Supervised Meta-blocking

George Papadakis1, George Papastefanatos1, Georgia Koutrika5
1 HP Labs, USA koutrika@hp.com
5 Institute for the Management of Information Systems, Research Center “Athena”, Greece (gpapadis,gpapas)@imis.athena-innovation.gr

ABSTRACT

Entity Resolution matches mentions of the same entity. Being an expensive task for large data, its performance can be improved by blocking, i.e., grouping similar entities and comparing only entities in the same group. Blocking improves the run-time of Entity Resolution, but it still involves unnecessary comparisons that limit its performance. Meta-blocking is the process of restructuring a block collection in order to prune such comparisons. Existing unsupervised meta-blocking methods use simple pruning rules, which offer a rather coarse-grained filtering technique that can be conservative (i.e., keeping too many unnecessary comparisons) or aggressive (i.e., pruning good comparisons). In this work, we introduce supervised meta-blocking techniques that learn classification models for distinguishing promising comparisons. For this task, we propose a small set of generic features that combine a low extraction cost with high discriminatory power. We show that supervised meta-blocking can achieve high performance with small training sets that can be manually created. We analytically compare our supervised approaches with baseline and competitor methods over 10 large-scale datasets, both real and synthetic.

1. INTRODUCTION

Entity Resolution (ER) is the process of finding and linking different instances (profiles) of the same real-world entity [9]. It is an inherently quadratic task, since, in principle, each entity profile has to be compared with all others. For Entity Resolution to scale to large datasets, blocking is used to group similar entities into blocks so that profile comparisons are limited within each block. Blocking methods may place each entity profile into only one block, forming disjoint blocks, or into multiple blocks, creating redundancy [4].

Redundancy is typically used for reducing the likelihood of missed matches – especially for noisy, highly heterogeneous data [9, 21]. In particular, redundancy-positive blocking is based on the intuition that the more blocks two entities share, the more likely they match [22]. To illustrate, consider the profiles in Figure 1(a): profiles p1 and p3 correspond to the same person and so do p2 and p4. As an example of a redundancy-based blocking method, let us consider Token Blocking [21], which creates one block for every distinct token that appears in at least two profiles. The resulting block collection is shown in Figure 1(b). We observe that both pairs of matching profiles can be detected, as they co-occur in at least one block.

However, redundancy brings about repeated comparisons between the same entity profiles in different blocks. In the example of Figure 1(b), block b1 repeats the comparison contained in block b2, while b5 repeats the comparison in b6. Hence, b2 and b5 contain one redundant comparison each. Furthermore, there are several comparisons between non-matching entities, which we call superfluous comparisons. Block b3 entails 3 superfluous comparisons between the non-matching profiles p4, p5 and p6. In b6, all 3 comparisons involving p6 are superfluous, while the rest are redundant, repeated in b5. Overall, while blocking improves entity resolution times, it still involves unnecessary comparisons that limit its performance: superfluous ones between non-matching entities, and redundant ones, which repeatedly compare the same entity profiles. In our example, the total number of comparisons in the blocks of Figure 1(b) is 13 compared to 15 of the brute-force method. This number could be further reduced – without affecting the recall of blocking-based ER – by avoiding the redundant and the superfluous comparisons.

Meta-blocking is a method that takes as input a redundancy-positive block collection and transforms it into a new block collection that generates fewer comparisons, but keeps most of the detected duplicates [22]. To achieve this, existing meta-blocking techniques operate in two phases. First, they map the input block collection to a graph, called blocking graph; its nodes are the entity profiles, while its edges connect two nodes if the corresponding profiles co-occur in at least one block. By definition, the graph eliminates all redundant comparisons: each pair of co-occurring profiles is connected with a single edge, which means that they will be compared only once. In the second phase, meta-blocking techniques use the graph to prune superfluous comparisons. For this task, each edge is assigned a weight leveraging the fundamental property of redundancy-positive block collections that the similarity of two entity profiles is proportional to their co-occurrences in blocks. High...
weights are given to the matching edges (i.e., edges likely connecting duplicates) and lower weights to the non-matching ones.

As an example, the blocks in Figure 1(b) can be mapped to the blocking graph depicted in Figure 2(a). The edge weights are typically defined in the interval [0,1] through normalization, but for simplicity, we consider that each edge weight in this example is equal to the number of blocks shared by its adjacent entities. Different pruning algorithms can be used to remove edges with low weights and hence discard part of the superfluous comparisons. For example, one such strategy, called Weight Edge Pruning, discards all edges having a weight lower than the average edge weight across the entire graph [22]. For the blocking graph of Figure 2(a), the average edge weight is 1.625. The resulting pruned blocking graph is shown in Figure 2(b). The output block collection is generated from the pruned blocking graph by placing the adjacent entities of every edge into a separate block as shown in Figure 2(c). As the result of meta-blocking, the new block collection contains just 5 comparisons and does not miss any matches.

Existing meta-blocking methods use simple pruning rules such as “if weight-threshold then discard edge” for removing comparisons. Consequently, they face two challenges: assigning representative weights to edges and choosing a good threshold for removing edges. We argue that determining if an edge is a good candidate for removal is in fact a multi-criteria decision problem. Combining these criteria into a single scalar value inevitably misses valuable information. Furthermore, pruning based on a single threshold on the weights is a rather coarse-grained filtering technique that can be conservative (i.e., keeping many superfluous comparisons) or aggressive (i.e., pruning good comparisons). In our example in Figure 2(c), the final block collection retains 3 superfluous comparisons in $b_i'$, $b_i''$ and $b_i'''$; increasing the threshold so as to further reduce these comparisons would prune the matching comparisons, as well, because they have the same weight as the superfluous ones.

In this paper, we argue that accurate identification of non-matching edges requires learning composite pruning models from the data. We formalize meta-blocking as a binary classification task, where the goal is to identify matching and non-matching edges. We propose supervised meta-blocking techniques that compose generic, schema-agnostic information about the co-occurring entities into comprehensive feature vectors, instead of summarizing it into unilateral weights, as unsupervised methods do.

For example, the blocks of Figure 1(b) can be mapped to the blocking graph of Figure 3(a), where each edge is associated with a feature vector $[a_1, a_2]$. The feature $a_1$ is the number of common blocks shared by the adjacent entities, and $a_2$ is the total number of comparisons contained in these blocks. The resulting feature vectors are fed into a classification algorithm that learns composite rules (or models) to effectively distinguish matching and non-matching edges. In our example, a composite rule could look like “if $a_1 \leq 2$ & $a_2 > 5$ then discard edge”, capturing the intuition that the more blocks two profiles share and the smaller these blocks are, the more likely the profiles match. Figure 3(b) shows the graph generated by this rule, and Figure 3(c) depicts the resulting blocks; compared to the blocks in Figure 2(c), they have no superfluous comparisons, thus achieving higher efficiency for the same recall.

We identify and examine three aspects that determine the performance of supervised meta-blocking techniques: (a) the set of features annotating the edges of the blocking graph, (b) the training data, and (c) the classification algorithm and its configuration.

Using more features may help make the pruning of the non-matching edges more accurate. However, the computational cost for meta-blocking gets higher. Moreover, the classification features should be generic enough to apply to any redundancy-positive block collection. With these issues in mind, we propose a small set of generic features that combine a low extraction cost with high discriminatory power and we evaluate their performance using real data. Furthermore, to facilitate the understanding of the space of possible features, we divide it according to five dimensions.

Selecting training data, we face two issues. The first one is a class imbalance problem: the vast majority of the edges in a blocking graph are non-matching. In order to build representative training sets, we select the most suitable technique for our task among established solutions. The second issue regards the training set size. In general, large training sets increase the accuracy and robustness of the learned model. However, they yield complex, inefficient classifiers that require time-consuming training. In addition, the manual creation of large training sets in the absence of ground-truth is a painful and challenging process. We show that we can achieve high performance with small training sets that can be manually created making supervised meta-blocking a practical solution.

We consider a representative sample of state-of-the-art classifiers: Naive Bayes, Bayesian Networks, Decision Trees and Support Vector Machines. We show that our supervised techniques are robust with respect to different classifiers and their configurations by examining their performance over several large-scale datasets.

Finally, we evaluate the performance of supervised meta-blocking by comparing to (a) the brute-force Entity Resolution, which executes all comparisons included in the input set of blocks, (b) the top-performing unsupervised meta-blocking methods [22], and (c) the iterative blocking [25]. Note that the iterative blocking constitutes the only other method in the literature that, similarly to meta-blocking, receives an existing block collection and aims at processing it in a way that improves its original performance: it propagates every detected pair of duplicates to all associated blocks in order to identify additional matches and to save unnecessary comparisons. We perform a scalability analysis, which involves 7 large-scale synthetic datasets of various sizes, ranging from 10 thousand to 2 million entities. Our experiments demonstrate that our supervised techniques exhibit significantly better time efficiency than the best alternatives, while achieving equivalent recall.

In summary, this paper makes the following contributions:

- We formalize supervised meta-blocking as a classification problem and we demonstrate how it can be used to significantly enhance the quality of a redundancy-positive block collection.
• We map the space of possible classification features along five dimensions and we select a small set of generic features that combine a low extraction cost with high discriminatory power. We evaluate their performance using real data.

• We show that small training sets, which can be manually created, can achieve high performance, making supervised meta-blocking a practical solution for Entity Resolution.

• We show that our supervised techniques are robust with respect to different classifiers and their configurations by examining their performance over several large-scale datasets.

• We perform a thorough scalability analysis, comparing supervised meta-blocking against the best competitor approaches.

The rest of the paper is structured as follows. Section 2 presents related work. Section 3 provides a brief overview of unsupervised meta-blocking. Section 4 introduces supervised meta-blocking, and Section 5 describes the real-world datasets and the metrics used in the evaluation. Sections 6 and 7 cover feature and training set selection, while in Section 8, we fine-tune the classification algorithms. Section 9 experimentally compares supervised meta-blocking with competitor techniques and finally, Section 10 concludes the paper.

2. RELATED WORK

There is a large body of work on Entity Resolution [9, 19]. Blocking techniques group similar entities into blocks so that profile comparisons are limited within each block. These methods can be distinguished into schema-based and schema-agnostic ones.

Schema-based methods (e.g., Sorted Neighborhood [12], Suffix Array [7], HARMA [14], Canopy Clustering [17], and q-grams blocking [10]) group entities based on knowledge about the semantics of their attributes. These approaches are only suitable for homogeneous information spaces, like databases, where the quality of the schema is known a-priori. In contrast, schema-agnostic blocking techniques cluster entities into blocks without requiring any knowledge about the underlying schema(s). For instance, in Token Blocking [21], every token that is shared by at least two entities creates an individual block. Total Description [20] improves on Token Blocking by considering the most discriminative parts of entity URIs instead of all their tokens. In the same category fall Attribute Clustering [21] and TypMatch [16]. These techniques are preferred in the context of heterogeneous information spaces, which involve large volumes of noisy, semi-structured data that are loosely bound to various schemata [11].

Both schema-based and schema-agnostic blocking methods usually produce redundancy-positive block collections [22]. Meta-blocking operates on top of them, improving the balance between precision and recall by restructuring the block collection [22].

All the aforementioned approaches rely on an unsupervised functionality. Supervised learning has been applied to blocking-based ER with the purpose of fine-tuning the configuration of schema-based blocking methods: in [1, 18], the authors propose methods for learning combinations of attribute names and similarity metrics that are suitable for extracting and comparing blocking keys. Supervised learning has also been applied to generic ER in order to classify pairs of entities into matching and non-matching, by adapting similarity metrics and the corresponding thresholds to a particular domain [2, 6, 8, 24]. Other works introduce methods for facilitating the construction of the training set [23], while in [3], the authors propose supervised techniques for combining the decisions of multiple ER systems into an ensemble of higher performance. No prior work has applied supervised learning techniques to the task of meta-blocking.

3. PRELIMINARIES

In this section, we introduce the main concepts and notation used in the paper and we provide a brief overview of existing unsupervised meta-blocking techniques. Table 1 summarizes notation.

An entity profile \( p \) is a uniquely identified collection of information described in terms of name-value pairs. An entity collection \( E \) is a set of entity profiles. Two profiles \( p_i, p_j \in E \) are duplicates if \( p_i, p_j \) match or matches (\( p_i \equiv p_j \)) if they represent the same real-world object.

Entity Resolution comes in two forms. Clean-Clean ER receives as input two duplicate-free but overlapping entity collections and returns as output all pairs of duplicate profiles they contain. Dirty ER receives as input a single entity collection that contains duplicates in itself and returns the set of matching entity profiles. Blocking can be used to scale both forms of ER to large entity collections by clustering similar profiles into blocks so that comparisons are restricted among the entity profiles within each block \( b \).

The quality of a block collection \( B \) can be measured in terms of two competing criteria, namely precision and recall, which are estimated through the following established measures [1, 7, 18, 21]:

(i) Pairs Quality (\( PQ \)) assesses precision, i.e., the portion of non-redundant comparisons between matching entities. It is defined as:

\[
PQ(\mathcal{B}) = \frac{|\mathcal{D}(\mathcal{B})|}{|\mathcal{B}|}
\]

which represents the set of detectable matches, i.e., the pairs of duplicate profiles that co-occur in at least one block, and \(|\mathcal{D}(\mathcal{B})|\) represents the size of \(|\mathcal{B}|\) for its size. \(|\mathcal{B}|\) is called aggregate cardinality and denotes the total number of comparisons contained in \( B \):

\[
|B| = \sum_{i \in B} |b_i|
\]

where \(|b_i|\) is the cardinality of \( i \), i.e., the number of pair-wise comparisons it entails. \( PQ \) takes values in \([0, 1]\), with higher values indicating higher precision for \( B \), i.e., fewer superfluous and redundant comparisons.

(ii) Pair Completeness (\( PC \)) assesses recall, i.e., the portion of duplicates that share at least one block and, thus, can be detected. It is formally defined as:

\[
PC(\mathcal{B}) = \frac{|\mathcal{D}(\mathcal{E})|}{|\mathcal{E}|}
\]

which represents the set of duplicates contained in the input entity collection \( E \), and \(|\mathcal{D}(\mathcal{E})|\) stands for its size. \( PC \) values are in the interval \([0, 1]\), with higher values indicating higher recall for \( B \).

