This is the accepted manuscript of the article, which has been published in 34th Annual ACM
Symposium on Applied Computing, SAC 2019. 2019, 422-430.
http://dx.doi.org/10.1145/3297280.3299730

A Noise Tolerant and Schema-agnostic Blocking Technique for Entity Resolution

Tiago Brasileiro Araújo
Federal University of Campina Grande
Campina Grande, Brazil
tiagobrasileiro@copin.ufcg.edu.br

Carlos Eduardo Santos Pires
Federal University of Campina Grande
Campina Grande, Brazil
cesp@dsc.ufcg.edu.br

Demetrio Gomes Mestre
State University of Paraíba
Campina Grande, Brazil
demetriogm@uepb.edu.br

Thiago Pereira da Nóbrega
State University of Paraíba
Campina Grande, Brazil
thiagonobrega@uepb.edu.br

Dimas Cassimiro do Nascimento
Federal Rural University of Pernambuco
Garanhuns, Brazil
dimascnf@uag.ufrpe.br

Kostas Stefanidis
University of Tampere
Tampere, Finland
kostas.stefanidis@uta.fi

ABSTRACT

The increasing use of Web systems has become a valuable source of semi-structured data. In this context, the Entity Resolution (ER) task emerges as a fundamental step to integrate multiple knowledge bases or identify similarities between the data items (i.e., entities). Usually, blocking techniques are widely applied as an initial step of ER approaches in order to avoid computing similarities between all pairs of entities (quadratic cost). In practice, heterogeneous and noisy data increase the difficulties faced by blocking techniques, since these issues directly interfere the block generation. To address these challenges, we propose the NA-BLOCKER technique, which is capable of tolerating noisy data to extract information regarding the data schema and generate high-quality blocks. NA-BLOCKER applies Locality Sensitive Hashing (LSH) to hash the attribute values of entities and enable the generation of high-quality blocks, even with the presence of noise in the attribute values. In our experimental evaluation, we use five real-world datasets, and highlight that NA-BLOCKER presents better results regarding effectiveness compared to the state-of-the-art technique. In terms of efficiency, NA-BLOCKER produces, on average, 34% less comparisons. However, due to the cost introduced by LSH, it results in an increase of the execution time at around 30%, on average.

CCS CONCEPTS

• **Information systems** → **Entity resolution**; *Semi-structured data*;

KEYWORDS

Entity resolution, Heterogeneous data, Metablocking, Noisy data

ACM Reference Format:
Tiago Brasileiro Araújo, Carlos Eduardo Santos Pires, Demetrio Gomes Mestre, Thiago Pereira da Nóbrega, Dimas Cassimiro do Nascimento, and Kostas Stefanidis. 2019. A Noise Tolerant and Schema-agnostic Blocking Technique for Entity Resolution. In *Proceedings of ACM SAC Conference (SAC'19).* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Currently, the increasing use of Web systems (e.g., digital libraries, social networks and e-commerce) has become a valuable source of semi-structured data [30]. This kind of data can be represented in different formats (e.g., XML, RDF, JSON, or text) and be contained in different knowledge bases or databases. A fundamental step to integrate multiple knowledge bases or identify similarities between entities is Entity Resolution (ER). ER is a task that matches records (the entity profiles) from several data sources (the entity collections) that refer to the same real-world entity [4]. The ER task is widely used by the Web community because it is commonly necessary to integrate multiple knowledge bases, which store semi-structured data [20, 28].

In the context of Web Data [5, 27], the ER task deals with two Vs: *volume*, as it handles a large amount of entities; and *variety*, since different formats are used to represent the entity profiles (heterogeneous data) [6–8]. Beyond the two Vs, we highlight here another problem that is tackled by the ER task: noisy data, commonly characterized by pronunciation/spelling errors and typos in the attribute values of the entities [14]. In practical scenarios, people are less careful with the lexical accuracy of the content written in informal virtual environments (e.g., social networks) or when they are submitted to some kind of pressure (e.g., business reports) [1]. For these reasons, real-world data often present noise that can impair data interpretations, data manipulation tasks, and decision-making [9]. In the ER context, noisy data directly impact the identification of similar entities, since the spelling difference of their attribute values may determine that two entities, truly similar in the real world, are not regarded as similar by the ER task. In this work, the two most common noise on data will be considered: typos and misspelling errors [1]. To deal with problems related to the large *volume* of data handled by the ER task, blocking techniques can be applied [19]. Blocking techniques group similar entities into blocks and perform comparisons between entities within the same block, avoiding the comparisons guided by the Cartesian product. Therefore, the block techniques aim to reduce the total number of comparisons to be performed in the ER task.

The *variety* of data is related to the fact that entity profiles do not share the same loose schema. In this sense, traditional blocking techniques (e.g., Sorted Neighborhood and Adaptive Window) do not present a satisfactory effectiveness (in the sense of grouping truly similar entities in the same block), since the generation of blocks is performed based on the entity profile schema [8]. In turn, schema-agnostic blocking techniques (e.g., Token Blocking and Attribute Clustering Blocking) have been proposed to address the variety challenge, since these techniques disregard the schema and consider only the values related to the entity attributes [6]. Among the schema-agnostic techniques, the Blocking with Loosely-Aware Schema Technique (BLAST [26]) emerges as one of the most promising techniques regarding effectiveness. Although BLAST is a schema-agnostic technique, it exploits possible schema information based on the data (i.e., statistics collected directly from the data) to enhance the quality of the blocks in a loosely schema-aware metablocking approach. However, the presence of noise in the attribute values compromises the effectiveness of BLAST, since it relies on the accuracy of the attribute values to exploit schema information as well as generate and prune the blocks.

Overall, the main contribution of our work is proposing the NA-BLOCKER (Noise-aware Schema-agnostic Blocking for Entity Resolution): a novel schema-agnostic blocking technique capable of tolerating noisy data to extract information regarding the schema from the data (i.e., group similar attributes based on the data) and enhance the quality of the generated blocks. To this end, the NA-BLOCKER applies Locality Sensitive Hashing (LSH) in order to hash the attribute values of the entities and enable the generation of high-quality blocks (i.e., blocks that contain a significant number of entities with high chances of being considered similar/matches), even with the presence of noise in the attribute values.

The NA-BLOCKER technique is evaluated against the state-of-the-art method, namely BLAST, regarding efficiency and effectiveness, using five real datasets. Based on this experimental evaluation, we highlight that NA-BLOCKER outperforms BLAST regarding effectiveness for all pairs of data sources, even with the presence of noise in the attribute values. NA-BLOCKER achieves better results due to the application of the LSH-based strategy, which generates the entity blocks based on approximate similarity, instead of the exact-match applied in BLAST. Moreover, the LSH-based strategy provides the building of high-quality blocks capable of minimizing the number of comparisons to be performed in the ER task, after the blocking step. This reduction in the number of comparisons denotes gains of efficiency in the process as a whole. However, the application of LSH in NA-BLOCKER increases the computational cost to generate the blocks. For this reason, our approach demands more time to be executed when compared with BLAST.

