# Meta-Blocking: Taking Entity Resolution to the Next Level

George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl

**Abstract**—Entity Resolution is an inherently quadratic task that typically scales to large data collections through blocking. In the context of highly heterogeneous information spaces, blocking methods rely on redundancy in order to ensure high effectiveness at the cost of lower efficiency (i.e., more comparisons). This effect is partially ameliorated by coarse-grained block processing techniques that discard entire blocks either a-priori or during the resolution process. In this paper, we introduce meta-blocking as a generic procedure that intervenes between the creation and the processing of blocks, transforming an initial set of blocks into a new one with substantially fewer comparisons and equally high effectiveness. In essence, meta-blocking aims at extracting the most similar pairs of entities by leveraging the information that is encapsulated in the block-to-entity relationships. To this end, it first builds an abstract graph representation of the original set of blocks, with the nodes corresponding to entity profiles and the edges connecting the co-occurring ones. During the creation of this structure all redundant comparisons are discarded, while the superfluous ones can be removed by pruning of the edges with the lowest weight. We analytically examine both procedures, proposing a multitude of edge weighting schemes, graph pruning algorithms as well as pruning criteria. Our approaches are schema-agnostic, thus accommodating any type of blocks. We evaluate their performance through a thorough experimental study over three large-scale, real-world datasets, with the outcomes verifying significant efficiency enhancements at a negligible cost in effectiveness.

**Index Terms**—Entity Resolution, Redundancy-positive Blocking, Meta-blocking

## 1 Introduction

*Entity resolution* (ER) is the task of identifying the same real-world object across different entity profiles. It constitutes an inherently quadratic process, as it requires every entity profile to be compared with all others. Therefore, it typically scales to large data collections through approximate methods that trade off effectiveness (i.e., percentage of detected duplicates) for efficiency (i.e., number of executed pair-wise comparisons). *Data blocking* [12], the most popular of these methods, groups similar entity profiles into *blocks* and exclusively performs the comparisons within each block. Blocking methods are generally distinguished in two categories: those forming non-overlapping blocks (i.e., *redundancy-free*), and those placing every entity profile into multiple blocks (i.e., *redundancy-bearing*).

Redundancy constitutes an indispensable and reliable means of reducing the likelihood of missed matches in the context of highly heterogeneous information spaces (HHIS), such as the Web of Data [4] and Dataspaces [15]. The reason is that HHIS involve extremely large volumes of data, high levels of noise, and loose schema binding. Though beneficial for effectiveness, redundancy comes at the cost of lower efficiency, as it increases the number of required pair-wise comparisons. In this work, we investigate ways of compensating for its effect on efficiency without sacrificing its high effectiveness.

**Motivating Examples.** As an example, consider the entity collection presented in Figure 1(a), where the entity profiles $p_1$ and $p_2$ describe the same real-world objects as profiles $p_3$ and $p_4$, respectively. Although the values of the duplicate profiles are relatively similar, every canonical attribute name has a different form in each of them; the name of a person, for instance, appears as “FullName” in $p_1$, as “name” in $p_2$ and as “full name” in $p_3$. This situation is further aggravated by the tag-style values; e.g., the name of person $p_4$ is not associated with any attribute value. In these settings, redundancy-free blocking methods can only be applied on top of a schema matching method that maps all entity profiles into a canonical schema with attributes of a-priori known quality. However, although schema matching seems straightforward in our example, it is not practical in large-scale collections of user-generated data: Google Base[1] alone encompasses 100,000 distinct schemata corresponding to 10,000 entity types [19]. Thus, in this work we exclusively consider redundancy-bearing blocking methods and aim at improving their efficiency.

Not all of these methods, though, share the same interpretation of redundancy. For the *redundancy-positive* blocking techniques, the number of common blocks between a pair of entity profiles is proportional to their similarity and, thus, the likelihood that they are matching. In this category fall methods that associate each profile with multiple blocking keys, such as q-grams [14], Suffix Array [1], [7], HARRA [17] and schema-agnostic blocking [25], [27]. To illustrate their functionality, consider Figure 1(b), which depicts the blocks that are produced after applying the

- *G. Papadakis is with the National Technical University of Athens, Greece and the L3S Research Center, Germany. E-mail: papadakis@L3S.de, gpapadis@mail.ntua.gr*
- *G. Koutrika is with HP Labs, USA. E-mail: koutrika@hp.com*
- *T. Palpanas is with the University of Trento, Italy. E-mail: themis@disi.unitn.eu*
- *W. Nejdl is with the L3S Research Center, Leibniz University of Hanover, Germany. E-mail: nejdl@L3S.de*

1. http://www.google.com/base

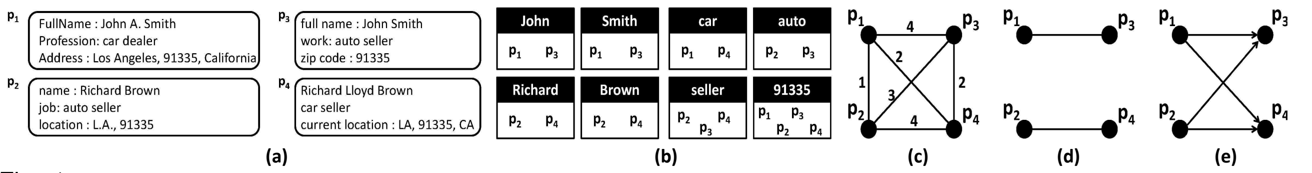


Fig. 1. (a) A noisy, heterogeneous entity collection, (b) the resulting set of attribute-agnostic blocks, (c) the blocking graph corresponding to it, (d) the pruned blocking graph, and (e) an alternative pruned blocking graph, discussed in Section 3.4.

simplest form of schema-agnostic blocking to the entity collection of Figure 1(a). Each block corresponds to a distinct token that has been extracted from at least one attribute value, regardless of the associated attribute name(s). Thus, the more blocks two entity profiles share, the more likely they are to describe the same real-world object.

In contrast, *redundancy-negative* blocking methods regard the high number of common blocks among two entity profiles as a strong indication that they are unlikely to be matching. For them, highly similar profiles share just one block. A typical example of this functionality is Canopy Clustering [21]: after selecting a random seed $p_i$, the most similar profiles are placed in the same block with $p_i$ and are removed from the pool of candidate matches; thus, they cannot be included in any other block.

In the middle of these two extremes lie *redundancy-neutral* blocking methods, which involve the same number of common blocks across all pairs of entity profiles (e.g., Sorted Neighborhood [16]). In this category also fall methods that are not suitable for drawing conclusions about the matching likelihood of two profiles from the blocks they have in common (e.g., Semantic Blocking [23]).

We observe that redundancy-negative and redundancy-neutral blocking methods are not applicable to HHIS. For example, even though Canopy Clustering and the Sorted Neighborhood approaches are scalable to large entity collections, they require an a-priori known schema in order to create blocks. The same applies to other related methods, such as the Adaptive Sorted Neighborhood [31] and the Sorted Blocks approach [10]. In contrast, the redundancy-positive techniques have been shown to apply to HHIS and scale to millions of entity profiles [17], [25], [27]. Therefore, our work focuses on improving the efficiency of redundancy-positive blocking methods by discarding the unnecessary comparisons of their blocks. In general, comparisons of this kind are distinguished into two categories: (*i*) the *redundant* ones, which repeatedly compare the same entities across different blocks, and (*ii*) the *superfluous* ones, which involve non-matching entities. Continuing our example, we can observe that the blocks of Figure 1(b) involve 9 redundant comparisons in the blocks "Smith", "Brown", "seller" and "91335". They also involve 6 superfluous comparisons between all possible pairs of non-matching entities in the blocks "car", "auto", "seller" and "91335". Skipping comparisons of these types increases blocking efficiency without affecting effectiveness.

Existing block processing techniques enhance the efficiency of redundancy-positive blocking methods mainly by operating at the coarse level of entire blocks. For example, Block Purging [25] a-priori discards *oversized blocks*, which involve an excessively high number of unnecessary comparisons. To illustrate this notion, consider the block of "91335" in Figure 1(b): it contains all possible comparisons of the entity profiles in Figure 1(a) and the only non-redundant comparisons it involves are superfluous. A similar technique is Block Pruning [25], which assumes an ordered set of blocks and terminates their processing as soon as *duplicate overhead* (i.e., the cost of identifying new duplicates) exceeds a predefined threshold $dh_{max}$. Processing the blocks of Figure 1(b) in their order of appearance, the initial duplicate overhead in block "John" is $dh = 1$ (i.e., one comparison for one pair of duplicates); the second pair of duplicates is identified in the fourth block "Richard" yielding a duplicate overhead $dh = 3$ (i.e., three comparisons for one pair of duplicates). For $dh_{max} = 2$, the remaining blocks will not be examined, thus saving 10 comparisons. Due to the coarse granularity of their functionality, though, existing block processing methods are unable to distinguish the redundant and superfluous comparisons from the matching ones (i.e., those involving a non-redundant pair of duplicate entity profiles). As a result, they enhance efficiency without controlling their impact on effectiveness.

**Work Overview and Contributions.** In this paper, we introduce *meta-blocking* as the task of developing efficient techniques that operate at the level of individual comparisons. These methods utilize abstract blocking information to achieve maximum efficiency gains for redundancy-positive blocking methods at a small and controllable impact on effectiveness. Meta-blocking goes beyond existing block processing methods by offering principled approaches that consider the information encapsulated in the set of *block assignments* (i.e., the associations between blocks and entity profiles). In essence, it aims at identifying the closest pairs of profiles so as to restructure a given set of blocks into a new one that involves significantly fewer comparisons, while maintaining the original level of effectiveness. Meta-blocking is independent from the underlying blocking method and generic enough to handle any redundancy-positive block collection, regardless of whether it is based on schema information or not.

We note that meta-blocking does not replace but complements the existing blocking methods. It builds on the intrinsic characteristic of redundancy-positive blocking that the similarity of two entity profiles is reflected on their common block assignments. Meta-blocking operates efficiently because it skips the high complexity of computing pair-wise, string-based entity similarities, relying instead on the block-to-entity profile associations of the input set of blocks. Although approximate, this information leads to an effective and efficient solution.

Based on these ideas, we introduce a family of meta-blocking methods that rely on the *blocking graph*. This is a structure that is extracted from the input block collection and connects with edges those pairs of entity profiles that are compared in at least one block. For instance, the graph corresponding to the blocks of Figure 1(b) is depicted in Figure 1(c); its nodes correspond to the profiles of the input entity collection (Figure 1(a)) and its edges connect the profiles that share at least one block. The edges are naturally undirected, and weighted according to a scheme that determines the trade-off between the computational cost and the gain of comparing the adjacent entity profiles (i.e., the benefit for the recall of the ER process, in case they are matching). In the example of Figure 1(c), we present the simplest scheme, which sets the weight of each edge equal to the number of blocks the adjacent entity profiles have in common. Also applicable are schema-based schemes, which set edge weights according to the values of one or more selected attributes.

The blocking graph forms the basis for enhancing efficiency through *pruning*: edges that do not satisfy a pre-defined criterion are removed, thus leading to a smaller number of comparisons. In our example, the blocking graph of Figure 1(d) can be derived from that of Figure 1(c) by discarding edges with a weight lower than 2, or by retaining the two edges with the highest weight. In any case, the remaining edges determine a new set of blocks that ideally places every pair of duplicate profiles in a separate block. Every retained edge is actually transformed into a new block that contains only its adjacent entity profiles. In our example, the pruned graph of Figure 1(d) yields two blocks, $b_1 = \{p_1, p_3\}$ and $b_2 = \{p_2, p_4\}$, that achieve the same recall as the blocks of Figure 1(b) with just 2 comparisons.

Overall, the contributions of our work are the following:

- We formalize the problem of meta-blocking and introduce the blocking graph as the cornerstone for a family of solutions that operate independently of the process that created the input blocks.
- We coin five schema-agnostic schemes for weighting the edges of a blocking graph.
- We present two schema-agnostic, orthogonal categories of pruning algorithms along with two orthogonal dimensions for specifying the corresponding pruning criteria.
- We examine the performance of our methods on three large-scale, real-world datasets, with the results validating the exceptional performance of our methods.

The rest of the paper is structured as follows: we formalize the task of meta-blocking in Section 2 and we present several techniques for building and pruning the blocking graph in Section 3. Section 4 analyzes the results of our experimental evaluation, and Section 5 wraps up our work. We discuss the state-of-the-art in blocking-based ER in the Appendix.

## 2 Problem Definition

**Entity Resolution.** At the core of entity resolution lie entity profiles describing real-world objects. An *entity profile* is a uniquely identified collection of information in the form of name-value pairs. Assuming an infinite set of identifiers $\mathcal{ID}$, we can formally define an entity profile as follows:

***Definition** 1 (Entity Profile):* An *entity profile* $p$ is a tuple $\langle id, A_p \rangle$, where $id \in \mathcal{ID}$ is a unique identifier, and $A_p$ is a set of name-value pairs $\langle n, v \rangle$.

Naturally, the value $v$ in a name-value pair $\langle n, v \rangle$ of an entity profile $p$ may be unspecified. Similarly, the attribute name $n$ may not be given, thus allowing for the representation of tag-style values, as illustrated in Figure 1(a). In general, the model of Definition 1 is flexible enough to accommodate entity representations of any complexity, such as those employed in Web and Dataspace applications [19]. In the following, we refer to this definition using the terms entity profile, profile and entity interchangeably.

