Big Data Linkage for Product Specification Pages

Disheng Qiu
Università Roma Tre
disheng@dia.uniroma3.it

Luciano Barbosa
Universidade Federal de Pernambuco
luciano@cin.ufpe.br

Valter Crescenzi
Università Roma Tre
crescenz@dia.uniroma3.it

Paolo Merialdo
Università Roma Tre
paolo.merialdo@uniroma3.it

Divesh Srivastava
AT&T Labs – Research
divesh@research.att.com

ABSTRACT

An increasing number of product pages are available from thousands of web sources, each page associated with a product, containing its attributes and one or more product identifiers. The sources provide overlapping information about the products, using diverse schemas, making web-scale integration extremely challenging. In this paper, we take advantage of the opportunity that sources publish product identifiers to perform big data linkage across sources at the beginning of the data integration pipeline, before schema alignment. To realize this opportunity, several challenges need to be addressed: identifiers need to be discovered on product pages, made difficult by the diversity of identifiers; the main product identifier on the page needs to be identified, made difficult by the many related products presented on the page; and identifiers across pages need to be resolved, made difficult by the ambiguity between identifiers across product categories. We present our RaF (Redundancy as Friend) solution to the problem of big data linkage for product specification pages, which takes advantage of the redundancy of identifiers at a global level, and the homogeneity of structure and semantics at the local source level, to effectively and efficiently link millions of pages of head and tail products across thousands of head and tail sources. We perform a thorough empirical evaluation of our RaF approach using the publicly available Dexter dataset consisting of $1.9M$ product pages from $7.1k$ sources of $3.5k$ websites, and demonstrate its effectiveness in practice.

ACM Reference format:
Disheng Qiu, Luciano Barbosa, Valter Crescenzi, Paolo Merialdo, and Divesh Srivastava. 2018. Big Data Linkage for Product Specification Pages. In *Proceedings of 2018 International Conference on Management of Data, Houston, TX, USA, June 10–15, 2018 (SIGMOD/PODS '18),* 15 pages.
https://doi.org/10.1145/3183713.3183757

1 INTRODUCTION

A huge and continually increasing amount of structured data is available on the Web, in the form of *named entity* pages. Each such page is associated with a single entity, contains attributes of the entity and their values, including one or more *names* that serve to identify the entity and distinguish it from other entities in the same domain. Some examples of named entity pages on the Web are Wikipedia entity pages, product specification pages, organization pages, etc. A named entity is usually associated with multiple named entity pages provided by different web sources, often with partially overlapping and sometimes conflicting information. Several techniques have been proposed to collect named entity pages from different web sources [3, 24, 28]. Integrating the data that they provide to create a comprehensive, unified view of each entity represents a foundational step that can enable uncountable applications. However, performing web-scale integration of named entity pages raises novel and intriguing challenges due to its volume, velocity, variety and veracity [13].

The product domain represents one of the largest and most challenging domains, because of its huge number and variety of web sources and entities. RaF (Redundancy as Friend) is an ongoing research project that addresses the issue of integrating data extracted from product specification pages, i.e., named entity pages that publish structured information about a product. RaF aims at processing a very large number of sources across the entire web, not just a small number of pre-selected sources.

One might argue that a relatively small number of web sources offers most of the data about most of the products (for example, consider Amazon.com, which publishes data about an impressive number of products). Therefore, focusing on data offered by these *head sources* could in principle mitigate the variety and heterogeneity of information and alleviate the burden of product data integration. However, valuable information is actually published by an enormous number of *tail sources* [9, 12], i.e., sources that each provide a small number of product entities. Their data are important because: (*i*) they can improve entity coverage, since they often refer to *tail products*, i.e. entities that are present in a very small number of sources; and (*ii*) they can improve attribute coverage because they might provide tail attributes of the target entity. The goal of RaF is to *integrate both head sources and tail sources to create a comprehensive, unified view of both head products and tail products.*

Traditional approaches to data integration rely on a pipeline process that consists of three major stages: schema alignment, data linkage (or entity resolution) and data fusion [13]. Unfortunately, with a very large number of sources, this traditional pipeline becomes infeasible because schema alignment techniques cannot address the huge variety and heterogeneity that occurs in practice.

To illustrate a concrete scenario, we report the results of an analysis conducted on the publicly available Dexter dataset [28], which consists of $1.9M$ pages with product specifications from $3.5k$

SIGMOD/PODS '18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3183757

websites. The specifications extracted from the dataset contained more than 86k distinct attribute names.[1] Most of the attributes (85k) are each present in less than 3% of the sources, while only 80 attributes occur in 10% of the sources, with the most popular attribute occurring in 38% sources. We obtained a more skewed distribution considering attribute pairs for analyzing how attributes co-occur: a vast majority of pairs (25M) appears in less than 3% of the sources, and only 11 pairs are present in 20% of the sources.

The RAF approach for integrating product specifications from thousands of sources aims at taking advantage of the *opportunity* that products are named entities, and hence a product specification page usually publishes the product identifier (or simply 'id'). From a manual analysis over 1100 product pages of the DEXTER dataset, we have observed that 91% of the pages have ids. The presence of ids suggests that, unlike the traditional data integration pipeline, a potential strategy to address Web scale data integration for product specifications is to tackle data linkage at the beginning of the pipeline, *before* performing schema alignment on the sources.

Ids are present in product specification pages mainly for commercial reasons: product pages are largely published by e-commerce websites that need to expose the product ids to let them be indexed by shopping agents and available to customers who search products for comparing prices or consulting specifications. Moreover, the largest e-commerce marketplaces strongly encourage sellers and retailers to publish product ids, as they improve efficiency both for the internal management of data and for the exchange of data with search engines like Google and Bing.

While the presence of ids represents an opportunity, the volume and variety of sources raise several *challenges* as well:

- *Discovering Ids* – It is not easy to locate, within the html of the product specification pages, the string that represents the id; some sources adopt microdata markups (such as schema.org), but their diffusion is limited [25]. Usually ids consist of a single token[2] that, for a few categories of products, follow specific patterns. But at Web scale, it is *not possible* to focus on discovering ids using specific patterns (as done, in [29]) or to generalize these patterns (as done, for instance, in [23], which concentrated on a handful of sources) because of their large variety and skewed distribution. In the DEXTER dataset we observed 20k patterns, with the most frequent one matching less than 6.62% of values.
- *Identifying the Primary Id* – Product specification pages usually contain ids not only for the main product presented in the page, but also for related products (e.g. similar products, suggested products, etc.). Consider Figure 1, which shows a product page. Observe that it contains many product ids, but only region (1) reports the id of the main product presented in the page.
- *Resolving Ids* – There are several kinds of ids (UPC, SKU, model number, etc.), classified either as *local* ids, used to distinguish products within the same source (hence, not useful for linkage across sources), or as *global* ids, used by multiple sources. Local ids from different sources may conflict; similarly, conflicts may occur among global ids of products from different categories. Hence, different product specification pages associated with the same id are not guaranteed to refer to the same product.

Figure 1: Regions (dotted rectangles) that contain identifiers (ellipses) in a page from amazon.com.

In this paper we present the RAF solution to the problem of *big data linkage* for product specifications on the Web. Unlike traditional record linkage solutions [13], RAF aims to scale not only with the volume of data, but also the number of sources. RAF leverages the presence of product ids in the pages, and exploits the opportunities that derive from the *redundancy of information at the global level*, and from the *uniqueness and homogeneity of information at the individual source level*, as follows:

- At the global level, we observe redundant information: head (popular) products are present in several head (large) sources as well as in many tail (small) sources. Therefore, we expect that the ids of head products are spread across many sources. Further, many head products in a category will often co-occur in multiple sources.
- At the individual source level, we observe uniqueness and homogeneity of information: sources usually do not publish multiple product pages for the same product, and the structure and the semantics of information, within every source, tend to be regular. Hence, we expect that (i) the id of the product presented in a specification page is published in the same html *region* for every page in the same source, and (ii) the id extracted from that region is unique within that source. Consider again Figure 1: every page from the same source was built from a local template, and the semantics of the six regions (red dotted rectangles) is consistent in all the pages of the source: region (1) contains the id of the primary product, while regions (2 – 6) contain ids of related products.

Figure 2 illustrates the key intuitions underlying our approach to meet the RAF goal of *effectively* and *efficiently* integrating head and tail sources by linking all pages of head and tail entities: starting from known head entities in head sources, on the one hand it takes advantage of uniqueness and homogeneity of information at the local level to extract a global id for tail entities in head sources (even head sources offer many tail entities); on the other hand, it exploits the presence of head entities across sources, to extract a global id for known head entities in tail sources (even tail sources offer a few head entities). Again, starting from the known head entities in tail

[1] Attribute names have been normalized by lowercasing the string and removing all the non alpha-numeric characters.
[2] From our analysis of the Dexter dataset, this is true for 97.5% of the ids found in product pages.

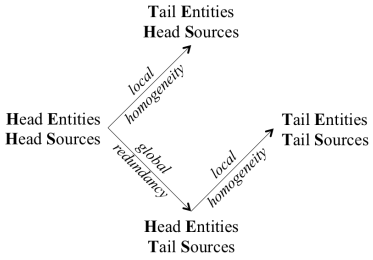


Figure 2: The RaF approach.

sources, it exploits uniqueness and homogeneity of information at the local level to extract a global id for tail entities in tail sources.

Our approach exploits the presence of product ids, but product specification pages also contain product titles and textual descriptions. Although these texts represent rich information, they are not effective to link across sources: different products can have very similar descriptions and titles (consider for example the MacBook descriptions in Figure 1), as well as, conversely, the same product can have different descriptions in different sources (some sources are terse, others publish verbose descriptions).

