BLAST: a Loosely Schema-aware Meta-blocking Approach for Entity Resolution

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

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

H.V. Jagadish
Univ. of Michigan, Ann Arbor
jag@umich.edu

ABSTRACT

Identifying records that refer to the same entity is a fundamental step for data integration. Since it is prohibitively expensive to compare every pair of records, blocking techniques are typically employed to reduce the complexity of this task. These techniques partition records into blocks and limit the comparison to records co-occurring in a block. Generally, to deal with highly heterogeneous and noisy data (e.g. semi-structured data of the Web), these techniques rely on redundancy to reduce the chance of missing matches.

Meta-blocking is the task of restructuring blocks generated by redundancy-based blocking techniques, removing superfluous comparisons. Existing meta-blocking approaches rely exclusively on schema-agnostic features.

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 this loose information by adopting a LSH-based step for efficiently scaling to large datasets. We experimentally demonstrate, on real-world datasets, how Blast outperforms the state-of-the-art unsupervised meta-blocking approaches, and, in many cases, also the supervised one.

1. INTRODUCTION

The Web has become a valuable source of structured and semi-structured data exponentially growing [3, 8]. The true potential of this data is expressed when different sources are integrated, as demonstrated by recent efforts in mining the web to extract entities, relationships, and ontologies to build large-scale general purpose knowledge bases, such as Freebase[1] and Yago[2] [8]. For enterprises, government agencies, and researchers this data can be even more valuable if integrated with their proprietary data.

[1]http://www.freebase.com/
[2]http://www.mpi-inf.mpg.de/YAGO/

Proceedings of the VLDB Endowment, Vol. 9, No. 12
Copyright 2016 VLDB Endowment 2150-8097/16/08.

One fundamental step in data integration is *Entity Resolution* (ER), namely the task of matching records (the entity *profiles*) from several data sources (the entity collections) that refer to the same real-world entity [5]. Comparing all possible pairs of profiles of an entity collection is inherently a quadratic problem: if the number of profiles grows linearly, then the number of possible comparison grows quadratically. Therefore, a *brute-force* approach becomes infeasible for very large datasets. For this reason, indexing techniques are widely employed to group similar profiles into *blocks* and execute the comparisons only among those appearing in the same block.

Blocking Background: Traditional *schema-based blocking* techniques generate blocks according to a blocking criterion (*blocking key*), either based on a single attribute, or a combination of attributes [5]. They suffer from two well-known issues [17]. Firstly, selecting which attributes to combine (and how) to define a good blocking is a difficult and error-prone task that generally requires domain experts. Alternatively, a classification algorithm can be employed to this end, but the need of labeled data is limiting. Secondly, if two datasets have different schemas, a schema alignment between the data sources must be executed before ER. Unfortunately, data on the Web is typically highly heterogeneous, noisy (missing/inconsistent data), and very large in volume, thus making traditional schema alignment techniques no longer applicable [13, 18]. 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 impracticable [13].

To solve these issues, *schema-agnostic* blocking approaches have been proposed [18, 12, 19, 17]. These approaches are completely unsupervised and do not use any schema information at all. The most general schema-agnostic technique is Token Blocking [17, 18]. It considers each token appearing in the dataset as a blocking key. Thus, each block is associated to a token and contains all the profiles in which that token appears (regardless of the attribute in which it appears) as shown in the example of Figure 1a-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).

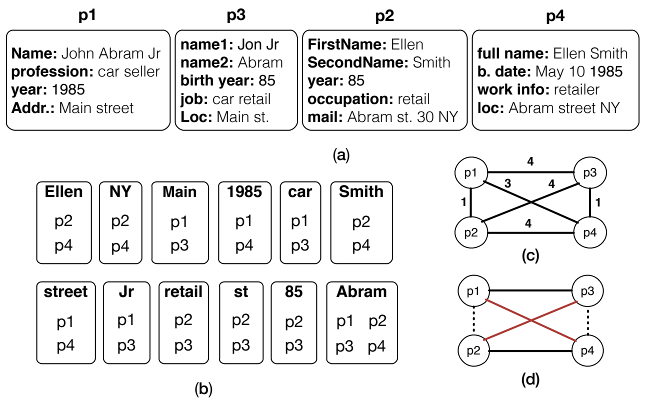

Figure 1: (a) A collection of entity profiles from four different data sources. (b) A block collection produced with Token Blocking. (c) The derived blocking graph. (d) The restructured blocking graph; dashed lines represent pruned edges, and red ones the superfluous comparisons not removed.

To overcome this issue, *meta-blocking* approaches have been proposed [19, 20]. *Meta-blocking* is the task of restructuring a set of blocks to retain only the most promising comparisons. *Unsupervised graph-based* meta-blocking [20] represents a block collection as a weighted graph, called *blocking graph* (see the example in Figure 1c), where 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; e.g., in Figure 1c the weight is the number of co-occurrences of profiles in the blocks, but more sophisticated weighting function can be employed. 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 1d). At the end of the process, each pair of nodes connected by an edge forms a new block. Differently, *supervised graph-based* meta-blocking [19] associates to each edge a vector of schema-agnostic features (e.g. graph topological measures), and treats the problem of identifying promising edge as a *classification* problem; hence, a training set of labeled data (matching/non-matching pairs) is required.

Our Contribution: We observe that existing meta-blocking techniques exclusively leverage schema-agnostic features extracted from the target block collection. Thus, inspired by the *attribute-match induction* approaches [18, 12], our idea is to exploit schema 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.

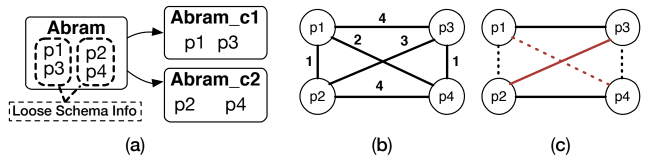

Figure 2: Blocking key disambiguation effect on meta-blocking.

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 "Abram" is divided into two new blocks (Figure 2a) affecting the blocking graph: the weights of the edges $e_{P_1-P_4}$ and $e_{P_2-P_3}$ both decrease (Figure 2b). 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 2c). 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 generate more significant blocking keys. Blast measures the information content of an attribute through the Shannon entropy. Then, it derives an *aggregate entropy* measure for each cluster of attributes (Figure 3a). Finally, it uses these values as a multiplicative factor for the weights of the blocking graph (Figure 3b). The final blocking graph after the pruning phase is showed in Figure 3c[3]. The superfluous edge $e_{P_2-P_3}$ has now been correctly removed.

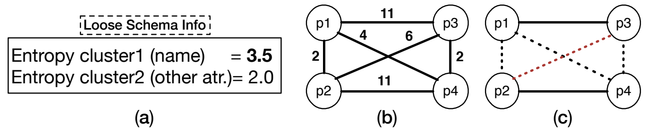
Loose Schema Info
Entropy cluster1 (name) = 3.5
Entropy cluster2 (other atr.)= 2.0
(a)
p1 p3 p2 p4
11 4 6 2 2 11
(b)
(c)

Figure 3: Attribute entropy information effect on meta-blocking.

