Generalized Supervised Meta-blocking

Luca Gagliardelli
U. of Modena and Reggio Emilia, Italy
luca.gagliardelli@unimore.it

George Papadakis
U. of Athens & PPC, Greece
gpapadis@di.uoa.gr

Giovanni Simonini
U. of Modena and Reggio Emilia, Italy
simonini@unimore.it

Sonia Bergamaschi
U. of Modena and Reggio Emilia, Italy
sonia.bergamaschi@unimore.it

Themis Palpanas
Université Paris Cité & IUF, France
themis@mi.parisdescartes.fr

ABSTRACT

Entity Resolution is a core data integration task that relies on Blocking to scale to large datasets. Schema-agnostic blocking achieves very high recall, requires no domain knowledge and applies to data of any structuredness and schema heterogeneity. This comes at the cost of many irrelevant candidate pairs (i.e., comparisons), which can be significantly reduced by Meta-blocking techniques that leverage the entity co-occurrence patterns inside blocks: first, pairs of candidate entities are weighted in proportion to their matching likelihood, and then, pruning discards the pairs with the lowest scores. Supervised Meta-blocking goes beyond this approach by combining multiple scores per comparison into a feature vector that is fed to a binary classifier. By using probabilistic classifiers, Generalized Supervised Meta-blocking associates every pair of candidates with a score that can be used by any pruning algorithm. For higher effectiveness, new weighting schemes are examined as features. Through extensive experiments, we identify the best pruning algorithms, their optimal sets of features, as well as the minimum possible size of the training set.

PVLDB Reference Format:
Luca Gagliardelli, George Papadakis, Giovanni Simonini, Sonia Bergamaschi, and Themis Palpanas. Generalized Supervised Meta-blocking. PVLDB, 15(9): 1902 - 1910, 2022.
doi:10.14778/3538598.3538611

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Gaglia88/sparker.

1 INTRODUCTION

Entity Resolution (ER) is the task of identifying entities that describe the same real-world object among different datasets [4, 10, 22, 32]. ER is a core data integration task with many applications that range from Data Cleaning in databases to Link Discovery in Semantic Web data [7, 10]. Despite the bulk of works on ER, it remains a challenging task [4, 14, 22, 24]. One of the main reasons is its quadratic time complexity: in the worst case, every entity has to be compared with all others, thus scaling poorly to large volumes of data.

Proceedings of the VLDB Endowment, Vol. 15, No. 9 ISSN 2150-8097.
doi:10.14778/3538598.3538611

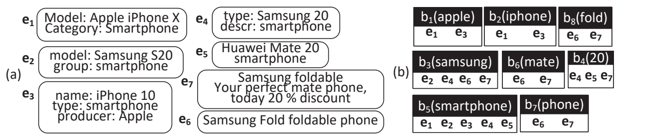
(a)
e1 Model: Apple iPhone X Category: Smartphone
e2 model: Samsung S20 group: smartphone
e3 name: iPhone 10 type: smartphone producer: Apple
e4 type: Samsung 20 descr: smartphone
e5 Huawei Mate 20 smartphone
e7 Samsung foldable Your perfect mate phone, today 20 % discount
e6 Samsung Fold foldable phone
(b)
b1(apple) e1 e3
b2(iphone) e1 e3
b8(fold) e6 e7
b3(samsung) e2 e4 e6 e7
b6(mate) e6 e7
b4(20) e4 e5 e7
b5(smartphone) e1 e2 e3 e4 e5
b7(phone) e6 e7

Figure 1: (a) The input entities (smartphone models), and (b) the redundancy-positive blocks produced by Token Blocking.

To tame its high complexity, *Blocking* is typically used [5, 6, 27, 28]. Instead of considering all possible pairs of entities, it restricts ER to *blocks* of entities that have identical or similar signatures. Extensive experimental analyses have demonstrated that the *schema-agnostic* signatures outperform the schema-based ones, without requiring domain or schema knowledge [5, 21]. As a result, parts of any attribute value in each entity can be used as signatures.

Example 1 (Schema-agnostic blocking). *The profiles in Figure 1a contain three duplicate pairs, $\langle e_1, e_3 \rangle$, $\langle e_2, e_4 \rangle$ and $\langle e_6, e_7 \rangle$, and are clustered using Token Blocking (a block is created for every token appearing in at least* 2 *profiles). The resulting blocks appear in Figure 1b. ER examines all pairs inside each block, detecting all duplicates.*

On the downside, the resulting blocks involve high levels of redundancy: every entity is associated with multiple blocks, thus yielding numerous *redundant* and *superfluous comparisons* [2, 31]. The former are pairs of entities that are repeated across different blocks, while the latter involve non-matching entities. For example, the pair $\langle e_1, e_3 \rangle$ is redundant in b_2, as it is already examined in b_1, while the pair $\langle e_2, e_6 \rangle \in b_3$ is superfluous, as the two entities are not duplicates. Both types of comparisons can be skipped, reducing the computational cost of ER without any impact on recall [20, 28].

To this end, *Meta-blocking* [23] discards all redundant comparisons, while reducing significantly the portion of superfluous ones. It relies on two components to achieve this goal:

1) A *weighting scheme*, which is a function that receives as input a pair of entities along with their associated blocks and returns a score proportional to their matching likelihood. The score is based on the co-occurrence patterns of the entities into the original set of blocks: the more blocks they share and the more distinctive (i.e., infrequent) the corresponding signatures are, the more likely they are to match and the higher is their score.

2) A *pruning algorithm*, which receives as input all weighted pairs and retains the ones that are more likely to be matching.

Example 2 (Unsupervised Meta-blocking). *Unsupervised Meta-blocking builds a blocking graph (Figure 2a) from the blocks in Figure 1b as follows: each entity profile is represented as a node; two nodes are connected by an edge if the corresponding profiles co-occur in*

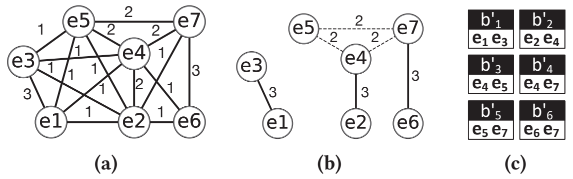

Figure 2: Unsupervised Meta-blocking example: (a) The blocking graph of the blocks in Figure 1b, using the number of common blocks as edge weights, (b) a possible pruned blocking graph, and (c) the new blocks.

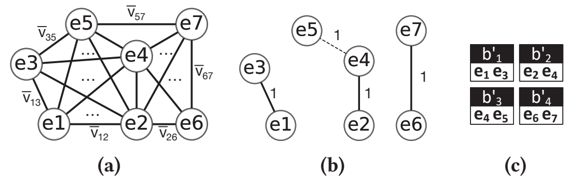

Figure 3: Supervised Meta-blocking example: (a) a graph where each edge is associated with a feature vector, (b) the graph pruned by a binary classifier, and (c) the output, which contains a new block per retained edge.

at least one block; each edge is weighted according to a weighting scheme—in our example, the number of blocks shared by the adjacent profiles. Finally, the blocking graph is pruned according to a pruning algorithm—in our example, for each node, we discard the edges with a weight lower than the average of its edges. The pruned blocking graph appears in Figure 2b, with the dashed lines representing the superfluous comparisons. A new block is then created for every retained edge. Figure 2c presents the final blocks, which involve significantly fewer pairs without missing the matching ones. This is a schema-agnostic process, just like the original blocking method.

Supervised Meta-blocking [25]. It models the restructuring of a set of blocks as a binary classification task. Its goal is to train a model that learns to classify every comparison as *positive* (i.e., likely to be matching) or *negative* (i.e., unlikely to be matching). Every pair is associated with a feature vector comprising the most distinctive weighting schemes that are used by learning-free meta-blocking.

