The Missing Links: Discovering Hidden Same-as Links among a Billion of Triples

George Papadakis
L3S Research Center
Hannover, Germany
papadakis@L3S.de

Gianluca Demartini
L3S Research Center
Hannover, Germany
demartini@L3S.de

Philipp Kärger
L3S Research Center
Hannover, Germany
kaerger@L3S.de

Peter Fankhauser
Fraunhofer IPSI
Darmstadt, Germany
fankhauser@ipsi.fraunhofer.de

ABSTRACT

The Semantic Web is constantly gaining momentum, as more and more Web sites and content providers adopt its principles. At the core of these principles lies the Linked Data movement, which demands that data on the Web shall be annotated and linked among different sources, instead of being isolated in data silos. In order to materialize this vision of a web of semantics, existing resource identifiers should be reused and shared between different Web sites. This is not always the case with the current state of the Semantic Web, since multiple identifiers are, more often than not, redundantly introduced for the same resources.

In this paper we introduce a novel approach to automatically detect redundant identifiers solely by matching the URIs of information resources. The approach, based on a common pattern among Semantic Web URIs, provides a simple and practical method for duplicate detection. We apply this method on a large snapshot of the current Semantic Web comprising 1.15 billion statements and estimate the number of hidden duplicates in it. The outcomes of our experiments confirm the effectiveness as well as the efficiency of our method, and suggest that URI matching can be used as a scalable filter for discovering implicit same-as links.

Categories and Subject Descriptors

H.3.3 [**DATABASE MANAGEMENT**]: Database Applications—*Data mining*

General Terms

Algorithms, Experimentation

Keywords

URI Matching, Web Data integration, Large scale information integration, Entity Resolution

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
iiWAS2010, 8-10 November, 2010, Paris, France.
Copyright 2010 ACM ACM 978-1-4503-0421-4/10/11 ...$10.00.

1. INTRODUCTION

The concept of Semantic Web is becoming more and more popular in many areas, as even content providers and news agencies use its techniques to annotate their web sites and document collections. Most importantly, though, major search engines, like Google and the Yahoo! SearchMonkey [16], rely on semantic data in order to improve user experience during search. Taking also into account the constantly growing number of web sites that employ technologies such as Microformats and RDFa, one can easily deduce that the Web of Semantics is becoming a reality nowadays.

Among Semantic Web's principles, Linked Data is one of the most fundamental, as it conveys exposing, sharing and connecting any kind of structured data with dereferenceable URIs on the Web [4]. However, the structured data that is currently available in a machine-understandable format is not as "linked" as this principle envisions. Content providers invariably use their own object identifiers and individual users tend to publish RDF data that contain duplicates and redundant identifiers instead of existing URIs. Consequently, islands of knowledge are created instead of connecting resources with co-references. It is no surprise, therefore, that among the billions of triples that have been exposed on the Semantic Web, there are only several millions of links between their resources [9].

In this paper, we target the problem of finding hidden duplicates on the Semantic Web by considering solely the characteristics of resource URIs[1]. Contrary to the general intuition that "there is no semantics in a URI", we follow the idea that terms contained in Semantic Web URIs usually carry some sense that can be used for identifying duplicate URIs (i.e., pairs of separate URIs actually referring to the same entity). Indeed, the performance of our method suggests that URI-based matching is not only feasible but also effective. The reason is that the people as well as the automated systems that generate semantic data tend to use meaningful terms in their URIs. These terms are certainly suitable for matching, allowing for an effective as well as efficient method that is also applicable to datasets of very large scale.

[1]For a demo see: `http://urimatch.L3S.uni-hannover.de`.

Our approach complements recent work (e.g., [9] and [18]) that serves the same goal, relying on the content of Semantic Web resources, i.e., the literals and other resources associated with them as objects in RDF statements. Contrary to these methods, our approach does not consider the semantic data that describe information resources, thus constituting a simple, yet practical and efficient technique that is able to process large volume of data with minimal space and time requirements. To this end, we sacrifice its applicability to some extend, since it cannot handle non-meaningful, random, numerical URIs. In addition, it does not cover blank nodes and spelling mistakes in URIs. However, the analysis of the large-scale dataset we employ along with the performance of our method suggest that such URIs comprise a relatively small part in the current state of the Semantic Web.

In more detail, we experimentally evaluate our approach by applying it on a large snapshot of the Semantic Web that comprises 1.15 billion triples: the Billion Triple Challenge dataset, published in 2009 (BTC09 for short) [2]. We analyse the structure of URIs contained in it, and observe that it justifies our technique for matching URIs, while the outcomes of our experiments indicate that URIs alone serve well for duplicate detection in the Semantic Web. More specifically, our method exhibits high efficiency in handling so large a dataset as well as high effectiveness, achieving a recall of around 70% and a precision well above 90%. Last but not least, our experiments enable us to provide an estimation on the number of missing links among all resources of this dataset (11.64 millions approximately).