2 NOISE-AWARE SCHEMA-AGNOSTIC BLOCKING FOR ENTITY RESOLUTION

In this section, we introduce NA-BLOCKER, a noise-aware schema-agnostic blocking technique for Entity Resolution. The main goals of NA-BLOCKER are: *i)* extract efficiently the loose-schema information directly from the data, without user interference (e.g., clerical review); and *ii)* generate high-quality blocks even in the presence of noisy data.

To extract loose-schema information, the attribute-match induction technique is applied [22, 26]. This technique aims to determine the similarity between pairs of attributes based on the attribute values associated with the attributes in question. In order to enhance the reliability of the blocks (i.e., prevent dissimilar entities from being inserted into the same block), the attribute-match induction extracts schema information from the attribute values [16]. Based on this information, groups of similar attributes belonging to two data sources (i.e., two entity collections) are generated. It is important to highlight that the attribute-match induction does not exploit the semantics of the attribute names, but only the attribute values. These groups of similar attributes are considered by schema-agnostic blocking techniques to disambiguate blocking tokens (keys), according to the attribute group from which they are derived [26]. In this sense, the attribute-match induction can be applied to avoid that tokens provided by completely distinct attributes promote the grouping of entities sharing these tokens.

For example, assume the entities e_1 : {*name: Jon Snow; house: Stark*}, e_2 : {*full_name: Arya; family: Stark*}, and e_3 : {*full_name: Tywin Lannister; enemy: Stark*}. If a token-based technique (which considers the exact linguistic similarity) is applied, the three entities will be grouped within the same block since all of them share the token "Stark". However, in the real world, the attributes "house" and "enemy" have completely different semantic meanings. Therefore, the entities e_1 and e_2 should not be inserted into the block that contains e_3. In this sense, the attribute-match induction can be applied in order to avoid that tokens provided by completely distinct attributes promote the grouping of entities sharing these tokens.

Regarding the noisy data, the development of noise-aware approaches is considered as an open research area by several works [14, 26]. Noise-aware approaches are capable of tolerating noisy data in order to avoid that the noise negatively interferes the effectiveness of these approaches. Since real-world data sources typically present noise in the data, blocking techniques need to deal with noise, such as pronunciation errors, typos, misspellings, slang, and abbreviations [10]. In the context of schema-agnostic blocking techniques, the noisy data directly impacts block generation since the blocks are generated based on the attribute values of the entities. For instance, assume two entities $e_1 : \{name : Jon\ Snow\}$ and $e_2 : \{full\_name : John\ Sn0w\}$. Notice that the token-based blocking techniques [3, 21, 23] extract tokens from the attribute values and consider the exact linguistic similarity between the tokens. Therefore, if a token-based technique is applied, due to the existence of typos and pronunciation errors in the attribute values, entities e_1 and e_2 will not be grouped into the same block, even though they can be considered truly similar entities.

To be able to work with noisy data, we apply Locality-Sensitive Hashing (LSH) in order to avoid the issues generated by the noisy data [14]. In general, LSH is used for approximating the near neighbour search in high-dimensional spaces [2]. It can be applied to reduce the dimensionality of a high-dimensional space, preserving the similarity distances and reducing significantly the number of the attribute values (or tokens) to be evaluated. For each attribute of an entity, a hash function (e.g., MinHash [2]) converts the attribute value into a probability vector, called signature (Minhash signature). Since the hash function preserves the similarity of the attribute values, it is possible to apply distance functions (e.g., Jaccard) to determine the similarity between attribute values of two distinct entities [2]. In the context of schema-agnostic blocking, the hash function can generate similarity vectors that would guide the block generation. Therefore, entities with similar vectors will be inserted into the same block.

3 THE NA-BLOCKER FRAMEWORK

NA-BLOCKER is based on the Metablocking technique [23], which exploits abstract blocking information to improve the efficiency gains with a minimum impact on the effectiveness. In other words, Metablocking aims to reduce the amount of comparisons generated by each block without discarding comparisons with high chances of resulting in correspondences (i.e., matches). To this end, Metablocking restructures a given set of blocks into a new one that involves significantly fewer comparisons, while maintaining the original level of effectiveness [23]. This process is called pruning. Initially, a schema-agnostic blocking technique, e.g., token blocking, is applied to block the heterogeneous data. Token blocking extracts tokens (e.g., keywords) from the attribute values of every entity and creates an individual block for every token that appears in at least two entities. It is important to highlight that blocks generated by token blocking result in a big number of redundant comparisons between entities. For this reason, the blocks generated by token blocking are transformed into a weighted graph, such that each entity is represented by one node and each edge between a pair of nodes infers that the pair of nodes shares at least one block in common. Based on the number of blocks in common between the pair of nodes (pair of entities) linked by the edge, the Metablocking technique defines the weight of each edge (in the graph). Finally, pruning criteria are applied to remove edges with weight below a threshold, which aims to discard comparisons between entities with few chances of being considered a correspondence.

Overall, NA-BLOCKER is divided into three steps: i) schema information extraction, ii) block generation, and iii) pruning, as depicted in Figure 1. The proposed technique receives as input two data sources D_1 and D_2. Each data source is an entity collection $D = \{e_1, e_2, e_3, ..., e_n\}$, such that n is the number of entities in D. The attributes contained in each data source are denoted by $A(D) = \{a_1, a_2, \cdots, a_k\}$, such that k is the number of attributes in D (notice that the value of k may vary for each data source). Since the entities can follow different loose schemas, each entity $e \in D$ has a specific attribute set and a value associated to each attribute, denoted by $A_e = \{\langle a_1, v_1\rangle, \langle a_2, v_2\rangle, \langle a_3, v_3\rangle, ..., \langle a_k, v_k\rangle\}$, such that k is the amount of attributes associated with e.

In the schema information extraction step, all attributes associated to the schemes of the entities belonging to D_1 and D_2 are extracted. Moreover, all values associated to the same attribute (a_i) are grouped into a set V_{a_i}, i.e., $V_{a_i} = \bigcup_{e \in D}(v \mid \langle a_i, v\rangle \in A_e)$. In turn, the pair $\langle a_i, V_{a_i}\rangle$ represents the set of values associated to a specific attribute a_i. The attributes present in D_1 and D_2 are grouped based on the similarity between their attribute values, denoted by $G(D_1, D_2) = \{g_1, g_2, g_3, ..., g_m\} \forall g \in G(D_1, D_2) : g \subseteq (A(D_1) \times A(D_2))$ *and* $\forall g \in G(D_1, D_2) \forall \langle a_i, a_j\rangle \in g : V_{a_i} \simeq V_{a_j}$. The sets V_{a_i} and V_{a_j} are considered similar if $sim(V_{a_i}, V_{a_j}) \geq \Phi$, where $sim(V_{a_i}, V_{a_j})$ calculates the similarity between the sets V_{a_i} and V_{a_j} and Φ is a given threshold[1].

