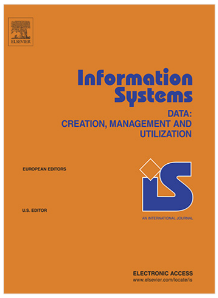
Accepted Manuscript



Scaling entity resolution: A loosely schema-aware approach

Giovanni Simonini, Luca Gagliardelli, Sonia Bergamaschi,
H.V. Jagadish

PII: S0306-4379(18)30408-3
DOI: https://doi.org/10.1016/j.is.2019.03.006
Reference: IS 1380

To appear in: *Information Systems*

Received date : 3 August 2018
Revised date : 16 February 2019
Accepted date : 17 March 2019

Please cite this article as: G. Simonini, L. Gagliardelli, S. Bergamaschi et al., Scaling entity resolution: A loosely schema-aware approach, *Information Systems* (2019), https://doi.org/10.1016/j.is.2019.03.006

Scaling Entity Resolution: A Loosely Schema-aware Approach

Giovanni Simonini[a,1], Luca Gagliardelli[b], Sonia Bergamaschi[b], H. V. Jagadish[c]

[a] *MIT CSAIL*
[b] *Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Italy*
[c] *University of Michigan, Ann Arbor, USA*

Abstract

In *big data* sources, real-world entities are typically represented with a variety of schemata and formats (e.g., relational records, JSON objects, etc.). Different *profiles* (i.e., representations) of an entity often contain redundant and/or inconsistent information. Thus identifying which profiles refer to the same entity is a fundamental task (called Entity Resolution) to unleash the value of big data. The naïve all-pairs comparison solution is impractical on large data, hence *blocking* methods are employed to partition a profile collection into (possibly overlapping) blocks and limit the comparisons to profiles that appear in the same block together. Meta-blocking is the task of restructuring a block collection, removing superfluous comparisons. Existing meta-blocking approaches rely exclusively on schema-agnostic features, under the assumption that handling the schema variety of big data does not pay-off for such a task.

In this paper, we demonstrate how "loose" schema information (i.e., statistics collected directly from the data) can be exploited to enhance the quality of the blocks in a holistic *loosely schema-aware* (meta-)blocking approach that can be used to speed up your favorite Entity Resolution algorithm. We call it *Blast* (Blocking with Loosely-Aware Schema Techniques). We show how *Blast* can automatically extract the loose schema information by adopting an LSH-based step for efficiently handling volume and schema heterogeneity of the data. Furthermore, we introduce a novel meta-blocking algorithm that can be employed to efficiently execute *Blast* on MapReduce-like systems (such as Apache Spark). Finally, we experimentally demonstrate, on real-world datasets, how *Blast* outperforms the state-of-the-art (meta-)blocking approaches.

Keywords: Entity Resolution, Meta-blocking, Big Data Integration, Data Cleaning, Apache Spark
2010 MSC: 00-01, 99-00

Email address: giovanni@csail.mit.edu (Giovanni Simonini)
[1] Corresponding author

1. Introduction

In the context of big data, real-world entities are typically represented in a variety of formats, such as: records of relational databases, RDF triples, JSON objects, etc. Moreover, the *profiles* (i.e., the representations) of a real-world entity often contain overlapping, complementary and/or inconsistent information. Hence, a fundamental task for unleashing the value of this data is Entity Resolution (ER) [1, 2, 3], which aims to identify and reconcile the entity profiles that refer to the same real-world entity.

Background: When the volume of the data is large, checking all possible profile pairs to find *matches* is not a practical solution due to its quadratic complexity. For this reason, typically, signatures (*blocking keys*) are extracted from the profiles and employed to index them into blocks [4]. Then, the all-pairs comparison is limited to profiles within a block, significantly reducing the complexity of ER.

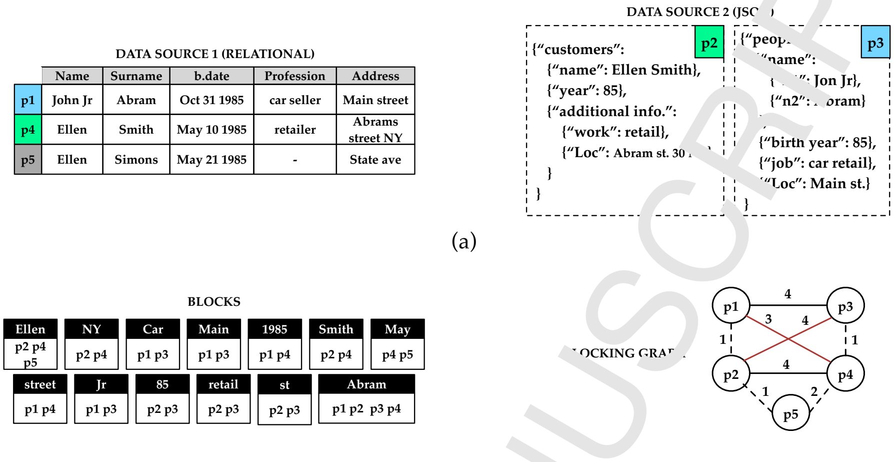
Traditional blocking techniques typically rely on a-priori schema knowledge to devise good blocking keys by combining attribute values; hence suffering from two well-known issues:

1. Given a known schema, selecting which attributes to combine requires either domain experts or labeled data to train a classification algorithm [5].

2. If two datasets have different schemata, a schema-alignment must be executed before ER. Unfortunately, big data is typically highly heterogeneous, noisy (missing/inconsistent data), and large in volume of schemata, so that traditional schema-alignment techniques are no longer applicable [6, 7]. For instance, Google Base contains over 10k entity types that are described with 100k unique schemata; in such a scenario, performing and maintaining a schema alignment is impractical [6].

To work around these issues, *schema-agnostic* blocking has been proposed [7, 8]. This approach extracts blocking keys from the profiles by treating them as *bags-of-words*. For instance, **Token Blocking** [7] considers each token in a profile as a blocking key; in other words, each pair of profiles sharing at least one token (regardless to the attribute in which it appears) is considered as a candidate match, as shown in the example of Figure 1(a-b). By placing each profile in multiple blocks, schema-agnostic techniques on one hand reduce the likelihood of missing matches, on the other hand increase the likelihood of placing non-matching profiles in the same blocks. This allows the achievement of high recall (i.e., the percentage of detected matching profiles), but at the expense of precision (i.e., the ratio between detected matching profiles and executed comparisons).

To improve the precision of schema-agnostic blocking, *meta-blocking* approaches have been proposed [8]. Meta-blocking is the task of restructuring a set of blocks to retain only the most promising comparisons. Meta-blocking represents a block collection as a weighted graph, called *blocking graph*, where



(a) (b) (c)

Figure 1: (a) A collection of entity *profiles* from a data lake where data is stored in different formats. (b) A block collection produced with Token Blocking; notice that the tokens appearing only in one profile do not generate any comparison (i.e., any block). (c) The derived blocking graph and the effect of meta-blocking: dashed lines represent pruned edges, and red ones are superfluous comparisons not removed. In this toy example, the weight of each edge connecting two profiles p_i and p_j is equal to the number of blocks in which p_i and p_j co-occur—other weighting functions can be employed [8]. For instance, p_1 and p_2 share only the block "Abram", so the weight of the edge that connects them is 1. Then, the pruning is performed computing a local threshold for each profile (e.g., the average of its edges' weights) and keeping only the edges having a weight higher than the local threshold. For instance, the weights of p_1 edges are $\{1, 3, 4\}$ and their average is 2.7, so the edge that connects p_1 with p_2 can be discarded, since $1 < 2.7$.

each entity profile is a node and an edge exists between two nodes if the corresponding profiles appear at least in one block together. The edges are weighted to capture the likelihood of a match. An example of a blocking graph is shown is Figure 1(c), where the weight of an edge is equal to the number of co-occurrences of its adjacent profiles in the blocks[2]. Then, an edge-pruning scheme is applied to retain only the most promising ones. The most accurate strategy to prune edges is to consider for each node all its adjacent edges, and retain only those having a weight higher than the local average (Figure 1(c)). At the end of the process, each pair of nodes connected by an edge forms a new block.

Our Approach: We observe that existing meta-blocking techniques exclusively leverage schema-agnostic features to restructure a block collection. Inspired by the *attribute-match induction* approaches [7, 9], our idea is to exploit schema

[2] *Co-occurrence in blocks* is employed for the sake of the example; more sophisticated weighting functions can be employed (see Section 3.3).

information extracted directly from the data for enhancing the quality of the blocks. Moreover, we argue that a holistic approach combining meta-blocking and *loosely schema-aware* techniques should be attempted. Hence, we introduce our approach called *Blast* (Blocking with Loosely-Aware Schema Techniques). *Blast* can easily collect significant statistics (e.g. similarities and entropies of the values in the attributes) that approximately describe the data sources schemas. This *loose* schema information is efficiently extracted even from highly heterogeneous and voluminous datasets, thanks to a novel LSH-based pre-processing step that guarantees a low time requirement. Then, the loose schema information is exploited during both the blocking and meta-blocking phases to produce high quality block collections.

To get an intuition of the benefits of loose schema information, consider the example in Figure 2. Say that, among the different data sources, only the attributes about person names have similar values to some extent. *Blast* clusters together these attributes, while the others ("*not enough similar*" to each other) are grouped in a unique general cluster. Thus, it can disambiguate the token "Abram" as person name from its other uses (e.g., street name). Consequently, the block associated to the token "Abram" is divided into two new blocks (Figure 2(a)) affecting the blocking graph: the weights of the edges $e_{p_1-p_4}$ and $e_{p_2-p_3}$ both decrease (Figure 2(b)). Therefore, the local thresholds for meta-blocking changes, and one further superfluous edge ($e_{p_1-p_4}$) is correctly removed in the pruning step (Figure 2(b)). The precision increases, while the recall remains the same. Yet, one superfluous comparison is still entailed ($e_{p_2-p_3}$) and loose schema information can be further employed to enhance the quality of the blocking. The intuition is that some attributes are more informative than others and can

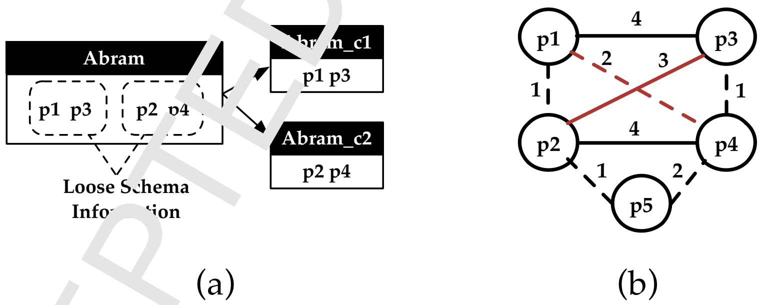


Figure 2: (a) The blocking key "Abram" is disambiguated by employing the loose schema information, as a consequence, the profiles p_1 and p_4 share one less block than before—this means also that the edge e_{1-4} decreases its weight accordingly, from 3 to 2. (b) The effect on the new blocking graph weights and on the meta-blocking process w.r.t. Figure 1(c): one further edge is correctly removed (e_{1-4}, dashed red line) compared to Figure 1(c). As a matter of fact, e_{1-4} is now pruned, since it has a weight (=2) lower than the local threshold of p_1 (=2.3); while in Figure 1(c), the weight of e_{1-4} is 3, which is greater than the local threshold of p_1 (=2.7)—notice that if the weight of e_{1-4} varies, the threshold of p_1 also changes, since the latter depends on the former.

generate more significant blocking keys. *Blast* measures the information content of an attribute through the Shannon entropy [10]. Then, it derives an *aggregate entropy* measure for each cluster of attributes. Finally, it uses these values as a multiplicative coefficient in the weighting function of the blocking graph. For our toy example, the aggregate entropies are listed in Figure 3(a) and the final blocking graph after the pruning phase is showed in Figure 3(b)[3], where the superfluous edge $e_{p_2-p_3}$ has now been correctly removed.

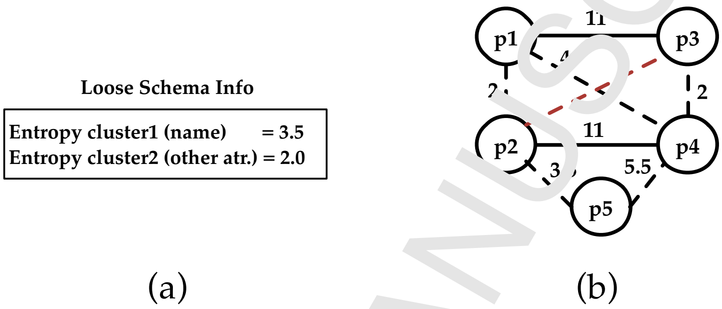


Figure 3: (a) Attribute entropy information and its effect (b) on the blocking graph pruning. In this toy example, the weighting function is: $w(p_i, p_j) = \sum_{k \in K_i \cap K_j} \mathcal{H}(b_k)$, where K_i and K_j are the set of blocking keys of p_i and p_j respectively, and $\mathcal{H}(b_k)$ is the aggregate entropy of the cluster to which b_k belongs to. In (b), the effect on the new blocking graph weights and on the meta-blocking process is shown w.r.t. Figure 2(b): one further edge is correctly removed (e_{2-3}, dashed red line) compared to Figure 2(b). As a matter of fact, e_{2-3} is now pruned, since it has a weight (=6) lower than the local threshold of p_1 (=6.3).

Contributions: Overall, we make the following main contributions:

- an approach to automatically extract *loose schema information* from a dataset based on an attribute-match induction technique;
- an unsupervised graph-based meta-blocking approach able to leverage this loose schema information;
- an LSH-based attribute-match induction technique for efficiently scale to large datasets with a high number of attributes;
- an algorithm to efficiently run *Blast* (and any other graph-based meta-blocking method) on MapReduce-like systems, to take full advantage of a parallel and distributed computation;

[3] For the sake of the example the weights are computed starting from the blocking graph of Figure 2(b); in the actual processing only one blocking graph is generated, and a unique pruning step is performed.

- the evaluation of our approach on seven real-world datasets, showing how *Blast* outperforms the state-of-the-art meta-blocking methods.

A preliminary version of *Blast* was published in [11]. In this paper, *Blast* has been extended to take advantage of a parallel and distributed computation for significantly reducing the overall execution time of the ER process, which is typically onerous in the big data context. We propose *broadcast meta-blocking* (Section 4): a novel algorithm to run any graph-based meta-blocking method (including *Blast*) on distributed MapReduce-like systems, such as Apache Spark. Finally, we provide more extensive experiments on large-scale datasets[4], which showcase that our solution efficiently scales on MapReduce-like systems and outperforms the state-of-the-art meta-blocking methods (Section 5).

Organization: The remainder of this paper is structured as follows. Section 2 provides preliminaries. Section 3 presents *Blast* and Section 4 describes basic concepts for distributed meta-blocking on MapReduce-like systems and discusses *Blast* parallelization. Section 5 presents the datasets, the evaluation metrics, and the experiments. Section 6 examines the related work. Finally, Section 7 concludes the paper.

2. Preliminaries

This section defines preparatory concepts and notation employed throughout the paper.

2.1. Blocking for Entity Resolution

An entity *profile* is a tuple composed of a unique identifier and a set of *name-value* pairs $\langle a, v\rangle$. $A_{\mathcal{P}}$ is the set of possible attributes a associated to a profile collection $\mathcal{P}$. An *profile collection* $\mathcal{P}$ is a set of profiles. Two profiles $p_i, p_j \in \mathcal{P}$ are *matching* ($p_i \approx p_j$) if they refer to the same real world object; *Entity Resolution* (ER) is the task of identifying those matches given $\mathcal{P}$.

The naive solution to ER implies $|\mathcal{P}_1|\cdot|\mathcal{P}_2|$ comparisons, where $|\mathcal{P}_i|$ is the cardinality of a profile collection $\mathcal{P}_i$. *Blocking* approaches aim to reduce this complexity by indexing similar profiles into *blocks* according to a *blocking key* (i.e., the indexing criterion), restricting the actual comparisons of profiles to those appearing in the same block.

