0.007	0.015	0.017	0.020	0.032	0.003	0.005	0.012	0.018
OTime	64 ms	142 ms	764 ms	3 s	74 s	15.9 hrs	1 s	2 s	4 min	27 s	22 min	35.8 hrs
RTime	405 ms	664 ms	3 s	9 s	2 min	16.3 hrs	2 s	3 s	4 min	45 s	24 min	36.2 hrs
(c) Original Meta-blocking												

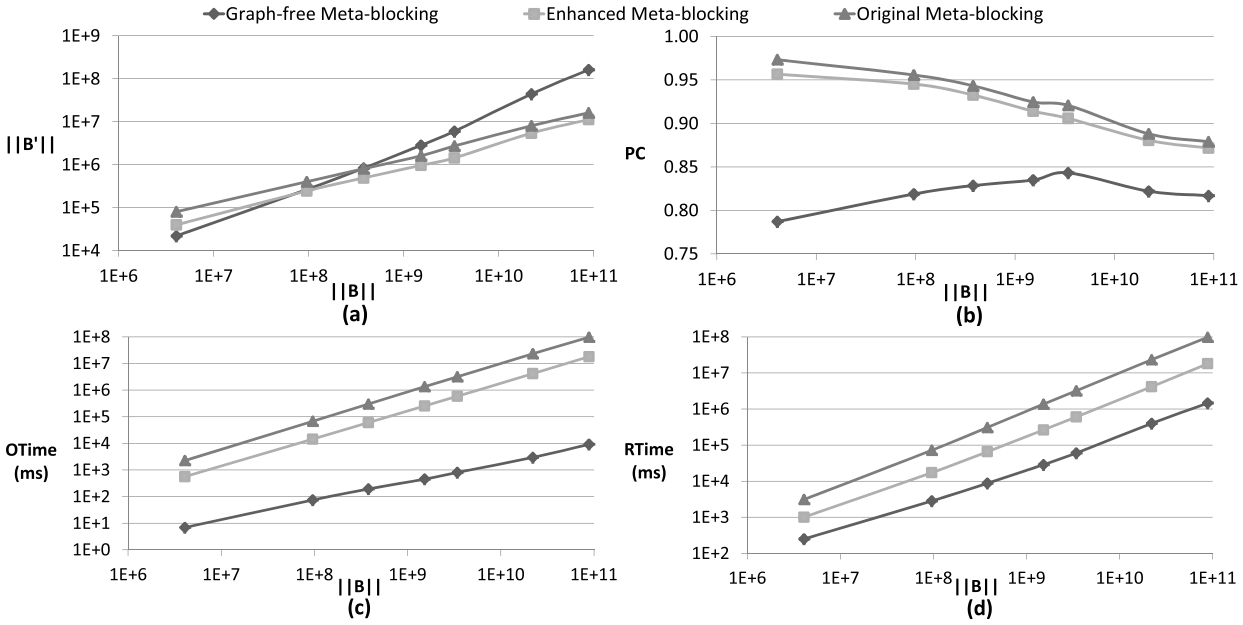

Fig. 17. Scalability analysis of Graph-free and Enhanced Meta-blocking for efficiency-intensive applications in comparison with the baseline method (CNP) over the synthetic datasets with respect to (a) total cardinality $||B'||$, (b) recall *PC*, (c) overhead time *OTime*, and (d) resolution time *RTime*. With the exception of the vertical axis in figure (b), all other axes are of logarithmic scale.

by the minimal overhead of the graph-free approach and would be reversed if we employed a more elaborate, time-consuming method for entity matching. For Clean–Clean ER, the results are mixed, as both methods achieve the best *RTime* for half the datasets.

These patterns are verified by the scalability analysis, which is presented in Fig. 17. Fig. 17(b) demonstrates that the recall of all methods consistently exceeds 0.80 – except for Graph-free Meta-blocking over the smallest dataset, S_1. In fact, the recall of this approach rises with the larger datasets from 0.79 to 0.82, while the recall of the graph-based methods drops steadily from 0.96 to 0.88. This is because the latter techniques scale much better to the largest datasets with respect to $||B'||$: the comparisons they retain rise by 2 orders of magnitude when moving from S_1 to S_7, while rising by 4 orders for Graph-free Meta-blocking. Note that all methods scale *sublinearly*, as the cardinality of the input block collections increases by 5 orders of magnitude from S_1 to S_7 (see Table 1(c)).

