Generalized Supervised Meta-blocking

George Papadakis
National and Kapodistrian University of Athens,Greece
Greece
gpapadis@di.uoa.gr

Giovanni Simonini
Università degli studi di Modena e Reggio Emilia
Italy
giovanni.simonini@unimore.it

Sonia Bergamaschi
Università degli studi di Modena e Reggio Emilia
Italy
sonia.bergamaschi@unimore.it

Themis Palpanas
Universite de Paris and the French UniversityInstitute
(IUF)
France
themis@mi.parisdescartes.fr

ABSTRACT

Entity Resolution constitutes a core data integration task that relies on Blocking in order to tame its quadratic time complexity. 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 through Meta-blocking techniques, i.e., techniques that leverage the co-occurrence patterns of entities inside the blocks: first, a weighting scheme assigns a score to every pair of candidate entities in proportion to the likelihood that they are matching and then, a pruning algorithm 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 an extensive experimental analysis, we identify the best pruning algorithms, their optimal sets of features as well as the minimum possible size of the training set. The resulting approaches achieve excellent performance across several established benchmark datasets.

PVLDB Reference Format:
George Papadakis, Giovanni Simonini, Sonia Bergamaschi, and Themis Palpanas. Generalized Supervised Meta-blocking. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code and data have been made available at https://sourceforge.net/p/erframework.

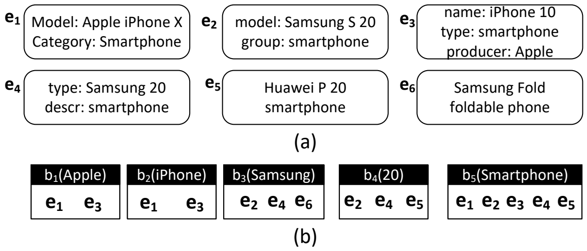


Figure 1: (a) The input entities, and (b) the redundancy-positive blocks produced by Token Blocking.

1 INTRODUCTION

Entity Resolution (ER) is the task of identifying entities that describe the same real-world object among different datasets [5]. ER constitutes a core data integration task with many applications that range from Data Cleaning in databases to Link Discovery in Semantic Web data [6, 9]. Despite the bulk of works on ER, it remains a challenging task [5]. 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.

To tame its high complexity, *Blocking* is typically used [4, 23]. Instead of considering all possible pairs of entities, it restricts the computational cost to entities that are similar. This is efficiently carried out by associating every entity with one or more signatures and clustering together 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 [17, 18]. As a result, parts of any attribute value in each entity can be used as signatures.

An example of schema-agnostic blocking is illustrated in Figure 1. The profiles in Figure 1(a) contain two duplicate pairs $\langle e_1, e_3 \rangle$ and $\langle e_2, e_4 \rangle$, which represent the same model of smartphone. The profiles are clustered together by using Token Blocking, i.e., a block is created for every token that appears in the values of each profile. The resulting blocks appear in Figure 1(b). ER examines all pairs inside each block and, thus, can detect all duplicate pairs, as they co-occur in at least one block.

The only drawback is that the resulting blocking involves high levels of redundancy: every entity is associated with multiple blocks, thus yielding numerous *redundant* and *superfluous comparisons* (i.e.,

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

pairs of entities) [2, 25]. The former are repeated across different blocks, while the latter involve pairs of entities that are non-matching. 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 [16, 20, 24].

A core approach to this end is *Meta-blocking* [19], which 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* 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* receives as input all weighted pairs and retains the ones that are more likely to be matching.

Figure 2 shows an example of learning-free meta-blocking based on the blocks in Figure 1(b). The blocking graph in Figure 2(a) is generated by creating one node for each entity profile; two nodes are connected by an edge if the corresponding profiles co-occur in at least one block. Then, each edge is weighted according to a weighting scheme – in our case, the number of blocks shared by the adjacent profiles. Finally, the blocking graph is pruned according to a pruning algorithm – in our case, for each node, we discard the edges with a weight lower than the average of its edges. The pruned blocking graph appears in Figure 2(b), with the dashed lines representing the superfluous comparisons. A block is then created for every retained edge, thus yielding a new block collection with much fewer comparisons, but the same matching ones as in Figure 1(c). Note that this is a schema-agnostic process, just like the original blocking method.

1.1 Supervised Meta-blocking

Supervised Meta-blocking models the restructuring of a set of blocks as a binary classification task [21]. Its goal is to train a model that learns to classify every comparison as *positive* (i.e., likely to be matching) and *negative* (i.e., unlikely to be matching). To this end, every pair is associated with a feature vector that comprises a combination of the most distinctive weighting schemes that are used by unsupervised meta-blocking. Thus, Supervised Meta-blocking considers more comprehensive evidence, outperforming the unsupervised approaches to a significant extent.

In more detail, the thorough experimental analysis in [21] performed an analytical feature selection that considered all combinations of 7 features to determine the one achieving the best balance between effectiveness and efficiency. The resulting vector comprises four features, yielding high time efficiency and classification accuracy.

As an example of Supervised Meta-blocking, consider Figure 3. Figure 3(a) presents the feature vectors generated for every distinct comparison in the blocks of Figure 1(b). Then, a binary classifier is trained with a sample of the labelled vectors and is used to predict whether the remaining ones are positive (1) or negative (0). The results appear in Figure 3(b). Only comparisons classified as positive are retained, yielding the new blocks in Figure 3(c).

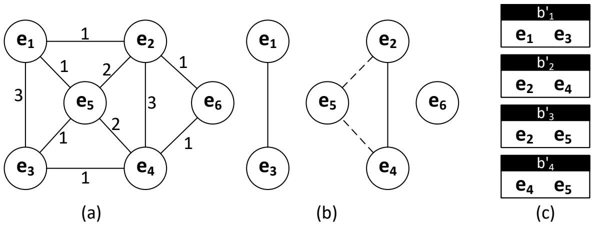


Figure 2: (a) The blocking graph corresponding to the blocks in Figure 1(b), using the number of common blocks as edge weights, and (b) a possible pruned blocking graph. The dashed lines indicate the superfluous comparisons that are retained after the pruning. (c) The new blocks.

Note, though, that ER suffers from intense class imbalance, since the vast majority of comparisons belongs to the negative class. To address it, *undersampling* is used to create a training set that is equally split between the two classes. Through extensive experiments, the number of labelled instances per class in the training set was set to 5% of the minority (positive) class in the ground-truth [21]. This size combined high effectiveness with high efficiency, through the learning of generic classification models. Additionally, the learned classification models were empirically verified to be robust with respect to the configuration of the classification algorithm: minor changes in the internal parameters of the algorithm yield minor changes in its overall performance.

Even though Supervised Meta-blocking outperforms its unsupervised counterpart to a significant extent, there is plenty of room for improving its classification accuracy, for reducing its overhead, i.e., running time, and for minimizing the size of the training set it requires, as we explain below.