Note that we follow a known best practice [1, 4, 18, 25], examining the quality of a block collection independently of profile matching techniques. In particular, we assume an oracle that correctly decides whether two entity profiles match or not. Thus, \( \mathcal{D}(\mathcal{B}) \) is equivalent to the set of matching comparisons in \( B \): for each profile pair \((p_i, p_j) \in B\), it is a duplicate if and only if \( p_i \equiv p_j \).

Given a redundancy-positive block collection \( B \), the goal of meta-blocking is to restructure \( B \) into a new collection \( B' \) that achieves significantly higher precision, while maintaining the original recall (\( PQ(\mathcal{B}') \geq PQ(\mathcal{B}) \) and \( PC(\mathcal{B}') = PC(\mathcal{B}) \)).

Existing meta-blocking techniques rely their functionality on the weighted blocking graph (\( G_B \)), a data structure that models the block assignments in the block collection \( B \). As illustrated in Figure 2(a), \( G_B \) is formed by creating a node for every entity profile in \( B \) and an undirected edge for every non-redundant pair of co-occurring profiles. Formally, this structure is defined as follows:
an entity profile

Table 1: Summary of main notation.

**Definition 1** (Weighted Blocking Graph). Given a block collection $B$, its weighted blocking graph is a graph $G_B = (V_B, E_B, W_B)$, where $V_B$ is the set of its nodes such that $V_B \subseteq V_B \times V_B$, $E_B \subseteq V_B \times V_B$ is the set of undirected edges between all pairs of co-occurring entity profiles in $B$, and $W_B$ is the set of edge weights that takes values in the interval $[0, 1]$ such that $V_{E_B} \subseteq V_B \times V_B$.

As explained in Section 1, the blocking graph enhances precision by eliminating all redundant comparisons without any impact on recall, since it contains no parallel edges. Then, meta-blocking applies a pruning algorithm in order to discard part of the superfluous comparisons at a small cost in recall. These algorithms are distinguished into four categories, based on their functionality and the type of threshold they incorporate [22]:

- **Cardinality Edge Pruning (CEP)** sorts all edges in descending order of their weight and retains those in the top $K$ ranking positions. Therefore, $K$ constitutes a global cardinality threshold that is applied to the entire graph.
- **Cardinality Node Pruning (CNP)** does the same, but retains the top $k$ edges for each node. $k$ is also a global cardinality threshold, but is applied to the neighborhood of each node.
- **Weight Edge Pruning (WEP)** discards all edges of the blocking graph that have a weight lower than a global weight threshold (the average edge weight in our case).
- **Weight Node Pruning (WNP)** applies a local weight threshold to the neighborhood of each node, discarding those adjacent edges with a weight lower than it.

4. SUPERVISED META-BLOCKING

We consider that a comparison between profiles $p_i$ and $p_j$ can be captured by a feature vector $f_{ij} = [a_1(p_i, p_j), a_2(p_i, p_j), \ldots, a_n(p_i, p_j)]$, where $a_1, a_2, \ldots, a_n$ is a set of features, and $a_i(p_i, p_j) = 1 \ldots n$ is the value of feature $a_i$ for this pair. For instance, the number of common blocks the adjacent profiles share could be such a feature.

By replacing edge weights with feature vectors, we extend the weighted blocking graph $G_B$ into the generalized blocking graph $G_B$, formally defined as follows:

**Definition 2** (Generalized Blocking Graph). Given a block collection $B$, its generalized blocking graph is a graph $G_B = (V_B, E_B, F_B)$, where $V_B$ is the set of nodes such that $V_B \subseteq V_B \times V_B$, $E_B \subseteq V_B \times V_B$ is the set of undirected edges between all pairs of co-occurring entity profiles in $B$, and $F_B$ is the set of feature vectors that are assigned to every edge such that $V_{E_B} \subseteq V_B \times V_B$.

The elements of $F_B$ are fed to a classifier that labels all edges of the blocking graph as likely_match or unlikely_match, if they are highly likely to connect two matching or non-matching entity profiles, respectively. We measure the performance of this process using the following notation:

- $TP(E_B)$ denotes the true positive edges of $E_B$, which connect matching profiles and are correctly classified as likely_match.
- $FP(E_B)$ are the false positive edges of $E_B$, which are adjacent to non-matching profiles, but are classified as likely_match.
- $TN(E_B)$ are the true negative edges of $E_B$, which connect non-matching profiles and are correctly categorized as unlikely_match.
- $FN(E_B)$ are the false negative edges of $E_B$, which connect matching profiles, but are categorized as unlikely_match.

After classifying all edges, supervised meta-blocking derives the pruned blocking graph $G_{B_p}$ by discarding those edges labeled as unlikely_match (i.e., $TN(E_B)$ and $FN(E_B)$). The edges retained in $G_{B_p}$ belong to the sets $TP(E_B)$ and $FP(E_B)$:

$$TP(E_B) = TP(E_B) \cup FN(E_B) = E_B - (TN(E_B) \cup FN(E_B)).$$

The output of supervised meta-blocking is the block collection $B_i$ that is derived from $G_{B_p}$ by creating a block of minimum size for every retained edge $e_i \in E_{B_p}$. Thus, its $PC$ and $PQ$ can be expressed in terms of the edges in $E_{B_p}$ as follows:

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

$$PQ(B_i) = \frac{|D(B_i)|}{|D(B)|} = \frac{|TP(E_B)|}{|D(E)|} + |FP(E_B)|.$$

We now formally define the task of supervised meta-blocking as:

**Problem 2** (Supervised Meta-blocking). Given a redundancy-positive block collection $B$, its generalized blocking graph $G_B = (V_B, E_B, F_B)$, the classes $C = \{\text{likely\_match, unlikely\_match}\}$, and a training set $E_T = \{(e_1, e_2) \in E_B \mid C(e_1, e_2)\}$, the goal of supervised meta-blocking is to learn a classification model that minimizes the cardinality of the sets $FN(E_B)$ and $FP(E_B)$ so that the block collection $B_i$ resulting from the pruned graph $G_{B_p}$ achieves higher precision than $B$ (i.e., $PQ(B_i) \approx PQ(B)$), while maintaining the original recall (i.e., $PC(B_i) \approx PC(B)$).

4.1 Classification Algorithms

In principle, any algorithm for supervised learning can be used for edge classification in supervised meta-blocking. However, it should have a limited overhead for correctly categorizing most edges of the blocking graph. Further, it should be compatible with the pruning algorithm at hand. Supervised meta-blocking learns global pruning models that apply to the entire blocking graph and not to a specific neighborhood. Thus, it can be applied to CEP, CNP and WEP, substituting their thresholds with a classification model. In the first two cases, though, the output of the classification model should sort the edges of the blocking graph in ascending order of the likelihood that they belong to the class likely_match. Given that most classifiers simply produce a category label for every instance, this is only possible with probabilistic classifiers: they associate every instance with the probability that it belongs to every class, thus enabling their sorting. Note, though, that supervised meta-blocking is not compatible with WNP: applying a global threshold or classification model to WNP renders it equivalent to WEP.

Based on the above, we have selected four state-of-the-art approaches that are commonly used in classification tasks [26]: (i) Naive Bayes (NB), (ii) Bayesian Networks (BN), (iii) C4.5 decision trees, and (iv) Support Vector Machines (SVM). For their implementation, we used the open-source library WEKA, version 3.6. Unless stated otherwise, we employ their default configuration, as provided by WEKA.

These approaches encompass two probabilistic classification algorithms that are compatible with CEP and CNP, namely Naive Bayes and Bayesian Networks. In addition, they involve functionalities of diverse sophistication. On the one extreme, SVM involves complex statistical learning, while on the other extreme, Naïve Bayes relies on simple probabilistic learning. The latter actually operates as a benchmark for deciding whether the additional computational cost of the advanced classifiers pays off: comparable performance across all algorithms provides strong indication for the robustness of our classification features.
To solve the supervised meta-blocking problem, we need to determine the features to annotate the edges of the blocking graph (Section 6) and the appropriate training set, both in terms of size and composition (Section 7). In Section 5, we introduce the datasets and metrics to be used for the evaluation of the proposed solution.

5. DATASETS & METRICS

Datasets. We consider both Clean-Clean and Dirty ER and we employ the real-world datasets used in the earlier meta-blocking work [22]. Table 2 summarizes the characteristics of the entity collections and their blocks for each dataset.

D_movies is a collection of 50,000 entities shared among the individually clean sets of IMDB and DBPedia movies. The ground truth for this dataset stems from the “imdbid” attribute in the profiles of the DBPedia movies. Its blocks were created using Token Blocking (cf. Section 2) in conjunction with Block Purging, which discards blocks containing more comparisons than a dynamically determined threshold [21]. The resulting block collection exhibits nearly perfect recall at the cost of 27 million comparisons. Out of them, 22 million comparisons are non-redundant, forming the edges of the blocking graph.

Our second Clean-Clean ER dataset, D_infobox, consists of two different versions of the DBPedia Infobox dataset1. They contain all name-value pairs of the infoboxes in the articles of Wikipedia’s English version, extracted at October 2007 for DBPedia, and October 2009 for DBPedia2. The large time period that intervenes between the two collections renders their resolution challenging, since they share only 25% of all name-value pairs among the common entities [21]. As matching entities, we consider those with the same URL. For the creation of blocks, we applied Token Blocking and Block Purging. The resulting block collection entails 40 billion comparisons; 34 billion of them are non-redundant.

Finally, our Dirty ER dataset D_BTC09 comprises more than 250,000 entities from the Billion Triple Challenge 2009 (BTC09) data collection2. Its ground-truth consists of 10,653 pairs that were identified through their identical value for at least one inverse functional property. Its blocks correspond to a subset of those derived by applying the Total Description approach [20] (cf. Section 2) to the entire BTC09 data collection (see [22] for more details on how we selected this subset). They achieve very high PC at the cost of 130 million comparisons, out of which 78 million are non-redundant.

Metrics. To assess the impact of supervised meta-blocking on blocking effectiveness, we consider the relative reduction in PC, formally defined as: \( \Delta PC = \frac{PC(B_{\text{orig}}) - PC(B_{\text{res}})}{PC(B_{\text{orig}})} \cdot 100\% \), where \( PC(B) \) and \( PC(B_{\text{res}}) \) denote the recall of the original and the restructured block collection, respectively. Negative values indicate that metablocking reduces PC, while positive ones indicate higher recall.

To assess the impact of supervised meta-blocking on blocking efficiency, we use the following metrics:

- **Classification time CT** is the average time (in milliseconds) required by the learned model to categorize an individual edge of the blocking graph – excluding the time to build its feature vector.
- **Overhead time OT** is the total time required by meta-blocking to process the input blocks, i.e., to train the model, build the feature vectors of all edges, classify them and produce the new blocks.
- **Resolution time RT** is the sum of OT and the time required to execute all comparisons that are classified as likelymatch using an entity matching technique. As such, we employ the Jaccard similarity of all tokens in the values of two entity profiles – regardless of the associated attribute names.

\[ \Delta PC = \frac{PC(B_{\text{orig}}) - PC(B_{\text{res}})}{PC(B_{\text{orig}})} \cdot 100\% \]

<table>
<thead>
<tr>
<th>Feature</th>
<th>DBP</th>
<th>IMDB</th>
<th>DBP1</th>
<th>DBP2</th>
<th>BTC09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entities</td>
<td>27,615</td>
<td>23,182</td>
<td>1,19·10⁶</td>
<td>2,16·10⁶</td>
<td>253,353</td>
</tr>
<tr>
<td>Name-Value Pairs</td>
<td>186,013</td>
<td>816,012</td>
<td>1,75·10⁷</td>
<td>3,67·10⁷</td>
<td>1,60·10⁹</td>
</tr>
<tr>
<td>Existing Matches</td>
<td>22,405</td>
<td>892,586</td>
<td>10,653</td>
<td>10,653</td>
<td>3.33·10⁸</td>
</tr>
<tr>
<td>Blocks</td>
<td>40,430</td>
<td>1,21·10⁵</td>
<td>2,67·10⁷</td>
<td>3,98·10⁸</td>
<td>1,31·10⁹</td>
</tr>
<tr>
<td>PC</td>
<td>99,99%</td>
<td>99,89%</td>
<td>99,99%</td>
<td>99,99%</td>
<td>99,99%</td>
</tr>
<tr>
<td>Comparisons in Blocks</td>
<td>26 min</td>
<td>~320 hours</td>
<td>64 min</td>
<td>64 min</td>
<td>64 min</td>
</tr>
<tr>
<td>Brute-force RT</td>
<td>2.67·10⁷</td>
<td>3.98·10⁸</td>
<td>1.31·10⁹</td>
<td>1.31·10⁹</td>
<td>1.31·10⁹</td>
</tr>
<tr>
<td>Edges</td>
<td>2.52·10⁷</td>
<td>1.41·10⁹</td>
<td>2.77·10⁹</td>
<td>2.53·10⁹</td>
<td>2.53·10⁹</td>
</tr>
<tr>
<td>Nodes</td>
<td>5.06·10⁴</td>
<td>3.33·10⁸</td>
<td>4.77·10⁵</td>
<td>6.53·10⁵</td>
<td>6.53·10⁵</td>
</tr>
</tbody>
</table>

Table 2: Overview of the real-world datasets.

- **CMP** denotes the absolute number of comparisons contained in the restructured block collection (i.e., \(|B_{\text{res}}|\)).

For these metrics, the lower their value is, the higher is the efficiency of meta-blocking. Note that OT and RT do not consider the time to randomly select the training set, due to its negligible computational cost (see Section 7). We also estimate efficiency using PQ, with higher values indicating more efficient meta-blocking.

