Benchmarking Blocking Algorithms for Web Entities

Vasilis Efthymiou, Kostas Stefanidis, Vassilis Christophides

This is the accepted manuscript of the article, which has been published in *IEEE Transactions on Big Data*.
http://dx.doi.org/10.1109/TBDATA.2016.2576463

Benchmarking Blocking Algorithms for Web Entities

Vasilis Efthymiou, Kostas Stefanidis, Vassilis Christophides

Abstract—An increasing number of entities are described by interlinked data rather than documents on the Web. Entity Resolution (ER) aims to identify descriptions of the same real-world entity within one or across knowledge bases in the Web of data. To reduce the required number of pairwise comparisons among descriptions, ER methods typically perform a pre-processing step, called *blocking*, which places similar entity descriptions into blocks and thus only compare descriptions within the same block. We experimentally evaluate several blocking methods proposed for the Web of data using real datasets, whose characteristics significantly impact their effectiveness and efficiency. The proposed experimental evaluation framework allows us to better understand the characteristics of the missed matching entity descriptions and contrast them with ground truth obtained from different kinds of relatedness links.

Index Terms—Blocking, Entity Resolution, Web of Data.

1 Introduction

Over the past decade, numerous *knowledge bases* (KBs) have been built to power large-scale knowledge sharing, but also an entity-centric Web search, mixing both structured data and text querying. These KBs offer comprehensive, machine-readable descriptions of a large variety of real-world entities (e.g., persons, places) published on the Web as *Linked Data* (LD). Traditionally, KBs are manually crafted by a dedicated team of knowledge engineers, such as the pioneering projects Wordnet and Cyc. Today, more and more KBs are built from existing Web content using information extraction tools. Such an automated approach offers an unprecedented opportunity to scale-up KBs construction and leverage existing knowledge published in HTML documents.

Although KBs (e.g., DBpedia, Freebase) may be derived from the same data source (e.g., Wikipedia), they may provide multiple descriptions of the same entities. This is mainly due to the different information extraction tools and curation policies employed by KBs, resulting to complementary and sometimes conflicting descriptions. *Entity resolution* (ER) aims to identify descriptions that refer to the same entity within or across KBs [1], [2]. ER is essential in order to improve *interlinking* in the Web of data, even by third-parties. In particular:

- The size of the Linking Open Data (LOD) cloud[1], in which nodes are KBs (aka RDF datasets) and edges are links crossing KBs, has roughly doubled between 2011 and 2014 [3], while data interlinking dropped by 30%. In general, the majority of the KBs are sparsely linked, while their popularity in links is heavily skewed[2]. Sparsely interlinked KBs appear in the periphery of the LOD cloud (e.g., Open Food Facts, Bio2RDF), while heavily interlinked ones lie at the center (e.g., DBpedia, GeoNames, FOAF). Encyclopaedic KBs, such as DBpedia, or widely used georeferencing KBs, such as GeoNames, are interlinked with the largest number of KBs both from the LOD center and the periphery.
- The descriptions contained in these KBs present a high degree of *semantic* and *structural* diversity, even for the same entity types. The former is due to the frequent creation of new names for entities that have been described in another KB, as well as the simultaneous annotation of descriptions with semantic types not necessarily originating from the same vocabulary. The latter is due to the diverse sets of properties used to describe entities both in terms of types and number of occurrences, even within a KB.

The *scale*, *diversity* and *graph structuring* of entity descriptions in the Web of data challenge the way two descriptions can be effectively compared in order to efficiently decide whether they are referring to the same real-world entity. This clearly requires an understanding of the relationships among *somehow similar* entity descriptions that goes beyond duplicate detection without always being able to merge related descriptions in a KB and thus improve its quality. Furthermore, the *very large volume* of entity collections that we need to resolve in the Web of data is prohibitive when examining pairwise all descriptions.

In this context of big Web data, *blocking* is typically used as a pre-processing step for ER to reduce the number of unnecessary comparisons, i.e., comparisons between descriptions that do not match. After blocking, each description can be compared only to others placed within the same block. The desiderata of blocking are to place (i) similar descriptions in the same block (*effectiveness*), and (ii) dissimilar descriptions in different blocks (*efficiency*). However,

- *V. Efthymiou is with the University of Crete and ICS-FORTH, Greece. E-mail: {vefthym,kstef}@ics.forth.gr*
- *K. Stefanidis is with the University of Tampere, Finland (work done while at ICS-FORTH). E-mail: ksts.stefanidis@gmail.com*
- *V. Christophides is with the University of Crete, Greece, and INRIA, Paris-Rocquencourt, France. E-mail: Vassilis.Christophides@inria.fr*

Manuscript received Month dd, 2016; revised Month dd, 2016.

1. http://lod-cloud.net

2. http://linkeddata.few.vu.nl/wod_analysis

efficiency dictates skipping many comparisons, possibly leading to many missing matches, which in turn implies low effectiveness. Thus, the main objective of blocking is to achieve a trade-off between the number of comparisons suggested and the number of missed matches.

Most of the blocking algorithms proposed in the literature (for a survey, refer to [4]) assume both the availability and knowledge of the schema of the input data, i.e., they refer to relational databases. To support a Web-scale resolution of heterogeneous and loosely structured entities across domains, recent blocking algorithms (e.g., [5], [6]) disregard strong assumptions about knowledge of the schema of data and rely on a minimal number of assumptions about how entities match (e.g., when they feature a common token in their description or URI) within or across sources. However, these algorithms have not yet been experimentally evaluated with LOD datasets exhibiting different characteristics in terms of the underlying number of entity types and size of entity descriptions (in terms of property-value pairs), as well as their structural (i.e., property vocabularies) and semantic (i.e., common property values and URLs) overlap.

In summary, in this paper:

- We design a large-scale evaluation on a cluster of 15 machines using real data. To capture the differences in the heterogeneity and overlap of entity descriptions, we distinguish between data originating from sources in the *center* (i.e., heavily interlinked) and the *periphery* (i.e., sparsely interlinked) of the LOD cloud.
- We empirically study the behavior of blocking algorithms for datasets exhibiting different semantic and structural characteristics. We are interested in quantifying the factors that make blocking algorithms take different decisions on whether two descriptions from real LOD sources potentially match or not.
- We investigate typical cases of missed matches of existing blocking algorithms and examine alternative ways for them to be retrieved. We finally present the results of blocking, when other kinds of links, different to *owl:sameAs*, are used as a ground truth.

The rest of the paper is organized as follows[3]: Section 2 describes the problem and existing solutions. Section 3 presents our implementation of blocking algorithms in MapReduce. Sections 4 and 5 analyze the setup and the evaluation results of our experiments, respectively. Section 6 overviews works related to ER and, finally, Section 7 summarizes the paper.

2 Blocking Algorithms

We consider that an *entity description* is expressed as a set of attribute-value pairs. Then, *entity resolution* is the problem of identifying descriptions of the same entity (called *matches*). In general, ER can be distinguished between *pairwise* and *collective*. *Pairwise ER* (e.g., [8]) compares two descriptions at a time, depending only on the data contained in these descriptions. *Collective ER* (e.g., [9]) compares a set of related descriptions, heavily relying on similarity evidence provided by neighboring descriptions.

Given as input of ER the descriptions of Figure 1, $\mathcal{E} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$, a possible output $P = \{\{e_1, e_6\}, \{e_2, e_5\}, \{e_3\}, \{e_4\}, \{e_7\}\}$ indicates that the descriptions e_1 and e_6 refer to the same real-world object, namely Eiffel Tower, e_2 and e_5 both represent another object, the Statue of Liberty, and e_3, e_4 and e_7 represent by themselves the entities Auguste Bartholdi, Joan Tower and Bartholdi Fountain, respectively. Such a collection is called *dirty*, since it contains duplicates, and the corresponding task is called *dirty ER*, while *clean-clean ER* is a special case of (dirty) ER [6], [10]; $\mathcal{E}$ consists of two clean, i.e., duplicate-free, but possibly overlapping collections, and ER targets at identifying their common descriptions[4]. In this work, we focus on both cases.

e_1 = {(about, Eiffel Tower), (architect, Sauvestre), (year, 1889), (located, Paris)}
e_2 = {(about, Statue of Liberty), (architect, Bartholdi Eiffel), (year, 1886), (located, NY)}
e_3 = {(about, Auguste Bartholdi), (born, 1834), (work, Paris)}
e_4 = {(about, Joan Tower), (born, 1938)}
e_5 = {(work, Lady Liberty), (artist, Bartholdi), (location, NY)}
e_6 = {(work, Eiffel Tower), (year-constructed, 1889), (location, Paris)}
e_7 = {(work, Bartholdi Fountain), (year-constructed, 1876), (location, Washington)}

Fig. 1. A set of entity descriptions.

Given $\mathcal{E}$, we define a blocking collection as a set of blocks containing the descriptions in $\mathcal{E}$.

Definition 1 (Blocking collection)**.** *Let $\mathcal{E}$ be a set of entity descriptions. A blocking collection is a set of blocks $B = \{b_1, \ldots, b_m\}$, such that, $\bigcup_{b_i \in B} b_i = \mathcal{E}$.*

In general, blocking can be used both for pairwise and collective ER, to reduce the number of required comparisons. Specifically, *token blocking* [6] relies on the minimal assumption that matching descriptions should at least share a common token. Each distinct token t in the values of a description, defines a new block b_t, essentially building an inverted index of descriptions. Two descriptions are placed in the same block, if they share a token in their values.

Given the entity collection of Figure 1, Figure 2 shows the blocks generated by token blocking. In the generated blocks, we save the comparisons (e_1, e_5), (e_1, e_7), (e_2, e_4), (e_3, e_4), (e_4, e_5), (e_5, e_6) and (e_6, e_7), and we successfully place the matches (e_1, e_6) and (e_2, e_5) in common blocks. Still, pairs, such as (e_1, e_2), (e_1, e_3), and (e_3, e_6), lead to unnecessary comparisons. Note also that the pair (e_1, e_6) is contained in 4 different blocks, which leads to redundant comparisons.

Next, we present two extensions: attribute clustering blocking, in which candidate matches should at least share a common token for similar attributes known globally, and prefix-infix(-suffix) blocking, in which candidate matches should additionally share a common URI infix.

Attribute clustering blocking [6] exploits schematic information of the descriptions to minimize the number of unnecessary comparisons. To achieve this, prior to token blocking, it clusters attributes based on the similarities of their values over the entire entity collection. Each attribute from one collection is connected to its most similar attribute in the other collection and connected attributes, taken by transitive closure, form non-overlapping clusters. Then, each token t in the values of an attribute, belonging to a cluster c, defines a block $b_{c.t}$. Hence, comparisons between

3. A preliminary abridged version of this paper appeared in [7].

4. *Dirty-dirty* and *clean-dirty ER* can be seen as equivalent to *dirty ER*. Other names include *record-linkage* (for linking clean KBs) and *deduplication* (for merging duplicates in a dirty KB).