Given the dataset of Figure 1(a), an example of *schema-agnostic* blocking key is shown in Figure 1(b). Otherwise, a *schema-based* blocking key might be the value of the attribute "name"; meaning that only profiles that have the same value for "name" will be compared (the dataset in Figure 1(a) would require a schema alignment). A set of blocks $\mathcal{B}$ is called *block collection*, and its *aggregate cardinality* is $\|\mathcal{B}\| = \sum_{b_i \in \mathcal{B}} \|b_i\|$, where $\|b_i\|$ is the number of comparisons implied by the block b_i.

[4] Two additional datasets are introduced in Section 5: `citation3` and `freebase`.

We follow best practices to establish the quality of a block collection [7, 12]: the problem of determining if two profiles actually refer to the same real-world object is the task of the *Entity Resolution Algorithm*. We assume there is such an algorithm able to determine whether two profiles are matching or not. In fact, *Blast* is independent of the *Entity Resolution Algorithm* employed, just as the other state-of-the-art blocking techniques compared in this paper [12, 13].

2.1.1. Dirty ER and Clean-Clean ER

Papadakis et al. [12] have formalized two types of ER tasks: *Dirty ER* and *Clean-Clean ER*. The former refers to those scenarios where ER is applied to a single data source containing duplicates; this problem is also known in literature as *deduplication* [14]. In the latter, ER is applied to two or more data sources, which are considered "clean", i.e., each source considered singularly does not contain duplicate. This type of ER is also known as *Record Linkage* [14]. As in [11, 12, 13, 15, 16], in this work, we adopt this classification as well.

Notice that, in *Clean-Clean ER* the comparisons among profiles that belong to the same data source are avoided (for any underlying blocking technique) [12]. Hence, the number of comparisons $\|b_i\|$ required for a block b_i depends on the type of ER: for *Dirty ER*, a block produces $\|b_i\| = \binom{|b_i|}{2}$, where $|b_i|$ is the cardinality of the block and all the possible comparisons are considered; while, for *Clean-Clean ER*, a block produces $\|b_i\| = \sum_{j=1}^{N}\sum_{k=j+1}^{N} |b_i^j| \cdot |b_i^k|$, where b_i^j is the subset of $\mathcal{P}^j$ indexed in the block b_i, and N is the number of data sources.

2.1.2. Metrics

We employ *Recall* and *Precision* to evaluate the quality of a block collection $\mathcal{B}$, as in [1]. The recall measures the portion of duplicate profiles that are placed in at least one block, while the precision measures the portion of useful comparisons, i.e., those that detect a match. Formally, precision and recall of a blocking method is determined from the block collection $\mathcal{B}$ that it generates:

$$recall = \frac{|\mathcal{D}^{\mathcal{B}}|}{|\mathcal{D}^{\mathcal{P}}|}; \qquad precision = \frac{|\mathcal{D}^{\mathcal{B}}|}{\|\mathcal{B}\|};$$

where $\mathcal{D}^{\mathcal{B}}$ is the set of duplicates appearing in $\mathcal{B}$ and $\mathcal{D}^{\mathcal{P}}$ is the set of all duplicates in the collection $\mathcal{P}$.

Typically, schema-agnostic blocking yields high recall, but at the expense of precision. The low precision is due to the unnecessary comparisons: *redundant* comparisons entail the comparison of profiles more than once; and *superfluous* comparisons entail the comparison of non-matching profiles ($p_i \not\approx p_j$).

For instance, considering the block collection of Figure 1(b), the pair of profiles (p_1, p_3) appears in many blocks ("Car", "Main", etc.), thus, if all the blocks are evaluated as traditional blocking techniques do [4] (i.e., without performing meta-blocking), p_1 and p_3 are compared more than once—performing *redundant* comparisons. Figure 1(b) also provides examples of *superfluous* comparisons, such as the comparisons between p_2 and p_5, and between p_4 and p_5, entailed by the block "Ellen"—we call these comparisons *superfluous* because p_5 do not match neither with p_2 nor p_4.

Attribute-match induction[5] approaches can be employed to enhance schema-agnostic blocking by limiting the superfluous comparisons. *Meta-blocking* is the state-of-the-art approach to reduce both superfluous and redundant comparisons from an existing block collection. In the following we formally define attribute-match induction and meta-blocking.

2.2. Attribute-match Induction

The goal of attribute-match induction is to identify groups of *similar* attributes between two profile collections $\mathcal{P}_1$ and $\mathcal{P}_2$ from the distribution of the attribute values, without exploiting the semantics of the attribute names. This information can be exploited to support a schema-agnostic blocking technique, i.e., to disambiguate blocking keys according to the attribute group from which they are derived (e.g. tokens "Abram" in Figure 1(b)).

Definition 1. *Attribute-match induction. Given two profile collections $\mathcal{P}_1, \mathcal{P}_2$, attribute-match induction is the task of identifying pairs $\{\langle a_i, a_j \rangle \mid a_i \in A_{\mathcal{P}_1}, a_j \in A_{\mathcal{P}_2}\}$ of similar attributes according to a similarity measure, and use those pairs to produce the attributes partitioning, i.e., to partition the attribute name space $(A_{\mathcal{P}_1} \times A_{\mathcal{P}_2})$ in non-overlapping clusters.*

This task is substantially different from the traditional schema-matching, which aims to detect exact matches, hierarchies, and containments among the attributes [17].

An attribute-match induction task can be defined through four components, formalized in the following: (i) the *value transformation function* (ii) the *attribute representation model*, (iii) the *similarity measure* to match attributes, and (iv) the *clustering algorithm*.

(i) **The value transformation function**. Given two profile collections $\mathcal{P}_1$ and $\mathcal{P}_2$, each attribute is represented as a tuple $\langle a_j, \tau(V_{a_j}) \rangle$, where: $a_j \in A_{\mathcal{P}_i}$ is an attribute name; V_{a_j} is the set of values that an attribute a_j can assume in $\mathcal{P}_i$; and τ is a *value transformation function* returning the set of *transformed* values $\{\tau(v) : v \in V_{a_j}\}$. The function τ generally is a concatenation of text transformation functions (e.g. *tokenization*, *stop-words* removal, *lemmatization*). Given a τ transformation function, the set of possible values in the profile collections is $T_A = T_{a_{\mathcal{P}_1}} \bigcap T_{a_{\mathcal{P}_2}}$, where $T_{a_{\mathcal{P}}} = \bigcup_{a_i \in A_{\mathcal{P}}} \tau(V_{a_i})$.

(ii) **The attribute representation model**. Each attribute a_i is represented as a vector $\mathcal{T}_i$ (called the *profile* of a_i), where each element $v_{in} \in \mathcal{T}_i$ is associated to an element $t_n \in T_A$. If $t_n \notin \tau(V_{a_i})$, then v_{in} is equal to zero.

[5] We call *attribute-match induction* the general approach to group similar attributes, while we refer to the specific technique proposed in [7] with *Attribute Clustering*.

While, if $t_n \in \tau(V_{a_i})$, then v_{in} assumes a value computed employing a weighting function, such as [7]: $TF\text{-}IDF(t_n)$ or the *binary-presence* of the element t_n in $\tau(V_{a_i})$ (i.e., v_{in}=1 if $t_n \in \tau(V_{a_i})$, 0 otherwise). For example, say that the value transformation function τ is the *tokenization* function, and that the function to weight the vector elements is the *binary-presence*. Then, the attributes are represented as a matrix: rows correspond to the attributes; the columns correspond to the possible tokens appearing in the profile collections; and each element v_{in} is either 1 (if the token t_n appear in the attribute a_i) or 0 (otherwise).

(iii) **The similarity measure**. For each possible pair of attributes $(a_j, a_k) \in (A_{\mathcal{P}_1} \times A_{\mathcal{P}_2})$, their profiles $\mathcal{T}_j$ and $\mathcal{T}_k$ are compared according to a *similarity measure* (e.g. Dice, Jaccard, Cosine). Notice that the similarity measure must be compatible with the attribute model representation; for instance, the *Jaccard* similarity cannot be employed with the $TF\text{-}IDF$ weighting.

(iv) **The clustering algorithm**. The algorithm takes as input the attribute names and the similarities of their profiles and performs the non-overlapping partitioning of the attribute names (See Section 3.1.1 for more details). Its output is called *attributes partitioning*.

2.3. Meta-blocking

The goal of meta-blocking[12] is to restructure a collection of blocks, generated by a redundant blocking technique, relying on the intuition that the more blocks two profiles share, the more likely they match.

Definition 2. Meta-blocking. *Given a block collection $\mathcal{B}$, meta-blocking is the task of restructuring the set of blocks, producing a new block collection $\mathcal{B}'$ with significantly higher precision and nearly identical recall, i.e.,: precision($\mathcal{B}'$)$\gg$precision($\mathcal{B}$) and recall($\mathcal{B}'$)$\simeq$recall($\mathcal{B}$).*

In *graph-based meta-blocking* (or simply *meta-blocking* from now on), a block collection $\mathcal{B}$ is represented by a weighted graph $\mathcal{G}_{\mathcal{B}}\{V_{\mathcal{B}}, E_{\mathcal{B}}, \mathcal{W}_{\mathcal{B}}\}$ called ***blocking graph***. V is the set of nodes representing all $p_i \in \mathcal{P}$. An edge between two entity profiles exists if they appear in at least one block together: $E = \{e_{ij} : \exists p_i, p_j \in \mathcal{P} \mid |\mathcal{B}_{ij}| > 0\}$ is the set of edges; $\mathcal{B}_{ij} = \mathcal{B}_i \cap \mathcal{B}_j$, where $\mathcal{B}_i$ and $\mathcal{B}_j$ are the set of blocks containing p_i and p_j respectively. $\mathcal{W}_{\mathcal{B}}$ is the set of weights associated to the edges. Meta-blocking methods weight the edges to capture the *matching likelihood* of the profiles that they connect. For instance, ***block co-occurrence frequency*** (a.k.a. CBS) [8, 18] assigns to the edge between two profiles p_u and p_v a weight equal to the number of blocks they shares, i.e.: $w_{uv}^{CBS} = |\mathcal{B}_u| \cap |\mathcal{B}_v|$. Then, edge-pruning strategies are applied to retain only more promising ones. Thus, at the end of the pruning, each pair of nodes connected by an edge forms a new block of the final, restructured blocking collection. Note that meta-blocking inherently prevents redundant comparisons since two nodes (profiles) are connected at most by one edge.

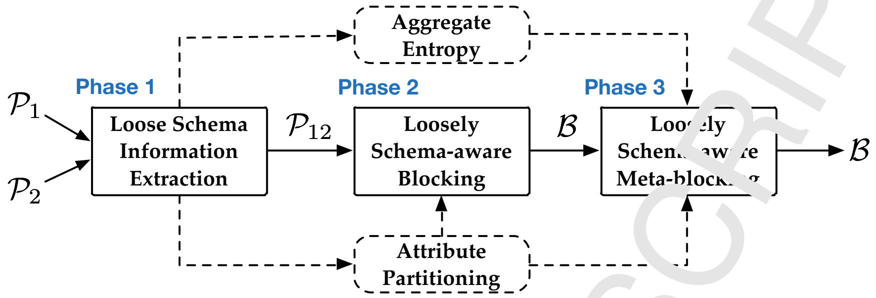


Figure 4: *Blast* logical overview.

Two classes of pruning criteria can be employed in meta-blocking: ***cardinality-based***, which aims to retain the *top-k* edges, allowing an a-priori determination of the number of comparisons (the *aggregate cardinality*) and, therefore, of the execution time, at the expense of the recall; and ***weight-based***, which aims to retain the "most promising" edges through a weight threshold. The scope of both pruning criteria can be either ***node-centric*** or ***global***: in the first case, for each node p_i the *top-k_i* adjacent edges (or the edges below a local threshold θ_i) are retained; in the second case, the *top-K* edges (or the edges below a global threshold Θ) are selected among the whole set of edges. The combination of those characteristics leads to four possible *pruning schemas*: *(i) Weight Edge Pruning* (*WEP*) discards all the edges with weight lower than Θ; *(ii) Cardinality Edge Pruning* (*CEP*) sorts all the edges by their weights in descending order, and retains only the first K; *(iii) Weight Node Pruning* (WNP [12]) considers in turn each node p_i and its adjacent edges, and prunes those edges that are lower than a local threshold θ_i; *(iv) Cardinality Node Pruning* (CNP [12]) similarly to WNP is node-centric, but instead of a weight threshold it employs a cardinality threshold k_i (i.e., retain the top-k_i edges for each node p_i).

3. The Blast Approach

The main goals of *Blast* are: to provide an efficient, scalable and automatic method to extract loose schema information from the data; to perform a holistic combination of blocking and meta-blocking for ER exploiting this loose schema information.

These are the main novelties w.r.t. other existing meta-blocking techniques, which are completely schema-agnostic [8, 12, 13].

Our approach takes as input two profile collections, and automatically generates a block collection. It consists of three main phases, as schematized in Figure 4: *loose schema information extraction*, *loosely schema-aware blocking*, and *loosely schema-aware meta-blocking*. In the following we give a high-level description of each phase, then we dedicate a subsection for the details of each phase in turn.

Phase 1) The loose schema information is extracted. It consists of: the *attributes partitioning*, and the *aggregate-entropy*. The former describes how

ACCEPTED MANUSCRIPT

the attributes are partitioned according to the similarity of their values; it is the result of the *attribute-match induction* task (Section 2.2). The latter is a measure associated to each cluster of attributes, derived from the attribute entropies. We also introduce a *Locality-Sensitive Hashing* (LSH) [19] optional step to reduce the computational complexity when dealing with data sources characterized by a high number of attributes.

Phase 2) A traditional schema-agnostic blocking technique is enhanced by exploiting the *attributes partitioning* to disambiguate keys according to the attribute partition from which they are extracted. In particular, *Blast* employs Token Blocking, and we call the derived method *Loose Schema Blocking*.

Phase 3) A graph-based meta-blocking is applied to the block collection generated in the previous phase. In particular, *Blast* meta-blocking exploits the *aggregate entropy* to weight the blocking graph. The basic idea is the following. Each edge in the blocking graph is associated to a set of blocking keys. Each blocking key is associated to an attribute. Each attribute has an information content that can be measured through its entropy. Hence, the weight of an edge can be proportional to the information content of its associated attributes. For instance, consider independent datasets containing records about people (as in Figure 1). Generally the attribute *year of birth* is less informative than the attribute *name*. This is because the number of distinct values of the former is typically lower than that of the latter. In fact, it is more likely that two people are born in the same year, than they have the same name. *Blast* tries to assess the attribute information content employing the Shannon entropy, and assigns a weight to each blocking key proportional to the entropy of the attribute from which it is derived. Thus, using *Blast*, records that share values of their *name* attributes are more likely indexed together than those sharing only values of their *year of birth* attributes. This process is completely unsupervised.

3.1. Loose Schema Information Extraction

(Phase 1 in Figure 4) In *Blast*, the loose schema information extraction is performed through an entropy extraction criterion applied in combination with the *Loose attribute-Match Induction*, an attribute-match induction technique presented here. Moreover, we propose an optional LSH-based step for guaranteeing scalability on large datasets, which is the main improvement w.r.t. Attribute Clustering [7].

3.1.1. Loose attribute-Match Induction (LMI)

Following the definitions of Section 2.2, Loose attribute-Match Induction (LMI) is composed of these four components: the *tokenization* as value transformation function; the *binary-presence* of a token as weight for the attribute representation model; the *Jaccard* coefficient as similarity measure and Algorithm 1 for clustering, a variation of the one introduced as *Attribute Clustering* (AC) in [7].

Basically, Algorithm 1 first collects the similarities of all possible attribute profile pairs of two profile collections, and their maximum values of similarity (lines 2-8). The *similarity* function (line 4) measures the Jaccard coefficient:

$$jaccard(\mathcal{T}_i, \mathcal{T}_j) = \frac{\mathcal{T}_i \cdot \mathcal{T}_j}{|\mathcal{T}_i|^2 + |\mathcal{T}_j|^2 - \mathcal{T}_i \cdot \mathcal{T}_j}.$$

where $\mathcal{T}_i, \mathcal{T}_j$ are the vectors representing the attributes a_i, a_j respectively (see Section 2.2).