6. FEATURES FOR META-BLOCKING

Features for supervised meta-blocking describe the edges of the generalized blocking graph and should pertain to the corresponding comparisons or to the adjacent entities. These features should adhere to the following principles: (i) they should be generic, so that they are not tailored to a specific application; (ii) they should be effective, so that they yield high classification accuracy distinguishing between likely and unlikely matches; (iii) they should be efficient, involving low extraction cost and overhead, so that the classification time is significantly lower than the comparison time of its adjacent entities, and (iv) they should be minimal, in the sense that incorporating additional features has marginal benefit on the performance of meta-blocking. Similar principles were defined in [3] for classification tasks related to Entity Resolution (see Section 2).

**Feature Categorization.** To help understand candidate features for supervised meta-blocking and their appropriateness, we divide their space along five dimensions: schema-awareness, source of evidence, target, complexity and scope (see Figure 4).

**Schema awareness.** Classification features can be divided into schema-agnostic and schema-based ones.

**Schema-agnostic features rely on the structural information of the blocking graph and the characteristics of the blocks.**

**Schema-based features rely on the quality and the semantics of the attribute names that describe the input entity profiles.** Thus, they exclusively consider blocks and parts of the blocking graph that are associated with specific attributes.

**Source of evidence.** Given a block collection B, there are two main sources for extracting classification features: the blocks contained in B and the blocking graph G_B. Block-based features exclusively consider evidence of the former type, while graph-based ones rely on topological information about the blocking graph. Iterative features are graph-based features associated with a node that depend on the scores assigned to its neighboring nodes. Similarly to link analysis techniques, such as PageRank, these features may be computed by assigning a prior value to every node (or edge) and iteratively adjusting it, processing the entire blocking graph according to a mathematical formula until convergence.

**Target.** Depending on the part of the graph they annotate, classification features are divided into edge-specific, which pertain to individual edges and node-specific, which pertain to individual nodes.

**Complexity.** Classification features can be categorized into raw and derived ones. The raw attributes encompass atomic information that is explicitly available in the input block collection or its

---

blocking graph; on the other hand, the derived features combine multiple features into a single, composite feature.

Scope. Classification features are local when they consider information that is directly related to the annotated item (i.e., the edge or its adjacent nodes). Global features consider information from the entire blocking graph.

Most criteria (with the exception of complexity) define two complementary categories. Thus, features from both categories can be combined into hybrid ones, which may exhibit higher performance.

6.1 Candidate Features

We introduce our candidate features and explain their appropriateness for our meta-blocking problem (notation is in Table 1).

Common Blocks. A schema-agnostic, block-based feature is Common Blocks, i.e., the number of blocks shared by two profiles:

\[ \text{Common Blocks}(e_i, j) = |B_{i,j}|. \]

This feature captures the inherent trait of redundancy-positive block collections that the more blocks two entity profiles share, the more likely they are to match.

Based on Common Blocks, we could define schema-based features that take into account the subset of common blocks stemming from the values of specific attributes (e.g., Common BlocksSame for the attribute “Title”). However, such schema-based features are application-specific and have limited generality. In addition, they are crafted for homogeneous information spaces, like databases, and cannot handle heterogeneous ones, characterized by very diverse schemata (e.g., Web of Data) and constituting the most common source of redundancy-positive blocking [22]. Therefore, we focus hereafter on schema-agnostic classification features, which are completely decoupled from the nature and the semantics of the attributes describing the input entity collections.

Entity Blocks. Another block-based feature is Entity Blocks, which is inversely proportional to the number of blocks containing a specific entity/node:

\[ \text{Entity Blocks}(v_i) = \log \frac{|B_i|}{|B|}. \]

This feature is inspired from the inverse document frequency (IDF) that is commonly used in Information Retrieval (IR). The rationale behind it is that the higher its value is (i.e., lower \(|B_i|\)), the more likely \(p_i\) is to match with one of its co-occurring entity profiles. In contrast, a profile that is contained in an excessively high number of blocks is highly likely to contain noise. For instance, it could be the result of falsely merging several profiles that correspond to different real-world objects.

Node Degree. This is a graph-based feature and it is equal to the degree of node \(v_i \in V_e\):

\[ \text{Node Degree}(v_i) = |V_i|. \]

In essence, Node Degree is equivalent to the number of non-redundant comparisons involving the entity \(p_i\) that corresponds to the node \(v_i\). The intuition here is that the lower its degree is, the more likely \(p_i\) is to match with one of its co-occurring entity profiles.

Iterative Degree. This is an iterative, graph-based feature that is based on the following premise: for every node \(v_i\), the lower the degrees of its neighboring nodes are, the higher is the likelihood that \(p_i\) is matching with one of them and, thus, the higher the score of \(v_i\) should be. It is similar to Node Degree, as it initially associates every node with a prior score equal to the portion of non-redundant comparisons involving it (i.e., \(|V_i|/|B_i|\)). They differ though in that the Iterative Degree gradually modifies the score of a node \(v_i\), \(ID(v_i)\), according to the following formula:

\[ ID(v_i) = ID_0(v_i) + \sum_{v_j \in V_i} \frac{ID(v_j)}{|V_i|}. \]

where \(ID_0(v_i)\) is the prior score assigned to \(v_i\) and \(v_j \in V_i\) are the nodes connected with \(v_i\) on the blocking graph. This formula is similar to the one defining PageRank with priors, where the damping factor \(d\), which determines the behavior of the random surfer, is set equal to 0. It can be calculated using a simple iterative algorithm; after converging, nodes connected with many nodes of high degree receive the lowest scores, while the highest scores are assigned to nodes connected with few nodes of low degree.

The extraction of such iterative, graph-based features is computationally expensive and it does not scale well to blocking graphs with millions of nodes and billions of edges. Nevertheless, we include Iterative Degree in our approach and we investigate whether its low efficiency is counterbalanced by high discriminatory power.

Transitive Degree. A possible surrogate of higher efficiency is the Transitive Degree feature. It lies in the middle of Node Degree and Iterative Degree, considering the aggregate degrees of the nodes that lie within the neighborhood of \(v_i\) as follows:

\[ \text{Transitive Degree}(v_i) = \sum_{v_j \in V_i} |V_j|. \]

Common Neighbors. This graph-based feature amounts to the portion of adjacent entity profiles shared by a pair of co-occurring profiles. More formally, it is defined as follows:

\[ \text{Common Neighbors}(e_i, j) = \frac{|V_i \cap V_j|}{|V_i \cup V_j|}. \]

High values indicate that \(p_i\) and \(p_j\) co-occur with the same entities, either in their common blocks or in blocks they do not share. In the latter case, the common neighbors actually help deal with patterns missed due to noise in entity profiles. For example, consider the entity profiles \(p_1=\langle\text{name, John}, \langle\text{surname, Smith}\rangle\rangle\), \(p_2=\langle\text{name, Jon}, \langle\text{surname, Smith}\rangle\rangle\), \(p_3=\langle\text{name, Smith}, \langle\text{surname, Jon}\rangle\rangle\), where \((p_1 \neq p_2) \neq p_3\); Token Blocking [21] (cf. Section 2) yields two blocks \(B_{\text{same}}=\{p_1, p_3\}\) and \(B_{\text{sum}}=\{p_2, p_3\}\), with \(p_1\) and \(p_2\) co-occurring in none of them. Nevertheless, Common Neighbors provides strong evidence for their match.

Jaccard Similarity. This feature captures the portion of all comparisons (including the redundant ones) that involve a specific pair of entity profiles:

\[ \text{Jaccard Sim}(e_i, j) = \frac{|p_i \cap p_j|}{|p_i \cup p_j|}. \]

Higher values of this ratio indicate a stronger pattern of co-occurrence for \(p_i\) and \(p_j\) and, hence, the more likely \(p_i\) and \(p_j\) are to match. Note that on the target dimension, Entity Blocks, Node Degree, Iterative Degree and Transitive Degree are node-specific, while Common Blocks, Common Neighbors and Jaccard Sim are edge-specific features. Although the distinction between edge- and node-specific features seems trivial, there are two major qualitative differences between them. First, a feature vector has to include two values for every node-specific feature – one for each of the adjacent entities – thus broadening the search space by two dimensions. In contrast, edge-specific attributes are computed only once per feature vector, adding a single dimension to the search space. Second,
edge-specific features are expected to exhibit higher discriminatory power than the node-specific ones, because every value of the latter participates in as many feature vectors as the degree of the corresponding node; in contrast, every value of the edge-specific features pertains to a single feature vector.

(Reciprocal) Aggregate Cardinality of Common Blocks. From the aforementioned features, only Common Blocks and Node Degree are raw. In general, there is no rule-of-thumb for a-priori determining which form of features achieves the best performance in practice. Even different forms of derived features may lead to significant differences in classification accuracy. As an example, consider two edge-specific features that use the same information, but in different forms: the Aggregate Cardinality of Common Blocks (ACCB) and the Reciprocal Aggregate Cardinality of Common Blocks (RACCB) attributes. The former sums the cardinalities of the blocks shared by the adjacent entities (raw feature): \(\text{ACCB} = \sum_{b \in B} |b_i|\). The latter sums the inverse of the cardinalities of common blocks (derived feature):

\[
\text{RACCB} = \sum_{b \in B} \frac{1}{|b_i|}.
\]

Both features rely on the premise that the smaller the blocks two entities co-occur in, the more likely they are to be matching. Hence, the lower the value of ACCB is, the more likely the co-occurring entities match, and vice versa for RACCB. Preliminary experiments, though, demonstrated that ACCB achieves significantly lower classification accuracy than RACCB, due to its low discriminativeness: it assigns identical or similar values to pairs of co-occurring entities that share blocks of totally different cardinalities. For instance, consider two pairs of entities: the first co-occurs in 3 blocks containing 1, 2 and 4 comparisons, while the second shares 2 blocks with 2 and 5 comparisons; ACCB takes the same value for both edges (7), while RACCB amounts to 1.75 and 0.70 for the first and the second pair, respectively, favoring the entities that are more likely to match.

Co-occurrence Frequency-Inverse Block Frequency. Derived features can come in more advanced forms than RACCB, combining multiple features through linear or non-linear functions. However, they should be used with caution for several reasons: (i) they involve a higher extraction cost than the comprising raw features; (ii) their form might be too complex to be interpretable; (iii) they are usually correlated with the raw features they comprise and, thus, are incompatible with them, when applied to classifiers with strong independence assumptions (e.g., Naïve Bayes); (iv) some classification algorithms may operate better with raw features, learning themselves the linear or non-linear associations between the input features. For these reasons, the derived features should involve a low extraction cost and transform as few raw features as possible.

Here, we combine Common Blocks and Entity Blocks into a feature inspired from the TF-IDF measure employed in IR. We call it Co-occurrence Frequency-Inverse Block Frequency (CF_IFB) and formally define it as:

\[
\text{CF_IFB} = |B_i| \cdot \log \left( \frac{|B|}{|B_i|} \right) \cdot \log \left( \frac{|B|}{|B|} \right).
\]

Experiments showed that this form outperforms the individual features, because Entity Blocks is of limited usefulness when used independently, but it is valuable for extending Common Blocks, which otherwise suffers from low discriminativeness (it amounts to 1 or 2 for the vast majority of edges). In this way, we also restrict the dimensionality of our approach by two degrees.

Table 3: Top 10 feature sets selected.

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS1</td>
<td>CF_IFB, RACCB, Node Degree</td>
</tr>
<tr>
<td>FS2</td>
<td>CF_IFB, RACCB, Transitive Degree</td>
</tr>
<tr>
<td>FS3</td>
<td>CF_IFB, RACCB, Node Degree, Transitive Degree</td>
</tr>
<tr>
<td>FS4</td>
<td>CF_IFB, RACCB, Jaccard Sim, Transitive Degree</td>
</tr>
<tr>
<td>FS5</td>
<td>CF_IFB, RACCB, Jaccard Sim, Node Degree</td>
</tr>
<tr>
<td>FS6</td>
<td>CF_IFB, RACCB, Transitive Degree, Iterative Degree</td>
</tr>
<tr>
<td>FS7</td>
<td>CF_IFB, RACCB, Node Degree, Iterative Degree</td>
</tr>
<tr>
<td>FS8</td>
<td>CF_IFB, RACCB, Jaccard Sim, Node Degree, Transitive Degree</td>
</tr>
<tr>
<td>FS9</td>
<td>CF_IFB, RACCB, Jaccard Sim, Node Degree, Iterative Degree</td>
</tr>
<tr>
<td>FS10</td>
<td>CF_IFB, RACCB, Jaccard Sim, Transitive Degree, Iterative Degree</td>
</tr>
</tbody>
</table>

6.2 Feature Selection

To satisfy the minimalism principle, we examine the relative performance of each combination of features, called feature set (FS), in order to identify the one achieving the best balance between effectiveness and efficiency. There is a clear trade-off here: fewer features mean less complex and less time-consuming learned model (higher efficiency), but lower effectiveness.

The selected features yield 63 combinations. Due to their high number, our feature selection process has two phases. First, we extracted the top 10 performing feature sets automatically. Then, we selected the best set by examining their relative performance analytically. We use all four classification algorithms over \(D_{max}\). The models were trained using 1,000 labeled edges, equally partitioned between matching and non-matching edges, that were randomly se-
selected from the entire blocking graph. The remaining edges formed the disjoint testing set. To derive accurate performance estimations, we repeated this process 10 times and considered the average value of each metric.

To identify the top 10 performing feature sets, we sort the feature sets in descending order of their total F-measure and select those placed in the top 10 positions. The Total F-measure (TF) of a feature set FS is the sum of the F-measures corresponding to each classification algorithm: \( TF(FS) = \sum_{i}^{N}(\text{RBF}, \text{SVM}, \text{BNN}) F_i(FS) \), where the F-measure for a feature set FS, and an algorithm \( j \) is computed as: \( F_j(FS) = 2 \cdot PC \cdot nPQ / (PC + nPQ) \), with \( nPQ \) denoting the normalized Pairs Quality across all feature sets for the algorithm \( j \) (i.e., \( nPQ(FS) = \max_{i}(PQ_i(FS)) \)).

The resulting top 10 feature sets are shown in Table 3. Collectively, they involve all features of our approach, a strong indication for the high utility of each feature. Moreover, each feature set comprises at least 3 features, revealing that our features are compatible and complementary, working best when used in conjunction.