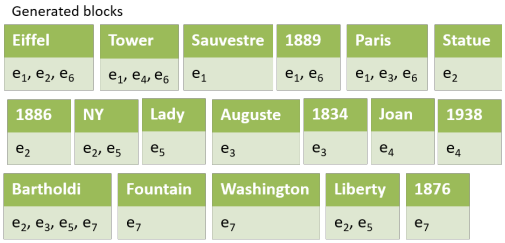

Fig. 2. Token blocking example. Descriptions having a common token are placed in a common block.

descriptions without a common token in a similar attribute, are discarded. Like token blocking, attribute clustering generates overlapping blocks. Compared to the blocks of token blocking, it produces a larger number of smaller blocks.

As an example, consider that the descriptions of Figure 1 consist of two clean collections, $D_1 = \{e_1, e_2, e_3, e_4\}$ and $D_2 = \{e_5, e_6, e_7\}$. Using Jaccard similarity, the attribute $work$ (with values: {Lady, Liberty, Eiffel, Tower, Bartholdi, Fountain}) of D_2 is the most similar attribute to $about$ of D_1. Similarly, the transitive closure of the pairs of most similar attributes between D_1 and D_2 (Figure 3(a) depicts such pairs), produce the clusters of attribute names (Figure 3(b)). A subset of the blocks constructed for each cluster is shown in Figure 3(c). This way, the comparisons (e_1, e_3) and (e_3, e_6) that were suggested by token blocking, due to the common token *Paris*, are now discarded, since the token *Paris* appears in different attribute clusters for e_3 than for e_1 and e_6, as shown in the bottom blocks of Figure 3(c). Again, both unnecessary (e.g., e_4 and e_6 are both placed in block $C1.Tower$ (Figure 3(c))), and redundant (e.g., (e_1, e_3) is still contained in 4 different blocks) comparisons are generated.

Unlike previous methods analyzing the content of descriptions, prefix-infix(-suffix) blocking [5] exploits the naming pattern in the descriptions' URIs. The prefix describes the domain of the URI, the infix is a local identifier, and the optional suffix contains details about the format, or a named anchor. For example, the prefix of "http://liris.cnrs.fr/olivier.aubert/foaf.rdf#me" is "http://liris.cnrs.fr", the infix is "/olivier.aubert" and the suffix is "/foaf.rdf#me". Given a set of descriptions, this method creates one block for each token in the descriptions literal values and one block for each URI infix. It is constrained by the extent to which common naming policies are followed by the KBs. In a favourable scenario, it creates additional blocks than token blocking for the names of the descriptions, which enables to consider matching descriptions, even with no common tokens in their literal values.

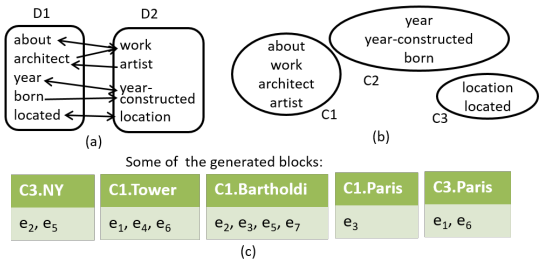

Fig. 3. Attribute clustering blocking example. Pairs of most similar attributes are linked (a). Connected attributes form clusters (b). Descriptions with a common token in the values of attributes of the same cluster, are placed in a common block (c).

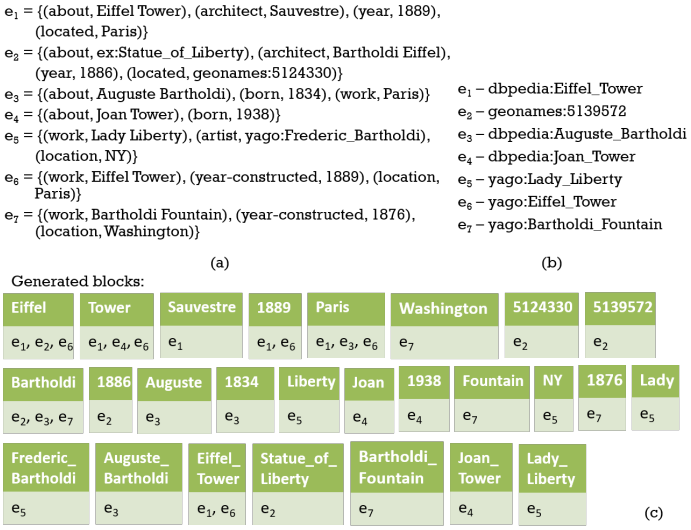

Fig. 4. Prefix-infix(-suffix) blocking example. A set of descriptions (a), their subject URIs (b), and the blocks from their tokens and infixes (c).

Figure 4(c) shows the blocks produced after applying prefix-infix(-suffix) blocking to the descriptions of Figure 4(a) (the descriptions of Figure 1, slightly modified to illustrate the characteristics of the method), while Figure 4(b) presents the URI identifiers of the descriptions.

Recent works have proposed using an iterative ER process, interleaved with blocking. The intuition is that the ER results of a processed block, may help identifying more matches in another block. Specifically, iterative blocking [11], applied on the results of blocking, examines one block at a time looking for matches. When a match is found in a block, the resulting merging of the descriptions that match is propagated to all other blocks, replacing the initial matching descriptions. This way, redundant comparisons between the same pair of descriptions in different blocks are saved and, in addition, more matches can be identified efficiently. The same block may be processed multiple times, until no new matches are found. Inherently, iterative block-

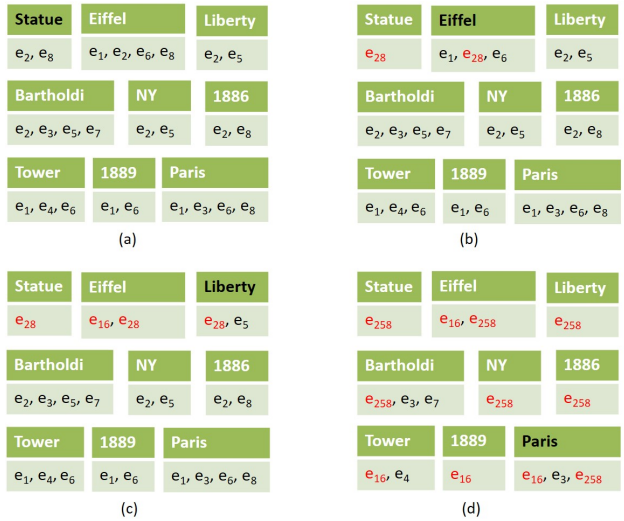

Fig. 5. Iterative blocking example. Given a set of blocks (a), the matches of each block are merged, propagating the results to the subsequent blocks (b),(c), until no more merges are possible (d).

TABLE 1
Criteria for placing descriptions in the same block.

Method	Criterion
Token Blocking	The descriptions have a common token in their values.
Attribute Clustering Blocking	The descriptions have a common token in the values of attributes that have similar values in overall.
Prefix-Infix(-Suffix) Blocking	The descriptions have a common token in their literal values, or a common URI infix.
Iterative Blocking	The descriptions are placed in a block by a blocking method, or after replacing descriptions that were merged in a previous iteration.

ing follows a sequential execution model, since the results of ER in one block directly affect other blocks.

Given the results of token blocking for the dirty collection of Figure 1 and an additional description e_8 = {(work, Statue of Lib.), (architect, Eiffel), (year-constructed, 1886)}, the process of iterative blocking is presented in Figure 5. The results of token blocking, excluding blocks with a single description, are shown in Figure 5 (a). Starting from the block $Statue$, (e_2, e_8) is found to be matching. The matching descriptions are merged into a new description e_{28}, representing both. Hence, in the next step (Figure 5 (b)), e_2 and e_8 are replaced by e_{28} in block $Eiffel$, so they are not compared again. Also, another match (e_1, e_6) is found in this block and the result of the merging e_{16} is replacing the descriptions e_1 and e_6 in the following step ((Figure 5 (c))). Also in this step, e_{28} is compared to e_5 and they are found to match, creating the new description e_{258}. Note that this match could not be identified by the other blocking methods, since e_5 and e_8 did not share any common blocks. Finally, when no new merges occur in an iteration, after processing all the blocks, the algorithm terminates, yielding the results $P = \{\{e_{16}\}, \{e_{258}\}, \{e_3\}, \{e_4\}, \{e_7\}\}$, as shown in Figure 5 (d). Those are the results of ER and not blocking.

Overall, Table 1 summarizes, simplified, the criteria employed by the aforementioned blocking methods to place two descriptions into the same block. In this context, the following questions naturally arise:

- Which blocking method performs best and for which dataset characteristics?
- How could we evaluate a blocking method, and its ability to reduce the number of suggested comparisons versus its ability to correctly place matching descriptions in common blocks?
- Does blocking place all matches in common blocks and what are the characteristics of the missed matches?
- Could blocking be used for identifying other types of relations, e.g., geographical, between descriptions?

3 MapReduce Implementation

Next, we present the MapReduce version of the evaluated methods, designed to cope with Web data[5]. Note that iterative blocking [11] is an inherently sequential process, hence, we do not provide an implementation for it in MapReduce.

3.1 Token Blocking

Token blocking is essentially an inverted index of descriptions. Each token is a key in this index, associated with a list of all the descriptions containing it. Our implementation

5. Source code and datasets available at csd.uoc.gr/~vefthym/minoanER/.

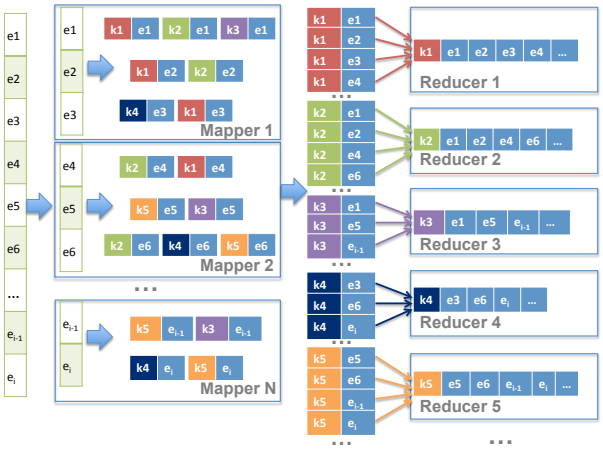

Fig. 6. Token blocking in MapReduce.

of token blocking in MapReduce is based on the procedure illustrated in Figure 6. In the map phase, one entity description of the local input split is processed at a time. For each token t in the values of a description e_i, a (t, e_i) pair is emitted by the mapper. In the reduce phase, all descriptions having a common token will be processed by the same reduce function, i.e., placed in the same block.