Then, (lines 9-13) LMI marks as *candidate* match of an attribute each attribute that is "nearly similar" to its most similar attribute by means of a threshold α (e.g.: $0.9 \cdot maxSimValue$). If an attribute a_i has attribute a_j among its candidates, then the edge $\langle a_i, a_j \rangle$ is collected. Finally, the connected components of the graph built with these edges, with cardinality greater than one, represent the clusters (line 14). Optionally, a *glue*-cluster can gather all the singleton components (i.e., components that have cardinality equal to one), as in [7], to ensure the inclusion of all the possible tokens (blocking keys).

Algorithm 1 Loose attribute-Match Induction (LMI)

```
Input: Attr. names: A_P1, A_P2; Attr. profiles: T_1, ..., T_z; threshold: α
Output: Set of attribute names clusters: K
 1: edges ← {}      sim ← Map⟨K, V⟩
 2: Max ← Map⟨K, V⟩ // most similar attr. for each attr.
 3: for each a_i ∈ A_P1, a_j ∈ A_P2 do
 4:     sim.push(⟨⟨a_i, a_j⟩, similarity(T_i, T_j)⟩)
 5:     if sim.get(⟨a_i, a_j⟩) > Max.get(a_i) then
 6:         Max.push(⟨a_i, sim⟩)
 7:     if sim.get(⟨a_i, a_j⟩) > Max.get(a_j) then
 8:         Max.push(⟨a_j, sim⟩)
    // matching-attr. candidates generation
 9: for each a_i ∈ A_P1, a_j ∈ A_P2 do
10:     if sim.get(⟨a_i, a_j⟩) ⩾ (α · Max.get(a_i)) then
11:         edges ← edges ∪ ⟨a_i, a_j⟩
12:     if sim.get(⟨a_i, a_j⟩) ⩾ (α · Max.get(a_j)) then
13:         edges ← edges ∪ ⟨a_j, a_i⟩
14: K ← getConnectedComponentsGrThan1(edges)
15: return K
```

LSH-based Loose attribute-Match Induction

The computation of the similarity of all possible pairs of attribute profiles has an overall time complexity of $\mathcal{O}(N_1 \cdot N_2)$, where N_1 and N_2 are the cardinality of $A_{\mathcal{P}_1}$ and $A_{\mathcal{P}_2}$, respectively. For the dimensions commonly involved in the semi-structured data of the Web (the data sources schema can commonly have even thousands of attributes) this is infeasible. However, only a few (or none) similar attributes are expected to be found similar for each attribute; therefore, employing techniques able to group the attributes approximately on the basis of their similarity can significantly reduce the complexity of the attribute-match inductions, without affecting the quality of the results. Hence, in *Blast* we introduce a pre-processing step that can be optionally employed with both LMI and AC.

LSH (*Locality-Sensitive Hashing*) allows to reduce the dimensionality of a high-dimensional space, preserving the similarity distances, reducing significantly the number of the attribute profile comparisons. Employing the attribute representation model of LMI[6] and Jaccard similarity, *MinHashing* and *banding* [20] can be adopted to avoid the quadratic complexity of comparing all possible attribute pairs.

The set of attributes is represented as a matrix, where each column is the vector $\mathcal{T}_j$ of the attribute a_j (see section 2.2). Permuting the rows of that matrix, the *minhash* value of one column is the element of that column that appears first in the permuted order. So, applying a set of n hashing function to permute the rows, each column is represented as a vector of n minhash; this vector is called *minhash signature*. The probability of yielding the same minhash value for two columns, permuting their rows, is equal to the Jaccard similarity of them; thus, MinHashing preserves the similarity transforming the matrix, with the advantage of reducing the dimension of the vectors representing the attributes. However, even for relatively small n, computing the similarity of all possible minhash signature pairs may be computationally expensive; therefore, the signatures are divided into *bands*, and only signatures identical in at least one band are considered to be *candidate pairs* and given as input to the attribute-match induction algorithm (adapted to iterate only through these candidate pairs - instead of all possible pairs).

Considering n minhash values as signature, b bands for the *banding* indexing, and $r = n/b$ rows for band, the probability of two attributes being identical in at least one band is $1 - (1 - s^r)^b$. This function has a characteristic S-curve form, and its inflection point represents the threshold of the similarity. The threshold can be approximated to $(1/b)^{1/r}$. For instance, choosing $b = 30$ and $r = 5$, the attribute pairs that have a Jaccard similarity greater than $\sim$0.5 are considered for the attribute-match induction. (example Figure 5).

Thus, LSH can be employed as pre-processing step, before executing Algorithm 1, for filtering out attribute pairs that are most likely not similar, i.e., under a certain threshold[7]. Furthermore, minhash values can be employed for efficiently estimating the Jaccard similarity [20] of two attributes (line 4 in Algorithm 1). *Blast* follows this approach and stores minhash values in an array, which dominates the space complexity of Algorithm 1. The space complexity of such an array is $\mathcal{O}(n \cdot (N_1 + N_2))$, where n is the number of minhash values, and N_1 and N_2 are the cardinalities of $A_{\mathcal{P}_1}$ and $A_{\mathcal{P}_2}$, respectively; thus, Algorithm 1 has a $\mathcal{O}(n \cdot (N_1 + N_2))$ space complexity.

---

[6] The LMI attribute representation model can be used with *Attribute Clustering* [7] as well.

[7] For our experiments we found that a threshold of 0.4 works well for all the dataset, but even lower thresholds can be employed; see Section 5.6 for experiments about the LSH threshold.

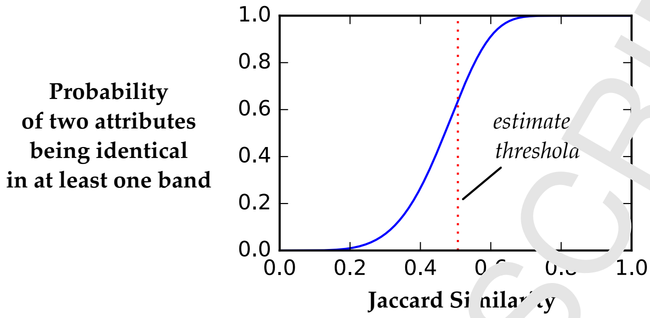


Figure 5: The depicted curve represents the probability of two attributes to be considered "similar" (y-axis) in function of their actual similarity (x-axis), when LSH is employed (with the parameters r=5 and b=30).

Entropy Extraction

To characterize each attribute cluster generated during the attribute-match induction, *Blast* employs the Shannon *entropy* of its attributes. The entropy of an attribute is defined as follows [21]:

Definition 3. Entropy. *Let X be an attribute with an alphabet $\mathfrak{X}$ and consider some probability distribution $p(x)$ of X. We define the entropy $H(X)$ by:*

$$H(X) = -\sum_{x \in \mathfrak{X}} p(x) \log p(x)$$

Intuitively, entropy represents a measure of *information content*: the higher the entropy of an attribute, the more significant is the observation of a particular value for that attribute. In other words, if the attribute assumes *predictable* values (e.g., there are only 2 equiprobable values), the observation of the same value in two different entity profiles does not have a great relevance; on the contrary, if the attribute has more *unpredictable* values (e.g., the possible equiprobable values are 100), observing two entity profiles that have the same value for that attribute can be considered a more significant clue for entity resolution.

For example, considering the data source 1 of Figure 1(a), the probability for a tuple to have as attribute Name the value "Ellen", i.e., $p(\text{"Ellen"})$, is $2/3 = 0.67$, while the probability of having "John jr" as value is $1/3 = 0.33$; thus, the entropy for the attribute Name is:

$$H(Name) = -p(\text{"Ellen"}) \cdot \log p(\text{"Ellen"}) - p(\text{"John jr"}) \cdot \log p(\text{"John jr"}) = 0.63$$

While, the entropy of the attribute Surname is 1.1, since all the tuples have different values for that attribute:

$$H(Surname) = -p(\text{"Abraham"}) \cdot \log p(\text{"Abraham"}) - p(\text{"Smith"}) \cdot \log p(\text{"Smith"})$$

$$-p(\text{“}\mathsf{Simons}\text{”}) \cdot \log p(\text{“}\mathsf{Simons}\text{”}) = 1.1$$

in this case $p(x) = 1/3 = 0.33$.

In *Blast* the importance of a blocking key is proportional to the entropy of the attribute from which it is derived. This is obtained weighting the blocking graph according to the entropies (shown in section 3.3.1). To do so, an entropy value for each group of attributes is derived by computing the *aggregate entropy*. The *aggregate entropy* of a group of attributes C_k is defined as:

$$\bar{H}(C_k) = \frac{1}{|C_k|} \cdot \sum_{A_j \in C_k} H(A_j) \tag{1}$$

When a schema-agnostic blocking (e.g. Token Blocking) is applied in combination with attribute-match induction, each blocking key b_i is uniquely associated with a cluster C_k, $b_i \mapsto C_k$. For instance, considering the example of Figure 1(b), the token “Abram”, disambiguated with attribute-match induction, can represent either the blocking key “Abram_c1” associated with the cluster C_1, or the blocking key “Abram_c2” associated with the cluster C_2; where C_1 is composed of the attributes Name of p_1 and FullName of p_3, while C_2 is composed of the attributes addr. of p_2 and Address of p_4.

For meta-blocking, *Blast* employs $h(\mathcal{B}_j)$, the entropy associated with a set of blocking keys $\mathcal{B}_j$:

$$h(\mathcal{B}_j) = \frac{1}{|\mathcal{B}_j|} \cdot \sum_{b_i \in \mathcal{B}_j} h(b_i) \tag{2}$$

where $h(b_i) = \bar{H}(C_k)$ is the entropy associated to a blocking key $b_i \mapsto C_k$.

3.2. Loosely Schema-aware Blocking

(Phase 2 in Figure 4) In *Blast* we employ Token Blocking, as in [7]. Other blocking techniques (e.g., employing q-grams instead of tokens, as in [22]) can be adapted to this scope as well, but comparing them is out of the scope of this paper. For sake of presentation, we call ***Loose Schema Blocking*** the combination of Loose attribute Match Induction and Token Blocking. The results is that each token (i.e., blocking key) can now be disambiguated according to the cluster of the attribute in which it appears, while in classical Token Blocking each token represents a unique blocking key. The example in Figure 2 gives an intuition of the benefits of this approach. Disambiguating the token “Abram” according to the attribute in which it appears avoids to index together some non-matching profiles. This affects the blocking graph weighting, and, at the end of the meta-blocking allows us to avoid one superfluous comparison.

3.3. Loosely Schema-aware Meta-blocking

(Phase 3 in Figure 4) *Blast* introduces a novel node-centric meta-blocking technique designed to exploit *loose schema information*.

Papadakis et al. [12] demonstrated that node-centric blocking-graph pruning criteria (i.e., WNP and CNP) outperforms the global ones (i.e., WEP and CEP),

	p_v (p_3)	$\neg p_v$ ($\neg p_3$)	
p_u (p_1)	n_{11} (4)	n_{12} (2)	n_{1+} (6)
$\neg p_u$ ($\neg p_3$)	n_{21} (3)	n_{22} (3)	n_{2+} (6)
	n_{+1} (7)	n_{+2} (5)	n_{++} (12)

Table 1: Contingency table for p_u, p_v. In parentheses an example derived from blocks in Figure 1(b).

and that *weight-based* pruning criteria outperform the *cardinality-based* ones in terms of recall, but at the expense of precision. Loose schema information can be exploited to significantly enhance precision; for this reason, and considering the aforementioned results achieved by [12], as a design choice, *Blast* employs a weight-based, node-centric pruning criterion (i.e., WNP).

In the following the two steps of *Blast* meta-blocking are described. In the first step, the blocking graph $\mathcal{G}_\mathcal{B}\{V_\mathcal{B}, E_\mathcal{B}, \mathcal{W}_\mathcal{B}\}$ is generated weighting the edges according to a *weighting schema* designed to capture the relevance of the profiles co-occurrence in the blocks, and to exploit the attribute entropies. The second step consists in a novel pruning criterion.

3.3.1. Blocking Graph Weighting

Considering two entity profiles p_u and p_v, the contingency table, describing their joint frequency distribution in a given block collection, is shown in Table 1. The table describes how entity profiles p_u and p_v co-occur in a block collection. For instance: the cell n_{12} represents the number of blocks in which p_u appears without p_v (the absence is denoted with "$\neg$"); the cell n_{2+} represents the number of blocks in which p_u is not present (independently of p_v). These values are also called *observed* values. As an example, the values in parentheses are values derived from the block collection of Figure 1(b) for the profiles p_1 and p_3.

Given this representation, *Blast* employs Pearson's chi-squared test (χ^2) [23] to quantify the independence of p_u and p_v in blocks; i.e., testing if the distribution of p_v, given that p_u is present in the blocks (first row of the table), is the same as the distribution of p_v, given that p_u is not present (the second row in the table). In practice, the chi-squared test measures the divergence of observed (n_{ij}) and expected (μ_{ij}) sample counts (for $i = 1, 2, j = i, 2$). The expected values are with reference to the null hypothesis, i.e., assuming that p_u and p_v appear independently in the blocks. Thus, the expected value for each cell of the contingency table is: $\mu_{ij} = (n_{i+} \cdot n_{+j})/n_{++}$.

Hence, the weight w_{uv} associated to the edge between the nodes representing the entity profiles p_u and p_v is computed as follows:

$$\begin{aligned} w_{uv} &= \chi^2_{uv} \cdot h(\mathcal{B}_{uv}) \\ &= \sum_{i\in\{1,2\}} \sum_{j\in\{1,2\}} \frac{(n_{ij} - \mu_{ij})^2}{\mu_{ij}} \cdot h(\mathcal{B}_{uv}) \end{aligned} \tag{3}$$

Notice that *Blast* uses the test statistic as a measure that helps to highlight particular profile pairs (p_u, p_v) that are highly associated in the block collection, and not to accept or refuse a null hypothesis. The correcting entropy value just weight the importance of the blocks in which a co-occurrence appear, since not all the blocks are equally important (as discussed in section 3.1.1).

3.3.2. Graph Pruning

Selecting the pruning threshold is a critical task. We identify a fundamental characteristic that a threshold selection method, in WNP, must present: the independence of the local number of adjacent edges, to avoid the sensitivity to the number of *low-weighted* edges in the blocking graph. In fact, this issue arises when employing threshold selection functions that depend on the number of edges, such as the *average* of the weights [12]. To illustrate this phenomenon,

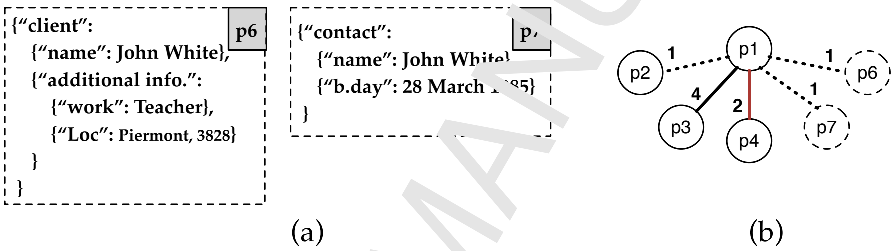


Figure 6: (a) Two additional profiles for the collection in Figure 1; (b) the node-centric representation of the blocking graph for p_1.

consider again the example in Figure 6. Figure 6(b) shows $\mathcal{G}_{p_1}$, the *node-centric* view of the $\mathcal{G}_{\mathcal{B}}$ for the profile p_1.