In summary, our main contributions in this paper are the following: first, we identify a structure common among the URIs of the current Semantic Web and explain how it can be used for matching resources. Then, we introduce an algorithm for transforming a collection of URIs into this suitable-for-matching form, together with a method for detecting duplicate URIs that incorporates it. Their effectiveness and efficiency is analysed on several, voluminous subsets of BTC09 that serve as ground-truth. Finally, we apply our method on the BTC09 dataset and estimate with the help of probabilistic analysis the number of hidden links included it.

The rest of the paper is structured as follows: in the next section, we discuss related work and explain how our method differs from existing ones. In Section 3 we present our approach to matching entities based on their URI, while in Section 4 we introduce several interesting aspects of the BTC09 dataset, along with the experimental evaluation of our approach on explicitly paired URIs. Section 5 contains a discussion on estimating the hidden duplicates of the complete BTC09 dataset, and Section 6 concludes the paper.

2. RELATED WORK

Although looking into Web identifiers (i.e., URIs) for eliciting clues about the corresponding entities is a novel approach in the context of Semantic Web, it is not new in the wider area of the Web. In the literature, successful methods for guessing the language [3] and the topic [2] of a Web page solely by considering its URL have been presented. These works are related to ours, since they all rely on the same hypothesis: more often than not, Web identifiers carry semantics.

[2]See `http://challenge.semanticweb.org/`.

Also relevant to our work is a task common in Web Information Retrieval, namely *Url Normalization* [13]. This task aims at identifying syntactically different URLs that are actually equivalent, (i.e., they correspond to the same resource). It constitutes a crucial procedure for the performance of crawling, indexing and retrieving resources given a query. To address it, numerous techniques have been proposed in the literature, with the most recent ones focusing on automatically learning general rules that encapsulate URL patterns [1, 12, 15]. These rules are employed for transforming URLs into a canonical form that facilitates the de-duplication of web pages. Contrary to our approach, these methods apply exclusively on URLs, aiming at identifying session ids, cookies and dynamic URL features appended to a canonical URL. However, these features do not serve for uniquely identifying resources on the Semantic Web, thus being uncommon among URIs. Moreover, these techniques are crafted for *intra-domain de-duplication*: they try to detect different URLs from the same domain that pertain to identical resources. On the other hand, our approach deals with *inter-domain de-duplication*, identifying equivalent resources that come from different sources. Hence, existing URL normalization techniques are inapplicable to the settings and the problem we are considering.

Duplicate detection—also known as record linkage or entity resolution—is a long standing problem in computer science. Initially, it was mainly employed for de-duplicating census data [11], but ever since then it has been applied to a wide variety of domains. There exists a large body of work on matching resources using literal content in the form of structured records [14] or of bag of tokens [6]. The common theme for these approaches is to weight the matching evidence provided by content based on its uniqueness and similarity. For a more comprehensive overview on general duplicate detection see [7]. None of these methods, though, employs exclusively the URIs of the resources in order to disambiguate them. The size of the evidence they consider makes them computationally expensive and thus inapplicable to large scale datasets with hundreds of millions of resources, like BTC09.

Linking data and finding hidden co-references on the Semantic Web is handled in various recent works, an overview of which is presented in [17]. Our motivation is prominently shared with [9], on the difference, though, that [9] interlinks Semantic Web data sources using semantic descriptions of linkage relations between data sources. A similar idea is followed by Silk [18], a framework that interprets declarative descriptions for linking data repositories and for keeping links updated when data changes. Contrariwise, our approach presents an additional, less declarative but more efficient way of matching entities on a large scale. It is worth exploring, nevertheless, how we could integrate our measures and indeces as an additional module to Silk.

Also related to our work is `sameas.org`, a project that crawls, stores, and manages same-as statements. It offers a collection of same-as clusters in a way that different URIs pointing to the same resource can easily be retrieved. The main difference with our work, though, is that `sameas.org`[3] principally focuses on crawling, storing, manipulating and reusing *explicit* same-as pairs [8]. An alternative approach is pursued in OKKAM [5], which aims at building an Entity

[3]See `http://www.sameas.org/publications.php` for an overview.

resource1			resource2		
prefix	infix	suffix	prefix	infix	suffix
http://dbpedia.org/resource/	High_Cumberland_ Jubilee		http://mpii.de/yago/resource/	High_Cumberland_ Jubilee	
http://dbpedia.org/resource/	Meteor_shower		http://umbel.org/umbel/sc/	MeteorShower	
http://dblp.l3s.de/d2r/resource /pub- lications/books/sp/ wooldridgeV99/	ThalmannN99		http://bibsonomy.org/uri/ bibtexkey/- books/sp/ wooldridgeV99/	ThalmannN99	dblp
http://liris.cnrs.fr/	olivier.aubert	/foaf.rdf#me	http://bat710.univ-lyon1.fr/	~oaubert	/foaf.rdf#me

Table 1: Examples of URI Pairs split into the PI(S) scheme

Name Service (ENS) that provides entities with a globally unique identifier.