3.2 Attribute Clustering Blocking

Given two clean entity collections, our implementation of attribute clustering blocking can be briefly sketched by the following steps, each representing a MapReduce job. Figure 7 illustrates a high-level flow of the process.

Attribute Creation. First, we gather the values of each attribute. In the map phase, we emit an (*attribute*, *value*) pair for each attribute-value pair in a description. We also keep the collection of this attribute in the *key*. In the reduce phase, all the values of an attribute are grouped together and their concatenation is emitted as the value of this attribute.

Attribute Similarities. In the second job, we compute the pairwise Jaccard similarities between the trigram sets of all attributes. A mapper outputs each input attribute, as many times, as the number of total mappers. Each time, a composite *key*, consisting of the current mapper id and another mapper id, will determine in which reducer the attribute will be placed, and to which other attributes it will be compared. For example, assuming 3 mappers in total, the mapper with id 2, emits for each input attribute, 3 different *keys*: 1_2, 2_2, and 2_3. The keys 1_2 and 2_3 will result in comparing the contents of mapper 2 to the contents of mappers 1 and 3, while 2_2 will result in comparing the contents of mapper 2 to each other. The *value* of each emitted pair is the input attribute with its values and the current mapper id. In the reduce phase, we compute similarities of attributes, ensuring that each comparison is performed once. For each pair of attributes, we emit a (*key*, *value*) pair, with one attribute being the *key* and the second attribute along with their similarity score being the *value*.

Best Match. In the third job, we use an identity mapper, which just forwards its input. A combiner keeps for each attribute of each collection, only the attribute of the other collection with the local highest similarity score. In the reduce phase, we pick for each attribute of each collection,

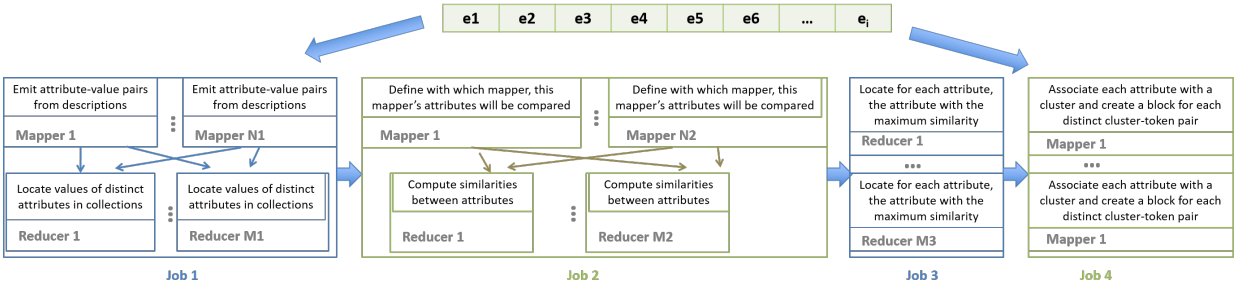

Fig. 7. Attribute clustering blocking in MapReduce.

the attribute with the maximum similarity score, in overall, from the other collection. Before this job ends, we start the first step of clustering the most similar attributes together. To accomplish that, we emit for each best-matching attribute pair, two (attribute, clusterId) pairs, one for each attribute, with the same clusterId. Ids of clusters with common attributes are marked, in order to be merged at the next step.

Final Clustering and Blocking. In the final job, we associate each attribute with a final cluster id, according to the marks of the previous step. Then, we perform token blocking (Section 3.1), with only difference that in each *key* emitted from a mapper, there is also a cluster prefix, enabling distinctions between blocks for the same token. For example, if the same token t appears in a description e_i for attributes in clusters c_j and c_k, then the mapper will emit the pairs $(c_j.t, e_i)$ and $(c_k.t, e_i)$, instead of a single (t, e_i).

3.3 Prefix-Infix(-Suffix)

Our MapReduce implementation of this method consists of three jobs. The first two are the MapReduce adaptation of the infix extraction algorithm [5]. The third job reads the descriptions, as well as the infixes produced by the second job and creates the blocks. A high-level representation of the process is depicted in Figure 8.

Prefix Removal. In the map phase, we output a (*key*, *value*) pair for each URI in a description. The *key* is the second token of the URI (after "http") and the *value* consists of the whole URI and the identifier of the entity description having this URI. This clusters the URIs according to their second token, which usually represents the domain (e.g., "dbpedia"), in the reduce phase. For each URI in a cluster, we find, among all its possible prefixes, the one with the largest set of distinct (immediately) next tokens. The part of the URI following the prefix is the *key* of each output pair, with *value* consisting of the input *key*, i.e., the second token of the URI, and the entity identifier having this URI.

Suffix Removal. We apply Prefix Removal, on each reverse URI (without prefix), to remove the suffix.

Infix&Token Blocking. We create the final blocks, based on the output of Suffix Removal and the initial entity collection. We use two different mappers, operating in parallel; an identity mapper, forwarding the output of Suffix Removal and the mapper of token blocking, operating on the tokens of literal values only of the input descriptions. In the reduce phase, all the descriptions having a common token or infix in their literals or URIs will be placed in the same block.

4 Experimental Setup

In this section, we present the experimental framework we have designed for evaluating existing blocking algorithms. We describe the datasets and the measures we employed to study the behavior of the blocking algorithms under different characteristics of entity descriptions in the LOD cloud. We have used a cluster of 15 Ubuntu 12.04.3 LTS servers (1 master, 14 slaves), each with 8 CPUs, 8GB RAM and 60GB of disk, provided by ~okeanos [12]. Each node could run simultaneously 4 map or reduce tasks, each with a heap size of 1250MB, leaving resources required for I/O and communication with the master. We used Apache Hadoop 1.2.0 and Java version 1.7.0_25 from OpenJDK.

4.1 Datasets

Our study relies on real data from the Billion Triples Challenge 2012 dataset[6] (BTC12), DBpedia, Kasabi[7], the Linked Archives Hub project[8], and OAEI benchmarks[9]. To capture the differences in the heterogeneity and semantic relationships of descriptions, we distinguish between data originating from sources in the *center* and the *periphery* of the LOD cloud. In general, central sources, such as DBpedia and Freebase, are derived from a common source, Wikipedia, from which they extract information regarding an entity. Such descriptions often refer to the original wiki page and feature synonym attributes whose values share a significant number of common tokens. Since they have been exhaustively studied in the literature, descriptions across central LOD sources are heavily interlinked using in their majority *owl:sameAs* links [3], expressing equivalence relations. In our experiments, we used the DBpedia (*BTC12DBpedia*) and Freebase (*BTC12Freebase*) datasets from BTC12, and the raw infoboxes from DBpedia 3.5 (*Infoboxes*), i.e., two different versions of DBpedia. From the OAEI benchmarks, we used the one including the *DBLP* and *Rexa* (OAEI 2009) - describing authors and publications - dataset, that has been widely used in the literature (e.g., in [13]). We also included a movies dataset, used in [6], extracted from DBpedia movies and *IMDB*, to validate the correctness of our algorithms.

On the other hand, data sources in the periphery of the LOD cloud are far more diverse to each other and sparsely interlinked. In our experiments, we considered the *BTC12Rest*, the *BBCmusic* and the *LOCAH* datasets. *BTC12Rest* originates from the BTC12 dataset, which consists of multiple data sources, like DBLP, geonames and

6. km.aifb.kit.edu/projects/btc-2012/
7. archive.org/details/kasabi
8. data.archiveshub.ac.uk/
9. oaei.ontologymatching.org/

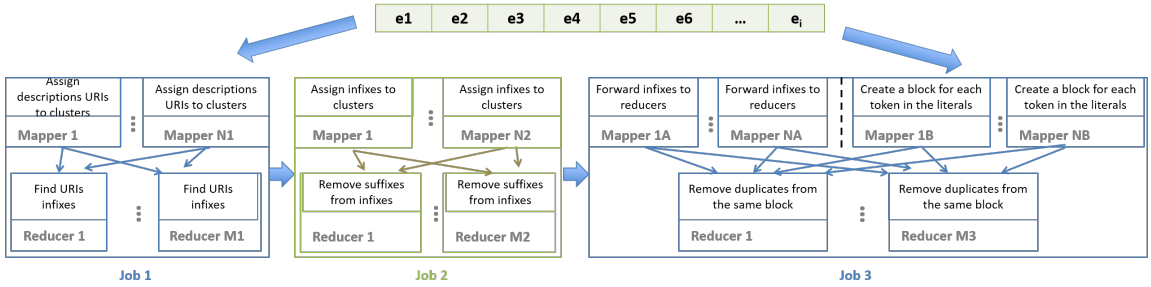

Fig. 8. Prefix-infix(-suffix) blocking in MapReduce.

drugbank. *BBCmusic* originates from Kasabi and contains descriptions regarding music bands and artists, extracted from MusicBrainz and Wikipedia. For *LOCAH*, we used the latest published version at Archives hub (March 2014). This, rather small dataset links descriptions of people, from UK archival institutions, with their descriptions in DBpedia.

Table 2 provides statistics about these datasets, for the number of contained triples, descriptions, attributes, and the average number of attribute-value pairs per description. We have also included the number of entity types, taken as the distinct values of the property *rdf:type*, when provided. Observe that *BTC12DBpedia* contains more types than attributes. This is due to the fact that DBpedia entities may have multiple types from taxonomic ontologies like Yago. *IMDB* is the dataset with the highest number of attribute-value pairs per description. Finally, we have included in each dataset the number of duplicate descriptions based on our ground truth, i.e., descriptions that have been reported to be equivalent (via *owl:sameAs* links) across all datasets of our testbed. Taking into account the transitivity of equality, those descriptions should be regarded as matches, too.

To investigate the ability of blocking algorithms in recognizing relatedness links beyond the *owl:sameAs* among descriptions, we considered the Kasabi *airports* and *airlines* datasets, containing data linked to DBpedia, the dataset with the highest number of references, with the *umbel:isLike* property. This property is used to associate entities that may or may not be equivalent, but are believed to be so. The *twitter* dataset contains data for the presentations of an ESWC conference. It is linked to DBpedia with the *dct:subject* property, which captures relatedness of entities to topics and it is also used in the *books* and *iati* datasets. *Books* describes books listed in the English language section of Dutch printed book auction catalogues of collections of scholars and religious ministers from the 17^{th} century. *Iati* contains data from the International Aid Transparency Initiative. *Iati* is also connected to DBpedia with the *dct:coverage* property, which associates an entity to its spatial or temporal topic, its spatial applicability, or the jurisdiction under which it is relevant. Finally, the *www2012* dataset contains data from the WWW2012 conference, linked to DBpedia with the *foaf:based_near* property, which associates an entity to an abstract notion of location. Table 3 details the type and the number of links of these datasets to DBpedia.