In the block generation step, each set of attribute values V (associated with an attribute a) is converted into a hash-signature S (provided by LSH), given by $hash(\langle a, V\rangle) = \langle a, S\rangle$. Notice that to compute the similarity of all possible pairs of attributes, the process takes an overall time complexity of $O(|U_{D_1}| \cdot |U_{D_2}|)$, such that $U_{D_1} = \bigcup_{a_i \in A(D_1)}(V_{a_i} \mid V_{a_i} \in \langle a_i, V_{a_i}\rangle)$ and $U_{D_2} = \bigcup_{a_j \in A(D_2)}(V_{a_j} \mid V_{a_j} \in \langle a_j, V_{a_j}\rangle)$. However, this time complexity is impractical for semi-structured data that appear on the Web, since data sources can commonly have hundreds of attributes and millions of attribute values [26]. For this reason, the LSH technique, which has a linear cost in relation to the set size, is applied to reduce the dimensionality of these sets, i.e., U_{D_1} and U_{D_2}, targeting at minimizing the time complexity to a linear cost [29].

The set of LSH-signatures S (from $\langle a, S\rangle$) guide the block generation, since entities with a similar LSH-signature are grouped into the same block. The loose-schema information (i.e., $G(D_1, D_2)$) is applied to the block generation step in order to avoid that similar LSH-signatures originated from attributes with different semantics (due to the fact that the attributes are not in the same g) being inserted into the same block by the blocking technique. The output of the block generation step is a collection of blocks B, as denoted by Equation 1.

$$\begin{aligned}
&B = \{b_1, b_2, b_3, ..., b_x\} \mid \\
&\quad \forall b \in B : (e_1 \in b \land e_2 \in b) \Leftrightarrow \\
&\quad (\exists \langle a_1, a_2\rangle \in (A(D_1) \times A(D_2)) : \\
&\quad \langle a_1, a_2\rangle \in \bigcup_{g \in G(D_1, D_2)} g \ \land \ hash(\langle a_1, v_1\rangle \in A_{e_1}) \sim \\
&\quad hash(\langle a_2, v_2\rangle \in A_{e_2}))
\end{aligned} \tag{1}$$

Finally, in the pruning step, Metablocking is applied in order to discard comparisons between entities with low-weight edge, representing low similarity. In this sense, the collection B provided by the block generation step is restructured relying on the intuition that the more blocks two entities share, the more likely they result in a correspondence. Then, the output of the pruning step is a restructured collection of blocks B'. Next, we describe in details each step of the NA-BLOCKER technique.

4 SCHEMA INFORMATION EXTRACTION

This step receives as input two data sources D_1 and D_2. As described in Algorithm 1, for each entity, the attribute values are read in order to extract the tokens (only the relevant words) associated with each attribute (illustrated by the function *extractSignatures*, lines 13 to 29). In other words, punctuation, special characters (e.g., *@*, *$*, ***, and

[1] In this work, we apply a threshold value equal to 0.35 ($\Phi = 0.35$), given the experiments in [26], demonstrating this threshold value as the best one.

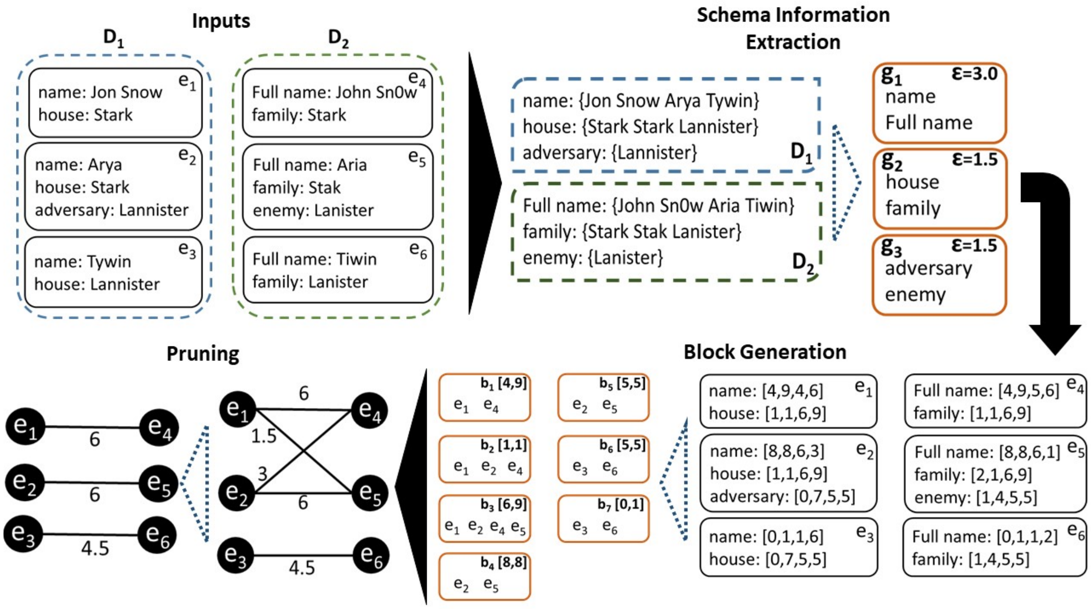

Figure 1: NA-BLOCKER workflow.

&) and stop words (e.g., *the, for, to, at, which*, and *on*) are removed from the attribute values. Thus, a set of values (keywords) (V), extracted from the attribute values, is associated to each attribute ($\langle a, V\rangle$) of an entity (lines 15 to 24). This transformation of the attribute values is important in order to remove characters or words that can degrade the similarity between attributes. To perform the attribute-match induction (i.e., group the similar attributes), the sets of keywords are clustered (merged) according to the attributes (lines 19 and 21). For instance, in Figure 1, all keywords of the attribute *name* are merged into the same set of values. The sets of values are used to measure the similarity between the attributes. In this sense, given two attributes and their respective sets of keywords, the Jaccard[2] function evaluates the similarity between the sets of values and, consequently, determines the similarity values between the attributes.

To enhance the efficiency of the attribute-match induction (i.e., avoid the quadratic complexity of comparing all possible attribute values in this step), the LSH algorithm is applied [2]. As a result, the hash function (MinHash, in our case) generates a signature (*MinHash* signature) for each set of values $\langle a, V\rangle \longrightarrow \langle a, S\rangle$, where S denotes the *MinHash* signature (lines 25 to 28). Since the hash function preserves the similarity between the sets of values, it is possible to apply a similarity function (e.g., Jaccard) to the signatures in order to determine the similarity between the attributes (line 6). It is important to highlight that computing the similarity (e.g., applying Jaccard) between attributes based on the signatures is faster than computing it based on the sets of keywords, since the quadratic cost is avoided. This fact motivates the application of LSH in the first two steps of NA-BLOCKER.

