IBEX: Harvesting Entities from the Web Using Unique Identifiers

Aliaksandr Talaika[1], Joanna Biega[1], Antoine Amarilli[2], Fabian M. Suchanek[2]
[1] Max Planck Institute for Informatics, Germany
[2] Télécom ParisTech, France

ABSTRACT

In this paper we study the prevalence of unique entity identifiers on the Web. These are, e.g., ISBNs (for books), GTINs (for commercial products), DOIs (for documents), email addresses, and others. We show how these identifiers can be harvested systematically from Web pages, and how they can be associated with human-readable names for the entities at large scale.

Starting with a simple extraction of identifiers and names from Web pages, we show how we can use the properties of unique identifiers to filter out noise and clean up the extraction result on the entire corpus. The end result is a database of millions of uniquely identified entities of different types, with an accuracy of 73–96% and a very high coverage compared to existing knowledge bases. We use this database to compute novel statistics on the presence of products, people, and other entities on the Web.

1. INTRODUCTION

Unique ids. The Web is a vast resource of named entities, such as commercial products, people, books, and organizations. The focus of this paper is on the entities that have unique *ids*. An id is any string or number that distinguishes the entity from other entities in a globally unique way. For example, commercial products have ids in the form of GTINs, printed below the bar code on the package or item. They also frequently appear on the Web. Figure 1 shows an excerpt from a Web page about a commercial product. The GTIN (8806085725072) appears at the bottom right.

Figure 1: A Web page snippet about a product

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. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
WebDB 2015, May 31, 2015, Melbourne, VIC, Australia.
ACM 978-1-4503-3627-7/15/05...$15.00
http://dx.doi.org/10.1145/2767109.2767116

Not only commercial products have ids; a surprisingly large portion of other entities also do. Companies have tax identification numbers; books have ISBNs; documents have document identifiers; chemical substances have CAS registry numbers, and so on. Web pages that talk about these entities often mention their ids.

Goal. Our goal is to harvest these ids together with the entity names at large scale from the Web. That is, we want to build a database that contains, for example, the entry ⟨8806085725072, Samsung Galaxy S4⟩. Ids and entity names can be extracted from the Web using named entity recognition methods (NER). Still, matching an id to its entity name is far from trivial. First, Web pages usually contain dozens of entity names, thus we must associate the proper entity name with each id. For example, we must find that the correct name for the id "8806085725072" is "Samsung Galaxy S4" – and not "Samsung", "VAT", or "GT-I9295ZAADBT". Moreover, some Web pages contain several ids and several entity names at the same time, so we must be able to correctly match the corresponding pairs of ids and names. Finally, if we want to find entity ids and names at Web scale, we need an approach that is both fast and resilient. It must run on hundreds of millions of Web pages and accept arbitrary pages, with possibly erroneous content, broken structure, or noisy information. This makes it impossible to rely on wrapper induction or any predefined or learnable DOM tree structure.

Contribution. In this paper, we show how to systematically collect unique ids from Web pages, and how to associate each id with the correct entity name. We first extract ids and candidate names from each Web page using vanilla NER methods. Then, we make use of the inherent characteristics of unique identifiers to filter the name candidates so as to keep only the correct entity names. Our method is scalable, fast, and resilient enough to run on arbitrary Web pages.

We extract millions of distinct entities from the Web with an accuracy of 73% to 96% depending on the entity type, yielding a database of entity ids and names together with the Web pages where they appear. The crucial advantage of this database is that every entity is guaranteed to be *unique*, which allows us, for instance, to count *distinct* entities without being biased by duplicates. We can thus perform a detailed study of entities that exist on the Web, including identification of Web sites that are hubs for books or documents, building statistics about frequent names of people, tracing the flow of products between different countries, or identifying top producers. In other words, we show not only how entities with unique ids can be extracted from the Web, but also how they are distributed on the Web and in the world. Our contributions are:

- The paradigm of *id-based entity extraction* (IBEX), harvesting entity ids with their names at large scale from the Web.
- A database of unique entities with millions of objects and an accuracy of 73–96%.
- Detailed analyses about the distribution of these objects.

2. RELATED WORK

2.1 Information Extraction (IE)

Named Entity Recognition. Named entity extraction (NER) is the task of recognizing and categorizing real-world entities in textual resources [32, 11]. We rely on NER to spot ids and candidate names in Web pages. However, our focus is rather on associating the correct name with each id, choosing from several extracted name candidates. While NER can spot entity names, it cannot determine the correspondence between ids and names if the page contains several of those. Our approach aims at solving this problem.