3. MATCHING SEMANTIC WEB RESOURCES BY URIS

In the following, we provide a formal definition of the problem we are tackling along with some observations on the structure of Semantic Web URIs. Based on them, we go on to introduce two methods for matching URIs; the first one is a straightforward approach that will serve as our baseline, whereas the second is especially crafted for taking these observations into account. Both approaches have been implemented and are available in an on-line demo[4]. A thorough evaluation and comparison between them is presented in Section 4.

3.1 Problem Definition

The problem we are tackling can be formally defined as follows:

Given a set of URIs, U, provide a binary response (yes/no) to any possible query q of the form $u_i \equiv u_j$ where $u_i, u_j \in U$.

In other words, our method should decide whether u_i and u_j match (i.e., whether they correspond to the same entity) or not by considering the evidence not only in the query itself but also in the rest of the dataset. The reason for taking into account the latter is that the URIs of the query alone do not suffice to deduce safe conclusions. Consider, for example, the query comprised of the URIs in the first row of Table 1; although for a human it is evident that the infix is identical, and thus they refer to the same resource, an automatic method needs additional evidence to deduce safely that the two strings `http://dbpedia.org/resource` and `http://mpii.de/yago/resource` can be discarded as prefixes.

3.2 The structure of Semantic Web URIs

Unique or almost unique identifiers which are shared across resources are crucial for duplicate detection based on URIs. Such identifiers typically arise from well established human labeling schemes, like names, titles, or addresses, as well as from explicit standardization efforts, such as industry wide product codes or digital object identifiers (DOI). Even though the W3C explicitly discourages users from putting any semantics into Semantic Web URIs [10], they constitute a natural source of evidence appropriate for matching Semantic Web resources. The reason is that the URIs in the Semantic Web typically follow a *Prefix-Infix(-Suffix) scheme*; the *Prefix* contains information about the source/-domain of the URI, the *Infix* is a sort of local identifier, while the *(optional) Suffix* contains either details about the format, like `.rdf` and `.n3`, or a named anchor, like `#me`.

Table 1 gives a few typical examples of URIs split into prefixes, infixes, and suffixes. These are actually pairs of URIs that have been randomly selected from the set of `owl:sameAs` statements: the first two pairs demonstrate that prefixes are invariably constant within a domain (e.g., the prefix `http://dbpedia.org/resource/` in the case of the left column); the third pair shows that suffixes (`dblp` in our case) are optional, and can be ignored when matching URIs, while the fourth pair exemplifies that the identifiers may not be identical (`olivier.aubert` vs. `~oaubert`), but could be matched using an appropriate similarity measure.[5]

In order to investigate the extent to which this structure applies to Semantic Web URIs, we examined a random sample of $2,895$ URIs, drawn from the union set of subject and object identifiers of the BTC09 dataset. Indeed, we were able to identify a prefix, an infix and a (potentially empty) suffix in all of them. This manually created dataset was also used for testing the performance of our method for automatically splitting and matching URIs, which is presented in section 3.4.

All in all, Semantic Web URIs seem to follow a Prefix-Infix-(Suffix) form, denoted as *PI(S) form* from now on, with the Infix containing the most discriminative information of each URI. The PI(S) form allows for an intuitive matching methodology based on infixes, which is introduced in Section 3.4. First, though, we present our baseline approach that considers URIs as bags of words (thus disregarding the PI(S) form).

3.3 URIs as bags of tokens

A straightforward approach to matching URIs is to tokenize them on all special characters (i.e., characters that are neither letters nor digits) and compare the resulting bags of tokens. Their similarity is, then, computed by measuring the cosine distances of the respective TF-IDF weighted vectors. Given the fact that prefixes remain constant within each domain, their tokens should be rather frequent, and, thus, should be weighted low by the inverse document frequency (IDF). The same holds for suffixes as well, because they are usually shared by many URIs (usually stemming from the same domain). As a result, prefixes and suffixes only have a marginal influence on the cosine distance between URIs, and matching them is principally based on the tokens of the infixes.

Nevertheless, this approach has two drawbacks, which we try to overcome with the method of the following section. First, prefixes should not be taken into account at all for

[4]See `http://urimatch.L3S.uni-hannover.de`.

[5]In this paper, we do not match based on similarity between identifiers but we rather use exact match of infixes.

matching; even though their tokens have a limited influence by IDF, they adversely affect the matching process whenever they are part of the infix of other resources. Second, matching on many tokens using standard Information Retrieval technology is rather inefficient, as it requires combining the (usually large) posting lists of the individual tokens.[6]

3.4 URIs in the PI(S) form

A more promising approach is to split URIs into domain (encoded in the prefix), identifier (infix), and a possible suffix; a standard index on the infix parts (i.e., on the identifiers) can, then, be used for efficiently detecting duplicate URIs using *exact string matching*. It should be stressed at this point that a simple tokenization cannot necessarily isolate the prefix from the infix, because the latter may consist of more than the last token. We need, therefore, a method for automatically transforming URIs into the PI(S) form.