In order to perform the attribute-match induction, the similarities between the attributes are evaluated. Therefore, attributes with high similarity are inserted into the same group g (lines 4 to 11). For example, based on the similarities of the sets of keywords, the attribute groups $g_1 = \{name, Full\ name\}, g_2 = \{family, house\}$ and $g_3 = \{adversary, enemy\}$ are built, as illustrated in Figure 1.

Moreover, this step calculates the entropy of each attribute (line 7). Intuitively, the entropy of an attribute indicates how significant is the attribute, i.e., the higher the entropy of an attribute, the more significant is the observation of a particular value for that attribute [26]. We specifically apply here the Shannon entropy [15] to represent the information distribution of a random attribute. Thus, assume a random attribute X with alphabet χ, and the probability distribution function $p(x) = Pr\{X = x\}, x \in \chi$. Then, the *Shannon entropy* is defined as: $H(X) = -\sum_{x\in\chi} p(x) \log p(x)$.

Thereafter, the aggregated entropy ε is generated from the entropy of each attribute contained in a particular attribute group (line 8). Thus, in Figure 1, the entropies ε associated with each attribute group are: $\langle g_1, 3.0\rangle$, $\langle g_2, 1.5\rangle$, and $\langle g_3, 1.5\rangle$. The entropy of an attribute group influences the weighting of the blocking graph, as will be discussed in the next steps.

Finally, since in this step we need to evaluate all possible attributes contained in our data sources, namely, $A(D_1)$ and $A(D_2)$, and the cost to compare the attributes (given by $\langle a, S\rangle$) is linear in relation to the size of S, the time complexity of this step is expressed as $O(|A(D_1)| \cdot |A(D_2)| \cdot |S|)$.

5 BLOCK GENERATION

In this step, the inputs are the entities (provided by D_1 and D_2) and the attribute groups generated by the previous step (with their respective entropies), as illustrated in Algorithm 2. Initially, the entities are read and, for each attribute of a particular entity, the

[2] $Jaccard(V_1, V_2) = \frac{|V_1 \cap V_2|}{|V_1|+|V_2|-|V_1 \cap V_2|}$.

Algorithm 1: Schema Information Extraction step

```
Data: $D_1$, $D_2$ and $\Phi$: the input data sources and the similarity threshold
Result: $G$: attribute groups
$attribSignaturesD_1 \leftarrow extractSignatures(D_1)$;
$attribSignaturesD_2 \leftarrow extractSignatures(D_2)$;
$attribGroup \leftarrow \emptyset$;
foreach $a_1$ in $attribSignaturesD_1$ do
    foreach $a_2$ in $attribSignaturesD_2$ do
        if $sim(a_1.S,\ a_2.S) > \Phi$ then
            $\varepsilon \leftarrow shanonEntropy(a_1,\ a_2)$;
            $attribGroup.append(\langle a_2,\ a_1,\ \varepsilon\rangle)$;
        end
    end
end
return attribGroup
Function extractSignatures(D) : map$\langle a, S\rangle$ do
    $attribSignatures \leftarrow \emptyset$;
    foreach $e$ in $D$ do
        foreach $a$ in $A_e$ do
            $V \leftarrow keyWords(a_{value})$;
            if $attribSignatures.contains(a)$ then
                $attribSignatures.get(a).union(V)$;
            else
                $attribSignatures.put(a,\ V)$;
            end
        end
    end
    foreach $\langle a,\ V\rangle$ in $attribSignatures$ do
        $S \leftarrow minHash(V)$;
        $attribSignatures.replace(a,\ S)$;
    end
    return attribSignatures
end
```

Algorithm 2: Block Generation step

```
   Data: $D_1$, $D_2$, $G$: the input data sources and the attribute groups
   Result: $B$: blocks of entities
1  $mapOfBlocks \leftarrow \emptyset$;
2  foreach $e_1$ in $D_1$ do
3      foreach $a_1$ in $A_{e_1}$ do
4          $S \leftarrow LSH(a_1.value)$;
5          $blockKeys \leftarrow splitSignature(S)$;
6          foreach $blockKey$ in $blockKeys$ do
7              if $mapOfBlocks.contains(blockKey.a_1)$ then
8                  $mapOfBlocks.get(blockKey.a_1).append(e_1)$;
9              else
10                 $mapOfBlocks.put(\langle blockKey.a_1,\ [e_1]\rangle)$;
11             end
12         end
13     end
14 end
15 foreach $e_2$ in $D_2$ do
16     foreach $a_2$ in $A_{e_2}$ do
17         $S \leftarrow LSH(a_2.value)$;
18         $blockKeys \leftarrow splitSignature(S)$;
19         $attribOfD_1InSameGroup \leftarrow attribGroup.get(a_2)$;
20         foreach $blockKey$ in $blockKeys$ do
21             foreach $a_1$ in $attribInSameGroup$ do
22                 if $mapOfBlocks.contains(blockKey.a_1)$ then
23                     $mapOfBlocks.get(blockKey.a_1)$
                         $.append(e_2)$;
24                 end
25             end
26         end
27     end
28 end
29 return mapOfBlocks
```

LSH algorithm is applied to generate a signature for the attribute value (lines 2 to 5 and 15 to 18). An entity e is denoted as follows: $e = \{\langle a_1, S_1\rangle, \langle a_2, S_2\rangle, \langle a_3, S_3\rangle, ..., \langle a_k, S_k\rangle\}$. For instance, to generate the attribute signatures of entity e_1, in Figure 1, the attribute values $\langle a_{name},$ *Jon Snow*$\rangle$ and $\langle a_{house},$ *Stark*$\rangle$ are converted into the signatures $\langle a_{name}, [4, 9, 4, 6]\rangle$ and $\langle a_{house}, [1, 1, 6, 9]\rangle$, respectively. Notice that the hash function always generates signatures with the same size, in terms of elements.

After defining the signatures, the blocks are generated based on each of the signatures. To this end, the signatures are split into α equal parts, termed subsignatures s. The subsignatures are used as the key for each block to be generated (lines 5 and 18). Thereafter, the entities that share a particular subsignature originated from attributes contained in the same attribute group (provided by the previous step) are inserted into the same block (lines 6 to 12 and 19 to 26). Furthermore, since each group of attributes has an associated entropy, the blocks generated from attributes contained in a particular group will assume the same entropy value of the attribute group.