To select the best feature set out of the top 10 performing ones, we evaluate their effectiveness through \( \Delta PC \) and their efficiency through \( CMP \) and \( CT \) (see Section 5). Figures 6(a), (b) and (c) present \( \Delta PC, CMP \) and \( CT \), respectively. The horizontal axes correspond to the feature sets. We can identify the optimal feature set by examining their relative performance across the three figures. The closer a feature set is placed to the horizontal axis of each figure across all classification algorithms, the better is its performance.

Figure 6(a) shows that most feature sets exhibit limited variation in \( \Delta PC \) (between -3.5% and -5% for most algorithms). Only Naïve Bayes is rather unstable, yielding five outliers: FS1, FS2, FS3, FS6 and FS7 have an unacceptable impact on recall (over -7%) and hence they are not considered any further.

For the remaining sets, Figures 6(a) and 6(b) reveal a trade-off between \( \Delta PC \) and \( CMP \); the higher \( \Delta PC \) for a particular feature set and classification algorithm, the lower \( CMP \) gets, and vice versa. Hence, none of the feature sets excels in both metrics. To identify the set with the best balance between \( \Delta PC \) and \( CMP \), we consider their average values across all classifiers. Only FS4 and FS5 achieve the most stable performance: FS4 requires \( 5.66 \pm 1.25 \times 10^3 \) comparisons, and FS5 gives \( 5.65 \pm 1.77 \times 10^3 \). They are also the most efficient compared to FS8, FS9 and FS10, which require 12.5% more comparisons, on average.

Finally, to decide between FS4 and FS5, we compare them in terms of \( CT \). Figure 6(c) shows negligible differences (in absolute numbers) between them – in the order of 1/100 of a millisecond. Given, though, that \( CT \) concerns a single edge, these differences become significant when meta-blocking is applied to large blocking graphs with millions of edges. We choose FS5 because it learns faster classification models than those of FS4 by 3%, on average.

In the following, we exclusively consider the feature set FS5, comprising \( \text{CF}_{-1BF}, \text{RACC}, \text{Jaccard} \_\text{Sim} \) and \( \text{Node Degree} \), which combines a low extraction cost with high discriminatory power.

7. TRAINING SET

The quality of the learned classification model also depends on the training set and in particular on its composition and size.

The definition of supervised meta-blocking (Problem 2) makes no assumptions about the training set. However, the vast majority of the edges in the blocking graph connect non-matching entities and thus correspond to superfluous comparisons. If the training set involves the same class distributions as the set of edges, \( E_B \), it will be heavily imbalanced in favor of the \( \text{unlikely} \_\text{match} \) class. As a result, the classifier would be biased towards assigning every instance to the majority class, pruning most of the edges.

This situation constitutes a class imbalance problem, which is typical in supervised learning (as an example, consider the task of spam filtering, where the vast majority of e-mails is not spam) with several solutions [15]: oversampling randomly replicates instances of the minority class until the class distribution is balanced, cost-sensitive learning incorporates high misclassification cost for the minority class when training the classifier, and ensemble learning trains a set of classifiers that collectively take classification decisions through a form of weighted voting. Unfortunately, cost-sensitive and ensemble learning increase the complexity of the classification model, while oversampling yields excessively large training sets prone to overfitting (too many repetitions of the same instances). Instead, we use undersampling, which randomly selects a subset of the same cardinality from both classes. The training set is equally partitioned between \( \text{likely} \_\text{match} \) and \( \text{unlikely} \_\text{match} \) edges. This approach is best for small training sets, which can be manually labeled in the absence of ground truth.

The size of the training set, called sample size, affects both the effectiveness and the efficiency of supervised meta-blocking: the smaller the sample size is, the lower is the complexity of the learned model and the more efficient is its training and its use. However, this comes at the cost of lower classification accuracy, as the simpler the learned model is, the more likely it is to miss patterns. To identify the break-even point in this trade-off, we experimentally perform a sensitivity analysis for the sample size with respect to effectiveness (\( \Delta PC \)) and efficiency (\( CT \) and \( CMP \)).

Training Set Selection. We apply the selected feature set to the four classifiers over \( D_{match} \) using training sets of various sizes. Due to undersampling, these training sets were specified in terms of the minority class cardinality (i.e., the number of matching entities in

Figure 6: Performance of the feature sets in Table 3 over \( D_{match} \) with respect to (a) the relative reduction in recall (\( \Delta PC \)), (b) the absolute number of retained comparisons (\( CMP \)) and (c) the classification time of an individual edge (\( CT \)).

Note that the F-measure for blocking-based ER is defined as \( F = \frac{2 \cdot PC \cdot PQ}{PC + PQ} \) [4]. We employ \( nPQ \) instead of \( PQ \), because the latter takes very low values for redundancy-positive block collections. In fact, \( PQ \) is lower than \( PC \) by one or two orders of magnitude, thus dominating \( F_1 \), which ends up assigning high scores to feature sets with a few comparisons and a few detected duplicates.
the ground truth). Each one was equally partitioned between (randomly selected) matching and non-matching edges, whose number ranged from 0.5% to 10% of the minority class size, with a step of 0.5%. For every sample size, we repeated the process 10 times and considered the average of the aforementioned metrics. Figures 7(a), (b) and (c) depict the learning curves with respect to ΔPC, CMP and CT, respectively.

Figure 7(a) shows that most classifiers exhibit rather stable recall, with a variance at most 1.2%. In all cases, the variance in ΔPC gets lower as the sample size increases. Especially for sample sizes that exceed 5% of the minority class cardinality (i.e., around 1,100 labeled instances per class), there is no variance in practice. These patterns demonstrate that our proposed feature set is comprehensive, robust enough and effective even when trained over small training sets – regardless of the classification algorithm.

Figure 7(b) shows the evolution of CMP with the increase in sample size. Most classifiers start from high values, but gradually converge to lower values as the sample size increases. Similar to ΔPC, the variance in CMP decreases in proportion to the sample size and becomes negligible for sample sizes larger than 5% of the minority class. Regarding CT, Figure 7(c) shows that all classifiers exhibit a relatively stable, good performance regardless of the sample size. The average CT is close to the time observed for the sample size equal to 5%.

Consequently, a sample size equal to 5% of the minority class achieves a performance equivalent to that of much larger ones. In the following, we exclusively consider training sets comprising 5% of the edges labeled as likely.match and an equal number of edges labeled as unlikely.match.

8. CLASSIFIERS CONFIGURATION

The performance of the selected classification algorithms depends on their internal parameter configuration. Fine-tuning may significantly enhance classification accuracy, but it may also lead to over-fitting, which increases the complexity of the learned model and inflates the overhead of meta-blocking. To assess how their configuration affects the performance of our approach, we perform analytical fine-tuning of their parameters. For each algorithm, we examine two parameters that are fine-tuned in parallel:

- **C4.5**: we study the maximum number of instances per leaf node, ranging from 2 to 5, and the confidence interval, ranging from 0.1 and 0.5 with a step of 0.05 (36 configurations in total). By default, Weka sets the former parameter to 2 and the latter to 0.25.
- **SVM**: we consider two kernels, the linear and the RBF, and we vary the complexity parameter C from 1 to 10 with a step of 1 (20 configurations in total). Weka’s default configuration incorporates a linear kernel function with the complexity constant C set to 1.
- **Bayesian Networks**: we use three search algorithms: simulated annealing, hill climbing and genetic search. We use each one with global and with local scope (6 configurations in total). The default configuration of Weka uses the hill climbing search of local scope.

- **Naïve Bayes**: two boolean parameters that determine the processing of numeric attributes, i.e., use of supervised discretization and of kernel density estimator (4 configurations in total). By default, Weka sets both parameters to false.

For each classifier, we apply every possible configuration to 10 randomly selected training sets from the blocking graph of D_movies using the sample size and the features determined above. Due to the large number of configurations, we consider only the default, the optimal and the average performance for each classification algorithm and metric. As optimal, we define the configuration with the largest F-measure (again, F1 employs nPQ instead of PQ). The following configurations were selected in this way: the use of both supervised discretization and kernel estimator for Naïve Bayes; 5 instances per leaf and confidence interval equal to 0.1 for C4.5; linear kernel with C=9 for SVM; simulated annealing with global scope for Bayesian Networks. Figures 7(a) to (c) depict the experimental outcomes with respect to ΔPC, CMP and CT, respectively.

We first investigate the relative performance of the default and the optimal configuration. For Naïve Bayes, the optimal one puts more emphasis on effectiveness, increasing ΔPC by executing more comparisons. However, its overall efficiency is significantly increased, as its overhead (CT) is reduced to 1/5. For C4.5 and SVM, the optimal configurations decrease CMP by 5%, while exhibiting practically identical ΔPC and CT with the default ones. On the other hand, the optimal configuration for the Bayesian Networks reduces CMP by 20% for almost the same PC as the default one, but puts a toll on efficiency: CT increases by 25%. Hence, we choose the default configuration for Bayesian Networks, while for the other algorithms we choose the optimal ones, due to their slightly better balance between effectiveness and efficiency.

We now examine the robustness of the classification algorithms with respect to their configuration based on the distance between the default, the optimal and the average performance for all configurations. We observe that C4.5 is practically insensitive to fine-tuning, despite the large numbers of configurations considered. The same applies to SVM with respect to ΔPC and CMP; its average CT (2.66 msec), though, is twice orders of magnitude higher than the default and the optimal one. This is because the RBF kernels are 10 times slower when classifying an individual edge than the linear ones, which exhibit a rather stable CT. A similar situation appears in the case of Naïve Bayes, where the average CT amounts to 3.80 msec, due to the inefficiency of a single configuration: supervised discretization for numeric attributes without the kernel estimator. The other two metrics, though, advocate that Naïve Bayes is rather sensitive to its configuration. Finally, the Bayesian Networks exhibit significant variance with respect to CMP and CT, but the overall efficiency is relatively stable across all configurations. We can conclude that for the selected feature set and sample size, most classifiers are rather robust with respect to their configuration.
9. EXPERIMENTAL EVALUATION

We now compare the performance of our supervised meta-blocking techniques with the best performing unsupervised ones over three pruning algorithms: WEP, CEP and CNP. Remember that WEP is compatible with all classification algorithms, while CEP and CNP are only compatible with Naïve Bayes and Bayesian Networks.

To compare the supervised and the unsupervised techniques on an equal basis, we adapted the latter so that they exclude the edges used for training by the former. Hence, we applied them 10 times to each dataset and derived their performance from the average of the relevant metrics (this explains why in some cases their performance is slightly different from that reported in [22]).

We also employ the state-of-the-art approach of Iterative Blocking [25] as an additional baseline method. Given that its performance depends heavily on the processing order of blocks, we apply it to 10 random shuffles of each dataset’s blocks and present the average value of each metric. Note that for Clean-Clean ER, we consider the best possible performance of iterative blocking, assuming that all pairs of detected matches are propagated to the subsequently processed blocks so that their entities do not participate in any other comparison.

We implemented our approaches in Java 1.7 and tested them on a server with Intel i7-4930K 3.40GHz and 32GB RAM, running Debian 7. Graphs were implemented using the JUNG framework.

9.1 In-depth Analysis over Real Datasets

Table 4 presents the performance of the baseline 6 and our supervised techniques with respect to the three pruning algorithms over the datasets of Table 2. For each dataset, we considered the unsupervised meta-blocking in conjunction with the weighting scheme that yields the best trade-off between PC and PQ; for D_movies and D_movies_foboxes, we used the weighting schemes CBS, EJS and ECBS for WEP, CEP and CNP, respectively, while for D_BTC90, we employed the ARCS scheme in all cases (see [22] for more details).

We now examine each pruning algorithm separately.

WEP. We observe that supervised meta-blocking consistently achieves a better balance between effectiveness and efficiency over D_movies and D_movies_foboxes. It executes almost an order of magnitude fewer comparisons than the unsupervised method with a minor increase of PC. As a result, precision consistently increases by at least 4 times. The higher overhead (OT) is counterbalanced by the considerably lower number of comparisons, resulting overall in significantly improved resolution times (RT).

For D_BTC90, supervised meta-blocking improves efficiency to a similar extent at the cost of slightly lower recall (the only exception is SVM, whose PC is significantly lower than the unsupervised method by 7.5%). Both the number of comparisons and the overhead time are almost half, leading to significantly better RT.

CEP. For D_movies and D_movies_foboxes, supervised meta-blocking achieves significantly higher recall, increased by more than 10%. Its overhead time, though, is more than twice that of unsupervised meta-blocking. Given that both approaches execute the same number of comparisons, the classification models exhibit notably increased resolution time. In the case of D_BTC90, supervised meta-blocking decreases PC and PQ to a minor extent, while increasing the resolution time by 25%. For each dataset, these patterns are consistent across both probabilistic models.

CNP. For D_movies and D_BTC90, supervised meta-blocking reduces the number of executed comparisons to a significant extent, at the cost of a lower PC. PQ almost doubles, but the higher overhead than unsupervised meta-blocking leads to an increased resolution time. The same applies to D_movies_foboxes, as both OT and RT are significantly higher than unsupervised meta-blocking. In this case, though, the number of comparisons is practically the same, while PQ gets slightly higher, because PC slightly increases.

Iterative Blocking. We observe that iterative blocking achieves the lowest overhead time and the highest recall across all datasets: for the Clean-Clean ER datasets D_movies and D_BTC90, PC is equal to that of the input block collection, while for the Dirty ER dataset (D_BTC90), it increases by 1%. However, this comes at the cost of rather low efficiency; iterative blocking actually executes so many comparisons that its resolution time is practically identical with the brute-force approach of performing all comparisons in the input block collection. For Clean-Clean ER, its run-time lies in the middle between supervised and unsupervised meta-blocking, due to the ideal settings we consider (i.e., none of the matched entities participates in any comparison after their detection). In a more realistic scenario, though, its efficiency is expected to be lower than that of unsupervised meta-blocking. We can conclude, therefore, that Iterative Blocking is only appropriate for applications that place recall in priority and are satisfied with rather conservative savings in efficiency. For the rest of them, supervised meta-blocking offers a better balance between effectiveness and efficiency.