Example 3 (Supervised Meta-blocking). *Figure 3a shows the blocking graph of the blocks in Figure 1b, where every edge is associated with a feature vector. For instance, each pair of entities $\langle e_i, e_j \rangle$ can be represented by a feature vector $v_{i,j} = \{CB(e_i, e_j), JS(e_i, e_j)\}$, where $CB(e_i, e_j)$ is the number of their common blocks and $JS(e_i, e_j)$ is the Jaccard coefficient of blocks associated with e_i and e_j. Then, a binary classifier is trained with a sample of labelled vectors and is used to predict whether a pair $\langle e_i, e_j \rangle$ is a match ($l_{i,j}$=1) or not ($l_{i,j}$=0). The pairs classified as positive are retained, as shown in Figure 3b (the dashed line indicates the superfluous pair $\langle e_4, e5 \rangle$). The end result, which includes a new block per retained pair, appears in Figure 3c.*

Supervised Meta-blocking involves the overhead of generating a labelled dataset, but by representing each edge with multiple features, it is more accurate in discriminating matching and non-matching pairs than Unsupervised Meta-blocking, which employs a single weight per edge. Indeed, Supervised Meta-blocking consistently yields better precision and recall than the unsupervised approach [25]. Yet, the binary classifier it employs acts as a learned, unique, *global* threshold used to prune the edges. Defining a *local* threshold for each node would allow a finer control on which edges to prune. This is the intuition behind Generalized Supervised Meta-blocking, as illustrated in the following example.

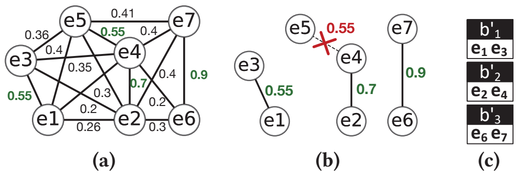

Figure 4: Generalized Supervised Meta-blocking example: (a) a graph weighted with a probabilistic classifier, (b) the pruned graph, and (c) the new blocks.

Example 4 (Generalized Supervised Meta-blocking). *Our new approach builds a graph where every edge is associated with a feature vector (as Supervised Meta-blocking does in Figure 3a) and trains a probabilistic classifier, which assigns a weight (the matching probability) to each edge (Figure 4a). Then, several weight- and cardinality-based algorithms can be applied. For example, Supervised WNP prunes the graph as follows: for each node, all adjacent edges with a weight lower that* 0.5 *are discarded; for the remaining edges, only those with a weight greater than the average one are kept. Figure 4b shows the result of this step: two edges may be assigned the same weight by the probabilistic classifier, e.g., $\langle e_1, e3 \rangle$ and $\langle e_4, e5 \rangle$, but they may be kept (e.g., the matching pair $\langle e_1, e3 \rangle$) or discarded (e.g., the non-matching pair $\langle e_4, e5 \rangle$) depending on their context, i.e., the weights in their neighborhood. Note that $\langle e4, e5 \rangle$ is not discarded by Supervised Meta-blocking in Figure 3b, which thus underperforms Generalized Supervised Meta-blocking in terms of precision (for the same recall).*

Our Contributions. Our work is motivated by a real-world application that aims to deduplicate a legacy customer database. It contains ~7.5 million entries that correspond to electricity supplies and, thus, are associated with an address, a customer name, and other optional attributes (e.g., tax id) that are typically empty. To exploit all available information, the quadratic computational cost of ER is reduced through schema-agnostic blocking. Our goal is to minimize the set of candidate pairs, using Supervised Meta-blocking, while restricting human involvement for the generation of the labelled instances. To this end, we go beyond Supervised Meta-blocking in the following ways:

- We generalize it from a binary classification task to a binary *probabilistic* classification process (Section 3).
- The resulting probabilities are used as comparison weights, on top of which we apply new pruning algorithms that are incompatible with the original approach [25] (Section 4).
- To further improve their performance, we use three new weighting schemes as features (Section 5).
- We perform an extensive experimental study that involves 9 real-world datasets. Its results demonstrate that the new pruning algorithms significantly outperform the existing ones. They also identify the top performing algorithms and feature vectors, showing that 50 labelled instances (25 per class) suffice for high performance.
- We also perform a scalability analysis over 5 synthetic datasets with up to 300,000 entities, proving that our approaches scale well both with respect to effectiveness and time-efficiency.

2 PRELIMINARIES

An entity profile e_i is defined as a set of name-value pairs, i.e., $e_i = \{\langle n_j, v_j \rangle\}$, where both the attribute names and the attribute values are textual. This simple model is flexible and generic enough to seamlessly accommodate a broad range of established data formats – from the structured records in relational databases to the semi-structured entity descriptions in RDF data [21]. Two entities, e_i and e_j, that describe the same real-world object are called *duplicates* or *matches*, denoted by $e_i \equiv e_j$. A set of entities is called *entity collection* and is denoted by E_l. An entity collection E_l is *clean* if it is duplicate-free, i.e., $\nexists\, e_i, e_j \in E_l : e_i \equiv e_j$.

We distinguish Entity Resolution into two tasks [5, 7, 22]: (i) *Clean-Clean ER* or *Record Linkage* receives as input two clean entity collections, E_1 & E_2, and detects the duplicates D between their entities, $D = \{(e_i, e_j) \subseteq E_1 \times E_2 : e_i \equiv e_j\}$; (ii) *Dirty ER* or *Deduplication* receives as input a dirty entity collection and detects the duplicates it contains, $D = \{(e_i, e_j) \subseteq E \times E : i \neq j \wedge e_i \equiv e_j\}$. The time complexity is quadratic with respect to the input, i.e., $O(|E_1| \times |E_2|)$ and $O(|E|^2)$, resp., as every entity profile has to be compared with all possible matches. To reduce this high computational cost, Blocking restricts the search space to similar entities [5, 22].

Meta-blocking operates on top of Blocking, refining an existing set of blocks B, a.k.a. *block collection*, as long as it is *redundancy-positive*. This means that every entity e_i participates into multiple blocks (i.e., $|B_i| \geq 1$, where $B_i = \{b \in B : e_i \in b\}$ denotes the set of blocks containing e_i), and the more blocks two entities share, the more likely they are to be matching, because they share a larger portion of their content. Such blocks emanate from Token, Q-Grams and Suffix Arrays Blocking and their variants among others [20, 28].

The redundancy-positive block collections involve a large portion of *redundant comparisons*, as the same pairs of entities are repeated across different blocks. These can be easily removed by aggregating for every entity $e_i \in E_1$ the set of all entities from E_2 that share at least one block with it [26]. The union of these individual sets yields the distinct set of comparisons, which is called *candidate pairs* and is denoted by C. Every non-redundant comparison between e_i and e_j, $\mathbf{c_{i,j}} \in \mathbf{C}$, belongs to one of the following types:

- *Positive pair* if e_i and e_j are matching: $e_i \equiv e_j$.
- *Negative pair* if e_i and e_j are not matching: $e_i \not\equiv e_j$.

These definitions are independent of Matching: two matching (non-matching) entities are positive (negative) as long as they share at least one block in B [5, 21]. The set of all positive and negative pairs in a block collection B are denoted by P_B and N_B, respectively. The goal of Meta-blocking is to transform a given block collection B into a new one B' such that $|P_{B'}| \approx |P_B|$ and $|N_{B'}| \ll |N_B|$.

Supervised Meta-blocking models every pair $c_{i,j} \in C$ as a feature vector $f_{i,j} = [s_1(c_{i,j}), s_2(c_{i,j}), ..., s_n(c_{i,j})]$, where each s_i is a weighting scheme score proportional to the matching likelihood of $c_{i,j}$. The feature vectors for all pairs in C are fed into a *binary classifier*, which labels them as `positive` or `negative`, if their entities are highly likely to match or not. Its performance is assessed through: *(i)* the true positive $TP(C)$ and negative $TN(C)$ pairs correctly classified as `positive` and `negative`, resp., and *(ii)* the incorrectly classified false positive $FP(C)$ and negative $FN(C)$ pairs.

Supervised Meta-blocking discards all candidate pairs labelled as `negative`, i.e., $TN(C) \cup FN(C)$, retaining those belonging to $TP(C) \cup FP(C)$. A new block is created for every `positive` pair, yielding the new block collection B'. Thus, the effectiveness of Supervised Meta-blocking is assessed through the following measures, defined in $[0, 1]$, with higher values indicating better performance:

- *Recall*, a.k.a. *Pairs Completeness*, expresses the portion of existing duplicates that are retained: $Re{=}|TP(C)|/|D|{=}(|D| - FN(C))/|D|$.
- *Precision*, a.k.a. *Pairs Quality*, is the portion of positive candidate pairs that are matching: $Pr = |TP(C)|/(|TP(C)| + |FP(C)|)$.
- *F-Measure* is the harmonic mean of the two: $F1{=}2{\cdot}\mathrm{Re}{\cdot}\mathrm{Pr}/(\mathrm{Re}{+}\mathrm{Pr})$.