Regarding the time complexity, in this step, it is necessary to evaluate all $\langle a, S\rangle$ (derived from A_e, defined in Section 3) of each entity $e \in D$ to generate the blocking keys. Furthermore, to evaluate each $\langle a, S\rangle$, it is necessary to take into account the number of subsignatures s derived from S, denoted by α. Therefore, the time complexity of this step is $O((||A_{D_1}|| + ||A_{D_2}||) \cdot \alpha)$, where $||A_{D_1}||$ is given by $\sum_{e \in D_1} |A_e|$ and $||A_{D_2}||$ is given by $\sum_{e \in D_2} |A_e|$.

In Figure 1, the NA-BLOCKER technique generates two blocks (b_5 and b_6) with key *[5, 5]* since one subsignature is provided by the attributes $house/family$ ($\in g_2$) and the other one by the attributes $adversary/enemy$ ($\in g_3$). In this sense, the entities e_2 and e_5 are inserted into b_5 since both entities contain the subsignature *[5, 5]* and the attributes $adversary$ (in e_2) and $enemy$ (in e_5) are contained in the same attribute group (g_3). Similarly, the entities e_3 and e_6 are inserted into b_6 because both entities contain the subsignature *[5, 5]* and the attributes $house$ (in e_3) and $family$ (in e_6) are contained in g_2. It is important to highlight that, even though there exists noisy data in data source D_2 (e.g., "John Sn0w", "Stak", "Aria", "Lanister", and "Tiwin") the entities with similar attributes have been inserted into the same blocks. This occurs due to the fact that the NA-BLOCKER technique benefits from the application of LSH, which generates signatures of the attributes. The signatures maintain the degree of similarity between attribute values even though the set of keywords is transformed into an array of integers. Thereby, entities whose attribute values are similar, even with the presence of noisy data, present high chances to share several subsignatures and, consequently, will be inserted into the same blocks. On the other hand, if a token blocking technique is applied, entity pairs, such as $\langle e_2, e_5\rangle$ and $\langle e_3, e_6\rangle$, will not be inserted into the same block since they do not share the same tokens.

6 PRUNING

The goal of this step is to discard redundant comparisons between entities, as well as comparisons with few chances of resulting in correspondences, as illustrated in Algorithm 3. To this end, Metablocking-based techniques [6, 23, 26] can be applied. These techniques receive as input the blocks generated at the previous

Algorithm 3: Pruning step

```
Data: $G$, $B$: the attribute groups and the blocks of entities
Result: $B'$: pruned blocks
1  $mapEntities \leftarrow \emptyset$;
2  foreach $block$ in $mapOfBlocks$ do
3      $entities \leftarrow block.values$;
4      while entities.size > 1 do
5          $e_{current} \leftarrow entities.pop()$;
6          foreach $e$ in $entities$ do
7              $\varepsilon \leftarrow attribGroup.getEntropy(block.key)$;
8              if $mapEntities.contains(e_{current}.e)$ then
9                  $mapEntities.get(e_{current}.e).sumWeight(\varepsilon)$;
10             else
11                 $mapEntities.put(e_{current}.e, \varepsilon)$;
12             end
13         end
14     end
15 end
16 $B' \leftarrow WNP(mapEntities)$;
17 return B'
```

Algorithm 4: NA-BLOCKER

```
Data: $D_1$, $D_2$: input data sources
Result: $B'$: set of pruned blocks
1  $G \leftarrow SchemaInformationExtraction(D_1, D_2)$;
2  $B \leftarrow BlockGeneration(D_1, D_2, G)$;
3  $B' \leftarrow Pruning(G, B)$;
4  return B'
```

step. In Figure 1, notice that the entities e_1 and e_4 are contained in blocks b_1, b_2 and b_3. Therefore, these entities should be compared three times (i.e., redundant comparisons). To avoid the redundant comparisons, the Metablocking technique restructures the input blocks into new ones that involve significantly fewer comparisons, while maintaining the original level of effectiveness [23]. Then, the input blocks are converted into a blocking graph (lines 2 to 15), such that each node represents an entity and each edge (between a pair of nodes) denotes that the pair of nodes sharing at least one block in common (lines 4 to 14). Every edge is associated with a weight, based on the number of blocks in common between the pair of nodes (entities) linked by the edge and the entropy value associated with the blocks in common (line 7).

Regarding the influence of entropy in the weight of edges, the entropy value determines the relevance of the blocks, since not all the blocks have the same importance. Therefore, the edge weight is given by the sum of entropies ε associated with each block in common between the pair of nodes linked by the edge (lines 9 and 11). For instance, in Figure 1, the edge that links nodes e_1 and e_4 assumes the weight of 6, since the pair of entities shares three blocks: b_1 (from group $\langle g_1, 3.0\rangle$), b_2 (from group $\langle g_2, 1.5\rangle$), and b_3 (from group $\langle g_2, 1.5\rangle$). Thus, the edge weight, which links e_1 and e_4, is given by $3.0 + 1.5 + 1.5 = 6$.

Once the graph is built, it is pruned according to a pruning criterion, which eliminates low-weighted edges to skip part of the redundant comparisons. Regarding the pruning criteria, the works [23, 26] propose different pruning algorithms that can be applied in this step. Particularly, in this work, we apply the WNP-based pruning algorithm [23] since it has achieved better results than other competitors [6]. The WNP algorithm applies the node-centric pruning algorithm with a local weight threshold that is given by the average edge weight of each neighborhood.

Table 1: Datasets characteristics.

Pairs of Datasets	$\lvert D_1 \rvert$	$\lvert D_2 \rvert$	Duplicates	$\lvert A_1 \rvert$	$\lvert A_2 \rvert$
Abt-Buy	1,076	1,076	1,076	3	3
Amazon-GP	1,354	3,039	1,104	4	4
DBLP-ACM	2,616	2,294	2,224	4	4
DBLP-Scholar	2,516	61,353	2,308	4	4
IMDB-DBpedia	27,615	23,182	22,863	4	7

Concerning the time complexity, in this step, the complexity is given by the sum of the cost to evaluate the entity pairs in each block $b \in B$ (i.e., the cardinality of B) and the cost of the WNP pruning algorithm. The time complexity of the WNP algorithm is $O(|N_B| \cdot |E_B|)$ [23], where $|N_B|$ is the number of nodes and $|E_B|$ is the number of edges in the graph generated from B. Therefore, the time complexity of the pruning step is $O(||B||) + O(|N_B| \cdot |E_B|)$.