An *entity collection* $\mathcal{E}$ is a set of entity profiles. Two entity profiles contained in $\mathcal{E}$, $p_i$ and $p_j$, are *duplicates* or *matches*, denoted by $p_i \equiv p_j$, if they represent the same real-world object. Given two input entity collections, $\mathcal{E}_1$ and $\mathcal{E}_2$, the goal of entity resolution is to identify the duplicate entity profiles they contain. Depending on the inputs, we distinguish the following types of ER:

- In *Clean-Clean ER*, both $\mathcal{E}_1$ and $\mathcal{E}_2$ are duplicate-free entity collections.
- In *Dirty-Clean ER*, $\mathcal{E}_1$ is a duplicate-free entity collection, and $\mathcal{E}_2$ is a dirty one (i.e., it contains duplicates in itself).
- In *Dirty-Dirty ER*, both $\mathcal{E}_1$ and $\mathcal{E}_2$ are dirty.

In all cases, the output comprises the pairs of duplicate profiles, $\mathcal{D}^{\mathcal{E}_1 \cup \mathcal{E}_2}$, that are contained in the union of the input entity collections (i.e., the duplicate profiles shared by $\mathcal{E}_1$ and $\mathcal{E}_2$ as well as those contained in the individual entity collections). Note that, for simplicity, we consider the last two sub-problems to be equivalent to *Dirty ER*: the input comprises a single entity collection $\mathcal{E}$ that contain duplicates in itself, as it is formed by the union of the given collections (i.e., $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$). In this case, the output comprises the set of matching pairs of entity profiles, $\mathcal{D}^{\mathcal{E}}$, that are contained in $\mathcal{E}$.

**Blocking for Entity Resolution.** ER constitutes an inherently quadratic task, requiring the pair-wise comparison of all profiles in the input entity collection(s). To make ER scale to large entity collections, blocking restricts the computational cost to comparisons between similar profiles: it clusters them into *blocks* and performs comparisons solely among the entity profiles within each block.

In more detail, block building techniques transform every entity profile into a (set of) *blocking key(s)* that is suitable for clustering. Profiles with the same (or similar) key(s) are grouped together into blocks (Figures 1(a) and (b)). The resulting set of blocks $\mathcal{B}$ is called *block collection*. Depending on the ER problem, its elements may be of two types:

- *Unilateral blocks* contain entity profiles from the same dirty entity collection (i.e., Dirty ER). Thus, they are all candidate matches and should be compared to each other.

- *Bilateral blocks* are internally partitioned in two sub-blocks that individually contain non-matching entity profiles from the same clean input collection (i.e., Clean-Clean ER). Thus, for a bilateral block $b_i$, comparisons are only allowed between its inner blocks $b_i^1$ ($\subseteq\mathcal{E}_1$) and $b_i^2$ ($\subseteq\mathcal{E}_2$).

**Improving Blocking through Meta-blocking.** The quality of a block collection $\mathcal{B}$ is measured in terms of two competing criteria: efficiency and effectiveness. The former is directly related to its *aggregate cardinality* ($||\mathcal{B}||$), i.e., the total number of comparisons it contains: $||\mathcal{B}|| = \sum_{b_i \in \mathcal{B}} ||b_i||$, where $||b_i||$ is the *individual cardinality* of $b_i$ (i.e., total number of comparisons entailed in block $b_i$); we have $||b_i||=|b_i|\cdot(|b_i|-1)/2$ for a unilateral block $b_i$ and $||b_j||=|b_j^1|\cdot|b_j^2|$ for a bilateral block. The effectiveness of $\mathcal{B}$ depends on the cardinality of the set $\mathcal{D}^{\mathcal{B}}$ of detectable matches (i.e., pairs of duplicate profiles compared in at least one block).

There is a clear trade-off between the effectiveness and the efficiency of $\mathcal{B}$: the more comparisons are executed (i.e., higher $||\mathcal{B}||$), the higher its effectiveness gets (i.e., higher $|\mathcal{D}^{\mathcal{B}}|$), but the lower its efficiency is, and vice versa. Successful block collections achieve a good balance between these two competing objectives, as estimated by the following, established measures [3], [7], [22], [25].

*(i)* **Pair Completeness** (*PC*) assesses the portion of duplicates that share at least one block and, thus, can be detected. It is formally defined as: $PC(\mathcal{B}) = |\mathcal{D}^{\mathcal{B}}|/|\mathcal{D}^{\mathcal{E}}|$, where $|\mathcal{D}^{\mathcal{E}}|$ is the number of duplicates in the input entity collection $\mathcal{E}$. *PC* takes values in the interval $[0, 1]$, with higher values indicating higher *effectiveness* for $\mathcal{B}$.

*(ii)* **Pairs Quality** (*PQ*) estimates the portion of non-redundant comparisons that involve matching entities. Formally, it is defined as: $PQ(\mathcal{B}) = |\mathcal{D}^{\mathcal{B}}|/||\mathcal{B}||$. It takes values in $[0, 1]$, with higher values indicating higher *efficiency* for $\mathcal{B}$ (i.e., fewer superfluous and redundant comparisons).

*(iii)* **Reduction Ratio** (*RR*) measures to which degree efficiency is enhanced with respect to a baseline block collection $\mathcal{B}_{bs}$. It is defined as: $RR(\mathcal{B}, \mathcal{B}_{bs}) = 1 - ||\mathcal{B}||/||\mathcal{B}_{bs}||$ and takes values in the interval $[0, 1]$ (for $||\mathcal{B}|| \leq ||\mathcal{B}_{bs}||$), with higher values denoting higher *efficiency* for $\mathcal{B}$.

Meta-blocking aims at restructuring a block collection $\mathcal{B}$ so as to improve its quality. It operates on its elements independently of their type (i.e., unilateral or bilateral blocks), relying primarily on the information encapsulated in their block assignments. Its output comprises a new block collection $\mathcal{B}'$ that maintains comparable levels of effectiveness (i.e., *PC*), while involving lower aggregate cardinality (i.e., higher efficiency). Formally, this task is defined as follows:

***Problem*** *1 (Meta-blocking):* Given a block collection $\mathcal{B}$, restructure it into a new one $\mathcal{B}'$ that achieves significantly higher levels of efficiency (i.e., $PQ(\mathcal{B}')\gg PQ(\mathcal{B})$ and $RR(\mathcal{B}', \mathcal{B})\gg 0$), while maintaining the original effectiveness (i.e., $PC(\mathcal{B}')\geq PC(\mathcal{B})$).

Note that the type of output blocks does not need to coincide with the input ones. As we will see in Section 3.3, a unilateral block collection can be transformed into a bilateral one, and vice versa. Note also that, in general, the effectiveness of the output block collection can be higher than that of the input one (i.e., $PC(\mathcal{B}') > PC(\mathcal{B})$). However, this can only be achieved by inferring new connections between entities from the original ones. We consider this inference problem to be orthogonal to the task we study in this paper, i.e., how to improve the efficiency of a block collection without affecting its effectiveness.

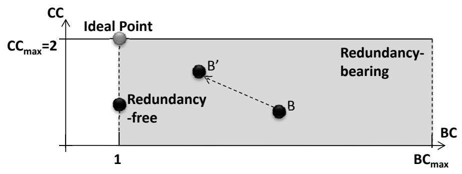


Fig. 2. The *BC-CC* metric space along with its main topological characteristics. The horizontal axis corresponds to Blocking Cardinality, which measures the redundancy of block collections, while the vertical one corresponds to Comparisons Cardinality, which estimates their efficiency.

**Metric Space for Blocking Techniques.** The goal of meta-blocking is to improve the balance between effectiveness and efficiency that a block collection $\mathcal{B}$ achieves. However, the impact on *PC* and *RR* can only be computed *after* examining analytically all blocks in $\mathcal{B}$ and $\mathcal{B}'$. Instead, we want to estimate their actual values *without* executing any comparison, so as to guide the restructuring process. A close, a-priori approximation of *PC* and *RR* is provided by the orthogonal measures of the *BC-CC* metric space, which was originally introduced in [27] for blocking-based Dirty ER. Here, we extend it to cover blocking-based Clean-Clean ER, as well, by adding the necessary definitions.

As depicted in Figure 2, the horizontal dimension of the *BC-CC* metric space corresponds to *Blocking Cardinality* (*BC*). This measure quantifies the redundancy of a block collection $\mathcal{B}$ as the average number of block assignments per entity of the input collection(s): $BC = \sum_{b_i \in \mathcal{B}} |b_i|/|\mathcal{E}|$, where $|b_i|$ denotes size (i.e., the number of entities) of block $b_i$. *BC* takes values in the interval $[0, \frac{2\cdot|\mathcal{E}_1|\cdot|\mathcal{E}_2|}{|\mathcal{E}_1|+|\mathcal{E}_2|}]$ for Clean-Clean ER and in $[0, |\mathcal{E}| - 1]$ for Dirty ER. Values lower than 1 indicate block collections that failed to place every entity profile in at least one block, values equal to 1 usually correspond to redundancy-free block collections (black dot in Figure 2), and values over 1 to redundancy-bearing ones (gray sub-plane in Figure 2). In general, the higher *BC* is, the higher is the level of redundancy in $\mathcal{B}$.

The vertical axis measures *Comparisons Cardinality* (*CC*), which estimates the efficiency of a block collection through the number of block assignments that account for each comparison: $CC = \sum_{b_i \in \mathcal{B}} |b_i|/||\mathcal{B}||$. *CC* takes values in the interval $[0, 2]$, with higher values corresponding to fewer comparisons per block assignment, and higher efficiency (i.e., smaller blocks, on average). Its maximum value actually corresponds to a block collection that exclusively contains blocks of minimum size (i.e., two entities).

The *BC-CC* mapping of a block collection can be efficiently computed in linear time (i.e., $O(|\mathcal{B}|)$) through a simple inspection of the size and the cardinality of its elements. It has been experimentally demonstrated that, for redundancy-positive blocking methods, *BC* is highly correlated with *PC* (i.e., higher *BC* values lead to higher effectiveness), while *CC* is directly related to *RR* (i.e., higher *CC* values convey higher efficiency) [27]. In conjunction, they can be used for a-priori comparing the performance of blocking schemes: the closer a blocking method is mapped to point (1,2) (gray dot in Figure 2), the better is its balance between *PC* and *RR* [27]. Indeed, this represents the *Ideal Point* that corresponds to the *optimal blocking method*, i.e., the method that builds a block of minimum size for each pair of duplicates, thus involving neither redundant nor superfluous comparisons. In this context, the goal of meta-blocking is to restructure a block collection so as to move its mapping closer to the *Ideal Point* (from $\mathcal{B}$ to $\mathcal{B}'$ in Figure 2). Section 3.3 explains how this is accomplished.

## 3 Meta-Blocking Approach

At the core of our approach to meta-blocking lies the notion of *blocking graph*. Given a block collection $\mathcal{B}$, the corresponding blocking graph $\mathcal{G}_{\mathcal{B}}$ models the block assignments in $\mathcal{B}$: as shown in Figure 1(c), every entity contained in $\mathcal{B}$ is mapped to a node in the blocking graph, and every pair of *co-occurring entities* (i.e., entities that are compared in at least one block) is connected with an undirected edge. Formally, the blocking graph for a unilateral block collection is defined as follows:

***Definition** 2 (Undirected Blocking Graph):* Given a *unilateral* block collection $\mathcal{B}^{\mathcal{E}}$, the *undirected blocking graph* derived from it is a graph $\mathcal{G}_{\mathcal{B}} = \{V_{\mathcal{B}}, E_{\mathcal{B}}, WS\}$, where $V_{\mathcal{B}}$ is the set of its nodes, $E_{\mathcal{B}}$ is the set its undirected edges, and $WS$ is the *weighting scheme* that determines the weight of every edge in the interval $[0, 1]$. $V_{\mathcal{B}}$ contains all entities of $\mathcal{E}$ that are placed in at least one block of $\mathcal{B}^{\mathcal{E}}$ (i.e., $\forall v_i \in V_{\mathcal{B}} : \exists p_i \in \mathcal{E} \wedge b_j \in \mathcal{B}^{\mathcal{E}} \wedge p_i \in b_j$), while $E_{\mathcal{B}}$ contains undirected edges between all pairs of co-occurring entities (i.e., $\forall e_{i,j} = \langle p_i, p_j\rangle \in E^{\mathcal{E}} : p_i \neq p_j \wedge \exists b_k \in \mathcal{B}^{\mathcal{E}} \wedge p_i \in b_k \wedge p_j \in b_k$).

The blocking graph over a set of bilateral blocks $\mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2}$ is defined analogously. The only difference is that it results in a *bipartite* graph, since its set of nodes $V_{\mathcal{B}}$ is separated into two disjoint sets, $V^1_{\mathcal{B}}$ and $V^2_{\mathcal{B}}$, which comprise entities stemming from the entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively (i.e., $V^1_{\mathcal{B}} \subseteq \mathcal{E}_1$ and $V^2_{\mathcal{B}} \subseteq \mathcal{E}_2$). More formally, $\forall v^k_i \in V^k_{\mathcal{B}} : \exists p_i \in \mathcal{E}_k \wedge b^{1,2}_j \in \mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2} \wedge p_i \in b^k_j$, where $k \in \{1, 2\}$. Thus, the set of edges $E_{\mathcal{B}}$ contains only connections between entities stemming from different entity collections: $\forall e_{i,j} = \langle p_i, p_j\rangle \in E_{\mathcal{B}} : \exists b^{1,2}_k \in \mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2} \wedge p_i \in b^1_k \wedge p_j \in b^2_k$.