In this context, Supervised Meta-blocking is formalized as [25]:

Problem 1. *Given the candidate pairs C of block collection B, the labels L={`positive`, `negative`} and a training set $T = \{\langle c_{i,j}, l_k \rangle : c_{i,j} \in C \wedge l_k \in L\}$, Supervised Meta-blocking aims to learn a classification model M that minimizes the cardinality of $FN(C) \cup FP(C)$ so that the new block collection B' achieves much higher precision than B, $Pr(B') \gg Pr(B)$, but maintains the original recall, $Re(B') \approx Re(B)$.*

The time efficiency of Supervised Meta-blocking is assessed through its running time, RT. This includes the time required to: (i) generate the feature vectors for all candidate pairs in C, (ii) train the classification model M, and (iii) apply M to C.

Pruning algorithms. To address Problem 1, three pruning algorithms were introduced in [25]: 1) The *Binary Classifier* (BCl), which simply retains all pairs classified as positive.

2) *Cardinality Edge Pruning* (CEP), which retains the top-K weighted candidate pairs, where K is set to half the sum of block sizes in the input blocks B, i.e., $K{=}\sum_{b_i \in B} |b|/2$, where $|b|$ stands for the size of block b, i.e., the number of entities it contains [23].

3) *Cardinality Node Pruning* (CNP), which adapts CEP to a local operation, maintaining the top-k weighted candidates per entity, where k amounts to the average number of blocks per entity: $k = max(1, \sum_{b \in B} |b|/(|E_1| + |E_2|))$ [23].

Weighting schemes. These algorithms were mostly combined with the following schemes, which are schema-agnostic and generic enough to cover any redundancy-positive block collection B [25]:

1) *Co-occurrence Frequency-Inverse Block Frequency* (CF-IBF). Inspired from Information Retrieval's TF-IDF, it assigns high scores to entities that participate in few blocks, but co-occur in many: $\mathit{CF\text{-}IBF}(c_{i,j}) = |B_i \cap B_j| \cdot \log |B|/|B_i| \cdot \log |B|/B_j|$.

2) *Reciprocal Aggregate Cardinality of Common Blocks* (RACCB). The smaller the blocks shared by a pair of candidates, the more distinctive information they have in common and, thus, the more likely they are to be matching: $RACCB(c_{i,j}) = \sum_{b \in B_i \cap B_j} 1/||b||$, where $||b||$ is the cardinality of block b–the number of its candidate pairs.

3) *Jaccard Scheme* (JS). It expresses the portion of blocks shared by a pair of candidates: $JS(c_{i,j}) = |B_i \cap B_j|/(|B_i| + |B_j| - |B_i \cap B_j|)$.

4) *Local Candidate Pairs* (LCP). It measures the number of candidates for a particular entity: $LCP(e_i){=}||e_i||{=}|\{e_j : i \neq j \wedge |B_i \cap B_j| > 0\}|$. The less candidates correspond to an entity, the more likely it is to match with one of them. Entities with many candidates convey no distinctive information, being unlikely for any match.

5) *Enhanced Jaccard Scheme* (EJS). Based on the same principle as LCP, it enhances JS with the inverse frequency of an entity's candidates in C: $EJS(c_{i,j}) = JS(c_{i,j}) \cdot \log |C|/||e_i|| \cdot \log |C|/||e_j||$.

The first four schemes formed the feature vector that achieves the best balance between effectiveness and time efficiency in [25]. LCP appears twice in the feature vector of c_{ij}, as LCP(e_i) and LCP(e_j).

3 PROBLEM DEFINITION

Generalized Supervised Meta-blocking is a new task that differs from Supervised Meta-blocking in two ways: (i) instead of a *binary* classifier that assigns class labels, it trains a *probabilistic* classifier that assigns a weight $w_{i,j} \in [0,1]$ to every candidate pair $c_{i,j}$. This weight expresses how likely it is to belong to the positive class. (ii) The candidate pairs with a probability lower than 0.5 are discarded, but the rest, called *valid pairs*, are further processed by a pruning algorithm. The ones retained after pruning yield the new block collection B', which contains a new block per retained valid pair.

Hence, the performance evaluation of Generalized Supervised Meta-blocking relies on the following measures: *(i)* $TP'(C)$, the probabilistic true positive pairs, involve duplicates that are assigned to a probability≥0.5 and are retained after pruning; *(ii)* $FP'(C)$, the probabilistic false positive pairs, entail non-matching entities, that are assigned to a probability≥0.5 and are retained by the pruning algorithm. *(iii)* $TN'(C)$, the probabilistic true negative pairs, entail non-matching entities that are assigned to a probability<0.5 and are discarded by the pruning algorithm. *(iii)* $FN'(C)$, the probabilistic false negative pairs, comprise matching entities, that are assigned to a probability<0.5 and are discarded by the pruning algorithm.

The measures of recall, precision and F-Measure are redefined accordingly. In this context, the task of Generalized Supervised Meta-blocking is formally defined as follows:

Problem 2. *Given the candidate pairs C of block collection B, the labels L={`positive`, `negative`}, and a training set $T = \{\langle c_{i,j}, l_k\rangle : c_{i,j} \in C \land l_k \in L\}$, the goal of Generalized Supervised Meta-blocking is to train a probabilistic classification model M that assigns a weight $w_{i,j} \in [0,1]$ to every candidate pair $c_{i,j} \in C$; these weights are then processed by a pruning algorithm so as to minimize the cardinality of $FN'(C) \cup FP'(C)$, yielding a new block collection B' that achieves much higher precision than B, $Pr(B') \gg Pr(B)$, while maintaining the original recall, $Re(B') \approx Re(B)$.*

The run-time of Generalized Supervised Meta-blocking, RT, adds to that of Supervised Meta-blocking the time required to process the assigned probabilities by a pruning algorithm.

4 PRUNING ALGORITHMS

To address Problem 2, our new supervised pruning algorithms operate as follows: given a specific set of features, they train a probabilistic classifier on the labelled instances. Then, they apply the trained classification model M to each candidate pair, estimating its classification probability. If it exceeds 0.5, a threshold determines whether the corresponding pair of entities will be retained or not.

Depending on the type of threshold, the pruning algorithms are categorized into two types: (i) The *weight-based algorithms* determine the weight(s), above which a comparison is retained. (ii) The *cardinality-based algorithms* determine the number k of top-weighted comparisons to be retained. In both cases, the determined threshold is applied either *globally*, on all candidate pairs, or *locally*, on the candidate pairs associated with every individual entity.

We define the following four weight-based pruning algorithms:

1) *Weighted Edge Pruning* (WEP). It iterates over the set of candidate pairs C twice: first, it applies the trained classifier to each pair in order to estimate the average probability $\bar{p}$ of the valid ones. Then, it applies again the trained classifier to each pair and retains only those pairs with a probability higher than $\bar{p}$

2) *Weighted Node Pruning* (WNP). It iterates twice over C, too. Yet, instead of a global average probability, it estimates a local average probability per entity. It keeps in memory two arrays: one with the sum of valid probabilities per entity and one with the number of valid candidates per entity. They are populated during the first iteration over C and are used to compute the average probability per entity. Finally, WNP iterates over C and retains a pair $c_{i,j}$ only if its probability $p_{i,j}$ exceeds either of the related average probabilities.

3) *Reciprocal Weighted Node Pruning* (RWNP). The only difference from WNP is that a comparison is retained if its classification probability exceeds both related average probabilities. This way, it applies a consistently deeper pruning than WNP.

4) *BLAST.* It is similar to WNP, but uses a different pruning criterion. Instead of the average probability per entity, it relies on the maximum probability per entity. It stores these probabilities in an array that is populated during the first iteration over C. The second iteration over C retains a valid pair $c_{i,j}$ if it exceeds a certain portion r of the sum of the related maximum probabilities.

We also define a new cardinality-based pruning algorithm: *Reciprocal Cardinality Node Pruning* (RCNP) performs a deeper pruning than CNP by retaining only the candidate pairs that are among the top-k weighted ones for both constituent entities.

Please refer to [12] for more detailed descriptions.

5 WEIGHTING SCHEMES

Among the features of [25], CF-IBF, JS and EJS rely on $|B_i \cap B_j|$, the number of blocks shared by a candidate pair $c_{i,j}$. In an effort to ensure high distinctiveness, avoiding ties, CF-IBF couples this number with a factor that discounts the contribution of e_i and/or e_j if they appear in many blocks, because such entities are typically dominated by noisy, ambiguous text (e.g., stop words). JS normalizes $|B_i \cap B_j|$ by considering the total number of blocks containing e_i and e_j, while EJS extends it with another factor that considers the distinctiveness of the textual information in an entity profile: the number of candidate pairs per e_i and e_j. The higher this number is, the less likely are e_i and e_j to match.