In this setting, we combine *BTC12DBpedia* with each of the datasets of Table 2 to produce the entity collections presented in Table 4, on which we finally ran our experiments. To combine two datasets, for the dirty ER setting, we simply concatenate them into a singe file, while for clean-clean ER, we seek candidate matches between those datasets.

TABLE 3
Characteristics of datasets with different types of links to *BTC12DBpedia*.

	RDF triples	entity descrip-tions	avg. att.-value pairs per description	link	links
airports	238,973	12,294	19.44	*umbel:isLike*	12,269
airlines	15,465	1,141	13.55	*umbel:isLike*	1,217
twitter	6,743	2,932	2.30	*dct:subject*	20,671
books	2,993	748	4.00	*dct:subject*	1,605
iati	378,130	31,868	11.87	*dct:subject*	23,763
				dct:coverage	7,833
www2012	11,772	1,547	7.61	*foaf:based_near*	1,562

- ***D*1** combines *BTC12DBpedia* with *Infoboxes*. Since it contains two versions of the same dataset, it is considered as a homogeneous collection. This is the biggest collection in terms of triples, as well as attributes.
- ***D*2** combines *BTC12DBpedia* with *BTC12Rest*. Since it is constructed by many different datasets, it is the most heterogeneous collection. Note that *BTC12Rest* has the highest number of attributes per entity type.
- ***D*3** combines *BTC12DBpedia* with *BTC12Freebase*. It is the biggest collection in terms of entity descriptions, matches, entity types and comparisons.
- ***D*4** combines *BTC12DBpedia* with *BBCmusic*. Note that *BBCmusic* extracts some of its data from MusicBrainz, which, in turn, extracts data from Wikipedia. Also, *BBCmusic* is edited and maintained by users and BBC staff.
- ***D*5** combines *BTC12DBpedia* with *LOCAH*, the smallest dataset, both in terms of triples and entity descriptions.
- ***D*6** combines DBpedia movies and *IMDB*, as originally used in [6]. It is the most homogeneous collection, it only contains descriptions of movies (i.e., a single entity type) using the smallest number of attributes among all collections. However, the significantly greater (even by six orders of magnitude, compared to the other collections) ratio of matches to non-matches is not typical of the collections we can find in the Web of data.
- ***D*7** combines *DBLP* and *Rexa*. Both datasets use the same ontology; *Rexa*'s attributes are a subset of those used by *DBLP*. Also, it is the collection with the lowest number of attribute-value pairs per description. Note that this collection is a typical benchmark used to evaluate instance matching algorithms.

Following the distinction of our datasets between central and peripheral, we also distinguish our collections between central ($D1$, $D3$, $D6$, and $D7$), composed of central datasets, and peripheral ($D2$, $D4$, and $D5$), part of which are peripheral datasets. For all the collections, we consider both their *clean-clean* and *dirty* versions. In practice, for our datasets, the *clean-clean* and *dirty* versions of a collection are the same; their distinction serves only as means for measuring how

TABLE 2
Datasets characteristics.

	RDF triples	entity descriptions	avg. attribute-value pairs per description	attributes	entity types	attributes/ entity types	duplicates
BTC12DBpedia	102,306,242	8,945,920	11.44	36,354	258,202	0.14	0
Infoboxes	27,011,880	1,638,149	16.49	31,857	5,535	5.76	0
BTC12Rest	849,656	31,668	26.83	518	33	15.7	863
BTC12Freebase	25,050,970	1,849,180	13.55	8,323	8,232	1.01	12,058
BBCmusic	268,759	25,359	10.60	29	4	7.25	372
LOCAH	12,932	1,233	10.49	14	4	3.5	250
DBpedia$_{mov}$	180,680	27,615	6.54	5	1	5	0
IMDB	816,012	23,182	35.20	7	1	7	0
DBLP	12,074,269	1,642,945	7.35	30	10	3	0
Rexa	64,787	14,771	4.39	12	3	4	0

TABLE 4
Entity collections characteristics.

	D1	D2	D3	D4	D5	D6	D7
RDF triples	129,318,122	103,155,898	127,357,212	102,575,001	102,319,174	996,692	12,139,056
entity descriptions	10,584,069	8,977,588	10,795,100	8,971,279	8,947,153	50,797	1,657,716
avg. attribute-value pairs per description	12.22	11.49	11.80	11.43	11.44	19.62	7.32
attributes	68,211	36,872	44,677	36,383	36,368	12	42
entity types	263,737	258,232	266,434	258,206	258,205	1	10
matches	1,564,311	30,864	1,688,606	23,572	1,087	22,405	1,532
matches (incl. duplicates)	1,564,311	31,727	1,700,664	23,944	1,337	22,405	1,532
matches/non-matches	$1.07 \cdot 10^{-7}$	$1.09 \cdot 10^{-7}$	$1.02 \cdot 10^{-7}$	$1.04 \cdot 10^{-7}$	$9.85 \cdot 10^{-8}$	$3.5 \cdot 10^{-5}$	$6.3 \cdot 10^{-8}$
matches/non-matches (dirty)	$2.79 \cdot 10^{-8}$	$7.87 \cdot 10^{-10}$	$2.92 \cdot 10^{-8}$	$5.95 \cdot 10^{-10}$	$3.34 \cdot 10^{-11}$	$1.74 \cdot 10^{-5}$	$1.1 \cdot 10^{-9}$
comparisons (w/o blocking)							
clean-clean	$1.47 \cdot 10^{13}$	$2.83 \cdot 10^{11}$	$1.65 \cdot 10^{13}$	$2.27 \cdot 10^{11}$	$1.1 \cdot 10^{10}$	$6.4 \cdot 10^{8}$	$2.4 \cdot 10^{10}$
dirty	$5.6 \cdot 10^{13}$	$4.03 \cdot 10^{13}$	$5.83 \cdot 10^{13}$	$4.02 \cdot 10^{13}$	$4 \cdot 10^{13}$	$1.29 \cdot 10^{9}$	$1.37 \cdot 10^{12}$

well a blocking method can identify links across different datasets and within the same dataset. We finally combine *BTC12DBpedia* with each peripheral dataset of Table 3 to produce entity collections for studying the ability of blocking algorithms to discover different relatedness attributes.

GroundTruth. Our ground truths were built using a methodology met in the literature (e.g., [5], [6]). For $D2$-$D5$, we consider the *owl:sameAs* links to/from DBpedia 3.7 (the version used in BTC12). For $D1$, we consider the subject URIs of *Infoboxes* that also appear as subjects in *BTC12DBpedia*. The ground truth of $D6$, provided in [6], is made of DBpedia movies connected with IMDB movies through the *imdbId* property. The ground truth of $D7$ is provided by OAEI, since it is a benchmark collection, containing equivalence links between authors, as well as publications. Based on the ground truth and the generated blocks, we say that a known matching pair of descriptions is correctly resolved, i.e., a true positive (TP), if there is at least a block, to which both these descriptions belong. Pairs belonging to the same block are candidate matches. A false positive (FP) is a distinct candidate match not contained in the ground truth. In the opposite, if a known pair of matching descriptions is not a candidate match, this pair is considered a false negative (FN). All remaining pairs of descriptions are considered to be true negatives (TN).

Similarly to $D2$-$D5$, we used the available types of links of the datasets of Table 3 to *BTC12DBpedia*, instead of *owl:sameAs*, to produce the ground truth of the corresponding entity collections. From all datasets, except $D6$ and $D7$, we removed the triples present in the ground truth, since identifying those links is the goal of our tasks.

Our pre-processing, implemented in MapReduce, parses RDF triples in order to transform them into entity descriptions, which are the input of the methods used in our study. It simply groups the triples by subject, and outputs each group as an entity description, using the subject as the entity identifier, removing triples containing a blank node. Moreover, we kept only the entity descriptions for which we know their linked description in *BTC12DBpedia* and removed the rest. This way, we know that any suggested comparison between a pair of descriptions outside the ground-truth is false.

TABLE 5
Quality Measures.

Name	Formula	Description
Recall	$\frac{TP}{TP+FN}$	Measure what fraction of the known matches are candidate matches.
Precision	$\frac{TP}{TP+FP}$	Measure what fraction of the candidate matches are known matches.
F-measure	$2\frac{Precision \cdot Recall}{Precision+Recall}$	The harmonic mean of precision and recall.
RR	$1 - \frac{\text{comparisons with blocking}}{\text{comparisons without blocking}}$	Returns the ratio of reduced comparisons when blocking is applied.
$H3R$	$2\frac{RR \cdot Recall}{RR+Recall}$	The harmonic mean of recall and reduction ratio.

4.2 Measures

The employed quality measures along with a short description are summarized in Table 5. The range of all measures is $[0, 1]$, with 1 being the ideal value. The recall of a blocking method is the upper recall threshold of a non-iterative ER algorithm, which takes its generated blocks as input. Therefore, (1-recall) represents the *cost of blocking*. RR is the percentage of comparisons that we save if we apply the given blocking method. Consequently, it reflects the *benefit of blocking*, since the reason for using blocking in the first place, is the reduction in the required comparisons.

In general, a good blocking method should have a low impact on recall, i.e., a low cost, and a great impact on the number of required comparisons, i.e., a high benefit.

Typically, this trade-off is captured by the F-measure, the harmonic mean of recall and precision. However, as we will see in the next section, the values of F-measure are dominated by the values of precision, which are many orders of magnitude lower than those of recall, so F-measure cannot be easily used to express this trade-off. Moreover, precision is not as important as recall is for blocking, since precision can only be improved by a non-iterative ER method that follows blocking, whereas the recall of blocking is the upper threshold of such ER methods. Thus, we define $H3R$ as the harmonic mean of recall and reduction ratio, a measure which has also been used in [14]. Similar to the F-measure, $H3R$ gives high values only when both recall and reduction ratio have high values. Unlike F-measure, $H3R$ manages to capture the trade-off between effectiveness and efficiency in a more balanced way. Note that $H3R$ does not estimate the performance of a blocking approach (as, for example, [5] does), but evaluates it based on the actual results.