Conclusions. For WEP, our techniques leverage small training samples and feature vectors to significantly increase efficiency at a negligible cost in effectiveness (if any). This consistent behavior is important, since WEP is compatible with practically any blocking-based ER application. It stems from the low computational cost and the comprehensiveness of our features. The latter aspect can be inferred from the performance of Naïve Bayes, which is directly comparable with the more complicated algorithms in all cases. The best performance, though, is achieved when combining supervised
meta-blocking with C4.5, which reduces the resolution time by 50% across all datasets for practically the same effectiveness.

With respect to CEP, which is only suitable for incremental ER, unsupervised meta-blocking exhibits significantly higher efficiency, due to its lower overhead. However, the high OT time of the classification models is rendered insignificant, when advanced, time-consuming entity matching methods are used. Then, supervised meta-blocking should be preferred due to its consistently higher recall. For the same reason, it should be used with all applications of incremental ER that place more emphasis on efficiency. For CNP, we cannot draw any safe conclusions, due to the unstable performance of supervised meta-blocking across the 3 datasets, caused by the incompatibility of its global training information with the local scope of this pruning algorithm. Finally, it is worth stressing that supervised meta-blocking consistently improves the run-time of the brute-force approach by at least 10 times (cf. Table 2).

### 9.2 Recall and Run-time Scalability

We now examine the scalability of our supervised techniques in relation to the three pruning algorithms. We apply them to seven synthetic datasets that were created by FEBRL [5] and have been widely used in the literature for this purpose [4, 13]. They pertain to Dirty ER and their sizes range from 10 thousand to 2 million entities. To derive redundancy-positive blocks, we applied Token Blocking and Block Purging to each dataset. The technical characteristics of the resulting block collections are presented in Table 5.

As baseline methods, we employ iterative blocking and unsupervised meta-blocking. The latter was combined with the ECBS weighting scheme across all pruning algorithms and datasets, as it consistently exhibited the best performance.

We evaluate the performance of all methods using two metrics: APC assesses the impact on effectiveness, while Relative Resolution Time (RRT) assesses the impact on efficiency. In essence, it expresses the portion of the input blocks’ resolution time that is required by the meta-blocking method. Formally, it is defined as:

\[
\text{RRT} = \frac{RT(B\prime)}{RT(B)} \cdot 100\% , \quad \text{where } RT(B) \text{ and } RT(B') \text{ are the resolution times of the original and the restructured block collections; the lower its value is, the more efficient is the meta-blocking method.}
\]

We applied supervised meta-blocking to WEP, CEP and CNP. The outcomes with respect to APC are depicted in Figures 9(a)-(c), while RRT is presented in Figures 9(d)-(f).

**Table 5: Overview of the synthetic datasets.**

<table>
<thead>
<tr>
<th>Entity Collections</th>
<th>Blocks Collections</th>
<th>Block Compar.</th>
<th>PC</th>
<th>Brute-force RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP</td>
<td>PQ</td>
<td>PC</td>
<td>ΔPC</td>
<td>OT</td>
</tr>
<tr>
<td>D_{init}</td>
<td>10,000</td>
<td>8,615</td>
<td>11,088</td>
<td>3.35×10^4</td>
</tr>
<tr>
<td>D_{init}</td>
<td>50,000</td>
<td>42,668</td>
<td>40,569</td>
<td>7.42×10^4</td>
</tr>
<tr>
<td>D_{init}</td>
<td>100,000</td>
<td>84,894</td>
<td>72,733</td>
<td>2.91×10^4</td>
</tr>
<tr>
<td>D_{init}</td>
<td>200,000</td>
<td>123,648</td>
<td>123,789</td>
<td>1.19×10^5</td>
</tr>
<tr>
<td>D_{init}</td>
<td>300,000</td>
<td>254,686</td>
<td>166,099</td>
<td>2.70×10^5</td>
</tr>
<tr>
<td>D_{init}</td>
<td>1,000,000</td>
<td>849,276</td>
<td>441,999</td>
<td>2.94×10^5</td>
</tr>
<tr>
<td>D_{init}</td>
<td>2,000,000</td>
<td>1,699,430</td>
<td>863,528</td>
<td>1.17×10^6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entity Collections</th>
<th>Blocks Collections</th>
<th>Block Compar.</th>
<th>PC</th>
<th>Brute-force RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4.5</td>
<td>5.69</td>
<td>3.56</td>
<td>95.35</td>
<td>-4.07</td>
</tr>
<tr>
<td>SVM</td>
<td>5.69</td>
<td>4.40</td>
<td>94.87</td>
<td>-5.45</td>
</tr>
<tr>
<td>Bayesian Networks</td>
<td>6.51</td>
<td>3.13</td>
<td>95.74</td>
<td>-3.67</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>6.50</td>
<td>3.14</td>
<td>95.74</td>
<td>-3.67</td>
</tr>
<tr>
<td>Bayesian Networks</td>
<td>5.69</td>
<td>3.56</td>
<td>95.35</td>
<td>-4.07</td>
</tr>
</tbody>
</table>

**Table 4: Performance of supervised meta-blocking and the baselines over all datasets with respect to (a) WEP, (b) CEP, (c) CNP.**
2/3, but achieves significantly lower recall. Compared to the brute-force approach, supervised meta-blocking improves the run-time by at least 5 times, as its RRT lies consistently lower than 20%.

10. CONCLUSIONS

In this work, we demonstrated how supervised meta-blocking can be used to enhance the performance of existing, unsupervised meta-blocking methods. For this task, we proposed a small set of generic features that combine a low extraction cost with high discriminatory power. We showed that supervised meta-blocking can achieve high performance with small training sets that can be manually created, and we verified that most configurations of established classification algorithms have little impact on the overall performance. We analytically compared our supervised approaches with baseline and competitor methods.

In the future, we will apply transfer learning techniques to supervised meta-blocking, so that a classification model trained over a labeled set maintains its high performance over another, unlabeled one. In addition, we will explore the use of active learning and crowdsourcing techniques in the creation of training sets.

Acknowledgements. This research has been co-financed by the EU (European Social Fund - ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

References


Figure 9: Scalability analysis over the synthetic datasets with respect to (a)-(d) WEP, (b)-(e) CEP and (c)-(f) CNP.
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 [13], 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 [16]. 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 $p_3$ 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 [20]. 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 [15], Suffix Array [1], [8], HARRA [18] and schema-agnostic blocking [27], [29]. To illustrate their functionality, consider Figure 1(b), which depicts the blocks that are produced after applying the

\(^1\) http://www.google.com/base
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 [22]: 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 [17]). 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 [25]).

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 [33] and the Sorted Blocks approach [11]. In contrast, the redundancy-positive techniques have been shown to apply to HHIS and scale to millions of entity profiles [18], [27]. [29]. 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 [27] 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 [27], 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 \( dh_{\text{max}} \). 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 \( dh_{\text{max}} = 2 \), 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 predefined 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 \( \langle \text{id}, A_p \rangle \), where \( \text{id} \in \mathcal{ID} \) is a unique identifier, and \( A_p \) is a set of name-value pairs \( (n, v) \).

Naturally, the value \( v \) in a name-value pair \( (n, v) \) 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 Dataspaces applications [20]. 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, \( \mathcal{D}^{\mathcal{E}_1, \mathcal{E}_2} \), 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 \( b_i^1 \) (\( \subseteq E_1 \)) and \( b_i^2 \) (\( \subseteq E_2 \)).

**Improving Blocking through Meta-blocking.** The quality of a block collection \( B \) is measured in terms of two competing criteria: efficiency and effectiveness. The former is directly related to its aggregate cardinality (\(|B|\)), i.e., the total number of comparisons it contains:

\[
|B| = \sum_{b \in B} |b|, \text{ where } |b| \text{ is the individual cardinality of } b_i \text{ (i.e., total number of comparisons entailed in block } b_i); \text{ we have } |b_i| = |b_i^1| + |b_i^2| \text{ for a unilateral block } b_i \text{ and } |b_i| = |b_i^1| + |b_i^2| \text{ for a bilateral block. The effectiveness of } B \text{ depends on the cardinality of the set } D^B \text{ 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 \( B \): the more comparisons are executed (i.e., higher \(|B|\)), the higher its effectiveness gets (i.e., higher \(|D^B|\)), 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], [8], [23], [27].

(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: \( PC(B) = \frac{|D^B|}{|B|} \), where \(|D^B|\) is the number of duplicates in the input entity collection \( E \). PC takes values in the interval \([0, 1]\), with higher values indicating higher effectiveness for \( B \).

(ii) Pairs Quality (PQ) estimates the portion of non-redundant comparisons that involve matching entities. Formally, it is defined as: \( PQ(B) = \frac{|D^B|}{|B|} \). It takes values in \([0, 1]\), with higher values indicating higher efficiency for \( B \) (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 \( B_{bs} \). It is defined as: \( RR(B, B_{bs}) = 1 - \frac{|B|}{|B_{bs}|} \) and takes values in the interval \([0, 1]\) for \(|B| \leq |B_{bs}|\), with higher values denoting higher efficiency for \( B \).

Meta-blocking aims at restructuring a block collection \( B \) 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 \( B' \) 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 \( B \), restructure it into a new one \( B' \) that achieves significantly higher levels of efficiency (i.e., \( PQ(B') \gg PQ(B) \) and \( RR(B', B) \gg 0 \)), while maintaining the original effectiveness (i.e., \( PC(B') \geq PC(B) \)).

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., \( PC(B') > PC(B) \)). However, this cannot 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.

**Metric Space for Blocking Techniques.** The goal of meta-blocking is to improve the balance between effectiveness and efficiency that a block collection \( B \) achieves. However, the impact on PC and RR can only be computed after examining analytically all blocks in \( B \) and \( B' \). 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 [29] 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 \( B \) as the average number of block assignments per entity of the input collection(s): \( BC = \sum_{b \in B} |b|/|E| \), where \(|b|\) denotes size (i.e., the number of entities) of block \( b \). BC takes values in the interval \([0, 1]\) for \(|B| \leq |E|\), with values denoting higher efficiency for \( B \).

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: \( CC = \sum_{b \in B} |b|/|B| \). CC takes values in the interval \([0, 2]|E|\) for Clean-Clean ER and in \([0, |E| - 1]\) 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 \( B \).

The BC-CC mapping of a block collection can be efficiently computed in linear time (i.e., \( O(|B|) \)) 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) [29]. 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 [29]. 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 B to 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 B, the corresponding blocking graph G_B models the block assignments in B: as shown in Figure 1(c), every entity contained in 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 B, the undirected blocking graph derived from it is a graph G_B = (V_B, E_B, WS), where V_B is the set of its nodes, E_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_B contains all entities of E that are present at least once. While E_B contains undirected edges between all pairs of co-occurring entities (i.e., ∀e_{i,j} ∈ E^E : pi ∈ E ∧ pj ∈ E ∧ pi ≠ pj), while E_B contains undirected edges between all pairs of co-occurring entities (i.e., ∀e_{i,j} ∈ E^E : pi ≠ pj ∧ ∀e_{k,l} ∈ E^E ∧ pi = e_k ∧ pj = e_l).

The blocking graph over a set of bilateral blocks B_{E1}×E2 is defined analogously. The only difference is that it results in a bipartite graph, since its set of nodes V_B is separated into two disjoint sets, V^1_B and V^2_B, which comprise entities stemming from the entity collections E_1 and E_2, respectively (i.e., V^1_B ⊆ E_1 and V^2_B ⊆ E_2). More formally, ∀e^1_{i} ∈ V^1_B : ∃e^2_{j} ∈ E_2 ∧ b^1_{i} ∈ E^E ∧ b^2_{j} ∈ E^E ∧ pi = e^1_{i} ∧ pj = e^2_{j}, where k ∈ {1,2}. Consequently, each pair of co-occurring 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 ↔ p_i ≠ p_j and e_{i,j}.weight = 1 ↔ p_i ≡ 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:
Graph Building receives a block collection $B$ and derives the blocking graph $G_B$ from its block assignments. We elaborate on this process in Section 3.1.

(ii) Edge Weighting takes as input a blocking graph $G_B$ and turns it into the weighted blocking graph $(G_B^*)$ 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 $(G_B^p)$ 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 $G_B^p$ and extracts from it a new block collection $B'$, 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 scheme-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 $B$ is outlined in Algorithm 1 (for unilateral blocks, the corresponding algorithm is simpler, and we omit it for brevity). Essentially, for each block in $B$, 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 $B$ (i.e., $O(|B|)$).

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.

```plaintext
Input: (i) $B$ a block collection, (ii) WS a weighting scheme
Output: $G_B$ the corresponding blocking graph
1 $V_B \leftarrow \emptyset$, $E_B \leftarrow \emptyset$; //initially empty graph
2 foreach $b_i \in B$ do // check all blocks
3     foreach $p_i \in b_1^i$ do // check all comparisons
4         $V_B \leftarrow V_B \cup \{v_i\}$; 
5         foreach $p_j \in b_2^j$ do
6             $E_B \leftarrow E_B \cup \{v_i, v_j\}$; //add node for $p_j$
7             $E_B \leftarrow E_B \cup \{v_i, v_j\}$; //add edge $<p_i, p_j>$
8 setEdgeWeights(WS, $B$, $V_B$, $E_B$);
9 normalizeEdgeWeight($E_B$);
10 return $G_B = (V_B, E_B, WS)$;
```

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 [21]. 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: $B_i \subseteq B$ denotes the set of blocks containing the entity $p_i$, $B_i \subseteq B$ is the set of blocks shared by the entities $p_i$ and $p_j$ (i.e., $B_{i,j} = B_i \cap B_j$), 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:

$$e_{i,j} = \sum_{b_i \in B_{i,j}} \frac{1}{|b_i|}.$$  

(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:

$$e_{i,j} = |B_{i,j}|.$$
\( e_{i,j,\text{weight}} = |B_{i,j}|. \)

(iii) \textbf{Enhanced Common Blocks Scheme (ECBS)}: This scheme improves on \textit{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:

\[
 e_{i,j,\text{weight}} = |B_{i,j}| \cdot \log \frac{|B|}{|B_i|} - \log \frac{|B|}{|B_j|.}
\]

(iv) \textbf{Jaccard Scheme (JS)}: Similar to \textit{ECBS}, this scheme aims at enhancing \textit{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 \( p_j \):

\[
 e_{i,j,\text{weight}} = \frac{|B_{i,j}|}{|B_i| + |B_j| - |B_{i,j}|}.
\]

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) \textbf{Enhanced Jaccard Scheme (EJS)}: This scheme improves on \textit{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:

\[
 e_{i,j,\text{weight}} = \frac{|B_{i,j}|}{|B_i| + |B_j| - |B_{i,j}|} \cdot \log \frac{|E_i|}{|V_i|} \cdot \log \frac{|E_j|}{|V_j|}.
\]

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 \textit{pruning algorithm}, which specifies the procedure that will be followed in the processing of the blocking graph, and (ii) the \textit{pruning criterion}, which determines the edges to be retained. The combination of a pruning algorithm with a pruning criterion forms a \textit{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 \textit{edge-centric algorithms} select the \textbf{globally} best comparisons by iterating over the \textit{edges} of a blocking graph in order to filter out those that do not satisfy the pruning criterion.