1.2 Our Contributions

In this work, we go beyond Supervised Meta-blocking in the following ways:

(1) We generalize it from a binary classification task to a binary *probabilistic* classification process. The resulting probabilities are used as comparison weights, on top of which we apply new pruning algorithms that are incompatible with the original approach.
(2) Using the original features, we prove that the new pruning algorithms significantly outperform the existing ones through an extensive experimental study that involves 9 real-world datasets. We also specify the top performers among the weight- and cardinality-based algorithms.
(3) To further improve their performance, we use four new weighting schemes as features for Generalized Supervised Meta-blocking. We examine the performance of all 255 feature combinations over all 9 datasets for the top-performing algorithms. We identify the top-10 feature sets per algorithm in terms of effectiveness and then, we select the optimal one by considering their time efficiency, too, significantly reducing the overall run-time.

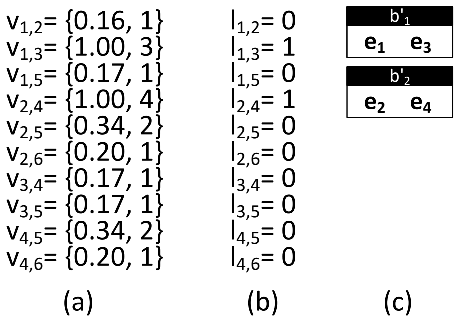


Figure 3: An example of applying Supervised Meta-blocking to the blocks in Figure 1(b). (a) Each comparison between entities e_i and e_j is represented by the feature vector $v_{i,j} = \{JS(e_i, e_j), CB(e_i, e_j)\}$, where $JS(e_i, e_j)$ stands for the Jaccard co-efficient of blocks associated with e_i and e_j and $CB(e_i, e_j)$ for the number of their common blocks. (b) A trained binary classifier is used to label every comparison between e_i and e_j as positive ($l_{i,j}$=1) or negative ($l_{i,j}$=0). (c) The output is formed by creating a new block for every positive comparison.

(4) For each algorithm, we examine how the size of the training set affects its performance. We experimentally demonstrate that it suffices to train our approaches over just 50 labelled instances, 25 from each class.
(5) We performn 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 under versatile settings.
(6) We have publicly released our data as well as an implementation of all pruning algorithms and weighting schemes in Java.[1]

The rest of the paper is organized as follows: Section 2 provides background knowledge on the task of Supervised Meta-blocking, Section 3 introduces the new pruning algorithms, and Section 4 discusses the weighing schemes that are used as features. The experimental analysis is presented in Section 5, the main works in the field are discussed in Section 6 and the paper concludes in Section 7 along with directions for future work.

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 [17]. 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., $\nexists\, e_i, e_j \in E_l : e_i \equiv e_j$.

In this context, we distinguish Entity Resolution into two tasks [4, 18]: (i) *Clean-Clean ER* or *Record Linkage* receives as input two clean entity collections, E_1 & E_2, and detects the set of 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, $D = \{(e_i, e_j) \subseteq E \times E : i \neq j \wedge e_i \equiv e_j\}$.

In both cases, the time complexity is quadratic with respect to the input, i.e., $O(|E_1| \times |E_2|)$ and $O(|E|^2)$, respectively, 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 [23].

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. Blocks of this type are generated by methods like Token Blocking, Suffix Arrays and Q-Grams Blocking and their variants [16, 24].

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 [22]. 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, $c_{i,j} \in C$, 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: $e_i \not\equiv e_j$.

Note that 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 [4, 17]. 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 the negative pairs are drastically reduced without any significant impact on the positive ones, i.e., $|P_{B'}| \approx |P_B|$ and $|N_{B'}| \ll |N_B|$.

2.1 Problem Definition

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 that returns a score proportional to the matching likelihood of $c_{i,j}$. The feature vectors for all pairs in C are fed to a *binary classifier*, which labels them as `positive` or `negative`, if their constituent entities are highly likely to match or not. We assess the performance of this process based on the following measures:

(1) $TP(C)$, the true positive pairs, involve matching entities and are correctly classified as `positive`.
(2) $FP(C)$, the false positive pairs, entail non-matching entities, but are classified as `positive`.
(3) $TN(C)$, the true negative pairs, entail non-matching entities and are correctly classified as `negative`.
(4) $FN(C)$, the false negative pairs, comprise matching entities, but are categorized as `negative`.

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

[1] See https://sourceforge.net/p/erframework for more details.

Supervised Meta-blocking is assessed with respect to the following measures:

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

All measures are defined in $[0, 1]$, with higher values indicating higher effectiveness.

In this context, the task of Supervised Meta-blocking is formally defined as follows [21]:

Definition 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 the trained classification model M to C.

2.1.1 Generalized Supervised Meta-blocking. This new task 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, which are called *valid pairs*, are further processed by a pruning algorithm. The ones retained after pruning give raise to the new block collection B', i.e., B' contains a new block per retained valid pair.

Hence, the performance evaluation of Generalized Supervised Meta-blocking relies on the following measures:

(1) $TP'(C)$, the probabilistic true positive pairs, involve matching entities that are assigned a probability $\geq$0.5 and are retained by the pruning algorithm.
(2) $FP'(C)$, the probabilistic false positive pairs, entail non-matching entities, that are assigned a probability $\geq$0.5 and are retained by the pruning algorithm.
(3) $TN'(C)$, the probabilistic true negative pairs, entail non-matching entities that are assigned a probability <0.5 and are discarded by the pruning algorithm.
(4) $FN'(C)$, the probabilistic false negative pairs, comprise matching entities, that are assigned 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:

Definition 2. *Given the candidate pairs C of block collection B, the labels L={*positive, negative*}, and a training set $T = \{\langle c_{i,j}, l_k \rangle : c_{i,j} \in C \wedge l_k \in L\}$, 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 time efficiency of Generalized Supervised Meta-blocking is assessed through its run-time, *RT*, which adds to that of Supervised Meta-blocking the time required to process the assigned probabilities by a pruning algorithm.

3 PRUNING ALGORITHMS

A supervised pruning algorithm operates as follows: given a specific set of features, it trains a probabilistic classifier on the available labelled instances. Then, it applies the trained classification model M to each candidate pair, estimating its classification probability. It it exceeds 0.5, it is compared with a threshold in order to determine 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:

(1) The *weight-based algorithms* determine the weight(s), above which a comparison is retained.
(2) 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. Below, we delve into the supervised algorithms of each category.

3.1 Weight-based pruning algorithms

This category includes the following four algorithms. None of them was considered in [21] - only WEP was approximated through the binary classification task in Definition 1.

Weigted Edge Pruning (WEP). Algorithm 1 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 (Lines 1-8). Then, it applies again the trained classifier to each pair and retains only those pairs with a probability higher than $\bar{p}$ (Lines 9-13).

Weighted Node Pruning (WNP). Algorithm 2 iterates twice over C, too. Yet, instead of a global average probability, it estimates a local average probability per entity. To this end, it keeps in memory two arrays: $\bar{p}[]$ with the sum of valid probabilities per entity (Line 1) and $counter[]$ with the number of valid candidates per entity (Line 2). These arrays are populated during the first iteration over C (Lines 3-9). The average probability per entity is then computed in Lines 10-11. Finally, WNP iterates over C and retains every comparison $c_{i,j}$ only if its estimated probability $p_{i,j}$ exceeds either of the related average probabilities (Line 15).