Contributions. We make several contributions in this paper. First, we introduce the novel RaF solution for linking a large diversity of both popular and rare product specifications from thousands of head and tail web sources, based on discovering and resolving product identifiers from the product specification pages (Section 2); the RaF big data linkage solution aims to scale not only with the volume of data, but also the number of sources. Second, we develop a two-phase architecture to solve the product identifier-based linkage problem: the first phase (Sections 3-5) extracts high-quality product identifiers, and the second phase (Section 6) generically resolves the potential ambiguity of identifiers across sources to determine which ones refer to the same products. Third, we report results from an extensive experimental evaluation of the RaF methodology on the publicly available Dexter dataset of product specification pages; the results (Section 7) demonstrate the effectiveness and efficiency of RaF in practice.

2 PROBLEM DEFINITION AND OVERVIEW

Websites usually organize product pages in categories by adopting locally defined taxonomies. From our perspective, local categories are interesting since product pages within the same site-category pair usually present a uniform html template, whereas even within the same website, product pages belonging to different categories may use quite different html templates. As a consequence, we find convenient to consider every site-category pair as an independent source of product pages:

Definition 2.1 (Source). A source s is the set of product pages that belong to the same local category in a website: $s = \{p_1, \ldots, p_m\}$.

Usually products are distinct within a source, so we assume uniqueness of information within every source,[3] but they may (and often do) overlap across different sources. We assume that every source publishes product pages having structural and semantic homogeneity: pages in the same source refer to the same local category and have a uniform structure, i.e. they conform to the same html template.

[3]This is not always true; for example, sources that publish used products may have multiple pages for different instances of the same product. However, this is a valid assumption for a vast majority of sites; and our techniques are tolerant to exceptions.

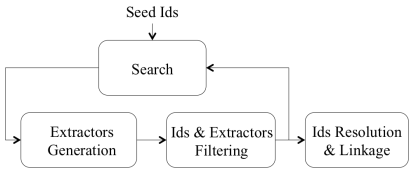


Figure 3: Overview of the RaF pipeline.

We define our problem as follows:

Definition 2.2 (Problem Definition). Given a set of sources $\mathcal{S}$, our goal is to compute a partitioning P of the pages in $\mathcal{S}$, such that each set in P contains all and only pages in $\mathcal{S}$ that refer to the same named entity (product).

A conventional solution to our problem would be to first perform schema alignment between all the sources, and then link products in these sources based on matches of their attribute names and values. This strategy, however, would be extremely costly since we are dealing with a very large number of sources. To have a feasible solution, we perform the linkage *up-front*, by using the ids of the products, which are usually available in product pages.

However, locating the product ids in those pages at the Web scale is challenging: (*i*) ids do not comply to any specific pattern, and standard html annotations are not widely adopted; (*ii*) in addition to the id of the main product present in the page, ids of several other products may be present in the same page, e.g., those of related and suggested products; (*iii*) some ids may only be local, not useful for linkage across sources, and some global ids may conflict with local ids of other products in other sources.

To address these challenges, we propose a *pipeline*, depicted in Figure 3, *that aims at extracting global ids that can be used to achieve high precision and high recall in product linkage.* Our approach leverages the redundancy of information that occurs at the global level and the regularities of the sources and uniqueness of information that occur at the individual source level. The RaF pipeline consists of an *iterative phase* followed by a *linkage* phase. The first iterative phase is composed of three stages, as follows:

- **Search**: We start from a *seed set* of high quality product ids, which are used as keywords for searching other product pages (global redundancy). As depicted in Figure 2, ids of head entities allow the retrieval of pages from both head and tail sources.
- **Extractors Generation**: Next, for every retrieved product page, we infer an extractor for every html region containing an occurrence of the searched ids. The extractors are used to obtain regions containing new ids from all the other product pages in the same source. The rationale is that due to the *local regularities of the sources*, it is likely that the same html region that contains an id in a certain page also contains an id in all the other pages of the same source. From every region returned by the extractors, we select a token that represents a candidate id for the primary product of the corresponding page.

- **Ids & Extractors Filtering**: The previous step may produce incorrect ids, for many reasons: a wrong selection of the candidate ids; a region returned by an inaccurate extractor; the presence of regions containing ids that do not refer to the main product of the page (e.g., similar products). To improve the precision, we take advantage of *local uniqueness* (usually, a source provides at most one page for a product) and consider the *duality between good extractors and correct ids*: an extractor is good if the ids of its regions are correct; similarly, an id is correct if it come from a region returned by a good extractor.

The fresh ids selected during the execution of the first phase are then iteratively used to trigger another search. To retrieve a larger number of pages, the search is performed preferring head ids, thus exploiting *global redundancy*.

The second phase of the pipeline consists of a single stage that resolves the ids collected by the iterative process to link products.

- **Ids Resolution & Linkage**: Due to the variety of information across sources, and the presence of local and global ids, different products could share the same id across sources. To improve the accuracy of linkage, we identify these conflicting ids. We consider that every source consists of a homogeneous set of products: although the criteria that define the uniformity of the product categories are local, not global, with a large enough dataset it is likely that many pairs of products co-occur in many sources because they adopt similar (even if they are not identical) categories. Then, we consider ids that co-occur with multiple ids in many sources more reliable for the purpose of linking.

To ensure efficiency of the process, we exploit the categorization of head/tail ids and head/tail sources. Head ids are used as keywords of search engine queries to identify new product pages and sources: we expect to get tail sources this way, since they might also contain at least a few head product ids. The learned extractors are then used to extract the ids of tail entities from the (head and tail) sources.

The search could happen within the pages obtained from a global crawl or may involve querying an external search engine. We adopt the first solution, working on the Dexter dataset to leverage the public availability of this dataset and enable reproducibility of our results, but our techniques apply equally to both cases.

3 REGIONS EXTRACTION AND IDS SELECTION

Every iteration begins with a search that returns a set of sources, those containing pages matching the ids used as keywords.

We compute an *extractor* for every *region* (of a page in a source) containing an occurrence of an id used as keyword in the previous search; then, we apply every extractor over all the pages of the corresponding source. Because of the local regularity, we expect that every region located by a given extractor has the same semantics in all the pages of the source (and hence contains an id). However, as shown in the example of Figure 1, the extracted regions may contain several tokens, and therefore we need to select the token that represents the global id of the main entity of the page.

In this section we describe our techniques to select, within the extracted regions, the token that most likely represents a global id of the product described in the page.

3.1 Validation of the Region Extractors

Our approach to generate the extractors is based on the studies on wrapper inference developed in [2, 5, 7, 8, 16]. We consider the document object model (DOM) tree representation of input web pages, and we adopt XPath expressions as concrete tool for extractors. Every textual leaf node of the tree is considered as a *region* of the page. Let r be an extractor and p a page: $r(p)$ denotes the region returned by the application of r over p. Let x be one of the regions of page p containing a token matching with an id. To produce an extractor, we compute the XPath expression rooted at the deepest ancestor node of x that occurs exactly once in a significant percentage of pages of the source.[4]

Example 3.1. Figure 4 presents the DOM trees of three pages, p_{11}, p_{12}, p_{13} from the (fictional) source s_1. Assume that MF839LLA was the keyword id that made the search return these pages. Rectangles denote leaf nodes matching such a keyword. The yellow rectangle corresponds to a region containing the correct product id in every page; the white rectangles correspond to regions that contain ids that are not suitable for our purposes, as they do not refer to the main product on the pages; the gray rectangle, whose region publishes a used article, could be correct, but for some pages it may miss the id (as for page p_{12}). Let us now concentrate on the node associated with the yellow rectangle on page p_{11}, and consider its ancestors. The node p is the deepest that appears (exactly) once in every page of the source. A straightforward XPath expression to locate its occurrences in every page is //p; this base expression is then used for building an XPath expression descending to the target region: //p/span[1]/text(). Note that such an expression works correctly also on p_{12} and p_{13}.

As a final validation step, we filter out extractors returning regions that are unlikely to contain the id of the product in the page, as follows. First, we eliminate extractors that produce null values for more than 50% of the pages in the source. The rationale is that a region that includes the main product id should be present in most pages of the source. Second, we eliminate extractors that do not return different values from page to page. Since we are assuming uniqueness of information at the source level, extractors that produce duplicates cannot be considered valid. These simple heuristics filter out not only wrong extractors, (e.g., those extracting pieces of the page template), but also some of the extractors that return regions containing the ids not related to the main product in the pages. Continuing our example referring to Figure 4, any extractor returning the region corresponding to the first similar item (e.g. /div[2]/span[2]/text()), would be discarded, as in pages p_{12} and p_{13}, every extracted node has identical contents.

3.2 Selecting the Identifiers

The extractors return regions that are likely to contain the single-token representing the ids of the products in the corresponding pages. However, since these regions typically consist of several tokens, for every page we need to identify the token that most likely represents an id for the main product presented in that page.

[4] We have empirically observed that 30% is a good threshold.

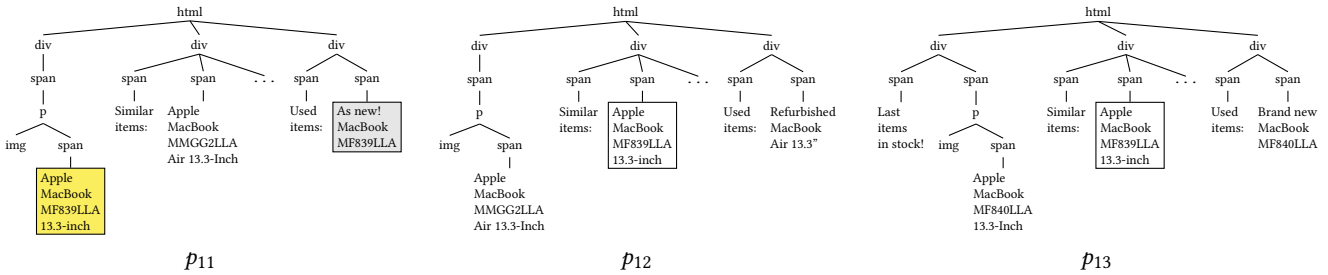


Figure 4: DOM tree representation of three (fictional) web pages.