The observation that a domain typically reuses the same prefix(es) for its URIs[7], suggests the following approach for automatically splitting URIs: Based on the *sorted* list of URIs one can determine repeated prefixes in a single scan, and thereby isolate the identifiers that apparently stemm from the same domain. These URIs invariably share the same prefix (at least partially), so a more elabore procedure for identifying it can be carried out solely in the context of their group, idependently of the others. An outline of this process is described in pseudocode in Algorithm 1.

Algorithm 1 Maximum common prefix

```
 1: sort URIs
 2: tokenize URIs at all special characters
 3: cluster URIs according to the first $n$ tokens
 4: for all clusters do
 5:    for all URIs in the cluster do
 6:       for all possible prefixes do
 7:          find the set of (distinct) next tokens $T$
 8:       end for
 9:    end for
10:    for all URIs in the cluster do
11:       set as a prefix the one with the largest $|T|$
12:       set as infix the substring following the prefix
13:    end for
14: end for
```

In more detail, our method clusters URIs according to their source (i.e. equal prefix) by considering their first two tokens ($n = 2$). This is because the first one is invariably 'http', the second is usually quite indicative of the domain, whereas the third token can already be part of the identifier. Then, the algorithm goes on to investigate all possible prefixes within each cluster independently, as there is no overlap of prefixes between separate domains. Starting with the prefix formed by the concatenation of the first $n+1$ tokens, the next possible prefix is constructed by adding the next token of the URI at hand at the end of the previous prefix. Each of these potential prefixes is associated with the set of the distinct tokens, T, that follow it in all URIs of the cluster containing it. After applying this procedure to the entire cluster, the candidate that is selected as prefix of each URI is the one among with the largest cardinality of the corresponding set of distinct next tokens, .

[6]This behaviour can also be observed in the on-line demo where the baseline approach has a mean execution time of one second.

[7]A domain may use hierarchical prefixes, but also in this case a single prefix is shared among URIs.

To illustrate the functionality of our algorithm we give the following example: all URIs of DBPedia are of the form `http://dbpedia.org/resource/X`, where `X` is the infix. By tokenizing them and removing the first two tokens (`http` and `dbpedia`) which are used for clustering, we have the following sets of distinct next tokens for the remaining tokens: `org` has only one distinct next token, namely `resource`. `resource` has all infixes used in DBPedia URIs as the set of distinct next tokens. The infix itself has no such following token. Given this, the algorithm identifies `resource` as the last token of the prefix because it has the largest set of distinct next tokens.

For identifying optional suffixes, we employ a similar approach. Suffixes are actually those endings that, like prefixes, are shared among many URIs, usually of the same source (e.g., `.rdf` and `.n3`). Therefore, they should not be considered as part of a local identifier, since they contain no evidence about the entity. The maximum common suffix of a set of URIs can be determined by applying Algorithm 1 with $n = 1$ to the *reversed* infixes within each domain. It is worth mentioning at this point that some domains make extensive use of hash URIs (e.g. `http://example.org/people#johndoe`), thus using "meaningful" anchors (`#johndoe`) that are not suffixes in our sense. Such cases are supported by our algorithm, because the corresponding suffix is typically not shared among other URIs.

All in all, our algorithm determines the local identifiers for each URI by heuristically discarding the noise introduced by prefixes and suffixes. Its complexity of $O(nlogn)$ ensures scalability, making it applicable to large collections of URIs, like the dataset we are considering. Its efficiency is even more enhanced by applying its memory intensive part (i.e., identifying the most diverse set of distinct next tokens) only to "meaningful" subsets of the dataset. In the following section we compare this approach with that of the bag-of-tokens in terms of effectiveness and efficiency.

4. EXPERIMENTAL EVALUATION

In this section we present an empirical evaluation of the performance of our approach in detecting duplicates through URI matching. In our tests, we employ three different ground-truth sets of explicitly declared identical URIs. Based on our findings, we then apply our method to a large and representative snapshot of the current Semantic Web with the aim of estimating the number of undetected duplicates (a.k.a. hidden links) in it.

In more detail, this section is structured as follows: Section 4.1 introduces the metrics used for our evaluation, whereas Section 4.2 describes the datasets we employed in our studies. In Section 4.3 we apply our approach on a small, representative dataset and compare its performance with the baseline method. Finally, in Section 4.4 we also estimate Precision and Recall on two larger ground-truth sets obtained from the `owl:sameAs` and the IFP (InverseFunctionalProperties) statements contained in BTC09.