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, i.e., $p\bar{[i]} \leq$

Algorithm 1: Supervised Weighted Edge Pruning

```
   Input: Learned Model $M$, Candidate Pairs $C$
   Output: New Candidate Pairs $C'$
1  $\bar{p} \leftarrow 0$
2  counter = 0
3  foreach $c_{i,j} \in C$ do
4      $p_{i,j} \leftarrow M$.getProbability($c_{i,j}$)
5      if $0.5 \leq p_{i,j}$ then
6          $\bar{p}$ += $p_{i,j}$
7          counter += 1
8  $\bar{p} \leftarrow \bar{p}$ / counter
9  $C' \leftarrow \{\}$
10 foreach $c_{i,j} \in C$ do
11     $p_{i,j} \leftarrow M$.getProbability($c_{i,j}$)
12     if $\bar{p} \leq p_{i,j}$ then
13         $C' \leftarrow C' \cup \{c_{i,j}\}$
14 return $C'$
```

Algorithm 2: Supervised Weighted Node Pruning

```
   Input: Learned Model $M$, Candidate Pairs $C$
   Output: New Candidate Pairs $C'$
1  $\bar{p}[] \leftarrow \{\}$
2  $counter[] \leftarrow \{\}$
3  foreach $c_{i,j} \in C$ do
4      $p_{i,j} \leftarrow M$.getProbability($c_{i,j}$)
5      if $0.5 \leq p_{i,j}$ then
6          $\bar{p[i]}$ += $p_{i,j}$
7          $counter[i]$ += 1
8          $\bar{p[j]}$ += $p_{i,j}$
9          $counter[j]$ += 1
10 foreach $i \in |\bar{p}|$ do
11     $\bar{p[i]} \leftarrow \bar{p[i]}$ / $counter[i]$
12 $C' \leftarrow \{\}$
13 foreach $c_{i,j} \in C$ do
14     $p_{i,j} \leftarrow M$.getProbability($c_{i,j}$)
15     if $\bar{p[i]} \leq p_{i,j} \vee \bar{p[j]} \leq p_{i,j}$ then
16         $C' \leftarrow C' \cup \{c_{i,j}\}$
17 return $C'$
```

$p_{i,j} \wedge \bar{p[j]} \leq p_{i,j}$. This way, it applies a consistently deeper pruning than WNP.

BLAST. This algorithm is similar to WNP, but uses a fundamentally different pruning criterion. Instead of the average probability per entity, it relies on the maximum probability per entity e_i. Algorithm 3 stores these probabilities in the array $max[]$ (Line 1), which is populated during the first iteration over C (Lines 2-8). 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 (Line 12).

Algorithm 3: Supervised BLAST

```
   Input: Learned Model $M$, Candidate Pairs $C$, Pruning Ratio
          $r \in (0, 1]$
   Output: New Candidate Pairs $C'$
1  $max[] \leftarrow \{\}$
2  foreach $c_{i,j} \in C$ do
3      $p_{i,j} \leftarrow M$.getProbability($c_{i,j}$)
4      if $0.5 \leq p_{i,j}$ then
5          if $max[i] < p_{i,j}$ then
6              $max[i] = p_{i,j}$
7          if $max[j] < p_{i,j}$ then
8              $max[j] = p_{i,j}$
9  $C' \leftarrow \{\}$
10 foreach $c_{i,j} \in C$ do
11     $p_{i,j} \leftarrow M$.getProbability($c_{i,j}$)
12     if $0.5 \leq p_{i,j} \wedge r \cdot (max[i] + max[j]) \leq p_{i,j}$ then
13         $C' \leftarrow C' \cup \{c_{i,j}\}$
14 return $C'$
```

3.2 Cardinality-based pruning algorithms

This category includes the three algorithms described below. Only the first two were considered in [21].

Cardinality Edge Pruning (CEP). This algorithm retains the top-K weighted comparisons among the candidate pairs, where K is set to half the sum of block sizes in the original block collection B, i.e., $K = \sum_{b_i \in B} |b|/2$, where $|b|$ stands for the number of entities in block b [19]. Algorithm 4 essentially maintains a priority queue Q (Line 1), which sorts the comparisons in decreasing probability. Q is populated through a single iteration over C (Lines 3-9). Every valid candidate pair that exceeds the minimum probability min_p (Line 5), is pushed to the queue (Line 6). Whenever the size of the queue exceeds K, the lowest-weighted comparison is removed from the queue and min_p is updated accordingly (Lines 7-9). At the end of the iteration, the contents of Q correspond to the new set of candidates C'.

Algorithm 4: Supervised Cardinality Edge Pruning

```
   Input: Learned Model $M$, Candidate Pairs $C$, $K$
   Output: New Candidate Pairs $C'$
1  $Q \leftarrow \{\}$
2  $min_p \leftarrow 0$
3  foreach $c_{i,j} \in C$ do
4      $p_{i,j} \leftarrow M$.getProbability($c_{i,j}$)
5      if $0.5 \leq p_{i,j} \wedge min_p < p_{i,j}$ then
6          $Q$.push($c_{i,j}$)
7          if $K < |Q|$ then
8              $c_{k,l} \leftarrow Q$.pop()
9              $min_p \leftarrow p_{k,l}$
10 return $Q$
```

Cardinality Node Pruning (CNP). Algorithm 5 adapts CEP to a local operation, maintaining an array $Q[]$ with a separate priority queue per entity (Line 1). The maximum size of each queue depends on the characteristics of the original block collection, as it amounts to the average number of blocks per entity: $k = max(1, \sum_{b \in B} |b|/(|E_1| + |E_2|))$ [19]. During the first iteration over C, CNP populates the priority queue of every entity following the same procedure as CEP (Lines 3-15); if the probability of the current candidate pair exceeds the minimum probability of one of the relevant queues (Lines 6 and 11), the pair is pushed into the queue (Lines 7 and 12). Whenever the size of a queue exceeds k (Lines 8 and 13), the least-weighted comparison is removed (Lines 9 and 14) and the corresponding threshold is updated accordingly (Lines 10 and 15). CNP involves a second iteration over C (Lines 17-21), which retains a candidate pair $c_{i,j}$ if its contained in the priority queue of e_i or e_j (Line 20).

Algorithm 5: Supervised Cardinality Node Pruning

```
   Input: Learned Model $M$, Candidate Pairs $C$, $k$
   Output: New Candidate Pairs $C'$
1  $Q[] \leftarrow \{\}$
2  $min_p[] \leftarrow \{\}$
3  foreach $c_{i,j} \in C$ do
4      $p_{i,j} \leftarrow M$.getProbability($c_{i,j}$)
5      if $0.5 \leq p_{i,j}$ then
6          if $min_p[i] < p_{i,j}$ then
7              $Q[i]$.push($c_{i,j}$)
8              if $k < |Q[i]|$ then
9                  $c_{l,m} \leftarrow Q[i]$.pop()
10                 $min_p[i] \leftarrow p_{l,m}$
11         if $min_p[j] < p_{i,j}$ then
12             $Q[j]$.push($c_{i,j}$)
13             if $k < |Q[j]|$ then
14                 $c_{l,m} \leftarrow Q[j]$.pop()
15                 $min_p[j] \leftarrow p_{l,m}$
16 $C' \leftarrow \{\}$
17 foreach $c_{i,j} \in C$ do
18     $p_{i,j} \leftarrow M$.getProbability($c_{i,j}$)
19     if $0.5 \leq p_{i,j}$ then
20         if $Q[i]$.contains($c_{i,j}$) $\vee$ $Q[j]$.contains($c_{i,j}$) then
21             $C' \leftarrow C' \cup \{c_{i,j}\}$
22 return $C'$
```

Reciprocal Cardinality Node Pruning (RCNP). This algorithm adapts CNP so that it performs a consistently deeper pruning, requiring that every retained comparison is contained in the priority queue of both constituent entities. That is, the condition of Line 20 in Algorithm 5 changes into a conjunction: $Q[i]$.contains($c_{i,j}$) $\wedge$ $Q[j]$.contains($c_{i,j}$).