We make the following main contributions in this paper:

- 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;
- a novel LSH-based pre-processing step to efficiently scale attribute-match induction to datasets with a high number of attributes;

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

- the evaluation of our approach on real-world datasets, showing how BLAST outperforms the state-of-the-art unsupervised meta-blocking and, in many cases, also the supervised ones.

Organization: The remainder of this paper is structured as follows. Section 2 provides preliminaries. Section 3 describes BLAST. Section 4 presents the datasets, the evaluation metrics, and the experiments. Section 5 reviews the related work. Finally, Section 6 concludes the paper.

2. PRELIMINARIES

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

An *entity profile* is a tuple composed of a unique identifier and a set of *name-value* pairs $\langle a, v\rangle$. $A_{\mathcal{E}}$ is the set of possible attributes a associated to an entity collection $\mathcal{E}$. An *entity collection* $\mathcal{E}$ is a set of profiles. Two profiles $p_i, p_j \in \mathcal{E}$ 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 $\mathcal{E}$.

There exist two kinds of ER [19]: *clean-clean* ER and *dirty* ER. The former takes as input two duplicate-free entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$ and compares pairs $\{(p_i, p_j) \mid p_i \in \mathcal{E}_1, p_2 \in \mathcal{E}_2\}$; the latter takes as input a single collection $\mathcal{E}_s$ containing duplicates and compares all possible pairs of profiles. The naive solutions to clean-clean and dirty ER imply respectively $|\mathcal{E}_1| \times |\mathcal{E}_2|$ and $\binom{|\mathcal{E}_s|}{2}$ comparisons, where $|\mathcal{E}_i|$ is the cardinality of an entity collection $\mathcal{E}_i$. *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. 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.

In this paper we follow best practices to establish the quality of a block collection [18, 20, 19]: the problem of determining if two profiles actually refer to the same real-world object is the task of the Entity Resolution Algorithm. In our work, we assume there is an *entity resolution 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 [19, 20].

Metrics: We employ *Pair Completeness* (*PC*) and *Pair Quality* (*PQ*) [5] to evaluate the quality of a block collection $\mathcal{B}$, which are surrogates of *recall* and *precision*, respectively. $PC(\mathcal{B})$ measures the portion of duplicate profiles that are placed in at least one block; while $PQ(\mathcal{B})$ measures the portion of useful comparison, i.e., those that detect a match. We also consider the F_1-score [5], useful to compare block collections that present different values of both PC and PQ. Formally:

$$PC(\mathcal{B}) = \frac{|\mathcal{D}^{\mathcal{B}}|}{|\mathcal{D}^{\mathcal{E}}|}; \qquad PQ(\mathcal{B}) = \frac{|\mathcal{D}^{\mathcal{B}}|}{\|\mathcal{B}\|};$$

$$F_1(\mathcal{B}) = 2 \cdot \frac{PC(\mathcal{B}) \cdot PQ(\mathcal{B})}{PC(\mathcal{B}) + PQ(\mathcal{B})}$$

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

Typically, schema-agnostic blocking yields high PC, but at the expense of PQ. The low PQ 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 ($p_i \not\approx p_j$).

Attribute-match induction[4] 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.

2.1 Attribute-match Induction

The goal of attribute-match induction is to induce groups of *similar* attributes between two entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$ 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 to disambiguate blocking keys according to the attribute group from which they are derived (e.g. tokens "Abram" in Figure 1b).

DEFINITION 1. ATTRIBUTE-MATCH INDUCTION. *Given two entity collections $\mathcal{E}_1, \mathcal{E}_2$, attribute-match induction is the task of identifying pairs $\{\langle a_i, a_j\rangle \mid a_i \in A_{\mathcal{E}_1}, a_j \in A_{\mathcal{E}_2}\}$ 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 $(A_{\mathcal{E}_1} \times A_{\mathcal{E}_2})$ 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 [21].

The partitioning of the attribute name space is based on four components: (i) the *value transformation function* (ii) the *attribute representation model*, (iii) the *similarity measure* to match attributes, and (iv) the *clustering algorithm.*

- **The value transformation function**. Given two entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$, each attribute is treated as a tuple $\langle a_j, \tau(V_{a_j})\rangle$, where $a_j \in A_{\mathcal{E}_i}$ is an attribute name, and τ is a *value transformation function* returning the set of *terms* derived from the values V_{a_j} that an attribute a_j can assume in $\mathcal{E}_i$. 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 entity collections is $T_A = T_{a_{\mathcal{E}_1}} \bigcap T_{a_{\mathcal{E}_2}}$, where $T_{a_{\mathcal{E}}} = \bigcup_{a_i \in A_{\mathcal{E}}} \tau(V_{a_i})$.
- **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. While, if $t_n \in \tau(V_{a_i})$, then v_{in} assumes a value computed employing a weighting function, such as [18]: $TF\text{-}IDF(t_n)$ 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

[4]We call *attribute-match induction* the general approach to group similar attributes, while we refer to the specific technique proposed in [18] with *Attribute Clustering.*

to the possible tokens appearing in the entity collections; and each element v_{in} is either 1 (if the token t_n appear in the attribute a_i) or 0 (otherwise).

• **The similarity measure**. For each possible pair of attributes $(a_j, a_k) \in (A_{\mathcal{E}_1} \times A_{\mathcal{E}_2})$, 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-IDF* weighting.

• **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. Its output is called *attributes partitioning*.

2.2 Meta-blocking

The goal of meta-blocking [20] 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 PQ and nearly identical PC, i.e.: $PQ(\mathcal{B}') \gg PQ(\mathcal{B})$ and $PC(\mathcal{B}') \simeq PC(\mathcal{B})$.*

In graph-based meta-blocking, 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 $p_i \in \mathcal{E}$. An edge between two entity profiles exists if they appear in at least one block together: $E = \{e_{ij} : \exists p_i, p_j \in \mathcal{E} \mid |\mathcal{B}_{ij}| > 0\}$ 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. The weights capture the likelihood of a match; this is at the base of the edge pruning strategies employed to retain only more promising comparisons. At the end of the pruning, each pair of nodes connected by an edge forms a new block. Meta-blocking inherently prevents redundant comparisons, since two profiles can appear together in the final block collection at most once.

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. Both pruning criteria can be applied either *locally* or *globally*. In the first case, the *top-k* edges and the weight threshold θ are computed and applied in a *node-centric* manner, i.e., for each node and its adjacent edges; while in the second case, the *top-K* edges are selected among the whole set of edges, and the threshold Θ is unique for all the edges.

The combination of those characteristics leads to four possible *pruning schemas*. *Weight Edge Pruning* (*WEP*) discards all the edges with weight lower than Θ. *Cardinality Edge Pruning* (*CEP*) sorts all the edges by their weights in descending order, and retains only the first K. *Weight Node Pruning* (*WNP*) considers in turn each node n_i and its adjacent edges, and prunes those edges that are lower than a local threshold θ_i. *Cardinality Node Pruning* (*CNP*) 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).