For example consider region 1 in Figure 1: the id `MF839LLA` is part of a long textual description of the product.[5]

Let $r(p) = [t_1, t_2, \dots]$ denote the sequence of tokens composing the region $r(p)$ located by the extractor r from page p. Our goal is to identify the token $id(r, p) \in r(p)$, that most likely represents the id for the entity in p.

To select ids within textual regions, we leverage on the *local uniqueness of information* within the sources. We take the frequency of the tokens to assess their ability to serve as ids, and we select the token of the region with the smallest number of occurrences within the source. The rarest and most selective tokens should be considered good candidate ids. However, considering every source separately does not work well when searching for global ids. It turns out that many sources, especially tail sources, contain some tokens that are locally rare but do not represent global ids. For example, in many tail sources there is just one laptop with a touchscreen: if the keyword `touchscreen` is used along with the description of the product, it would be very selective (at the source level) and thus it could be erroneously regarded as a good candidate id. Even if these tokens might appear as very selective at local level, they are much more frequent if considered globally, especially in the head sources. Continuing the above example, `touchscreen` is a pretty frequent token at the global level.

Therefore, we introduce a scoring function by considering the collection frequency, defined as the total number of occurrences of a token in the whole collection of input pages $\mathcal{P}$. We compute for each token its frequency in $\mathcal{P}$ and, namely, for each token $t \in r(p)$, we compute the $idScore(t, \mathcal{P})$, as follows:

$$idScore(t, \mathcal{P}) = log \frac{|\mathcal{P}|}{|\{p \in \mathcal{P} : t \in r(p)\}|}. \quad (1)$$

Given a ranking of all the tokens $t \in r(p)$, with $p \in \mathcal{P}$, we select as candidate id from the region r, the token with the highest score:

$$id(r, p) = \arg\max_{t \in r(p)} idScore(t, \mathcal{P}).$$

Our definition of the *idScore* leverages both local and global characteristics: A good id is required to be both as selective as possible at local level, and as redundant as possible at global level.

Example 3.2. Consider again the pages in Figure 4. Suppose we obtained two extractors, r_{11} and r_{12}, which return the regions in the yellow and gray nodes, respectively (the extractors inferred from the regions corresponding to the nodes in the white rectangles are eliminated by the extractor validating step). Figure 5 shows the sequences of tokens extracted by r_{11} and r_{12} from the pages p_{11}, p_{12}, and p_{13}, ordered by the associated *idScore*. As the product ids are much less frequent than tokens such as `MackBook` or `Apple`, they appear first. Note that r_{12} associates the token `Refurbished` with page p_{12}. In the next section, we discuss our approach to filter out these erroneous regions and ids.

4 CORRECT IDS AND GOOD EXTRACTORS

Due to the irregularities in the html template of the pages, or to the quality of the ids used to infer the extractors, the previous step might select tokens which are not correct ids.

Our approach to filter out these incorrect ids is based on the intuition that a good extractor provides regions containing correct ids, and vice-versa correct ids are located in regions returned by good extractors. The redundancy of information among sources is the key ingredient to exploit this duality. We jointly evaluate the correctness of ids and the quality of extractors by using a mutual reinforcement approach.

We represent the relationships between extractors and ids by means of a weighted bipartite graph between the set of values of the candidate ids extracted by the previous stage $\mathcal{I} = \{id_1, \dots, id_n\}$,[6] and the set of extractors computed for the whole dataset, $\mathcal{R} = \{r_1, \dots, r_m\}$, where there is an edge between an extractor r and an id id, if there is a region, extracted by r, that contains id as candidate id. Edges are weighed based on our belief that a candidate id is a correct id for the entity published in the page from which it has been extracted by extractor r. We consider natural properties that derive from the *global redundancy of information* of our setting. Namely, we decrease our belief that id is correct if it has been discarded by some other source (it was in the region, but another token, with a higher *idScore*, has been preferred as candidate id). Similarly, considering our *local uniqueness of information* assumption, we decrease our belief on the correctness of id extracted by r if it is proposed as candidate id for multiple pages from the same source.

Following these observations, we introduce a weight for the edges of our bipartite graph between the ids and extractors. Let $\mathbb{1}(\cdot)$ denote the indicator function, which returns 1 if its argument is true, 0 otherwise. Given the extractor r_j for the source s, and a

[5] Our analysis on the Dexter dataset reports that 46% of the ids are in textual descriptions. Appendix A.3 reports the distribution of the average number of tokens returned by the extractors computed in a complete run of our pipeline on the Dexter dataset.

[6] We are considering distinct values: the same id can be extracted from different sources.

s_1	s_2	s_3
$r_{11}(p_{11})$ = [MF839LLA, MacBook, Apple]	$r_{21}(p_{21})$ = [MF839LLA, Refurbished, MacBook, Guaranteed]	$r_{31}(p_{31})$ = [MF839LLA, MacBook]
$r_{11}(p_{12})$ = [MMGG2LLA, MacBook, Apple, Air]	$r_{21}(p_{22})$ = [MMGG2LLA, MacBook, as, new,]	$r_{31}(p_{32})$ = [MMGG2LLA, MacBook]
$r_{11}(p_{13})$ = [MF840LLA, MacBook, Apple]	$r_{21}(p_{23})$ = [MF840LLA, fully, working,box, original]	
$r_{12}(p_{11})$ = [MF839LLA, MacBook, New!]	$r_{22}(p_{21})$ = [MF839LLA, compare]	$r_{32}(p_{31})$ = [MF839LLA]
$r_{12}(p_{12})$ = [Refurbished, MacBook, Air]	$r_{22}(p_{22})$ = [accessories, this, product, for]	$r_{32}(p_{32})$ = [MMGG2LLA]
$r_{12}(p_{13})$ = [MF840LLA, MacBook, Brand new]	$r_{22}(p_{23})$ = [accessories, related]	

Figure 5: Extractors and tokens from three sources.

page $p \in s$, the weight of id_i wrt r_j is set as follows:

$$idWeight(id_i, r_j) = \frac{1}{1 + c_1(id_i, r_j) + c_2(id_i, r_j)}, \tag{2}$$

where:

- $c_1(id_i, r_j) = \mathbb{1}(\exists p' \in s', r' | id_i \in r'(p') \setminus \{id(r', (p')\} \wedge s' \neq s)$, that is, id_i has been discarded by another source s', $s' \neq s$;
- $c_2(id_i, r_j) = \mathbb{1}(\exists p' \in s | id_i \in r_j(p'))$, that is, id_i has been considered an id extracted using the same r_j for two distinct pages of the same source s.

We represent our weighted bipartite graph by means of the bi-adjacency $n \times m$ matrix $\mathbf{C}$, called *confidence matrix*: if id_i is a candidate id extracted by r_j, $\mathbf{C}(i, j)$ is equal to $weight(id_i, r_j)$, i.e., our confidence that it is a correct id, otherwise $\mathbf{C}(i, j)$ is equal to 0.

4.1 Mutual Reinforcement of Belief

We exploit the duality between correct ids and good regions to mutually reinforce our belief of their quality. Let $\mathbf{I}$ denote a n-dimensional vector, where each element $\mathbf{I}(i)$ is a weight indicating our belief that id_i is a correct id. Similarly, let $\mathbf{R}$ denote a m-dimensional vector, where each element $\mathbf{R}(j)$ is a weight indicating our belief that the extractor r_j returns regions that contain correct ids.

Given the belief of correctness of the ids and the confidence matrix $\mathbf{C}$, we can update our belief in the extractors. In particular, if the ids extracted by r_j are correct, we consider r_j a good extractor. This can be represented by: $\mathbf{R}(j) = \sum_{k=1}^{m} \mathbf{I}(k) \cdot \mathbf{C}(j, k)$ which can be written in matrix-like form: $\mathbf{R} = \mathbf{I} \times \mathbf{C}$.

Analogously, we can update $\mathbf{I}$ from $\mathbf{R}$ and $\mathbf{C}$: if an extractor is good, the ids that come from the regions that it produces are likely to be correct. This can be represented by: $\mathbf{I}(i) = \sum_{k=1}^{n} \mathbf{R}(k) \cdot \mathbf{C}(k, i)$ which can be written in matrix-like form: $\mathbf{I} = \mathbf{R} \times \mathbf{C}^T$.

The above equation exploits the *global redundancy of information*: the more regions select an id, the more reliable is such an id.

As reported in Algorithm 1, we exploit the duality between good extractors and correct ids by jointly computing the two vectors iteratively until convergence.[7] We begin from a configuration in which all the extractors are considered equally good, i.e., we set their initial weights to $\alpha = 0.5$. At the end of the iteration, we select the extractors and ids whose weights are greater than a threshold, T_{id} and T_r respectively.[8]

[7] It is worth observing that our algorithm corresponds to Kleinberg's HITS [22] on a bipartite graph, whose convergence is proved [18].