Note that for reasons explained in Section 3.3, the edges of a blocking graph can be directed, as well. An edge pointing from entity $p_i$ to $p_j$ is represented by $\bar{e_{i,j}}$ to distinguish it from the undirected edge $e_{i,j}$ that connects the same entities. A blocking graph with directed edges is called *directed blocking graph* and is symbolized as $\bar{\mathcal{G}_{\mathcal{B}}}$.

The purpose of the blocking graph is to facilitate efficiency improvements over the input block collection. An immediate contribution to this goal is the *elimination of redundant comparisons* without any impact on effectiveness (i.e., *PC*). Redundant comparisons are easily identified during the creation of the blocking graph, as the corresponding entities have already been connected with an edge. In such cases, we simply skip connecting them with an additional edge and, thus, each pair of comparable entities is connected with at most one edge, regardless of the total number of comparisons between them entailed in $\mathcal{B}$. Consequently, each pair of co-occurring (i.e., adjacent) entities is examined only once. While improving efficiency, effectiveness is not affected, since the set of comparable entity pairs remains unchanged.

Additional efficiency enhancements can be achieved through the *pruning* of the blocking graph: edges between *non-matching* entities can be gradually removed from the graph, discarding unnecessary comparisons without affecting *PC*. This process is carried out according to a *pruning algorithm* and theoretically can result in a graph that exclusively contains edges between matching entities, as in Figure 1(d). In practice, though, we can only approximate this ideal case by exploiting the evidence that is encapsulated in the given block collection: how entities are assigned to blocks provides reliable indications for the similarity of adjacent entities, which can be quantified by assigning a weight to the corresponding edge. In the context of redundancy-positive blocking methods, the more blocks two entities share, the more similar they are and the higher the weight of their adjacent edge should be. In this way, the pruning of the blocking graph becomes the process of removing edges with low weights on the grounds that they (are likely to) link dissimilar entities.

In more detail, the weight $e_{i,j}.weight$ of an edge $e_{i,j}$ expresses the utility of the comparison between the profiles $p_i$ and $p_j$; that is, it quantifies the trade-off between the cost $c_{i,j}$ of comparing the adjacent entities and the gain $g_{i,j}$ of executing this comparison (i.e., $e_{i,j}.weight = g_{i,j}/c_{i,j}$). The cost $c_{i,j}$ pertains to the number of comparisons required by the corresponding edge and is always equal to 1 (since, by definition, each edge in the blocking graph captures one comparison). Thus, the edge weight is always equal to the gain of the corresponding comparison, which is 0 if the compared entities are not matching and 1 if they are duplicates (i.e., $e_{i,j}.weight = 0 \leftrightarrow p_i \not\equiv p_j$ and $e_{i,j}.weight = 1 \leftrightarrow p_i \equiv p_j$).

However, it is not possible to estimate the real value of $g_{i,j}$, and correspondingly $e_{i,j}.weight$, without actually executing the comparison between $p_i$ and $p_j$. For this reason, we use a *weighting scheme* that a-priori approximates the weight of each edge by considering the features of the blocking graph (e.g., the number of blocks shared by an edge's adjacent entities and the corresponding individual cardinalities). In Section 3.2, we will present five such weighting schemes for redundancy-positive blocking methods (i.e., the more similar two entities are, the higher the weight of the corresponding edge is). Edges with low weights are discarded by a *pruning criterion* that bounds either the number or the weight of the retained edges.

Overall, our approach to meta-blocking involves four successive steps, which are illustrated in Figure 3:

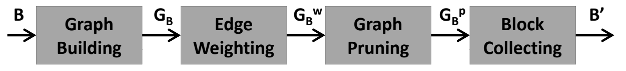


Fig. 3. The internal functionality of our meta-blocking approach.

**(i)** *Graph Building* receives a block collection $\mathcal{B}$ and derives the blocking graph $\mathcal{G}_{\mathcal{B}}$ from its block assignments. We elaborate on this process in Section 3.1.

**(ii)** *Edge Weighting* takes as input a blocking graph $\mathcal{G}_{\mathcal{B}}$ and turns it into the *weighted blocking graph* ($\mathcal{G}^w_{\mathcal{B}}$) by determining the weights of its edges. We introduce several weighting schemes for this procedure in Section 3.2.

**(iii)** *Graph Pruning* receives as input the weighted blocking graph and derives the *pruned blocking graph* ($\mathcal{G}^p_{\mathcal{B}}$) from it by removing some of its edges. We delve into the pruning algorithms and the pruning criteria involved in this procedure in Section 3.3.

**(iv)** *Block Collecting* is given as input the pruned blocking graph $\mathcal{G}^p_{\mathcal{B}}$ and extracts from it a new block collection $\mathcal{B}'$, which actually constitutes the final output of the entire meta-blocking process. We analyze this step in Section 3.4.

Note that the weighting scheme, the pruning algorithm, and the pruning criterion can entail a *schema-dependent*, *schema-agnostic*, or *hybrid* functionality. In the following, we focus on schema-agnostic techniques since they are applicable to any *blocking settings*, i.e., any combination of a blocking scheme and a (pair of) entity collection(s).

## 3.1 Building the Blocking Graph

The process of extracting the blocking graph from a bilateral block collection $\mathcal{B}$ is outlined in Algorithm 1 (for unilateral blocks, the corresponding algorithm is simpler, and we omit it for brevity). Essentially, for each block in $\mathcal{B}$, we consider every distinct pair of entities it contains (Lines 2-5); for bilateral blocks, this process requires that the considered entities belong to different inner blocks (i.e., $p_i \in b^1_i$ and $p_j \in b^2_j$). For each pair, we add the corresponding nodes to the initially empty blocking graph (Lines 4 and 6) and connect them with an edge (Line 7). The edge weights are specified after the structure of the blocking graph has been settled, because — as explained in the next subsection — it is possible for a blocking scheme to rely on it (Line 8). To restrict them to the interval $[0, 1]$ regardless of the input weighting scheme (cf. Definition 2), we normalize them by dividing with the maximum one (Line 9). The time complexity of this procedure is equal to the aggregate cardinality of $\mathcal{B}$ (i.e., $O(||\mathcal{B}||)$).

**Graph Materialization.** The blocking graph constitutes a conceptual model that aims at facilitating the interpretation and the development of our meta-blocking techniques. In the context of large entity collections with millions of entities (nodes) and billions of comparisons (edges), its materialization actually poses significant technical challenges. For this reason, it can be indirectly implemented in two ways: *(i)* through inverted indices, which associate each entity with the list of the blocks containing it, and *(ii)* with the help of bit arrays, which represent each entity as a vector with a zero value in all places, but those corresponding to the blocks containing it (these are valued 1). Both approaches scale well in the context of HHIS and accommodate all the weighting schemes of Section 3.2.

**Algorithm 1**: **Building the Blocking Graph.**

```
Input: (i) $\mathcal{B}$ a block collection,
       (ii) $WS$ a weighting scheme
Output: $\mathcal{G}_{\mathcal{B}}$ the corresponding blocking graph
1  $V_{\mathcal{B}} \leftarrow \{\}$; $E_{\mathcal{B}} \leftarrow \{\}$; //initially empty graph
2  foreach $b_i \in \mathcal{B}$ do // check all blocks
3      foreach $p_i \in b^1_i$ do // check all comparisons
4          $V_{\mathcal{B}} \leftarrow V_{\mathcal{B}} \cup \{v_i\}$;
5          foreach $p_j \in b^2_i$ do
6              $V_{\mathcal{B}} \leftarrow V_{\mathcal{B}} \cup \{v_j\}$; //add node for $p_j$
7              $E_{\mathcal{B}} \leftarrow E_{\mathcal{B}} \cup \{e_{i,j}\}$; //add edge <$p_i, p_j$>
8  setEdgeWeights($WS$, $\mathcal{B}$, $V_{\mathcal{B}}$, $E_{\mathcal{B}}$);
9  normalizeEdgeWeights($E_{\mathcal{B}}$);
10 return $\mathcal{G}_{\mathcal{B}} = \{V_{\mathcal{B}}, E_{\mathcal{B}}, WS\}$;
```

**Efficiency of Construction.** Theoretically, the construction of the blocking graph has the same complexity as the naïve method that iterates over all pairs of comparable entities. In practice, though, meta-blocking exhibits a lower running time when implemented on the basis of inverted indices or bit arrays, because it exclusively involves operations with integers. Thus, the computation of edge weights is much faster than the comparison of entity profiles, which invariably relies on string matching algorithms. The reason is that the latter typically have a non-trivial complexity of their own. As an example, consider edit distance, one of the simplest string comparison techniques, whose complexity even for an optimized implementation is $O(n^2/\log n)$, when $n$ is the length for both of the compared strings [20]. We analytically examine the time requirements of our meta-blocking approaches in Section 4.4.

## 3.2 Edge Weighting

We introduce five schema-agnostic weighting schemes that rely exclusively on evidence drawn from the input block collection. We use the following notations: $\mathcal{B}_i \subseteq \mathcal{B}$ denotes the set of blocks containing the entity $p_i$, $\mathcal{B}_{i,j} \subseteq \mathcal{B}$ is the set of blocks shared by the entities $p_i$ and $p_j$ (i.e., $\mathcal{B}_{i,j} = \mathcal{B}_i \cap \mathcal{B}_j$), and $|v_i|$ symbolizes the degree of node $v_i$ (i.e., the number of edges connected to it). Next, we describe our weighting schemes and explain the rationale behind them.

**(i)** *Aggregate Reciprocal Comparisons Scheme* (*ARCS*): This scheme is based on the premise that the more entities a block contains, the less likely they are to match. The reason is that the information forming this block is not distinctive enough to group highly similar entities. For instance, in the case of attribute-agnostic blocking, common words would cluster together a large part of the input entity collection. In this context, the aggregate similarity of two co-occurring entities, $p_i$ and $p_j$, is defined as the sum of the reciprocal individual cardinalities of their common blocks. Formally, the weight of an edge $e_{i,j}$ is defined as follows:

$$e_{i,j}.weight = \sum_{b_k \in \mathcal{B}_{i,j}} \frac{1}{||b_k||}.$$

**(ii)** *Common Blocks Scheme* (*CBS*): A strong indication of the similarity of two entities is provided by the number of blocks they have in common; the more blocks they share, the more likely they are to match. Therefore, the weight of an edge connecting entities $p_i$ and $p_j$ is set equal to:

$$e_{i,j}.weight = |\mathcal{B}_{i,j}|.$$

**(iii)** *Enhanced Common Blocks Scheme* (*ECBS*): This scheme improves on *CBS* by adding contextual information to its weights. Instead of merely considering the number of common blocks, it takes into account the total number of blocks that are associated with each one of the co-occurring entities. Inspired from the IDF metric of Information Retrieval, the fewer blocks an entity is placed in, the higher should be the weights of the edges associated with it. More formally, the weight of an edge is set equal to:

$$e_{i,j}.weight = |\mathcal{B}_{i,j}| \cdot \log \frac{|\mathcal{B}|}{|\mathcal{B}_i|} \cdot \log \frac{|\mathcal{B}|}{|\mathcal{B}_j|}.$$

**(iv)** *Jaccard Scheme* (*JS*): Similar to *ECBS*, this scheme aims at enhancing *CBS* by considering the total number of blocks associated with the co-occurring entities. To this end, it sets the weight of the edge $e_{i,j}$ equal to the Jaccard similarity of the lists of blocks associated with its adjacent entities, $p_i$ and $p_j$:

$$e_{i,j}.weight = \frac{|\mathcal{B}_{i,j}|}{|\mathcal{B}_i| + |\mathcal{B}_j| - |\mathcal{B}_{i,j}|}.$$

The resulting weights take values in the interval $[0, 1]$, with 0 indicating the absence of common blocks and 1 corresponding to identical block lists. In essence, these weights reveal the percentage of common blocks shared by the adjacent entities.

**(v)** *Enhanced Jaccard Scheme* (*EJS*): This scheme improves on *JS* by adding contextual information to the Jaccard similarity of the associated blocks. Namely, it considers the total number of edges (i.e., comparisons) associated with each one of the adjacent nodes. Based on IDF, the fewer edges connected with a node, the higher should their individual weights be. Thus, we have:

$$e_{i,j}.weight = \frac{|\mathcal{B}_{i,j}|}{|\mathcal{B}_i| + |\mathcal{B}_j| - |\mathcal{B}_{i,j}|} \cdot \log \frac{|E_{\mathcal{B}}|}{|v_i|} \cdot \log \frac{|E_{\mathcal{B}}|}{|v_j|}.$$

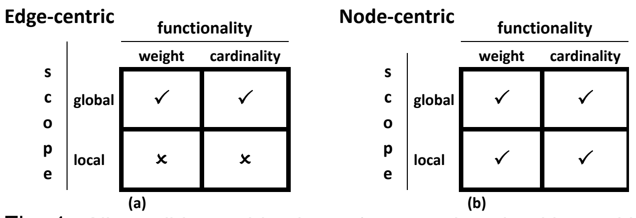
Note that the above weighting schemes rely on the principle of redundancy-positive blocking methods that the similarity of block assignments provides a good representation of matching probability. Thus, the more blocks two entities share, the more similar their profiles are expected to be. In Section 4, we experimentally analyze the effect of these weighting schemes on the performance of meta-blocking.