In Figure 1, the pruning criteria evaluates locally the weight of the edges and discards the low-weighted comparisons (i.e., comparisons with few chances to result in correspondences). Hence, the resulting graph (pruned graph) infers only three comparisons: $\langle e_1, e_4\rangle$, $\langle e_2, e_5\rangle$, and $\langle e_3, e_6\rangle$. The significant decrease in the number of comparisons occurs due to the enhancing in the quality of blocks built in the Block Generation step (Step 2), which includes in each block only truly similar entities. Therefore, such step becomes fundamental in order to provide high-quality blocks, in terms of effectiveness, especially in the presence of noisy data, since the blocks built in the Block Generation step directly influence the edge weight of the graph (in Pruning step). The overview of the NA-BLOCKER technique, with the application of each step described previously, is summarized in Algorithm 4.

7 EXPERIMENTS

In this section, we evaluate the NA-BLOCKER[3] technique against BLAST [26], the state-of-the-art method, in terms of effectiveness and efficiency. We run our experiments on a Windows 7 computer with 16GB of memory and Intel Core I7-4790 3.60 GHz. In our experimental evaluation, five real-world pairs of datasets[4] (provided by [26]) were used, as described in Table 1: i) Abt-Buy: product profiles provided by abt.com and buy.com; ii) Amazon-GP: product profiles provided by amazon.com and google.com; iii) DBLP-ACM: scientific article profiles provided by dblp.org and dl.acm.org; iv) DBLP-Scholar: scientific article profiles provided by dblp.org and scholar.google.com; and v) IMDB-DBpedia: movie profiles provided by imdb.com and dbpedia.org. Table 1 shows the amount of entities (D) and attributes (A) contained in each dataset as well as the number of duplicates (i.e., matches) present in each pair of datasets.

To measure the effectiveness of the techniques, three quality metrics have been applied: i) Pair Completeness (PC) - similar to recall - estimates the portion of correspondences that were identified, denoted by $PC = \frac{|D(B')|}{|D(E)|}$, where $|D(B')|$ is the amount of duplicate entities in the set of pruned blocks B' and $|D(E)|$ is the amount of duplicate entities in dataset E. PC takes values in the interval [0, 1], with higher values indicating a better result; ii) Pair Quality (PQ) -

[3] https://bitbucket.org/tbrasileiro/na-blocker/

[4] Available in the project repository.

similar to precision - estimates the portion of executed comparisons that result in correspondences, denoted by $PQ = \frac{|D(B')|}{||B'||}$, where $||B'||$ is the amount of comparisons to be performed in the pruned blocks. PQ takes values in [0, 1], with higher values indicating a better result; iii) F-Measure (FM) - defined as the harmonic mean between PC and PQ - is defined by $FM = \frac{2 \cdot PC \cdot PQ}{PC + PQ}$. Regarding efficiency, we measure the whole execution time, including all steps, of the techniques. Since the number of comparisons to be executed in the ER task directly impacts on the efficiency of the task as a whole, we also evaluate the *aggregate cardinality* measure of the blocking techniques that computes the total number of comparisons $||B'||$ for all generated (and pruned) blocks.

To evaluate the effectiveness results of the techniques in different scenarios of noisy data, we insert synthetically typos and misspellings (i.e., noise) into the attribute values of the entities contained in a dataset of each pair. In order to simulate the occurrence of typos/misspellings [14], for all attributes of an entity, one character of each token (i.e., relevant words) present in the attribute values is randomly exchanged by other characters, or additional characters are inserted into the tokens[5]. In this sense, we vary the level of noise in the dataset. The noise level varies between 0 (i.e., no noise is inserted into the attribute values of any entity) to 1 (i.e., noise is inserted into the attribute values of all entities). For instance, the noise level of 0.4 indicates that 40% of the entities (contained in the first dataset) had their attribute values modified (i.e., noise was inserted). For these experiments, the execution time results are given by the average of three executions (of the blocking techniques) for each dataset pair.

Effectiveness. Figure 2 illustrates the results of the comparative effectiveness analysis for each pair of datasets. Regarding the effectiveness metrics (i.e., pair completeness, pair quality and F-Measure), NA-BLOCKER outperforms BLAST for all variations of noise level. It is important to highlight that as the noise level increases, the effectiveness metrics decrease for both techniques. This decrease occurs due to the fact that the noise on the data negatively interferes the block generation, as discussed in Section 2. However, the decrease in effectiveness metrics for BLAST occurs abruptly when compared to NA-BLOCKER. Since the latter applies strategies to tolerate noisy data (addressed in Section 3), the effectiveness decrease is amortized. Even for a high level of noise (i.e., a noise level of 1.0), NA-BLOCKER has achieved a pair completeness greater than 60% for the first three datasets.

Regarding F-Measure (FM), NA-BLOCKER reaches an average, with respect to all pairs of datasets, of 48% in the proportional decrease[6], whereas BLAST reaches 90%. The most significant result achieved by NA-BLOCKER was in terms of pair quality. In this case, it achieved an average pair quality (considering all pairs of datasets) two times better than the BLAST technique, in scenarios without noisy data. The main reason for that is the generation of multiple tokens per entity attribute (based on a particular attribute value) as blocking keys, in the BLAST technique. Since non-matching entities eventually share multiple tokens, they are included in the same block erroneously. On the other hand, the proposed technique generates a single hash value based on a particular attribute value. Thus, non-matching entities sharing the same hash value are harder to occur than non-matching entities sharing tokens in common. For this reason, NA-BLOCKER enhances the pair quality metric. Finally, based on the experimental results and the pair-wise (considering the level of noise) distribution *T-Student* test (with 95% of confidence), we concluded that our technique has achieved a better effectiveness than the BLAST technique.

Efficiency. Regarding efficiency, the execution time (in seconds) of the NA-BLOCKER and BLAST techniques are evaluated for each pair of datasets, as depicted in Figure 3. BLAST achieves better results than NA-BLOCKER for all pairs of datasets. On average, the NA-BLOCKER increased the execution time around 30% (or seven seconds). These results are already expected, since the proposed technique requires more time to block the entities. This is due to the fact that our technique needs more time to generate the LSH-signatures and determine the similarity of their attribute values based on the approximate similarity.

On the other hand, it is important to highlight that NA-BLOCKER achieved better results compared to BLAST regarding the *aggregate cardinality* (i.e., $||B'||$) for all dataset pairs, as depicted in Figure 4. In other words, NA-BLOCKER requires less comparisons to be executed in the ER task. On average, the blocks generated by NA-BLOCKER indicate a total number of comparisons 34% (or more than seven thousand comparisons) less when compared to the blocks generated by BLAST. Thus, the efficiency results achieved by NA-BLOCKER may be compensated by efficiency gains generated by the execution of fewer comparisons between entities to be performed in the following steps of the ER task, particularly the Comparison step. For instance, considering that the comparison of an entity pair in the ER task requires one unit of time, then NA-BLOCKER will provide a reduction of 34% in the execution time of the Comparison step. Therefore, NA-BLOCKER does not significantly affect the efficiency of the ER task as a whole, since the Comparison step is the most costly step (in terms of execution time) of the ER task [4].