On the whole, Graph-free Meta-blocking retains more comparisons for lower recall, thus yielding the lowest precision across the five largest datasets. This should be attributed to its coarse-grained functionality. In contrast, Redundancy CNP consistently achieves the best precision: on average, its *PQ* is higher than CNP and the graph-free approach by 1.7 and 5.1 times, respectively. This is because it retains at least 40% less comparisons, while maintaining the second best recall across all datasets.

In terms of time efficiency, Figs. 17(c) and (d) show that Redundancy CNP is faster than CNP by 5 times, on average, with respect to both *OTime* and *RTime*. Yet, the most efficient method by far is Graph-free Meta-blocking; its *OTime* is lower than Redundancy

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

Table 8
Technical characteristics of the restructured block collections of the real datasets for the best effectiveness-intensive methods: (a) stand-alone Block Filtering with $r = 0.64$, (b) Block Filtering with $r = 0.80$ in conjunction with Partition WNP (ECBS) for Clean–Clean ER and Reciprocal WNP (ECBS) for Dirty ER, (c) WNP in combination with ECBS and (d) Iterative Blocking.

	Clean–Clean ER						Dirty ER					
	R_{1C}	R_{2C}	R_{3C}	R_{4C}	R_{5C}	R_{6C}	R_{1D}	R_{2D}	R_{3D}	R_{4D}	R_{5D}	R_{6D}
$\|\|B'\|\|$	$3.98 \cdot 10^4$	$5.68 \cdot 10^4$	$4.24 \cdot 10^5$	$4.34 \cdot 10^5$	$1.22 \cdot 10^7$	$5.33 \cdot 10^9$	$2.07 \cdot 10^5$	$2.42 \cdot 10^5$	$2.21 \cdot 10^7$	$1.85 \cdot 10^6$	$5.68 \cdot 10^7$	$9.34 \cdot 10^9$
$PC(B')$	0.951	0.997	0.984	0.977	0.970	0.997	0.971	0.998	0.996	0.991	0.971	0.996
$PQ(B')$	0.026	0.039	0.005	0.002	0.002	$1.67 \cdot 10^{-4}$	0.005	0.009	$1.04 \cdot 10^{-4}$	$5.91 \cdot 10^{-4}$	$3.90 \cdot 10^{-4}$	$9.52 \cdot 10^{-5}$
OTime	10 ms	10 ms	50 ms	262 ms	3 s	25 min	70 ms	38 ms	3 s	2 s	55 s	43 min
RTime	593 ms	600 ms	4 s	50 s	11 min	43 hrs	4 s	3 s	4 min	2 min	61 min	76 hrs
(a) Graph-free Meta-blocking												
$\|\|B'\|\|$	$1.47 \cdot 10^4$	$2.43 \cdot 10^4$	$2.38 \cdot 10^5$	$2.11 \cdot 10^5$	$4.93 \cdot 10^6$	$3.30 \cdot 10^9$	$9.62 \cdot 10^4$	$1.23 \cdot 10^5$	$8.99 \cdot 10^6$	$9.14 \cdot 10^5$	$3.70 \cdot 10^7$	$3.54 \cdot 10^9$
$PC(B')$	0.921	0.997	0.970	0.966	0.967	0.990	0.943	0.998	0.990	0.957	0.958	0.981
$PQ(B')$	0.067	0.091	0.009	0.005	0.004	$2.68 \cdot 10^{-4}$	0.011	0.018	$2.54 \cdot 10^{-4}$	$1.16 \cdot 10^{-3}$	$5.93 \cdot 10^{-4}$	$2.44 \cdot 10^{-4}$
OTime	23 ms	31 ms	219 ms	768 ms	13 s	3 hrs	577 ms	502 ms	63 s	14 s	8 min	12 hrs
RTime	240 ms	284 ms	2 s	29 s	4 min	29 hrs	2 s	2 s	3 min	62 s	49 min	41 hrs
(b) Enhanced Meta-blocking												
$\|\|B'\|\|$	$5.65 \cdot 10^4$	$1.11 \cdot 10^5$	$1.15 \cdot 10^6$	$9.63 \cdot 10^5$	$2.76 \cdot 10^7$	$1.55 \cdot 10^{10}$	$6.46 \cdot 10^5$	$9.77 \cdot 10^5$	$9.41 \cdot 10^7$	$5.54 \cdot 10^6$	$2.62 \cdot 10^8$	$2.93 \cdot 10^{10}$
$PC(B')$	0.964	0.998	0.987	0.990	0.976	0.998	0.977	1.000	0.996	0.995	0.976	0.995
$PQ(B')$	0.018	0.020	0.002	0.001	0.001	$5.73 \cdot 10^{-5}$	0.002	0.002	$2.44 \cdot 10^{-5}$	$1.98 \cdot 10^{-4}$	$8.53 \cdot 10^{-5}$	$3.03 \cdot 10^{-5}$
OTime	66 ms	149 ms	866 s	3 s	74 s	17 hrs	1 s	2 s	3 min	16 s	14 min	31 hrs
RTime	1 s	1 s	11 s	2 min	25 min	∼140 hrs	13 s	13 s	19 min	5 min	5 hrs	∼265 hrs
(c) Original Meta-blocking												
$\|\|B'\|\|$	$1.61 \cdot 10^4$	$5.33 \cdot 10^4$	$1.76 \cdot 10^6$	$1.87 \cdot 10^6$	$1.32 \cdot 10^7$	$2.34 \cdot 10^{10}$	$1.45 \cdot 10^6$	$1.37 \cdot 10^6$	$2.95 \cdot 10^8$	$2.31 \cdot 10^7$	$6.53 \cdot 10^8$	$4.81 \cdot 10^{10}$
$PC(B')$	0.983	0.999	0.994	0.999	0.980	0.999	0.991	1.000	0.999	1.000	0.982	0.999
$PQ(B')$	0.066	0.042	0.001	0.001	0.002	$3.81 \cdot 10^{-5}$	0.001	0.002	$7.82 \cdot 10^{-6}$	$4.78 \cdot 10^{-5}$	$3.44 \cdot 10^{-5}$	$1.85 \cdot 10^{-5}$
OTime	9 ms	11 ms	74 ms	101 ms	4 s	1.7 hrs	467 ms	454 ms	60 s	5 s	6 min	10 hrs
RTime	246 ms	579 ms	17 s	4 min	12 min	∼190 hrs	15 s	10 s	22 min	11 min	4 hrs	∼400 hrs
(d) Iterative Blocking												