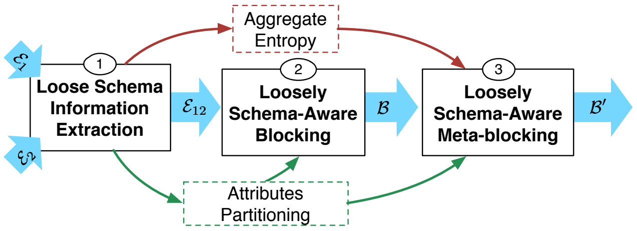

Figure 4: Blast logical overview.

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; and to perform a holistic combination of blocking ad meta-blocking for clean-clean Entity Resolution exploiting this loose schema information. Nevertheless, Blast can be adapted to work with dirty ER as well (as we experimentally show in Section 4.5).

Our approach takes as input two Entity 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*.

(***Phase 1***) The loose schema information is extracted. It consists of: the *attributes partitioning*, and the *aggregate-entropy*. The former describes how the attributes are partitioned according to the similarity of their values; it is the result of the *attribute-match induction* task (Section 2.1). The latter is a measure associated to each cluster of attributes, derived from the attribute entropies. We also introduce a *Locality-Sensitive Hashing* (LSH) [4] 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*. In particular, Blast employs Token Blocking. An example is shown in Figure 2: disambiguating the token "Abram" as a person-name/street-name saves one superfluous comparison during the meta-blocking phase (Figure 2c).

(***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.

In the following we describe in detail the three phases.

3.1 Loose Schema Information Extraction.

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 [18].

3.1.1 Loose Attribute-match Induction

Following the definitions of Section 2.1, 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 [18].

Basically, Algorithm 1 first collects the similarities of all possible attribute profile pairs of two entity collections, and their maximum values of similarity (lines 2-8). The *similarity* function (line 4) measures the Jaccard coefficient[5]. 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 (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 (lines 14-16). Finally, the connected components of the graph built with these edges, with cardinality greater than one, represent the clusters (line 17). Optionally, a *glue*-cluster can gather all the singleton components, as in [18], to ensure the inclusion of all the possible tokens (blocking keys).

3.1.2 LSH-based 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 $A_{\mathcal{E}_1}$ and $A_{\mathcal{E}_2}$, 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 attribute are expected to be found similar for each attribute; therefore, employing techniques able to group the attribute 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* [11] can be adopted to avoid the quadratic complexity of comparing all possible attribute pairs.

Algorithm 1: Loose Attribute-Match Induction (LMI)

```
Input: Attr. names: $A_1, A_2$; Attr. profiles: $\mathcal{T}_1, \ldots, \mathcal{T}_z$
Output: Set of attribute names clusters: $K$
1  $edges \leftarrow \{\}$        $Sim \leftarrow Map\langle K, V\rangle$
2  $Max \leftarrow Map\langle K, V\rangle$        $Cand \leftarrow Map\langle K, \{V\}\rangle$
   // most similar attr. for each attr.
3  foreach $a_i \in A_1, a_j \in A_2$ do
4      $Sim \leftarrow (\langle a_i, a_j\rangle, similarity(\mathcal{T}_i, \mathcal{T}_j))$
5      if $Sim.get(\langle a_i, a_j\rangle) > Max.get(a_i)$ then
6          $Max \leftarrow (a_i, sim)$
7      if $Sim.get(\langle a_i, a_j\rangle) > Max.get(a_j)$ then
8          $Max \leftarrow (a_j, sim)$
   // matching-attr. candidates generation
9  foreach $a_i \in A_1, a_j \in A_2$ do
10     if $Sim.get(\langle a_i, a_j\rangle) \geqslant (\alpha \cdot Max.get(a_i))$ then
11         $Candidates \leftarrow (a_i, a_j)$
12     if $Sim.get(\langle a_i, a_j\rangle) \geqslant (\alpha \cdot Max.get(a_j))$ then
13         $Candidates \leftarrow (a_j, a_i)$
14 foreach $a_i \in A_1, a_j \in Candidates.get(a_i)$ do
15     if $a_i \in Candidates.get(a_j)$ then
16         $edges \leftarrow \langle a_i, a_j\rangle$
17 $K \leftarrow getConnectedComponentsGrThan1(edges)$
   return $K$
```

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 2.1). 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 to be 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 ~ 0.5 are considered for the attribute-match induction, otherwise no (example Figure 5).

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

[6] The LMI attribute representation model can be used with *Attribute Clustering* [18] as well.

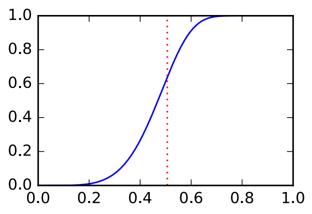

Figure 5: LSH S-curve for $r = 5$ and $b = 30$. Dashed line represents the estimated threshold.

3.1.3 *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 [6]:

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.

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 attribute is derived by computing the *aggregate entropy*. The *aggregate entropy* of a group of attributes C_k is defined as:

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

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 1b, 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 $\mathcal{B}_j$:

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

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

	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 1b.

3.2 Loosely Schema-aware Blocking

In BLAST we employ Token Blocking, as in [18]. Other blocking techniques [17] (e.g., employing q-grams instead of tokens, as in [7]) can be adapted to this scope as well, but comparing them is out of the scope of this paper.

3.3 Loosely Schema-aware Meta-blocking

BLAST introduces a novel WNP meta-blocking technique designed to exploit *loose schema information*.

[20] demonstrated that WNP and CNP generally outperform WEP and CEP, and that *weight-based* pruning criteria outperform the *cardinality-based* ones in terms of PC, but at the expense of PQ. Loosely schema-aware techniques can help to significantly enhance PQ; thus, for this reason and considering the aforementioned results achieved by [20], as a design choice, BLAST employs a weight-based and node-centric pruning criterion (i.e., WNP). Nonetheless, we show in the experimental evaluation (Section 4) that the loose-schema information automatically extracted by BLAST and the BLAST blocking-graph weighting can be employed also to enhance PC for CNP.

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 1b for the profiles p_1 and p_3.

Given this representation, BLAST employs Pearson's chi-squared test (χ^2) [1] 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:

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

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 3.1.3).

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 [20]. To illustrate this phenomenon, consider again the example in Figure 6. Figure 6b shows $\mathcal{G}_{p_1}$, the *node-centric* view of the $\mathcal{G}_\mathcal{B}$ for the profile p_1.

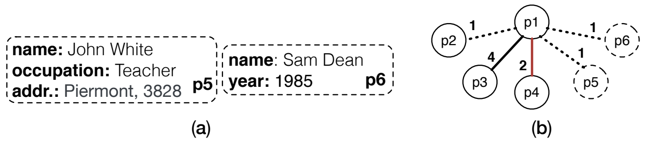

Figure 6: (a) Two additional profiles; (b) the node-centric representation of the blocking graph for p_1.