5 Experimental Evaluation

In this section, we analyze the quality and performance of the evaluated blocking methods, taking into consideration the specific features of each dataset. Then, we present the results of blocking, when different kinds of links, other than *owl:sameAs*, are used as ground truth and conclude with a discussion of the lessons learned from this analysis. In our evaluation, we use the adaptation of token blocking ToB, attribute clustering AtC and prefix-infix(-suffix) blocking PIS in MapReduce. Both ToB and PIS can be used with either *clean-clean* Cl or *dirty* Di entity collections, while AtC is suitable for Cl collections. Moreover, the process of AtC requires a similarity function; we use Jaccard similarity over the set of trigrams from the values (similar to [6]).

5.1 Quality Results

5.1.1 Identified Matches (TPs)

Token blocking: The premise of this algorithm is that matching descriptions should at least share a common token, disregarding the comparisons between descriptions that do not share common tokens. Therefore, the higher the number of common tokens, i.e., tokens shared by the datasets composing an entity collection, a description has, the higher the chances it will be placed in a block with a matching description, increasing recall. Figure 9 (left) presents the distributions of common tokens per description, showing that descriptions in central collections feature many more common tokens than those in peripheral ones[10]. For example, 41.43% and 44% of descriptions in $D1$ and $D3$, respectively, have 2-4 common tokens, while for $D2$, $D4$ and $D5$ the corresponding values are 33.26%, 26.03% and 12.97%. We observe a big difference in the distributions of $D6$ and $D7$, which contain many more common tokens per description, to those of the other collections. Only 23.75% of the descriptions in $D6$ and 44% of the descriptions in $D7$ have 0 - 10 common tokens. Figure 9 (left) also shows that a big number of descriptions in peripheral collections, do not share any common tokens. Those are hints that the recall of token blocking in central collections is higher than in peripheral collections.

Indeed, $D6$ is the dataset with the highest recall (99.92%) and the highest number of common tokens per entity (19), while $D5$ is the dataset with the lowest recall (72.13%) and number of common tokens per entity (0). There is a big difference in the number of common tokens in $D6$, compared to $D1$ and $D3$, which is not reflected by their small difference in recall. Due to the high ratio of matches to non-matches in $D6$ (Table 4), descriptions in this collection have many common tokens and this leads to high recall.

Attribute clustering blocking: The goal of attribute clustering is to improve the precision of token blocking, while retaining its recall as much as possible (it cannot have higher recall). To do this, it restricts the number of attributes on which descriptions, featuring a common token, should be compared. Comparisons between descriptions that do not share a common token in a common attribute cluster, are discarded. Hence, descriptions with many common tokens in common clusters are more likely to be matched. Figure 9 (right) presents the distributions of the number of common tokens in common attribute clusters per entity. It shows a clearer distinction between central and peripheral collections than Figure 9 (left); the descriptions in central collections have many more common tokens in common clusters, while many descriptions in peripheral collections do not have any common token in a common cluster. This occurs, because values in the descriptions of peripheral collections are much less similar than those of central collections, leading to a bad clustering of the attributes and, thus, to lower recall. In fact, $D6$ is the dataset with the highest recall (99.55%) and the highest number of common tokens in common attribute clusters per entity (19). On the other hand, $D2$ and $D5$, which have the lowest recall values (68.42% and 71.11%) also have the lowest number of common token in common attribute clusters per entity (0).

In central collections ($D1$, $D3$, $D6$, $D7$), many, small clusters of similar attributes are formed, as the values of the descriptions are similar. This leads to a minor (or zero, in $D7$) decrease in recall, compared to token blocking, while it significantly improves its precision (even by an order of magnitude in $D3$). $D1$ forms many (16,886), small attribute clusters (of 2 attributes in the median case), since in most cases there is a 1-1 mapping between the attributes of the datasets that compose it. These clusters contain the same attribute used by the two versions of DBpedia.

However, this approach has a substantial impact on recall in peripheral collections ($D2$, $D4$, $D5$), even if it still improves precision in all collections (even by an order of magnitude for $D4$). The descriptions in those collections have few common tokens, in the first place, which leads to a bad clustering of attributes; few clusters of many attributes, not similar to each other, are formed. Hence, if we make the blocking criterion of token blocking stricter, by also considering attributes, then the more distinct attributes used per entity type, the more difficult it is for an entity description, to be placed in a common block with a matching description. For *BTC12Rest* (part of $D2$), the ratio between attributes and entity types (last row of Table 2) is the highest (15.7), leading to a great impact on recall (-24.04%). This dataset has the biggest number of data sources that compose it and many

10. We take the median values and not the averages, as the latter are highly influenced by extreme values and our distributions are skewed.

TABLE 6
Statistics and evaluation of blocking methods.

	D1	D2	D3	D4	D5	D6	D7
Token blocking statistics:							
blocks	1,639,962	122,340	1,019,501	57,085	2,109	40,304	18,553
comparisons (clean-clean)	$1.68 \cdot 10^{12}$	$3.74 \cdot 10^{10}$	$6.56 \cdot 10^{11}$	$2.39 \cdot 10^{10}$	$8.72 \cdot 10^{8}$	$2.91 \cdot 10^{8}$	$1.45 \cdot 10^{9}$
RR (clean)	88.51%	86.81%	96.03%	89.48%	92.09%	54.50%	94.04%
comparisons (dirty)	$5.56 \cdot 10^{12}$	$3.68 \cdot 10^{12}$	$4.27 \cdot 10^{12}$	$4.02 \cdot 10^{12}$	$1.01 \cdot 10^{12}$	$2.05 \cdot 10^{9}$	$2.35 \cdot 10^{11}$
RR (dirty)	90.08%	90.87%	92.67%	90.01%	97.48%	−58.85%	82.93%
common tokens per entity (median)	4	3	4	2	0	19	12
Attribute clustering blocking statistics:							
blocks	5,602,644	150,293	1,673,855	39,587	3,724	43,716	19,148
comparisons	$3.22 \cdot 10^{11}$	$4.20 \cdot 10^{9}$	$1.84 \cdot 10^{11}$	$1.43 \cdot 10^{9}$	$7.13 \cdot 10^{8}$	$2.13 \cdot 10^{8}$	$8.38 \cdot 10^{8}$
RR	97.80%	98.52%	98.89%	99.37%	93.54%	66.80%	96.55%
common tokens in common att. clusters per entity (median)	4	0	4	2	0	19	11
attribute clusters	16,886	124	2,106	6	8	4	8
attributes per attribute cluster (median)	2	142	9	4,261	3,946	3	3.5
Prefix-Infix(-Suffix) blocking statistics:							
blocks	3,266,798	141,517	789,723	45,403	2,098	N/A	18,442
comparisons (clean-clean)	$1.10 \cdot 10^{12}$	$1.78 \cdot 10^{10}$	$2.75 \cdot 10^{11}$	$2.30 \cdot 10^{9}$	$4.08 \cdot 10^{8}$	N/A	$1.28 \cdot 10^{9}$
RR (clean)	92.48%	93.72%	98.34%	98.99%	96.30%	N/A	94.72%
comparisons (dirty)	$4.39 \cdot 10^{12}$	$3.45 \cdot 10^{12}$	$5.34 \cdot 10^{12}$	$3.32 \cdot 10^{12}$	$1.76 \cdot 10^{12}$	N/A	$2.23 \cdot 10^{11}$
RR (dirty)	92.16%	91.44%	90.84%	91.76%	95.59%	N/A	83.78%
Recall:							
Token blocking (clean-clean)	98.38%	92.46%	95.52%	87.76%	72.13%	99.92%	99.54%
Token blocking (dirty)	98.38%	89.99%	94.85%	87.95%	77.34%	99.92%	99.54%
Attribute clustering blocking	97.31%	68.42%	92.10%	76.84%	71.11%	99.55%	99.54%
Prefix-Infix(-Suffix) blocking (clean-clean)	100%	91.71%	87.68%	95.44%	68.17%	N/A	99.54%
Prefix-Infix(-Suffix) blocking (dirty)	100%	89.25%	87.06%	95.50%	74.12%	N/A	99.54%
Precision:							
Token blocking (clean-clean)	$1.56 \cdot 10^{-6}$	$1.00 \cdot 10^{-6}$	$2.49 \cdot 10^{-6}$	$1.30 \cdot 10^{-6}$	$1.13 \cdot 10^{-6}$	$1.21 \cdot 10^{-4}$	$1.18 \cdot 10^{-6}$
Token blocking (dirty)	$3.64 \cdot 10^{-7}$	$5.14 \cdot 10^{-9}$	$3.78 \cdot 10^{-7}$	$1.05 \cdot 10^{-8}$	$1.29 \cdot 10^{-9}$	$7.51 \cdot 10^{-5}$	$6.5 \cdot 10^{-9}$
Attribute clustering blocking	$8.51 \cdot 10^{-6}$	$5.76 \cdot 10^{-6}$	$1.01 \cdot 10^{-5}$	$1.41 \cdot 10^{-5}$	$1.35 \cdot 10^{-6}$	$1.52 \cdot 10^{-4}$	$1.97 \cdot 10^{-6}$
Prefix-Infix(-Suffix) blocking (clean-clean)	$1.87 \cdot 10^{-6}$	$2.19 \cdot 10^{-6}$	$5.72 \cdot 10^{-6}$	$1.01 \cdot 10^{-5}$	$2.05 \cdot 10^{-6}$	N/A	$1.19 \cdot 10^{-6}$
Prefix-Infix(-Suffix) blocking (dirty)	$6.04 \cdot 10^{-7}$	$8.21 \cdot 10^{-9}$	$2.77 \cdot 10^{-7}$	$1.23 \cdot 10^{-8}$	$6.99 \cdot 10^{-10}$	N/A	$6.84 \cdot 10^{-9}$
F-measure:							
Token blocking (clean-clean)	$3.13 \cdot 10^{-6}$	$2.00 \cdot 10^{-6}$	$9.72 \cdot 10^{-7}$	$2.06 \cdot 10^{-8}$	$1.94 \cdot 10^{-9}$	$2.42 \cdot 10^{-4}$	$2.35 \cdot 10^{-6}$
Token blocking (dirty)	$7.28 \cdot 10^{-7}$	$1.03 \cdot 10^{-8}$	$7.55 \cdot 10^{-7}$	$2.10 \cdot 10^{-8}$	$2.59 \cdot 10^{-9}$	$1.50 \cdot 10^{-4}$	$1.30 \cdot 10^{-8}$
Attribute clustering blocking	$1.70 \cdot 10^{-5}$	$1.15 \cdot 10^{-5}$	$2.02 \cdot 10^{-5}$	$2.82 \cdot 10^{-5}$	$2.69 \cdot 10^{-6}$	$3.04 \cdot 10^{-4}$	$3.94 \cdot 10^{-6}$
Prefix-Infix(-Suffix) blocking (clean-clean)	$3.75 \cdot 10^{-6}$	$4.38 \cdot 10^{-6}$	$9.98 \cdot 10^{-7}$	$2.02 \cdot 10^{-5}$	$4.11 \cdot 10^{-6}$	N/A	$2.38 \cdot 10^{-6}$
Prefix-Infix(-Suffix) blocking (dirty)	$1.21 \cdot 10^{-6}$	$1.64 \cdot 10^{-8}$	$5.55 \cdot 10^{-7}$	$2.46 \cdot 10^{-8}$	$1.40 \cdot 10^{-9}$	N/A	$1.37 \cdot 10^{-8}$
$H3R$:							
Token blocking (clean-clean)	93.18%	89.55%	95.77%	88.61%	80.90%	70.53%	97.04%
Token blocking (dirty)	94.05%	90.43%	93.75%	88.97%	86.25%	N/A ($RR < 0$)	90.48%
Attribute clustering blocking	97.55%	80.76%	95.37%	86.66%	80.80%	79.95%	98.16%
Prefix-Infix(-Suffix) blocking (clean-clean)	96.09%	92.70%	92.70%	97.18%	79.83%	N/A	97.07%
Prefix-Infix(-Suffix) blocking (dirty)	95.92%	90.33%	88.91%	93.59%	83.50%	N/A	90.98%