CNP by 2 to 3 orders of magnitude, while its *RTime* is lower by 8 times, as it executes more comparisons.

In summary, we conclude that Enhanced Meta-blocking offers the best choice for efficiency-intensive applications, as it consistently achieves the highest precision among the three alternatives. It scales sublinearly to large datasets, but involves significant time and space requirements (still, they are much lower than the original CNP). For applications that cannot afford them, Graph-free Meta-blocking poses a very efficient alternative that requires minimum resources. For example, it was able to process the blocks of S_7 on a laptop with Intel Core i5 (1.9 GHz) and 8 GB RAM, running Windows 8.1 (64 bit) within 62 seconds (*OTime*).

Effectiveness-intensive methods We configured Graph-free Meta-blocking by setting the ratio of Block Filtering to the smallest value that ensures a recall higher than 0.95 across all real datasets: $r = 0.64$. As baseline methods, we consider Iterative Blocking and the original WNP in combination with the same weighting scheme as Enhanced Meta-blocking (ECBS). The functionality of the former was optimized by ordering the blocks in ascending order of cardinality (i.e., from the smallest to the largest block). Its functionality was further optimized for Clean–Clean ER by assuming that two matching entities are not compared to other co-occurring entities after their detection (apparently, this assumption corresponds to an ideal performance). The detailed performance of these methods is presented in Table 8.

Starting with Graph-free Meta-blocking, we observe that its *PC* is consistently lower than the baseline methods to a minor extent – less than 1%, on average. This small loss in recall suffices for significant gains in precision in comparison with both baseline methods: for Clean–Clean ER, it executes half as much comparisons, rising precision by 2.5 times; for Dirty ER, it retains at least 75% less comparisons, rising precision by more than 3.7 times. With respect to *OTime*, it is consistently faster than WNP by a whole order of magnitude. Compared to Iterative Blocking, it is faster by 9 times over Dirty ER, but the results are mixed over Clean–Clean ER, due to the optimized performance of the baseline. Finally, the resolution time of Graph-free Meta-blocking is lower than both baseline methods by 50% and 75% over Clean–Clean and Dirty ER, respectively.