If the entity collection (as in Figure 1a) 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 6a are added to the entity 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 entity 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 by considering a fraction of this upper bound: $\theta_i = \frac{M}{c}$, 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 PC, but at the expense of PQ.

Having determined the local threshold for each node, the last step to perform is the retention of the edges. Though,

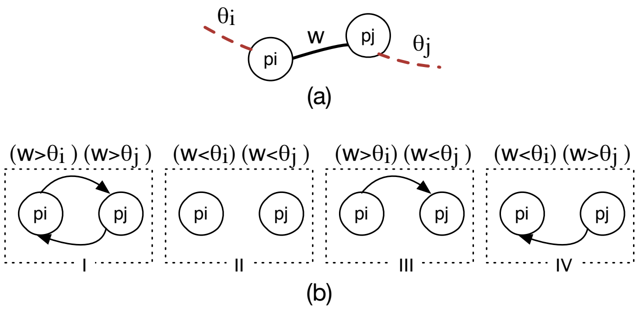

Figure 7: Weight threshold. A directed edge from p_i to p_j indicates that the weight of the edge e_{ij} is higher than θ_i; a directed edge from p_j to p_i indicates that the weight of the edge e_{ij} is higher than θ_j.

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 7a); where θ_i and θ_j are the threshold associated to p_i and p_j, respectively. Hence, as depicted in Figure 7b, 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 [20] propose two different approaches to solve this ambiguity: *redefined WNP* (wnp1) retains e_{ij} if its weight is higher than at least one of the two thresholds, while *reciprocal WNP* (wnp2) retains the edge if it is greater than both θ_i and θ_j. Here in BLAST we choose to employ a unique general threshold, equals to: $\theta_{ij} = (\theta_i + \theta_j)/d$, 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.

4. EVALUATION

Datasets: In the experimental evaluation we employ established benchmarks [17, 19, 18, 12, 20][7] composed of real-world datasets with different characteristics and volume. Table 2 lists the characteristics of the five pairs of datasets compared. The number of entities and attribute names in each profile collection is denoted by $|\mathcal{E}|$ and $|A|$ respectively; the number of name-value pairs corresponds to *nvp*; and $|\mathcal{D}^\mathcal{E}|$ represents the total number of actual duplicates. Each comparison consists of a pair of record sets extracted from on-line data sources of different domains (bibliographic, e-commerce, movies, and general): `ar1` matches article profiles from dblp.org and dl.acm.org; `ar2` matches article profiles from dblp.org and scholar.google.com; `prd` matches product profiles from Abt.com and Buy.com; `mov` matches movie profiles from imdb.com and dbpedia.org; `dbp` matches entity profiles from two different snapshots of DBpedia (2007 and 2009) – 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. The comparison

[7]Here we employ the version used in [17]: `http://sourceforge.net/projects/erframework/files/CleanCleanERDatasets/`

	fully mappable			
	$\lvert\mathcal{E}_1\rvert$ - $\lvert\mathcal{E}_2\rvert$	$\lvert A_1\rvert$ - $\lvert A_2\rvert$	*nvp*	$\lvert\mathcal{D}^{\mathcal{E}}\rvert$
ar1 $^{\text{DBLP}}_{\text{ACM}}$	2.6k - 2.3k	4 - 4	10k - 9.2k	2.2k
ar2 $^{\text{DBLP}}_{\text{Scholar}}$	2.5k - 61k	4 - 4	10k - 198k	2.3k
prd $^{\text{Abt}}_{\text{Buy}}$	1.1k - 1.1k	4 - 4	2.6k - 2.3k	1.1k

	partially mappable			
	$\lvert\mathcal{E}_1\rvert$ - $\lvert\mathcal{E}_2\rvert$	$\lvert A_1\rvert$ - $\lvert A_2\rvert$	*nvp*	$\lvert\mathcal{D}^{\mathcal{E}}\rvert$
mv $^{\text{IMDB}}_{\text{DBp}}$	28k - 23k	4 - 7	155k - 816k	23k
dbp $^{\text{DBp07}}_{\text{DBp09}}$	$1.2M$ - $2.2M$	30k - 50k	$17M$ - $35M$	893k

Table 2: Datasets characteristics.

		baseline			after block filtering		
		PC(%)	PQ(%)	$\lVert\mathcal{B}_o\rVert$	PC(%)	PQ(%)	$\lVert\mathcal{B}_f\rVert$
ar1	T	100	$3.4{\cdot}10^{-2}$	$6.7{\cdot}10^{6}$	99.7	1.6	$1.4{\cdot}10^{5}$
	L	100	$4.5{\cdot}10^{-2}$	$4.9{\cdot}10^{6}$	99.6	2.5	$9.0{\cdot}10^{4}$
ar2	T	99.9	$2.6{\cdot}10^{-3}$	$8.7{\cdot}10^{7}$	99.0	$3.3{\cdot}10^{-3}$	$7.0{\cdot}10^{5}$
	L	99.9	$3.3{\cdot}10^{-3}$	$7.1{\cdot}10^{7}$	98.7	$4.0{\cdot}10^{-3}$	$5.7{\cdot}10^{5}$
prd	T	99.2	$1.4{\cdot}10^{-3}$	$7.8{\cdot}10^{5}$	96.8	1.3	$8.1{\cdot}10^{4}$
	L	98.6	$1.7{\cdot}10^{-3}$	$6.3{\cdot}10^{5}$	96.6	1.3	$7.8{\cdot}10^{4}$
mov	T	98.2	$7.4{\cdot}10^{-3}$	$3.0{\cdot}10^{8}$	97.6	$8.1{\cdot}10^{-2}$	$2.8{\cdot}10^{7}$
	L	98.1	$7.8{\cdot}10^{-3}$	$2.9{\cdot}10^{8}$	97.5	$6.2{\cdot}10^{-2}$	$3.6{\cdot}10^{7}$
dbp	T	99.9	$1.4{\cdot}10^{-5}$	$6.5{\cdot}10^{12}$	99.8	$6.9{\cdot}10^{-3}$	$1.3{\cdot}10^{10}$
	L	99.9	$6.8{\cdot}10^{-5}$	$1.3{\cdot}10^{12}$	99.9	$9.6{\cdot}10^{-3}$	$9.3{\cdot}10^{9}$

Table 3: block collection characteristics; the baseline corresponds to Token Blocking applied with or without LMI ("T" and "L" respectively labeled).

can involve datasets whose attributes can be mapped with either 1:1 associations (i.e., *fully mappable*), or 0:n associations (i.e., partially mappable).

Evaluation Metrics: We evaluate the quality of the produced block collections in terms of precision and recall, through their surrogates PC, PQ and F_1-score (section 2). The comparison against a baseline is expressed with

$$\Delta PC(\mathcal{B},\mathcal{B}') = \frac{PC(\mathcal{B}')-PC(\mathcal{B})}{PC(\mathcal{B})};$$

$$\Delta PQ(\mathcal{B},\mathcal{B}') = \frac{PQ(\mathcal{B}')-PQ(\mathcal{B})}{PQ(\mathcal{B})}$$

where $\mathcal{B}$ is the baseline block collection, and $\mathcal{B}'$ is the compared block collection. When comparing Blast with other techniques X, we also use the notation $\Delta PC(X,\text{Blast})$ $=\Delta PC(\mathcal{B},\mathcal{B}')$, where $\mathcal{B}$ is the collection produced with X and $\mathcal{B}'$ the collection produced with Blast (the same for ΔPQ). If not explicit in the text, we always assume Blast to be the second term of $\Delta PC/\Delta PQ$; thus, if the delta is positive Blast performs better than the technique compared, worse otherwise.