4 WEIGHTING SCHEMES

The goal of *weighting schemes* is to infer the matching likelihood of candidate pairs from their co-occurrence patterns in the input blocks [19]. All schemes are schema-agnostic, being generic enough to apply to any redundancy-positive block collection. In [21], four weighting schemes formed the optimal feature vector in the sense that it achieves the best balance between effectiveness and time efficiency:

(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 of them. More formally:
$$CF - IBF(c_{i,j}) = |B_i \cap B_j| \cdot \log \frac{|B|}{|B_i|} \cdot \log \frac{|B|}{|B_j|}.$$
(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. This idea is captured by the following sum:
$$RACCB(c_{i,j}) = \sum_{b \in B_i \cap B_j} \frac{1}{||b||},$$
where $||b||$ denotes the total number of candidate pairs in block b (including the redundant ones).
(3) *Jaccard Scheme* (JS). It expresses the portion of blocks shared by a pair of candidates:
$$JS(c_{i,j}) = \frac{|B_i \cap B_j|}{|B_i| + |B_j| - |B_i \cap B_j|}$$
This captures the core characteristic of redundancy-positive block collections that the more blocks two entities share, the more likely they are to match.
(4) *Local Candidate Pairs* (LCP). It measures the number of candidates for a particular entity. More formally:
$$LCP(e_i) = |\{e_j : i \neq j \wedge |B_i \cap B_j| > 0\}|.$$
The rationale is that the less candidate matches 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.

The last feature applies to an individual entity. Thus, the feature vector of c_{ij} includes both LCP(e_i) and LCP(e_j) [21].

In this work, we aim to enhance the effectiveness of the resulting feature vector. To this end, we additionally consider the following new weighting schemes [1]:

(1) *Enhanced Jaccard Scheme* (EJS). Similar to TF-IDF, it enhances JS with the inverse frequency of an entity's candidates in the set of all candidate pairs:
$$EJS(c_{i,j}) = JS(c_{i,j}) \cdot \log \frac{||B||}{||e_i||} \cdot \log \frac{||B||}{||e_j||},$$
where $||B|| = \sum_{b \in B} ||b||$ and $||e_l|| = \sum_{b \in B_l} ||b||$.
(2) *Weighted Jaccard Scheme* (WJS). Its goal is to alter JS so that it considers the size of the blocks containing every entity, promoting the smallest (and most distinctive) ones in terms of the

Name	$\|E_1\|$	$\|E_2\|$	$\|D$	$\|C\|$
AbtBuy	1.1k	1.1k	1.1k	36.7k
DblpAcm	2.6k	2.3k	2.2k	46.2k
ScholarDblp	2.5k	61.3k	2.3k	83.3k
AmazonGP	1.4k	3.3k	1.3k	84.4k
ImdbTmdb	5.1k	6.0k	1.9k	109.4k
ImdbTvdb	5.1k	7.8k	1.1k	119.1k
TmdbTvdb	6.0k	7.8k	1.1k	198.6k
Movies	27.6K	23.1k	22.8k	26.0M
WalmartAmazon	2.5K	22.1k	1.1k	27.4M

Table 1: Technical characteristics of the real-world Clean-Clean ER datasets used in the experiments. $|E_x|$ stands for the number of entities in a constituent dataset, $|D|$ for the number of duplicate pairs, and $|C|$ for the number of distinct candidate pairs generated in the corresponding block collection. The datasets are ordered in decreasing $|C|$.

total number of candidates. Thus, it multiplies every block in the Jaccard coefficient with its inverse size:

$$WJS(c_{i,j}) = \frac{\sum_{b \in B_i \cap B_j} \frac{1}{||b||}}{\sum_{b \in B_i} \frac{1}{||b||} + \sum_{b \in B_j} \frac{1}{||b||} - \sum_{b \in B_i \cap B_j} \frac{1}{||b||}}.$$

WJS can be seen as normalizing RACCB.

(3) *Reciprocal Sizes Scheme* (RS). It is similar to ARCS, but considers the number of entities in common blocks, rather than the number of candidate pairs:

$$RS(c_{i,j}) = \sum_{b \in B_i \cap B_j} \frac{1}{|b|}.$$

(4) *Normalized Reciprocal Sizes Scheme* (NRS). It normalizes RS, multiplying every block in the Jaccard coefficient with its inverse size:

$$NRS(c_{i,j}) = \frac{\sum_{b \in B_i \cap B_j} \frac{1}{|b|}}{\sum_{b \in B_i} \frac{1}{|b|} + \sum_{b \in B_j} \frac{1}{|b|} - \sum_{b \in B_i \cap B_j} \frac{1}{|b|}}.$$

5 EXPERIMENTAL EVALUATION

5.1 Experimental setup

Hardware and Software—All the experiments are 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 [11] to perform blocking and features generation.

Unless stated otherwise, we perform machine learning analysis using Python 3.7. In particular, all classification models are implemented through the Support Vector Classification (SVC) model of scikit-learn[2]. 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. Note that we have performed all experiments with logistic regression, too, obtaining almost identical results. Due to space limitations, we do not report them in this paper.

Datasets—Table 1 lists the 9 real-world datasets employed in our experiments. 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 [14]. `DblpAcm` matches scientific articles extracted from dblp.org and dl.acm.org [14]. `ScholarDblp` matches scientific articles extracted from scholar.google.com and dblp.org [14]. `ImdbTmdb`, `ImdbTvdb` and `TmdbTvdb` match movies and TV series extracted from IMDB, TheMovieDB and TheTVDB [15], as suggested by their names. `Movies` matches information about films that are extracted from imdb.com and dbpedia.org [17]. Finally, `WalmartAmazon` matches products from Walmart.com and Amazon.com [7].

Dataset	Recall	Precision	$F1$
AbtBuy	0.948	$2.78\cdot10^{-2}$	$5.40\cdot10^{-2}$
DblpAcm	0.999	$4.81\cdot10^{-2}$	$9.18\cdot10^{-2}$
ScholarDblp	0.998	$2.80\cdot10^{-3}$	$5.58\cdot10^{-3}$
AmazonGP	0.840	$1.29\cdot10^{-2}$	$2.54\cdot10^{-2}$
ImdbTmdb	0.988	$1.78\cdot10^{-2}$	$3.50\cdot10^{-2}$
ImdbTvdb	0.985	$8.90\cdot10^{-3}$	$1.76\cdot10^{-2}$
TmdbTvdb	0.989	$5.50\cdot10^{-3}$	$1.09\cdot10^{-2}$
Movies	0.976	$8.59\cdot10^{-4}$	$1.72\cdot10^{-3}$
WalmartAmazon	1.000	$4.22\cdot10^{-5}$	$8.44\cdot10^{-5}$

Table 2: Performance of the block collections that are given as input to the supervised meta-blocking methods.