Regarding Enhanced Meta-blocking, we observe that its pruning is deeper than both baseline methods, but violates the recall constraint only for the smallest dataset of each ER task (R_{1C},R_{1D}). On average, its *PC* is lower by 2%, allowing for massive gains in efficiency: $||B'||$ is reduced by 71% and 90%, while precision rises by 4.7 and 13.2 times for Clean–Clean and Dirty ER, respectively. In terms of time efficiency, Enhanced Meta-blocking outperforms WNP with respect to both *OTime* and *RTime*; the former is reduced by 63% and the latter by 81%, on average. Compared to Iterative Blocking, it involves a higher overhead, but reduces *RTime* by 63% and 86% for Clean–Clean and Dirty ER, respectively.

Juxtaposing Enhanced with Graph-free Meta-blocking, we notice a clear trade-off between precision and recall. The former approach emphasizes precision, based on its fine-grained functionality that operates on the level of pairwise comparisons, instead of individual entities. It retains 50% less comparisons at the cost of a 2% decrease in recall, on average. As a result, its precision is consistently higher by 2 times and its *RTime* lower by 49% and 36% for Clean–Clean and Dirty ER, respectively. The only advantage of Graph-free Meta-blocking is its minimal overhead; its *OTime* is lower than Enhanced Meta-blocking by 72% and 83% over Clean–Clean and Dirty ER, respectively.

These patterns are verified by our scalability analysis, which is presented in Fig. 18. Fig. 18(b) demonstrates that all methods consistently satisfy the recall constraint. The baseline methods achieve the highest $PC(> 0.99)$ across all datasets, followed by Graph-free Meta-blocking and then by Reciprocal WNP. The same ordering appears in Fig. 18(a), which shows a *linear increase* in the retained comparisons for all methods when moving from S_1 and S_7. In combination, these two figures show that Enhanced Meta-blocking consistently retains an order of magnitude less comparisons than the baseline methods for a 3.7% decrease in recall, on average. As

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

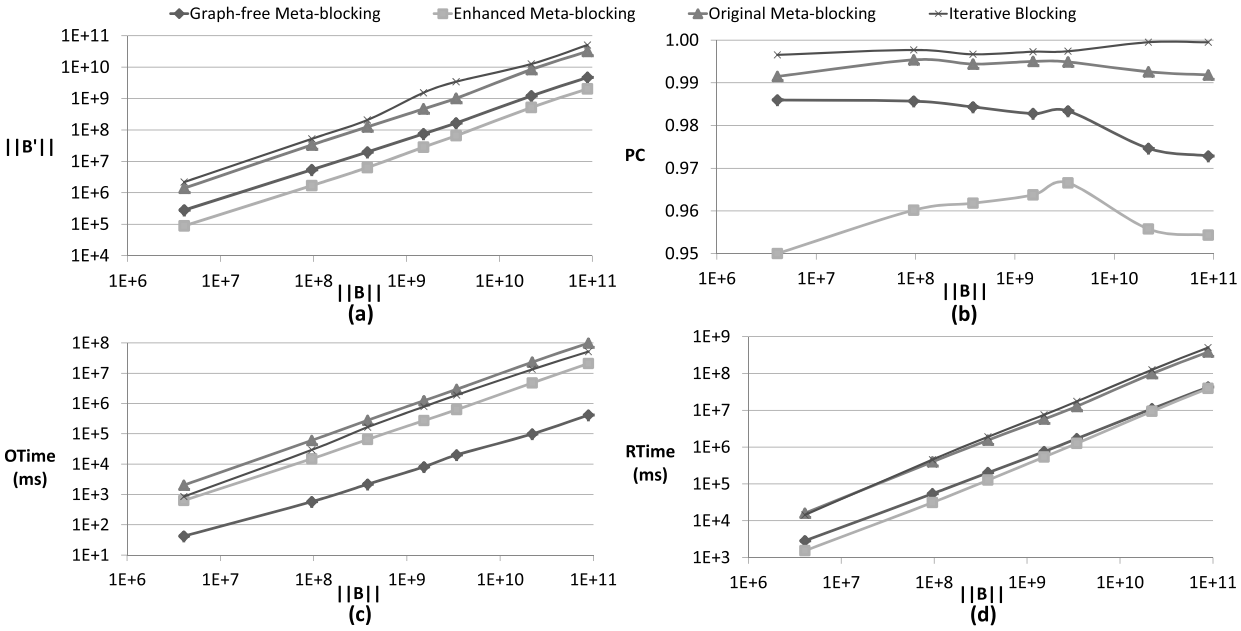

Fig. 18. Scalability analysis of Graph-free and Enhanced Meta-blocking for effectiveness-intensive applications in comparison with the baseline methods over the synthetic datasets with respect to (a) total cardinality $||B'||$, (b) recall *PC*, (c) overhead time *OTime*, and (d) resolution time *RTime*. With the exception of the vertical axis in figure (a), all other axes are of logarithmic scale.

a result, its precision is consistently higher than WNP and Iterative Blocking by 16.4 and 33.5 times, respectively. Graph-free Meta-blocking retains 88% less comparisons for a 1.4% decrease in recall, on average, and its precision surpasses that of WNP and Iterative Blocking by 6.1 and 12.6 times, respectively.