To compare the scalability of the analyzed approaches we consider the overhead time t_o; t_o includes also the overhead time of the attribute-match induction technique, if applied. The time t_o of meta-blocking approaches can be insignificant compared to the time for entity-matching, particularly when advanced and time-consuming entity matching methods are employed [19, 8]. Nevertheless, our concern in this paper is the blocking and meta-blocking, rather than the downstream entity matching.

We implemented Blast in `Java 8` as an extension[8] of the open source framework presented in [18]; all the approaches compared to Blast in this paper have been implemented in the same framework. The experiments have been performed under `Ubuntu 14.04`, with 40GB of ram, and `Intel Xeon E5-2670v2` 2.50 GHz.

4.1 High Quality Blocking

In the experimental evaluation, for each comparison of datasets we extract the initial block collection with a redundant blocking technique (either Token Blocking [18] or the combination of Token Blocking with attribute-match induction), and then applied Block Purging [18] and Block Filtering [20], following the workflow proposed in [20]. Table 3 lists the characteristics of the block collections extracted with Token Blocking, alone ("T") and in combination with LMI ("L"), and the characteristics of the block collections after the purging and filtering phase.

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). Notice that we are not applying any text-processing technique to the entity profiles, such as *stop-words removal*; applying such pre-processing techniques might lead to better results, but their application is an orthogonal problem, independent of the blocking approach proposed here.

Block Filtering aims to restructure the block collection removing entity profiles from blocks that are less important for them, performing a light-weight *schema-free* meta-blocking. We filter out the 20% least significant blocks per profile[9].

The time required by both Block Purging and Block Filtering is negligible compared to the meta-blocking phase, and yields high benefits: by reducing the size of the initial block collection, the blocking-graph building and weighting phase is faster, and the final PQ is higher. The Token Blocking time (either applied with LMI or not) is negligible as well ($10-20$ minutes for `dbp`).

4.1.1 Blast Performance

Tables 4 and 5 present the performance of Blast compared with the state-of-the-art meta-blocking [20]. Blast adopts a WNP approach, but, for the sake of completeness we compare it against both the state-of-the-art WNP and CNP meta-blocking. We exclude from the comparison WEP (and CEP), since the recently introduced *redefined* and *reciprocal* WNP (and CNP) have been demonstrated to outperform them [20], both in terms of quality and completeness of the final restructured block collection.

Redefined WNP, reciprocal WNP, redefined CNP, and reciprocal CNP are labeled with *wnp1*, *wnp2*, *cnp1*, and *cnp2* respectively. For each of these meta-blocking technique we list the average values obtained with the five possible weighting schemes (ARCS, JS, EJS, CBS, and ECBS) employed in traditional graph-based meta-blocking [20].

Moreover, we adapted CNP to work with loose schema-information. In particular, we perform traditional CNP in combination with LMI, and weighted the blocking graph with the Blast weighting function (based on χ^2 and aggregate entropy). The results are listed in Tables 4 and 5, in the rows *cnp1* χ^2_H and *cnp2* χ^2_H.

[8]Available at: `http://stravanni.github.io/blast/`
[9]Experimental analyses have shown that 20% is a tradeoff that almost does not affect PC.

			PC(%)	PQ(%)	F_1	t_o(s)	$\|\mathcal{B}\|$
unsup. MB	wnp1	T	99.6	9.1	.167	0.09	$2.4{\cdot}10^4$
		L	99.0	13.8	.242	2.31	$1.7{\cdot}10^4$
	wnp2	T	99.1	15.1	.259	0.08	$1.4{\cdot}10^4$
		L	98.4	23.5	.375	2.15	$9.2{\cdot}10^3$
	cnp1	T	99.5	8.4	.154	0.07	$2.6{\cdot}10^4$
		L	99.1	9.8	.178	2.07	$2.2{\cdot}10^4$
		Blast Lχ^2_h	99.6	5.2	.099	2.09	$4.2{\cdot}10^4$
	cnp2	T	98.7	18.3	.309	0.05	$1.2{\cdot}10^4$
		L	98.2	20.4	.338	1.89	$1.1{\cdot}10^4$
		Blast Lχ^2_h	99.6	8.9	.165	1.95	$2.4{\cdot}10^4$
sup. MB			92.4	49.4	.644	5.65	$4.1{\cdot}10^3$
Blast			99.0	60.6	.752	2.30	$3.6{\cdot}10^3$

(a) ar1

			PC(%)	PQ(%)	F_1	t_o(s)	$\|\mathcal{B}\|$
unsup. MB	wnp1	T	97.90	0.65	.013	0.85	$3.4{\cdot}10^5$
		L	97.61	0.77	.015	7.70	$2.9{\cdot}10^5$
	wnp2	T	95.29	3.16	.060	0.79	$6.9{\cdot}10^4$
		L	95.01	3.47	.066	7.68	$6.3{\cdot}10^4$
	cnp1	T	95.50	1.36	.027	0.79	$1.6{\cdot}10^5$
		L	94.48	2.01	.040	7.60	$1.1{\cdot}10^5$
		Blast Lχ^2_h	97.18	0.9	.018	7.50	$2.4{\cdot}10^5$
	cnp2	T	88.00	31.15	.460	0.70	$6.5{\cdot}10^4$
		L	85.23	45.64	.594	7.61	$4.3{\cdot}10^4$
		Blast Lχ^2_h	92.94	18.17	.304	7.63	$1.1{\cdot}10^4$
sup. MB			90.3	11.1	.198	9.09	$1.8{\cdot}10^4$
Blast			95.28	5.39	.102	7.70	$4.1{\cdot}10^4$

(b) ar2

			PC(%)	PQ(%)	F_1	t_o(s)	$\|\mathcal{B}\|$
unsup. MB	wnp1	T	92.60	05.90	.111	0.07	$1.7{\cdot}10^4$
		L	92.86	06.30	.119	1.52	$1.6{\cdot}10^4$
	wnp2	T	88.79	09.68	.173	0.05	$1.0{\cdot}10^4$
		L	89.07	09.62	.172	1.23	$1.0{\cdot}10^4$
	cnp1	T	91.62	06.72	.125	0.05	$1.5{\cdot}10^4$
		L	93.03	06.41	.120	1.39	$1.6{\cdot}10^4$
		Blast Lχ^2_h	95.91	3.7	.071	1.50	$2.8{\cdot}10^4$
	cnp2	T	85.22	14.34	.245	0.03	$6.5{\cdot}10^3$
		L	87.08	12.93	.225	1.35	$7.4{\cdot}10^3$
		Blast Lχ^2_h	93.77	6.12	.115	1.41	$1.6{\cdot}10^4$
sup. MB			71.6	22.1	.338	3.35	$3.6{\cdot}10^3$
Blast			87.45	21.60	.347	1.53	$4.5{\cdot}10^3$