Another type of valuable matching evidence in redundancy-positive block collections is the sum of the inverse cardinalities of common blocks. RACCB assumes that the higher this sum is, the more distinctive is the information shared by e_i and e_j, thus being more likely to match. Unlike $|B_i \cap B_j|$, RACCB does not need to be combined with any discount factor (as in CF-IBF and EJS), because it produces highly distinctive scores. However, it considers only the blocks shared by e_i and e_j, disregarding the contextual information about the rest of the blocks that contain these two entities.

To address this issue, the *Weighted Jaccard Scheme* (WJS) normalizes RACCB with the cardinality of all blocks containing each entity [1]: $WJS(c_{i,j}) = \frac{\sum_{b \in B_i \cap B_j} 1/||b||}{\sum_{b \in B_i} 1/||b|| + \sum_{b \in B_j} 1/||b|| - \sum_{b \in B_i \cap B_j} 1/||b||}$. WJS promotes candidate pairs co-occurring in the most and smallest blocks, sharing a larger portion of their distinctive textual content.

Another type of matching evidence in redundancy-positive block collections, which has been overlooked in the literature, is the inverse size of common blocks. Similar to RACCB, the smaller the common blocks are, the more likely are the corresponding candidate pairs to be matching. This is encapsulated by the *Reciprocal Sizes Scheme* (RS) [1]: $RS(c_{i,j}) = \sum_{b \in B_i \cap B_j} 1/|b|$.

Table 1: The datasets used in the experimental study.

	Dataset	$\|E_1\|$	$\|E_2\|$	$\|D\|$	$\|C\|$	Recall	Precision	$F1$
(a)	AbtBuy	1.1k	1.1k	1.1k	36.7k	0.948	$2.78\cdot10^{-2}$	$5.40\cdot10^{-2}$
	DblpAcm	2.6k	2.3k	2.2k	46.2k	0.999	$4.81\cdot10^{-2}$	$9.18\cdot10^{-2}$
	ScholarDblp	2.5k	61.3k	2.3k	832.7k	0.998	$2.80\cdot10^{-3}$	$5.58\cdot10^{-3}$
	AmazonGP	1.4k	3.3k	1.3k	84.4k	0.840	$1.29\cdot10^{-2}$	$2.54\cdot10^{-2}$
	ImdbTmdb	5.1k	6.0k	1.9k	109.4k	0.988	$1.78\cdot10^{-2}$	$3.50\cdot10^{-2}$
	ImdbTvdb	5.1k	7.8k	1.1k	119.1k	0.985	$8.90\cdot10^{-3}$	$1.76\cdot10^{-2}$
	TmdbTvdb	6.0k	7.8k	1.1k	198.6k	0.989	$5.50\cdot10^{-3}$	$1.09\cdot10^{-2}$
	Movies	27.6k	23.1k	22.8k	26.0M	0.976	$8.59\cdot10^{-4}$	$1.72\cdot10^{-3}$
	WMAmazon	2.5k	22.1k	1.1k	27.4M	1.000	$4.22\cdot10^{-5}$	$8.44\cdot10^{-5}$
(b)	D_{10k}	10k		8.7k	$2.69\cdot10^{7}$	0.999	$3.23\cdot10^{-4}$	$6.47\cdot10^{-4}$
	D_{50k}	50k		43.1k	$6.73\cdot10^{8}$	0.999	$6.40\cdot10^{-5}$	$1.28\cdot10^{-4}$
	D_{100k}	100k		85.5k	$2.69\cdot10^{9}$	0.999	$3.17\cdot10^{-5}$	$6.34\cdot10^{-5}$
	D_{200k}	200k		172.4k	$1.08\cdot10^{10}$	1.000	$1.60\cdot10^{-5}$	$3.19\cdot10^{-5}$
	D_{300k}	300k		257.0k	$2.43\cdot10^{10}$	0.999	$1.06\cdot10^{-5}$	$2.12\cdot10^{-5}$

Similar to RACCB, RS is context-agnostic, considering exclusively information from the common blocks of two entities. To enhance it, the *Normalized Reciprocal Sizes Scheme* (NRS) extends RS with the contextual information of all blocks containing the constituent entities of a candidate pair [1]:
$$NRS(c_{i,j}) = \frac{\sum_{b\in B_i\cap B_j} 1/|b|}{\sum_{b\in B_i} 1/|b| + \sum_{b\in B_j} 1/|b| - \sum_{b\in B_i\cap B_j} 1/|b|}.$$

In general, the normalized weighting schemes yield more distinctive features, because they encompass more information about a given pair of candidate matches. Moreover, the size and cardinality of blocks provide more distinctive information about e_i and e_j than the mere number of blocks they share. For this reason, the new features (i.e., WJS, RS and NRS) are expected to enhance significantly the performance of the pruning algorithms.

6 EXPERIMENTAL EVALUATION

Hardware and Software. All the experiments were performed on a machine equipped with four Intel Xeon E5-2697 2.40 GHz (72 cores), 216 GB of RAM, running Ubuntu 18.04. We employed the *SparkER* library [13] to perform blocking and features generation. Unless stated otherwise, we perform machine learning analysis using Python 3.7 and the Support Vector Classification (SVC) model of scikit-learn [30], in particular. We used the default configuration parameters, enabling the generation of probabilities and fixing the random state so as to reproduce the probabilities over several runs. We performed all experiments with logistic regression, too, obtaining almost identical results, but we omit them for brevity.

Datasets. Table 1a lists the 9 real-world datasets employed in our experiments ($|E_x|$ stands for the number of entities in an entity collection, $|D|$ for the number of duplicate pairs). They have different characteristics and cover a variety of domains. Each dataset involves two different, but overlapping data sources, where the ground truth of the real matches is known. AbtBuy matches products extracted from Abt.com and Buy.com [18]. DblpAcm matches scientific articles extracted from dblp.org and dl.acm.org [18]. ScholarDblp matches scientific articles extracted from scholar.google.com and dblp.org [18]. ImdbTmdb, ImdbTvdb and TmdbTvdb match movies and TV series extracted from IMDB, TheMovieDB and TheTVDB [19], as suggested by their names. Movies matches information about films that are extracted from imdb.com and dbpedia.org [21]. WMAmazon matches products from Walmart.com and Amazon.com [8].

Blocking. To each dataset, we apply Token Blocking, the only parameter-free redundancy-positive blocking method [28]. The original blocks are then processed by Block Purging [21], which discards all the blocks that contain more than half of all entity profiles in a parameter-free way. These blocks correspond to highly frequent signatures (e.g., stop-words) that provide no distinguishing information. Finally, we apply Block Filtering [26], removing each entity e_i from the largest 20% blocks in which it appears.

The performance of the resulting block collections is reported in Table 1a. We observe that in most cases, the block collections achieve an almost perfect recall that significantly exceeds 90%. The only exception is AmazonGP, where some duplicate entities share no *infrequent* attribute value token – the recall, though, remains quite satisfactory, even in this case. Yet, the precision is consistently quite low, as its highest value is lower than 0.003. As a result, F1 is also quite low, far below 0.1 across all datasets. *These settings undoubtedly call for Supervised Meta-blocking.*

To apply Generalized Supervised Meta-blocking to these block collections, we performed 10 runs and averaged the values of precision, recall, and F1. In each run, a different seed is used to sample the pairs that compose the training set. Using undersampling, we formed a balanced training set per dataset that comprises 500 labelled instances. Due to space limitations, we mostly report the average performance of every approach over the 9 block collections.

Pruning Algorithm Selection. We now investigate which are the best-performing weight- and cardinality-based pruning algorithms for Generalized Supervised Meta-blocking among those discussed in Section 4. As baseline methods, we employ the pruning algorithms proposed in [25]: the binary classifier **BCl** for weight-based algorithms as well as CEP and CNP for the cardinality-based ones. We fixed the training set size to 500 pairs and used the feature vector proposed in [25] as optimal; every candidate pair $c_{i,j}$ is represented by the vector: $\{CF\text{-}IBF(c_{i,j}), RACCB(c_{i,j}), JS(c_{i,j}), LCP(e_i), LCP(e_j)\}$. Based on preliminary experiments, we set the pruning ratio of BLAST to r=0.35. The average effectiveness measures of the weight- and cardinality based algorithms across the 9 block collections of Table 1a are reported in Tables 2a and b, respectively.