## 3.3 Pruning the Blocking Graph

This process is based on two essential components: *(i)* the *pruning algorithm*, which specifies the procedure that will be followed in the processing of the blocking graph, and *(ii)* the *pruning criterion*, which determines the edges to be retained. The combination of a pruning algorithm with a pruning criterion forms a *pruning scheme*. In this work, we introduce a series of pruning schemes that rely on schema-agnostic pruning algorithms and criteria, thus being applicable to any blocking graph.

**Pruning algorithms.** In general, they can be categorized in two classes:

- The *edge-centric algorithms* select the *globally* best comparisons by iterating over the *edges* of a blocking graph in order to filter out those that do not satisfy the pruning criterion.



Edge-centric
functionality
weight
cardinality
scope
global
local
✓
✓
×
×
(a)
Node-centric
functionality
weight
cardinality
scope
global
local
✓
✓
✓
✓
(b)

Fig. 4. All possible combinations of our pruning algorithms with our pruning criteria.

- The *node-centric algorithms* iterate over the *nodes* of a blocking graph with the aim of selecting the *locally* best comparisons for each entity (i.e., the adjacent entities with the largest edge weights).

We analytically examine the relative performance of these two types of pruning algorithms in Section 4.2.

**Pruning criteria.** In general, they can be categorized in a two-dimensional taxonomy formed by the orthogonal but complementary dimensions of functionality and scope. The *functionality* of pruning criteria distinguishes them into *weight thresholds*, which specify the minimum weight for the edges to be retained, and *cardinality thresholds*, which determine the maximum number of retained edges. The *scope* of pruning criteria distinguishes them into *global thresholds*, which define conditions that are applicable to the entire blocking graph (i.e., all the edges of the graph), and *local thresholds*, which specify conditions that apply to a subset of it (i.e., the adjacent edges of a specific node).

Cardinality thresholds should be preferred in applications that have predefined temporal resources (i.e., available processing time), because they allow for a-priori determining the number of executed comparisons. In contrast, weight thresholds are convenient for applications that put more emphasis on controlling effectiveness, since the harshness of their pruning is analogous to their value. Both classes, though, are suitable for incremental ER (a.k.a., Pay-As-You-Go ER) [29], where the goal is to execute most of the matching comparisons in the first iterations, decreasing their number gradually, as ER progresses. For weight (cardinality) thresholds, this can be simply achieved by decreasing (increasing) its value in every iteration.

**Pruning Schemes.** The composition of pruning schemes is determined by the scope of pruning thresholds. In Figure 4, we illustrate all possible combinations of pruning algorithms with pruning criteria. Starting with the edge-centric algorithms, we observe that they can only be combined with global criteria — regardless of their functionality. The reason is that it is impossible to employ a local threshold, when trying to select the top weighted edges across the entire blocking graph. The combination of edge-centric algorithms with global weight thresholds (i.e., *WEP*) is analyzed in Section 3.3.1 and their coupling with global cardinality thresholds (i.e., *CEP*) in Section 3.3.2.

By definition, the node-centric algorithms are compatible with local thresholds — regardless of their functionality. However, they can be combined with global thresholds, as well. Their combination with a global weight threshold is actually identical to *WEP*, as they both retain the edges that are weighted higher than the given threshold. Their

**Algorithm 2**: **Weight Edge Pruning.**

```
Input: (i) $\mathcal{G}^{in}_{\mathcal{B}}$ the blocking graph, and
       (ii) $w_{min}$ the global weight pruning criterion.
Output: $\mathcal{G}^{out}_{\mathcal{B}}$ the undirected pruned blocking graph
1 foreach $e_{i,j} \in E_{\mathcal{B}}$ do
2     if $e_{i,j}.weight < w_{min}$ then // discard every edge with
3         $E_{\mathcal{B}} \leftarrow E_{\mathcal{B}}$ - { $e_{i,j}$ }; // weight lower than $w_{min}$
4 return $\mathcal{G}^{out}_{\mathcal{B}} = \{V_{\mathcal{B}}, E_{\mathcal{B}}, WS\}$;
```

coupling with a global cardinality threshold retains the same number of adjacent edges among all nodes (e.g., the 2 top-weighted edges per node). In contrast, their combination with a local cardinality threshold derives the number of retained edges for each node from its degree (e.g., $|v_i|/10$ of the top weighted edges for every node $v_i$); this approach is substantially different from $CEP$, which keeps the top weighted edges across the entire blocking graph. The pruning schemes that combine node-centric algorithms with local weight thresholds (i.e., $WNP$) are examined in Section 3.3.3, while those coupling them with cardinality thresholds — of any scope — (i.e., $CNP$) are examined in Section 3.3.4.

Before elaborating on the functionality of the pruning schemes, it should be stressed that the node-centric algorithms yield a directed, pruned blocking graph, unlike the edge-centric algorithms that produce an undirected one.

### 3.3.1 Weight Edge Pruning (WEP)

This scheme consists of the edge-centric algorithm coupled with a global weight threshold: the minimum edge weight. Its functionality is outlined in Algorithm 2. It iterates over all edges (Line 1) and discards (Line 3) those having a weight lower that the input threshold (Line 2). The remaining edges form the pruned blocking graph of the output. The time complexity of this algorithm is equal to the aggregate cardinality of the original block collection (i.e., $O(||\mathcal{B}||)$).

The most critical part of this algorithm is the selection of the minimum edge weight $w_{min}$. Its precise value depends on the underlying weighting scheme and the resulting distribution of edge weights, in particular. In general, though, the matching entities are expected to be connected with edges of higher weights than the non-matching ones. Thus, the goal is to identify the break-even point that distinguishes the former type of edges from the latter. Experimental evidence with real-world datasets suggests that the average edge weight provides an efficient (i.e., requires just one iteration over all edges) as well as reliable (i.e., low impact on effectiveness) estimation of this break-even point, regardless of the underlying weighting scheme (see Section 4.2 for details).

### 3.3.2 Cardinality Edge Pruning (CEP) or Top-K Edges

This scheme combines the edge-centric pruning algorithm with a global cardinality threshold $K$ that specifies the total number of edges retained in the pruned graph. The goal is to retain the $K$ edges with the maximum weight. As illustrated in the outline of Algorithm 3, this technique employs a

**Algorithm 3**: **Cardinality Edge Pruning**.

```
Input: (i) $\mathcal{G}^{in}_{\mathcal{B}}$ the blocking graph, and
       (ii) $K$ the global cardinality pruning criterion.
Output: $\mathcal{G}^{out}_{\mathcal{B}}$ the undirected pruned blocking graph
1 $SortedStack \leftarrow \{\}$; // sorts edges in descending weight
2 foreach $e_{i,j} \in E_{\mathcal{B}}$ do          // add every edge
3     $SortedStack$.push($e_{i,j}$); // in the sorted stack
4     if $K < SortedStack.size()$ then // remove the edge with
5         $SortedStack$.pop();        // the (K+1)$^{th}$ top weight
6 foreach $e_{i,j} \in E_{\mathcal{B}}$ do          // discard all edges
7     if $e_{i,j} \notin SortedStack$ then // that are not among the
8         $E_{\mathcal{B}} \leftarrow E_{\mathcal{B}}$ - { $e_{i,j}$ }; // the top-K weighted ones
9 return $\mathcal{G}^{out}_{\mathcal{B}} = \{V_{\mathcal{B}}, E_{\mathcal{B}}, WS\}$;
```

*sorted stack* in order to store the edges in descending order of weights, thus efficiently removing (i.e., pop) the edge with the lowest weight. The algorithm iterates over all edges of the input blocking graph twice: the first iteration (Lines 2-5) identifies the top-$K$ edges and stores them in the sorted stack; the second iteration (Line 6-8) removes from the graph those edges that are not contained in the sorted stack. Similar to $WEP$, the time complexity of this algorithm is equal to the aggregate cardinality of original block collection (i.e., $O(||\mathcal{B}||)$).

To specify the optimal value for $K$, we employ a technique that relies on the $BC$-$CC$ mapping of the initial blocking graph and its pruned version. The goal is to map the latter closer to the Ideal Point (1,2) than the former. Given that the pruned graph results in a bilateral block collection with $K$ blocks of size 2 (cf. Section 3.4), its $CC$ takes the maximum value (i.e., $CC_{out}$=2)[2], while its $BC$ is equal to $BC_{out}=\frac{2K}{|\mathcal{E}|}$, where $\mathcal{E}$ is the size of the input entity collection. Thus, $CC_{out}$ is greater than or equal to $CC_{in}$ of the input blocking graph in all cases and, for an improved $BC$-$CC$ mapping, it suffices to have: $BC_{out} \leq BC_{in} \Leftrightarrow \frac{2K}{|\mathcal{E}|} \leq BC_{in} \Leftrightarrow K \leq \lfloor \frac{BC_{in} \cdot |\mathcal{E}|}{2} \rfloor$, where $BC_{in}$ stands for the BC value of the input blocking graph. Therefore, the maximum meaningful value for $K$ is specified with respect to the level of redundancy of the input block collection. In cases where $CC_{in} \ll CC_{out}$, we can set $K = \lfloor \frac{BC_{in} \cdot |\mathcal{E}|}{2} \rfloor$ in order to ensure higher redundancy and, thus, higher $PC$. Although this approach maintains the same levels of redundancy (i.e., the same number of block assignments), efficiency is significantly improved; unlike the input block collection, which contains blocks of various sizes, the output exclusively comprises blocks of minimum size (i.e., two entities per block). This means that $CEP$ minimizes the number of pairwise comparisons for a specific level of redundancy.

### 3.3.3 Weight Node Pruning (WNP)

This scheme combines the node-centric pruning algorithm with a local weight threshold. In essence, it applies the $WEP$ to the neighborhood of each node $v_i$, i.e., the subgraph $\mathcal{G}_{v_i}$ that comprises the nodes of $\mathcal{G}_{\mathcal{B}}$ connected with $v_i$

2. $CC$ expresses the ratio of block assignments over comparisons (i.e., $CC=\sum_{b_i \in \mathcal{B}} |b_i| / \sum_{b_i \in \mathcal{B}} ||b_i||$. Given that the output of $CEP$ involves only blocks of size 2, there are two block assignments for every comparison, thus leading to $CC_{out}$=$CC_{max}$=2.

**Algorithm 4**: **Weight Node Pruning.**

```
Input: (i) $\mathcal{G}^{in}_{\mathcal{B}}$ the blocking graph, and
       (ii) $wt$ function for defining local weight pruning criteria.
Output: $\mathcal{G}^{out}_{\mathcal{B}}$ the directed pruned blocking graph
1  $E^{out}_{\mathcal{B}} \leftarrow \{\}$;          // the set of retained directed edges
2  foreach $v_i \in V_{\mathcal{B}}$ do                       // for every node get
3      $\mathcal{G}_{v_i} \leftarrow$ getNeighborhood($v_i$, $\mathcal{G}_{\mathcal{B}}$);  //its neighborhood and
       $t_{v_i} \leftarrow wt(\mathcal{G}_{v_i})$;                 // its local weight threshold
4      foreach $e_{i,j} \in E_{v_i}$ do              // retain every adjacent
5          if $t_{v_i} \leq e_{i,j}.weight$ then     // edge with weight
6              $E^{out}_{\mathcal{B}} \leftarrow E^{out}_{\mathcal{B}} \cup \{ \bar{e_{i,j}} \}$; // higher than $t_{v_i}$
7  return $\mathcal{G}^{out}_{\mathcal{B}} = \{V_{\mathcal{B}}, E^{out}_{\mathcal{B}}, WS\}$;
```

— denoted by $V_{v_i}$ — along with the edges connecting them — denoted by $E_{v_i}$. Its functionality, though, differs from $WEP$ in two aspects: *(i)* it employs a different threshold for each neighborhood, and *(ii)* it replaces the retained, undirected edges with directed ones that point from $v_i$ to a neighboring node. Algorithm 4 presents the pseudo-code for this procedure: it iterates over all nodes of the input blocking graph (Line 2) and extracts the corresponding neighborhood $\mathcal{G}_{v_i}$ (Line 3). Based on this, it specifies the minimum edge weight of the neighborhood according to the input local threshold criterion (Line 4). Then, it iterates over all edges of $E_{v_i}$ (Line 5) and adds one directed edge to the pruned graph for every undirected edge that exceeds the specified local threshold (Lines 6-7). In the worst case, the input blocking graph is a complete one, thus accounting for a time complexity of $O(|V_{\mathcal{B}}|\cdot|E_{\mathcal{B}}|)$; in practice, though, it is significantly lower, as the underlying blocking scheme ensures that not all nodes are connected with each other.

To specify the optimal threshold for each neighborhood, we rely on the same rationale as $WEP$: weighting schemes assign high values to edges connecting matching entities and low values to edges connecting non-matching nodes. Regardless of the selected scheme, the corresponding break-even point can be approximated by the mean weight of the edges in each neighborhood $\mathcal{G}_{v_i}$.

### 3.3.4 *Cardinality Node Pruning ($CNP$) or k-Nearest Entities*

At the core of this scheme lies the node-centric pruning algorithm in conjunction with a local cardinality threshold. Its goal is to select for each node $v_i$, the $k$ neighboring nodes that are connected with the top edge weights (i.e., $k$-nearest entities). To this end, it applies the $CEP$ algorithm to the neighborhood $\mathcal{G}_{v_i}$ of $v_i$, as depicted in Algorithm 5. In more detail, it iterates over all entities of the input blocking graph (Line 2), extracting their neighborhood (Line 4) and setting the maximum number of retained entities (Line 5). Subsequently, it iterates over the edges of the current neighborhood and places them into the sorted stack (Line 6-9). For each of the selected undirected edges, a new, directed one is added to the pruned blocking graph of the output (Lines 10-12). The time complexity of this algorithm is the same as that of $WNP$: $O(|V_{\mathcal{B}}|\cdot|E_{\mathcal{B}}|)$.