(c) prd

			PC(%)	PQ(%)	F_1	t_o(s)	$\|\mathcal{B}\|$
unsup. MB	wnp1	T	96.37	0.48	.010	16.1	$4.6{\cdot}10^6$
		L	96.38	0.38	.008	29.2	$5.8{\cdot}10^6$
	wnp2	T	94.48	0.88	.017	14.5	$2.5{\cdot}10^6$
		L	94.66	0.68	.014	27.2	$3.2{\cdot}10^6$
	cnp1	T	94.22	2.54	.049	8.6	$8.5{\cdot}10^5$
		L	94.21	2.55	.050	22.4	$8.5{\cdot}10^5$
		Blast Lχ^2_h	96.02	1.44	.028	24.1	$1.5{\cdot}10^6$
	cnp2	T	88.59	8.42	.153	8.4	$2.4{\cdot}10^5$
		L	88.82	8.43	.154	22.3	$2.4{\cdot}10^5$
		Blast Lχ^2_h	94.95	3.1	.059	23.9	$7.1{\cdot}10^5$
sup. MB			94.87	4.40	.084	228.0	$5.0{\cdot}10^5$
Blast			94.35	20.63	.339	28.2	$1.0{\cdot}10^5$

(d) mov

Table 4: Comparisons.

			PC(%)	PQ(%)	F_1	t_o(h)	$\|\mathcal{B}\|$
unsup. MB	wnp1	T	99.4	0.05	.001	9.0	$1.8{\cdot}10^9$
		L*	98.8	0.10	.002	10.7	$9.3{\cdot}10^8$
	wnp2	T	97.7	0.15	.002	8.9	$5.8{\cdot}10^8$
		L*	96.9	0.23	.005	10.6	$9.3{\cdot}10^8$
	cnp1	T	96.5	2.6	.050	3.7	$3.3{\cdot}10^7$
		L*	97.4	1.3	.026	5.6	$3.4{\cdot}10^7$
		Blast Lχ^2_h	97.6	1.4	.028	5.5	$6.2{\cdot}10^7$
	cnp2	T	91.2	14.2	.245	3.6	$1.2{\cdot}10^7$
		L*	92.1	6.7	.122	5.4	$1.3{\cdot}10^7$
		Blast Lχ^2_h	94.1	6.5	.121	5.3	$1.3{\cdot}10^7$
	sup. MB		97.3	0.18	.004	33	$4.8{\cdot}10^8$
Blast			93.4	26.3	.411	19.8	$3.2{\cdot}10^6$
Blast*			93.4	26.3	.411	9.6	$3.2{\cdot}10^6$

Table 5: Comparison dbp. Starred methods employ the LSH-based step.

We also compare Blast against supervised meta-blocking, using as training set the 10% of the entity profiles matched in the ground truth (as in [19]). In this case, we employ WEP schema in combination with Support Vector Machine (SVM), since it is the classification algorithm that, in average, has the best F_1 score. The choice of WEP is due to incompatibility of WNP with supervised meta-blocking, since it always selects a global optimum threshold [19]; nevertheless weight-based pruning remains the best choice to maximize PC.

To demonstrate that traditional meta-blocking cannot fully take advantage of loosely schema-aware blocking, we also compare the naïve combination of LMI and unsupervised meta-blocking for each block: in Tables 4 and 5, when the starting block collection is extracted with Token Blocking alone, the row is marked with "T"; while, when the collection is extracted with the support of LMI, the row is marked with "L". In Table 5, "L*" and "Blast*" indicates the employment of the LSH-based LMI.

Blast vs. Schema-based Blocking. Finally, we compare Blast against schema-based *Standard Blocking* [5, 17] on the *fully-mappable* datasets (ar1, ar2, and prd). Standard Blocking is one of the best performing approaches in the literature [5], and is compatible with meta-blocking[10]. Thus, for a fair comparison, we adapted the Blast meta-blocking to work with it. We experimentally observed that they achieve the exact same PC and PQ. (We do not report here the results for the sake of brevity.) In fact, interestingly, the attribute partitioning induced by LMI is equivalent to the manual schema-alignment on all the *fully-mappable* datasets. However, Blast can be employed even if schema is unknown or hard to induce.

4.1.2 Blast Components Evaluation

The results in Tables 4 and 5 show how Blast can actually take full advantage of attribute-match induction. In fact, Blast achieves significantly higher F_1-score than the one achieved by traditional meta-blocking, maintaining high level of *PC*. Differently, Token Blocking combined with LMI and traditional meta-blocking achieves a slightly higher F_1-score than that achieved by Token Blocking alone.

Here, we analyze the contribution of the components that characterize the Blast WNP meta-blocking: *aggregate entropy*, the *chi-squared weighting*. Hence, we compare Blast meta-blocking in three different settings (chi, wsh, and bch) described in the following. chi: We run Blast switching off the *aggregate entropy*, i.e., the weights of the blocking graph are computed using χ^2 without the multiplicative entropy factor. wsh: We run Blast replacing the χ^2-based weighting function with the traditional weighting scheme

[10]Standard Blocking is equivalent to Token Blocking modified to exploit schema-mapping to disambiguate tokens according to the attribute in which they appear.

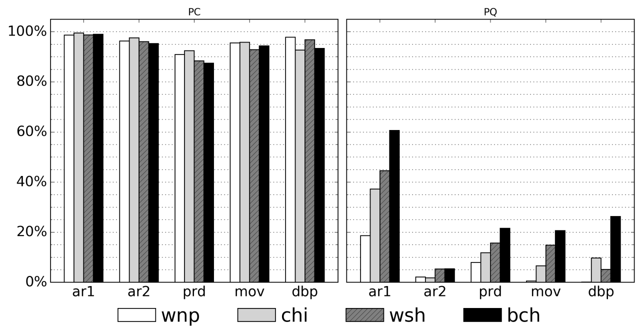

Figure 8: PC and PQ of: classical WNP (average of $wnp1$ and $wnp2$); blast without considering the aggregate entropies; blast working with classical weighting schemas (WS) of traditional meta-blocking adapted to exploit the aggregate entropies; and blast.

(WS) [20] adapted to work with *aggregate entropy.* For instance, for the *Jaccard Scheme*, we run Blast substituting χ^2 score with the Jaccard Similarity in the weighting function presented in Section 3.3.1. `bch`: We run Blast with its standard configuration. The results are shown in Figure 8. The inputs are block collections generated from the datasets of Table 2 with LMI and Token Blocking. Figure 8 also reports the averages of PC and PQ resulting from *wnp1* and *wnp2* (`wnp`). For the experiments involving traditional weighting schemes we report the average PC and PQ.