[8] We set $T_{id} = T_r = .5$. We have empirically observed the approach is robust w.r.t. the thresholds.

input : set of candidate ids, $\mathcal{I} = \{id_1, \ldots, id_n\}$
input : set of extractors, $\mathcal{R} = \{r_1, \ldots, r_m\}$
output: set of extractors $R \subseteq \mathcal{R}$
output: set of id $I \subseteq \mathcal{I}$

```
for i = 1 ... m do
    R(i) = α;
end
repeat
    I = R × C^T;
    R = I × C;
    normalize I and R;
until R does not changes;
R = {r_i ∈ ℛ | R(i) > T_r};
I = {id_i ∈ ℐ | I(i) > T_id};
return R, I;
```

Algorithm 1: Filtering identifiers and extractors by mutual reinforcement of belief.

Example 4.1. Consider again source $s_1 = \{p_{11}, p_{12}, p_{13}\}$, from Example 3.1. Let $s_2 = \{p_{21}, p_{22}, p_{23}\}$, $s_3 = \{p_{31}, p_{32}\}$ be two more sources of our dataset. Suppose that s_1 is associated with the extractors r_{11}, r_{12}; s_2 with r_{21}, r_{22}; s_3 with r_{31}, r_{32}. Also, suppose that these extractors, applied over the pages of the corresponding sources, produce the sequences of tokens (ordered by *idScore*) shown in Figure 5. Observe that MF839LLA, MMGG2LLA, and MF840LLA are always proposed as the best candidate ids. Conversely, Refurbished is ranked first once in $r_{12}(p_{12})$, while it is out-ranked by another token in $r_{21}(p_{21})$, thus decreasing the confidence on the correctness of it as an id, as well as of the extractors that produce it. Similarly, notice that the token accessories is extracted from two different pages in the same source s_2. Also in this case, our confidence on the correctness of the involved id and extractor is lower. The bipartite graph between the set of ids $\mathcal{I}$ and the set of extractors $\mathcal{R}$ is then represented by the following matrix:

$$\mathbf{C} = \begin{array}{c} \\ \texttt{MF839LLA} \\ \texttt{MMGG2LLA} \\ \texttt{MF840LLA} \\ \texttt{Refurbished} \\ \texttt{accessories} \end{array} \begin{array}{c} \begin{array}{cccccc} r_{11} & r_{12} & r_{21} & r_{22} & r_{31} & r_{32} \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{1+1+0} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{1+0+1} & 0 & 0 \end{pmatrix} \end{array}$$

Applying Algorithm 1 to the above matrix, we obtain the following weights: $\mathbf{R}$ = [1.0, 0.69, 1.0, 0.43, 0.75, 0.75] and $\mathbf{I}$ = [1.0, 0.75, 0.58, 0.07, 0.04]. Note that r_{11} and r_{21} (whose weights

are 1) are considered better than r_{12} and r_{22} (whose weights are 0.69 and 0.43, respectively). Similarly, ids `MF839LLA`, `MMGG2LLA`, and `MF840LLA` (whose weights are 1, 0.75 and 0.58, respectively) are considered more correct than `Refurbished` and `accessories` (whose weights are 0.07 and 0.04, respectively).

5 SEARCH AND ITERATIONS

The ids extracted and filtered in the first phase are used as keywords to trigger new searches and iterate the process. Every search returns a set of sources, namely those containing pages that match the keywords. From these sources new ids are obtained. The iteration concludes when no more ids are found. Two main aspects affect the performance of the iteration: *(i)* the choice of the sources from where to extract the ids in the current iteration, and *(ii)* the selection of the ids for feeding the search in the next iteration.

Our strategy to govern the iterations leverages information redundancy. As for the choice of the sources, there is a twofold advantage to process sources having many matches with the ids used by the previous searches. First, there are many more matching ids belonging to regions that refer to the same extractor, so favoring the production of robust extractors. Second, if several ids match the pages of the source, the overall costs can be significantly reduced, as one of the most expensive operation, i.e., parsing html pages to build DOM trees, needs to be performed only once for many matches, instead of being repeated after every match.

As for the selection of the ids, we exploit the evidence accumulated in the previous iterations: the more popular is an id across sources, the more likely it is a correct global id. Therefore, we rank ids by their number of occurrences in the sources, and we feed every iteration of the search process with a bulk of head ids. As illustrated in Figure 2, by expanding the search with head ids we can also reach and analyze tail sources as soon as there is enough evidence for including them in the process. Because of their popularity, head ids are more likely than tail ids to bring out multiple new tail sources.

Concretely, the RaF strategies to govern the iteration of the process can be detailed as follows: we perform searches using the top-k head ids; from the retrieved sources we process those accumulating a sufficient number of matching ids. Since the distribution of the size of the sources is skewed, we process a new source only if it has a minimum number of matches, which works for head sources, or a minimum percentage of matches with respect to the source size, which works for tails sources that could not reach the threshold on the minimum number of matches.

6 CONFLICTING IDS RESOLUTION

Once we have associated an id with every page, we could conclude the process by trivially grouping pages that share the same id and thus considering the linkage of the corresponding product entities. However, because of the heterogeneity of the data, it may happen that some ids *conflict*: though identical, they refer to different products. To give an example, consider the id `H4614`, which is associated both with a model of eyeglasses and with a toilet accessory (a towels hook). Another typical example of these conflicting ids are product codes *locally* assigned by the sources: the scope of these ids should be local to the sources, but it happens that some codes are incidentally adopted by different sources for completely unrelated products. If these codes are selected as ids by the previous iterative phase of the pipeline, for our data linkage purposes, they might create clusters of pages that actually refer to different entities. Conflicting ids may also arise due to noise introduced by the extraction and filtering stages. For example, in the pages where the product is an accessory related to another product (e.g., headphones for smartphones), the ids of the two distinct items (accessory and related product) can be confused.

To reduce the number of these incorrectly linked pages, we exploit the semantic homogeneity that locally occurs in every source, and the global redundancy of information across sources. Our key intuition is that an id is unlikely to correctly link together two products if it has been associated with several products that belong to different categories, e.g., sunglasses vs toilet accessories.

Ideally, we could consider the input coarse categories of the Dexter dataset. However, due to the heterogeneity of the sources, any coarse classification is inherently noisy. For example, we have found sources that offer products belonging to distinct categories (e.g., monitors with laptops). Also, coarse categories, due to classification errors or ambiguities, might include heterogeneous sources, such as, a source about lenses in the cameras category.

Yet another option is to consider the names of the local categories as associated by the website containing the source itself, and provided by the focused crawler used to gather up the pages of the Dexter dataset. Unfortunately, also this solution is infeasible because of the significant variety among the sources. In the Dexter dataset there are 1,028 distinct category names, with a power law distribution (only 20 names occur in more than 10 sources). Such a heterogeneity prevents a direct adoption of local categories; an id for a product under the category *Smartphone* of a given website could be considered inconsistent if associated with a product under the category *Samsung Mobile Phone* of another website. Therefore, to conclude, every source uses a local, sometimes surprising, taxonomy for its products, and the local classification schemes adopted by such big number of sources can hardly be reconciled under an unified superimposed taxonomy.

To overcome this issue, we adopt a data-driven approach, creating clusters of sources that are more likely to be similar based on the products they include. Our approach consists of computing a partition of the sources in $\mathcal{S}$, and to consider valid for linkage only the ids that belong to sources grouped in the same cluster; conversely, we filter out linkages coming from ids whose sources have been classified in different clusters. The clustering procedure is implemented by means of the *Louvain Method* [4], a popular, efficient and scalable graph clustering algorithm. We create a complete graph, where nodes represents the sources in $\mathcal{S}$, and the weight of every edge is the similarity between the connected pair of sources.

A crucial aspect here is how to compute the similarity between two sources. As a source is a set of products whose ids have been already found, one may consider a standard approach such as the Jaccard-index over the set of ids discovered in the previous phase, that is: $Jaccard(s_i, s_j) = \frac{|\{id, id \in s_i\} \cap \{id, id \in s_j\}|}{|\{id, id \in s_i\} \cup \{id, id \in s_j\}|}$.

However, in our setting the Jaccard-index is not reliable and, most importantly, it has all the limitations of any *local* (i.e., based only on information of the two compared sources) similarity score,

as it does not consider the context (that is to say, for our setting, all other sources) in which the two sources occur. Consider for example a pair of sources of unrelated products (say, sunglasses and toilet accessories) and suppose that the two sources adopt the same internal codes for their products, e.g., simple sequential numbers in the same range. The Jaccard-index would consider the two sources very similar, which is undesirable, even if no other website would list all these ids within a single (local) category.

6.1 Global Co-Occurrence Source Similarity

We introduce a novel similarity index tailored to our problem setting, denoted G_β, which aims at exploiting global evidence from our big data context. G_β is based on the observation that the greater the number of sources in which two ids co-occur, the more confident we are that these two ids are associated with the same product. Every time that a website puts in the same local category, i.e., the same source, two ids, the website is somehow expressing a "vote" on the semantic homogeneity of the corresponding products, at least according to the taxonomy of categories it locally adopted.

Given a pair of ids id_a and id_b such that $id_a \neq id_b$, let $\beta(id_a, id_b)$ be the number of sources where id_a and id_b co-occur, normalized with respect to the number of sources, as follows:

$$\beta(id_a, id_b) = \frac{|\{s \in \mathcal{S}\ s.t.\ id_a \in s \wedge id_b \in s\}|}{|\mathcal{S}|}.$$

Observe that $\beta(id_a, id_b)$ equals 0 if the two ids never co-occur, 1 if they co-occur in every source. Given a pair of sources s_i and s_j, we define the similarity index $G_\beta(s_i, s_j)$, as follows:

$$G_\beta(s_i, s_j) = \frac{\sum_{(id_a, id_b) \in s_i \times s_j} \beta(id_a, id_b)}{|s_i \times s_j|}.$$

Note that $G_\beta(s_i, s_j)$ assumes higher values if s_i and s_j share many ids that globally co-occur in many sources. It equals 1 if s_i and s_j share all the ids, and every pair $(id_a, id_b) \in s_i \times s_j$ occurs in all the sources $\mathcal{S}$. Conversely, if s_i and s_j share all the ids, but these do not co-occur in any other source, $G_\beta(s_i, s_j)$ assumes a small value, whereas the standard Jaccard-index would be 1.