8 RELATED WORK

Over the years, several works proposed matching approaches related to the Web context, such as entity resolution [11, 12], link discovery [20, 25], instance matching [13], and knowledge graphs [31]. However, these works cannot be directly compared against blocking techniques since matching approaches determine correspondent individuals (i.e., matches), while blocking techniques group similar individuals. In this sense, since the blocking techniques aim to reduce the number of comparisons to be performed in matching tasks (i.e., efficiency gains), the NA-BLOCKER approach can be applied as a preprocessing step for matching tasks.

Recently, several works, which address blocking techniques for ER, have been published [3, 6, 17–19, 21, 23, 26]. However, some blocking techniques [17–19] need a schema alignment to block the entities. Therefore, in the context of heterogeneous data, the schema-based blocking techniques are not suitable. This way, schema-agnostic blocking techniques [3, 6, 21, 23, 26] were proposed. In [5], schema-agnostic blocking techniques are described and classified according to their features: Token Blocking, Attribute Clustering Blocking, Frequent Itemsets, and Metablocking. Although

[5] All data sources (with/without noise) are available at the project repository.

[6] $Proportional\ decrease = 1 - \frac{FM(noise{=}1.0)}{FM(noise{=}0.0)}$

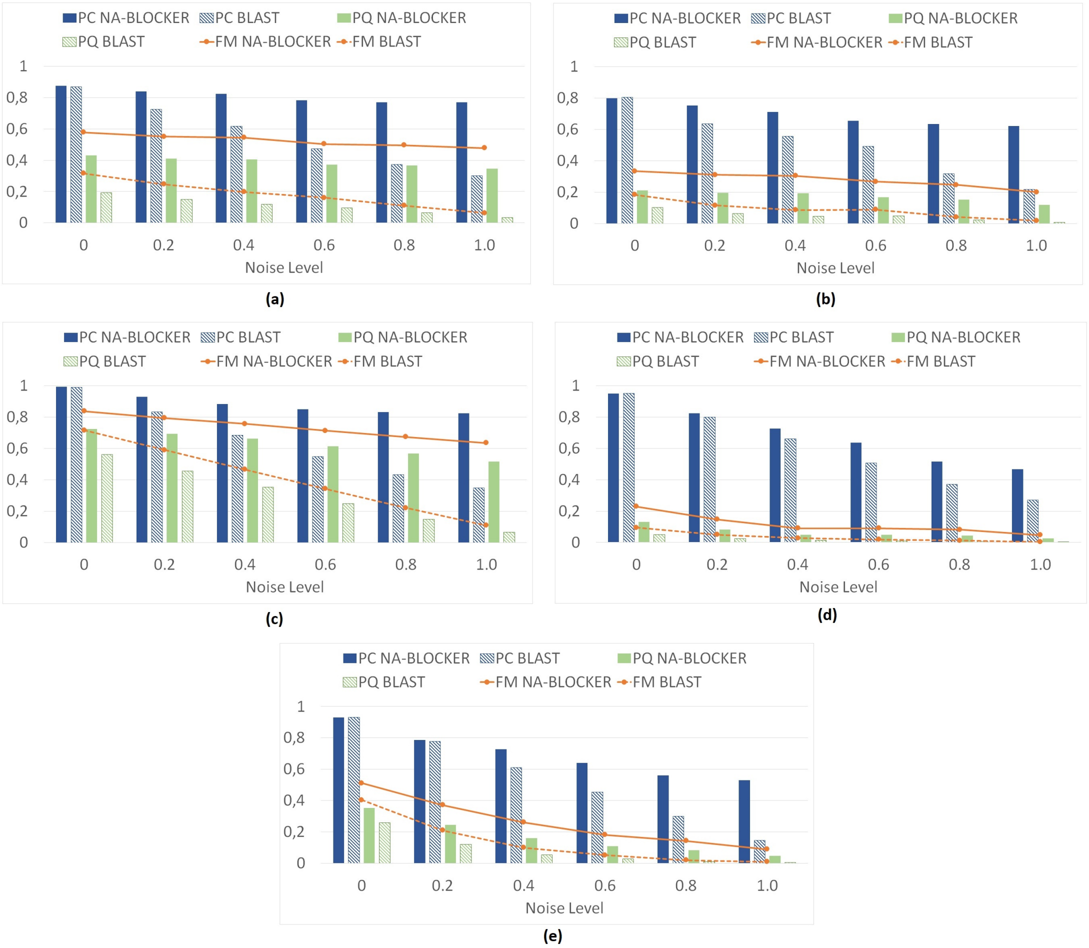

Figure 2: Effectiveness results of the datasets: (a) Abt vs. Buy, (b) Amazon vs. Google Product, (c) DBLP vs. ACM, (d) DBLP vs. Google Scholar, and (e) IMDB vs. DBpedia.

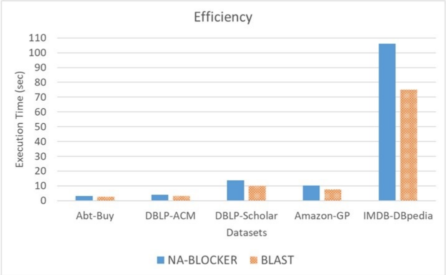

Figure 3: Execution time of NA-BLOCKER and BLAST techniques.

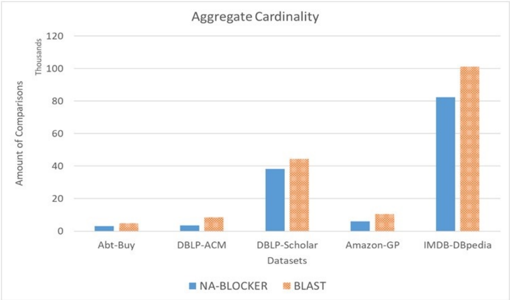

Figure 4: Aggregate Cardinality of NA-BLOCKER and BLAST techniques.

the Token Blocking technique presents satisfactory results regarding effectiveness [21], it does not reduce satisfactorily the amount of comparisons to be performed in the ER task. In this sense, the works [3, 6, 23] propose the Metablocking technique (and its variations), which aims to further reduce the comparisons between entities determined by the Token Blocking technique. Metablocking builds a weighted graph based on the blocks generated by Token Blocking and applies pruning algorithms to discard entity pairs that have low chances to result in matching.