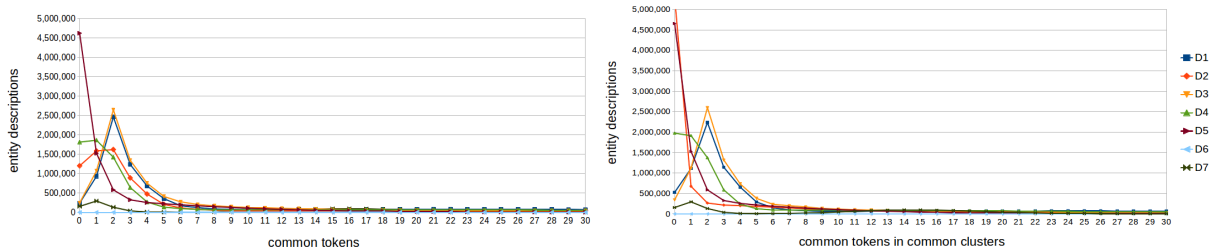

Fig. 9. Common tokens (left) and common tokens in common clusters (right) per entity description distributions for $D1$-$D7$.

different attribute names can be used for the same purpose; hence, big attribute clusters are formed. *LOCAH* (part of $D5$) only has 3.5 attributes per entity type. Thus, the recall of attribute clustering blocking is insignificantly reduced (-1.02%), compared to that of token blocking.

Prefix-Infix(-Suffix) blocking: Prefix-Infix(-Suffix) blocking is built on the premise that many URIs contain useful information. Its goal is to extend token blocking and improve both its recall, by also considering the subject URIs of the descriptions, and its precision, by disregarding some unneeded tokens in the URI values (either in the prefix or suffix). It achieves good recall values in datasets with similar naming policies in the URIs, as in $D4$, part of which is *BBCmusic*, which also has Wikipedia as a source. However, it misses many matching pairs of descriptions, when the names of the URIs do not contain useful information, as in $D3$ that uses random strings as ids, or have different policies, as in $D5$, which uses concatenations of tokens, without

delimiters, as URIs. The recall of $D1$ is 100%, because the collection is constructed this way; it consists of two versions of the same dataset, DBpedia, and the URIs appearing as subjects in *Infoboxes* are only those URIs that also appear as subjects in *BTC12DBpedia*. PIS is not applicable (marked N/A) to $D6$, since URIs have been replaced with numerical ids in the provided datasets. In $D7$, recall is the same as in the other blocking methods, since the matches can be found by tokens in the literal values of the descriptions.

5.1.2 Missed Matches (FNs)

A non-negligible number of matching pairs of descriptions do not share any common tokens at all. Such descriptions, constituting the false negatives of token blocking, should not be assumed faulty, or noisy. We distinguish two different sources of information that can be exploited for successfully placing descriptions of missed matches in common blocks:

(i) The matches of their neighbors: Given that a description can have, as one of its values, another description, neighborhoods of related descriptions are formed, spinning the Web of data. The knowledge of matches in the neighbors of a description is valuable for correctly matching this description. For example, if a description e_{10} is related to e_1, e_{20} is related to e_2, and we know that e_{10} and e_{20} match, then we can use this knowledge as a hint that e_1 and e_2 could possibly match, too.

(ii) A third, matching description: In dirty collections (typically peripheral), which are composed of datasets that potentially contain duplicate descriptions, a description e_1 could have more than one matching description, e.g., both e_2 and e_3. Identifying one of these matches, e.g., (e_1, e_3), knowing that (e_2, e_3) is a match, leads to also identify the missing match (e_1, e_2).

Table 7 provides details about the number and the characteristics of false negative pairs of descriptions, and the set of individual descriptions that constitute these pairs[11].

We focus first on the neighbors of these descriptions, namely descriptions that appear in their values. We found that almost all the descriptions in the false negatives have at least one neighbor (second row of Table 7). Looking more thoroughly, we counted the percentage of descriptions in false negatives that have at least one neighbor belonging to the ground truth (third row of Table 7). In all cases, this percentage is more than 10% and goes up to 58% for $D4$. This means that, not only do these descriptions have neighbors, but many of these neighbors can be matched to other descriptions in the same collection as well. Then, we counted the percentage of descriptions in false negatives that have neighbors, which have already been matched to another description (fourth row of Table 7). This percentage is over 20% in most collections, while it reaches up to 51.84% for $D4$. Finally, we counted the percentage of false negative pairs, whose descriptions have neighbors, which match to each other (fifth row of Table 7). This percentage is 0 for $D1$, as matches in this collection are defined as descriptions that have the same subject URI. However, in some peripheral collections ($D2$, $D4$), examining the matches of the neighbors of the descriptions is meaningful.

11. $D6$ is excluded, as it does not contain any descriptions with neighbors and $D7$ is excluded, as it only yields 7 missed matches.

TABLE 7
Characteristics of the missed matches of token blocking.

	D1	D2	D3	D4	D5
FNs	25,419	3,176	87,672	2,886	303
descriptions in FNs, with neighbor(s)	99.64%	100%	99.99%	100%	100%
descriptions in FNs, with neighbor(s) in ground truth	22.60%	53.94%	36.43%	58.36%	11.57%
descriptions in FNs, with neighbor(s) with an identified match	20.94%	48.54%	34.05%	51.84%	7.59%
FNs with matching neighbors	0%	24.81%	0.38%	37.63%	0%
FNs with common, identified matches	0%	25.35%	10.54%	0.14%	8.58%

Another useful piece of information for the missed matches of dirty collections is whether their descriptions have been correctly matched to a third description. The last row of Table 7 quantifies this statistic, showing that there are collections, both peripheral ($D2$, $D5$) and central ($D3$), for which this kind of information could, indeed, be useful.

The information of Table 7 is lost when we only consider the tokens in the values of the descriptions to create the blocks in a single round, but it could be useful to an iterative method. Iterative blocking [11], based on some initial blocks, aims to identify matches of type (ii), as well as eliminate redundant comparisons. In our experiments, the recall of iterative blocking, given the blocks of token blocking from the dirty collection with the smallest number of comparisons ($D6$), was the same as that of token blocking (99.92%), since both of its datasets contain no duplicates (Tables 2, 4), but the number of comparisons performed was almost half of those suggested by token blocking. We also applied iterative blocking to the dirty collection with the lowest recall ($D5$), giving the blocks generated by token blocking as input. The process did not terminate within a reasonable amount of time, even so, the recall of iterative blocking was 78.09% after a first pass, whereas the recall of token blocking was 77.34%.

Regarding attribute clustering blocking, it misses the matches that are also missed by token blocking, plus matches that, even if they share common tokens, those tokens appear in the values of attributes in different clusters. The matches missed by prefix-infix(-suffix) blocking are those with no common tokens in their literal values and no common infixes in their URIs.

5.1.3 Non-matches (FPs and TNs)

Next, we examine the ability of blocking methods to identify non-matches, namely their ability to avoid placing non-matching descriptions in the same block. A key statistic for this, regarding the datasets, is the ratio of matches to non-matches (Table 4). The higher the ratio, the easier it is for a blocking method to have better precision, as it statistically has better chances of suggesting a correct comparison. $D6$ is the collection with the highest such ratio and precision, while $D5$ has the lowest ratio and, in most blocking methods, the lowest precision, too. It is clear from Table 6 that attribute clustering is the most precise method, since, in almost every case, it results in the fewest wrong suggestions. On the contrary, the least precise method is token blocking, in all cases. The differences in precision, in some cases even by an order of magnitude, also determine F-measure, since

TABLE 8
Analysis of 1K sampled matches and 1K sampled non-matches.

	D1	D2	D3	D4	D5	D7
matches with neighbors	967	956	913	918	859	973
non-matches with neighbors	966	955	912	917	854	973
neighbors of matches (median)	17	80	100	138	121	1
neighbors of non-matches (median)	72	80	105	171	121	1
matches with matching neighbors	862	254	7	766	570	966
non-matches with matching neighbors	32	22	0	0	542	590

the differences in recall are not that big. All the evaluated methods have very low precision, i.e., the vast majority of suggested comparisons correspond to non-matches.

5.1.4 Structural Analysis of Matches and Non-matches

To better understand the characteristics of matches versus those of non-matches in the evaluated collections, we have analyzed sample pairs of matching and non-matching descriptions. In particular, we have taken 1,000 random pairs of matches and non-matches from each collection and we have focused on their neighbor pairs of descriptions. The results of this analysis are presented in Table 8.

First, we counted the number of pairs of descriptions that both have neighbors. We found that those numbers, presented in the first two rows of Table 8 for matches and non-matches, respectively, are almost the same. Practically, almost all the pairs of descriptions are linked to other pairs of description, in all collections. Then, we measured the median number of neighbors (pairs of descriptions) that a match has (Table 8, row 3) and the same median number for non-matches (Table 8, row 4). Again, there are no significant differences between those two lines. Those numbers vary greatly from collection to collection, ranging from 1 (for $D7$) to 171 (for $D4$). Finally, we counted the number of pairs, whose neighbor pairs match. For matches (Table 8, row 5), this number is always higher than the corresponding number for non-matches (Table 8, row 6). Intuitively, this means that when a match is found, the chances that there is another match in its neighbor pairs are increased.