Example 6.1. Consider Figure 6, which shows a set of sources with their related ids. Note that some sources have ids of products from heterogeneous categories (we use suitable placeholders t_i and m_j for ids of tvs and monitors, respectively). β(`MF839LLA`, `MMGG2LLA`) equals 1/3, since the pair (`MF839LLA`, `MMGG2LLA`) occurs in 2 out of 6 sources (s_1 and s_4). Similarly β(`MF839LLA`, t_2) is 0 as no source contains them both. Observe source s_2: it shares 2 ids with s_1 (`MMGG2LLA` and `MF840LLA`), as well as with s_6 (t_1 and t_2). While the Jaccard-index for s_2 is equally similar to s_1 and s_6 (namely, 0.4), the G_β index considers it closer to s_1 (0.22 vs. 0.14), as its ids co-occur more often in the dataset. All the values of the indexes for this example are detailed in Appendix A.5.

source (categories)	set of ids
s_1 (laptop)	{`MF839LLA`, `MMGG2LLA`, `MF840LLA`}
s_2 (laptop and tv)	{`MMGG2LLA`, `MF840LLA`, t_1, t_2}
s_3 (laptop)	{`MMGG2LLA`, `MF840LLA`, `MPTT2LLA`}
s_4 (laptop and monitor)	{`MF839LLA`, `MMGG2LLA`, `MF840LLA`, m_1}
s_5 (laptop and monitor)	{`MF839LLA`, `MF840LLA`, m_1, m_2}
s_6 (tv)	{t_1, t_2, t_3}

Figure 6: Sources and ids for Example 6.1.

7 EXPERIMENTS

For the experimental evaluation of the RaF pipeline we use the publicly available Dexter dataset: $1.9M$ pages from $3.5k$ websites that have been collected by means of a focused crawler [28] trained to gather product pages from 10 coarse categories: camera, cutlery, headphone, monitor, notebook, shoes, software, sunglasses, toilets, tv. The crawler grouped pages in 7,145 clusters, corresponding to the local categories exposed by the websites (e.g., "led monitors"). These clusters correspond to our definition of *sources*.

Although the goal of the Dexter system was to crawl only detailed product specification pages describing individual products, the corpus contains also several pages that, in our perspective, represent noise: summary pages listing many products, pages of news, general error pages. From our analysis of a random sample of 1100 pages, we estimated that these noisy pages cover about 30% of the corpus.

We classify every source as either head or tail by taking the median of the source size distribution (the sum of the sizes of heads equals the sum of the sizes of tails) as a threshold between the two categories. In the Dexter dataset, the vast majority of sources are tails as the median for sources is 133 pages. We provide additional details on the Dexter dataset in Appendix A.1.

The seed set to start the RaF pipeline was obtained by leveraging the microdata annotations that are present in a portion of the sources. We considered the pages adopting the schema.org standard, and we extracted all the values marked with the following properties: gtin(8-14), productID, sku, mpn, model. In total, we obtained $127k$ ids, from which we randomly chose 10% as seed set to start our pipeline. The remaining are used as a ground truth for a evaluating the recall of id discovery.

We now present the results of our experimental analysis to evaluate the RaF pipeline. We present experiments to assess the performance of the various components, considering them both in isolation and during end-to-end pipeline executions.

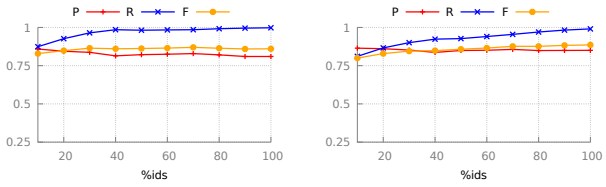
7.1 Extraction and Ids Selection

To evaluate the quality of the extraction process we compare the values extracted by a golden set of hand-crafted rules, with those returned by the set of extractors generated by our technique.

To build the golden set we have randomly selected 125 sources from our dataset and we have written all the XPath expressions extracting the regions with the main product ids.

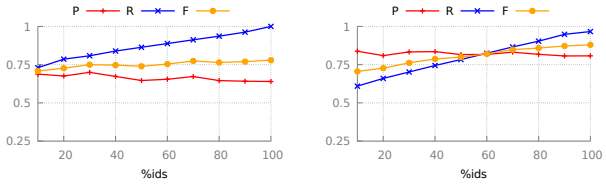
As metrics we adopt precision and recall at the level of the extracted values. Given an extractor r, let $r(U)$ denote the set of values extracted from the set of pages U; we compute precision (P), recall (R), and F-measure (F) wrt the corresponding golden rule r_g, as follows: $P = \frac{|r_g(U) \cap r(U)|}{|r(U)|}$; $R = \frac{|r_g(U) \cap r(U)|}{|r_g(U)|}$; $F = 2\frac{P \cdot R}{P+R}$.

The number of ids used as keywords in every search may vary, as it depends on the ids that have been already discovered. Therefore, the quality of the extractors can be affected by the quality and quantity of the ids used as keywords during every search. In this first



(a) without validation (b) with validation

Figure 7: Quality of region extraction.



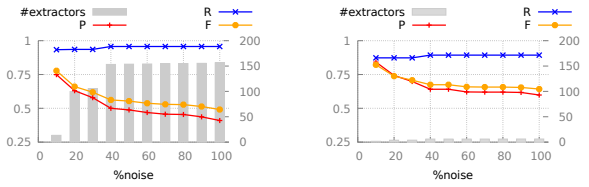
(a) without validation (b) with validation

Figure 8: Quality of region extraction and id selection.

experiment, we considered increasing percentage of input correct ids to infer the extractors: Figure 7(a) shows average precision and recall in the absence of the validation step for extractors. Whereas the recall improves with the percentage of ids, the precision slightly decreases. This negative effect can be explained by considering that a larger number of ids entails a larger number of matches with wrong regions, i.e., regions containing ids that do not correspond to the main product presented in the page. Figure 7(b) presents the same results with the extractor validation step enabled. Even if there is a loss in precision, it is more important not to inject noisy ids, since recall can be recovered. We will see that the positive effects of the validation are much more evident for the selection of the ids, and in the presence of noise.

Ids Selection. To evaluate the ids selection based on the *IdScore*, we have applied our method over the regions extracted in the previous experiment, and we have compared the set of the tokens with the highest score with a golden set of manually selected correct ids from the same set of 125 sources. Figure 8 shows the results of this experiment. It is worth observing that, compared to the results of the extraction in isolation (Figure 7), the validation produces a much more apparent gain in precision. There is also a loss in recall, but since this can be recovered by subsequent iterations of the pipeline, it is even more important not to inject noisy ids, as we discuss in the next experiment.

Influence of search quality on region extraction. The quality of the extractors is influenced by the quality of the ids used as keywords to locate the regions during the search. To evaluate how our extraction process is resilient to this aspect, we have set up another experiment by considering for every source of the previous experiment a set of keywords containing 20% of the correct ids; to this set we added a set of noisy tokens randomly taken from the leaf nodes of the pages of the source (to let them match with the



(a) without validation (b) with validation

Figure 9: Quality of region extraction corrupting the search with noisy keywords.

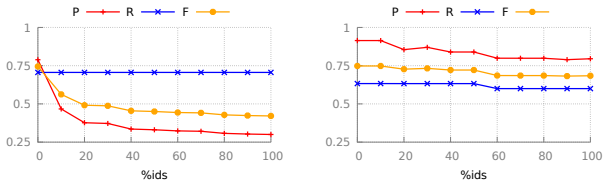
pages themselves). Then, we have evaluated again precision and recall of the generated extractors, varying the percentage of noisy ids. Figure 9 shows how important is the role of the validation, as it improves significantly the precision, with a small drop of the recall. The validation step produces also positive results by reducing the number of generated extractors by one order of magnitude: Figure 9 plots the average number of generated extractors (gray bars, whose scale is in the right vertical axis) with and without validation by varying the percentage of noisy ids in the search. While in absence of noise the number of extractors is rather limited (about 2, on average), the validation step is important to prevent the noisy ids from introducing too many noisy extractors.

Results on the end-to-end pipeline. Overall, out of 967k extractors generated during a run of the pipeline, the validation removes 644k extractors (66%). Specifically, 628k are eliminated because they extract too many null values, and 15k (on average 2 for every source) because of duplicated values. The average number of regions per source is 56, a surprising high number that can be explained by considering the results of the experiment depicted in Figure 9 (it shows how the number of generated extractors increases with the noise), and the noise in the input Dexter dataset, which contains several product listings pages (as they publish many ids, they give rise to a large number of regions). A manual inspection of the sources generating a large number of extractors revealed that this phenomenon is more frequent on the smallest sources, because the heuristics of the extractors validation step are less effective on small collections of pages. Overall, the majority of pages have less than 7 extractors, as shown in Figure 11(a), which plots the distribution of regions per page. As we discuss in the next section, many other regions are filtered in the following phase of the pipeline.