If the profile collection (as in Figure 1(a)) is composed only of the profile set $\{p_1, p_2, p_3, p_4\}$, the resulting graph $\mathcal{G}_{p_1}$ has only 4 nodes and 3 edges. In this scenario the average of the edge weights (the local pruning-threshold) is slightly greater than 2. Thus, only the edge between p_1 and p_3 is retained in the pruning phase. But, if the two entity profiles in Figure 6(a) are added to the profile collection, then two nodes and two edges are added to $\mathcal{G}_{p_1}$. This influences the threshold that became 1.8. Consequently, the edge between p_1 and p_4 is retained in the pruning phase. Therefore, the comparison of p_1 and p_4 depends on the presence or absence of p_5 and p_6 in the profile collection, even though the similarity between those two profiles does not depend on p_5 and p_6.

In *Blast* we introduce a *weight threshold selection schema* independent of the number of edges in the blocking graph.

Local Threshold Selection. In the node-centric view of the blocking graph, the edge with the highest weight represents the upper bound of similarity for the combination of the underlying blocking technique and weighting function; so, we propose to select a threshold independent of the number of adjacent edges

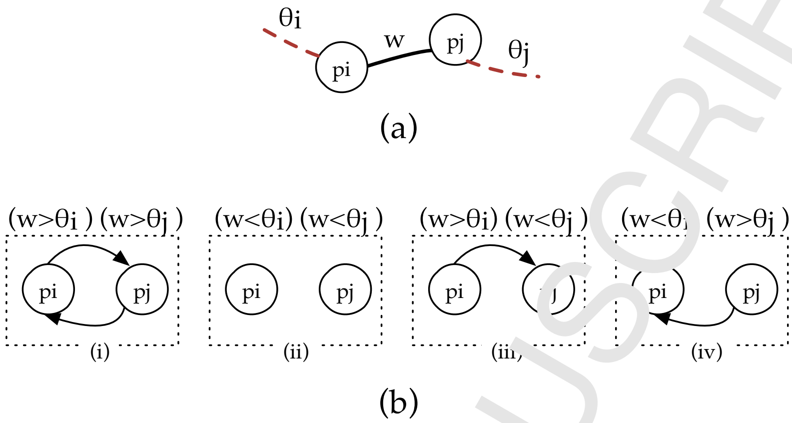


Figure 7: Weight threshold. A directed edge from p_i to p_j indicates that the weight of the edge e_{ij} is higher than θ_i; a directed edge from p_j to p_i indicates that the weight of the edge e_{ij} is higher than θ_j.

by considering a fraction of this upper bound:

$$\theta_i = \frac{M}{c} \tag{4}$$

where M is the local maximum weight, and c an arbitrary constant. A value for c that has shown to be efficacious with real dataset is c=2; a higher value for c can achieve higher recall, but at the expense of precision.

Having determined the local threshold for each node, the last step to perform is the retention of the edges. Though, in node centric pruning, each edge e_{ij} between two nodes p_i and p_j is related to two thresholds: θ_i and θj (Figure 7(a)); where θ_i and θ_j are the threshold associated to p_i and p_j, respectively. Hence, as depicted in Figure 7(b), each edge e_{ij} has a weight that can be: (i) lower than both θ_i and θ_j, (ii) higher than both θ_i and θ_j, (iii) lower than θ_i and higher than θ_j, or (iv) higher than θ_i and lower than θ_j. Cases (i) and (ii) are not ambiguous, therefore e_{ij} is discarded in the first case, and retained in the second one. But, cases (iii) and (iv) are ambiguous.

Existing meta-blocking papers [12] propose two different approaches to solve this ambiguity: *redefined* WNP retains e_{ij} if its weight is higher than at least one of the two thresholds (i.e., a logical disjunction, so we cal this method WNP_{OR}), while *reciprocal* WNP retains the edge if it is greater than both the threshold (i.e., a logical conjunction, so we cal this method WNP_{AND}). Here in *Blast* we choose to employ a unique general threshold, equals to:

$$\theta_{ij} = \frac{\sqrt{\theta_i + \theta_j}}{d} \tag{5}$$

where d is a constant; for $d = 2$ the resulting threshold θ_{ij} is equal to the mean of the two involved local threshold, and has shown to perform well with real datasets.

The experimental section 5.3 shows how the parameters c and d influence the performances of *Blast* and in particular, the tradeoff of precision and recall for an ER task.

4. Distributed Meta-blocking

We now introduce basic concepts of MapReduce-like systems and then describe what is needed to parallelize *Blast* for taking full advantage out of parallel and distributed computation.

4.1. MapReduce-like Systems

In MapReduce-like Systems, programs are written in functional style and automatically executed in parallel on a cluster of machines. These systems also provide automatic mechanisms for load balancing and to recover from machine failures without recomputing the whole program by leveraging on the functional programming abstraction (e.g., *lazy evaluation* in Apache Spark [24]). In the following, we present the main *functions* employed to formalize MapReduce-like algorithms in this paper with a concise and *Spark-like* syntax. These functions are defined w.r.t. *Resilient Distributed Dataset* (RDD [24]), which are the basic data structure in Apache Spark. In a nutshell, an RDD is a distributed and resilient collection of objects (e.g.: *integers*, *strings*, etc.).

Basic Functions for MapReduce-like Algorithms

- `map` (`map` in MapReduce [25]) applies a given function to all elements of the RDD returning a new RDD.
- `mapPartitions`: applies a given function to each RDD partition returning a new RDD.
- `reduceByKey` (`reduce` in MapReduce [25]) reduces the elements for each key of an RDD using a specified commutative and associative binary function.
- `groupByKey`: groups the values for each key in the RDD into a single collection.
- `join`: performs a hash join between two RDDs.
- `broadcast`: broadcasts a read-only variable to each node in the cluster (which cache it).

We employ this set of functions for the sake of presentation of our algorithms for MapReduce-like systems (Section 4.2). Yet, the algorithms discussed in this paper employing such functions are general enough to run on any MapReduce-like systems.

In MapReduce-like systems implementations, functions like `join` and `groupByKey` are notoriously expensive, due to the so-called `shuffling` of data across the network [26]. In fact, they involve redistribution of the data across partitions with the consequent overheads: data serialization/deserialization, transmission of data across the network, disk I/O operations. For instance, `join` implies that all the records that have the same key are sent to the same node. Whereas, `map` and `mapPartitions` are usually fast to compute, because data is locally processed in memory, and no shuffling across the network is required [26].

4.2. Blast on MapReduce-like Systems

4.2.1. Distributed Blocks Generation

For the *loose information extraction* and *loosely schema-aware blocking* (Phases 1 and 2 in Figure 4), adapting the proposed solution of Section 3 to the MapReduce paradigm is straightforward. It only requires an underlying MapReduce-based LSH algorithm (such as [27]). Then, adapting Token Blocking to the MapReduce paradigm is straightforward as well (it essentially builds an inverted index).

The main challenge for the parallelization of *Blast* is related to the graph-based meta-blocking step. In fact, the blocking graph, defined in Section 2, is an abstract model useful to formalize and devise meta-blocking methods. However, materializing and processing the whole blocking-graph may be challenging in the context of big data due to the size of such a graph. For this reason, algorithms for processing the blocking-graph have been proposed to scale meta-blocking to large datasets on MapReduce-like systems [13]. Their basic idea is to distribute the blocking-graph processing on multiple machines, trading a fast execution for high resource occupation.

In the following, firstly we revise the state-of-the-art blocking-graph processing algorithm, i.e., *repartition meta-blocking*[8] [13], discussing its limitations; then, we present our novel algorithm called *broadcast meta-blocking*, which overcome these limitations.

4.2.2. Distributed Blocking-graph Processing

Repartition meta-blocking— At the core of *repartition meta-blocking* [13] there is a full materialization of the blocking graph.

Algorithm 2 describes the *repartition meta-blocking* with pseudocode. Firstly, for each $profile$ and for each of its blocking key, a pair $\langle key, profile\rangle$ is generated (Lines 3-5). The result can be seen as a table P^K with two columns: *key* and *profile*. Then, a self-join on P^K (Line 6) and a group by profile (Lines 7) are performed. In practice, this corresponds to a graph materialization, since each node is associated with a copy of its local neighborhood. As a matter of fact, each element of P^G (Line 7) is a set of pairs $\langle p_i,\ p_j\rangle$, where p_i is fixed and p_j is a profile sharing at least one blocking key with p_i.

[8] In [13] this algorithm is called *entity-based parallel meta-blocking* (an example is shown in Figure 4 of [13]) and it is the state-of-the-art (i.e., fastest and efficient) algorithm for performing node-centric pruning on the blocking graph; we coined the term *repartition meta-blocking* for the analogy with the *repartition join* algorithm [28].

Algorithm 2 *Repartition Meta-blocking* [13]

Input: P, the profile collection
Output: C, the list of retained comparisons

1: $P^K \leftarrow \varnothing$
2: $C \leftarrow \{\}$ `// retained comparisons`
3: **map** $\langle profile \rangle \in P$
4:     **for each** $k \in getKeys(profile)$ **do**
5:         $P^K \leftarrow P^K \cup \langle key,\ profile \rangle$
6: $P^J \leftarrow P^K$ **join** P^K **on** key `// self-join`
7: $P^G \leftarrow$ **groupByKey** (P^J)
8: **map** $\langle profileNeighborhood \rangle \in P^G$
9:     $C_p \leftarrow$ **prune(*profileNeighborhood*)**
10:     $C.append(C_p)$

Finally, for each profile p_i and its neighborhood (Lines 8-10), a pruning function computes a local threshold θ_i and retains only the edges with a weight higher than θ_i (Lines 9)[9].

Optimization note— When implementing *repartition meta-blocking*, for alleviating the network communication bottleneck, blocks and profiles are represented by their ids, as proposed in [13]. This means that, for Algorithm 2, the pair $\langle key,\ profile \rangle$ (in Line 5) is a pair of identifiers: the first id represents the *key* (i.e., the block), the second id represents the entity profile.

Example 1. *An example of the execution steps of repartition meta-blocking is shown in Figure 8. Five profiles are grouped in three partitions: $\{p_1\}$, $\{p_2; p_3\}$ and $\{p_4; p_5\}$. Each partition is assigned to a worker (i.e., a physical computational node) that computes the $\langle key, profile \rangle$ pairs (Step 1). The resulting set of pairs P^K is then employed for a self join in order to yield the bag of all the comparison pairs $\langle p_i, p_j \rangle$; this step (Step 2) requires a shuffling of the data (P^K) through the network (note that only the ids of the profiles are sent around the network). The comparison pairs are assigned to a worker according to their keys, so the group by operator partitions them to materialize the neighborhoods within each worker (Step 3). Thus, in parallel, each neighborhood can be processed to generate the final restructured block collection (Step 4).*

The bottleneck of *repartition meta-blocking* is the `join` (Line 6 in Algorithm 2). In fact, Efthymiou et al. [13] describe it as a *standard repartition join* [28] (a.k.a. *reduce-side join*), a notoriously expensive operator for MapReduce-like systems[10]. A workaround for this issue could be the employment of *broadcast*

---

[9] Some pruning functions requires as input both the local threshold of the current node p_i and the local threshold of its neighbors; in this case, (Lines 8-10) are executed two times: first, for computing all the thresholds (which are then broadcasted); then, for the actual pruning.

[10] We make explicit the *join* operator: Efthymiou et al. present their algorithms in [13] by using only `map` and `reduce` functions.

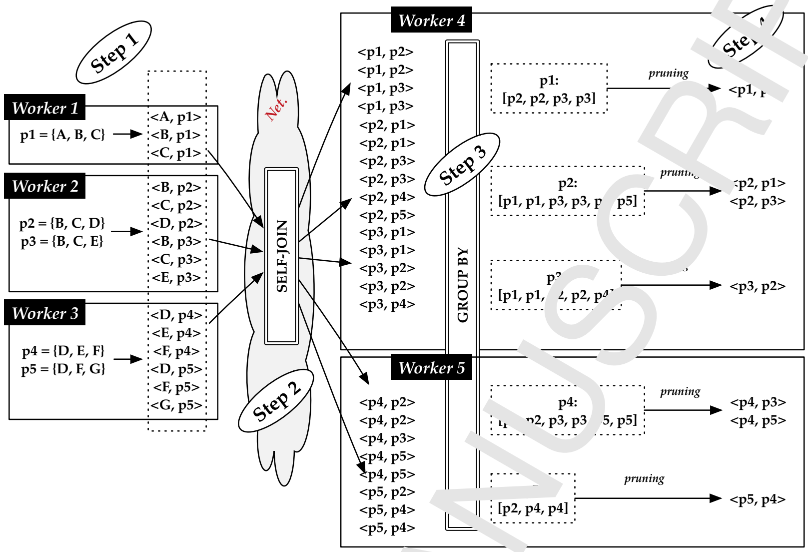


Figure 8: Repartition meta-blocking example

join [28], a join operator for MapReduce-like systems that is very efficient if one of the join tables can fit in main memory. Unfortunately, P^K (Line 6 in Algorithm 2) typically cannot fit in memory with large dataset (e.g., those employed in our experiments in Section 5). Thus, *broadcast join* cannot be employed in Algorithm 2.

Broadcast meta-blocking— To avoid the repartition join bottleneck, we propose a novel algorithm for parallel meta-blocking inspired by the broadcast join. The key idea of our algorithm is the following: instead of materializing the whole blocking graph, only a portion of it is materialized in parallel. This is possible by partitioning the nodes of the graph and sending in broadcast (i.e., to each partition) all the information needed to materialize the neighborhood of each node one at a time. Once the neighborhood of a node is materialized, the pruning functions that can be applied are the same employed in *repartition meta-blocking* [13], and (non-parallel) *meta-blocking* [12, 8].

The pseudocode of *broadcast meta-blocking* is shown in Algorithm 3 and described in the following. Given the profile collection $\mathcal{P}$ the block index I_B is generated (Lines 1-2): it is an inverted index listing the profile ids of each block (blocks are represented through ids as well). When executing *Blast*, the functions *buildBlocks* and *buildBlockIndex* also extract the loose schema information—i.e., they basically perform what is described in Section 4.2.1. Then, I_B is broadcasted to all workers (Line 4), in order to make it available to them. On each partition, an index I_P is built (Lines 5-6): for each profile it lists the block identifiers in which it appears. Then, for each partition and for each profile, by using the I_P and I_B indexes, a profile's neighborhood at a time is built locally (Lines 7-9): for each block id contained in I_P it is possible to obtain

Algorithm 3 *Broadcast Meta-blocking*

Input: P, the profile collection
Output: C, the list of retained comparisons
1: $B \leftarrow buildBlocks(P)$
2: $I_B \leftarrow buildBlockIndex(B)$
3: $C \leftarrow \{\}$ `// retained comparisons`
4: **broadcast**(I_B)
5: **map partition** $\langle part \rangle \in P$
6: $\quad I_P \leftarrow buildProfileBlockIndex(I_B)$
7: $\quad$ **for each** $profile \in part$ **do**
8: $\quad\quad B_{ids} \leftarrow I_P[profile.id]$
9: $\quad\quad profileNeighborhood \leftarrow buildLocalGraph(B_{ids}, I_B)$
10: $\quad\quad C_p \leftarrow$ **prune**(***profileNeighborhood***)
11: $\quad\quad C.append(C_p)$

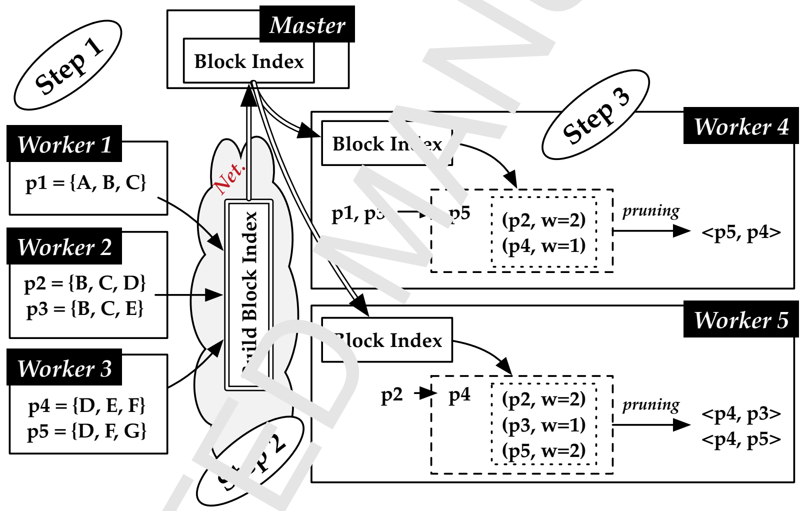


Figure 9: Broadcast meta-blocking example

from I_B the list of profile ids (the neighbors). Finally, it performs the pruning (Lines 10-11)[11].