- The \textit{node-centric algorithms} iterate over the \textit{nodes} of a blocking graph with the aim of selecting the \textbf{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 \textit{functionality} of pruning criteria distinguishes them into \textbf{weight} thresholds, which specify the minimum weight for the edges to be retained, and \textbf{cardinality} thresholds, which determine the maximum number of retained edges. The \textit{scope} of pruning criteria distinguishes them into \textbf{global} thresholds, which define conditions that are applicable to the entire blocking graph (i.e., all the edges of the graph), and \textbf{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) [31], 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 our pruning algorithms with our pruning criteria.

![Fig. 4. All possible combinations of our pruning algorithms with our pruning criteria.](image-url)

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

### 3.3.1 Pruning Schemes with \textit{global} thresholds

In Figure 4, we illustrate all possible combinations of our pruning algorithms with our pruning criteria.

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

By definition, the \textit{node-centric algorithms} are compatible with \textbf{local} thresholds — regardless of their functionality. However, they can be combined with \textbf{global} thresholds, as well. Their combination with a \textbf{global} weight threshold is actually identical to \textit{WEP}, as they both retain the edges that are weighted higher than the given threshold.
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., $O(|E|)$).

The most critical part of this algorithm is the selection of the minimum edge weight $w_{\text{min}}$. 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 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., $O(|E|)$).

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., $CC_{\text{out}}=2^2$), while its BC is equal to $BC_{\text{out}}=2^{K+1}/2^2$, where $E$ is the size of the input entity collection. Thus, $CC_{\text{out}}$ is greater than or equal to $CC_{\text{in}}$ of the input blocking graph in all cases and, for an improved BC-CC mapping, it suffices to have: $BC_{\text{out}} \leq BC_{\text{in}} \Rightarrow \sum_{e \in E} w_e \geq K \frac{B_{\text{out}}}{B_{\text{in}}} \Rightarrow BC_{\text{in}} \equiv K \{ \frac{1}{1} \}_{BC_{\text{in}}}$, where $BC_{\text{in}}$ 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 $CC_{\text{in}} \ll CC_{\text{out}}$, we can set $K=\{ \frac{BC_{\text{out}}}{BC_{\text{in}}} \}$ 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 $G_i$ that comprises the nodes of $G_B$ connected with $v_i$.

---

**Algorithm 2: Weight Edge Pruning.**

Input: (i) $G_B^w$ the blocking graph, and (ii) $w_{\text{min}}$ the global weight pruning criterion.  
Output: $G_B^w$ the undirected pruned blocking graph.

1. foreach $e_{ij} \in E_B$ do
   2. if $e_{ij}.weight < w_{\text{min}}$ then // discard every edge with
      3. $E_B \leftarrow E_B \setminus \{ e_{ij} \}$: // weight lower than $w_{\text{min}}$
   4. return $G_B^w = \{ V_B, E_B, WS \}$

---

**Algorithm 3: Cardinality Edge Pruning.**

Input: (i) $G_B^w$ the blocking graph, and (ii) $K$ the global cardinality pruning criterion.  
Output: $G_B^w$ the undirected pruned blocking graph.

1. $\text{SortedStack} \leftarrow \{ \}$ // sorts edges in descending weight
2. foreach $e_{ij} \in E_B$ do // add every edge
   3. $\text{SortedStack}.push(e_{ij})$: // in the sorted stack
   4. if $K < \text{SortedStack}.size()$ then // remove the edge with
      5. $\text{SortedStack}.pop()$: // the (K+1)th top weight
   6. foreach $e_{ij} \in E_B$ do // discard all edges
      7. if $e_{ij} \notin \text{SortedStack}$ then // that are not among the
      8. $E_B \leftarrow E_B \setminus \{ e_{ij} \}$: // the top-K weighted ones
   9. return $G_B^w = \{ V_B, E_B, WS \}$
Algorithm 4: Weight Node Pruning.

Input: (i) $G_B^i$ the blocking graph, and
(ii) wt function for defining local weight pruning criteria.
Output: $G_{B_{out}}^i$ the directed pruned blocking graph
1 $E_{B_{out}}^i \leftarrow \{\}$; // the set of retained directed edges
2 foreach $v_i \in V_B$ do // for every node get
3 $G_i \leftarrow \text{neighborhood}(v_i, G_B)$; // its neighborhood and
4 $t_i \leftarrow \text{wt}(G_i)$; // its local weight threshold
5 foreach $e_{ij} \in E_B$ do // retain every adjacent
6 \hspace{1em} if $t_i \leq e_{ij}.\text{weight}$ then // edge with weight
7 \hspace{2em} $E_{B_{out}}^i \leftarrow E_{B_{out}}^i \cup \{e_{ij}\}$; // higher than $t_i$
8 return $G_{B_{out}}^i = \{V_B, E_{B_{out}}^i, WS\}$;

Algorithm 5: Cardinality Node Pruning.

Input: (i) $G_B^i$ the blocking graph, and
(ii) cf function for defining local cardinality pruning criteria.
Output: $G_{B_{out}}^i$ the directed pruned blocking graph
1 $E_{B_{out}}^i \leftarrow \{\}$; // the set of retained directed edges
2 foreach $v_i \in V_B$ do // for every node get
3 $\text{SortedStack}_k \leftarrow \{\}$; // for every node get
4 $G_i \leftarrow \text{neighborhood}(v_i, G_B)$; // its neighborhood and
5 $k \leftarrow \text{cf}(G_i)$; // its local cardinality threshold
6 foreach $e_{ij} \in E_B$ do // add every adjacent
7 \hspace{1em} $\text{SortedStack}_k$, push($e_{ij}$); // edge in sorted stack
8 if $k < \text{SortedStack}_k, \text{size}()$ then // remove the
9 \hspace{1em} $\text{SortedStack}_k, \text{pop}()$; // $(k+1)^{th}$ edge
10 return $G_{B_{out}}^i = \{V_B, E_{B_{out}}^i, WS\}$;

— denoted by $V_v$ — along with the edges connecting them — denoted by $E_v$. 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 $G_v$ (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 $E_v$ (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 $O(|V_B||E_B|)$; 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 $G_v$.

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 $G_v$ 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 $WNP$: $O(|V_B||E_B|)$. 

In general, the cardinality threshold for each neighborhood depends on its size (e.g., $k=\lceil 0.1 \cdot |E_v| \rceil$). For simplicity, 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 $CC_{out}^i = \frac{k+1}{k}$, while its BC is equal to $BC_{out}^i = k+1$. Thus, $k$ is specified with respect to the CC and the BC of the input block collection: $\frac{1}{1-CC_{in}^i} \leq k \leq BC_{in}^i-1$. In cases where $CC_{in}^i=1$, we can safely set $k=[BC_{in}^i-1]$, ensuring significantly higher efficiency ($CC_{out}^i>1$) 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) is transformed into the blocks $b_1 = \{(p_3), (p_5, p_4)\}$ 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
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.net4. 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 [27], [28], [29], namely $D_{\text{movies}}$, $D_{\text{infoboxes}}$ and $D_{\text{BTC09}}$. 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 researchers5. Their technical characteristics are summarized in Table 1.

$D_{\text{movies}}$ 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, $D_{\text{infoboxes}}$, consists of two different versions of the DBPedia Infobox dataset6. 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 are shared among them [27]. As matching entities, we consider those with the same entity URL.

Finally, $D_{\text{BTC09}}$ 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 collection7. 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 [27]. For $D_{\text{movies}}$, 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 $D_{\text{infoboxes}}$ 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 $D_{\text{BTC09}}$ were extracted from those produced by Total Description [29] 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 [29] 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 [32]. 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 $\Delta PC = \frac{PC(B') - PC(B)}{PC(B)}$, with $PC(B)$ and $PC(B')$ 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 $D_{\text{movies}}$, $D_{\text{infoboxes}}$ and $D_{\text{BTC09}}$. 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 weighing

---

5. See http://sourceforge.net/projects/erframework/files for instructions on how to download them.
schemes across all datasets. For $D_{\text{movies}}$ and $D_{\text{infoboxes}}$, 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 $D_{\text{BTC99}}$: 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 $D_{\text{BTC99}}$. In contrast, for $D_{\text{movies}}$ and $D_{\text{infoboxes}}$, 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 $D_{\text{movies}}$ (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 $D_{\text{movies}}$ 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 $D_{\text{movies}}$ and $D_{\text{infoboxes}}$. 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 $D_{\text{BTC99}}$, 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 $D_{\text{movies}}$, $D_{\text{infoboxes}}$ and $D_{\text{BTC99}}$, 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 $D_{\text{BTC99}}$). 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 $D_{\text{movies}}$, $D_{\text{infoboxes}}$ and $D_{\text{BTC99}}$, 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 $D_{\text{movies}}$: $PQ$ 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 $JS$.

(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 $D_{movies}$. 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 superfluous 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 Dmovies (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-t to 1.9-t with a step of 0.1-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-t, 0.5-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 meta-blocking, 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 (≤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 Dmovies (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 [18] and Iterative Blocking [32].

Acknowledgements. This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

References


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 post-doctoral 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.
There is a large amount of work on entity resolution ranging from string similarity metrics [7] to methods relying on transformations [30] and entity relationships [10]. Analytical overviews can be found in these surveys [9], [12], [14], tutorials [19], [26], and books [5], [24]. 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 \(E_1\) and \(E_2\) in Figure 5(a) and creates a set of blocks \(B\). 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 [17], bi-grams [2] and q-grams [15] blocking, Suffix Array [1], [8], HARRA [18], and Canopy Clustering [22]. A comparative analysis can be found in [6]. As this study points out, one of their major drawbacks is the fine-tuning of multiple parameters [8]. To ameliorate this issue, automatic methods can be trained to select the optimal parameter values [3], [23].

On the other hand, schema-agnostic blocking creates blocks solely on the basis of attribute values, i.e., without knowledge of the input schema(s). Semantic Indexing [25] 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 [27]. For RDF data, Total Description exploits semantics in the entity URIs, links between entities and tokens in the literal values of every profile [29]. 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.

Block processing receives as input a set of blocks \(B\) and produces as output the set of detected duplicates \(D_{\text{detected}}\) 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 \(D_{\text{detected}}\). This can be achieved by eliminating the redundant and the superfluous comparisons contained in \(B\). To this end, Block Purging [27] discards the largest blocks, while Block Scheduling [27] sorts blocks according to a probabilistic measure that estimates their likelihood to contain duplicates. Thus, it forms the basis for applying Block Pruning [27] and Duplicate Propagation [32]; 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 [32] 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 \(B\) created by such a method and its output is a new set of blocks \(B'\) that involves fewer comparisons than \(B\), 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 \(B\) 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 [28], 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).
Fig. 6. Two-dimensional taxonomy of block building methods.

**Experimental Outcomes**

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

<table>
<thead>
<tr>
<th>D_{movie}</th>
<th>D_{infoboxes}</th>
<th>D_{BTC09}</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBPedia</td>
<td>IMDB</td>
<td>DBP1</td>
</tr>
<tr>
<td>Entities</td>
<td>27,615</td>
<td>23,182</td>
</tr>
<tr>
<td>Name-Value Pairs</td>
<td>186,012</td>
<td>816,012</td>
</tr>
<tr>
<td>Blocks</td>
<td>40,430</td>
<td>22.52</td>
</tr>
<tr>
<td>BC</td>
<td></td>
<td>15.38</td>
</tr>
<tr>
<td>CC</td>
<td>4.27·10^2</td>
<td>1.29·10^-3</td>
</tr>
<tr>
<td>Brute Force Comp.</td>
<td>6.40·10^4</td>
<td>2.58·10^2</td>
</tr>
<tr>
<td>Block Comp.</td>
<td>2.67·10^5</td>
<td>3.98·10^10</td>
</tr>
<tr>
<td>Original RR</td>
<td>95.83%</td>
<td>98.46%</td>
</tr>
<tr>
<td>Existing Matches</td>
<td>22,405</td>
<td>892,562</td>
</tr>
<tr>
<td>Original PC</td>
<td>99.39%</td>
<td>99.89%</td>
</tr>
<tr>
<td>Original PQ</td>
<td>9.83·10^-4</td>
<td>2.24·10^-5</td>
</tr>
<tr>
<td>Edges</td>
<td>2.26·10^7</td>
<td>3.41·10^10</td>
</tr>
<tr>
<td>Nodes</td>
<td>5.06·10^4</td>
<td>3.33·10^6</td>
</tr>
</tbody>
</table>

**TABLE 1**

Overview of the evaluation datasets.