7.2 Ids and Extractors Filtering

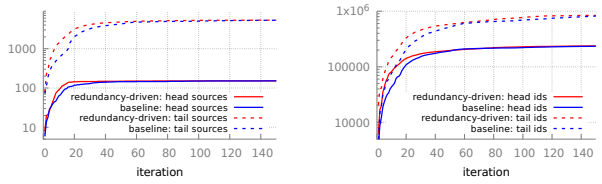
The mutual reinforcement of belief aims at selecting correct ids as well as good extractors, filtering out those that are unlikely to extract regions containing correct ids.

In order to evaluate effects on precision and recall of this process we have randomly chosen 25% of the golden ids, i.e., the ids extracted by the set of hand crafted rules, from each of the 125 sources of the previous experiment. Then, we retrieved the pages matching these golden ids searching the whole Dexter dataset. Before performing the searches, we simulated the presence of noise by injecting noisy keywords as in the previous experiments. On the set of returned pages, we extracted and filtered the ids and



(a) IdScore (b) Mutual reinforcement of belief

Figure 10: Quality of ids filtering vs percentage of noise.

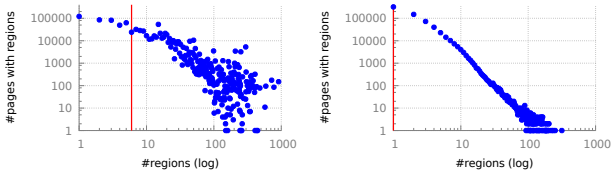


(a) Discovered sources (log scale) (b) Discovered identifiers (log scale)

Figure 12: Ids and sources discovered during the iterations.

we computed the precision and recall of the resulting ids, according to the *idScore* (Eq. 1), and the score obtained from the mutual reinforcement of belief approach by using the *idWeight* (Eq. 2). Figure 10 compares the results of mutual reinforcement of belief to *idScore* and shows that the mutual reinforcement of belief is robust to noise, as it produces highly precise results and prevents the low quality extractors from introducing too many noisy ids within the pipeline iterative phase.

Results on the end-to-end pipeline. In a complete run of the RAF pipeline, 65% of the extractors surviving to the validation step did not survive the filtering based on the mutual reinforcement of belief (only $112k$ extractors out of $322k$ were saved). Similarly, about 60% of the ids ($1.6M$ out of $2.61M$) were filtered out by Algorithm 1. Figure 11(b) plots the distribution of regions after the filtering step. Compared to the same distribution before the filtering (Figure 11(a)), it is apparent that the mutual reinforcement of belief has polished many regions: the total number of pages that contain at most 3 regions after filtering is almost doubled ($541k$ vs. $288k$).



(a) Before filtering (b) After filtering

Figure 11: Number of regions per pages. The red line indicates the median of the distribution.

7.3 Iterations

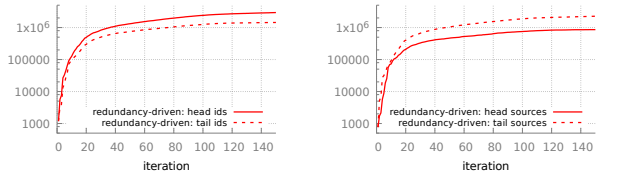
We now focus on evaluating our redundancy-driven strategies to govern the iteration (Section 5). To this end, we have run the first iterative phase of the pipeline (composed of three stages) with different configurations, until termination. We set as a baseline a *random* approach: in every iteration a bulk of $100k$ random ids is used to feed the search, and we randomly select the sources to process in the next iteration. A second configuration adopts our *redundancy-driven* strategy: we select the top $100k$ head ids, and we process the sources that have a least 10% or at least 100 pages matching with those ids. To compare the costs of the two configurations, we consider a simple cost model assigning an unit to every html page processed during the iterations. Also, since we aim at studying the evolution of the iterations across several configurations, we fix a bound to the cost of every iteration (that is, to the total number of processed pages) so that we can fairly compare the results obtained during the end-to-end execution of the pipeline in terms of iterations.

Figures 12(a) and 12(b) plot the (cumulative) number of sources and ids discovered by the iterative phase of the RAF pipeline. The prioritized configuration discovers sources more efficiently, because of the exploitation of head ids in the search process.

The two configurations converged to the same results (they differ for less than 1%) demonstrating that our redundancy-driven strategy does not introduce any significant bias: they found $1.07M$ distinct ids, with $1.55M$ occurrences in $652k$ pages (on average about 2.37 ids per page) from 5,477 sources (152 heads, and 5,325 tails), 75% of the sources in the DEXTER dataset. Using the ground truth composed by the ids extracted with microdata annotations, RAF discovered 70% of the ids. To understand these results, we note that the DEXTER dataset contains about $1.9M$ pages, but about 30% of them are noisy. Analyzing the sources that we lost, we realized that they include a dozen of large sources containing reviews and descriptions of used products. These sources, which are not product specification pages (and hence may be considered out of the main scope of our goals) violate our assumptions, e.g., used product pages in the same ebay.com category do not have the same template as sellers are free to customize the layout of a portion of the page; also, several distinct used product pages of a source contain the same id.

The two configurations produce the same number of ids, but have different costs. The redundancy-driven strategy allows the pipeline to converge to 99% of the ids in 132 iterations, while the baseline needs 152 iterations to obtain the same results. The gain is evident also considering the coverage of the sources, i.e., the number of sources for which at least one id has been extracted. In this case 99% of the sources has been discovered after 103 iterations vs. 136 discovered by the baseline. The strongest gains occur during the early iterations (in the first 15 iterations, redundancy-driven is about 2× more effective than the baseline) wrt the number of sources, as shown by the solid lines in Figure 12(a), while parsing many fewer pages.

This experiment has been conducted by using quite a loose cost bound ($100k$ pages) per iteration. This facilitates the baseline as it has good chances to pick up a significant number of head entity ids in one iteration even if it makes random choices. By repeating the experiment with a smaller cost per iteration, ranging from 100 to $10k$, the advantages of the redundancy-driven approach become



(a) Linkages (log scale)

(b) Linkages grouped by sources (log scale)

Figure 13: Linkages discovered during the iterations.

much more apparent: by using a bound for the cost as low as 100, the random baseline cannot even find any id for several iterations.

Figure 12(b) shows that the redundancy-driven strategy performs better also for discovering ids. Similarly to sources, to classify ids as either head or tail we consider the median of the distribution of id occurrences as a threshold separating head ids from tail ids. We found 238k head id occurrences, and 842k tail id occurring at least in two sources. However, as expected, many ids (868k) occur in just one source and cannot be used for linkage. By eliminating those singletons, tail ids and head ids are 38k and 168k, respectively.

Pages in linkage. As our goal is to extract ids to perform data linkage on the product specification pages, we analyze the number of pairs of pages that are associated with the same id discovered. The head ids (38k) gave rise to 3.0M linkages, whereas the tail ones (168k) generated 1.4M linkages. Conversely, in terms of sources, linkages involving pairs of tail sources are many more than those with pairs of head sources: 2.3M versus 864k.

Figures 13(a) and 13(b) plot the evolution of the linkages, i.e., groups of pages sharing the same id, during the iterations, considering our classification of head and tail ids, head and tail sources. Notice that many tail sources are rapidly found in the first iterations.

To evaluate the recall of the RaF output linkage we have taken a random set of 1000 unique ids from the ground truth extracted by using microdata annotations (and disjoint from the set of seed ids) and we used them as keywords for searching the Dexter corpus. We computed all the unique pairs in every group of pages matching the same id. From the resulting set of pairs, we randomly picked a sample of 1500 pairs of pages. A manual inspection of all these pairs produced a ground truth of only 284 pairs of pages in linkage, i.e., pages actually showing as *main* product the same product. We looked for these pairs in the set of linkages computed at the end of the iterations and we found 187 pairs (65.8%) of the ground truth.

7.4 Id Conflict Resolution

We conclude this section presenting experiments on the last step of our pipeline, which aims to resolve conflicting ids. Running this step over the pairs of ids obtained after the completion of the iterations filtered out about 44% of the linkages: we reduced the 4.4M linkages produced by the iterative phase of our pipeline to 2.5M.

We compare the results of our conflicting ids resolution technique with different alternatives: *Baseline* – the linkage obtained without applying any conflict resolution, i.e., linkages are based only on the equality of the ids; *Cosine* – we also consider page contents, i.e., cosine similarity between set of words contained in the page.[9] Then, we also considered several other configurations that filter linkages of pages associated with the same ids by further analyzing the categories of the two pages: *Category* – we directly use the coarse categories bundled within the Dexter dataset (only linkages involving pages of the same category are considered valid); *Louvain GB* – we check whether our id resolution approach based on the clustering with the G_β index classifies the two ids in the same cluster; and finally, *Louvain GB & Category* – we consider the conjunction of two latter methods: a pair is consider valid if its pages belong to the same Dexter coarse category and to the same cluster as computed by our redundancy-driven approach.

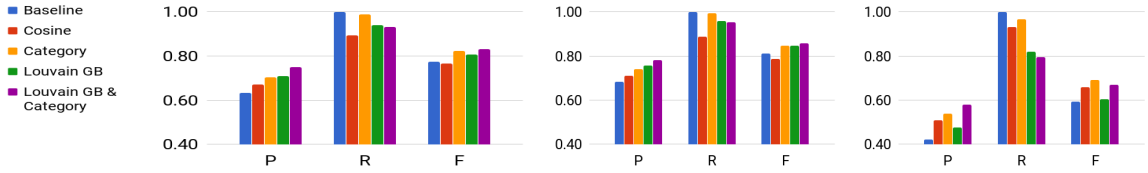
To compare these configurations, we have built a ground truth by manually inspecting the pages of 3000 linkages (that is, pairs of pages with identical ids) randomly chosen among those computed at the end of the iterations. We used the ground truth to count: the number of pairs linked correctly (true positives, tp), i.e., the number of pairs of pages actually showing the same product on both pages of the pair; the number of pairs linked incorrectly (false positives, fp); the number of pairs correctly not linked (true negatives, tn); and the number of missing pairs (false negatives, fn). Then we have computed precision ($P = \frac{tp}{tp+fp}$), recall ($R = \frac{tp}{tp+fn}$), and F-measure ($F = 2\frac{P \cdot R}{P+R}$).