Among the weight-based algorithms, we observe that the new pruning algorithms trade slightly lower recall for significantly higher precision and F1. Comparing BCl with WEP, recall drops by -5.9%, while precision raises by 60.8% and F1 by 42.9%. This pattern is more intense in the case of RWNP, which reduces recall by -7.2%, increasing precision by 68.5% and F1 by 46.3%. These two algorithms actually monopolize the highest F1 scores in every case: for ImdbTmdb, ImdbTvdb and TmdbTvdb, WEP ranks first with RWNP second and vice versa for the rest of the datasets. Their aggressive pruning, though, results in very low recall ($\ll$0.8) in four datasets. E.g., in the case of AbtBuy, BCl's recall is 0.852, but WEP and RWNP reduce it to 0.755 and 0.699, respectively.

The remaining algorithms are more robust with respect to recall. Compared to BCl, WNP reduces recall by just -0.2%, while increasing precision by 26.8% and F1 by 19.7%. Yet, BLAST outperforms WEP with respect to all effectiveness measures: recall, precision and F1 raise by 1.3%, 13.8% and 11.5%, respectively. This means that BLAST is able to discard much more non-matching pairs, while retaining a few more matching ones, too.

Among the cardinality-based algorithms, we observe that RCNP is a clear winner, outperforming both CEP and CNP. Compared to the former, it reduces recall by -1.1%, while increasing precision by 44% and F1 by 34.4%; compared to the latter, recall drops by -3.5%, but precision and F1 raise by 37.5% and 29.3%, respectively.

Table 2: The average performance of all pruning algorithms over the block collections of Table 1.

Alg.	Recall	Precision	F1
BCl	.8673	.1700	.2559
WEP	.8163	.2734	.3656
WNP	.8659	.2156	.3063
RWNP	.8047	.2864	.3744
BLAST	.8784	.1936	.2852

(a) weight-based pruning algorithms

Alg.	Recall	Precision	F1
CEP	.8632	.1744	.2639
CNP	.8854	.1827	.2743
RCNP	.8513	.2434	.3484

(b) cardinality-based pruning algorithms

Table 3: The 10 feature sets with the highest F1 per algorithm.

ID	Feature set	Recall	Pre-cision	F1	RT (minutes)	
					Movies	WMAm.
72	{CF-IBF, RACCB, JS, RS}	.8816	.1932	.2892	2.73	2.96
74	{CF-IBF, RACCB, JS, NRS}	.8816	.1932	.2892	2.48	2.99
75	{CF-IBF, RACCB, JS, WJS}	.8816	.1932	.2892	2.44	2.86
78	{CF-IBF, RACCB, RS, NRS}	.8816	.1932	.2892	2.15	2.37
79	{CF-IBF, RACCB, RS, WJS}	.8816	.1932	.2892	2.52	2.83
82	{CF-IBF, RACCB, NRS, WJS}	.8816	.1932	.2892	2.47	2.83
86	{CF-IBF, JS, RS, WJS}	.8816	.1932	.2892	2.52	3.01
89	{CF-IBF, JS, NRS, WJS}	.8816	.1932	.2892	2.49	3.00
96	{CF-IBF, RS, NRS, WJS}	.8816	.1932	.2892	2.52	2.82
190	{CF-IBF, RACCB, JS, RS, NRS, WJS}	.8816	.1932	.2892	2.57	3.21

(a) BLAST

ID	Feature set	Recall	Pre-cision	F1	RT (minutes)	
					Movies	WMAm.
184	{CF-IBF, RACCB, JS, LCP, RS}	.8489	.2463	.3527	6.41	11.13
187	{CF-IBF, RACCB, JS, LCP, WJS}	.8490	.2464	.3526	6.20	10.35
193	{CF-IBF, RACCB, LCP, RS, NRS}	.8490	.2463	.3526	6.46	11.18
200	{CF-IBF, JS, LCP, RS, NRS}	.8488	.2474	.3526	6.60	11.66
227	{CF-IBF, RACCB, JS, LCP, RS, NRS}	.8493	.2473	.3537	6.63	12.22
228	{CF-IBF, RACCB, JS, LCP, RS, WJS}	.8494	.2473	.3537	6.46	11.51
231	{CF-IBF, RACCB, JS, LCP, NRS, WJS}	.8496	.2473	.3537	6.68	11.04
235	{CF-IBF, RACCB, LCP, RS, NRS, WJS}	.8496	.2473	.3536	6.57	11.00
239	{CF-IBF, JS, LCP, RS, NRS, WJS}	.8494	.2473	.3534	6.50	10.87
250	{CF-IBF, RACCB, JS, LCP, RS, NRS, WJS}	.8502	.2479	.3542	6.51	11.27

(b) RCNP

Table 4: BLAST&RCNP vs best existing algorithms, BCl&CNP.

Algorithm	Recall	Precision	F1	*RT*(Movies)	*RT*(WMAmazon)
BCl	0.8673	0.1700	0.2559	7.11 min.	10.67 min.
BLAST	0.8816	0.1932	0.2892	3.32 min.	3.37 min.
CNP	0.8858	0.1827	0.2639	7.53 min.	10.93 min.
RCNP	0.8490	0.2464	0.3526	7.01 min.	11.11 min.

Overall, RCNP constitutes the best choice for cardinality-based pruning algorithms, which are crafted for applications that promote precision at the cost of slightly lower recall [23, 26]. BLAST is the best among the weight-based pruning algorithms, which are crafted for applications that promote recall at the cost of slightly lower precision [23, 26]. Note that their F1 is significantly higher than the original ones in Table 1a, but still far from perfect. The reason is that (Supervised) Meta-blocking merely produces a new block collection, not the end result of ER. This block collection is then processed by a Matching algorithm, whose goal is to raise F1 close to 1.

Feature selection. We now fine-tune the selected algorithms, BLAST and RCNP, by identifying the feature sets that optimize their performance in terms of effectiveness and time-efficiency. We adopted a brute force approach, trying all the possible combinations of the eight features presented in Sections 2 and 5. Fixing again the training set size to a random sample of 500 balanced instances, the top-10 feature vectors with respect to F1 for BLAST and RCNP are reported in Tables 3a and b, respectively.

We observe that both algorithms are robust with respect to the top-10 feature sets, as they all achieve practically identical performance, on average. For BLAST, we obtain recall=0.882, precision=0.193 and F1=0.289 when combining *CF-IBF* and *RACCB* with any two features from f={JS, RS, NRS, WJS}; even *RACCB* can be replaced with a third feature from f without any noticeable impact. For RCNP, we obtain recall=0.850, precision=0.248 and F1=0.353 when combining *CF-IBF*, *RACCB* and *LCP* with any pair of features from {JS, RS, NRS, WJS}. In this context, we select the best feature set for each algorithm based on *time efficiency*.

In more detail, we compare the top-10 feature sets per algorithm in terms of their running times. This includes the time required for calculating the features per candidate pair and for retrieving the corresponding classification probability (we exclude the time required for producing the new block collections, because this is a fixed overhead common to all feature sets of the same algorithm). Due to space limitations, we consider only the two datasets with the most candidate pairs, as reported in Table 1a: `Movies` and `WMAmazon`. We repeated every experiment 10 times and took the mean time.

In Table 3a, we observe that the feature set **78** is consistently the fastest one for BLAST, exhibiting a clear lead. Compared to the second fastest feature sets over `movies` (75) and `WMAmazon` (96), it reduces the average run-time by 11.9% and 16.0%, respectively. For RCNP, the differences are much smaller, yet the same feature set (**187**) achieves the lowest run-time over both datasets. Compared to the second fastest feature sets over `movies` (184) and `WMAmazon` (239), it reduces the average run-time by 3.3% and 4.8%, respectively.

Overall, BLAST models each candidate pair as the 4-dimensional feature vector (ID 78 in Table 3a): {CF-IBF, $RACCB$, RS, NRS}. Compared to the feature set of [25], recall raises by ~0.5% and F1 by ~1.5%, while the run-time is reduced to a significant extent (>50% as explained below), due to the absence of the time-consuming LCP feature. RCNP represents every candidate pair with the 5-dimensional feature vector (ID 187 in Table 3b): {CF-IBF, $RACCB$, JS, LCP, WJS}. This reduces recall by <0.3%, but raises precision and F-Measure by 1.2%, which is in-line with the desiderata of cardinality-based algorithms.