<table>
<thead>
<tr>
<th>D_{movie}</th>
<th>D_{infoboxes}</th>
<th>D_{BTC09}</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>CEP</td>
<td>PC</td>
</tr>
<tr>
<td>ARCS</td>
<td>89.16%</td>
<td>94.13%</td>
</tr>
<tr>
<td>CBS</td>
<td>80.42%</td>
<td>95.20%</td>
</tr>
<tr>
<td>ECBS</td>
<td>87.17%</td>
<td>96.69%</td>
</tr>
<tr>
<td>JS</td>
<td>89.22%</td>
<td>94.93%</td>
</tr>
<tr>
<td>EJS</td>
<td>91.03%</td>
<td>95.98%</td>
</tr>
</tbody>
</table>

**TABLE 2**

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

**TABLE 3**

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

<table>
<thead>
<tr>
<th>D_{movie} (minutes)</th>
<th>D_{infoboxes} (hours)</th>
<th>D_{BTC09} (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT</td>
<td>RT</td>
<td>CT</td>
</tr>
<tr>
<td>MT</td>
<td>RT</td>
<td>CT</td>
</tr>
<tr>
<td>MT</td>
<td>RT</td>
<td>CT</td>
</tr>
<tr>
<td>Baseline</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>ARCS</td>
<td>1.6</td>
<td>1.0</td>
</tr>
<tr>
<td>CBS</td>
<td>1.1</td>
<td>1.0</td>
</tr>
<tr>
<td>ECBS</td>
<td>1.1</td>
<td>1.2</td>
</tr>
<tr>
<td>JS</td>
<td>1.1</td>
<td>2.1</td>
</tr>
<tr>
<td>EJS</td>
<td>1.1</td>
<td>2.3</td>
</tr>
<tr>
<td>WB</td>
<td>2.5</td>
<td>5.9</td>
</tr>
<tr>
<td>WC</td>
<td>2.5</td>
<td>19.8</td>
</tr>
<tr>
<td>RC</td>
<td>1.8</td>
<td>17.2</td>
</tr>
<tr>
<td>PC</td>
<td>1.9</td>
<td>20.2</td>
</tr>
<tr>
<td>ARCS</td>
<td>1.6</td>
<td>1.3</td>
</tr>
<tr>
<td>CBS</td>
<td>1.1</td>
<td>1.0</td>
</tr>
<tr>
<td>ECBS</td>
<td>1.1</td>
<td>2.1</td>
</tr>
<tr>
<td>JS</td>
<td>1.1</td>
<td>3.0</td>
</tr>
<tr>
<td>EJS</td>
<td>1.1</td>
<td>3.6</td>
</tr>
<tr>
<td>ARCS</td>
<td>1.6</td>
<td>2.9</td>
</tr>
<tr>
<td>CBS</td>
<td>1.1</td>
<td>2.4</td>
</tr>
<tr>
<td>ECBS</td>
<td>1.2</td>
<td>2.4</td>
</tr>
<tr>
<td>JS</td>
<td>1.2</td>
<td>2.4</td>
</tr>
<tr>
<td>EJS</td>
<td>1.2</td>
<td>3.6</td>
</tr>
<tr>
<td>ARCS</td>
<td>1.6</td>
<td>3.0</td>
</tr>
<tr>
<td>CBS</td>
<td>1.1</td>
<td>2.4</td>
</tr>
<tr>
<td>ECBS</td>
<td>1.2</td>
<td>2.4</td>
</tr>
<tr>
<td>JS</td>
<td>1.2</td>
<td>2.4</td>
</tr>
<tr>
<td>EJS</td>
<td>1.2</td>
<td>3.6</td>
</tr>
</tbody>
</table>

**Fig. 7.** Normalized histograms of the weight distributions in all blocking graphs of D_{movie}, where w denotes the average edge weight of the blocking graph for each weighting scheme.
<table>
<thead>
<tr>
<th>Comp.</th>
<th>RR</th>
<th>PC</th>
<th>ΔPC</th>
<th>PQ</th>
<th>Iterative Bl.</th>
<th>RR</th>
<th>PC</th>
<th>ΔPC</th>
<th>PQ</th>
<th>BTC09</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(%</td>
<td>(%)</td>
<td>(% 10^3)</td>
<td>(%)</td>
<td></td>
<td>(%</td>
<td>(%)</td>
<td>(% 10^3)</td>
<td>(%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(%</td>
<td>(%)</td>
<td>(% 10^3)</td>
<td>(%)</td>
<td></td>
<td>(%</td>
<td>(%)</td>
<td>(% 10^3)</td>
<td>(%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(%</td>
<td>(%)</td>
<td>(% 10^3)</td>
<td>(%)</td>
<td></td>
<td>(%</td>
<td>(%)</td>
<td>(% 10^3)</td>
<td>(%)</td>
<td></td>
</tr>
</tbody>
</table>

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

(a) WEP

| 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 |

(b) WNP

| 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 | 98.68 | -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 |

(c) CEP

| ARCS | 0.57 | 97.87 | 82.75 | -16.74 | 3.25 | 0.26 | 99.94 | 94.46 | -20.46 | 276.83 | 0.09 | 99.85 | 92.17 | -4.92 | 103.99 |
| CBS | 0.57 | 97.87 | 75.19 | -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 |

(d) CNP

| 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 | -1.28 | 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 |

TABLE 4

Performance of all pruning schemes in combination with all weighting schemes over the three datasets of our study.
Sentiment Extraction from Tweets: Multilingual Challenges

Nantia Makrynioti(E) and Vasilis Vassalos

Athens University of Economics and Business, 76 Patission Street,
GR10434 Athens, Greece
{makriniotik,vassalos}@aueb.gr

Abstract. Every day users of social networks and microblogging services share their point of view about products, companies, movies and their emotions on a variety of topics. As social networks and microblogging services become more popular, the need to mine and analyze their content grows. We study the task of sentiment analysis in the well-known social network Twitter (https://twitter.com/). We present a case study on tweets written in Greek and propose an effective method that categorizes Greek tweets as positive, negative and neutral according to their sentiment. We validate our method’s effectiveness on both Greek and English to check its robustness on multilingual challenges, and present the first multilingual comparative study with three pre-existing state of the art techniques for Twitter sentiment extraction on English tweets. Last but not least, we examine the importance of different preprocessing techniques in different languages. Our technique outperforms two out of the three methods we compared against and is on a par to the best of those methods, but it needs significantly less time for prediction and training.

1 Introduction

Users have integrated microblogging services and social networks in their daily routine, and tend to share through them increasingly more thoughts and experiences of their lives. As a result, platforms, such as Twitter, are a goldmine for the tasks of opinion mining and sentiment analysis, providing valuable information on topics of timeliness or not, by users of varying social, educational and demographic background.

In this paper, we examine sentiment analysis in Twitter with emphasis on tweets written in Greek and we suggest a method based on supervised learning. Sentiment analysis is defined as the task of classifying texts, in case of Twitter these correspond to tweets, into categories depending on whether they express

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

© Springer International Publishing Switzerland 2015
DOI: 10.1007/978-3-319-22729-0_11
positive or negative emotion or whether they enclose no emotion at all. As a consequence, sentiment analysis solves two classification tasks, the identification of objective and subjective tweets and the categorization of the latter according to their polarity. Given a number of tweets, our task is to categorize them in three classes, positive, negative and neutral depending on the presence of features that indicate emotion or not, as most of the times this is consistent with the sentiment of the message [12].

Although recently many papers study the task of sentiment analysis and many approaches have been proposed, almost all of them regard English text and work for other languages is limited. Moreover, many studies do not report results from comparisons with other pre-existent methods and each technique is usually evaluated on a single dataset. Evaluation on different datasets, including data of more than one languages, is an interesting process, which cross-checks the performance of the methods among languages.

The contributions of our paper are summarized below:

1. We propose a novel method for classification of tweets into three categories, positive, negative and neutral, and we evaluate our classifier on real Greek and English tweets. Our method outperforms two of the three compared approaches while giving statistically indistinguishable results to the third but with significant less time.

2. We present a case study of sentiment analysis in the context of the Greek language, unlike English that are much more studied and understood. For this purpose we collected and manually annotated a corpus of posts in Greek from Twitter, in order to be used as training and test data.

3. We present extensive evaluation results and comparisons to three existing methods developed for English on a Greek as well as an English dataset. The purpose of these experiments is to provide the first comparative study of different state of the art techniques over Greek data, and examine their generalizability to address multilingual challenges. We also examine the contribution of specific preprocessing and postprocessing steps through ablation tests that demonstrate the degree to which certain steps of the proposed method improve the accuracy of the system with regard to Greek or English.

The rest of the paper is organized as follows. Section 2 presents some representative approaches on the problem of sentiment analysis and sect. 3 analyzes the data used for training and testing. In Sect. 4 at first we give an overview of our method and then we describe in detail every step of it. Results from the evaluation of the classifier and the comparative analysis are reported in Sect. 5. Finally, Sect. 6 concludes and presents ideas for future work.

2 Related Work

The mining and analysis of unstructured data from social networks has attracted considerable attention in recent years. Go et al. [9] dealt with sentiment analysis.
in Twitter, but their work was limited to positive and negative sentiments, and does not involve the recognition of objective (neutral) tweets. The machine learning algorithms that were applied are Multinomial Naive Bayes, Support Vector Machines (SVM) \[24\] and Maximum Entropy, whereas unigrams, bigrams as well as the combination of these two were used as features. Maximum accuracy reached 83% and was achieved with Maximum Entropy and both unigrams and bigrams. Pak and Paroubek \[20\] emphasized the preprocessing of tweets before classification and adopted bigrams, trigrams, negation and part-of-speech tags as features. They used entropy and introduced a variant of it called "salience" to select the most representative features. Their results show that bigrams outperform trigrams and salience discriminated n-grams better than entropy. The method described in \[5\] divides the classification of tweets into two stages. The first stage classifies subjective and objective tweets, while the second categorizes subjective tweets into positive and negative. Part-of-speech tags are used as features in this paper too. Dictionaries of subjective terms and syntax features of Twitter, such as hashtags, links, punctuation and words in capital letters, were also employed. The classifier used SVM and maximum error rate for the first stage reached 18.1%, whereas for the second stage it reached 18.7%.

Even though the paper by Pang et al. \[21\] is not about Twitter, it is a benchmark and a comparison point with all the studies mentioned above. The paper addresses the task of sentiment analysis in movie reviews. Features include unigrams, bigrams and negation. Multinomial Naive Bayes and Maximum Entropy were tested, but SVM achieved 82.9%, which was the maximum accuracy. Finally, a very recent approach by Mohammad et al. \[17\], which used a variety of features, including ngrams, syntax, lexicon and negation features, achieved the highest average F-score (69.02%) with a SVM classifier in SemEval 2013 (International Workshop on Semantic Evaluation) and the task of sentiment analysis in Twitter \[18\]. Our work falls into the same category with the aforementioned studies, but apart from the certain difference of experimenting on Greek data, we apply a different combination of features and preprocessing steps, followed by a novel postprocessing negation identification step, which attempts to recognize the structure of negation in text and reverse the given prediction, rather than affect the features used for classification. Moreover, we reproduce published methods and present comparisons of them on a multilingual fashion, experimenting on datasets from two languages, Greek and English. All the above approaches belong to the category of supervised learning, but many studies have also performed unsupervised sentiment analysis. Due to limited space, we do not mention them here.

As stated earlier, there is lack of studies concerning other languages than English and the task is not sufficiently examined from this perspective. The paper by Atteveldt et al. \[4\] presents a system for automatically determining the polarity of relations between actors, e.g. politicians and parties, and issues, such as unemployment and healthcare, in Dutch text. To determine the polarity of relations, the authors use existing techniques for sentiment analysis in English and show that these methods can be translated to Dutch. Another study
that addresses the multilingual perspective of the task is presented by Boiy and Moens [6]. The authors propose a supervised method for sentiment analysis and perform experiments on English, Dutch and French blog reviews and forum texts. There is also work about sentiment analysis on Modern Standard Arabic at the sentence level [2]. Arabic is a morphologically-rich language in contrast to English and the authors propose some Arabic-specific features along with the more commonly used and language-independent ones. Another work by Abbasi et al. [1] performs sentiment analysis on hate/extremist group forum postings in English and Arabic, and evaluates a variety of syntactic and stylistic features for this purpose. A method on Chinese data is also proposed by Zhao et al. in [26]. We are aware of a paper regarding reputation management on Greek data [22], but it presents a commercial product very briefly and in the abstract, and cannot be reproduced. Thus, our method not only is described extensively and in detail, but is also compared with other methods in the literature.

Finally, with regard to papers that compare methods and systems of sentiment analysis, such as [10] and [3], we take a step further and present comparisons in more than one languages.

## 3 Data

In this section we describe the datasets that are used for training and testing. Details about the size and contents of each dataset are given by Table 1. The Greek training data were collected between August 2012 and January 2015. Part of positive and negative tweets are based on subjective terms and around 20% of neutral tweets were gathered from accounts of newspapers and news sites. The rest were streamed randomly. Respectively, Greek test set consists of random tweets posted between October 2013 and January 2015. We used Twitter Search and Streaming API\(^2\) for the collection. Both training set and test set were labeled by three annotators. The calculated Fleiss’ kappa [7] for the training set is 0.83, which is interpreted as almost perfect agreement, whereas for the test set is 0.691, which denotes substantial agreement. We will refer to the Greek training and test set as GR–train and GR–test.

For experiments on English we use training and test data provided by the organizers of SemEval 2013 [18] for the task of sentiment analysis in Twitter. The organizers collected tweets according to popular topics, which included

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Positive</th>
<th>Negative</th>
<th>Neutral</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR–train</td>
<td>1870</td>
<td>2940</td>
<td>3190</td>
<td>8000</td>
</tr>
<tr>
<td>GR–test</td>
<td>261</td>
<td>249</td>
<td>378</td>
<td>888</td>
</tr>
<tr>
<td>Sem–train</td>
<td>3287</td>
<td>1601</td>
<td>4175</td>
<td>9063</td>
</tr>
<tr>
<td>Sem–test</td>
<td>1572</td>
<td>601</td>
<td>1640</td>
<td>3813</td>
</tr>
</tbody>
</table>

\(^2\) https://dev.twitter.com/.
named entities previously extracted by a Twitter-tuned NER system [23], and used Mechanical Turk for annotation. We will refer to SemEval training and test set as Sem–train and Sem–test respectively.