In general, the cardinality threshold for each neighborhood depends on its size (e.g., $k_i$=$\lceil 0.1\cdot|E_{v_i}|\rceil$). For simplicity,

**Algorithm 5**: **Cardinality Node Pruning.**

```
Input: (i) $\mathcal{G}^{in}_{\mathcal{B}}$ the blocking graph, and
       (ii) $ct$ function for defining local cardinality pruning criteria.
Output: $\mathcal{G}^{out}_{\mathcal{B}}$ the directed pruned blocking graph
1  $E^{out}_{\mathcal{B}} \leftarrow \{\}$;          // the set of retained directed edges
2  foreach $v_i \in V_{\mathcal{B}}$ do
3      $SortedStack_{v_i} \leftarrow \{\}$;               //for every node get
4      $\mathcal{G}_{v_i} \leftarrow$ getNeighborhood($v_i$, $\mathcal{G}_{\mathcal{B}}$); //its neighborhood and
5      $k \leftarrow ct(\mathcal{G}_{v_i})$;       // its local cardinality threshold
6      foreach $e_{i,j} \in E_{v_i}$ do              // add every adjacent
7          $SortedStack_{v_i}$.push($e_{i,j}$);     // edge in sorted stack
8          if $k < SortedStack_{v_i}.size()$ then       // remove the
9              $SortedStack_{v_i}$.pop();                // (k+1)$^{th}$ edge
10     foreach $e_{i,j} \in E_{v_i}$ do              // retain every adjacent
11         if $e_{i,j} \in SortedStack$ then          // edge contained in
12             $E^{out}_{\mathcal{B}} \leftarrow E^{out}_{\mathcal{B}} \cup \{ \bar{e_{i,j}} \}$;       // the SortedStack
13 return $\mathcal{G}^{out}_{\mathcal{B}} = \{V_{\mathcal{B}}, E^{out}_{\mathcal{B}}, WS\}$;
```

though, we assume in the following that $k$ takes the same value for each neighborhood. To identify its optimal value, we rely on the $BC$-$CC$ mapping of the input and the output blocking graph. Again, the goal is to ensure that the latter is closer to (1,2) than the former. Given that the block collection contains bilateral blocks with inner block sizes of 1 and $k$ (cf. Section 3.4), the $CC$ of the pruned graph is equal to $CC_{out} = \frac{k+1}{k}$, while its $BC$ is equal to $BC_{out}$=$k+1$. Thus, $k$ is specified with respect to the $CC$ and the $BC$ of the input block collection: $\frac{1}{1-CC_{or}}$≤$k$≤$BC_{in}$-1. In cases where $CC_{in}$≪1, we can safely set $k$=$\lfloor BC_{in}$-1$\rfloor$, ensuring significantly higher efficiency ($CC_{out}$>1) at equal levels of redundancy and $PC$.

## 3.4 Collecting the new blocks

The procedure for transforming a pruned blocking graph into a new block collection depends on the type of the graph. For the undirected pruned blocking graphs, which are produced by the edge-centric pruning algorithms, block collecting is straightforward: every retained edge lays the basis for creating a bilateral block of minimum size that contains the adjacent entities. As a result, the new block collection is redundancy-free (i.e., non-overlapping blocks). For example, the pruned blocking graph of Figure 1(d) is transformed in the blocks $b_1 = \{\{p_1\},\{p_3\}\}$ and $b_2 = \{\{p_2\},\{p_4\}\}$.

For the directed pruned blocking graphs, which are derived from the node-centric pruning algorithms, block collecting creates a bilateral block for every node $v_i$. Its inner blocks have the following property: one of them contains the entity that is mapped to $v_i$, while the other contains the entities connected with $v_i$ through the retained, outgoing edges. For instance, the pruned blocking graph of Figure 1(e)[3] is transformed into the blocks $b_1 = \{\{p_1\},\{p_3,p_4\}\}$ and $b_2 = \{\{p_2\},\{p_3,p_4\}\}$. In this way, the new block collection involves redundant comparisons, since it is possible for two retained edges with different direction to connect the same entities. This means that its

3. For clarity we have excluded the outgoing edges of nodes $p_3$ and $p_4$.

efficiency can be further enhanced with block processing techniques.

## 4 Evaluation

The goal of our experimental study is manifold: *(i)* to demonstrate the benefits of meta-blocking over existing blocking methods, *(ii)* to compare the edge-centric pruning schemes with the node-centric ones, *(iii)* to compare the weight pruning criteria with the cardinality ones, *(iv)* to compare the weighting schemes for building blocking graphs, *(v)* to compare meta-blocking with the state-of-the-art approach of Iterative Blocking, *(vi)* to examine the robustness of our pruning schemes, and *(vii)* to investigate the time requirements of meta-blocking over large blocking graphs with millions of nodes and billions of edges. Section 4.1 elaborates on the set-up of our experiments, and Section 4.2 examines the objectives *(i)* to *(v)*, analyzing the performance of all meta-blocking settings with respect to $RR$, $PC$ and $PQ$. Section 4.3 focuses on goal (*vi*) and Section 4.4 on goal (*vii*). Note that we had to place all figures and tables detailing our experimental results in the appendix, due to lack of space.

### 4.1 Setup

Our approaches were implemented in Java 1.6 and are publicly available through SourceForge.net[4]. Our experiments were performed on a server with Intel Xeon X5670 2.93GHz and 16GB of RAM, running Scientific Linux 5.8.

**Datasets.** In our evaluation, we used the same datasets as in our previous works [25], [26], [27], namely $D_{movies}$, $D_{infoboxes}$ and $D_{BTC09}$. In this way, we allow for a direct comparison with their outcomes. Note that we have publicly released all datasets, so that they can be used as a benchmark by other researchers[5]. Their technical characteristics are summarized in Table 1.

$D_{movies}$ is a collection of 50,000 entities shared among the individually clean collections of IMDB and DBPedia movies. The ground-truth for this Clean-Clean ER dataset stems from the "imdbid" attribute in the profiles of the DBPedia movies.

Our second Clean-Clean ER dataset, $D_{infoboxes}$, consists of two different versions of the DBPedia Infobox dataset[6]. They contain all name-value pairs of the infoboxes in the articles of Wikipedia's English version, extracted at specific points in time. The older collection, $DBPedia_1$, is a snapshot from October 2007, whereas $DBPedia_2$ dates from October 2009. The large time period that intervenes between the two collections renders their resolution challenging, since only 25% of all name-value pairs is shared among them [25]. As matching entities, we consider those with the same entity URL.

Finally, $D_{BTC09}$ is the Dirty ER dataset of our study, comprising more than 250,000 entities, a subset of those contained in the Billion Triple Challenge 2009 (BTC09) data collection[7]. Its ground-truth consists of 10,653 pairs of matching entities that were identified through their identical value for at least one inverse functional property.

**Baseline method.** To evaluate the performance of our meta-blocking techniques, the baseline for the two Clean-Clean ER datasets was specified as the attribute-agnostic blocking method in conjunction with Block Purging [25]. For $D_{movies}$, the resulting blocks exhibit nearly perfect effectiveness ($PC$ = 99.39%) combined with high efficiency ($RR$ = 95.83% with respect to the naïve method of comparing all DBPedia movies with the IMDB ones). The former can be actually attributed to the high levels of redundancy, as each entity is placed in 22 blocks, on average. The corresponding blocking graph is medium-sized, entailing 50 thousand nodes and 22 million edges. Similarly, the resulting block collection for $D_{infoboxes}$ achieves an excellent balance between efficiency and effectiveness (i.e., $RR$ = 98.46% and $PC$ = 99.89%). It involves high redundancy ($BC$≈15) and produces a large blocking graph with 3.3 million nodes and 34 billion edges.

The blocks of $D_{BTC09}$ were extracted from those produced by *Total Description* [27] when applied to the entire BTC09 data collection. To restrict the originally massive dataset to a moderate block collection that facilitates our thorough experimental analysis, we first purged those blocks that contained none of the ground-truth entities. We then removed the singleton entities, which were associated with just one block after sampling, in order to ensure a redundancy-positive block collection ($BC$>1) that allows for applying meta-blocking. Finally, we discarded the invalid blocks, which were left with just one entity, and applied Block Purging [27] on the remaining set of blocks. The resulting block collection combines a high $RR$(>99%) with a high $PC$(≈97%) and yields a blocking graph with 250 thousand nodes and 77 million edges.

Note that in all datasets, we do not measure the effect of meta-blocking on efficiency against a stand-alone block building method. Instead, we estimate $RR$ with respect to Block Purging, which yields a significant reduction in the aggregate cardinality of the original blocks. In addition, we consider as a baseline the state-of-the-art approach of Iterative Blocking [30]. In essence, this method propagates every new pair of duplicates to all associated blocks (even if they have already been examined) in order to identify additional matches and to save unnecessary comparisons.

To assess the impact of meta-blocking on effectiveness, we consider the relative reduction in $PC$ ($\Delta PC$), which is formally defined as $\Delta PC = \frac{PC(\mathcal{B}')-PC(\mathcal{B})}{PC(\mathcal{B})} \cdot 100\%$, where $PC(\mathcal{B})$ and $PC(\mathcal{B}')$ denote the effectiveness of the original and the restructured block collection, respectively.

### 4.2 Measuring the blocks of Meta-blocking

In this section, we examine the first five of our evaluation objectives. To this end, we applied all pruning schemes to all blocking graphs (i.e., weighting schemes) that can be derived from $D_{movies}$, $D_{infoboxes}$ and $D_{BTC09}$. We categorized the results according to the corresponding pruning scheme and analytically present them in Tables 4(a) to 4(d).

**(i)** *Effect of meta-blocking on blocking.* Table 4(a) depicts the performance of $WEP$ in conjunction with all weighting

4. `http://sourceforge.net/projects/erframework`
5. See `http://sourceforge.net/projects/erframework/files` for instructions on how to download them.
6. `http://wiki.dbpedia.org/Datasets`
7. http://km.aifb.kit.edu/projects/btc-2009

schemes across all datasets. For $D_{movies}$ and $D_{infoboxes}$, we notice that all weighting schemes convey significant enhancements in efficiency ($RR$>70%), while incurring moderate reduction in $PC$ ($\Delta PC$<10%). Similar patterns are exhibited for $D_{BTC09}$: in the worst case $\Delta PC \approx 10\%$, while $RR$ remains higher than 95% for all weighting schemes. The performance of most of them is actually very close over $D_{BTC09}$. In contrast, for $D_{movies}$ and $D_{infoboxes}$, there is a clear trade-off between $RR$ and $PC$: the higher the former gets, the lower the latter is and vice versa. Note, though, that the evolution of $PQ$ suggests that $RR$ decreases faster than $PC$ increases.

These patterns can be explained by the weight distribution lying at the core of each weighting scheme. As an example, consider Figures 7(a) and (b), which depict the distribution for every weighting scheme over $D_{movies}$ (similar distributions are exhibited in the other two datasets, as well, but we omit the corresponding diagrams, due to lack of space). In all histograms, the bucket size is set equal to half the average edge weight ($\bar{w}$) of the corresponding scheme across the entire blocking graph (i.e., including the links between matching and non-matching nodes/entities). Thus, the two leftmost bars correspond to the pruned edges and the remaining eight bars to the retained ones. We observe a clear polarization for all weighting schemes: the vast majority of the matching edges is concentrated on the two right-most intervals, with a negligible portion of them lying in the left half (the opposite applies to non-matching edges). In fact, the higher the $PC$ of a weighting scheme over $D_{movies}$ is, the lower is the corresponding number of matching edges in the first two intervals. On the other hand, the higher its $RR$ is, the lower is the portion of non-matching edges placed in the intervals $[1.5\cdot\bar{w}, 5\cdot\bar{w}]$.

Table 4(b) illustrates the performance of $WNP$ for all weighting schemes over all datasets. Similar to $WEP$, there is a clear trade-off between effectiveness and efficiency for $D_{movies}$ and $D_{infoboxes}$. It is interesting to note that ranking the weighting schemes in descending order of $RR$ (i.e., ascending order of $PC$) results in the same order as in Table 4(a). For $D_{BTC09}$, all weighting schemes achieve similar, high performances with respect to all metrics. Compared to $WEP$, though, the combination of every weighting scheme with $WNP$ yields significantly higher $PC$ as well as lower $RR$ and $PQ$.

Table 4(c) presents the performance of $CEP$ in combination with all weighting schemes across the three datasets. By definition, they all achieve the same $RR$, which amounts to 97.48%, 99.94% and 99.85% for $D_{movies}$, $D_{infoboxes}$ and $D_{BTC09}$, respectively. In absolute numbers, this corresponds to 11, 15 and 3 comparisons per entity, respectively, thus requiring 2 orders of magnitude fewer comparisons than the input block collection. Apparently, this is at the cost of lower effectiveness, since $PC$ is reduced in all datasets by more than 14%, regardless of the weighting scheme (the only exception is $ARCS$ for $D_{BTC09}$). The worst performance usually corresponds to $CBS$ and $JS$, because there are many pairs of entities that share exactly the same number or portion of blocks, respectively. Again, this behavior can be explained by the normalized histograms in Figures 7(a) and (b), since $CEP$ generally retains the edges of the rightmost interval; the more matching edges and the less non-matching ones it contains, the higher is the $PC$ of the corresponding weighting scheme.