4.2 Considerations

Overall, the experimental results presented above show that Blast outperforms traditional meta-blocking (both WNP and CNP) in terms of PQ and F_1-score on 2 out of 3 of the fully-mappable dataset comparisons, and on 2 out of 2 of the partially-mappable ones. In particular, compared to traditional WNP techniques, Blast increases PQ up to two order of magnitudes, with a small degradation (if any) of PC. In fact, $\Delta PC(\mathcal{B}_{\text{WNP}}, \mathcal{B}_{\text{BLAST}})$ is in the range $(0\%, -6\%)$ for all the datasets. While, compared to traditional CNP techniques, Blast yields a lower PQ and F_1-score than *cnp2* only in `ar2`; though $\Delta PC(\mathcal{B}_{cnp2}, \mathcal{B}_{\text{BLAST}})$ is $+12\%$.

Compared with supervised meta-blocking, Blast yields a lower F_1-score only with `ar1`, though ΔPC is $+6\%$. Moreover, ΔPC is negative only for `dbp`, but the ΔPQ is $+14{,}511\%$.

In the following we report some considerations about the benefits yielded by the different Blast components.

4.2.1 Blast Components Contribution

The experiments described in Section 4.1.2 aim to quantify the contribution of each component of Blast. The results are shown in Figure 8.

Aggregate Entropy. The comparison of `bch-chi` (i.e., Blast executed with and without considering the aggregate entropy) intends to quantify the contribution of exploiting the attribute entropies. PC is almost identical, while PQ increases up to a factor $5\times$. This demonstrates that entropy can be actually exploited to enhance the quality of meta-blocking.

Chi-squared weighting. Traditional weighting schemes [20] basically compute simple similarity measures between two entity profiles. For instance, JS assigns to an edge a weight equals to the Jaccard Similarity of its adjacent profiles. Differently, Blast employs a statistical test designed to quantify the significance of the co-occurrences, hence better suited for our problem. To demonstrate that, we compare `bch-wsh`, i.e., Blast operating with χ^2_h and traditional weighting schemes WS respectively. With χ^2_h PQ is increased up to a factor $5\times$ for `dbp`. PC is almost identical for all the datasets.

Finally, consider the experimental results of Tables 4 and 5 where the Blast χ^2_h-based weighting function is employed in conjunction with the traditional CNP methods (labeled with "Blast$L_{\chi^2_h}$"). Recall that CNP retains the top k comparisons for each profile. We observe that with this setting PC is always high (above $\sim$93%), while for traditional CNP it drops down for some datasets ($\sim$85%). This means that χ^2_h can better capture the significance of the co-occurrences in the block collections. In other words, if the recall depends on the top-k edges (as for CNP), then a high recall means that matching profiles are ranked higher with χ^2_h than with traditional weighting functions.

4.2.2 LSH-based LMI Benefit

Finally, we want to stress the concept that the time consumed to restructure a block collection saves a much greater required for superfluous comparisons (which are removed by meta-blocking). For instance, considering `dbp` and the "simple" Jaccard similarity of profiles for determining the matching[11], the time to execute the comparison of the final block collection produced by Blast is $\sim$2 hours, while the comparison of the original block collection is $\sim$50 hour. The time saved would have been even higher if more sophisticated and time consuming techniques were employed to perform the comparisons [19, 17].

Nevertheless, saving time is still possible; in fact, the last two rows of Table 5 show the performance of Blast with and without the support of the LSH-based step for LMI. Considering the only attribute-match induction phase, the LSH-step allows to run LMI on `dbp` in $\sim$2 hours, instead of $\sim$12 hours, obtaining identical results in terms of PC and PQ (see section 4.3), and reducing the overall t_o of $\sim$50% compared to Blast applied without LSH-step.

4.3 LMI vs. AC

Figure 9 reports the results of the comparison between the two attribute-match induction techniques, namely LMI and AC (section 3.1). The main difference between LMI and AC is that LMI tries to produce cohesive cluster of attributes (i.e., all the attributes in the cluster are all highly similar to each other); whereas AC aims to group together attributes similar to other similar attributes (i.e., each attribute inserted in a group has at least one highly similar attribute in its cluster). On large datasets, the behavior is similar, and the final results of the meta-blocking phase are identical; while, for the small datasets, LMI has been proven to enhance PQ up to 9.8%.

4.4 LSH-based Attribute-Match Induction

As pointed out in Section 4.2, LSH-based step allows to run LMI on `dbp` in $\sim$2 hours, instead of $\sim$12 hours. In this

[11]Profiles are treated as strings, without considering metadata, we compute the Jaccard coefficient of the profiles according to the final block collection

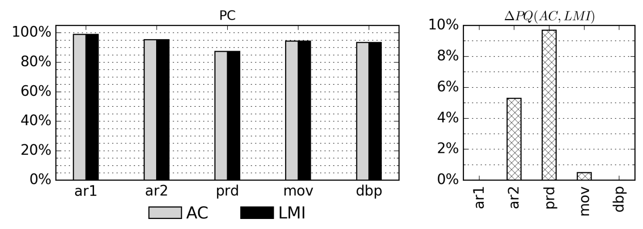

Figure 9: PC comparison and ΔPQ between Blast with AC and LMI.

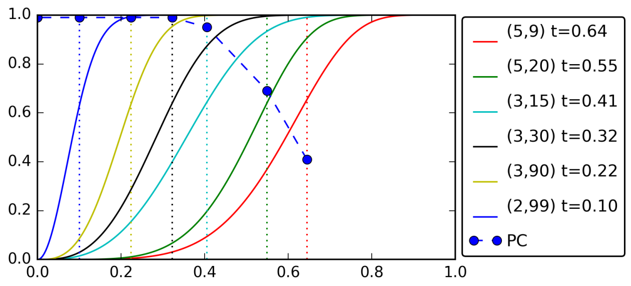

Figure 10: PC with different LSH configurations in combination with LMI. In the legend n. of rows and n. of bands for LSH are in parenthesis, and t is the estimated threshold.

–	$LSH_{.10}$	$LSH_{.22}$	$LSH_{.32}$	$LSH_{.41}$	$LSH_{.55}$	$LSH_{.64}$
12.5 h	1.9 h	1.5 h	1.3 h	1.2 h	0.9 h	0.7 h

Table 6: LMI run time varying the LSH threshold.

section we assess the benefit of the LSH-based step. To do that, consider the worst case scenario: when attribute-match induction does not identify any similar attribute, all the attributes are grouped in a unique all-encompassing cluster (the *glue cluster* [18]). In this scenario, the blocks generated combining attribute-match induction with Token Blocking are identical to those generated with Token Blocking alone. On the other hand, if the attribute-match induction correctly groups some similar attributes, separating them from the glue cluster, the PQ of the produced block collection increases, while PC remains almost the same.

Ideally, the more the similar attributes are correctly grouped, the higher the PQ of the generated blocks is, without affecting PC. Hence, to demonstrate the advantage of LSH-based attribute-match induction, we perform a set of experiments disabling the glue cluster in the LSH-LMI algorithm (LMI applied in combination with the LSH-based step), and varying the threshold of LSH. 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 PC 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[12] than the threshold, LMI does not consider them as a candidate pair, and they cannot be indexed in the same group.