4.1 Evaluation Metrics

The problem we defined in Section 3.1 clearly constitutes an Information Retrieval (IR) task. To measure the performance of our method we employ, therefore, the most typical IR metrics, namely Precision and Recall. In our settings, the former expresses the number of identified matches that are true, whereas the latter denotes the portion of existing matches that are detected by our algorithm. Thus, they are defined as follows:

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn},$$

where

- tp represents the number of true positive queries, where u_i and u_j **match** in the ground-truth and the response of our method is **yes**.
- fp is the number of false positive queries, where u_i and u_j **do not match** in the ground-truth and the response of our method is **yes**.
- fn stands for the number of false negative queries, where u_i and u_j **match** in the ground-truth and the response of our method is **no**.

It is worth noting at this point that we can only have a mere pessimistic estimation of the real value of Precision. The reason is that for an accurate estimation of Precision a list of all the non-matching URI pairs is needed. In the absence of it, however, we can only assume that URIs that are not associated with explicit `owl:sameAs` statements are non-matching. In other words, two URIs that have the same infix but are not in the same cluster are considered false positives, although they could actually match if some same-as statements weren't missing. This is, though, quite simplistic, since, as we discuss in Section 5, there are numerous hidden links. To ameliorate this situation to some extent, we compute the transitive closure of the same-as statements and consider as non-matching those URI pairs that are not included in the same equivalence class.

4.2 Experimental dataset

The Billion Triple Challenge 2009 (BTC09) provides a dataset of 1.15 billion RDF statements crawled from the Web. The organizers of the challenge have already provided some statistics based on a small sample of the dataset, but here we present a deeper analysis with respect to those aspects that are particularly interesting for our task. Hence, we first explore to which extent the URIs contained in it correspond to meaningful, shared identifiers (i.e. non blank nodes), enabling the applicability of our method. Subsequently, we extract key subsets that could serve as ground-truth for our evaluation, by providing both explicit and implicit same-as links between equivalent resources.

General statistics.

Overall, the BTC09 collection contains 182 million unique URIs that appear either as subjects or as objects. Out of these URIs, 12.55% appear only as subjects, 30.49% appear only as objects, while the others (56.96%) appear both as subjects and objects. This indicates a strong connection between nodes in the graph and the presence of many relations between entity identifiers. As we discuss in Section 5, though, there are yet several millions of hidden links that are not explicitly stated.

Another important aspect of the dataset is the coverage of blank nodes (a.k.a. bnodes). More specifically, bnodes constitute anonymous nodes in an RDF graph, that are mainly used whenever there is no information available about the corresponding resource. Their identifiers cannot carry, therefore, any semantics, thus turning our method inapplicable to them. Nevertheless, they are not expected to pose any serious limitations on the generality of our method, since they are used less frequently than URIs.

Indeed, only 32.61% of all distinct URIs are bnodes, with the vast majority of them (95%) appearing both as as subjects and objects. Similarly, out of the 1.15 million triples of the dataset, less than a third of them (namely 29.45%) has a blank node as subject. In the case of objects, the impact of blank nodes is even lower, with only 4.99% of the statements having a bnode in that place in comparison to 64.89% that have object URIs and to 30.12% that contain literal values. All in all, though fairly common, bnodes are significantly outweighed by URIs, thus restricting the applicability of our method only to a minor extent.

Same-As statements dataset.

In order to evaluate our approach we need a benchmark dataset containing pairs of matching URIs. The `owl:sameAs` statements present in the BTC09 dataset constitute an apparently suitable gold standard for this purpose, as there are 6.57 million such triples involving 9.22 million distinct URIs. Before using this subset, we had to clean it from possible errors and the noise caused by the crawling process in order to ensure a high quality ground-truth. We proceeded as follows.

We first discarded 783,507 sameAs statements that are actually duplicates crawled from different Web sites. Then, we produced all the sets of clusters of equivalent URIs (i.e., the transitive closure of all `owl:sameAs` statements in equivalence classes) and obtained 4 million distinct clusters. Their sizes clearly exhibit a power law distribution, as the vast majority of them (89.28%) is of size 2 (i.e., exactly two URIs referring to the same object), and the larger the cluster size, the lower its frequency. The extremely large, oversized equivalence classes (the largest one contains 4,344 URIs) are very likely to involve significant amount of noise. For this reason, we chose to disregard the 10,842 clusters that contain more than 10 URIs thus yielding a high quality gold standard. On the whole, our dataset (denoted as *SameAs* from now on) encompasses 3.93 million clusters that generate 5.99 million queries of the form $q : (u_i \equiv u_j)$.

Note that this ground-truth has, to some extent, an implicit bias: it contains URIs that have been explicitly stated to be equivalent, with a considerable part of them stemming from machine generated same-as relationships that could well follow URI patterns. Therefore, to have a better understanding of the performance and the robustness of our algorithm, we need an additional dataset that includes a higher variety of equivalence relationships derived from a diversity of sources. To this end, we created the following ground-truth of implicit duplicates.

Inverse Functional Properties dataset.