5.2 Performance Results

Table 6 shows that all the evaluated methods manage to greatly reduce the number of comparisons that would be required if blocking was not employed, e.g., by one ($D1$-$D4$, $D7$) or two ($D5$) orders of magnitude for token blocking. This is reflected by high RR in all cases. An exception is $D6$, which is much smaller in terms of descriptions and, consequently, comparisons without blocking. Moreover, its descriptions contain many more common tokens than the other collections, leading to more comparisons per entity. Therefore, token blocking does not save many of the comparisons that would be required without blocking and in $D6$ dirty, it even produces twice as many comparisons.

With respect to $H3R$, we notice that, in general, central collections have higher scores, i.e., they present a better balance between recall and reduction ratio. This means that in these collections, comparisons that are discarded by blocking mostly correspond to non-matches, while many of the comparisons discarded by blocking in peripheral collections correspond to matches. Again, $D6$ has a different behaviour, since it initially contains a much smaller number of comparisons and a high ratio of matches to non-matches, so the reduction ratio for this collection is limited. These measures are not applicable to token blocking, when applied to $D6$ dirty, since in that case the reduction ratio is negative.

5.3 Different Types of Links

In order to evaluate the ability of blocking methods to identify more types of links, semantically close or even not that close to equivalence links, we have run a set of experiments with the peripheral collections consisting of each of the datasets of Table 3 and *BTC12DBpedia*. We have chosen a wide range of link types, in order to show that the same blocking method can better identify some specific link types, and fail to identify other link types. Table 9 provides the recall of token blocking, when applied to each of those collections. Similarly to the *owl:sameAs* links, token blocking performs well for links with the semantics of equivalence (i.e., *umbel:isLike*, expressing a possible equivalence), as in the *airports* and *airlines* datasets with recall values close to 100%. It also manages to identify many subject associations (i.e., *dct:subject*, expressing the topic of a description), as in the cases of *books* and *iati* datasets. It performs poorly in identifying this kind of association, however, in the *twitter* dataset, where its recall values fall to below 10%. This could be justified by the nature of this dataset, which, in most cases, simply states who created a slideset. Regarding spatial associations (i.e., *dct:coverage*, expressing the spatial topic of a description, and *foaf:based_near*, relating two spatial objects), token blocking manages to identify a mere 39% of the *coverage* associations of the *iati* dataset, but it performs much better in identifying the *based_near* associations of *www2012*, with a recall of 63%. The spatial relationships of *coverage* are looser than those of *based_near*, hence the related entities are not so strongly related in the former type of links. For example, in *iati*, the description of a project regarding the evaluation of cereal crop residues is linked to the DBpedia resource describing Latin America and the Caribbean, through the *coverage* relation, while, in *www2012*, a Greek professor is linked to the DBpedia resource describing Greece, through the *based_near* relation.

5.4 Lessons Learned

We now present the key points of our evaluation. *Central* collections are mostly derived from Wikipedia, from which they extract information regarding an entity. This way, descriptions in such collections follow similar naming policies and feature many common tokens (Figure 9) in the values of semantically similar, or equivalent attributes (see the small size of clusters in Table 6). Those are exactly the premises on which the evaluated blocking methods are built.

For these reasons, the recall achieved by token blocking in central entity collections is very high (ranges from 99.92% to 94.85%). With the exception of $D6$ (featuring a higher ratio of matching to non-matching descriptions), the precision achieved by token blocking in these collections ranges from $2.49 \cdot 10^{-6}$ to $3.64 \cdot 10^{-7}$. The gains in precision

TABLE 9
Recall of the collections composed of datasets of Table 3 and *BTC12DBpedia*.

	airports	**airlines**	**twitter**	**books**	**iati**		**www2012**
link	*umbel:isLike*	*umbel:isLike*	*dct:subject*	*dct:subject*	*dct:subject*	*dct:coverage*	*foaf:based_near*
Recall of token blocking	97.47%	99.75%	9.52%	63.55%	49.13%	39.46%	62.61%

brought by attribute clustering blocking in central entity collections are up to one order of magnitude (for $D3$), with a minor cost on recall (from 0% to 3.42%). Prefix-infix(-suffix) blocking can improve both recall and precision of token blocking for central collections, as in $D1$, but, it can also deteriorate these values, as in the dirty case of $D3$, which uses random identifiers as URIs, in which recall drops by 7.79% and precision by 26.72%. In a nutshell, many redundant comparisons are suggested by blocking algorithms in all entity collections (see precision and F-measure in Table 6), due to the small ratio of matches to non-matches in the collections (Table 4). However, as $H3R$ reveals, the comparisons that are discarded by blocking in central collections mostly correspond to non-matches.

On the contrary, descriptions in *peripheral* KBs are more diverse, following different naming policies and sharing few common tokens (Figure 9), since they stem from various sources. The lack of similar values in those descriptions leads to a bad clustering of attributes; big clusters of attributes not similar to each other are formed (Table 6).

For these reasons, the recall of token blocking for peripheral collections drops even to 72.13%, while precision ranges from $1.3 \cdot 10^{-6}$ to $1.29 \cdot 10^{-9}$. The gains in precision brought by attribute clustering blocking (up to one order of magnitude) in peripheral collections, come at the cost of a drop in recall up to 24.04% (corresponding to 7,421 more missed matches). Prefix-infix(-suffix) blocking can improve the precision of token blocking in peripheral collections, even by an order of magnitude (for $D4$), or decrease it by an order of magnitude (for $D5$), while it decreases recall from 0.74% to 3.96%, i.e., more matches are missed. In the case of $D4$, in which both datasets use Wikipedia as a source, recall is improved by up to 7.68%. In overall, however, $H3R$ reveals that many of the comparisons that are discarded by blocking in peripheral collections correspond to matches.

Nevertheless, information for the missed matches, e.g., from the neighborhoods of their descriptions (Table 7), sets the ground for a new generation of ER algorithms, which will exploit this information to identify more matches, in an iterative fashion. In Table 8, we have shown that even a single match in the neighborhood of a candidate pair is a good match-indication for that pair, too.

Finally, in peripheral collections, there are several types of relations, other than equivalence, between descriptions. Token blocking identifies some of them, depending on the dataset, the specific type of such links, and the immediacy of those relations (Table 9). It does not perform well when the data do not contain much information (e.g., see the characteristics of *twitter* in Table 3), or when the relationship of the entities is loose (e.g., see the recall of *iati* for *dct:coverage* in Table 9). Thus, for a quantitative evaluation of blocking methods ground truth should not be restricted only to *owl:sameAs* links. We could potentially take other relations into account, to identify more such links, or more *owl:sameAs* links, using iterative algorithms.

6 Related Work

In this section, we briefly overview representative ER techniques that have been proposed in the Database and Semantic Web communities and can be classified under two general axes: (a) *high* and *low similarity* in the *content* and (b) *high* and *low similarity* in the *structure* of entity descriptions.

More precisely, *deduplication* techniques [15] are essentially ER techniques for highly (low) similar in structure (content) descriptions from one relation, *record linkage* for structured [15] or semi-structured Web data [16] targets highly (low) similar in content (structure) descriptions from two relations, while in the Web of data, descriptions hosted in a network of KBs exhibit low similarity both in content and structure (i.e., are *somehow* similar). It is worth noticing that *instance matching* techniques [17] (also called data linking or link discovery), aiming to semantically relate entities described in two KBs, are variations of the record linkage problem for semi-structured descriptions (e.g., in RDF) and relationships going beyond entity equivalence (e.g., sameAs). Web-scale ER techniques essentially require to address these problems for a large number of entity types and hosting KBs, having limited domain knowledge regarding how description schemas and instances could match, as well as representative ground truth and training sets. In this context, resolution decisions regarding one pair of descriptions essentially provide evidence for the matching of others (i.e., the network effect).

In the rest of this section, we focus on blocking (and meta-blocking) methods outside the scope of our work and explain the reasons why they have not been included in our experimental evaluation. We finally compare our work with other benchmarking and large-scale evaluation efforts.

Blocking. Besides the methods we detailed in Section 2, alternative methods for reducing the number of unnecessary comparisons include: Locality-Sensitive Hashing, similarity joins, frequent itemsets, and clustering.

The key idea of blocking with Locality-Sensitive Hashing (LSH) (e.g., [18]) is to hash descriptions multiple times, using a family of hash functions, in such a way that similar descriptions (e.g., with Jaccard similarity, approximated by minhasing [19]) are more likely (with probabilistic guarantees) to be placed into the same bucket than dissimilar ones. Any two descriptions that hash at least once into the same bucket, for any of the employed hash functions, are considered to be a candidate pair. This technique assumes an a-priori knowledge of a minimum similarity threshold between entity description pairs, above which, such pairs are considered candidate matches. However, as we will see in our experimental evaluation (see Section 5), often, matching descriptions do not share many common tokens and thus, have very low, even zero, similarity when computed only on the values of their attributes. Those matches would not be placed in the same bucket and thus, they would not be considered candidate matches. Effectively choosing a minimum similarity threshold also depends on the KBs.

For example, when seeking matches between two central KBs, a high similarity threshold can be used, since such KBs usually have more similar values. Using a lower threshold in central KBs would result in many false candidate pairs. Accordingly, using a high similarity threshold in peripheral KBs, in which descriptions have lower similarity values, would yield many missed matches. Consequently, applying LSH across domains is an open research problem, due to the difficulty in knowing or tuning a similarity threshold that can be generalized to identify matches across several domains in an effective and efficient way.

String-similarity join algorithms (e.g., [20], [21], [22]) construct blocks which are guaranteed to contain all pairs of descriptions whose string values similarities are above a certain threshold and potentially some pairs whose string values similarities are below that threshold. To achieve that, without computing the similarity of all pairs of descriptions, this family of algorithms build an inverted index from the tokens of the attribute values of the descriptions. However, unlike token blocking, this inverted index is created only by the first non-frequent tokens of each description (i.e., the most discriminating), based on the *prefix filtering* principle [20]. [21] additionally applies a *size filtering* [23] on the sets of tokens to disregard some of the candidate pairs, based on the fact that $Jaccard(x, y) \geq t \Rightarrow t \cdot |x| \leq |y|$. The ppjoin+ algorithm [22] introduces a *positional filtering*, i.e., the position in the ordered set of tokens, in which a token appears, to further reduce the number of candidate pairs. Tuning the appropriate similarity threshold is non-trivial and it also affects the performance of the string-similarity join algorithms [24]. Smaller thresholds entail less pruning, and thus, more time. Furthermore, [25] proves experimentally that algorithms based on prefix filtering are only effective when the similarity threshold is extremely high. However, this is not the case in the Web of data, where highly heterogeneous descriptions, yielding very low similarity in their literal values, can refer to the same entity.