Finally, Table 4(d) presents the performance of $CNP$ for all weighting schemes across all datasets. Similar to its edge-centric counterpart, it exhibits excessively high efficiency for both datasets (i.e., $RR$>95%). In absolute numbers, this corresponds to 22, 28 and 7 comparisons per entity for $D_{movies}$, $D_{infoboxes}$ and $D_{BTC09}$, respectively. Its impact on effectiveness is rather limited, reducing $PC$ at most by 5% for the Clean-Clean ER datasets and less than 14% for the Dirty ER one.

**(ii)** *Edge-centric vs. node-centric pruning schemes.* The relative performance of these two types of pruning schemes depends on the pruning criteria that lie at their core. Thus, an equal basis comparison requires exactly the same configuration. This is impossible, though, for the weight criteria: $WEP$ can only be combined with a global one, while $WNP$ makes sense only when coupled with a local one (its conjunction with a global threshold renders it identical to $WEP$).

The configuration of Section 3.3 approximates the ideally equal settings, assuming similar criteria for both algorithms (i.e., average edge weight). For this configuration, our experiments suggest that the edge-centric algorithms perform a deeper pruning that results in the lowest number of comparisons and detected matches (i.e., lowest $\Delta PC$). Nevertheless, they are more accurate in discarding superfluous comparisons, achieving higher $PQ$ across all datasets and weighting schemes. For example, consider the combination of $ARCS$ with $WEP$ and $WNP$ over $D_{movies}$: $PQ$ suggests that for every 100 comparisons, the former identifies around 1.5 matches and the latter almost half of them.

On the other hand, the node-centric schemes are more conservative in pruning edges, retaining even double as much comparisons. Thus, they have a significantly smaller impact on $PC$, which is also ensured by the more even distribution of comparisons among entities; unlike the edge-centric algorithms, which completely disregard the entities/nodes that are associated with none of the top weighted edges, they ensure that *every* node remains connected with the most similar of its co-occurring entities.

In the case of cardinality pruning criteria, it is possible to apply the same global threshold to both $CEP$ and $CNP$. However, these settings merely allow for comparing the relative effectiveness, since they involve the same number of executed comparisons for both algorithms. We put these settings into practice using as threshold for $CEP$ the total number of comparisons required by $CNP$. The outcomes with respect to $PC$ are presented in Table 2 and confirm that the node-centric algorithms achieve a significantly higher effectiveness than the edge-centric ones, across all datasets and weighting schemes.

In summary, the most appropriate meta-blocking settings for the application at hand depend on its performance requirements and the available resources (assuming the configuration of Section 3.3). The node-centric pruning schemes are suitable for applications emphasizing on effectiveness, provided that they can afford the high space requirements (these pruning schemes store a threshold or a certain number of comparisons *per entity*). They are also particularly useful for tasks that are inherently expressed in terms of entities (e.g., applications like social networks that seek duplicates for a specific subset of the input entities) and for entity collections that are expected to contain a large portion of duplicate profiles (i.e., there is a matching entity for most of the nodes). In contrast, the edge-centric pruning schemes are suitable for applications like incremental ER that focus on efficiency, especially when the portion of matching entities is expected to be rather low; in these settings, the top weighted edges are more likely to correspond to the few duplicate profiles.

**(iii)** *Weight vs. cardinality pruning criteria.* There is a clear pattern in the relative performance of weight and cardinality pruning thresholds for the configuration of Section 3.3: the former put more emphasis on effectiveness and the latter on efficiency. In fact, the combination of any weighting scheme with a cardinality threshold requires at least half the comparisons than its combination with the corresponding weight one, regardless of the selected pruning algorithm. In most of the cases, this difference amounts to a whole order of magnitude in the actual number of comparisons. Note, though, that this radical increase in efficiency is accompanied by a moderate difference in effectiveness, due to the efficacy of cardinality thresholds in distinguishing the matching comparisons from the superfluous ones. Comparing the $PQ$ of $CEP$ ($CNP$) with that of $WEP$ ($WNP$), we observe that the former is usually higher than the latter by a whole order of magnitude. Still, weight thresholds exhibit higher $PC$, reducing it — in the worst case — half as much as the corresponding meta-blocking settings with a cardinality criterion. Therefore, there is no dilemma when choosing the appropriate criterion with respect to the application requirements. Note, though, that this decision also depends on the available resources, since the cardinality criteria have higher memory requirements.

**(iv)** *Comparison between weighting schemes.* For $D_{BTC09}$, $ARCS$ consistently achieves the highest performance with respect to all block quality metrics, while the rest of the weighting schemes exhibit similar, but lower performance in most of the cases. For the Clean-Clean ER dataset, the choice depends on the functionality of the pruning criterion. In more detail, $ECBS$ offers a balanced choice for the weight pruning criteria, combining high efficiency enhancements with negligible reductions in $PC$. For the cardinality pruning criteria, where $RR$ remains stable across all weighting schemes, $EJS$ consistently achieves the (nearly) best efficiency-effectiveness balance, scoring the highest $PC$ values in most of the cases.

Of particular interest, though, is the comparison between the plain weighting schemes and their enhanced versions; that is, between $CBS$ and $ECBS$ as well as between $JS$ and $EJS$. The actual question is whether the more information included in the enhanced schemes leads to a better balance between $RR$ and $PC$ than the plain ones. The weight pruning criteria does not offer a clear answer; we can merely observe that the enhanced schemes offer lower $RR$ and lower $PQ$ at the benefit of higher $PC$. In contrast, the cardinality pruning criteria allow for a direct comparison: $RR$ is the same across all weighting schemes, but the enhanced ones achieve higher $PC$ in practically all the cases. $PQ$ also takes significantly higher values for $ECBS$ and $EJS$. We can conclude, therefore, that the enhanced schemes convey significant enhancements in the performance of $CBS$ and $JS$.

**(iv)** *Comparison with Iterative Blocking.* Before examining the performance of Iterative Blocking, it is worth clarifying that its functionality in the context of Clean-Clean ER is reduced to discarding part of the superfluous comparisons. In fact, it propagates all detected duplicates to the subsequently processed blocks and merely saves those comparisons that involve at least one entity that has been matched to some other. This approach conveys significant efficiency enhancements when applied to redundancy-positive block collections: its $RR$ exceeds 60% for $D_{movies}$ and 35% for $D_{infoboxes}$. All meta-blocking methods, though, achieve higher efficiency gains, as they have a broader scope, targeting all superfluous comparisons. This is also verified by $PQ$, which indicates that Iterative Blocking executes the highest portion of superfluous comparisons across both datasets. Its only advantage is that it incurs no impact on effectiveness. In practice, though, this is of minor importance, given that most meta-blocking approaches have limited cost in effectiveness in the context of Clean-Clean ER.

The real strength of Iterative Blocking lies in Dirty ER, especially in applications that involve equivalence classes of high cardinality. In these settings, it puts more emphasis on identifying additional matches, thus yielding the highest $PC$ among all methods. This is exactly the case with $D_{BTC09}$: although the original $PC$ is already high, amounting to 97%, Iterative Blocking increases it by more than 1%. The re-examination of large blocks, though, increases the number of executed comparisons and prevents significant enhancements in efficiency. Indeed, it merely saves around 1% of all comparisons in the case of $D_{BTC09}$. Thus, its efficiency is significantly lower than meta-blocking, which again discards more superflous comparisons.

In summary, Iterative Blocking is only appropriate for applications that place effectiveness in priority and are satisfied with rather conservative savings in efficiency. For the rest of them, meta-blocking offers a better balance between effectiveness and efficiency.

**Discussion.** In summary, we can conclude that among the weighting schemes, the Enhanced Common Blocks Scheme consistently offers a good balance between effectiveness and efficiency over Clean-Clean ER. For Dirty ER, though, the Aggregate Reciprocal Comparisons Scheme offers the best approach. We also observe that the node-centric approaches perform a shallow pruning that yields lower $PQ$ and $RR$ values than edge-centric ones. This allows them to retain almost intact the original effectiveness, especially when combined with weight thresholds. Therefore, applications that place more emphasis on effectiveness should opt for node-centric pruning schemes, while those focusing on efficiency should consider the edge-centric ones. Among the two types of pruning criteria, the weight thresholds are more robust with respect to effectiveness, while the cardinality thresholds are appropriate for applications emphasizing on efficiency, such as incremental ER.

### 4.3 Sensitivity Analysis

As mentioned above, the performance of pruning algorithms depends largely on the underlying pruning criterion — regardless of its scope or functionality. To examine how our pruning schemes behave as a function of their thresholds, we performed sensitivity analyses of $RR$ and $PC$ for all schemes over the three datasets of our study. In Figures 8(a) to (d), we present the behavior of each pruning algorithm in combination with a specific weighting scheme over $D_{movies}$ (for each algorithm, the rest of the weighting schemes demonstrated similar patterns and, thus, are omitted for brevity. Nevertheless, we tried to cover all of them, considering in each diagram a different one.). Every diagram was derived by incrementing the pruning threshold from $0.1{\cdot}t$ to $1.9{\cdot}t$ with a step of $0.1{\cdot}t$, where $t$ denotes the threshold derived from the configuration of Section 3.3 (e.g., the average edge weight in the case of $WEP$).

In every figure, we observe that there is a clear trade-off between $RR$ and $PC$. Higher thresholds increase $RR$ and reduce $PC$ for the weight pruning criteria, and vice versa for the cardinality ones. In fact, the evolution of $PC$ is practically linear for all pruning schemes. The same applies to $RR$ for the cardinality criteria, whereas for the weight ones, the linear evolution is preceded by a steep rise for the interval $[0.1{\cdot}t, 0.5{\cdot}t]$. The thresholds of Section 3.3 correspond to the vertical dotted line intersecting the middle of the $x$-axes. We observe that in every case, small variations in the size of $t$ lead to small variations in the resulting performance. This suggests that the threshold we selected for each pruning scheme achieves a good balance between effectiveness and efficiency. Thus, it provides a good basis for adjusting a meta-blocking method to the requirements of the application at hand. For example, an application employing $CEP$ could double the threshold specified by our approach in order to rise $PC$ by 10% for double as many comparisons.

In summary, the sensitivity analysis of Figures 8(a) to (d) demonstrate that our meta-blocking methods are robust with respect to the threshold configurations of Section 3.3.

### 4.4 Time Requirements of Meta-blocking

The real usefulness of meta-blocking depends on the relation between the time required for building and pruning the blocking graph and the time consumed while performing the (spared) pairwise comparisons. The goal of this section is to examine whether the former is significantly lower than the latter, thus justifying the use of our approaches. To this end, we evaluate the time requirements of meta-blocking using three measures:

- *Materialization Time* ($MT$) refers to the time required by the first two steps of metablocking, i.e., graph building and edge weighting.
- *Restructure Time* ($RT$) corresponds to the last two steps of meta-blocking, i.e., graph pruning and block collecting.
- *Comparison Time* ($CT$) indicates the time required for performing the (retained) pairwise comparisons.

As the baseline method, we consider the one that iterates over the input blocks, executing all the comparisons they entail, without any further processing (i.e., its processing time exclusively corresponds to $CT$, while $MT{=}RT{=}0$). For all methods, the similarity of entity profiles is defined as the Jaccard coefficient of their tokenized attribute values; any other entity comparison technique is also applicable, but this choice is orthogonal to the proposed method, thus not altering our experimental results.

The outcomes of our experiments are presented in Table 3. We notice the following patterns for the vast majority of meta-blocking approaches across all datasets: first, the overall processing time of the weighting pruning criteria is dominated by $CT$, with $MT$ and $RT$ merely accounting for a fraction of it. Exception to this rule is $ARCS$ in conjunction with $WEP$ and $WNP$, as the low discernibility of its weights ($\ll$0.1 in most of the cases) results in a time-consuming meta-blocking process. Second, there is a balance between $CT$ and $MT + RT$ for the cardinality pruning criteria, since they entail a very low number of comparisons with respect to the size of the graph. Again, $ARCS$ corresponds to the least efficient meta-blocking process.

We also notice that for every dataset, $MT$ and $RT$ take almost identical values for all weighting schemes, with the small variations corresponding to the different functionality of each weighting scheme. Regarding $CT$, we observe that it takes significantly lower values for the cardinality pruning criteria than for the weight ones. This overhead is caused not only by the lower number of comparisons retained by the former, but also by the fact that the latter iterate over all edges of the blocking graph during the comparisons phase.

In summary, we observe that all combinations of pruning schemes with a weighting one require significantly less time than the baseline method. For example, the most efficient meta-blocking techniques for $D_{movies}$ (i.e., $CEP$ in conjunction with $CBS$ or $JS$) are 35 times faster than the baseline. Even the most time-consuming meta-blocking settings for each dataset run at least 2 times faster than the baseline. As explained in Section 3.1, this should be attributed to the efficient materialization of the blocking graph, which involves lower complexity than the string-based techniques for comparing entity profiles.

Note that optimization techniques can be integrated into the implementation of the meta-blocking and the entity comparison methods. For instance, during the pruning of the blocking graph, edges with weights lower than the specified threshold can be identified more efficiently with the help of prefix filtering. No such technique was considered, though, in our experimental study, since it is orthogonal to our evaluation.