Note that the `prune` function employed in Algorithm 2 (Line 9) and Algorithm 3 (Line 10) takes as input a profile's neighborhood and can be any node-centric pruning function, e.g., the one described in Section 3.3.

Example 2. *An example of the execution steps of broadcast meta-blocking is shown in Figure 9. In Step 1 the profiles are partitioned and assigned to the workers. Then, in Step 2, the inverted index of blocks (the Block Index) is*

[11] As for Algorithm 2, for some pruning functions, this last iteration has to be performed twice: the first time for computing all the thresholds, the second for the actual pruning.

built—for the sake of the example, the intermediate steps to build the inverted index are not depicted. This step requires a shuffling of data though the network, but at a significantly lower extent compared to that needed for the self-join operation of repartition meta-blocking. Then, the Block Index is broadcasted to all the workers that perform the last phase of the processing (Step 2). Finally, in Step 3, each worker processes a partition of the profile set: it materializes a neighborhood at a time by exploiting the local instance of the Block Index, and performs pruning to yield the final restructured block collection.

5. Evaluation

The experimental evaluation aims to answer the following questions:

*Q*1: *What is the performance of Blast in terms of precision, recall, and execution time compared to the state-of-the-art [12]?* (Section 5.1)

*Q*2: *What is the contribution of each Blast component to the overall performance (e.g., how the performance changes by employing the aggregate entropy)?* (Section 5.2)

*Q*3: What are good parameters *c* and *d* for the pruning threshold of *Blast* (see Section 3.3.2) for a good recall/precision tradeoff? (Section 5.3)

*Q*4: *How efficient is broadcast meta-blocking, compared to repartition meta-blocking [13]?* (Section 5.4)

*Q*5: *How does Blast (with broadcast meta-blocking) scale when varying the number of machines available for the ER processing?* (Section 5.5)

*Q*6: *How does the LSH-based step affects the Blast processing?* (Section 5.6)

*Q*7: *What is the performance of Blast w.r.t. traditional meta-blocking when no schema-alignment is required (i.e., with a single data source with known schema containing duplicates)?* (Section 5.7)

*Q*8: *What is the performance of Blast w.r.t. traditional meta-blocking in a multi-data source context (i.e., when the number of data sources is greater than 2)? (Section 5.8)*

Experimental Setup

Hardware and Software—All the experiments are performed on a ten-node cluster; each node has two Intel Xeon E5-2670v2 2.50 GHz (20 cores per node) and 128 GB of RAM, running Ubuntu 14.04. All the software is implemented in Scala 2.11.8 and available at [29]. To assess the performance of the state-of-the-art meta-blocking methods we re-implemented all of them for running on Apache Spark as well. We employ Apache Spark 2.1.0, running 3 executors on

	Size	$\|\mathcal{P}_1\| - \|\mathcal{P}_2\|$	$\|\mathcal{A}_1\| - \|\mathcal{A}_2\|$	$\|\mathcal{D}_P\|$
`articles1` (*)	small	2.6k - 2.3k	4 - 4	2.2k
`articles2` (*)	small	2.5k - 61k	4 - 4	2.3k
`products` (*)	small	1.1k - 1.1k	4 - 4	1.1k
`movies`	small	28k - 23k	4 - 7	23k
`articles3` (*)	large	1.8M - 2.5M	7 - 7	0.6M
`dbpedia`	large	1.2M - 2.2M	30k - 50k	0.9M
`freebase`	large	4.2M - 3.7M	37k - 11k	1.5M

Table 2: Dataset characteristics: number of entity profiles, number of *attribute names*, and number of existing matches. An exact schema alignment can be achieved only on starred "(*)" datasets.

each node, reserving 30 GB of memory for the master node. We set the default parallelism to twice the number of cores as suggested by best practice[12].

Datasets—Table 2 lists the 7 real-world datasets employed in our experiments. They have different characteristics and are from a variety of domains. The small datasets (i.e., `articles1`, `articles2`, `products`, and `movies`) are used only when evaluating the performance in terms of recall and precision, since their time performance on distributed setting is not significant. (Table 4 reports the definition of precision and recall from Section 2.)

All the datasets match two different data sources for which the ground truth of the real matches is known. From [30]: `articles1` matches scientific articles extracted from dblp.org and dl.acm.org; `articles2` matches scientific articles extracted from dblp.org and scholar.google.com. `products` matches products extracted from Abt.com and Buy.com. From [7]: `movies` matches movies extracted from imdb.com and dbpedia.org; `dbpedia` matches entity profiles from two different snapshots of DBpedia (2007 and 2009)[13]. From [31]: `articles3` matches scientific articles extracted from Citeseer and DBLP. Finally, `freebase` is derived from the Billion Triple Challenge 2012 Dataset [32]: it is composed by two datasets, one contains the data of DBpedia 3.7, the other one the data of Freebase; we cleaned these two datasets keeping only the information in English, removing other languages; the ground truth is represented by the *owl:sameAs* relationships between them.

Methods Configurations and Results Analysis—For each dataset, the initial block collection is extracted through a redundant blocking technique (either Token Blocking or Loose Schema Blocking). Then, the block collection is processed with Block Purging and Block Filtering [12], which aim to remove/shrink the largest blocks in the collection. Block Purging discards all the blocks that contain more than half of the entity profiles in the collection, corresponding to highly frequent blocking keys (e.g. stop-words). Block Filtering removes

[12]https://spark.apache.org/docs/latest/tuning.html

[13]Only 25% of the name-value pairs are shared among the two snapshots, due to the constant changes in DBpedia, therefore the ER is not trivial.

each profile p_i from the largest 20% blocks in which it appears[14]. The time required by both Block Purging and Block Filtering is negligible compared to the meta-blocking phase, thus not listed in the experimental results.

The schema-agnostic meta-blocking methods can be executed on blocks generated with both Token Blocking and with Loose Schema Blocking, while *Blast* is compatible with the latter only, since it exploits the loose schema information.

For the schema-agnostic meta-blocking methods, we report the average values of recall, precision, F1-score[15] and time obtained by executing each method in combination with each of the five weighting schemas proposed in [7][16]. We also report that no traditional weighting schema and pruning strategy combination performs better than the other on the considered datasets, confirming the results of [7].

Finally, for the time measurement, we report the values obtained by averaging the times recorded for five runs. Table 3 summarizes the acronyms used in this Section.

5.1. Blast vs. State-of-the-art Meta-blocking

Table 3 summarizes the acronyms and configurations employed in this experiment. WNP and CNP is applied on block collections generated both with Token Blocking (TB) and Loose Schema Blocking (LSB), and employing both *redefined* (WNP_{OR}/CNP_{OR}) and *reciprocal* (WNP_{AND}/CNP_{AND}) approaches (see Section 3.3.2).

Figure 11 shows the result of the execution of *Blast* and traditional meta-blocking on all the datasets. Compared to WNP approaches, *Blast* achieves significantly higher precision and basically the same level of recall on all the datasets. In particular *Blast* always outperforms LSB+$\text{WNP}_{\text{OR/AND}}$, demonstrating that the *Blast* weight-based pruning is actually more effective than the traditional ones.

Compared to TB+$\text{CNP}_{\text{OR/AND}}$, *Blast* achieves higher precision on all the datasets, with the exception of `articles2` and `freebase`, where CNP_{AND} has a higher precision (Figure 11(i) and Figure 11(n)). Notice though that on `articles2` and on `freebase` *Blast* achieves a recall significantly higher (Figure 11(b) and Figure 11(g)). On all the other datasets, the recall of *Blast* is almost the same of TB+$\text{CNP}_{\text{OR/AND}}$ (Figure 11(a-g)), or slightly higher (Figure 11(b) and Figure 11(g)). Similarly, *Blast* outperforms LSB+$\text{CNP}_{\text{OR/AND}}$

[14] This heuristic has shown to not affect recall in practice, while lighting the blocking-graph handling [12].

[15] Hand et al. [33] have recently discussed how F1-score may be an unreliable measure for comparing different ER algorithms. We report F1-score for the sake of completeness—it has been used in many related works [5, 34, 35]—yet we draw conclusions on the basis of precision and recall only.

[16] Among the weighting schemas proposed in [7], we did not identify an overall best performer and an overall worst performer, confirming the results reported in [13], for this reason we report the average precision, recall, F1-score and execution time.

Blocking

TB	Token Blocking [7] (see Section 1)
LSB	Loose-Schema Blocking (see Section 3.2)

Meta-blocking

WNP	Weight Node Pruning [12] (see Section 2.3)
CNP	Cardinality Node Pruning [12] (see Section 2.3)
$WNP_{OR}(CNP_{OR})$	The *redefined* WNP (CNP) approach [12] (see Section 3.3.1). An edge is not pruned if it weight is greater than any of its adjacent node's local thresholds (OR condition)
$WNP_{AND}(CNP_{AND})$	The *reciprocal* WNP (CNP) approach [12] (see Section 3.3.1). An edge is not pruned if it weight is greater than both of its adjacent node's local thresholds (AND condition)

Blast

$Blast_{\chi^2}$	*Blast* approach, without employing the aggregate entropy to compute the weights of the edges (see Section 3.3.1).
$Blast^{\mathcal{H}}$	*Blast* approach, using the weighting schema proposed in [12] instead of χ^2 to weight the edges (see Section 3.3.1). The entropy is used. The results reported are the average of all the weighting schema.
$Blast^{\mathcal{H}}_{\chi^2}$ (or simply *Blast*)	*Blast* approach (i.e., with χ^2 and aggregate entropy, see Section 3).

Table 3: Acronyms and configurations.

$\lVert\mathcal{B}\rVert$	Number of comparisons entailed by a block collection $\mathcal{B}$
$\lvert\mathcal{D}^{\mathcal{P}}\rvert$	Number of duplicates (matches) in a profile collection $\mathcal{P}$
$\lvert\mathcal{D}^{\mathcal{B}}\rvert$	Number of duplicates (matches) indexed in at least one block $b \in \mathcal{B}$
$recall(\mathcal{B})$	$\lvert\mathcal{D}^{\mathcal{B}}\rvert/\lvert\mathcal{D}^{\mathcal{P}}\rvert$
$precision(\mathcal{B})$	$\lvert\mathcal{D}^{\mathcal{B}}\rvert/\lVert\mathcal{B}\rVert$

Table 4: Metrics.

in terms of precision on all the datasets but `articles2` and `freebase` (Figure 11(i) and Figure 11(n)). Yet, on these datasets *Blast* yields a higher recall (Figure 11(b) and Figure 11(g)).

We also considered the overall execution time of the methods. For the comparison, we employed our Spark implementation of them, employing *broadcast-meta-blocking* as core blocking-graph processing algorithm, running on a single node (for scalability and performance on multiple nodes see Section 5.5). In such a configuration, for the small datasets the results are not reported: the overhead introduced by Spark in each execution does not allow to properly record the actual time efficiency of such configuration when the size of

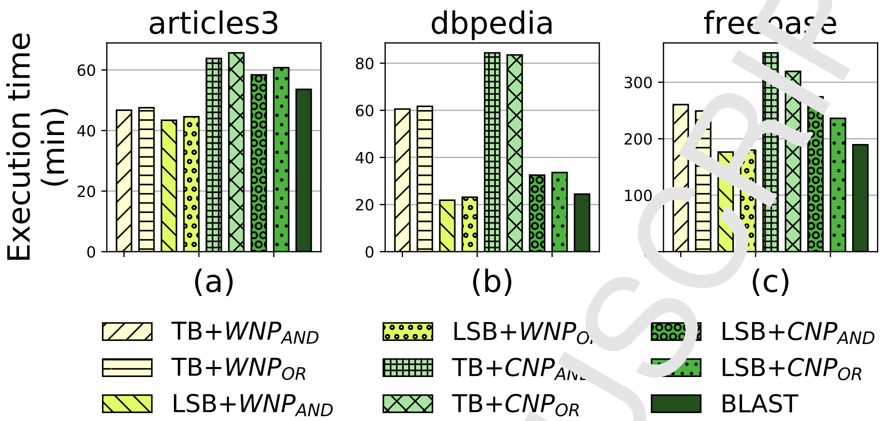


Figure 10: Execution time of the different methods applied on blocks obtained with the Token Blocking (TB+WNP$_{ADN/OR}$/CNP$_{ADN/OR}$) and with the Loose Schema Blocking (LSB+WNP$_{ADN/OR}$/CNP$_{ADN/OR}$). The execution time is referred to the meta-blocking, and it was taken on a single node on the biggest datasets.

the data is small[17]. The results are shown in Figure 10. *Blast* is always significantly faster than CNP$_{OR/AND}$ on all the considered datasets and all the configurations (up to 3.8× on `dbpedia` in Figure 10(b)). It is also faster than TB+WNP$_{OR/AND}$ on `dbpedia` (2.8× in Figure 10(b)) and `freebase` (1.6× in Figure 10(c)); while, on `articles3` is slightly slower (Figure 10(a)). Compared to LSB+WNP$_{OR/AND}$, *Blast* has almost the same execution time on `dbpedia` (Figure 10(b)) and `freebase` (Figure 10(c)); while on `articles3` is slightly slower (Figure 10(a)).

Overall, we conclude that *Blast* yields the same recall and a significantly higher precision of the best performing schema-agnostic meta-blocking methods [12], on each dataset[18]. The only exception is LSB+CNP$_{OR/AND}$, which achieves higher recall than *Blast* on two of the seven considered datasets (Figure 11(i) and Figure 11(n)), but at the same time has lower recall (Figure 11(b) and Figure 11(g)) and is always slower than *Blast* Figure 10. Finally, we also observe that *Blast* has time performance similar to the fastest schema-agnostic method.

[17] In [11] the time for these datasets are reported for the Java implementation and the results are analogous.

[18] The differences between *Blast* and WNP/CNP are statistically significant according to Student's T-Test (with p-value < 0.05).