Comparison with Supervised Meta-blocking [25]. We now compare BLAST and RCNP in combination with the features selected above with BCl and CNP, which use the feature set proposed in [25], {CF-IBF, $RACCB$, JS, LCP}. All algorithms were trained over the same randomly selected set of 500 labeled instances, 250 from each class, and were applied to all datasets in Table 1a. Their average performance is presented in Table 4.

We observe that BLAST outperforms BCl with respect to all effectiveness measures: its recall, precision and F1 are higher by 1.6%, 13.6% and 13%, respectively, on average. Thus, *BLAST is much more accurate in the classification of the candidate pairs and more suitable than BCl for recall-intensive applications.* Among the cardinality-based algorithms, RCNP trades slightly lower recall than CNP for significantly higher precision and F1: on average, across all datasets, its recall is lower by -4.1%, while its precision and F1 are higher by 34.9% and by 33.6%, respectively. As a result, *RCNP is more suitable than CNP for precision-intensive applications.*

Regarding the running times of these algorithms on the largest datasets, i.e., `Movies` and `WMAmazon`, we observe that BCl, CNP and RCNP exhibit similar *RT* in both cases, since they all employ more complex feature sets that include the time-consuming feature *LCP*. BLAST is substantially faster than these algorithms, reducing *RT* by more than 50%. In particular, comparing it with its weight-based competitor, *we observe that BLAST is faster than BCl by 2.1 times over* `Movies` *and by 3.2 times over* `WMAmazon`.

The effect of training set size. We now explore how the performance of BLAST and RCNP changes when varying the training set size. We used the features sets selected above (ID 78 and 187

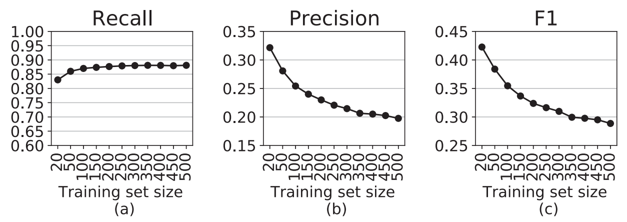

Figure 5: The effect of the training set size on BLAST.

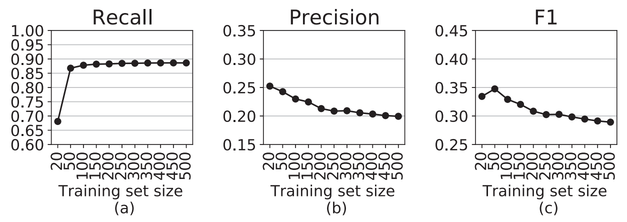

Figure 6: The effect of the training set size on RCNP.

in Tables 3a and b, respectively) and varied the number of labelled instances starting from 20, then from 50 to 500 with a step of 50. Figures 5 and 6 report the results in terms of recall, precision and F1, on average across all datasets, for BLAST and RCNP, respectively.

Notice that both algorithms exhibit the same behavior: as the training set size increases, recall gets higher at the expense of lower precision and F1. However, the increase in recall is much lower than the decrease in the other two measures. More specifically, comparing the largest training set size with the smallest one, the average recall of BLAST raises by 2.4%, while its average precision drops by 29.7% and its average F1 by 24.8%. Similar patterns apply to RCNP: recall raises by 2.1%, but precision and F1 drop by 17.8% and 16.8%, respectively, when increasing the labelled instances from 50 to 500. This might seem counter-intuitive, but is caused by the distribution of matching probabilities: for small training sets, these probabilities are relatively evenly distributed in [0.5, 1, 0], but for larger ones, they are concentrated in higher scores, closer to 1.0, while the pruning threshold remains practically stable. As a result, more true and false positives exceed the threshold with the increase in the training set size, as explained in detail in [12].

Given that 20 labelled instances yield very low recall, especially for RCNP, but with 50 instances, the recall becomes quite satisfactory for both algorithms ($\gg$0.85, on average, across all datasets), we can conclude that *the optimal training set involves just 50 labelled instances, equally split among positive and negative ones.*

Comparison with Supervised Meta-blocking [25]. Table 5a-c reports a full comparison between the main weight-based algorithms, i.e., BLAST and BCl (note that BCl$_2$ uses the training set specified in [25], i.e., a random sample involving 5% of the positive instances in the ground-truth along with an equal number of randomly selected negative instances). We observe that on average, BLAST outperforms BCl$_2$ with respect to all effectiveness measures, increasing the average recall, precision and F1 by 7.1%, 5.0% and 9.9%, respectively. Compared to BCl$_1$, BLAST increases the average recall by 3.95%, at the cost of slightly lower precision and F1 (5.9% and 2.2%, respectively). Recall drops below 0.8 in four datasets for BCl$_1$ (and BCl$_2$), whereas BLAST violates this limit in just two datasets. This should be attributed to duplicate pairs that share just one block in the original block collection, due to missing or erroneous values, as

Table 5: Performance of the main weight- and cardinality-based algorithms across all datasets in a-c and d-f, respectively. *RT* is the mean run-time (in seconds) over 10 repetitions.

	Abt Buy	Dblp Acm	Scholar Dblp	Amazon GP	Imdb Tmdb	Imdb Tvdb	Tmdb Tvdb	Movies	Walmart Amazon
Re	0.8345	0.9511	0.9638	0.7001	0.8223	0.7483	0.8466	0.9151	0.9587
Pr	0.2037	0.6509	0.3418	0.1441	0.5756	0.2304	0.2477	0.1300	0.0025
F1	0.3265	0.7690	0.4988	0.2385	0.6726	0.3456	0.3770	0.2221	0.0050
RT	6.58	5.62	11.90	6.83	6.46	6.36	7.51	96.01	107.82

(a) BLAST with 50 labelled pairs and $\{CF\text{-}IBF, RACCB, RS, NRS\}$

	Abt Buy	Dblp Acm	Scholar Dblp	Amazon GP	Imdb Tmdb	Imdb Tvdb	Tmdb Tvdb	Movies	Walmart Amazon
Re	0.8345	0.9521	0.9588	0.6265	0.7889	0.6966	0.6972	0.9039	0.9500
Pr	0.1821	0.5971	0.3595	0.1607	0.6445	0.2616	0.3737	0.0972	0.0020
F1	0.2981	0.7303	0.5195	0.2572	0.7086	0.3785	0.4613	0.1735	0.0041
RT	5.40	5.66	10.51	6.02	5.79	5.49	6.69	82.71	107.51

(b) BCl$_1$ with 50 labelled pairs and $\{CF\text{-}IBF, RACCB, RS, NRS\}$

	Abt Buy	Dblp Acm	Scholar Dblp	Amazon GP	Imdb Tmdb	Imdb Tvdb	Tmdb Tvdb	Movies	Walmart Amazon
Re	0.8183	0.9513	0.9303	0.7316	0.7872	0.7074	0.8172	0.9100	0.5757
Pr	0.2039	0.6130	0.3921	0.1131	0.5969	0.2323	0.2312	0.0239	0.0001
F1	0.3261	0.7425	0.5401	0.1908	0.6604	0.3395	0.2991	0.0465	0.0001
RT	15.07	9.37	27.73	13.22	11.04	9.68	10.86	1,328.81	276.19

(c) BCl$_2$ with the training set and the features of [25], i.e., $\{CF\text{-}IBF, RACCB, JS, LCP\}$

	Abt Buy	Dblp Acm	Scholar Dblp	Amazon GP	Imdb Tmdb	Imdb Tvdb	Tmdb Tvdb	Movies	Walmart Amazon
Re	0.8405	0.9759	0.9623	0.7358	0.8395	0.7465	0.8696	0.9275	0.9122
Pr	0.1764	0.6463	0.3591	0.1264	0.3540	0.2325	0.1848	0.0992	0.0050
F1	0.2914	0.7747	0.5190	0.2148	0.4971	0.3498	0.2954	0.1758	0.0100
RT	6.20	5.67	11.73	6.83	6.55	6.77	8.32	126.13	107.56