The results of these experiments are shown in Figure 14. For every configuration, we report precision, recall and F-measure for ids classified as either head or tail ids. Observe that Louvain G_β has the highest precision over all ids and over head ids. Conversely, it loses precision with tail ids. This is expected, as our techniques are strongly based on the redundancy which, by definition, is less available for tail ids thus making the G_β index less reliable. The best results come from the combination of Louvain G_β and the input coarse categories of the dataset. The redundancy-driven approach of the Louvain G_β for clustering sources is able to correct many of the errors arising from the noise in the input dataset. Interestingly, the loss in recall is compensated by a gain in precision.

Analyzing the clusters computed by the Louvain G_β approach we have observed that many disjoint sources, i.e., not sharing any id, are correctly grouped in the same cluster. On average, 36% of the pairs of sources of every cluster are disjoint.

Influence of local ids. Our method for removing conflicting ids can be influenced by the presence of local ids that overlap among different sources. In presence of many common local ids (e.g., ID0001, ID0002, ...) recurring in several sources, their co-occurrences can induce our approach to create false communities. To evaluate this aspect, we have taken all the ids discovered at the end of the first phase of the pipeline for a pair of large categories (tv and monitor); then, in order to simulate the presence of local ids that overlap across sources, we have artificially injected to a given percentage of their sources a number of fake overlapping ids. We have run our conflicting ids resolution algorithm (in the *Louvain* G_β configuration) on these corrupted sources, by varying from 10% to 30% both the percentage of the corrupted sources and of their overlapping ids (our analysis of the Dexter dataset show that these scenarios is

[9] The linkage of a pair of pages associated with the same ids are discarded whenever their cosine similarity is lower than a threshold (set to 10% in our experiments) that we have empirically tuned to be good.



(a) All (head and tail) identifiers. (b) Head identifiers. (c) Tail identifiers.

Figure 14: Linkage Resolution.

rather pessimistic). To evaluate the performance of the resolution algorithm, we used the same metrics (precision, recall, F-measure) and the same ground truth of the previous experiment (but selecting only the 821 pairs involving the sources of this experiment). A significant drop of the F-measure (from 85% to 73%) is observable only when 30% of the sources contain at least 20% of overlapping fake ids. The complete results of this experiment are in Appendix A.6.

8 RELATED WORK

There has been a great number of previous studies in the area of record linkage, but traditional solutions mainly concentrate on the volume of data (see [6] for a survey), while RaF aims at processing a very large number of sources. In the context of big data linkage, Efthymiou et al. [14] propose a scalable blocking strategy that uses MapReduce to parallelize the comparisons of entities. They share with us the idea of exploiting the co-occurrence of entities, but their approach is designed for a small number of very large sources.

Many works have addressed the record linkage issue in the product domain. De Bakker et al. [10] presented a method to perform record linkage of products from different websites. Their approach first tries to match products based on the presence of ids in the title of pages. If the ids do not match, it tries to match products based on text similarity measures. A similar approach performs entity resolution of product pages based on enriched representations of their titles [15]. The enrichment is performed querying a search engine with the original title: tokens with high frequency in the title of the result pages are added to the original title. Pages are matched based on the cosine similarity of their enriched titles. Our solution, that exploits local regularities to infer extractors, generalizes these approaches, as we can extract the identifier from any leaf node of the page. Another drawback for these approaches is due to the restrictions imposed by search engines. Nguyen et al. [26] proposed a method to add product offers to a product catalog, containing a big set of products, and a product taxonomy. Their approach assumes schema alignment is solved and the linkage is based on the similarity of attributes. Agrawal and Ieong [1] studied the problem of determining the price history of a product, given a set of offers from different sources. They consider the existence of incorrect linkages, i.e., offers that do not correspond to the same product, and propose a probabilistic framework to come up with the correct price history of the product and a correction of erroneous matches. Similarly, Kannan et al. [21] consider the problem of linking product text snippets to the structured data of a large product catalog. These proposals address the variety issue of big data integration, but they require a reliable, structured head source.

Talaika et al. presented a method, IBEX, to collect product ids from Web pages and to associate those ids with their product names [29]. To identify the ids, they use regular expressions for pre-defined types of ids, such as, GTIN for products, ISBN for books, CAS for chemical substances. In contrast, we do not rely on any predefined pattern, which have a limited coverage, as shown by our experimental analysis. An entity resolution approach for products from e-shops is proposed in [20]: the authors assume the product name has been extracted and develop a solution based on a weighted combination of different similarity measures (e.g., Levenshtein and Jaccard) to match the product names. Petrovski et al. [27] developed a pipeline to integrate products from websites that use microdata. As they rely on the annotated attributes they follow a traditional data integration pipeline, where entity resolution (performed by means of a genetic programming approach) is done after schema alignment. The main limitation of this approach is due to the necessity of microdata annotations.

Gulhane ed alt. [17] proposed a system to exploit content redundancy over templated websites by taking as input a few annotated examples. However their techniques target a dataset composed of a handful of head sources from *vertical* domains, such as restaurants and bibliography, whose heterogeneity is significantly less pronounced than in our setting.

Our work is related also to projects that aim at building and enriching large knowledge graphs with web data, such as, Knowledge Vault [11] and YAGO [19]. However these projects follow a *closed Information Extraction* model: they identify only entities that are present in a reference (head) source (Freebase for Knowledge Vault, Wikipedia for YAGO). Our approach differs because one of our major goals is to include tails entities of tail sources, as well.

9 CONCLUSIONS AND FUTURE WORK

We have presented the RaF approach to big data linkage for product specification pages. RaF leverages the opportunities offered by the redundancy of information to address typical big data challenges, not only for the volume of data but also for the variety of the sources and of the entities involved. Our approach does not rely on any a priori knowledge, such as product ids patterns, which cannot work at web scale, but exploits natural, statistical, structural and semantic properties that hold in the product domain. As future work, we plan to solve the big data schema alignment and data fusion problems on product specifications, given the linkages computed by our techniques in this paper, in a holistic way.

REFERENCES

[1] Rakesh Agrawal and Samuel Ieong. 2012. Aggregating web offers to determine product prices. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 435–443.

[2] Arvind Arasu and Hector Garcia-Molina. 2003. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 337–348.

[3] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. 2008. Supporting the automatic construction of entity aware search engines. In *Proceedings of the 10th ACM workshop on Web information and data management*. ACM, 149–156.

[4] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[5] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. 2013. Extraction and integration of partially overlapping web sources. *Proceedings of the VLDB Endowment* 6, 10 (2013), 805–816.

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

[7] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. 2013. A framework for learning web wrappers from the crowd. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 261–272.

[8] Nilesh Dalvi, Philip Bohannon, and Fei Sha. 2009. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 335–348.

[9] Nilesh Dalvi, Ashwin Machanavajjhala, and Bo Pang. 2012. An analysis of structured data on the web. *Proceedings of the VLDB Endowment* 5, 7 (2012), 680–691.

[10] Marnix de Bakker, Flavius Frasincar, and Damir Vandic. 2013. A hybrid model words-driven approach for web product duplicate detection. In *International Conference on Advanced Information Systems Engineering*. Springer, 149–161.

[11] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 601–610.

[12] Xin Luna Dong. 2016. How Far Are We from Collecting the Knowledge in the World?. In *Keynote at 19th International Workshop on Web and Databases*. ACM.

[13] Xin Luna Dong and Divesh Srivastava. 2015. *Big data integration*. Vol. 7. Morgan & Claypool Publishers. 1–198 pages.

[14] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. 2017. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Information Systems* 65 (2017), 137–157.

[15] Vishrawas Gopalakrishnan, Suresh Parthasarathy Iyengar, Amit Madaan, Rajeev Rastogi, and Srinivasan Sengamedu. 2012. Matching product titles using web-based enrichment. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 605–614.

[16] Pankaj Gulhane, Amit Madaan, Rupesh Mehta, Jeyashankher Ramamirtham, Rajeev Rastogi, Sandeep Satpal, Srinivasan H Sengamedu, Ashwin Tengli, and Charu Tiwari. 2011. Web-scale information extraction with vertex. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 1209–1220.

[17] Pankaj Gulhane, Rajeev Rastogi, Srinivasan H. Sengamedu, and Ashwin Tengli. 2010. Exploiting Content Redundancy for Web Information Extraction. *Proc. VLDB Endow.* 3, 1-2 (Sept. 2010), 578–587.

[18] Xiangnan He, Ming Gao, Min-Yen Kan, and Dingxian Wang. 2017. Birank: Towards ranking on bipartite graphs. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2017), 57–71.

[19] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence* 194 (2013), 28–61.

[20] Andrea Horch, Holger Kett, and Anette Weisbecker. 2016. Matching product offers of e-shops. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 248–259.

[21] Anitha Kannan, Inmar E Givoni, Rakesh Agrawal, and Ariel Fuxman. 2011. Matching unstructured product offers to structured product specifications. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 404–412.

[22] Jon M Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)* 46, 5 (1999), 604–632.

[23] Hanna Köpcke, Andreas Thor, Stefan Thomas, and Erhard Rahm. 2012. Tailoring entity resolution for matching product offers. In *Proceedings of the 15th International Conference on Extending Database Technology*. ACM, 545–550.