4 Overview of Approach

The approach we adopt consists of three main steps: (1) Preprocessing of data. (2) Feature engineering. (3) Reversal of classifier’s prediction for a tweet due to negation identification. The proposed method takes into account not only inflection but also word stress, both characteristics of morphologically-rich languages, and suggests a novel technique to reduce the negative effect of the combination of both in classification performance. Moreover, it treats identification of negation as a postprocessing step and attempts to capture its structure, which is a much different approach than adding a special suffix to bag-of-words features that most methods do until now. The aforementioned steps are described in detail in the following sections.

4.1 Preprocessing

Preprocessing is applied to both training and test set. The first step of preprocessing is the removal of noise from the data. Elements that do not indicate the polarity of a tweet are considered as noise. Such elements are listed below.

1. URL links.
2. Mentions of other users.
3. The abbreviation RT, which indicates that a tweet is a retweet of another one.
4. Stop words, including articles and pronouns. Stop words are extremely common words, which appear to be of little value in deciding the sentiment of a text.

Because users use plethora of emoticons/hashtags, we choose to replace positive emoticons with the emoticon “:)” and negative emoticons with the emoticon “:(”. A number of hashtags, such as #fail and #win, are also replaced with the former two emoticons. The aim of this step is to group the emoticons/hashtags in two categories and to avoid the need of importing tweets in the training set for each one of them. In addition to the above replacements, possible repetitive vowels encountered in a word are reduced to one, whereas repetitive consonants are reduced to two.

Capitalization and removal of accent marks are the next steps. An accent mark over the vowel in the stressed syllable is used in Greek to denote where the stress goes, e.g. ‘καλημέρα’ (good morning). In order to avoid mistakes due to omission of stress marks and incorrect use of capital letters versus lowercase letters, we remove these marks from tweets and transform them to uppercase. Stemming is the third and last step, and is used mostly to compensate for data sparseness. Stems are generated by George Ntais’ Greek stemmer [19] for Greek and by Lovins stemmer [15] via the Weka data mining software [11] for English.

3 List of positive emoticons: :-), :, :o), :], :3, :c), :, :, =], =), :, :, :, :, :, :, :, :, :, :, :, :, :, :.
4 List of negative emoticons: >:[, :(-, :,-c, :c, :<, :<, :<], :{, :{, :{'(, :/.
The previous steps are applied to the test set too. However, the preprocessing of test set involves an additional step: part-of-speech tagging. It takes place before stemming and is an auxiliary step for the process of negation identification (Sect. 4.3), which follows classification. After the replacements, we annotate the words of each tweet with part-of-speech tags, which are not taken into account by the classifier to predict the class, but are used in patterns whose intention is to detect negation. A Greek part-of-speech tagger is used for the tagging process in Greek, whereas in English the Carnegie Mellon University (CMU) Twitter Natural Language Processing (NLP) tool was selected.

4.2 Features

Feature engineering follows the bag-of-words representation with unigrams and term presence. Due to the limit in the number of characters that compose a tweet, a unigram is enough to denote the sentiment in most of the cases. For some unigrams there is a dependency with a particular class, while others do not give any information under any circumstances about the polarity. We decided to keep only a subset of them in order to eliminate noisy features and build a simpler model. We experimented with two methods, Information Gain and Chi-Squared. They both gave equally good results, so Information Gain is chosen arbitrarily for the experiments displayed in sections below.

Apart from word ngrams, lexicons of subjective terms, which contain terms with association to positive and negative sentiments, may provide various features for sentiment analysis. There are plenty of subjective lexicons in English, but we are not aware of any such lexicons in Greek. Nonetheless, we attempted to create manually two simple Greek subjective lexicons, one with positive words and one with negative words according to their prior polarity. Words were derived from random tweets, not contained in GR–train or GR–test, or translated from subjective English lexicons. The positive lexicon contains 199 words and the negative one consists of 292 words. We use two simple features, the presence of positive/negative terms of such lexicons in Greek tweets and more sophisticated features, such as those proposed in the aforementioned paper, for English data. In the aforementioned paper, lexicon-based features proved to be useful for the task of sentiment analysis. We present our conclusions about this kind of features in Sect. 5.

4.3 Negation Identification and Polarity Reversal

Negation identification is based on patterns of part-of-speech tags combined with negation words. We attempt to identify these patterns in each tweet and store the token that is negated. For example, the Greek word (not) followed by a verb and an adjective constitutes a negation pattern. If a tweet contains the phrase (The movie wasn't good), the former negation pattern will be identified due to the word, the verb (both correspond to wasn't) and the adjective (good). Then the token, which is the one that is negated, will be stored. Nine frequent patterns are recognized for Greek and eight for English. The detection of negation aims to reverse the prediction...
given by the classifier for a tweet from positive to negative or from negative to positive. If the prediction is neutral, no change is made. So following the decision of the classifier, we first check if a negated token is stored for the tweet. If yes, then we examine if this token belongs to the features that are present in the tweet. Suppose we have the aforementioned tweet for which we have kept the token as the negated token. If the unigram is one of the features and its value is 1, which indicates that this feature is present in the tweet, then the appropriate reversal of polarity will be performed. Otherwise, it will not.

4.4 Challenges of the Greek Language

The Greek language has a highly inflective nature that reduces the effectiveness of usual bag-of-words features. Greek verbs and adjectives are inflected for person, number and gender, which affects mostly the suffixes of the words. The various suffixes due to inflection increase ngram features, many of which are not contained in the training set. Hence, classification performance decreases. As a countermeasure to the inflective nature of Greek, the words of each tweet are replaced by their stems, assuming that stems are enough to denote the sentiment of a tweet in most cases.

Except from inflected verbs and adjectives, stress marks used in Greek make things even more complicated. Twitter users often forget to add these marks or they add them at the wrong syllable, creating this way a number of different versions of the same word (e.g. is a different unigram from ).

As stated earlier, we chose to remove accent marks in order to reduce ngram features, but in case of stemming this choice may lead to mistaken predictions. Specifically, although stemming operates positively and helps the method to generalize better on unseen data (a conclusion that is drawn from the ablation tests included in Sect. ), there is a case where it operates negatively: the stem of two words is the same whereas their polarities are different. For example the Greek words (agree) and (according to) have completely different meaning. The first word has positive polarity, whereas the second is neutral. Since the stem of both words is , there is no information to reveal the original word before stemming. As unigrams are used for predictions after stemming, the above case may be handled incorrectly. The described phenomenon can be frequently seen in Greek, even with words that are spelled exactly the same, but because they are stressed differently, their meaning changes. Note that these words are not homonyms as the word "like" for example, which serves both as verb and as proposition. In fact, purely homonyms with different polarity are extremely rare if non existent in the Greek language.

In order to handle the particular cases properly, a database is created with each record storing the following information: (i) stem, (ii) part-of-speech tag, (iii) polarity. The presence of unigrams (if any), on which the classifier based each prediction, are stored. If one of them, along with its part-of-speech tag, exists in the database, it is replaced with another one that has the same polarity. Specifically, if the unigram that exists in the database is positive, it will be replaced with the emoticon :) whereas if it is negative, it will be replaced with :(
the unigram is neutral, an article will replace it. At the moment, seven such cases are identified and stored in the database. However, this database can be continually improved by a user feedback mechanism.

The described particularities show that depending on language, different pre-processing steps may improve the performance of the classifier and thus it is not trivial to suggest a method that proves to be best for every language.

5 Experiments

There are two versions of the proposed method that are developed for the experiments. The first one uses SVM as the classification algorithm and we will refer to it as #Sentiment\textsubscript{v1}. The second version is called #Sentiment\textsubscript{v2} and uses Logistic Regression. SVM uses linear kernel and the value of parameter C is 1.0. The implementations of both algorithms are provided by the Weka data mining software. The section of experiments is divided in two parts. The first part presents the results of the evaluation on the Greek dataset collected by us, whereas the second includes results of experiments on English data provided by SemEval 2013. In these subsections we also compare the proposed method to three pre-existing methods developed for English \cite{5,9,17}, which we followed and implemented according to the descriptions in the corresponding papers. We will refer to these methods as Go\textsubscript{method}, Mohammad\textsubscript{method} and Barbosa\textsubscript{method} according to the first author. Due to space restrictions we do not describe these methods, but of course we provide the corresponding references for details.

The evaluation metrics we report in the experiments are average precision, recall and F-score, i.e. the sum of the corresponding metrics for each class divided by the number of classes. We also use McNemar's test \cite{16} to check the statistical significance of the difference in performance between systems in each experiment.

5.1 Greek Data

The experiments of this section concern the evaluation on Greek data gathered by us, i.e. the GR–test. We present a comparison between the two versions of our system and the three pre-existing methods described above. For Barbosa\textsubscript{method} we implement only two lexicon features, number of positive and number of negative words, as all other lexicon features depend on the structure of the MPQA lexicon \cite{25}, which is separated into strong subjective and weak subjective terms (this distinction does not exist in current Greek lexicons). Figure 1 displays the evaluation results of the five systems, #Sentiment\textsubscript{v1}, #Sentiment\textsubscript{v2}, Go\textsubscript{method}, Mohammad\textsubscript{method} and Barbosa\textsubscript{method} on Greek data. As far as our method is concerned, the difference in F-score between #Sentiment\textsubscript{v1} and #Sentiment\textsubscript{v2} on GR–test is statistically significant, which means that SVM outperforms Logistic Regression. However, our performance is statistically indistinguishable from Mohammad\textsubscript{method}. The other two methods by Go and Barbosa achieve much lower average F-scores.
The main conclusion of this experiment is that just the use of unigrams as features is not enough to achieve high accuracy in a classification problem with three classes. The Go method was originally tested on a two-class classification of English tweets and generated good results, but the extension of the method to three classes and on another language seems not so simple and would need further preprocessing steps/features to work. This is demonstrated by #Sentimentv1, #Sentimentv2 and Mohammad method, which also support the use of unigrams, but extend it with lexicon features, more preprocessing, such as stemming or feature selection, and achieve to reach higher average F-scores. Mohammad method addresses inflection by keeping all ngram features, which however means a much larger model and more training time.

5.2 English Data

This section is dedicated to experiments on English data provided by the organizers of SemEval 2013. Again we present a comparison between the system proposed (only #Sentimentv1, which is the best version according to the previous experiments) and the other three methods.

The evaluation results on the SemEval dataset (Sem–train and Sem–test) are displayed in Fig. 2. #Sentimentv1 and Mohammad method are again statistically indistinguishable and give the highest F-scores. Again methods that include both ngram and lexicon features, along with preprocessing and feature selection techniques, perform better on English data.
5.3 Time Consumption

In this section we present time performance results of the methods using the number of predicted tweets per second and training time. All methods ran on a single machine with an Intel Core i5 processor at 2.6 GHz and 16 GB of RAM. Details about time consumption of each method are given by Table 2.

Table 2. Time consumption

<table>
<thead>
<tr>
<th>Method</th>
<th>Predicted tweets/sec</th>
<th>Training time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Sentiment v1</td>
<td>16 tweets/sec</td>
<td>8.45 min</td>
</tr>
<tr>
<td>Mohammad method</td>
<td>9 tweets/sec</td>
<td>14.91 min</td>
</tr>
<tr>
<td>Go method</td>
<td>807 tweets/sec</td>
<td>5.9 min</td>
</tr>
<tr>
<td>Barbosa method</td>
<td>8 tweets/sec</td>
<td>15 min</td>
</tr>
</tbody>
</table>

Although #Sentiment v1 and Mohammad method are indistinguishable in terms of F-score, #Sentiment v1 needs 43% less prediction and training time. This difference in time performance is reasonable, since Mohammad method generates more features, such as part-of-speech and Twitter syntax features (RTs, hashtags, e.g.), which based on the experimental results they do not contribute that much to accuracy, but they increase processing time. Go method is by far the fastest method. This is because it only involves unigram features, which are quickly generated. Nevertheless, they fail to predict test data effectively as experimental results demonstrated.
5.4 Sensitivity Analysis

We also performed ablation tests in order to check how the omission of different steps of our method affects performance. Table 3 shows the effect of negation identification, feature selection and stemming on Greek and English data. The remarkable change in F-score in Greek after the omission of stemming is expected due to the inflective nature of the language. Notably, negation identification does not seem to matter a lot. This is probably due to the fact that many tweets are neutral and their polarity is not reversed, but also that the technique suffers from low recall. It tends to be quite precise and correctly reverse polarity when a negation pattern is captured and the negated token is one of the classification features. However, in many cases the negated token does not belong to the features and even though the pattern is again captured, no reversal takes place.

Table 3. Results of sensitivity analysis

<table>
<thead>
<tr>
<th>Modification</th>
<th>Avg F-score on Greek</th>
<th>Avg F-score on English</th>
</tr>
</thead>
<tbody>
<tr>
<td>No modification</td>
<td>68.6%</td>
<td>64.2%</td>
</tr>
<tr>
<td>Without negation identification</td>
<td>68.7%</td>
<td>64.1%</td>
</tr>
<tr>
<td>Without feature selection</td>
<td>66.7%</td>
<td>62.2%</td>
</tr>
<tr>
<td>Without stemming</td>
<td>63.1%</td>
<td>62.2%</td>
</tr>
</tbody>
</table>

6 Conclusion and Future Work

We present a method for sentiment analysis in Twitter focused on the Greek language. We perform the first multilingual comparative analysis and report comparison results to three leading existing methods, from experiments on two different datasets (Greek and English). Our method clearly outperforms two of the three methods we compared against in sentiment extraction, while being statistically indistinguishable from the third. However, the proposed method needs 43% less time for predictions and training. These experiments reveal that the generalization of a method to different languages or from a two to a three class classification problem is not trivial. Moreover, they give evidence about the effect of different preprocessing steps and features, such as stemming, in performance for Greek and English. An interesting idea to pursue in the future is the assignment of sentiment to the correct entity in the tweet.

References