Regarding time efficiency, Fig. 18(c) demonstrates that Graph-free Meta-blocking is again the fastest technique by far. Its *OTime* is lower than all other methods by 2 orders of magnitude. The second fastest method is Enhanced Meta-blocking, whose *OTime* is lower than WNP and Iterative Blocking by 4.3 and 2.4 times, respectively. Due to its high precision, this distance increases to a whole order of magnitude in the case of *RTime*. The resolution time of Enhanced Meta-blocking is lower than Graph-free Meta-blocking, as well. However, their difference amounts to just 29%, on average, and decreases with larger datasets. This difference would be much larger in case we employed a more elaborate and time-consuming method for entity matching.

In short, the comparative analysis of effectiveness-intensive methods leads to similar conclusions as the efficiency-intensive ones. Enhanced Meta-blocking consistently exhibits the highest precision across all methods, while satisfying the recall constraint across most datasets. It scales linearly to large datasets, but involves significant space and time complexity. Graph-free Meta-blocking offers a reliable alternative that retains more comparisons, but achieves higher recall and requires minimum resources; it processed the blocks of S_7 on a laptop with Intel Core i5 (1.9 GHz) and 8 GB RAM, running Windows 8.1 (64 bit) within 26 minutes (*OTime*).

6. Conclusions

In this paper, we introduced three new node-centric pruning algorithms and compared them with the existing ones through an extensive experimental study. Redundancy Pruning does not affect recall, yet it saves around 30% more comparisons. Reciprocal Pruning decreases recall to a limited extend, but discards more than 66% additional comparisons. Graph Partitioning prunes 50% more comparisons for practically no impact on recall. The last method applies only to Clean–Clean ER, whereas the other two cover Dirty ER, as well. All of them increase the precision of existing techniques by 30% to 800%.

	Clean-Clean ER	Dirty ER
Efficiency-intensive applications (PC≥0.80)	Block Filtering (r=0.8) + Reciprocal CNP (JS)	Block Filtering (r=0.8) + Redundancy CNP (ARCS)
Effectiveness-intensive applications (PC≥0.95)	Block Filtering (r=0.8) + Partition WNP (ECBS)	Block Filtering (r=0.8) + Reciprocal WNP (ECBS)

Fig. 19. The best configurations of Enhanced Meta-blocking, as derived from the experimental evaluation, covering both Clean–Clean and Dirty ER, for efficiency- or effectiveness-intensive applications.

Combined with the pre-processing technique of Block Filtering, they raise precision by more than an order of magnitude. We experimentally derived the best configurations that maximize precision for the main types of ER applications and tasks. They are called Enhanced Meta-blocking and are summarized in Fig. 19. Our scalability analysis verified that they minimize the time and the space requirements of Meta-blocking even for entity collections with millions of entities and billions of comparisons. As a result, the overall resolution time improves almost by an order of magnitude.

An alternative approach for applications with limited resources is Graph-free Meta-blocking, which combines Block Filtering with Comparison Propagation. Its precision is lower than Enhanced Meta-blocking, but its lightweight functionality is able to process datasets with millions of entities within few minutes even on commodity hardware.

In the future, our goal is to adapt our approach to a parallelization framework, such as MapReduce, in order to minimize both the overhead of Meta-blocking and the cost of executing the retained comparisons. We also plan to adapt Enhanced Meta-blocking to Incremental Entity Resolution.

Acknowledgements

This work was partially supported by EU H2020 BigDataEurope (#644564) project.