(d) RCNP with 50 labelled pairs and $\{CF\text{-}IBF, RACCB, JS, LCP, WJS\}$

	Abt Buy	Dblp Acm	Scholar Dblp	Amazon GP	Imdb Tmdb	Imdb Tvdb	Tmdb Tvdb	Movies	Walmart Amazon
Re	0.8294	0.9613	0.9218	0.7462	0.8045	0.7615	0.8641	0.8200	0.7087
Pr	0.1797	0.5984	0.3745	0.1031	0.5471	0.1867	0.1720	0.0090	0.0002
F1	0.2939	0.7355	0.5095	0.1748	0.6394	0.2847	0.2487	0.0177	0.0004
RT	5.95	5.80	11.33	6.40	5.91	6.19	6.89	122.72	107.62

(e) CNP$_1$ with 50 labelled pairs and $\{CF\text{-}IBF, RACCB, JS, LCP, WJS\}$

	Abt Buy	Dblp Acm	Scholar Dblp	Amazon GP	Imdb Tmdb	Imdb Tvdb	Tmdb Tvdb	Movies	Walmart Amazon
Re	0.8347	0.9539	0.9581	0.7742	0.8345	0.7641	0.8677	0.9347	0.2332
Pr	0.1895	0.6158	0.2184	0.0848	0.4132	0.1764	0.1484	0.0291	0.0001
F1	0.3081	0.7457	0.3453	0.1514	0.5247	0.2754	0.2363	0.0564	0.0002
RT	15.61	9.64	28.51	13.63	11.37	9.99	11.41	1,351.54	365.03

(f) CNP$_2$ with the training set and the features of [25], i.e., $\{CF\text{-}IBF, RACCB, JS, LCP\}$

explained in detail in [12]. BCl$_1$ outperforms BCl$_2$ in all respects, demonstrating the effectiveness of the new feature set.

In terms of run-time, BLAST is slower than BCl$_1$ by 8.2%, on average, because it iterates once more over all candidate pairs. Compared to BCl$_2$, BLAST is 6.7 times faster, on average across all datasets, because of *LCP* and of the large training sets, which learn complex binary classifiers with a time-consuming processing.

Regarding the cardinality-based algorithms, we observe in Table 5d-f that RCNP typically outperforms both baseline methods with respect to all effectiveness measures. Compared to CNP$_1$ (CNP$_2$), RCNP raises the average recall by 5.3% (9.2%), while achieving the highest precision and F1 across all datasets, except for `AbtBuy` and `ImdbTmdb` (and `ScholarDblp` in the case of CNP$_1$). The relative increase in precision in comparison CNP$_1$ to ranges from 7.5% over `TmdbTvdb` to 10 and 24 times over `Movies` and `WalmartAmazon`, respectively. Compared to CNP$_2$, precision raises from 5.0% over `DblpAcm` to 49 times over `WalmartAmazon`. In all cases, F1 increases to a similar extent. These patterns suggest that RCNP is typically more accurate in classifying the positive candidate pairs.

In terms of run-time, RCNP is slower than CNP$_1$ by 6.2%, on average, as it retains the candidate pairs that are among the top-k weighted ones for both constituent entities (i.e., it searches for pairs in two lists), whereas CNP$_1$ simply merges the lists of all entities. CNP$_2$ employs a much larger training set, yielding more complicated and time-consuming classifiers than RCNP, which is 3 times faster, on average, across all datasets.

Overall, Generalized Supervised Meta-blocking outperforms Supervised Meta-blocking to a significant extent, despite using a balanced training set of just 50 labelled instances.

Scalability Analysis. We assess the scalability of our approaches as the number of candidate pairs $|C|$ increases, verifying their robustness under versatile settings: instead of real-world Clean-Clean

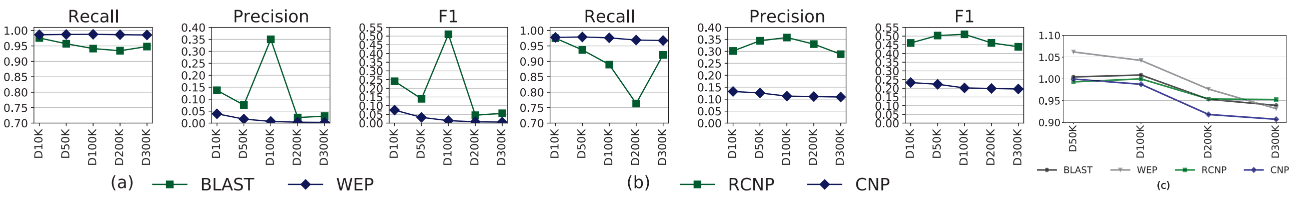

Figure 7: Scalability over the datasets in Table 1b: (a) the weight-based pruning algorithms, (b) the cardinality-based ones, and (c) speedup.

ER datasets, we now consider the synthetic Dirty ER datasets, and instead of SVC, we train our models using Weka's default implementation of Logistic Regression [17].

The characteristics of the datasets, which are widely used in the literature [5, 28], appear in Table 1b. To extract a large block collection from every dataset, we apply Token Blocking. In all cases, the recall is almost perfect, but precision and F1 are extremely low.

We consider four methods: BCl and CNP with the features and the training set size specified in [25] as well as BLAST and RCNP with the features in Tables 5a and 5d, resp., trained over 50 labelled instances (25 per class). In each dataset, we performed three repetitions per algorithm and considered the average performance.

The effectiveness of the weight- and cardinality-based algorithms over all datasets appear in Figures 7a and 7b, respectively. BLAST significantly outperforms BCl in all cases: on average, it reduces recall by 3.5%, but consistently maintains it above 0.93, while increasing precision and F1 by a whole order of magnitude. Note that BLAST's precision is much higher than expected over D_{100K}, due to the effect of random sampling: a different training set is used in every one of the three iterations, with two of them performing a very deep pruning, for a minor decrease in recall.

RCNP outperforms CNP to a significant extent: on average, it reduces recall by 7.9%, but maintains it to very high levels – except for D_{200K}, where it drops to 0.77, due to the effect of random sampling; yet, precision raises by 2.8 times and F1 by 2.3 times. These results verify the strength of our approaches, even though they require orders of magnitude less labelled instances than [25].

Most importantly, our approaches scale better to large datasets, as demonstrated by speedup in Figure 7c. Given two sets of candidate pairs, $|C_1|$ and $|C_2|$, such that $|C_1| < |C_2|$, this measure is defined as follows: $speedup = |C_2|/|C_1| \times RT_1/RT_2$, where RT_1 (RT_2) denotes the running time over $|C_1|$ ($|C_2|$) – in our case, C_1 corresponds to D_{10K} and C_2 to all other datasets. In essence, speedup extrapolates the running time of the smallest dataset to the largest one, with values close to 1 indicating linear scalability, which is the ideal case. We observe that all methods start from very high values, but BCl and CNP deteriorate to a significantly larger extent than BLAST and RCNP, respectively, achieving the lowest values for D_{300K}. This should be attributed to their lower accuracy in pruning the non-matching comparisons, which deteriorates as the number of candidate pairs increases. As a result, they end up retaining and processing a much larger number of comparisons, which slows down their functionality.

Overall, Generalized Supervised Meta-blocking scales much better to large datasets than Supervised Meta-blocking [25] for both weight- and cardinality-based algorithms. For a bit lower recall, it raises precision and F1 by ≥2 times and maintains a much higher speedup.

7 RELATED WORK

The *unsupervised* pruning algorithms WEP, WNP, CEP, and CNP were introduced in [23]. WNP and CNP were redefined in [26] so that they do not produce block collections with redundant comparisons. *Unsupervised* Reciprocal WNP and Reciprocal CNP were coined in [26], while *unsupervised* BLAST was proposed in [31].

Over the years, more unsupervised pruning algorithms have been proposed in the literature. [36] proposes a variant of CEP that retains the top-weighted candidate pairs with a cumulative weight higher than a specific portion of the total sum of weights. Crafted for Semantic Web data, MinoanER [11] combines meta-blocking evidence from two complementary block collections: the blocks extracted from the names of entities and from the attribute values of their neighbors. BLAST2 [2] leverages loose schema information in order to boost the performance of Meta-blocking's weighting schemes. Finally, a family of pruning algorithms that focuses on the comparison weights inside individual blocks is presented in [9]. Our approaches can be generalized to these algorithms, too, but their analytical examination lies out of our scope.