Blocking—For each dataset, the initial block collection is extracted through Token Blocking,the only parameter-free redundancy-positive blocking method [23]. The original block collection is then processed by Block Purging [17], which discards all the blocks that contain more than half of the entity profiles in the collection in a parameter-free way. These blocks correspond to highly frequent signatures (e.g. stop-words) that provide no distinguishing information. Subsequently, we apply Block Filtering [22], removing each entity e_i from the largest 20% blocks in which it appears. Finally, the features described in Section 4 are generated for each pair of candidates.

Table 2 reports the performance of the resulting block collections. 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 raises precision and F1 by orders of magnitude, at a small cost in recall.*

To apply Generalized Supervised Meta-blocking to these block collections, we take special care to avoid the bias derived from the randomly selected pairs to be labelled and used for training. To this end, 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, equally split between the positive and the negative class. Due to space limitations, in the following we mostly report the average performance of every approach across all 9 block collections.

5.2 Pruning Algorithm Selection

The goal of this experiment is to discover which are the best-performing weighted-based and cardinality-based pruning algorithms for Generalized Supervised Meta-blocking among those discussed in Section 3. As baseline methods, we employ the pruning algorithms proposed in [21]: the binary classifier that approximates

[2] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

WEP for weight-based algorithms, denoted by **BCl** in the following, 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 [21] as optimal; that is, every candidate pair $c_{i,j}$ is represented by the following 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 2 are reported in Figures 4 and 5, respectively.

Among the the weight-based algorithms, we observe that the new pruning algorithms trade sligthy 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: it 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. Thus, RWNP achieves the highest average F1 (0.374), followed in close distance by WEP (0.366). However, their aggressive pruning results in very low recall (≪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, resp.

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. Given that the weight-based pruning algorithms are crafted for applications that promote recall at the cost of slightly lower precision [19, 22], we select BLAST as the best one in this category, even though it does not achieve the highest F1, on average.

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%, 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. Given that cardinality-based pruning algorithms are crafted for applications that promote precision at the cost of slightly lower recall [19, 22], RCNP constitutes the best choice for this category.

ID	Feature set	Recall	Precision	$F1$
72	{CF-IBF, RACCB, JS, RS}	.8816	.1932	.2892
74	{CF-IBF, RACCB, JS, NRS}	.8816	.1932	.2892
75	{CF-IBF, RACCB, JS, WJS}	.8816	.1932	.2892
78	{CF-IBF, RACCB, RS, NRS}	.8816	.1932	.2892
79	{CF-IBF, RACCB, RS, WJS}	.8816	.1932	.2892
82	{CF-IBF, RACCB, NRS, WJS}	.8816	.1932	.2892
86	{CF-IBF, JS, RS, WJS}	.8816	.1932	.2892
89	{CF-IBF, JS, NRS, WJS}	.8816	.1932	.2892
96	{CF-IBF, RS, NRS, WJS}	.8816	.1932	.2892
190	{CF-IBF, RACCB, JS, RS, NRS, WJS}	.8816	.1932	.2892

Table 3: The 10 feature sets that achieve the highest F1 with BLAST.

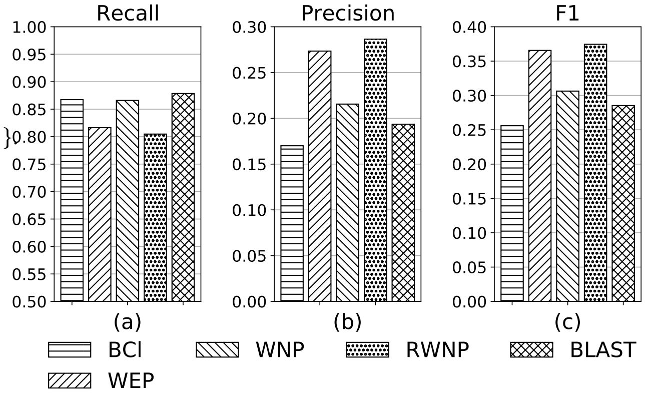


Figure 4: The average performance of all weight-based pruning algorithms over the block collections of Table 2.

Note that the F1 of BLAST and RCNP is significantly higher than the original ones in Table 2. They are still far from a perferct F1, but (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.

5.3 Feature selection

The goal of this experiment is to fine-tune the selected algorithms, namely BLAST and RCNP, by identifying the feature sets that optimize their performance in terms of both effectiveness and time-efficiency. To this end, we adopted a brute force approach, trying all the possible combinations of the eight features discussed in Section 4. Fixing again the training set size to 500 random samples, equally split between positive and negative instances, the top-10 feature vectors with respect to F1 for BLAST and RCNP are reported in Table 3 and Table 4, respectively.

We observe that for each algorithm, all feature sets achieve practically identical performance, on average. For BLAST, we obtain recall=0.882, precision=0.193 and F1=0.289, while for RCNP, we obtain recall=0.850, precision=0.248 and F1=0.353. Compared to their average performance with the original feature vector proposed in [21], we observe that the average recall and precision raise by ~0.5%, while the average F1 increases by ~1.5%. This means that the *effectiveness* of both algorithms is quite robust with respect to the underlying feature set. Therefore, we select the best one 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 1: movies and WalmartAmazon. We repeated every experiment 10 times and took the mean time.

The resulting running times appear in Figures 6 and 8 for BLAST and RCNP, respectively. In the former figure, we observe that the feature set **78** is consistently the fastest one for BLAST, exhibiting a

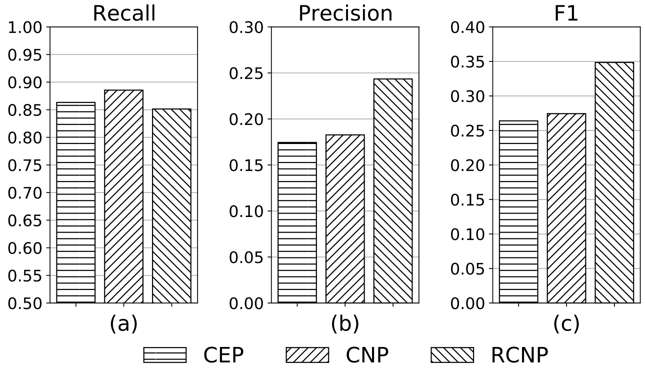


Figure 5: The average performance of all cardinality-based pruning algorithms over the block collections of Table 2.

clear lead. Compared to the second fastest feature sets over `movies` (75) and `WalmartAmazon` (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 `WalmartAmazon` (239), it reduces the average run-time by 3.3% and 4.8%, respectively.

The actual features comprising the selected feature sets, 78 and 187, appear in Tables 3 and 4, respectively. In the former, we observe that BLAST models each candidate pair as the 4-dimensional feature vector:

$$\{CF-IBF, RACCB, RS, NRS\}. \qquad (1)$$

In the latter table, we observe that RCNP represents every candidate pair with the 5-dimensional feature vector:

$$\{CF-IBF, RACCB, JS, LCP, WJS\}. \qquad (2)$$

It is worth noting that BLAST works better with a smaller feature set than RCNP. In general, all top-10 feature sets selected for the former algorithm are much simpler than those selected for the latter. Most importantly, BLAST avoids the feature *LCP*, whose calculation is quite expensive (it iterates over all blocks containing an entity in order to gather its distinct candidates). Therefore, it is no surprise that BLAST is 2 to 3 times faster than RCNP.