InverseFunctionalProperties (IFPs) provide a reliable, alternative means of discovering implicit equivalence relation-

Dataset	URIs	Queries
ManuallySplit	$2,895$	$2,622$
IFP	$17,990$	$11,553$
SameAs	$8,665,201$	$5,988,533$

Table 2: Overview of the Datasets

ships in the Semantic Web[8]: *any two resources that share the same value for an IFP are, actually, identical.* Taking advantage of this principle, we collected all distinct properties that are direct subclasses of IPF, amounting to $1,336$ in total. From these properties, we selected the 35 most frequently used, so as to exclude the erroneous and noisy ones and ensure a high quality in the gold standard. We, then, generated the equivalence classes of these IFPs and obtained $18,714$ distinct clusters of size ≥ 2 corresponding to around 1.3 million same-as statements. To further restrict the impact of noise, we selected again those classes that contain up to 10 URIs, resulting in a dataset (symbolised *IFP* from now on) of $17,990$ distinct URIs that comprises $11,553$ queries.

Manually Split URIs dataset.

In order to verify our theory about the structure principally followed by URIs, we created a small set of URIs that were manually transformed into the PI(S) form. This set was derived through a random sampling of $2,895$ URIs corresponding to $2,622$ queries from the set of distinct URIs in the BTC09. We manually split these URIs identifying for each one of them the most descriptive set of tokens (i.e., the URI's infix). In the following, we use this dataset, denoted *ManuallySplit* from now on, for evaluating the accuracy of Algorithm 1 in transforming URIs into the PI(S) form and for comparing our approach to the baseline (which does not exploit splittings).

It is worth noting that we took special care to make sure that the origin Web sites of the selected URIs is well distributed and not dominated by a single domain. In case of domination, the outcomes of our evaluation would be significantly distorted and could by no means be indicative of the performance of our method. However, due to the pluralism of domains contributing to the BTC09 dataset, our subset contains a representative sample from numerous domains, thus being appropriate for our experiments.

A summary of the aforementioned datasets is presented in Table 2.

4.3 Evaluation on ManuallySplit

Though the smallest dataset, *ManuallySplit* serves multiple purposes: It is used for assessing the splitting accuracy of our algorithm in transforming URIs into the PI(S) form, as well as for comparing its effectiveness with that of the baseline.

Splitting accuracy.

We estimated the accuracy of Algorithm 1 over the $2,895$ URIs of *ManuallySplit* as follows: for each URI, we compared the infix of the automatic split with that of the manual split and considered as *trueSplit* only the cases where they were exactly equal. We, thus, ensured that all three parts were correctly identified, because even the slightest deviation in the infix entails an error in at least two URI parts ($falseSplit$). In these settings, accuracy was defined as:

$$a = \frac{trueSplit}{falseSplit + trueSplit}.$$

For our algorithm it was estimated to 97.13%.

[8]For a detailed definition see `http://www.w3.org/TR/owl-ref/#InverseFunctionalProperty-def`.

Matching URIs in the PI(S) form.

We can now evaluate the matching approach based on the PI(S) form. Most importantly, we can also assess to which extent errors in the splitting step affect the overall effectiveness of our approach. Table 3 presents a comparison of the matching results on the *ManuallySplit* dataset between the manual and the automatic split. It is evident that the impact of split errors is minor on both Precision and Recall. Actually, in the case of Precision the difference merely lies in 39 false positive pairs that were additionally introduced by our splitting algorithm. Manual inspection of these pairs showed that most of them were indeed correct matches, for which, though, there was no explicit same-as statement in the ground-truth (in other words, we already discovered some of the hidden same-as links).

	Precision	Recall	F-Measure
Manual Split	0.98	0.59	0.74
Automatic Split	0.97	0.57	0.72

Table 3: Matching algorithm effectiveness for automatic/manual split on *ManuallySplit*.

Matching URIs as bags of tokens.

In order to evaluate the vector-based approach to URI similarity, we consider the following settings: given a query $q : (u_i \equiv u_j)$, the system will respond yes if u_j appears in the top-k matching candidates of u_i, otherwise it will respond no. Table 4 presents the effectiveness of this approach based on the inverted index of tokens of the URIs in *ManuallySplit*. We can see that Precision is high, because it is unlikely that the system replies "yes" for URIs that do not match; they always have a rather low TF-IDF similarity. Recall, on the other hand, is generally low and increases only when raising the number k of retrieved URIs.

k	Precision	Recall	F-Measure
1	0.99	0.40	0.57
2	0.98	0.50	0.66
3	0.98	0.53	0.69
4	0.98	0.53	0.69
5	0.98	0.54	0.70
10	0.98	0.55	0.70
50	0.97	0.56	0.71
100	0.97	0.56	0.71
500	0.97	0.57	0.72
1000	0.97	0.57	0.72

Table 4: Effectiveness of the baseline for various values of threshold k on *ManuallySplit*.

Performance Comparison.