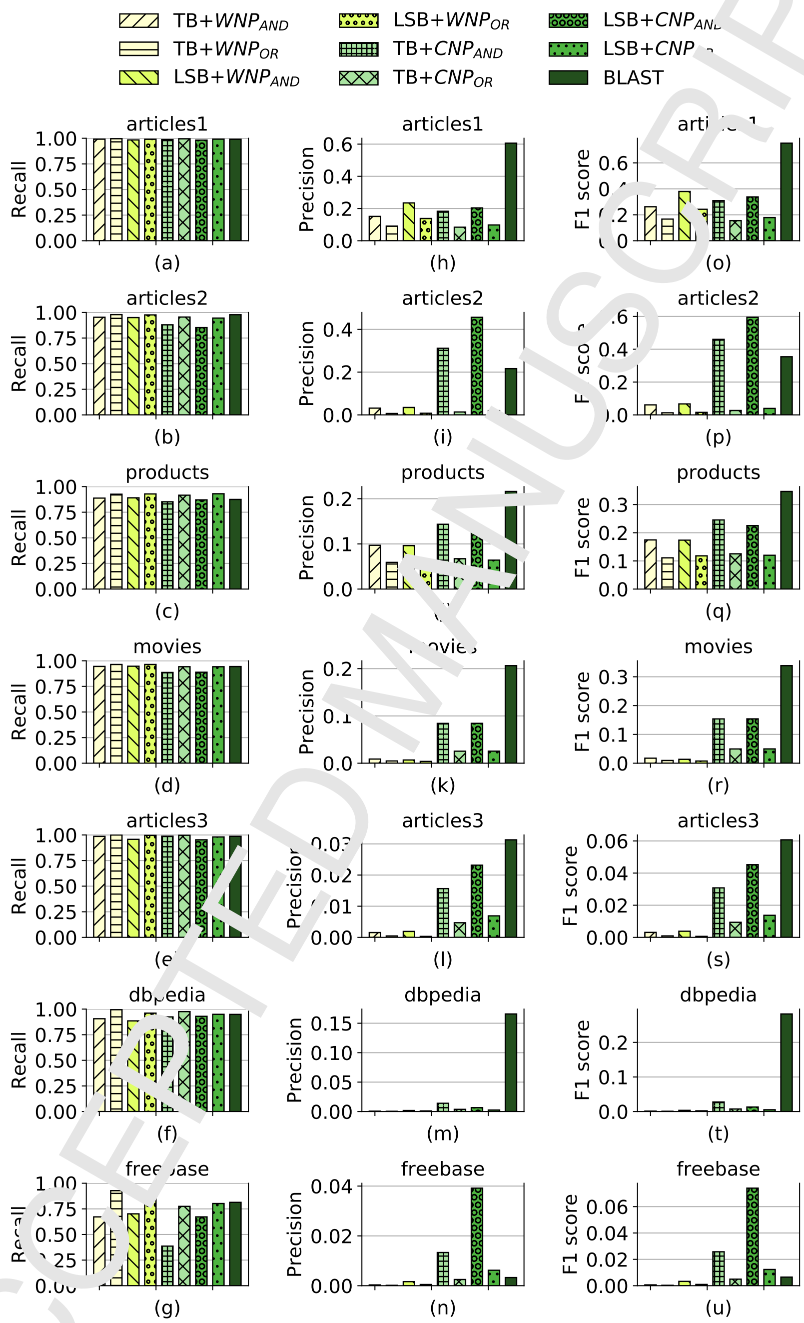


Figure 11: Recall and precision achieved by the considered methods on all the datasets. Traditional meta-blocking ($\mathrm{WNP}_{\mathrm{ADN/OR}}$ and $\mathrm{CNP}_{\mathrm{ADN/OR}}$) has been combined both with Token Blocking (TB+$\mathrm{WNP}_{\mathrm{ADN/OR}}$/$\mathrm{CNP}_{\mathrm{ADN/OR}}$) and Loose Schema Blocking (LSB+$\mathrm{WNP}_{\mathrm{ADN/OR}}$/$\mathrm{CNP}_{\mathrm{ADN/OR}}$). *Blast* is based on Loose Schema Blocking for the extraction of the loose schema information, thus it is not applicable on block collection generate with Token Blocking.

5.2. Blast Components Evaluation

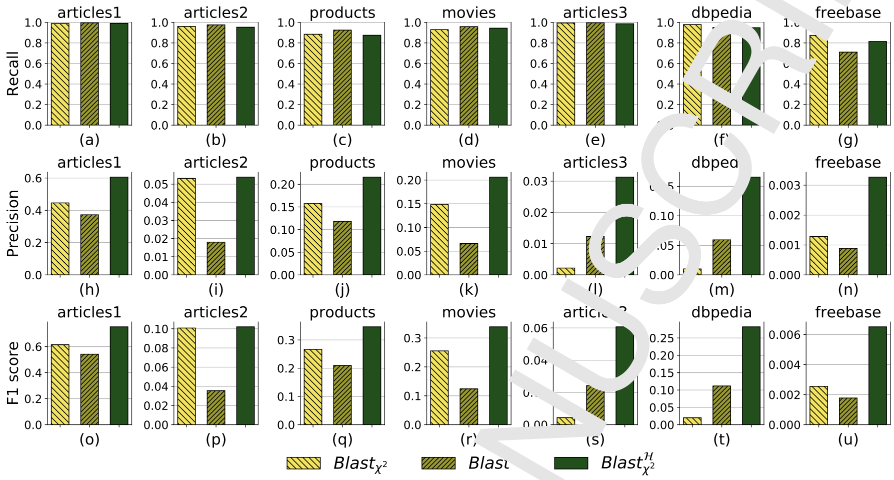


Figure 12: *Blast* running: without considering the *aggregate entropy* ($Blast_{\chi^2}$); in combination with traditional schema-agnostic weighting functions ($Blast^{\mathcal{H}}$); standard configuration ($Blast^{\mathcal{H}}_{\chi^2}$).

In this experiment we evaluate the contribution provided by each component characterizing *Blast*: the *aggregate entropy* and the *weighting function*. The results are reported in Figure 12.

We compare three different configurations of meta-blocking performed on a block collection generated through *Loose Schema Blocking*: $Blast_{\chi^2}$, $Blast^{\mathcal{H}}_{\chi^2}$, $Blast^{\mathcal{H}}_{\chi^2}$, as described in Table 3.

Aggregate Entropy

The comparison of $Blast_{\chi^2}$ and $Blast^{\mathcal{H}}_{\chi^2}$ allows us to assess the contribution of the *aggregate entropy*. The result in Figure 12(h-n) shows that by employing the *aggregate entropy* precision increases from 1.6 (Figure 12(h)) to 3.7 times (Figure 12(n)). At the same time, recall is almost the same on all datasets (Figure 12(a-g)). On `freebase`, $Blast^{\mathcal{H}}_{\chi^2}$ even achieves both recall and precision significantly higher than $Blast_{\chi^2}$ (Figure 12(g) and Figure 12(n)).

We conclude that *aggregate entropy* actually enhances meta-blocking.

Chi-squared weighting

Blast employs a weighting function derived from the chi-squared (χ^2) statistical test designed to quantify the significance of the co-occurrences (see Section 3.3). For assessing the performance of this weighting function, $Blast^{\mathcal{H}}$ is compared with $Blast^{\mathcal{H}}_{\chi^2}$. The result is shown in Figure 12. Recall is almost the same for all the datasets for $Blast^{\mathcal{H}}$ and $Blast^{\mathcal{H}}_{\chi^2}$ (Figure 12(a-g)), while $Blast^{\mathcal{H}}_{\chi^2}$ achieves a considerably higher precision (Figure 12(h-n)), e.g. on `dbpedia` (Figure 12(m)) precision has a 16× improvement. The only exceptions are `articles2` and `freebase`: on the former, $Blast^{\mathcal{H}}_{\chi^2}$ achieves almost the same recall and precision yielded by $Blast^{\mathcal{H}}$ (Figure 12(b) and Figure 12(i)); on the latter, $Blast^{\mathcal{H}}$ has a 4.6% higher recall, yet $Blast^{\mathcal{H}}_{\chi^2}$ yields a precision more than twice higher than $Blast^{\mathcal{H}}$ (Figure 12(n)).

We conclude that our weighting function actually enhances meta-blocking performance.

5.3. Blast sensitivity to parameters

From Section 3.3.2, to perform the graph pruning, *Blast* computes a local threshold θ_i for every profile p_i. This local threshold is computed as $\theta_i = \frac{M}{c}$ (from Equation 4), where M is the local maximum weight, and c is an arbitrary constant. Then, for retaining an edge between two profiles p_i, p_j, a unique threshold θ_{ij} is computed as $\theta_{ij} = \frac{\sqrt{(\theta_i^2+\theta_j^2)}}{d}$ (from Equation 5), where d is an arbitrary constant.

The constants c and d can be reduced to a unique constant $t = c \cdot d$, as shown below:

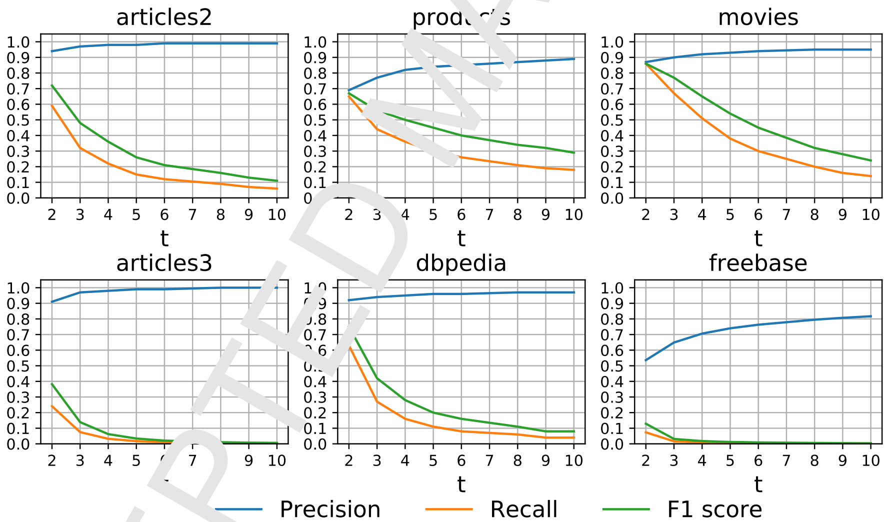


Figure 13: *Blast* sensitivity: these charts shown the variations of precision, recall, and F1 score in function of the t parameter.

$$\begin{aligned}\theta_{ij} &= \frac{1}{c} \cdot \sqrt{\left(\frac{\theta_i}{d}\right)^2 + \left(\frac{\theta_j}{d}\right)^2} = \frac{1}{c} \cdot \sqrt{\frac{\theta_i^2}{d^2} + \frac{\theta_j^2}{d^2}} = \frac{1}{c} \cdot \sqrt{\frac{1}{d^2} \cdot (\theta_i^2 + \theta_j^2)} \\ &= \frac{1}{c \cdot d} \cdot \sqrt{\theta_i^2 + \theta_j^2} = \frac{1}{t} \cdot \sqrt{\theta_i^2 + \theta_j^2} \qquad (6)\end{aligned}$$

We perform a preliminary experiment by varying t in the range (2, 10) in order to choose the best values for c and d. Notice that it is not possible to set $t \leqslant 1$, otherwise $\theta_{ij} > max(\theta_i, \theta_j)$, so every edge will be pruned. Furthermore, we limit $t \geqslant 2$ because, in practice, lower values of t yields very poor recall for many of the analyzed datasets.

The results are shown in Figure 13. In general, we observe that the recall increases as t increases, but at the expense of precision. As a trade-off for precision and recall, for all the experiments in this paper, we employ $t = 4$ (setting $c = 2$ and $d = 2$). As a matter of fact, on all the datasets, increasing t above 4 the loss of precision is traded for a little gain in the recall.

5.4. Broadcast vs. Repartition Meta-blocking

The goal of this experiment is to compare the efficiency of *broadcast meta-blocking* (Algorithm 2) and *repartition meta-blocking* (Algorithm 3). Both the algorithms can be employed as core graph-processing algorithms for any meta-blocking method. Thus, we evaluate them in combination with WNP and CNP, in order to analyze how they perform on both family of meta-blocking, i.e., those based on weight-threshold, and those based on cardinality-threshold (see Section 2.3). To minimize additional overhead, we run them in combination with the computationally cheapest weighting function, i.e., *block co-occurrence frequency* (we record analogues trends with other weighting functions). The experiment was performed on 10 nodes. We consider only the large datasets since the overhead introduced by Spark does not pay off on the small ones on multiple nodes. Notice that both algorithms perform the same logical operation, that is the final recall and precision are the same on all the datasets, hence not reported here.

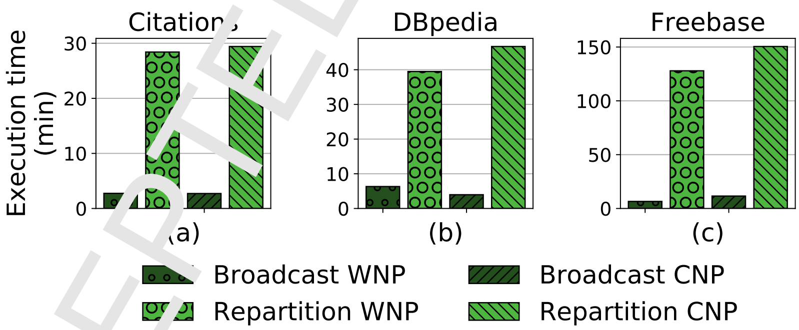


Figure 14: *Repartition* vs. *Broadcast meta-blocking*. For each dataset we report two different strategies for the ***prune*** functions, i.e., the weight- and cardinality-based pruning. This times was taken on 10 nodes.

The results are reported in Figure 14: *broadcast meta-blocking* is faster than *repartition meta-blocking* from 4.9 to 12.7 times for WNP, and from 7.7 to 1[illegible].1 times for CNP. To analyze the scalability of the algorithms, we report in Figure 15 their execution times in function of the number of nodes (from 1 to

10) on `freebase` (the largest dataset). In our setting, *repartition meta-blocking* is not able to run with less than 7 nodes; whereas *broadcast meta-blocking* on a single node is 3 to 4 times faster than the execution time of the *repartition meta-blocking* on 10 nodes.

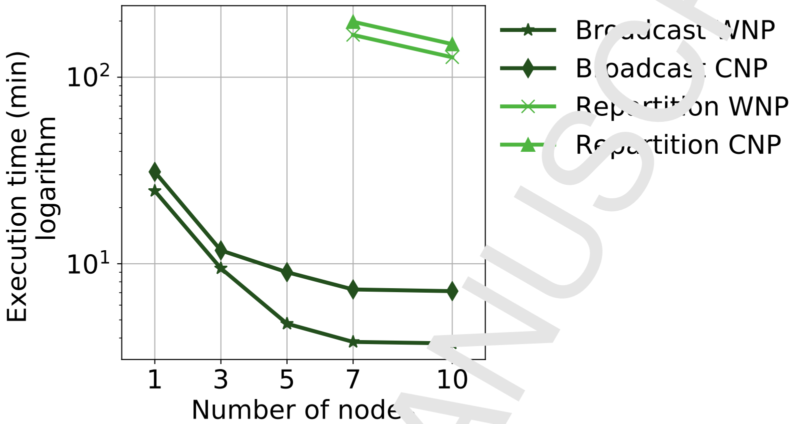


Figure 15: Scalability comparison: *repartition* vs. *broadcast meta-blocking* on `freebase`.

We conclude that the *broadcast meta-blocking* is always faster than the *repartition meta-blocking*.

5.5. Parallel-blast scalability

Finally, we assess the scalability of parallel *Blast* by varying the number of nodes in the cluster (1, 3, 5, 7 and 10 nodes). For this experiment we employ `freebase`, which is the heaviest dataset to process due to the huge number of comparisons yielded by the blocking phase (2.23×10^{13} comparisons), and to its large number of attributes (47,945 distinct attributes).

Figure 17 shows the scalability of each blocking step, i.e.: Loose Schema Blocking (LSB, which is composed of Loose attribute-Match Induction in combination of Token Blocking), and Loose Schema Meta-Blocking (LS-MB). Figure 16 shows the speedup of each blocking step, which is sub-linear to the number of nodes in the cluster (i.e. 10x nodes, the overall speedup do not reach 5). For each step, we observe at least a 50% reduction of execution time from 1 to 3 nodes. Then, the execution times continuously decrease until reaching an overall speedup on 10 nodes of 4.2×.

The time and speedup reported so far only consider the blocking and meta-blocking phase of an ER process. In practice, all the comparisons generated through any blocking process have to be compared by means of an *Entity Resolution Algorithm*, which is a binary function that takes as input two profiles and decides whether or not they are matching [36, 5]. Such a function is typically expensive, e.g., involving string similarity computations, calls to external resources or even human intervention (i.e., crowdsourcing). Thus, the more the employed function is expensive, the more useful a good blocking (and meta-blocking) method is; in other words: the resources saved avoiding superfluous comparisons are proportional to the complexity of the *Entity Resolution Algorithm*. Hence, we now compare *Blast* and WNP using a naïve (i.e., cheap) *Entity Resolution Algorithm* for showing that *Blast* significantly reduce the overall execution time of a complete ER process. We employ as *Entity Resolution Algorithm* the computation of the *Jaccard Similarity* of the two profiles involved in each comparison[19].