Figure 10 shows how LSH affects the final results of Blast combined with LMI in terms of PC. Table 6 reports the execution times of the experiment. We consider PC of the block collection produced with LSH-LMI in combination with Token Blocking only, without considering the meta-blocking phase. Basically, up to a threshold value of .35 (i.e., Jaccard similarity equals to .35), the PC is not affected ($PC = 99.99\%$), meaning that all the matching profile pairs are successfully indexed in the block collection. PQ is not reported, but for the points where $PC = 99.99\%$ is identical, i.e., it is not affected by the LSH threshold. For threshold greater than .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, LMI fails to recognize similar attributes and produces incomplete cluster of attributes. Nevertheless, even for conservative threshold (e.g. .10), the execution of LMI, overall, is under 2h (instead of ~12h).

4.5 Dirty ER

Finally, we adapted Blast to work with dirty ER. LMI (Section 3.1.1) 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). Typically, in dirty ER 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 LMI to work with dirty ER. For dirty ER, Blast meta-blocking approach needs no changes.

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

To evaluate the performances of Blast we compared it against traditional meta-blocking techniques on 3 real-world benchmark datasets [5]. Both Blast and traditional meta-blocking are applied in combination with LMI[13].

Results. The characteristics of the datasets and the results are listed in Table 7. Blast achieves higher PQ and F_1-score than traditional WNP, and a slightly lower PC. The only exception is on `cora`, where $\Delta PC(wnp1, \text{Blast})$ is -9% (though ΔPQ=56%). Compared to CNP, Blast outperforms *cnp1* on `cora` and `cddb`, while fall behind it on `census`. On `census` and `cddb`, *cnp2* outperforms Blast, but in `cora` its PC is considerably low (46%).

Overall, for dirty ER, Blast can be a 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. RELATED WORK

Blocking techniques have been commonly employed in *Entity Resolution* (ER) [10, 15], and can be classified into two broad categories: the *schema-based* (Suffix Array [7], q-grams blocking [9], Canopy Clustering [14]) techniques, and the *schema-agnostic* ones (Token Blocking [18], and *Attribute-match induction* [18, 12]).

Attribute-match induction. Among the schema-agnostic techniques, *Attribute Clustering* (AC) [18] and *TYPiMatch*

[13]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 LMI does not significantly affect the total execution times.

	Blast	wnp1	wnp2	cnp1	cnp2
$PC(\%)$	74.7	78.3	68.3	84.4	78.7
$PQ(\%)$	8.90	8.02	11.5	8.8	14.2
F_1	.1590	.1448	.1965	.1608	.2361

1k profiles, Ground Truth: 300 matches
(5 attributes - 2 clusters with LMI)
(a) census

	Blast	wnp1	wnp2	cnp1	cnp2
$PC(\%)$	82.1	90.3	81.2	66.9	46.2
$PQ(\%)$	84.0	53.8	69.4	65.7	82.4
F_1	.8302	.6726	.7377	.6637	.5917

1k profiles, Ground Truth: 17k matches
(12 attributes - 4 clusters with LMI)
(b) cora

	Blast	wnp1	wnp2	cnp1	cnp2
$PC(\%)$	93.7	97.3	96.1	96.8	94.9
$PQ(\%)$	0.13	0.03	0.04	0.08	0.18
F_1	.0027	.0005	.0008	.0015	.0036

10k profiles, Ground Truth: 600 matches
(106 attributes - 16 clusters with LMI)
(c) cddb

Table 7: Comparison dirty ER datasets.

[12] 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"*) frequent on generic dataset on the Web, and uses this information to select blocking keys; but it cannot efficiently scale to large dataset.

Meta-blocking. Meta-blocking [19, 20] aims to enhance *block collections* produced by an underlying blocking technique. Existing techniques are completely schema-agnostic and can be *supervised* [19] or *unsupervised* [19]. The technique proposed in Blast is unsupervised and exploits the *loose schema information*.

Metadata exploitation. Finally, there is excellent related work in the semantic Web community [21, 16, 22, 2]. For instance, LIMES [16] (an ER approach for the Web of Data), and LOV [22] (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.

6. CONCLUSION

In this paper we presented an 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 datasource schemas). We demonstrated that Blast can efficiently scale to large and highly heterogeneous datasets (such as data on the Web) through an LSH-based optional step. Finally, we experimentally evaluated it on real world datasets. The experimental results showed how Blast outperforms the existing state-of-the-art meta-blocking approaches: compared to traditional weighted-based ones, it enhances precision up to two order of magnitudes, while the variation in recall (ΔPC) is at worst -6%; compared to the cardinality-based ones, recall and precision are nearly always higher.

7. REFERENCES

[1] A. Agresti and M. Kateri. Categorical data analysis. In *International Encyclopedia of Statistical Science*, pages 206–208. 2011.

[2] S. Bergamaschi, D. Ferrari, F. Guerra, G. Simonini, and Y. Velegrakis. Providing insight into data source topics. *Journal on Data Semantics*, pages 1–18, 2016.

[3] C. Bizer, T. Heath, and T. Berners-Lee. Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, 5(3):1–22, 2009.

[4] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–. IEEE Computer Society, 1997.

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

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

[7] T. de Vries, H. Ke, S. Chawla, and P. Christen. Robust record linkage blocking using suffix arrays and bloom filters. *TKDD*, 5(2):9, 2011.

[8] X. L. Dong and D. Srivastava. Big data integration. *Synthesis Lectures on Data Management*, 2015.

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

[10] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data Knowl. Eng.*, 69(2):197–210, 2010.

[11] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets, 2nd Ed.* 2014.

[12] Y. Ma and T. Tran. Typimatch: type-specific unsupervised learning of keys and key values for heterogeneous web data integration. In *WSDM*, pages 325–334, 2013.

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

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

[15] F. Naumann and M. Herschel. An introduction to duplicate detection. *Synthesis Lectures on Data Management*, 2010.

[16] A. N. Ngomo and S. Auer. LIMES - A time-efficient approach for large-scale link discovery on the web of data. In *IJCAI*, pages 2312–2317, 2011.

[17] G. Papadakis, G. Alexiou, G. Papastefanatos, and G. Koutrika. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *PVLDB*, 9(4):312–323, 2015.

[18] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *TKDE*, 25(12):2665–2682, 2013.

[19] G. Papadakis, G. Papastefanatos, and G. Koutrika. Supervised meta-blocking. *PVLDB*, 7(14):1929–1940, 2014.

[20] G. Papadakis, G. Papastefanatos, T. Palpanas, and M. Koubarakis. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. In *EDBT*, pages 221–232, 2016.

[21] P. Shvaiko and J. Euzenat. Ontology matching: State of the art and future challenges. *TKDE*, 25(1):158–176, 2013.

[22] P. Vandenbussche and B. Vatant. Linked open vocabularies. *ERCIM*, 2014(96), 2014.