5.3.1 *Comparison with Supervised Meta-blocking [21].*

Based on the selected feature sets, we compare the performance of Generalized Supervised Meta-blocking with Supervised Meta-blocking. The former is represented by BLAST and RCNP in combination with the features in Formulas 1 and 2, respectively, while for the latter, we use the feature set proposed in [21], $\{CF-IBF, RACCB, JS, LCP\}$, in combination with BCl and CNP. All algorithms were trained over a randomly selected set of 500 labelled instances, 250 from each class, and were applied to all datasets in Figure 1. Their average performance with respect to effectiveness is presented in Figure 7.

ID	Feature Set	Recall	Precision	$F1$
184	{CF-IBF, RACCB, JS, LCP, RS}	.8489	.2463	.3527
187	{CF-IBF, RACCB, JS, LCP, WJS}	.8490	.2464	.3526
193	{CF-IBF, RACCB, LCP, RS, NRS}	.8490	.2463	.3526
200	{CF-IBF, JS, LCP, RS, NRS}	.8488	.2474	.3526
227	{CF-IBF, RACCB, JS, LCP, RS, NRS}	.8493	.2473	.3537
228	{CF-IBF, RACCB, JS, LCP, RS, WJS}	.8494	.2473	.3537
231	{CF-IBF, RACCB, JS, LCP, NRS, WJS}	.8496	.2473	.3537
235	{CF-IBF, RACCB, LCP, RS, NRS, WJS}	.8496	.2473	.3536
239	{CF-IBF, JS, LCP, RS, NRS, WJS}	.8494	.2473	.3534
250	{CF-IBF, RACCB, JS, LCP, RS, NRS, WJS}	.8502	.2479	.3542

Table 4: The 10 feature sets that achieve the highest F1 when applied to RCNP.

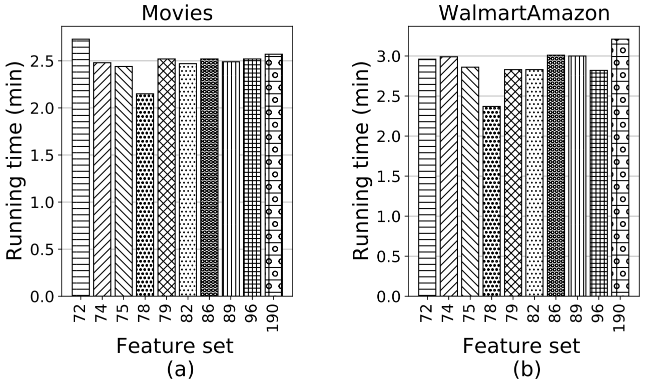


Figure 6: Running time of top-10 features sets when applied to BLAST.

Among the weight-based algorithms, BLAST achieves higher recall than BCl, by 1.6% on average. Thus, BLAST is more suitable for recall-intensive applications. Most importantly, it outperforms BCl with respect to the other measures, too: the average precision raises by 13.6% and the average F1 by 13%. This indicates that BLAST is much more accurate in the classification of the candidate pairs.

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 for precision-intensive applications.

Regarding time efficiency, Figure 9 reports the running times of these algorithms on the largest datasets, i.e., `Movies` and `WalmartAmazon`. 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

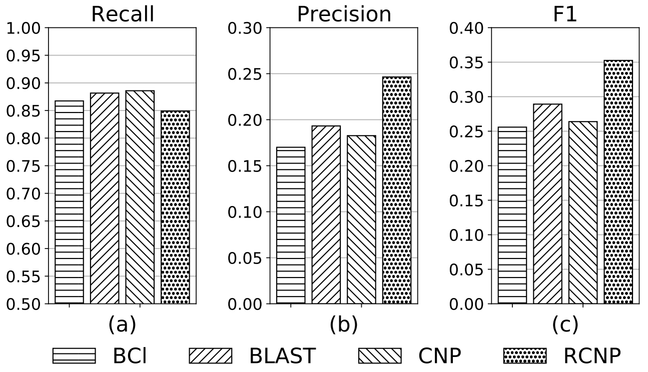


Figure 7: Comparison of the best algorithms for Supervised (BCl, CNP) and Generalized Supervised Meta-blocking (BLAST, RCNP).

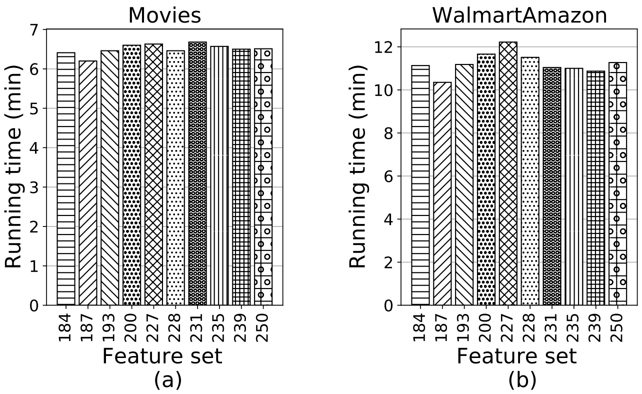


Figure 8: Running time of top-10 features sets when applied to RCNP.

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 `WalmartAmazon`.

We can conclude, therefore, that Generalized Supervised Meta-blocking conveys significant improvements with respect to Supervised Meta-blocking.

5.4 The effect of training set size

To examine whether active learning is necessary for BLAST and RCNP, we perform an experiment that explores how their performance changes when varying the training set size. We used the features sets specified in Formulas 1 and 2 and varied the number of labelled instances from 50 to 500 with a step of 50.[3] Figures 10 and 11 report the results in terms of recall, precision and F1, on average across all datasets, for BLAST and RCNP, respectively.

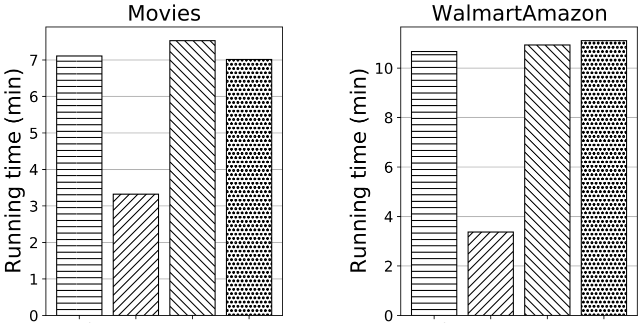


Figure 9: Comparison of the best algorithms for Supervised (BCl, CNP) and Generalized Supervised Meta-blocking (BLAST, RCNP).

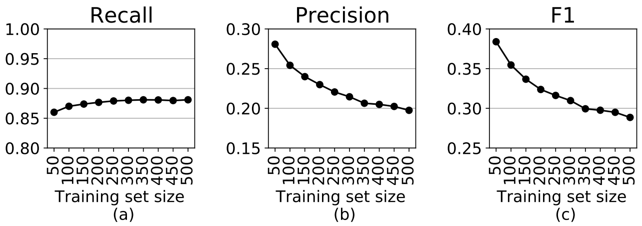


Figure 10: The effect of the training set size on BLAST.