The above works consider Meta-blocking in a static context that ignores Matching. A dynamic approach that leverages Meta-blocking is *pBlocking* [16]. After applying Matching to the smallest blocks, intersections of the initial blocks are formed and scored based on their ratio of matching and non-matching entities. Meta-blocking is then applied to produce a new set of candidates to be processed by Matching. This process is iteratively applied until convergence. *BEER* [15] is an open-source implementation of pBlocking.

On another line of research, BLOSS [3] introduces an active learning approach that reduces the size of the labelled set required by Supervised Meta-blocking. It partitions the unlabelled candidate pairs into similarity levels based on CF-IBF, it applies rule-based active sampling inside every level and cleans the labelled sample from non-matching outliers with high Jaccard weight. Our approaches render BLOSS unnecessary, as they require just 50 labelled instances.

8 CONCLUSIONS

We presented Generalized Supervised Meta-blocking, which casts Meta-blocking as a probabilistic binary classification task and weights all candidate pairs in a block collection through a trained probabilistic classifier. Its weights are processed by pruning algorithms that are weight-based, promoting recall, or cardinality-based, promoting precision. BLAST and RCNP constitute the best algorithms, resp. We showed that four new weighting schemes give rise to feature sets that outperform the existing ones [25], while a very small, balanced training set with just 50 labelled instances suffices for high effectiveness, high time efficiency and high scalability. In the future, we will apply our approaches to Progressive ER [29, 33–35].

REFERENCES

[1] N. Augsten, R. Kwitt, M. Lissandrini, W. Mann, T. Palpanas, and G. Papadakis. 2021. *New Weighting Schemes for Meta-blocking*. Technical Report LIPADE-TR 5. Laboratoire d'Informatique PAris DEscartes (LIPADE). Available at http://lipade.mi.parisdescartes.fr/wp-content/uploads/2021/10/LipadeTR-5.pdf.

[2] Domenico Beneventano, Sonia Bergamaschi, Luca Gagliardelli, and Giovanni Simonini. 2020. *BLAST2*: An Efficient Technique for Loose Schema Information Extraction from Heterogeneous Big Data Sources. *ACM J. Data Inf. Qual.* 12, 4 (2020), 18:1–18:22.

[3] Guilherme Dal Bianco, Marcos André Gonçalves, and Denio Duarte. 2018. BLOSS: Effective meta-blocking with almost no effort. *Inf. Syst.* 75 (2018), 75–89.

[4] Peter Christen. 2012. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer.

[5] Peter Christen. 2012. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *TKDE* 24, 9 (2012), 1537–1555.

[6] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2021. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.* 53, 6 (2021), 127:1–127:42. https://doi.org/10.1145/3418896

[7] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. 2015. *Entity Resolution in the Web of Data*. Morgan & Claypool.

[8] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. [n.d.]. The Magellan Data Repository. https://sites.google.com/site/anhaidgroup/projects/data.

[9] Dimas Cassimiro do Nascimento, Carlos Eduardo Santos Pires, and Demetrio Gomes Mestre. 2020. Exploiting block co-occurrence to control block sizes for entity resolution. *Knowl. Inf. Syst.* 62, 1 (2020), 359–400.

[10] Xin Luna Dong and Divesh Srivastava. 2015. *Big Data Integration*. Morgan & Claypool Publishers.

[11] Vasilis Efthymiou, George Papadakis, Kostas Stefanidis, and Vassilis Christophides. 2019. MinoanER: Schema-Agnostic, Non-Iterative, Massively Parallel Resolution of Web Entities. In *EDBT*. 373–384.

[12] Luca Gagliardelli, George Papadakis, Giovanni Simonini, Sonia Bergamaschi, and Themis Palpanas. 2022. *Generalized Supervised Meta-blocking (Extended Version)*. Technical Report. Available at http://arxiv.org/abs/2204.08801.

[13] Luca Gagliardelli, Giovanni Simonini, Domenico Beneventano, and Sonia Bergamaschi. 2019. SparkER: Scaling Entity Resolution in Spark. In *EDBT*. 602–605.

[14] Luca Gagliardelli, Giovanni Simonini, and Sonia Bergamaschi. 2020. RulER: Scaling Up Record-level Matching Rules. In *EDBT*. OpenProceedings.org, 611–614. https://doi.org/10.5441/002/edbt.2020.76

[15] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. 2021. BEER: Blocking for Effective Entity Resolution. In *SIGMOD*. 2711–2715.

[16] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. 2021. Efficient and effective ER with progressive blocking. *VLDB J.* 30, 4 (2021), 537–557.

[17] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter* 11, 1 (2009), 10–18.

[18] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *PVLDB* 3, 1-2 (2010), 484–493.

[19] Daniel Obraczka, Jonathan Schuchart, and Erhard Rahm. 2021. EAGER: Embedding-Assisted Entity Resolution for Knowledge Graphs. *arXiv preprint arXiv:2101.06126* (2021).

[20] George Papadakis, George Alexiou, George Papastefanatos, and Georgia Koutrika. 2015. Schema-agnostic vs Schema-based Configurations for Blocking Methods on Homogeneous Data. *PVLDB* 9, 4 (2015), 312–323.

[21] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederée, and Wolfgang Nejdl. 2012. A blocking framework for entity resolution in highly heterogeneous information spaces. *TKDE* 25, 12 (2012), 2665–2682.

[22] George Papadakis, Ekaterini Ioannou, Emanouil Thanos, and Themis Palpanas. 2021. *The Four Generations of Entity Resolution*. Morgan & Claypool Publishers. https://doi.org/10.2200/S01067ED1V01Y202012DTM064

[23] George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl. 2014. Meta-Blocking: Taking Entity Resolution to the Next Level. *TKDE* 26, 8 (2014), 1946–1960.

[24] George Papadakis, Georgios M. Mandilaras, Luca Gagliardelli, Giovanni Simonini, Emmanouil Thanos, George Giannakopoulos, Sonia Bergamaschi, Themis Palpanas, and Manolis Koubarakis. 2020. Three-dimensional Entity Resolution with JedAI. *Inf. Syst.* 93 (2020), 101565. https://doi.org/10.1016/j.is.2020.101565

[25] George Papadakis, George Papastefanatos, and Georgia Koutrika. 2014. Supervised meta-blocking. *PVLDB* 7, 14 (2014), 1929–1940.

[26] George Papadakis, George Papastefanatos, Themis Palpanas, and Manolis Koubarakis. 2016. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking.. In *EDBT*. 221–232.

[27] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and Filtering Techniques for Entity Resolution: A Survey. *ACM Comput. Surv.* 53, 2 (2020), 31:1–31:42. https://doi.org/10.1145/3377455

[28] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. 2016. Comparative Analysis of Approximate Blocking Techniques for Entity Resolution. *PVLDB* 9, 9 (2016), 684–695.

[29] Thorsten Papenbrock, Arvid Heise, and Felix Naumann. 2015. Progressive Duplicate Detection. *IEEE Trans. Knowl. Data Eng.* 27, 5 (2015), 1316–1329.

[30] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[31] Giovanni Simonini, Sonia Bergamaschi, and H. V. Jagadish. 2016. BLAST: a Loosely Schema-aware Meta-blocking Approach for Entity Resolution. *PVLDB* 9, 12 (2016), 1173–1184.

[32] Giovanni Simonini, Luca Gagliardelli, Sonia Bergamaschi, and H. V. Jagadish. 2019. Scaling entity resolution: A loosely schema-aware approach. *Inf. Syst.* 83 (2019), 145–165. https://doi.org/10.1016/j.is.2019.03.006

[33] Giovanni Simonini, George Papadakis, Themis Palpanas, and Sonia Bergamaschi. 2019. Schema-Agnostic Progressive Entity Resolution. *IEEE Trans. Knowl. Data Eng.* 31, 6 (2019), 1208–1221. https://doi.org/10.1109/TKDE.2018.2852763

[34] Giovanni Simonini, Luca Zecchini, Sonia Bergamaschi, and Felix Naumann. 2022. Entity Resolution On-Demand. *PVLDB* 15, 7 (2022), 1506–1518. https://doi.org/10.14778/3523210.3523226

[35] Steven Euijong Whang, David Marmaros, and Hector Garcia-Molina. 2013. Pay-As-You-Go Entity Resolution. *IEEE Trans. Knowl. Data Eng.* 25, 5 (2013), 1111–1124.

[36] Fulin Zhang, Zhipeng Gao, and Kun Niu. 2017. A pruning algorithm for meta-blocking based on cumulative weight. In *Journal of Physics*, Vol. 887.