With respect to *effectiveness*, the performance of our method corresponds approximately to the results of the baseline for $k = 500$. For $k = 1$, the Recall of our approach (0.57) surpasses that of the baseline (0.40) to a great extent without much loss in precision. This means that our method returns 17% more true positives by means of a binary decision, while the vector space approach returns up to 500 possible candidate matches to reach the same recall. Among them, however, only one is the correct, true positive.

As far as *efficiency* is concerned, our approach outperforms the baseline significantly; as pointed out in Section 3.3, each query contains several highly frequent tokens (e.g. 'http' and 'dbpedia') that require merging the corresponding large posting lists. The complexity of the query process is, therefore, quite high; the time that elapsed on average between issuing a query to the inverted index (implemented using the Lucene library[9]) and getting the response amounted to $1s$. For this reason, it was infeasible to apply the baseline method to the other, larger datasets, and we restricted its performance evaluation to this small dataset. On the other hand, the approach using the PI(S) form employs a relational database table that contains the infix of each URI, and requires, on average, $14ms$ for answering a query. In this way, our method can respond to queries in real time, while scaling to very large dataset.

4.4 Evaluation on SameAs and IFP

We applied the method using the PI(S) form on the other two, much larger ground-truths, namely *SameAs* and *IFP*. Table 5 summarizes the effectiveness measures. We notice that Recall is higher for *SameAs*, while Precision lies at the same level for both datasets. The difference in the Recall should be expected, due to the presence of machine generated, straightforward links between resources in the *SameAs* dataset. Contrariwise, the *IFP* dataset contains statements that are mostly manually generated, and thus more difficult to be identified. The fairly competitive recall of 0.66 verifies the robustness of our approach as well as its applicability to URIs that have not been mapped automatically.

	Precision	**Recall**	**F-Measure**
SameAs	0.91	0.73	0.82
IFP	0.92	0.66	0.77

Table 5: Matching algorithm effectiveness on the SameAs and Ifp datasets.

5. MATCHING ON A SEMANTIC WEB SCALE

In this section we present results of an analysis performed on the entire BTC09 dataset. From the Recall value of 0.73 in Table 5 we can observe that 73% of the explicit same-as statements in the BTC09 dataset link URIs which share a common local identifier. On this basis we try to estimate the expected number of hidden same-as links in the full dataset that also share a common local identifier.

To this end, we used the Algorithm 1 introduced in Section 3.4 to transform all URIs of the dataset (122,275,114 in total, excluding blank nodes) into the PI(S) form. We considered the infix as identifier, and determined the frequency of each one of them. This procedure brought about 46,303,085 identifiers that are shared by at least two URIs. Thus 37.87% of the URIs can be considered as candidate duplicates, that may be linked by a hidden same-as statement. Of course this is a gross overestimation, because just sharing an identifier does not necessarily imply a match.

[9]See `http://lucene.apache.org`.

A better estimate of the number of hidden duplicates can be accomplished by inspecting the actual frequency distributions of the identifiers. Given two URIs with a shared identifier i, the probability that they match depends on the uniqueness of the identifier. The more URIs share an identifier, the less likely any random pair of these URIs will match. More formally, this can be estimated via an application of Bayes' Rule:

$$P(M|i \equiv i) = \frac{P(i \equiv i|M)P(M)}{P(i \equiv i)}, \qquad (1)$$

where

- $P(M|i \equiv i)$ is the probability that two URIs with shared identifier i match.
- $P(i \equiv i|M)$ is the probability that two URIs share an identifier provided they match. It can be estimated by $|i|/|N|$, which is simply the relative frequency of identifier i.
- $P(M)$ is the prior matching probability. It can be estimated by $2 \cdot |M|/(|N| \cdot (|N|-1))$, where $|M|$ is the number of matching pairs of URIs in the *SameAs* dataset, and $|N|$ is the overall number of distinct URIs it contains. Note that this may be an overestimation, as it can be expected that URIs in the ground-truth have a higher probability to match with some other URI than URIs in the full dataset.
- $P(i \equiv i)$ is the probability that a randomly chosen pair of URIs shares the identifier i, which can be estimated by $|i| * (|i| - 1)/(|N| * (|N| - 1))$. In other words, it is equal to the ratio of the two relative frequencies, i.e., of the number of pairs having the same identifier and the total number of pairs in the whole dataset.