## 5 Conclusions

In this paper, we introduced meta-blocking as a generic task that can be applied on top of any blocking method to increase its efficiency at a minor cost in effectiveness. We described a family of techniques, at the core of which lies the blocking graph; they prune its edges with the lowest weight in order to derive a new set of blocks that sacrifices a negligible amount of matches to save a large number of comparisons. We thoroughly evaluated all combinations of the proposed techniques over two large, real-world datasets. The results demonstrate the high efficiency enhancements conveyed by our meta-blocking techniques, with the Weight Node Pruning involving two orders of magnitude less comparisons at a minor cost in *PC* (less than 3% reduction). In absolute values, meta-blocking helps process the original set of blocks 10 to 50 times faster, reducing the required comparisons by a whole order of magnitude.

In the future, we plan to enhance the efficiency of meta-blocking through the incorporation of schema information that depends on the underlying application. We also acknowledge that meta-blocking depends on the level of redundancy entailed by the underlying block collection, which — for some block building methods — can be configured by tuning the corresponding parameter(s). Thus, we intend to investigate the effect of these parameters on the performance of meta-blocking. Last but not least, we will study the interplay of meta-blocking with blocking methods that consider profile merges in the context of Dirty ER, such as HARRA [17] and Iterative Blocking [30].

## References

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

[2] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *SIGKDD*, volume 3, pages 25–27, 2003.

[3] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, pages 87–96, 2006.

[4] C. Bizer, T. Heath, T. Berners-Lee, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

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

[6] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[7] T. de Vries, H. Ke, S. Chawla, and P. Christen. Robust record linkage blocking using suffix arrays. In *CIKM*, pages 1565–1568, 2009.

[8] A. Doan and A. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 26(1):83–94, 2005.

[9] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, pages 85–96, 2005.

[10] U. Draisbach and F. Naumann. A comparison and generalization of blocking and windowing algorithms for duplicate detection. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, pages 51–56, 2009.

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

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

[13] L. Getoor and C. Diehl. Link mining: a survey. *SIGKDD Expl.*, 7(2):3–12, 2005.

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

[15] A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of dataspace systems. In *PODS*, pages 1–9, 2006.

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

[17] H. Kim and D. Lee. HARRA: fast iterative hashed record linkage for large-scale data collections. In *EDBT*, pages 525–536, 2010.

[18] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, pages 802–803, 2006.

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

[20] W. Masek and M. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System sciences*, 20(1):18–31, 1980.

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

[22] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *AAAI*, pages 440–445, 2006.

[23] J. Nin, V. Muntés-Mulero, N. Martínez-Bazan, and J.-L. Larriba-Pey. On the use of semantic blocking techniques for data cleansing and integration. In *IDEAS*, pages 190–198, 2007.

[24] A. Ouksel and A. Sheth. Semantic interoperability in global information systems: A brief introduction to the research area and the special section. *SIGMOD Record*, pages 5–12, 1999.

[25] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *WSDM*, pages 535–544, 2011.

[26] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl. To compare or not to compare: making entity resolution more efficient. In *SWIM Workshop*, 2011.

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

[28] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *KDD*, pages 350–359, 2002.

[29] S. Whang, D. Marmaros, and H. Garcia-Molina. Pay-as-you-go entity resolution. *IEEE Trans. Knowl. Data Eng. (to appear)*, 2012.

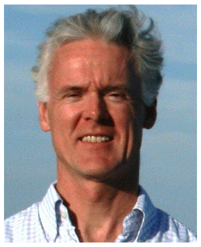
[30] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, pages 219–232, 2009.

[31] S. Yan, D. Lee, M.-Y. Kan, and C. L. Giles. Adaptive sorted neighborhood methods for efficient record linkage. In *JCDL*, pages 185–194, 2007.

**George Papadakis** is a PhD student at the Leibniz University of Hanover. He holds a Diploma in Computer Engineering from the National Technical University of Athens (NTUA) and has worked at the NCSR “Demokritos”, NTUA and the L3S Research Center. His research interests include entity resolution and web data mining. He has received the best paper award from ACM Hypertext 2011.

**Georgia Koutrika** is a senior researcher at HP Labs, Palo Alto. Prior to that, she was a postdoctoral researcher at IBM Almaden Research Center and Stanford University. She holds a PhD in Computer Science from the University of Athens in Greece. Her research interests include personalized search, recommendations, user modeling, social media, information extraction, resolution and integration, and search interfaces.

**Themis Palpanas** is a professor of computer science at the University of Trento, Italy. Before that he worked at the IBM T.J. Watson Research Center, and has also worked for the University of California at Riverside, Microsoft Research and IBM Almaden Research Center. He is the author of five US patents, three of which are part of commercial products. He has received three best paper awards and is General Chair for VLDB 2013.



**Wolfgang Nejdl** received M.Sc. and Ph.D. degrees from the Technical University of Vienna. Currently, he is a professor of computer science at the University of Hanover, where he leads the L3S Research Center and the Distributed Systems Institute/Knowledge-Based Systems. His research focuses on information retrieval, peer-to-peer infrastructures, databases, technology-enhanced learning, and artificial intelligence.

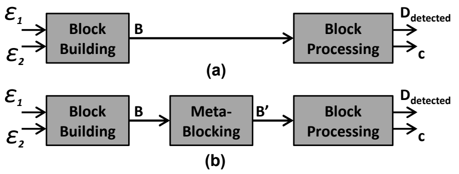


Fig. 5. (a) The process of traditional blocking-based ER, and (b) the blocking-based ER with meta-blocking. In both cases, the input comprises the entity collections to be resolved ($\mathcal{E}_1$ and $\mathcal{E}_2$), while the output consists of the detected duplicates $D_{detected}$ and the computational cost $c$ (i.e., number of executed comparisons).

## Appendix

## Related Work

There is a large amount of work on entity resolution ranging from string similarity metrics [6] to methods relying on transformations [28] and entity relationships [9]. Analytical overviews can be found in these surveys [8], [11], [13], tutorials [18], [24] and books [**?**], [**?**]. Due to their quadratic complexity, ER methods typically scale to large data collections through blocking. The blocking-based ER process conceptually consists of two main steps: block building and block processing (see Figure 5(a)).

*Block building* receives as input two entity collections ($\mathcal{E}_1$ and $\mathcal{E}_2$ in Figure 5(a)) and creates a set of blocks $\mathcal{B}$. Methods of this type are categorized according to two orthogonal criteria: their relation to redundancy and to schema information. The former criterion was analyzed in Section 1, while the latter distinguishes them into *schema-based* and *schema-agnostic* blocking methods; that is, into those techniques that integrate schema information in their functionality and those that completely disregard such evidence. The resulting two-dimensional taxonomy of block building methods is illustrated in Figure 6.

On the one hand, schema-based blocking methods extract from each entity a blocking key that summarizes the values of selected attributes. Entity profiles are then placed in blocks on the basis of equal or similar blocking keys. Schema-based blocking methods include Sorted Neighborhood [16], bi-grams [2] and q-grams [14] blocking, Suffix Array [1], [7], HARRA [17], and Canopy Clustering [21]. A comparative analysis can be found in [5]. As this study points out, one of their major drawbacks is the fine-tuning of multiple parameters [7]. To ameliorate this issue, automatic methods can be trained to select the optimal parameter values [3], [22].

On the other hand, schema-agnostic blocking creates blocks solely on the basis of attribute values, i.e., without knowledge of the input schema(ta). Semantic Indexing [23] creates blocks based exclusively on the relationships between entity profiles. Attribute-agnostic Blocking creates a distinct block for each token shared by at least two input entity profiles [25]. For RDF data, Total Description exploits semantics in the entity URIs, links between entities and tokens in the literal values of every profile [27]. Both techniques do not require tuning (i.e., their functionality is parameter-free) and exhibit high robustness and effectiveness, due to the high levels of redundancy they involve.

| | Redundancy-free | Redundancy-bearing | | |
|---|---|---|---|---|
| | | Redundancy-negative | Redundancy-neutral | Redundancy-positive |
| **Schema-based** | Standard Blocking [13] | Canopy Clustering [22] | (Adaptive) Sorted Neighborhood [17,33] | bi-/q-grams blocking [2,15] Suffix Array [1,8] HARRA [18] |
| **Schema-agnostic** | - | - | Semantic Indexing [25] | Attribute-agnostic [27] Total Description [29] |

Fig. 6. Two-dimensional taxonomy of block building methods.

*Block processing* receives as input a set of blocks $\mathcal{B}$ and produces as output the set of detected duplicates $D_{detected}$ along with their computational cost $c$, in terms of the number of executed comparisons (see Figure 5(a)). Its goal is to process the input set of blocks in such a way that minimizes $c$ without any significant impact on the cardinality of $D_{detected}$. This can be achieved by eliminating the redundant and the superfluous comparisons contained in $\mathcal{B}$. To this end, Block Purging [25] discards the largest blocks, while Block Scheduling [25] sorts blocks according to a probabilistic measure that estimates their likelihood to contain duplicates. Thus, it forms the basis for applying Block Pruning [25] and Duplicate Propagation [30]; the former terminates the entire processing as early as possible, while the latter maximizes the number of superfluous comparisons that can be spared by the early detection of duplicate profiles. On another line of research, *Iterative Blocking* [30] propagates the latest match decisions to all associated blocks: every time two entity profiles are found duplicates, they are replaced by the merged profile in all blocks containing either of them. These blocks are then scheduled for processing, even if they have already been examined. In this way, a block collection is processed iteratively in order to increase the matching accuracy (and, thus, the blocking effectiveness) and to spare repeated comparisons.

The proposed meta-blocking procedure is fundamentally different from both block building and block processing. It is a specialized procedure applicable to redundancy-positive block building methods. Its input comprises the set of blocks $\mathcal{B}$ created by such a method and its output is a new set of blocks $\mathcal{B}'$ that involves fewer comparisons than $\mathcal{B}$, while placing (almost) the same number of matching entity profiles in at least one block. Block Purging and Block Pruning have a similar interface, but their functionality is restricted in discarding some of the input blocks. In contrast, meta-blocking techniques aim at *restructuring* the given block collection $\mathcal{B}$ based on the block-to-entity associations it entails. For this reason, it is performed between block building and block processing, improving the output of the former in order to facilitate the performance of the latter, as shown in Figure 5(b). A similar idea was explored in Comparison Pruning [26], which discards comparisons between entity profiles that share a rather small portion of blocks in the context of redundancy-positive methods. Thus, it can be viewed as a specific instantiation of our meta-blocking framework; in fact, it is equivalent to applying *WEP* (see Section 3.3.1) on a blocking graph with Jaccard similarities as weights (see Section 3.2).

## Experimental Outcomes

| | $\mathbf{D_{movies}}$ DBPedia | $\mathbf{D_{movies}}$ IMDB | $\mathbf{D_{infoboxes}}$ $\mathbf{DBP_1}$ | $\mathbf{D_{infoboxes}}$ $\mathbf{DBP_2}$ | $\mathbf{D_{BTC09}}$ |
|---|---|---|---|---|---|
| Entities | 27,615 | 23,182 | $1,19{\cdot}10^6$ | $2,16{\cdot}10^6$ | 253,353 |
| Name-Value Pairs | 186,013 | 816,012 | $1.75{\cdot}10^7$ | $3.67{\cdot}10^7$ | $1,60{\cdot}10^6$ |
| Blocks | 40,430 | | $1.21{\cdot}10^6$ | | 106,462 |
| *BC* | 22.52 | | 15.38 | | 7.45 |
| *CC* | $4.27{\cdot}10^{-2}$ | | $1.29{\cdot}10^{-3}$ | | $1.44{\cdot}10^{-2}$ |
| Brute Force Comp. | $6.40{\cdot}10^8$ | | $2.58{\cdot}10^{12}$ | | $3.21{\cdot}10^{10}$ |
| Block Comp. | $2.67{\cdot}10^7$ | | $3.98{\cdot}10^{10}$ | | $1.31{\cdot}10^8$ |
| Original *RR* | 95.83% | | 98.46% | | 99.59% |
| Existing Matches | 22,405 | | 892,586 | | 10,653 |
| Original *PC* | 99.39% | | 99.89% | | 96.94% |
| Original *PQ* | $9.83{\cdot}10^{-4}$ | | $2.24{\cdot}10^{-5}$ | | $7.89{\cdot}10^{-5}$ |
| Edges | $2.26{\cdot}10^7$ | | $3.41{\cdot}10^{10}$ | | $7.77{\cdot}10^7$ |
| Nodes | $5.06{\cdot}10^4$ | | $3.33{\cdot}10^6$ | | $2.53{\cdot}10^5$ |

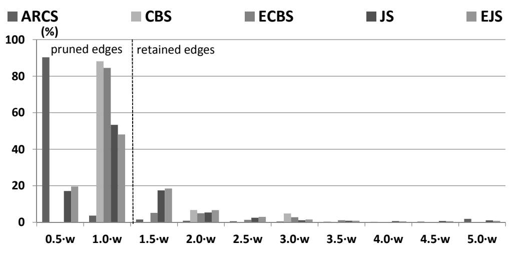
TABLE 1
Overview of the evaluation datasets.

| | $\mathbf{D_{movies}}$ | | $\mathbf{D_{infoboxes}}$ | | $\mathbf{D_{BTC09}}$ | |
|---|---|---|---|---|---|---|
| | $PC_{CEP}$ | $PC_{CNP}$ | $PC_{CEP}$ | $PC_{CNP}$ | $PC_{CEP}$ | $PC_{CNP}$ |
| *ARCS* | 89.16% | **94.13%** | 83.82% | **96.87%** | 93.22% | **95.60%** |
| *CBS* | 80.42% | **95.20%** | 60.46% | **96.34%** | 31.97% | **88.70%** |
| *ECBS* | 87.17% | **96.69%** | 67.85% | **97.72%** | 65.78% | **86.25%** |
| *JS* | 89.22% | **94.93%** | 86.02% | **96.86%** | 35.97% | **83.79%** |
| *EJS* | 91.03% | **95.98%** | 85.26% | **97.18%** | 51.85% | **84.50%** |