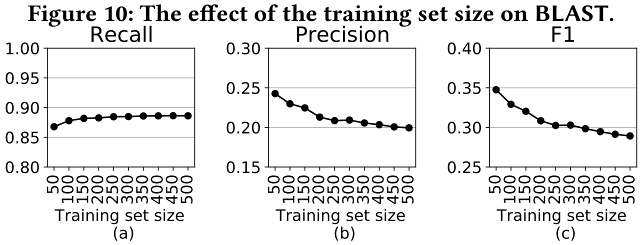


Figure 11: The effect of the training set size on RCNP.

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. Given that the initial levels of recall are 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.* This renders active learning techniques unnecessary, given that their cost is not compensated, when such a small training set is required.

5.4.1 Comparison with Supervised Meta-blocking [21]. Based on the above experimental results, we now compare the final algorithms of Generalized Supervised Meta-blocking with their baseline counterparts from [21].

Regarding the weight-based algorithms, Table 5 reports a full comparison between BLAST and BCl. The former uses the features in Formula 1 along with 50 labelled instances, while the latter combines the best feature set from [21], $\{CF-IBF, RACCB, JS, LCP\}$, with two different training sets: BCl_1 uses the same 50 labelled instances that are used by BLAST, while BCl_2 uses the training set specified in [21] (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 both baseline methods with respect to all effectiveness measures: compared to BCl_1 (BCl_2), the average recall increases by 10.4% (7.1%), the average precision by 5.1% (5.0%), and the average F1 by 10.8% (9.9%) – as expected, BCl_2 performs better than BCl_1, due to its larger training sets. Notice that BLAST emphasizes recall, as it consistently achieves the highest one across all datasets, except `AmazonGP` – the most difficult dataset in terms of recall, as verified in Table 2. Yet, BLAST excels in precision and F1, too, except for `ScholarDblp`, where BCl_1 and BCl_2 trade lower recall for higher precision and F1. In terms

[3] Note that we tried to use BLOSS [3] as a baseline method, but we couldn't reproduce its performance, since our implementation of the algorithm exclusively selected non-matching candidate pairs – instead of a balanced training set. We contacted the authors, but they were not able to provide us with their own implementation. Nevertheless, our experimental results demonstrate that active learning is not necessary for our approaches, given that they achieve high performance with just 50 labelled instances.

	AbtBuy	DblpAcm	ScholarDblp	AmazonGP	ImdbTmdb	ImdbTvdb	TmdbTvdb	Movies	WalmartAmazon
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

Re	0.8197	0.9523	0.8856	0.7145	0.7834	0.6981	0.8169	0.7591	0.5841
Pr	0.2018	0.6061	0.4352	0.1254	0.5430	0.2524	0.2366	0.0029	0.0001
$F1$	0.3233	0.7372	0.5525	0.2041	0.6228	0.3627	0.3091	0.0058	0.0001
RT	5.60	5.46	10.71	5.99	5.49	5.79	6.22	82.41	106.01

(b) BCl$_1$

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	1328.81	276.19

(c) BCl$_2$

Table 5: The performance of the main weight-based algorithms across all datasets. BLAST combines the features in Formula 1 with 50 randomly selected labelled instances, equally split between the two classes. BCl$_1$ couples the features in [21] with the same 50 instances, while BCl$_2$ uses both the features and the training set specified in [21]. ***RT*** **reports the average running times (over 10 repetitions) in seconds.**

of run-time, BLAST is slower than BCl$_1$ by 10.8%, on average. The reason is that the small training sets yield simple and, thus, efficient binary classification models that iterate once over all candidate pairs, unlike BLAST, which iterates twice over them. Compared to BCl$_2$, BLAST is 3.3 times faster, on average across all datasets, because the large training sets learn complex binary classifiers with a time consuming processing.

Similar patterns are observed in the case of cardinality-based algorithms in Table 7, where RCNP with the features in Formula 2 and 50 labelled instances is compared with CNP, which uses the best feature set from [21] and two different training sets: the same 50 labelled instances as RCNP (CNP$_1$) and the training set specified in [21] (CNP$_2$).

In more detail, RCNP outperforms both baseline methods for all effectiveness measures. Compared to CNP$_1$ (CNP$_2$), RCNP raises the average recall by 12.6% (9.2%), the average precision by 14.7% (16.4%) and the average F1 by 16.8% (18.3%). Most importantly, it achieves the highest precision in all datasets, except for AbtBuy and ImdbTmdb. The same applies to F1, too, indicating that recall is not excessively sacrificed in favor of precision. Instead, RCNP is typically more accurate in classifying the positive candidate pairs, since it yields the maximum recall in half the datasets. In terms of run-time, RCNP is slower than CNP$_1$ by 6.1%, on average, as the latter employs less features, learning simpler and faster classification models than the former when using the same labelled instances. CNP$_2$ employs a much larger training set, yielding more complicated and time-consuming classifiers than RCNP. As a result, the latter is 3 times faster than the former, on average, across all datasets.

Overall, Generalized Supervised Meta-blocking outperforms Supervised Meta-blocking to a significant extent.

5.5 Scalability Analysis

The goal of this experiment is to assess the scalability of our approaches as the number of candidate pairs, $|C|$, increases and to verify their robustness under versatile settings: instead of real-world Clean-Clean ER datasets, we now consider the synthetic Dirty ER datasets, and instead of SVC, we train our models using Logistic Regression – Weka's default implementation in Java (https://sourceforge.net/projects/weka).

Name	|E|	|D	|C|	Recall	Precision	F1
D_{10k}	10k	8.7k	$2.69{\cdot}10^{7}$	0.999	$3.23{\cdot}10^{-4}$	$6.47{\cdot}10^{-4}$
D_{50k}	50k	43.1k	$6.73{\cdot}10^{8}$	0.999	$6.40{\cdot}10^{-5}$	$1.28{\cdot}10^{-4}$
D_{100k}	100k	85.5k	$2.69{\cdot}10^{9}$	0.999	$3.17{\cdot}10^{-5}$	$6.34{\cdot}10^{-5}$
D_{200k}	200k	172.4k	$1.08{\cdot}10^{10}$	1.000	$1.60{\cdot}10^{-5}$	$3.19{\cdot}10^{-5}$
D_{300k}	300k	257.0k	$2.43{\cdot}10^{10}$	0.999	$1.06{\cdot}10^{-5}$	$2.12{\cdot}10^{-5}$

Table 6: Technical characteristics of the synthetic Dirty ER datasets used in the scalability analysis.

The characteristics of the datasets, which are widely used in the literature [4, 24], appear in Table 6. To extract a large block collection from every dataset, we apply Token Blocking. We observe that in all cases, the recall is almost perfect, but precision and F1 are extremely low.

On these block collections, we applied four methods: BCl and CNP with the features and the training set size specified in [21] as well as BLAST and RCNP with the features of Formulas 1 and 2, respectively, trained over 50 labelled instances (25 positive and 25 negative ones). In each dataset, we trained every algorithm over the same 3 random training sets and considered the average performance.

The effectiveness of the weight- and cardinality-based algorithms over all datasets appear in Figures 12(a) and 12(b), resp. We observe that 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. Similarly, RCNP outperforms CNP to a signficant extent: on average, it reduces recall by 7.9%, but maintains it to very high levels, except D_{200K}, where it drops to 0.77; yet, precision raises by 2.8 times and F1 by 2.3 times. These results verify the strength of our approaches, which require orders of magnitude less labelled instances than the ones in [21].