[26] introduces a method for building blocks based on Maximal Frequent Itemsets (MFI). Abstractly, each MFI (an itemset can be a set of tokens) of a specific attribute in the schema of a description defines a block, and descriptions containing the tokens of an MFI for this attribute are placed in a common block. Using frequent itemsets to construct blocks may significantly reduce the number of candidates for matching pairs. However, since many matching descriptions share few, or even no common tokens, further requiring that those tokens are parts of frequent itemsets is too restrictive for those pairs of matching descriptions, resulting in many missed matches in the Web of data. Moreover, MFI blocking requires a-priori knowledge of the desired block sizes, and is also based on the notion of a schema, information which is unavailable at the Web of data.

Canopy clustering [27] is an unsupervised clustering method that has been used as blocking. Initially, one random description is chosen as a canopy center and it is compared to all other descriptions. All the descriptions within a loose distance threshold to this canopy center are added to this canopy, while those within a tighter distance threshold, are removed from the candidate canopy centers. This process repeats, until there are no more candidate canopy centers. The problem with this approach, even with the parallel version of Mahout[12] or our own MapReduce implementation, is that it fails to scale to the volumes of our datasets.

Block post-processing. Further processing steps on the results of blocking have been proposed in the literature for further reducing the number of comparisons to be performed by an ER task (e.g., [28], [29], [30]). Such steps make sense to be used, when blocking results in missing only few matches, and the whole process is faster than exhaustively performing the comparisons between all descriptions. For example, [28] proposes to reconstruct the blocks of a given blocking collection in order to discard redundant, as well as unnecessary comparisons. [28] essentially transforms a given blocking collection into a blocking graph, whose nodes correspond to entity descriptions, while its undirected edges connect the co-occurring descriptions. Every edge is associated with a weight representing the likelihood that the adjacent entities are matching candidates. Low-weighted edges are pruned, so as to discard comparisons between unlikely to match descriptions. To minimize the missed matches, an iterative entity resolution process can exploit in a *pay-as-you-go* fashion any intermediate results of blocking and matching, discovering new candidate matches. Such iterative process may consider matching evidence provided by entity descriptions placed into the same block (e.g., [11]) or being structurally related in the original entity graph (e.g., [13], [31]). In this work, we are focusing on blocking algorithms, since those further steps also benefit from a better initial blocking, but we briefly show how such an algorithm [11] uses the results of blocking.

Benchmarks and evaluation frameworks. Existing works in ER benchmarks [32], [33] and evaluation frameworks [34], [35] focus on the similarity of descriptions and how these similarities affect the matching decision of entity resolution; not on blocking, explicitly. In all cases, data collections are built from central datasets of a single domain, e.g., only bibliographic. Those data variations are not adequate to evaluate the blocking algorithms suitable for cross-domain ER involving a large number of entity types. Finally, many works on ontology and instance matching, e.g., [13], [36], have been using the OAEI benchmarks in their evaluations. Typically, those collections are composed of two ontologies with a 1-1 mapping in their attributes, or even a single ontology, whose instances, i.e., entity descriptions, have some modifications in their values. We have included and analyzed one of those benchmarks in this study.

7 Summary

In this work, we evaluated, for the first time, ER blocking algorithms for somehow (not only highly) similar descriptions in the Web of data. We have investigated the data characteristics of such descriptions that impact blocking algorithms' effectiveness and efficiency. Highly similar descriptions, met in central LOD collections, feature many common tokens in the values of common attributes, while somehow similar descriptions, met in peripheral collections, have significantly fewer common tokens in attributes that are not necessarily semantically related. Hence, the former can be compared only on their content (i.e., values), while

12. mahout.apache.org/users/clustering/canopy-clustering

the latter require contextual information, e.g., the similarity of neighborhood descriptions, linked with different types of relationships. Since a single similarity function cannot identify such matches in a single pass, multiple iterations of matching (focusing on context) and/or blocking (focusing on content) are needed. Towards this end, we are interested in progressive ER algorithms that try to maximize the benefit (e.g., number of resolved entities, number of links between resolved entities) of each iteration, by dynamically adapting their execution plan, based on previous results.

Acknowledgements: This work was partially supported by the EU FP7 SemData (#612551) project.

REFERENCES

[1] V. Christophides, V. Efthymiou, and K. Stefanidis, *Entity Resolution in the Web of Data*, ser. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers, 2015.
[2] X. L. Dong and D. Srivastava, *Big Data Integration*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2015.
[3] M. Schmachtenberg, C. Bizer, and H. Paulheim, "Adoption of the linked data best practices in different topical domains," in *ISWC*, 2014.
[4] P. Christen, *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*, ser. Data-centric systems and applications. Springer, 2012.
[5] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl, "Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data," in *WSDM*, 2012.
[6] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl, "A blocking framework for entity resolution in highly heterogeneous information spaces," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 12, pp. 2665–2682, 2013.
[7] V. Efthymiou, K. Stefanidis, and V. Christophides, "Big data entity resolution: From highly to somehow similar entity descriptions in the web," in *IEEE Big Data*, 2015.
[8] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom, "Swoosh: a generic approach to entity resolution," *VLDB J.*, vol. 18, no. 1, pp. 255–276, 2009.
[9] I. Bhattacharya and L. Getoor, "Collective entity resolution in relational data," *TKDD*, vol. 1, no. 1, 2007.
[10] H. Kim and D. Lee, "HARRA: fast iterative hashed record linkage for large-scale data collections," in *EDBT*, 2010.
[11] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina, "Entity resolution with iterative blocking," in *SIGMOD*, 2009.
[12] V. Koukis, C. Venetsanopoulos, and N. Koziris, "˜okeanos: Building a cloud, cluster by cluster," *IEEE Internet Computing*, vol. 17, no. 3, pp. 67–71, 2013.
[13] S. Lacoste-Julien, K. Palla, A. Davies, G. Kasneci, T. Graepel, and Z. Ghahramani, "Sigma: simple greedy matching for aligning large knowledge bases," in *KDD*, 2013.
[14] M. Kejriwal and D. P. Miranker, "An unsupervised algorithm for learning blocking schemes," in *ICDM*, 2013.
[15] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 9, pp. 1537–1555, 2012.
[16] O. Hassanzadeh, "Record linkage for web data," Ph.D. dissertation, 2013.
[17] M. Nentwig, M. Hartung, A.-C. N. Ngomo, and E. Rahm, "A survey of current link discovery frameworks," *Semantic Web Journal*, 2015.
[18] P. Malhotra, P. Agarwal, and G. Shroff, "Graph-parallel entity resolution using LSH & IMM," in *EDBT/ICDTWorkshops*, 2014.
[19] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *J. Comput. Syst. Sci.*, vol. 60, no. 3, pp. 630–659, 2000.
[20] S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," in *ICDE*, 2006.
[21] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *WWW*, 2007.
[22] C. Xiao, W. Wang, X. Lin, and J. X. Yu, "Efficient similarity joins for near duplicate detection," in *WWW*, 2008.
[23] A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins," in *VLDB*, 2006.
[24] Y. Jiang, G. Li, J. Feng, and W. Li, "String similarity joins: An experimental evaluation," *PVLDB*, vol. 7, no. 8, pp. 625–636, 2014.
[25] A. Metwally and C. Faloutsos, "V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors," *PVLDB*, vol. 5, no. 8, pp. 704–715, 2012.
[26] B. Kenig and A. Gal, "MFIBlocks: an effective blocking algorithm for entity resolution," *Inf. Syst.*, vol. 38, no. 6, pp. 908–926, 2013.
[27] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *KDD*, 2000.
[28] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl, "Meta-blocking: Taking entity resolutionto the next level," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1946–1960, 2014.
[29] V. Efthymiou, G. Papadakis, G. Papastefanatos, K. Stefanidis, and T. Palpanas, "Parallel meta-blocking: Realizing scalable entity resolution over large, heterogeneous data," in *IEEE Big Data*, 2015.
[30] S. E. Whang, D. Marmaros, and H. Garcia-Molina, "Pay-as-you-go entity resolution," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 5, pp. 1111–1124, 2013.
[31] V. Rastogi, N. N. Dalvi, and M. N. Garofalakis, "Large-scale collective entity matching," *PVLDB*, vol. 4, no. 4, pp. 208–218, 2011.
[32] E. Ioannou, N. Rassadko, and Y. Velegrakis, "On generating benchmark data for entity matching," *J. Data Semantics*, vol. 2, no. 1, pp. 37–56, 2013.
[33] A. Ferrara, S. Montanelli, J. Noessner, and H. Stuckenschmidt, "Benchmarking matching applications on the semantic web," in *ESWC*, 2011.
[34] H. Köpcke, A. Thor, and E. Rahm, "Evaluation of entity resolution approaches on real-world match problems," *PVLDB*, vol. 3, no. 1, pp. 484–493, 2010.
[35] O. Hassanzadeh, F. Chiang, R. J. Miller, and H. C. Lee, "Framework for evaluating clustering algorithms in duplicate detection," *PVLDB*, vol. 2, no. 1, pp. 1282–1293, 2009.
[36] F. M. Suchanek, S. Abiteboul, and P. Senellart, "PARIS: probabilistic alignment of relations, instances, and schema," *PVLDB*, vol. 5, no. 3, pp. 157–168, 2011.

Vasilis Efthymiou is a PhD candidate at the University of Crete, Greece, and a research assistant at ICS-FORTH, Greece. The topic of his PhD is entity resolution in the Web of data. He got his MSc and BSc degrees from the same university in 2012 and 2010, respectively. He has received undergraduate and postgraduate scholarships from FORTH, working in the areas of Semantic Web and Ambient Intelligence.

Kostas Stefanidis is an Associate Professor at the University of Tampere, Finland. He got his PhD in personalized data management from the University of Ioannina, Greece, in 2009. His research interests lie in the intersection of databases, Web and information retrieval. Kostas co-authored more than 35 papers in peer-reviewed conferences and journals, including ACM SIGMOD, IEEE ICDE and ACM TODS.

Vassilis Christophides is a Professor at the University of Crete, Greece. He has been recently appointed to an advanced research position at INRIA Paris - Rocquencourt. Previously, he worked as Distinguished Scientist at Technicolor, R&I Center in Paris. He studied Electrical Engineering at the National Technical University of Athens, Greece, 1988, he received his DEA in computer science from the University PARIS VI, 1992, and his Ph.D. from the Conservatoire National des Arts et Metiers of Paris, 1996. His main research interests include Databases and Web Information Systems, as well as Big Data Processing and Analysis. He has published over 130 articles in high-quality international conferences, journals and workshops. He received the 2004 SIGMOD Test of Time Award and the ISWC Best Paper Award in 2003 and 2007.