Combining these estimates in Formula 1 gives:

$$P(M|i \equiv i) = |M|/((|i| - 1) \cdot |N|) \qquad (2)$$

Moreover, the number of candidate pairs given an identifier i with frequency $|i|$ is $|i| \cdot |i - 1|/2$, each with $P(M|i \equiv i)$ probability to match. Based on Formula 2 , we can estimate the expected number of same-as links by summing over all identifiers i_k in the full dataset.

$$\begin{aligned} E(\#sameAs) &= \sum_k |i_k| \cdot (|i_k| - 1)/2 \cdot P(M|i_k \equiv i_k) \\ &= \frac{|M|}{2|N|} \sum_k |i_k| \qquad (3) \end{aligned}$$

We estimate $E(\#sameAs)$ from the exact number of $|M|$ and $|N|$ for the *SameAs* dataset as presented in Table 2 (5,988,553 and 8,665,201 respectively). With the actual frequency distributions in the full dataset, we arrive at an expected number of 16 million hidden same-as statements. If we exclude the actual present same-as statements in the ground-truth that follow the PI(S) scheme (6 million · 0.73) this leaves 11.6 million hidden same-as statements.

This estimate should be taken with some caution. First, it only considers URIs following the PI(S) scheme, and thus can be considered as an underestimation. Second, the prior matching probability $P(M)$ is overestimated, as pointed out above. Nevertheless, it does give an indication for the expected redundancy in the BTC09 dataset, clearly motivating the need for some concerted effort to discover hidden duplicates.

6. CONCLUSIONS AND FUTURE WORK

Missing same-as links on the Semantic Web is an important problem, and solving it would improve the quality and the usefulness of the Web to a great extent. In this paper, we presented an approach for detecting hidden co-references in a collection of resources solely by looking into their URIs. It turns out that the way URIs are currently built is suitable for bridging isolated data sources. To be more accurate, 73% of the explicit and 66% of the implicit same-as statements in a large, representative snapshot of the Semantic Web are actually covered by our observation that the URIs of equivalent resources share the same infix. Generalizing this observation to the whole dataset leads to the conjecture that approximately 11.64 million URIs have duplicates that are not explicitly linked with a same-as relation.

In the future, we plan to extend our approach with more advanced string similarity measures in order to cover the cases of naming variations and spelling mistakes. Moreover, we intend to employ the PI(S) form of URIs in the context of more elaborate techniques for resource matching that are based on profiles consisting of infixes of associated resources (i.e., objects associated to URIs). In this way, we will be able to cover the cases where URI matching is infeasible: blank nodes as well as random, numerical URIs. Last but not least, we will compare our method thoroughly with similar ones that match resources using their content (i.e., literal values associated with URIs) in combination with their associated resources. The goal will be to estimate to which extent computationally expensive methods involving content or related resources can identify duplicate resources that URI matching misses.

Acknowledgments

This work was partially supported by the FP7 EU Projects OKKAM (contract no. ICT-215032) and LivingKnowledge (contract no. 231126).

7. REFERENCES

[1] A. Agarwal, H. S. Koppula, K. P. Leela, K. P. Chitrapura, S. Garg, P. K. GM, C. Haty, A. Roy, and A. Sasturkar. Url normalization for de-duplication of web pages. In *CIKM*, pages 1987–1990, 2009.

[2] E. Baykan, M. R. Henzinger, L. Marian, and I. Weber. Purely url-based topic classification. In *WWW*, pages 1109–1110. ACM, 2009.

[3] E. Baykan, M. R. Henzinger, and I. Weber. Web page language identification based on urls. *PVLDB*, 1(1):176–187, 2008.

[4] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[5] P. Bouquet, H. Stoermer, C. Niederée, and A. Mana. Entity name system: The back-bone of an open and scalable web of data. In *ICSC*, pages 554–561, 2008.

[6] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[7] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[8] H. Glaser, A. Jaffri, and I. Millard. Managing co-reference on the semantic web. In *WWW2009 Workshop: on the Web (LDOW2009)*, April 2009.

[9] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. J. Miller, and M. Wang. A framework for semantic link discovery over relational data. In *CIKM*, pages 1027–1036, 2009.

[10] I. Jacobs and N. Walsh. Architecture of the world wide web, volume one. *W3C Recommendation*, December 2004.

[11] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, June 1989.

[12] H. S. Koppula, K. P. Leela, A. Agarwal, K. P. Chitrapura, S. Garg, and A. Sasturkar. Learning url patterns for webpage de-duplication. In *WSDM*, pages 381–390, 2010.

[13] S. H. Lee, S. J. Kim, and S.-H. Hong. On url normalization. In *ICCSA (2)*, pages 1076–1085, 2005.

[14] P. Lehti and P. Fankhauser. Unsupervised duplicate detection using sample non-duplicates. *J. Data Semantics VII*, pages 136–164, 2006.

[15] T. Lei, R. Cai, J.-M. Yang, Y. Ke, X. Fan, and L. Zhang. A pattern tree-based approach to learning url normalization rules. In *WWW*, pages 611–620, 2010.

[16] P. Mika. Microsearch: An interface for semantic search. In *SemSearch*, pages 79–88, 2008.

[17] SWEO Community Project: Linking Open Data on the Semantic Web. Equivalence mining and matching frameworks. `http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/EquivalenceMining`.

[18] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In *ISWC 2009*, volume 5823 of *Lecture Notes in Computer Science*, pages 650–665. Springer, 2009.