Most importantly, our approaches scale better to large datasets, as demonstrated in Figure 13, which reports the speedup across the four larger datasets. Given two sets of candidate pairs, $|C_1|$ and $|C_2|$, such that $|C_1| < |C_2|$, this 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 over 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, respe., 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, our approaches scale much better to large datasets in all respects than the approaches in [21].

6 RELATED WORK

The unsupervised pruning algorithms WEP, WNP, CEP, and CNP were introduced in [19]. WNP and CNP were redefined in [22] so that they do not produce block collections with redundant comparisons. Unsupervised Reciprocal WNP and Reciprocal CNP were coined in [22], while unsupervised BLAST was proposed in [25].

Over the years, more unsupervised pruning algorithms have been proposed in the literature. [27] 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 [10] 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 [8]; for example, Low Entity Co-occurrence Pruning removes from every block a specific portion of the entities with the lowest average weights. 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 the outcomes of Matching. A dynamic approach that leverages Meta-blocking to make the most of the feedback of Matching is *pBlocking* [13]. 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 the next set of candidate pairs that will be processed by Matching. This process is iteratively applied until convergence. *BEER* [12] is an open-source tool that implements pBlocking.

The work closest to ours is BLOSS [3]. It introduces an active learning approach that reduces significantly the size of the labelled set required by Supervised Meta-blocking. Initially, it partitions the unlabelled candidate pairs into similarity levels based on CF-IBF. Then, it applies rule-based active sampling inside every level in order to select the unlabelled pairs with the lowest commonalities with the already labelled ones so as to maximize the captured information. In the final step, BLOSS cleans the labelled sample from non-matching outliers with high Jaccard weight.

7 CONCLUSIONS

We have presented Generalized Supervised Meta-blocking, which casts Meta-blocking as a probabilistic binary classification task and weights all candidate pairs in a block collection with the probabilities produced by the trained classifier. These weights are processed by a pruning algorithm that can be: (i) weight-based, determining the minimum weight of retained pairs in a way that promotes recall, or (ii) cardinality-based, determining the maximum number of retained pairs in a way that promotes precision. Through a thorough experimental study over 9 established, real-world datasets, we verified that BLAST and RCNP constitute the best weight- and cardinality-based pruning algorithms, respectively. We also demonstrated that four new weighting schemes give rise to feature sets that outperform the one determined in [21] as optimal. Finally, we showed that a very small, balanced training set with just 50 labelled instances suffices for consistently achieving high effectiveness, high time efficiency and high scalability.

In the future, we plan to leverage Generalized Supervised Meta-blocking as a means for optimizing the performance of Progressive Entity Resolution [26].

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

[5] 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.

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

[7] 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.

[8] 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.

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

[10] 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.

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

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

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

[14] 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.

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

[16] 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.

[17] 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.

[18] George Papadakis, Ekaterini Ioannou, Emanouil Thanos, and Themis Palpanas. 2021. *The Four Generations of Entity Resolution.* Morgan & Claypool Publishers.

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

[20] George Papadakis and Wolfgang Nejdl. 2011. Efficient entity resolution methods for heterogeneous information spaces. In *ICDE Phd Workshop*. 304–307.

	AbtBuy	DblpAcm	ScholarDblp	AmazonGP	ImdbTmdb	ImdbTvdb	TmdbTvdb	Movies	WalmartAmazon
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
*F*1	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

(a) RCNP

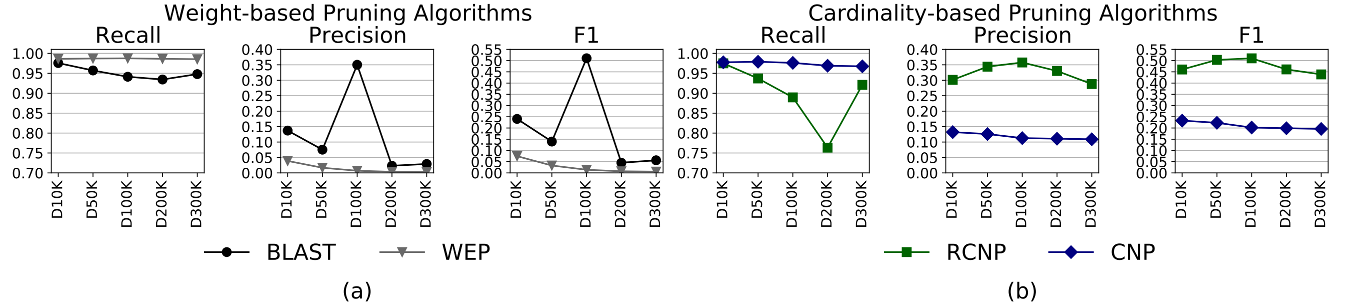
	AbtBuy	DblpAcm	ScholarDblp	AmazonGP	ImdbTmdb	ImdbTvdb	TmdbTvdb	Movies	WalmartAmazon
Re	0.8348	0.9590	0.9272	0.7662	0.8230	0.7402	0.8969	0.7020	0.2879
Pr	0.1877	0.5946	0.3218	0.0900	0.3512	0.2077	0.1403	0.0104	0.0001
*F*1	0.3055	0.7302	0.4546	0.1587	0.4622	0.3150	0.2316	0.0205	0.0002
RT	5.93	5.74	11.32	6.33	5.86	6.09	6.87	121.85	107.82

(b) CNP$_1$

	AbtBuy	DblpAcm	ScholarDblp	AmazonGP	ImdbTmdb	ImdbTvdb	TmdbTvdb	Movies	WalmartAmazon
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
*F*1	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	1351.54	365.03

(c) CNP$_2$

Table 7: The performance of the main cardinality-based algorithms across all datasets. RCNP combines the features in Formula 2 with 50 randomly selected labelled instances, equally split between the two classes. CNP$_1$ couples the features in [21] with the same 50 instances, while CNP$_2$ uses both the features and the training set specified in [21]. *RT* reports the average running times (over 10 repetitions) in seconds.



(a)

(b)

Figure 12: Scalability analysis over the datasets in Table 6. (a) The effectiveness measures of BCl, which uses the features and the training set specified in [21], and BLAST, which combines the features in Formula 1 with 50 randomly selected labelled instances, equally split between the two classes. (b) The effectiveness measures of CNP, which uses the features and the training set specified in [21], and RCNP, which combines the features in Formula 2 with 50 randomly selected labelled instances, equally split between the two classes.

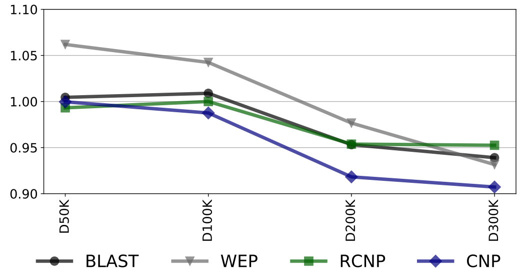


Figure 13: The speedup of the algorithms used in the scalability analysis of Figure 12.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] Giovanni Simonini, George Papadakis, Themis Palpanas, and Sonia Bergamaschi. 2019. Schema-Agnostic Progressive Entity Resolution. *TKDE* 31, 6 (2019), 1208–1221.

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