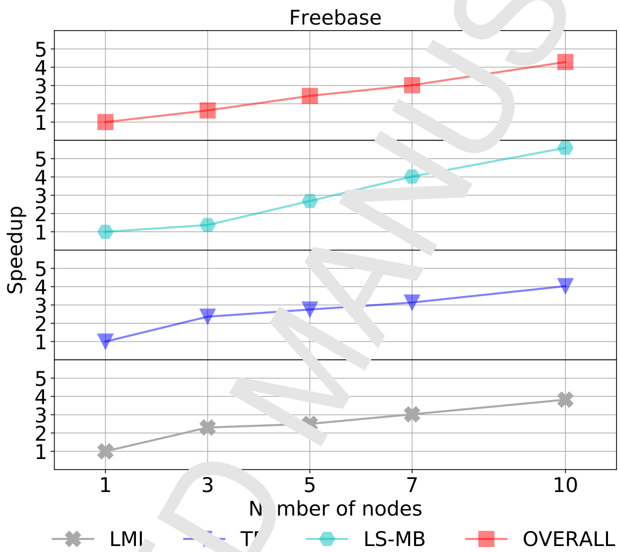


Figure 16: Speedup of *Blast* on `freebase`.

[19] In a real-world scenario, a threshold would be required to discriminate between matching and non-matching pairs.

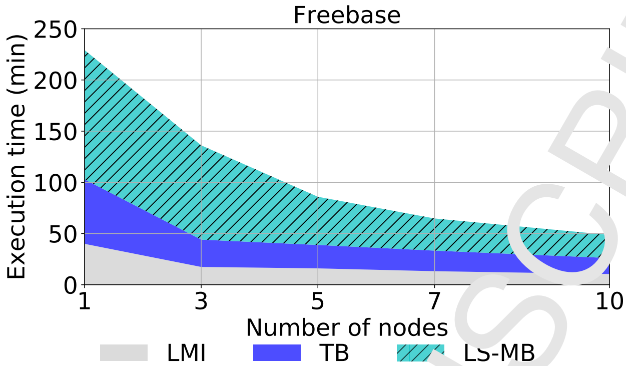


Figure 17: Execution time of *Blast* on `freebase`.

Figure 18 shows the execution time of *Blast* and WNP in combination with the naïve *Entity Resolution Algorithm*[20] and by varying the number of nodes. We observe that the meta-blocking phase of *Blast* is slower than standard schema-agnostic WNP. This is not surprising, since *Blast* performs an additional step compared to WNP (i.e., *Loose Attribute-Match Induction*). Yet, the overall ER process employing *Blast* is significantly faster that employing WNP, since it retains much fewer comparisons ($3.80 \cdot 10^8$ of *Blast* vs. $2.17 \cdot 10^{10}$ of WNP). Please, recall that *Blast* and WNP, on `freebase`, achieve the same recall (Figure 11(g)).

[20]The average comparison time on `freebase` is 0.05 ms.

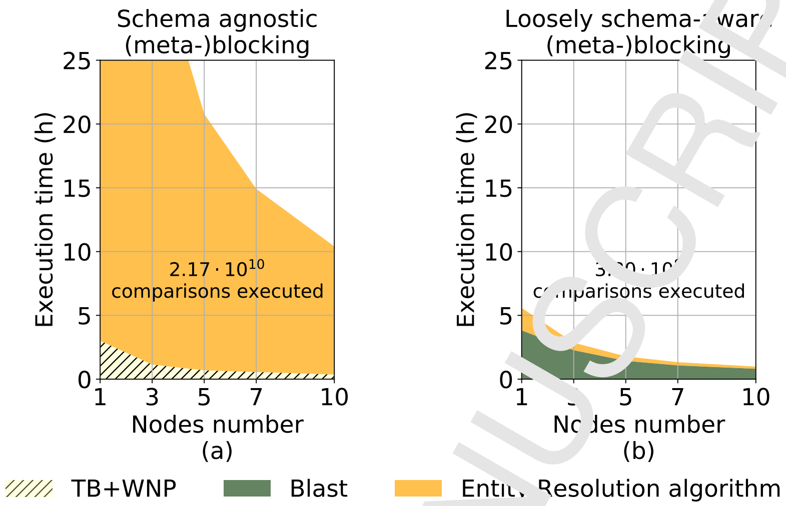


Figure 18: Execution time of the complete ER process on `freebase`, varying the number of execution nodes in the cluster. The whole ER process is composed of a blocking phase, which generates candidate pairs that are compared through an Entity Resolution Algorithm. In (a), the blocking method employed is Token Blocking in combination with WNP meta-blocking. In (b), the blocking method employed is *Blast*.

5.6. LSH-based Loose Schema Blocking

This section aims at assessing the benefit of the LSH-based step. To do that, consider the worst case scenario: when *Loose Schema Blocking* (see Section 3.2) does not identify any similar attribute, all the attributes are grouped in a unique all-encompassing cluster (the *glue cluster* [7]). In this scenario, the blocks generated combining *Loose Schema Blocking* are identical to those generated with Token Blocking alone. On the other hand, if *Loose Schema Blocking* correctly groups some similar attributes, separating them from the glue cluster, the precision of the produced block collection increases, while recall remains almost the same.

Ideally, the more the similar attributes are correctly grouped, the higher the precision of the generated blocks is, without affecting the recall. Hence, to demonstrate the advantage of LSH-based *Loose Schema Blocking*, we perform a set of experiments "disabling" the glue cluster and varying the threshold of LSH. This means that, without the glue cluster, all the attributes that are not indexed in a group of similar attributes are discarded, and so are the tokens of their values. If significant tokens are not employed as blocking key, the recall of the final blocks is negatively affected. So, varying the threshold of LSH changes the group of similar attributes. In fact, if two attributes are less similar[21] than

[21] Jaccard similarity, since we are employing min-hash.

the threshold, *Loose Schema Blocking* does not consider them as a candidate pair, and they cannot be indexed in the same group.

Figure 19 shows how LSH affects the final results of *Blast* combined with *Loose Schema Blocking* in terms of recall on `dbpedia` (other datasets yields analogous resutls). Table 5 reports the execution times of the experiment. We consider the recall of the block collection produced with *Loose Schema Blocking* only, without considering the meta-blocking phase. Basically, up to a threshold value of 0.35 (i.e., Jaccard similarity equals to 0.35), the recall is not affected ($recall = 99.99\%$), meaning that (almost[22]) all the matching profile pairs are successfully indexed in the block collection. The precision is not reported, but for the points where $recall = 99.99\%$ is identical, i.e., it is not affected by the LSH threshold. For a threshold greater than 0.35, on the contrary, the techniques start failing to index some profile pairs, entailing a degradation of the final result. In other words, for thresholds that exclude too many attribute comparisons, *Loose Schema Blocking* fails to recognize similar attributes and produces an incomplete cluster of attributes. Nevertheless, even for a conservative threshold (e.g. 0.10), the execution of *Loose Schema Blocking*, overall, is under 2h (instead of ∼12h).

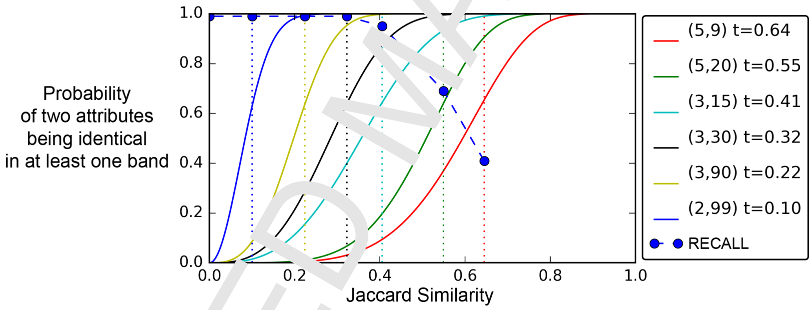


Figure 19: Recall with different LSH configurations in combination with *Loose Schema Blocking* on `dbpedia`. In the legend number of rows and number of bands for LSH are in parenthesis, and t is the estimated threshold.

–	$LSH_{0.10}$	$LSH_{0.22}$	$LSH_{0.32}$	$LSH_{0.41}$	$LSH_{0.55}$	$LSH_{0.64}$
12.5 h	1.9 h	1.5 h	1.3 h	1.2 h	0.9 h	0.7 h

Table 5: *Loose Schema Blocking* run time varying the LSH threshold. The leftmost column reports the execution time of *Loose Schema Blocking* without employing LSH (i.e., computing the Jaccard similarity of all possible pair of attributes).

[22] Loose Schema Blocking (as any other blocking technique) may yield false negative, i.e., pairs of profile that are not indexed in any block; for this reason the recall is not 100%.

	Blast	WNP_{OR}	WNP_{AND}	CNP_{OR}	CNP_{AND}
recall(%)	74.7	78.3	68.3	84.4	78.7
precision(%)	8.90	8.02	11.5	8.8	14.2
F_1	0.1590	0.1448	0.1965	0.1608	0.2361

1k profiles, Ground Truth: 300 matches
(5 attributes - 2 clusters with *Loose Schema Blocking*)
(a) `census`

	Blast	WNP_{OR}	WNP_{AND}	CNP_{OR}	CNP_{AND}
recall(%)	82.1	90.3	81.2	66.9	[illegible]
precision(%)	84.0	53.8	69.4	65.7	[illegible]
F_1	0.8302	0.6726	0.7377	0.6637	[illegible]

1k profiles, Ground Truth: 17k matches
(12 attributes - 4 clusters with *Loose Schema Blocking*)
(b) `cora`

	Blast	WNP_{OR}	WNP_{AND}	CNP_{OR}	CNP_{AND}
recall(%)	93.7	97.3	96.1	[illegible]	94.9
precision(%)	0.13	0.03	0.04	[illegible]	0.18
F_1	0.0027	0.0005	0.0008	0.0015	0.0036

10k profiles, Ground Truth: 600 matches
(106 attributes - 16 clusters with *Loose Schema Blocking*)
(c) `cddb`

Table 6: Dirty ER results.

5.7. Dirty ER

Loose Schema Blocking is designed to identify similar attributes among data sources that have different schemas (e.g. to identify which attributes refers to person names in the example of Figure 1). There is a particular class of Entity Resolution problems, called *dirty ER*, where single data source with known schema is considered, as outlined in [12] (see Section 2.1.1). In this scenario, there is inherently no need to perform loose attribute-match induction (or schema-alignment), because there is only a single source involved that has a unique schema. However, grouping similar attributes (if any) and extracting aggregate entropy is possible; thus, we modified *Loose Schema Blocking* to work with dirty ER (see Section 2.1.1). For the meta-blocking phase, there is no need for changes.

To evaluate the performance of *Blast* we compared it against traditional meta-blocking techniques on 3 real-world benchmark datasets [1]. Both *Blast* and traditional meta-blocking are applied in combination with *Loose Schema Blocking*[23].

[23] Traditional meta-blocking in combination with Token Blocking has always worse performances, thus we do not report here the results. The execution times for these datasets are of the order of milliseconds and *Loose Schema Blocking* does not significantly affect the total execution times.

	#profiles	#attributes
IMDB	6.4k	12
Rotten	7.3k	16
Amazon	5.3k	6
TMD	10k	5

	#duplicates
Amazon-TMD	760
Amazon-Rotten	5
Amazon-IMDB	2
IMDB-Rotten	876
IMDB-TMD	53
TMD-Rotten	[illegible]2

Table 7: Dataset characteristics: number of entity profiles, and number of *attribute names*. On the right side, the number of duplicates between each dataset.

Results

The characteristics of the datasets and the results are listed in Table 6. Besides recall and precision, we also consider F_1-score, which is the harmonic means of the two. This helps us to discuss the comparison of two methods that show significantly different values of both recall and precision. *Blast* achieves higher precision and F_1-score than traditional WNP, and a slightly lower recall.

The only exception is on `cora`, where WNP_{OR} achieves ~8% higher recall (though *Blast* has a ~30% higher precision). Compared to CNP, *Blast* outperforms CNP_{OR} on `cora` and `cddb`, while falls behind it on `census`. On `census` and `cddb`, CNP_{AND} outperforms *Blast*, but in `cora` its recall is considerably low (46%).

Overall, for dirty ER, *Blast* can be an effective blocking technique when the priority is to achieve high precision, without giving up a high level of recall (e.g., to save computational resources performing ER in a cloud-computing environment).

5.8. Multiple Data Sources

In this experiment we want to explore the multi-data source scenario [18], i.e. when the number of input datasets is greater than 2.

The datasets employed in this experiment have been collected from the Magellan repository [31], in particular we consider a collection of heterogeneous records gathered online from `IMDB.com`, `RottenTomatoes.com`, `TMDmoviez.com` and `Amazon.com`, all about movies. These datasets have been used for evaluating ER algorithms in [5]. Considered singularly, none of these datasets contains duplicates; thus, this ER problem can be formalized as a *Clean-Clean ER* problem (a.k.a. *Record Linkage*) [14, 12] (see *Clean-Clean ER* in Section 2.1.1). Thus, *Blast* and meta-blocking can be employed without any modification for this experiment. Notice that if each dataset considered singularly could contain duplicates, the overall problem can be reduced to a *Dirty ER* problem (see Section 2.1.1) on a single dataset that is the union of all the considered datasets [12].

The datasets characteristics are reported in Table 7. All the considered datasets have different schemas [5]. The ground truth has been generated using the Magellan framework [5], the number of identified duplicates between each dataset are reported in Table 7.

Figure 20 reports the achieved results. *Blast* obtains better results both in term of recall and precision w.r.t the standard meta-blocking (Figure 20(a-b)).

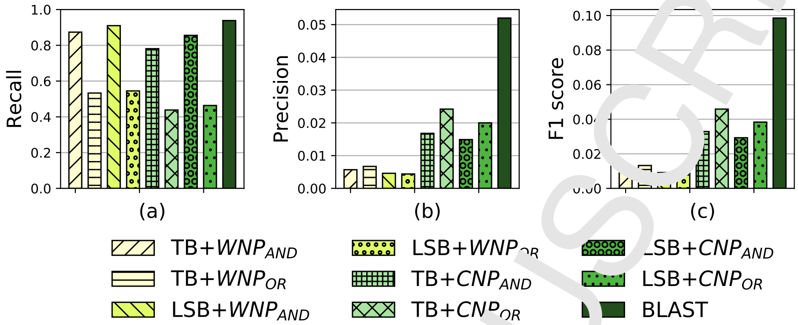


Figure 20: Recall and precision achieved by the considered methods on the multi-source datataset.

6. Related Work

Blocking techniques have been commonly employed in Entity Resolution (ER) [37, 38, 14, 5, 39, 40, 41, 42, 43], and can be classified into two broad categories: *schema-based* (Suffix Array [22], q-grams blocking [44], Canopy Clustering [45]), and *schema-agnostic* (Token Blocking [7], Progressive ER [16, 46, 47, 48, 49, 50], and *Attribute-match induction* [7, 9]).

Attribute-match induction—Among the schema-agnostic techniques, *Attribute Clustering* (AC) [7] and *TYPiMatch* [9] try to extract statistics to define efficient blocking keys. AC relies on the comparison of all possible pairs of attribute profiles of two datasets to find the pairs of those most similar; this is a inefficient process, because the vast majority of comparisons are superfluous. Our LSH-based preprocessing step aims to address this specific issue. *TYPiMatch* tries to identify the latent *subtypes* from generic attributes (e.g. description, info, etc.) frequent on generic dataset on the Web, and uses this information to select blocking keys, but it cannot efficiently scale to large dataset.

Block manipulation—In this paper, we tackled the problem of meta-blocking, i.e., how to restructure (*manipulate*) an existing blocking collection, for improving the quality of the overall ER process. The state-of-the-art, *unsupervised* and schema-agnostic meta-blocking has been presented in [12]. *Blast* was shown to outperform them in Section 5. *Supervised* meta-blocking [51, 52] extends the blocking graph model by representing each edge as a vector of schema-agnostic features (e.g. graph topological measures), and treats the problem of identifying most promising edge as a *classification* problem; hence, a training set of labeled data (matching/non-matching pairs) is required. *Blast* exploits the *loose schema information* and does not require any training set (i.e., it is completely unsupervised).