More recently, the BLAST technique [26] applies the loose schema information strategy in order to collect statistical information about the schema directly from the data. Based on the statistical information, the attributes are partitioned (clustered) according to the similarity of their values, following the strategy proposed in [16]. Thereafter, the BLAST technique employs Token Blocking [21] to disambiguate some tokens by exploiting the attribute partitioning. Thus, only entities whose tokens belong to attributes in the same partition will be compared. This information is used by Metablocking to enhance the quality of the blocks. Although NA-BLOCKER follows a workflow similar to BLAST, there are differences related to the execution of the steps. BLAST adopts LSH only to determine the linkages between attributes of two large data sources in order to address efficiency issues. To determine the entity blocks, BLAST applies the traditional Token Blocking technique. On the other hand, NA-BLOCKER applies LSH (through the signatures) to guide the whole process of block generation, achieving better results regarding effectiveness and aggregate cardinality (discussed in Section 4).

In contrast to the previously mentioned works, our work provides a novel schema-agnostic blocking technique that is able to tolerate noise on data (particularly, in the attribute values). Furthermore, we also propose the application of LSH to guide the building of high-quality blocks, minimizing the negative impact of noisy data in the effectiveness of the blocking results. In particular, differently from Attribute Clustering Blocking [22] (which also benefits from the attributes information), our work applies the attribute-match induction strategy that induces groups of similar attributes from the distribution of the attribute values, without exploiting the semantics of the attribute names or external information (e.g., thesaurus or dictionaries).

9 CONCLUSIONS AND FUTURE WORK

Blocking techniques are largely applied as a preprocessing step in ER approaches in order to avoid the quadratic cost of the ER task. In this context, heterogeneous data and noisy data increase the difficulties faced by blocking techniques. In this paper, we propose the NA-BLOCKER technique, which is capable of tolerating noisy data, extracting information regarding the schema of the data sources, generating groups of similar attributes, and pruning the generated blocking results in order to enhance the quality of the final blocks. Since Web approaches need to deal with data sources that present noisy and heterogeneous data, the proposed technique can be useful for these approaches, such as LIMES [20], LOV [28] and JedAI [24]. Based on the experimental results, we can highlight that NA-BLOCKER presents better results regarding effectiveness and aggregate cardinality than the state-of-the-art technique.

ACKNOWLEDGMENTS

This work has been partially supported by the Virpa D project funded by Business Finland.

REFERENCES

[1] Sumeet Agarwal, Shantanu Godbole, Diwakar Punjani, and Shourya Roy. 2007. How much noise is too much: A study in automatic text classification. In *ICDM*.

[2] Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. 2014. Beyond locality-sensitive hashing. In *ACM-SIAM Symposium on Discrete Algorithms*.

[3] Tiago Brasileiro Araújo, Carlos Eduardo Santos Pires, and Thiago Pereira da Nóbrega. 2017. Spark-based Streamlined Metablocking. In *ISCC*.

[4] Peter Christen. 2012. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection.* Springer Science & Business Media.

[5] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. 2015. Entity Resolution in the Web of Data. *Synthesis Lectures on the Semantic Web* 5, 3 (2015).

[6] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. 2015. Parallel meta-blocking: Realizing scalable entity resolution over large, heterogeneous data. In *IEEE Big Data*.

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

[8] Vasilis Efthymiou, Kostas Stefanidis, and Vassilis Christophides. 2015. Big data entity resolution: From highly to somehow similar entity descriptions in the Web. In *IEEE Big Data*.

[9] Salvador García, Julián Luengo, and Francisco Herrera. 2015. *Data preprocessing in data mining.*

[10] Raiza Hanada, Maria da Graça C Pimentel, Marco Cristo, and Fernando Anglada Lores. 2016. Effective Spelling Correction for Eye-based Typing using domain-specific Information about Error Distribution. In *CIKM*.

[11] Lars Kolb, Andreas Thor, and Erhard Rahm. 2012. Dedoop: efficient deduplication with Hadoop. *PVLDB* 5, 12 (2012), 1878–1881.

[12] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. 2016. Magellan: Toward building entity matching management systems. *PVLDB* 9, 12 (2016), 1197–1208.

[13] Juanzi Li, Zhichun Wang, Xiao Zhang, and Jie Tang. 2013. Large scale instance matching via multiple indexes and candidate selection. *Knowledge-Based Systems* 50 (2013), 112–120.

[14] Huizhi Liang, Yanzhe Wang, Peter Christen, and Ross Gayler. 2014. Noise-tolerant approximate blocking for dynamic real-time entity resolution. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*.

[15] Jianhua Lin. 1991. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information theory* 37, 1 (1991), 145–151.

[16] Yongtao Ma and Thanh Tran. 2013. Typimatch: Type-specific unsupervised learning of keys and key values for heterogeneous web data integration. In *WSDM*.

[17] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*.

[18] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. 2016. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.

[19] Demetrio Gomes Mestre, Carlos Eduardo Santos Pires, Dimas Cassimiro Nascimento, Andreza Raquel Monteiro de Queiroz, Veruska Borges Santos, and Tiago Brasileiro Araujo. 2017. An efficient spark-based adaptive windowing for entity matching. *Journal of Systems and Software* 128 (2017), 1–10.

[20] Axel-Cyrille Ngonga Ngomo and Sören Auer. 2011. Limes-a time-efficient approach for large-scale link discovery on the web of data.. In *IJCAI*.

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

[22] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederee, and Wolfgang Nejdl. 2013. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE TKDE* 25, 12 (2013), 2665–2682.

[23] George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl. 2014. Meta-blocking: Taking entity resolutionto the next level. *IEEE TKDE* 26, 8 (2014), 1946–1960.

[24] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. 2017. JedAI: The Force behind Entity Resolution. In *ESWC*.

[25] Minh C Phan, Aixin Sun, Yi Tay, Jialong Han, and Chenliang Li. 2017. NeuPL: Attention-based Semantic Matching and Pair-Linking for Entity Disambiguation. In *CIKM*.

[26] Giovanni Simonini, Sonia Bergamaschi, and HV Jagadish. 2016. BLAST: a loosely schema-aware meta-blocking approach for entity resolution. *PVLDB* 9, 12 (2016), 1173–1184.

[27] Kostas Stefanidis, Vassilis Christophides, and Vasilis Efthymiou. 2017. Web-Scale Blocking, Iterative and Progressive Entity Resolution. In *ICDE*.

[28] Pierre-Yves Vandenbussche, Ghislain A Atemezing, María Poveda-Villalón, and Bernard Vatant. 2017. Linked Open Vocabularies (LOV): a gateway to reusable semantic vocabularies on the Web. *Semantic Web* 8, 3 (2017), 437–452.

[29] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. 2016. Learning to hash for indexing big data - a survey. *Proc. IEEE* 104, 1 (2016), 34–57.

[30] Yang Yang, Yizhou Sun, Jie Tang, Bo Ma, and Juanzi Li. 2015. Entity matching across heterogeneous sources. In *SIGKDD*.

[31] Linhong Zhu, Majid Ghasemi-Gol, Pedro Szekely, Aram Galstyan, and Craig A Knoblock. 2016. Unsupervised entity resolution on multi-type graphs. In *ISWC*.