[24] Robert Meusel, Peter Mika, and Roi Blanco. 2014. Focused crawling for structured data. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, 1039–1048.

[25] Robert Meusel, Petar Petrovski, and Christian Bizer. 2014. The webdatacommons microdata, rdfa and microformat dataset series. In *International Semantic Web Conference*. Springer, 277–292.

[26] Hoa Nguyen, Ariel Fuxman, Stelios Paparizos, Juliana Freire, and Rakesh Agrawal. 2011. Synthesizing products for online catalogs. *Proceedings of the VLDB Endowment* 4, 7 (2011), 409–418.

[27] Petar Petrovski, Volha Bryl, and Christian Bizer. 2014. Integrating product data from websites offering microdata markup. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 1299–1304.

[28] Disheng Qiu, Luciano Barbosa, Xin Luna Dong, Yanyan Shen, and Divesh Srivastava. 2015. Dexter: Large-scale discovery and extraction of product specifications on the web. *Proceedings of the VLDB Endowment* 8, 13 (2015), 2194–2205.

[29] Aliaksandr Talaika, Joanna Biega, Antoine Amarilli, and Fabian M Suchanek. 2015. IBEX: harvesting entities from the web using unique identifiers. In *Proceedings of the 18th International Workshop on Web and Databases*. ACM, 13–19.

A APPENDIX

A.1 The DEXTER dataset

The DEXTER contains $1.9M$ pages from 10 product categories (camera, cutlery, headphones, monitor, notebook, shoes, software, sunglasses, toilets, tv). Pages were collected from $3.5k$ websites; the crawler grouped them in 7,145 sources, corresponding to the local categories exposed by the websites.

Figure 15 plots the size (number of pages) of the sources, in non-increasing order of source size. The vertical red line indicates the median of the curve (the sum of the sizes of the sources on its left equals to the sum of the sizes of the sources on the right), which we consider the ideal border between head (large) and tail (small) sources. It occurs at 113: the vast majority of sources belong to the tail.

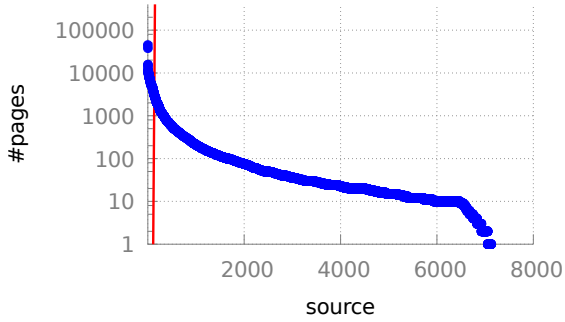


Figure 15: Size (number of pages) of the sources in the DEXTER dataset, in non-increasing order of the size of sources.

Table 1 details the percentage of head sources for the 10 product categories of the DEXTER dataset, together with number of sources and pages.

A.2 At the Web scale patterns are infeasible

We rely on the DEXTER dataset to concretely illustrate this problem. Table 2 reports the distributions of the most frequent patterns for the ids from the seed set (left) and from the output of a complete run of our pipeline (right). To describe the patterns, we use a simplified form of regular expressions: a matches with a single ASCII alphabetic character in [A-Z, a-z], d indicates a single digit in [0-9], and the number in curly braces indicates the repetition count. So,

category	#sources	#pages	%head
toilets	102	78,608	2%
sunglasses	551	233,959	5%
shoes	189	116,848	5%
tv	998	137,077	3%
cutlery	588	155,258	2%
notebook	847	289,879	3%
headphone	559	195,508	2%
camera	1,478	351,418	3%
monitor	1,175	195,542	2%
software	658	139,357	2%

Table 1: Sources in the DEXTER dataset.

Seed Set		Pipeline output	
Pattern	ids (%)	Pattern	ids(%)
d{8}	22.37%	d{5}	6.62%
d{7}	14.38%	d{6}	4.51%
d{6}	6.04%	a{7}	4.07%
d{15}	5.81%	a{8}	3.93%
d{10}	3.60%	a{6}	3.83%
d{9}	3.10%	a{9}	3.36%
GTIN-13	3.02%	GTIN-12	3.26%
a{1}n{14}	2.94%	a{5}	2.89%
GTIN-12	2.89%	a{10}	2.73%
GTIN-8	2.64%	a{11}	2.03%
other patterns	33.21%	other patterns	62.78%

Table 2: Distribution of patterns for the ids in the Seed Set and in the output of the RAF pipeline from the DEXTER dataset.

for example, the id 'abcd123e' would be considered as obeying to the pattern `a{4}d{3}a{1}`. After a basic normalization on the extracted token values, the seed set contains ids matching with 930 different patterns out of 33,281 values: the frequency of the most adopted pattern (a sequence of 8 digits) is less than 23%. In the set of ids produced by a complete run of our pipeline, we counted 20,020 different patterns; the most frequent one (5 digits) occurring just 6.62%.

A.3 Distribution of tokens

Figure 16 plots the distribution of the average number of tokens returned by the extractors computed in a complete run of our pipeline. The power-law distribution makes clear that a good part of the extractors return regions containing several tokens. Less than 15% of the extractors return a single token for at least 50% of the pages in the source.

A.4 Distribution of identifiers

Figure 17 plots the distribution of the number of ids per page. About 52% of the pages (for which we found an id) have more than one id. About 20% of the pages have more than four ids.

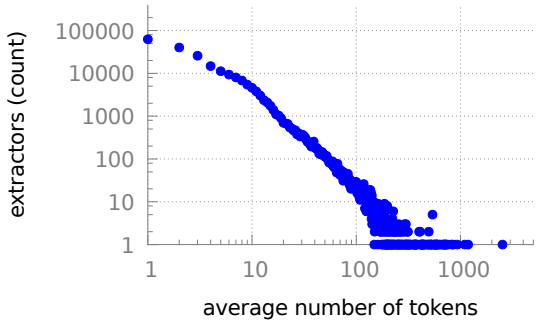


Figure 16: Distribution of the average number of tokens in the regions returned by the extractors.

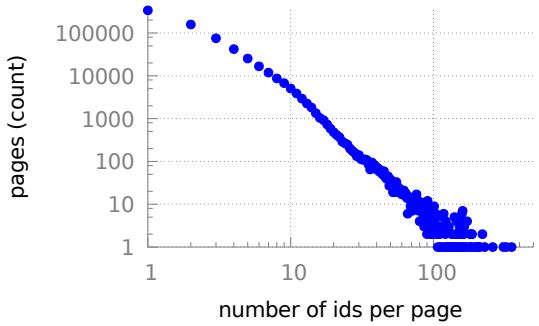


Figure 17: Distribution of the number of ids per per page.

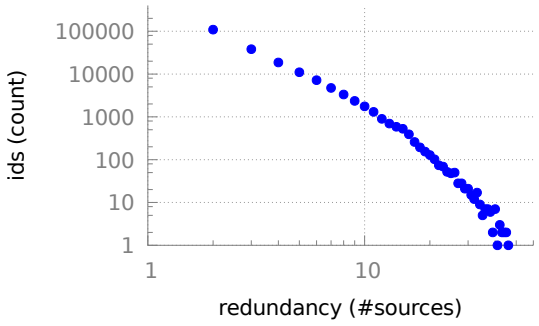


Figure 18: Redundancy of the ids.

Figure 18 shows the redundancy (wrt to the number of sources) of the ids produced by the first phase of the RAF pipeline.

A.5 Source similarities

Table 3 reports the G_β index and the Jaccard-index (within parenthesis) for the sources of Example 6.1.

	s_1	s_2	s_3	s_4	s_5	s_6
s_1	-	.22 (.4)	.28 (.5)	.32 (.75)	.26 (.4)	.07 (0)
s_2	.22 (.4)	-	.19 (.4)	.21 (.33)	.17 (.14)	.14 (.4)
s_3	.28 (.5)	.19 (.4)	-	.38 (.25)	.19 (.17)	.08 (0)
s_4	.32 (.75)	.21 (.33)	.38 (.25)	-	.25 (.6)	.06 (0)
s_5	.26 (.4)	.17 (.14)	.19 (.17)	.25 (.6)	-	.02 (0)
s_6	.07 (0)	.14 (.4)	.08 (0)	.06 (0)	.02 (0)	-

Table 3: G_β (Jaccard) index between sources of Example 6.1.

A.6 Influence of local ids

Figure 19 shows the detailed results of the experiment to evaluate the influence of local ids on our id conflict resolution technique described in Section 7.4. Using the original sources we obtained: P=.85, R=.85, F=.85.

	10% overlap			20% overlap			30% overlap		
	P	R	F	P	R	F	P	R	F
10% sources	.84	.87	.85	.85	.85	.85	.85	.85	.85
20% sources	.80	.84	.82	.85	.85	.85	.80	.68	.73
30% sources	.81	.81	.81	.80	.76	.78	.79	.68	.73

Figure 19: Influence of local ids on the conflicting resolution.

A.7 Performance and running times

All the experiments have been run on a single machine with 32 cores and 128GB ram. To complete the iterations, the system run for 336 hours. Every iteration processed 100k pages (remember that the number of pages is bounded) from 200 sources (on average). Figure 20 reports detailed running times for the main tasks performed in the pipeline. Observe that most of the costs are related to pages processing during the iterative phase and it is worth observing that most of the performed tasks that can be largely parallelized and scaled on several machines.

Task	Running Time	Percentage
extractors generation	2,310 min.	11.2%
extraction	15,400 min.	74.4%
filtering	1,540 min.	7.4%
conflicting id resolution	1,440 min.	7.0%
Total	14.37 days	

Figure 20: Running times for a complete execution.