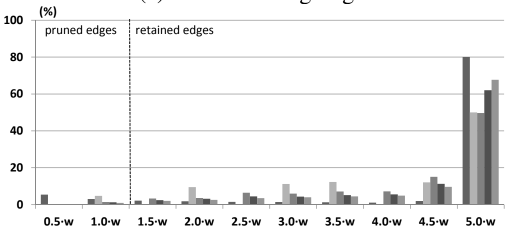
TABLE 2
Comparing effectiveness between *CEP* and *CNP* for the same number of comparisons across all datasets.

| | | $\mathbf{D_{movies}}$ (minutes) | | | | $\mathbf{D_{infoboxes}}$ (hours) | | | | $\mathbf{D_{BTC09}}$ (minutes) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *MT* | *RT* | *CT* | $\sum$ | *MT* | *RT* | *CT* | $\sum$ | *MT* | *RT* | *CT* | $\sum$ |
| **Baseline** | | .0 | .0 | 14 | 14 | .0 | .0 | 128 | 128 | 0 | 0 | 111 | 111 |
| | *ARCS* | .1 | .6 | 1.0 | 1.6 | 3.2 | 24.4 | 25.7 | 53.3 | **.2** | **2.5** | **5.9** | **8.7** |
| **W** | *CBS* | **.1** | **.1** | **.9** | **1.1** | **3.3** | **7.0** | **21.2** | **31.6** | .2 | 1.5 | 19.8 | 21.5 |
| **E** | *ECBS* | .1 | .2 | 1.2 | 1.4 | 3.1 | 6.7 | 30.8 | 40.6 | .2 | 1.8 | 17.2 | 19.2 |
| **P** | *JS* | .1 | .1 | 2.1 | 2.3 | 3.2 | 6.0 | 51.2 | 60.4 | .2 | 1.9 | 20.2 | 22.3 |
| | *EJS* | .1 | .2 | 2.3 | 2.5 | 3.2 | 6.7 | 52.0 | 62.0 | .2 | 2.0 | 20.2 | 22.4 |
| | *ARCS* | .1 | .6 | 1.3 | 1.9 | 3.5 | 25.9 | 28.7 | 58.1 | **.2** | **2.7** | **21.5** | **24.5** |
| **W** | *CBS* | **.1** | **.1** | **1.0** | **1.2** | **3.2** | **6.2** | **24.4** | **33.9** | .2 | 1.7 | 24.3 | 26.3 |
| **N** | *ECBS* | .1 | .2 | 2.1 | 2.4 | 3.6 | 7.5 | 33.4 | 44.6 | .2 | 2.1 | 30.9 | 33.2 |
| **P** | *JS* | .1 | .1 | 3.0 | 3.2 | 3.5 | 7.0 | 55.4 | 65.9 | .2 | 2.1 | 37.7 | 40.1 |
| | *EJS* | .1 | .2 | 3.6 | 3.8 | 3.6 | 8.0 | 58.5 | 70.1 | .2 | 2.4 | 39.6 | 42.1 |
| | *ARCS* | .1 | .6 | .2 | .9 | 3.2 | 24.5 | .1 | 27.9 | .2 | 2.6 | .8 | 3.6 |
| **C** | *CBS* | .1 | .1 | .2 | .4 | 4.2 | 7.4 | .1 | 11.7 | **.2** | **1.5** | **.8** | **2.5** |
| **E** | *ECBS* | .1 | .2 | .2 | .4 | 4.4 | 8.0 | .1 | 12.6 | .2 | 1.9 | .8 | 2.9 |
| **P** | *JS* | .1 | .2 | .2 | .4 | 4.2 | 7.5 | .1 | 11.8 | .2 | 1.9 | .8 | 2.9 |
| | *EJS* | **.1** | **.2** | **.2** | **.4** | **3.2** | **7.1** | **.1** | **10.4** | .2 | 2.2 | .8 | 3.2 |
| | *ARCS* | .1 | .6 | .3 | 1.0 | 3.2 | 24.7 | .2 | 28.1 | .2 | 2.7 | 1.5 | 4.4 |
| **C** | *CBS* | **.1** | **.1** | **.3** | **.5** | 3.8 | 6.7 | .2 | 10.8 | **.2** | **1.6** | **1.5** | **3.3** |
| **N** | *ECBS* | .1 | .2 | .3 | .6 | 3.7 | 6.9 | .2 | 10.9 | .2 | 2.0 | 1.5 | 3.6 |
| **P** | *JS* | .1 | .2 | .3 | .6 | **3.2** | **6.3** | **.2** | **9.8** | .2 | 1.9 | 1.5 | 3.6 |
| | *EJS* | .1 | .2 | .3 | .6 | 3.2 | 7.1 | .2 | 10.6 | .2 | 2.3 | 1.5 | 4.0 |

TABLE 3
Processing time for all meta-blocking methods over the three datasets of our experimental study.

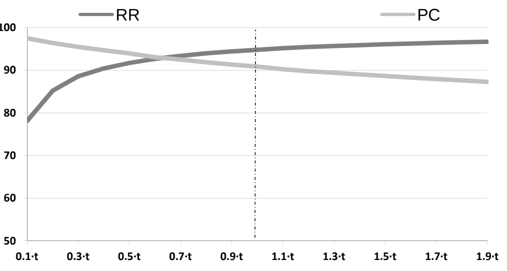


(a) Non-matching edges.

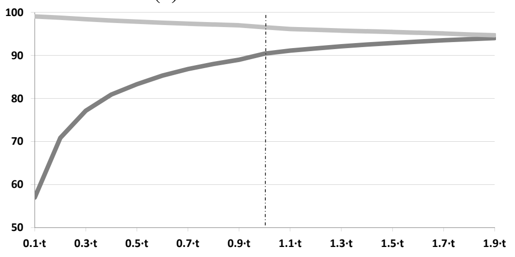


(b) Matching edges.

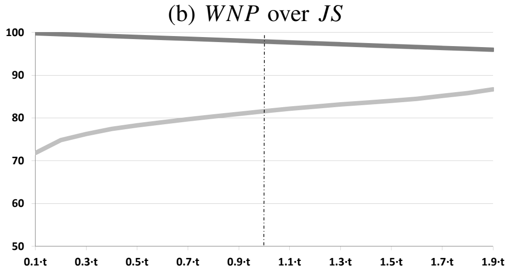
Fig. 7. Normalized histograms of the weight distributions in all blocking graphs of $D_{movies}$, where $w$ denotes the average edge weight of the blocking graph for each weighting scheme.

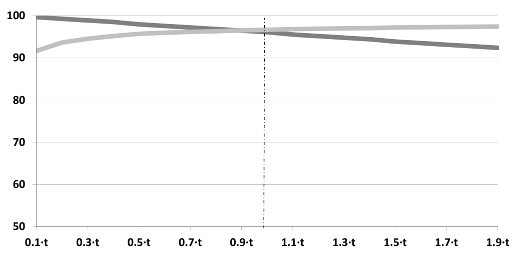


(a) *WEP* over *ARCS*



(b) *WNP* over *JS*





(d) *CNP* over *CBS*

Fig. 8. Sensitivity analysis of every pruning algorithm in conjunction with a specific weighting scheme.

| | $D_{movies}$ | | | | | $D_{infoboxes}$ | | | | | $D_{BTC09}$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Comp. | *RR* | *PC* | $\Delta PC$ | *PQ* | Comp. | *RR* | *PC* | $\Delta PC$ | *PQ* | Comp. | *RR* | *PC* | $\Delta PC$ | *PQ* |
| | ($\times 10^6$) | (%) | (%) | (%) | ($\times 10^{-2}$) | ($\times 10^8$) | (%) | (%) | (%) | ($\times 10^{-4}$) | ($\times 10^7$) | (%) | (%) | (%) | ($\times 10^{-4}$) |
| Iterative Bl. | 10.41 | 61.06 | 99.39 | 0 | 0.21 | 255.94 | 35.67 | 99.89 | 0 | 0.35 | 12.98 | 0.84 | 98.22 | 1.32 | 0.81 |
| *ARCS* | 1.38 | 94.82 | 90.89 | -8.55 | 1.47 | 2.85 | 99.28 | 92.45 | -7.45 | 29.00 | **0.41** | **99.35** | **94.77** | **-2.24** | **24.85** |
| *CBS* | 2.71 | 89.88 | 94.68 | -4.74 | 0.78 | 33.97 | 91.46 | 95.47 | -4.42 | 2.51 | 2.16 | 96.57 | 86.84 | -10.42 | 4.29 |
| *ECBS* | **3.52** | **86.82** | **97.95** | **-1.45** | **0.62** | **57.71** | **85.50** | **99.66** | **-0.23** | **1.54** | 1.81 | 97.12 | 86.60 | -10.67 | 5.08 |
| *JS* | 6.71 | 74.90 | 97.93 | -1.46 | 0.33 | 112.21 | 71.80 | 99.73 | -0.16 | 0.79 | 2.15 | 96.58 | 87.13 | -10.12 | 4.31 |
| *EJS* | 7.34 | 72.54 | 98.32 | -1.07 | 0.30 | 110.14 | 72.32 | 99.77 | -0.11 | 0.81 | 2.13 | 96.61 | 89.01 | -8.18 | 4.45 |
| **(a) WEP** | | | | | | | | | | | | | | | |
| *ARCS* | 2.55 | 90.44 | 96.55 | -2.86 | 0.85 | 14.84 | 96.27 | 99.41 | -0.48 | 5.98 | **2.25** | **96.43** | **95.72** | **-1.26** | **4.54** |
| *CBS* | 2.86 | 89.31 | 97.19 | -2.21 | 0.76 | 35.65 | 91.04 | 99.35 | -0.54 | 2.49 | 2.69 | 95.72 | 91.46 | -5.66 | 3.62 |
| *ECBS* | **6.92** | **74.10** | **98.64** | **-0.75** | **0.32** | **99.37** | **75.02** | **99.75** | **-0.14** | **0.90** | 3.42 | 94.56 | 91.13 | -5.99 | 2.84 |
| *JS* | 10.00 | 62.59 | 98.68 | -0.71 | 0.22 | 195.93 | 50.76 | 99.87 | -0.02 | 0.46 | 4.22 | 93.29 | 91.43 | -5.68 | 2.31 |
| *EJS* | 11.81 | 55.77 | 99.16 | -0.23 | 0.19 | 199.96 | 49.74 | 99.88 | -0.01 | 0.45 | 4.41 | 93.00 | 92.52 | -4.56 | 2.24 |
| **(b) WNP** | | | | | | | | | | | | | | | |
| *ARCS* | 0.57 | 97.87 | 82.75 | -16.74 | 3.25 | 0.26 | 99.94 | 79.46 | -20.46 | 276.83 | **0.09** | **99.85** | **92.17** | **-4.92** | **103.99** |
| *CBS* | 0.57 | 97.87 | 75.78 | -23.75 | 2.98 | 0.26 | 99.94 | 51.71 | -48.37 | 179.68 | 0.09 | 99.85 | 24.07 | -75.17 | 27.16 |
| *ECBS* | 0.57 | 97.87 | 81.58 | -17.92 | 3.20 | 0.26 | 99.94 | 62.14 | -37.79 | 216.49 | 0.09 | 99.85 | 42.81 | -56.05 | 48.07 |
| *JS* | 0.57 | 97.87 | 79.12 | -20.40 | 3.11 | 0.26 | 99.94 | 82.09 | -17.83 | 285.98 | 0.09 | 99.85 | 25.77 | -99.55 | 29.07 |
| *EJS* | **0.57** | **97.87** | **84.87** | **-14.61** | **3.33** | **0.26** | **99.94** | **79.61** | **-20.30** | **277.37** | 0.09 | 99.85 | 45.85 | -52.71 | 51.73 |
| **(c) CEP** | | | | | | | | | | | | | | | |
| *ARCS* | 1.10 | 95.88 | 94.13 | -5.39 | 1.91 | 0.50 | 99.88 | 96.87 | -3.02 | 174.63 | **0.18** | **99.72** | **95.60** | **-1.38** | **58.22** |
| *CBS* | 1.10 | 95.88 | 95.20 | -3.48 | 1.95 | 0.50 | 99.88 | 96.34 | -3.56 | 173.68 | 0.18 | 99.72 | 88.70 | -8.50 | 54.02 |
| *ECBS* | 1.10 | 95.88 | 96.69 | -2.71 | 1.97 | 0.50 | 99.88 | 97.72 | -2.17 | 176.17 | 0.18 | 99.72 | 84.34 | -11.03 | 52.53 |
| *JS* | 1.10 | 95.88 | 94.93 | -4.45 | 1.93 | 0.50 | 99.88 | 96.86 | -3.03 | 174.62 | 0.18 | 99.72 | 83.79 | -13.57 | 51.03 |
| *EJS* | **1.10** | **95.88** | **95.98** | **-3.43** | **1.95** | **0.50** | **99.88** | **97.18** | **-2.71** | **175.19** | 0.18 | 99.72 | 84.50 | -12.83 | 51.46 |
| **(d) CNP** | | | | | | | | | | | | | | | |

TABLE 4
Performance of all pruning schemes in combination with all weighting schemes over the three datasets of our study.