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002

References

[1] P. Christen, A survey of indexing techniques for scalable record linkage and deduplication, IEEE Trans. Knowl. Data Eng. 24 (9) (2012) 1537–1555.
[2] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, W. Nejdl, A blocking framework for entity resolution in highly heterogeneous information spaces, IEEE Trans. Knowl. Data Eng. 25 (12) (2013) 2665–2682.
[3] B. Kenig, A. Gal, MFIBlocks: an effective blocking algorithm for entity resolution, Inf. Syst. 38 (6) (2013) 908–926.
[4] P. Christen, Data Matching, Data-Centric Systems and Applications, Springer, 2012.
[5] A. Elmagarmid, P. Ipeirotis, V. Verykios, Duplicate record detection: a survey, IEEE Trans. Knowl. Data Eng. 19 (1) (2007) 1–16.
[6] J. Madhavan, S. Cohen, X.L. Dong, A.Y. Halevy, S.R. Jeffery, D. Ko, C. Yu, Web-scale data integration: you can afford to pay as you go, in: CIDR, 2007, pp. 342–350.
[7] G. Papadakis, G. Koutrika, T. Palpanas, W. Nejdl, Meta-blocking: taking entity resolution to the next level, IEEE Trans. Knowl. Data Eng. 26 (8) (2014) 1946–1960, http://dx.doi.org/10.1109/TKDE.2013.54.
[8] Giovanni Simonini, Sonia Bergamaschi, H.V. Jagadish, BLAST: a loosely schema-aware meta-blocking approach for entity resolution, PVLDB 9 (12) (2016) 1173–1184, http://www.vldb.org/pvldb/vol9/p1173-simonini.pdf.
[9] G. Papadakis, E. Ioannou, C. Niederée, P. Fankhauser, Efficient entity resolution for large heterogeneous information spaces, in: WSDM, 2011, pp. 535–544.
[10] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, W. Nejdl, Eliminating the redundancy in blocking-based entity resolution methods, in: JCDL, 2011, pp. 85–94.
[11] S.E. Whang, D. Menestrina, G. Koutrika, M. Theobald, H. Garcia-Molina, Entity resolution with iterative blocking, in: SIGMOD, 2009, pp. 219–232.
[12] I. Fellegi, A. Sunter, A theory for record linkage, J. Am. Stat. Assoc. (1969) 1183–1210.
[13] A.N. Aizawa, K. Oyama, A fast linkage detection scheme for multi-source information integration, in: WIRI, 2005, pp. 30–39.
[14] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, D. Srivastava, Approximate string joins in a database (almost) for free, in: VLDB, 2001, pp. 491–500.
[15] A. McCallum, K. Nigam, L. Ungar, Efficient clustering of high-dimensional data sets with application to reference matching, in: KDD, 2000, pp. 169–178.
[16] M. Hernández, S. Stolfo, The merge/purge problem for large databases, in: SIGMOD, 1995, pp. 127–138.
[17] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, W. Nejdl, Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data, in: WSDM, 2012, pp. 53–62.
[18] G. Papadakis, G. Papastefanatos, G. Koutrika, Supervised meta-blocking, Proc. VLDB Endow. 7 (14) (2014) 1929–1940.
[19] S.E. Whang, D. Marmaros, H. Garcia-Molina, Pay-as-you-go entity resolution, IEEE Trans. Knowl. Data Eng. 25 (5) (2013) 1111–1124.
[20] M.J. Welch, A. Sane, C. Drome, Fast and accurate incremental entity resolution relative to an entity knowledge base, in: CIKM, 2012, pp. 2667–2670.
[21] G. Demartini, D.E. Difallah, P. Cudré-Mauroux, Large-scale linked data integration using probabilistic reasoning and crowdsourcing, VLDB J. 22 (5) (2013) 665–687.
[22] Y. Altowim, D.V. Kalashnikov, S. Mehrotra, Progressive approach to relational entity resolution, Proc. VLDB Endow. 7 (11) (2014) 999–1010.
[23] F. Hüffner, Algorithm engineering for optimal graph bipartization, J. Graph Algorithms Appl. 13 (2) (2009) 77–98.
[24] H. Köpcke, A. Thor, E. Rahm, Evaluation of entity resolution approaches on real-world match problems, Proc. VLDB Endow. 3 (1) (2010) 484–493.
[25] A. Thor, E. Rahm, MOMA – a mapping-based object matching system, in: CIDR, 2007, pp. 247–258.
[26] P. Christen, A. Pudjijono, Accurate synthetic generation of realistic personal information, in: PAKDD, 2009, pp. 507–514.

Further reading

[27] I. Ermilov, M. Martin, J. Lehmann, S. Auer, Linked open data statistics: collection and exploitation, in: KESW, 2013, pp. 242–249.

Please cite this article in press as: G. Papadakis et al., Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Big Data Res. (2016), http://dx.doi.org/10.1016/j.bdr.2016.08.002