Recently, in the context of multi data-source ER, Ranbaduge et al. [18] have proposed a blocking manipulation method for identifying entities whose profiles span among g data sources, where g is a user bounded parameter. In order to do that, given a block collection, the proposed method selects and combines (*manipulate*) blocks that are the most promising for finding profiles of g data sources that match together. The user can also specify a set F of data sources, and the final result is required to have matches that involves that set F of data sources. In [18], this task is called Multidatabase record linkage (MDRL). Formalizing MDRL by employing the *blocking graph* model (Section 2.3): MDRL is the task of identifying the hyperedges of the blocking graph that span among g nodes that belong to g distinct sources, and that are have high weight (remember that the weight in the blocking graph corresponds to the matching likelihood). Hence, the scope of MDRL is orthogonal to the scope of meta-blocking [12] (and thus *Blast*), which tries to prune edges that correspond to not-promising comparisons. Furthermore, the existing MDRL solution [18] has been applied only in the context of structured data sources with well known schemas; while *Blast* does not require a predefined schema (since it relies on the loose schema information). Thus, the combination of the two methods is not trivial, but it is surely a future direction that we aim to explore, since the promising results achieved by *Blast* in the multi source scenario of Section 5.8 (where the g and F parameters are not considered).

Metadata exploitation—There is excellent related work in the semantic Web community [17, 53, 54, 55]. For instance, LIMES [53] (an ER approach for the Web of Data), and LOV [54] (a system attempting to standardize vocabularies) propose techniques to exploit metadata, which may also be valuable to our problem, but are orthogonal to our approach. In fact, *Blast* addresses the blocking problem based purely on the attribute values, without considering the semantics of the schema at all.

Entity Resolution with MapReduce-like Systems—Parallel and distributed versions of traditional (schema-based) blocking techniques have been extensively studied in [56, 57]. Altowim and Mehrota [58] have investigated how to generate candidate profile pairs on MapReduce-like systems in *pay-as-you-go* (i.e., progressive) fashion. Their proposed solution relies on the definition of schema-based blocking keys. Finally, Efthymiou et al. [13] have proposed the *repartition meta-blocking* algorithm to run graph-based meta-blocking methods on MapReduce. In Sections 4 and 5, we extensively compare it against our proposed *broadcast meta-blocking* algorithm.

Araújo et al. [59] have proposed a novel schema-agnostic pruning strategy called *Global Weighted Node Pruning* (GWPN) that combines a local threshold with a global one. The local threshold is computed for each profile as for the WNP, while the global one is computed as the average of all the edges weights. This strategy aims to discard the edges with a low weight that connects only profiles with a very low local threshold. Compared to traditional WNP, GWNP improves precision of 0.01%, while achieving the same recall, on DBpedia dataset [59]. Araújo et al. also discuss a Spark implementation for their strategy, which is based on the MapReduce parallel meta-blocking proposed in [13], and suffers from the same limitations (see Section 4.2.2).

7. Conclusion and Future Work

In this paper we presented a holistic (meta-)blocking approach, *Blast*, able to automatically collect and exploit *loose schema information* (i.e., statistics gathered directly from the data for approximately describing the data source schemas). We explained how this loose schema information can be extracted efficiently even from highly heterogeneous and voluminous datasets through an LSH-based step. We proposed a novel algorithm for efficiently running any meta-blocking technique on MapReduce-like Systems. Finally, we experimentally evaluated it on real-world datasets. The experimental results showed that: *(i) Blast* outperforms the existing state-of-the-art meta-blocking approaches in terms of quality of the results; *(ii)* our *broadcast meta-blocking* is always faster than the existing state-of-the-art when leveraging on distributed and parallel computation of MapReduce-like Systems.

Relevant research problem can be explored as future directions: in the context of multi-data source ER, we aim to investigate how to combine our Loose-Schema Aware (meta-)blocking method with MDRL solution [18] (presented in Section 6). In the context of progressive ER (a.k.a. pay-as-you-go ER) [47], we aim to investigate how to exploit broadcast meta-blocking to yield progressive results, maximizing the recall on the basis of a limited resource budget (e.g., limited execution time, and/or computational resources). Finally, we are planning to combine our blocking technique for scaling to large data set advanced similarity functions that leverage on external knowledge bases, such as [60], with other MapReduce-like systems [61] and on real-world applications, such as the deduplication of web pages tags [62].

[1] P. Christen, A survey of indexing techniques for scalable record linkage and deduplication, IEEE transactions on knowledge and data engineering 24 (9) (2012) 1537–1555.

[2] X. L. Dong, D. Srivastava, Big data integration, Synthesis Lectures on Data Management 7 (1) (2015) 1–198.

[3] S. Bergamaschi, D. Beneventano, F. Mandreoli, R. Martoglia, F. Guerra, M. Orsini, L. Po, M. Vincini, G. Simonini, S. Zhu, et al., From data integration to big data integration, in: A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years, Springer, 2018, pp. 43–59.

[4] R. Baxter, P. Christen, T. Churches, et al., A comparison of fast blocking methods for record linkage, in: ACM SIGKDD, Vol. 3, Citeseer, 2003, pp. 25–27.

[5] P. Konda, S. Das, P. Suganthan GC, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, et al., Magellan: Toward building entity matching management systems, Proceedings of the VLDB Endowment 9 (12) (2016) 1197–1208.

[6] J. Madhavan, S. R. Jeffery, S. Cohen, X. Dong, D. Ko, C. Yu, A. Halevy, Web-scale data integration: You can only afford to pay as you go, in: Proceedings of CIDR, 2007, pp. 342–350.

[7] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, W. Nejdl, A blocking framework for entity resolution in highly heterogeneous information spaces, IEEE Transactions on Knowledge and Data Engineering 25 (12) (2013) 2665–2682.

[8] G. Papadakis, G. Koutrika, T. Palpanas, W. Nejdl, Meta-blocking: Taking entity resolution to the next level, IEEE Transactions on Knowledge and Data Engineering 26 (8) (2014) 1946–1960.

[9] Y. Ma, T. Tran, Typimatch: Type-specific unsupervised learning of keys and key values for heterogeneous web data integration, in: Proceedings of the sixth ACM international conference on Web search and data mining, ACM, 2013, pp. 325–334.

[10] C. E. Shannon, A mathematical theory of communication, SIGMOBILE Mob. Comput. Commun. Rev. 5 (1) (2001) 3–55. doi:10.1145/584091.584093.

[11] G. Simonini, S. Bergamaschi, H. Jagadish, Blast: a loosely schema-aware meta-blocking approach for entity resolution, Proceedings of the VLDB Endowment 9 (12) (2016) 1173–1184.

[12] G. Papadakis, G. Papastefanatos, T. Palpanas, M. Koubarakis, Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking., in: EDBT, 2016, pp. 221–232.

[13] V. Efthymiou, G. Papadakis, G. Papastefanatos, K. Stefanidis, T. Palpanas, Parallel meta-blocking for scaling entity resolution over big heterogeneous data, Information Systems 65 (2017) 137–157.

[14] P. Christen, Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection, Data-Centric Systems and Applications, Springer, 2012. doi:10.1007/978-3-642-31164-2.

[15] V. Christophides, V. Efthymiou, K. Stefanidis, Entity resolution in the web of data, Synthesis Lectures on the Semantic Web 5 (3) (2015) 1–122.

[16] G. Simonini, G. Papadakis, T. Palpanas, S. Bergamaschi, Schema-agnostic progressive entity resolution, IEEE Trans. Knowl. Data Eng. (2018)doi:10.1109/TKDE.2018.2852763.

[17] P. Shvaiko, J. Euzenat, Ontology matching: state of the art and future challenges, IEEE Transactions on knowledge and data engineering 25 (1) (2013) 158–176.

[18] T. Ranbaduge, D. Vatsalan, P. Christen, A scalable and efficient subgroup blocking scheme for multidatabase record linkage, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2018, pp. 15–27.

[19] A. Z. Broder, On the resemblance and containment of documents, in: Compression and complexity of sequences 1997. proceedings, IEEE, 1997, pp. 21–29.

[20] J. Leskovec, A. Rajaraman, J. D. Ullman, Mining of massive datasets, Cambridge university press, 2014.

[21] T. M. Cover, J. A. Thomas, Elements of information theory, John Wiley & Sons, 2012.

[22] T. De Vries, H. Ke, S. Chawla, P. Christen, Robust record linkage blocking using suffix arrays, in: Proceedings of the 18th ACM conference on Information and knowledge management, ACM, 2009, pp. 305–314.

[23] A. Agresti, M. Kateri, Categorical data analysis, in: International encyclopedia of statistical science, Springer, 2011, pp. 206–208.

[24] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), USENIX, San Jose, CA, 2012, pp. 15–28.

[25] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.

[26] [link].
URL `https://spark.apache.org/docs/2.1.0/programming-guide.html#shuffle-operations`

[27] A. S. Das, M. Datar, A. Garg, S. Rajaram, Google news personalization: scalable online collaborative filtering, in: Proceedings of the 16th international conference on World Wide Web, ACM, 2007, pp. 271–280.

[28] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, Y. Tian, A comparison of join algorithms for log processing in mapreduce, in: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM, 2010, pp. 975–986.

[29] [link].
URL `http://stravanni.github.io/blast/`

[30] H. Köpcke, A. Thor, E. Rahm, Evaluation of entity resolution approaches on real-world match problems, Proceedings of the VLDB Endowment 3 (1-2) (2010) 484–493.

[31] S. Das, A. Doan, P. S. G. C., C. Gokhale, P. Konda, The magellan data repository, `https://sites.google.com/site/anhaidgroup/projects/data`.

[32] A. Harth, Billion triples challenge data set (2012).

[33] D. Hand, P. Christen, A note on using the f-measure for evaluating record linkage algorithms, Statistics and Computing 28 (3) (2018) 539–547.

[34] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, N. Tang, Distributed representations of tuples for entity resolution, Proceedings of the VLDB Endowment 11 (11) (2018) 1454–1467.

[35] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, Deep learning for entity matching: A design space exploration, in: Proceedings of the 2018 International Conference on Management of Data, ACM, 2018, pp. 19–34.

[36] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, J. Widom, Swoosh: a generic approach to entity resolution, The VLDB Journal—The International Journal on Very Large Data Bases 18 (1) (2009) 255–276.

[37] H. Köpcke, E. Rahm, Frameworks for entity matching: A comparison, Data & Knowledge Engineering 69 (2) (2010) 197–210.

[38] F. Naumann, M. Herschel, An Introduction to Duplicate Detection, Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2010. `doi:10.2200/S00262ED1V01Y201003DTM003`.

[39] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, S. Xu, Data curation at scale: The data tamer system, in: CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings, 2013.

[40] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, M. Koubarakis, The return of jedai: End-to-end entity resolution for structured and semi-structured data, PVLDB 11 (12) (2018) 1950–1953. `doi:10.14778/3229863.3236232`.

[41] V. Efthymiou, G. Papadakis, K. Stefanidis, V. Christophides, Simplifying entity resolution on web data with schema-agnostic, non-iterative matching, in: 34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018, 2018, pp. 1296–1299.
URL `https://doi.org/10.1109/ICDE.2018.00134`

[42] A. D. Sarma, A. Jain, A. Machanavajjhala, P. Bohannon, CBLOCK: an automatic blocking mechanism for large-scale de-duplication tasks, CoRR abs/1111.3689. arXiv:1111.3689.
URL http://arxiv.org/abs/1111.3689

[43] U. Draisbach, F. Naumann, A generalization of blocking and windowing algorithms for duplicate detection, in: 2011 International Conference on Data and Knowledge Engineering, ICDKE 2011, Milano, Italy, September 6, 2011, 2011, pp. 18–24. doi:10.1109/ICDKE.2011.6053920.

[44] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, D. Srivastava, Approximate string joins in a database (almost) for free, in: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy, 2001, pp. 491–500.

[45] A. McCallum, K. Nigam, L. H. Ungar, Efficient clustering of high-dimensional data sets with application to reference matching, in: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000, 2000, pp. 169–178. doi:10.1145/347090.347123.

[46] G. Simonini, G. Papadakis, T. Palpanas, S. Bergamaschi, Schema-agnostic progressive entity resolution, in: 34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018, 2018, pp. 53–64. doi:10.1109/ICDE.2018.00015.

[47] S. E. Whang, D. Marmaros, H. Garcia-Molina, Pay-as-you-go entity resolution, IEEE Trans. Knowl. Data Eng. 25 (5) (2013) 1111–1124. doi:10.1109/TKDE.2012.43.

[48] T. Papenbrock, A. Heise, F. Naumann, Progressive duplicate detection, IEEE Trans. Knowl. Data Eng. 27 (5) (2015) 1316–1329. doi:10.1109/TKDE.2014.2359666.

[49] D. Firmani, B. Saha, D. Srivastava, Online entity resolution using an oracle, PVLDB 9 (5) (2016) 384–395. doi:10.14778/2876473.2876474.
URL http://www.vldb.org/pvldb/vol9/p384-firmani.pdf

[50] D. Firmani, S. Galhotra, B. Saha, D. Srivastava, Robust entity resolution using a crowdoracle, IEEE Data Eng. Bull. 41 (2) (2018) 91–103.
URL http://sites.computer.org/debull/A18june/p91.pdf

[51] G. Papadakis, G. Papastefanatos, G. Koutrika, Supervised meta-blocking, PVLDB 7 (14) (2014) 1929–1940. doi:10.14778/2733085.2733098.

[52] G. Dal Bianco, M. A. Gonçalves, D. Duarte, Bloss: Effective meta-blocking with almost no effort, Information Systems 75 (2018) 75–89.

[53] A. N. Ngomo, S. Auer, LIMES - A time-efficient approach for large-scale link discovery on the web of data, in: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, 2011, pp. 2312–2317. doi:10.5591/978-1-57735-516-8/IJCAI11-385.

[54] P. Vandenbussche, B. Vatant, Linked open vocabularies, ERCIM News 2014 (96).

[55] S. Bergamaschi, D. Ferrari, F. Guerra, G. Simonini, Y. Velegrakis, Providing insight into data source topics, J. Data Semantics 5 (4) (2016) 211–228. doi:10.1007/s13740-016-0063-6.

[56] L. Kolb, A. Thor, E. Rahm, Dedoop: Efficient deduplication with hadoop, PVLDB 5 (12) (2012) 1878–1881. doi:10.14778/2367502.2367527.

[57] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, Y. Park, Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services, in: Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017, 2017, pp. 1431–1446. doi:10.1145/3035918.3035960.

[58] Y. Altowim, S. Mehrotra, Parallel progressive approach to entity resolution using mapreduce, in: 33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017, 2017, pp. 909–920. doi:10.1109/ICDE.2017.139.

[59] T. B. Araújo, C. E. S. Pires, T. P. da Nóbrega, Spark-based streamlined metablocking, in: Computers and Communications (ISCC), 2017 IEEE Symposium on, IEEE, 2017, pp. 844–850.

[60] F. Benedetti, D. Beneventano, S. Bergamaschi, G. Simonini, Computing inter-document similarity with context semantic analysis, Inf. Syst. 80 (2019) 136–147. doi:10.1016/j.is.2018.02.009.

[61] S. Bergamaschi, L. Gagliardelli, G. Simonini, S. Zhu, Bigbench workload executed by using apache flink, Procedia Manufacturing 11 (2017) 695–702.

[62] F. Guerra, G. Simonini, M. Vincini, Supporting image search with tag clouds: A preliminary approach, Adv. in MM 2015 (2015) 439020:1–439020:10. doi:10.1155/2015/439020.

Highlights

- An unsupervised graph-based meta-blocking approach (called Blast) able to leverage this loose schema information;
- an LSH-based attribute-match induction technique for efficiently scale to large datasets with a high number of attributes;
- an algorithm to efficiently run Blast (and any other graph-based meta-blocking method) on MapReduce-like systems, to take full advantage of a parallel and distributed computation;
- the evaluation of our approach on seven real-world datasets, showing how Blast outperforms the state-of-the-art meta-blocking methods.

ACCEPTED MANUSCRIPT