Wrapper induction. Wrapper induction [42, 13, 12, 16, 43, 14, 15, 8, 20] learns the structure of a Web page and produces a so-called *wrapper*, which can then be applied to extract information from other Web pages of the same form. These approaches exploit the fact that large Web sites are typically generated from a source by the help of templates. In our setting, we cannot make such an assumption, as we target arbitrary Web pages of arbitrary sites. We may have only a handful of pages for each site, plus a large number of pages that do not belong to any large site. The DIADEM project [17] can deal with such a variety, but is domain-specific and targets the deep Web, while we target the surface Web.

Structured IE. A large suite of approaches (e.g., [2, 23, 24, 45]) aims at extracting information from structured sources – using visual clues [33, 6], the DOM structure [41], or the schema [4]. Unlike these, our method does not assume any particular structure in Web pages. It does not require the pages to resemble each other, it does not need training data, and it does not assume any given schema. Rather, it works on both structured and unstructured sources across arbitrary sites and domains.

Product extraction. One of the applications of our work is the extraction of commercial products. Previous work on product extraction focused on matching product offers to products and their attributes [21, 22]. The work by [22] succinctly mentions also manufacturer extraction from product titles. Supervised learning approaches have been proposed to update product catalogues using new offers [27], or to determine product prices [1]. These approaches, however, build on an existing catalog of products, while our goal is the creation of such a catalog.

Other projects [31, 30, 19] handle product attribute extraction. [29] gives a method to discover product information regions on Web pages. While these approaches share our goal of extracting product data from Web pages, they do not target the creation of a global database of unique products.

Knowledge bases. Recent work has led to the automated creation of large knowledge bases [34, 3, 10, 5]. These contain many popular and important entities but do not aim at being exhaustive. DBpedia, e.g., contains "only" a few ten thousand instances of the class *Product*, most of them being named ships. This is because these KBs focus more on popular entities than on the long tail. Our goal, in contrast, is systematic collection of products from the Web.

Web-scale databases. Several projects [38, 18, 7, 28, 44] construct a queryable database of Web objects. While we share this goal, we use ids to achieve it. This has two advantages. First, we can run extraction even on the Web pages with poor structure, or no structure at all. Second, we build a database of *unique* entities, where each entity is guaranteed to appear at most once.

Entity databases. There are several databases of unique Web entities. The UPC Database (`http://www.upcdatabase.com`) contains 1.6M ids of **commercial products**, but is not available for download. Other id databases are GTIN13.com (`http://gtin13.com`) and Smoopa (`http://www.smoopa.com/`), but no information on their content is freely available. The International DOI Foundation (`http://www.doi.org/`) gives identifiers to **text documents**. There are 84M document ids, but the foundation does not provide a search capability across all documents, and the data cannot be downloaded. The Chemical Abstracts Service (CAS) (`http://www.cas.org`) maintains a registry of more than 71M **organic and inorganic substances**. However, this data is not available for free. The Common Chemistry Website (`http://www.commonchemistry.org/`) provides publicly available data, but it covers only 7,900 substances. As for databases about **people**, there are social networks that collect personal data and commercial Websites that scrape the Web for it (e.g. `http://www.yasni.com/`). Neither of these provides downloadable datasets.

Table 1: Id types (* indicates pseudo-ids)

Id type	Entities	Id type	Entities
ISBN	Books	ISAN	Audiovisual material
GTIN	Products	Pub#	US Patents
CAS	Chemicals	ILU	Containers
DOI	Documents	MESH	Chemicals
VATIN	Companies	OMIM	Diseases
BIC	Banks	ICD-10	Diseases
ISIN	Stocks	Email	People/organizations*
VIN	Vehicles	IBAN	People/organizations*
GRID	Digital recordings	Phone	People/organizations*

3. PROBLEM STATEMENT

Our goal is to build a database of unique entity ids from Web pages, and to associate a human-readable name with each of those. We now formally define our notion of ids and the problem we study.

Ids. For us, an *entity* is any real-world object such as a person, a book, a product model, or a shipping container. An *id* is a string that is used as an identifier for an entity.

There are different *types* of ids, i.e., groups of ids that have the same syntax and refer to entities of the same domain. For example, ISBNs are sequences of 10-13 digits that identify books. CAS numbers are sequences of 8 digits that identify chemicals. We only assume that: (1.) no two entities can have the same id (e.g., one ISBN cannot refer to two books), and (2.) one entity can only have at most one id of each type (e.g., a book can only have one ISBN, but it can have also a GTIN, since books are also products).

For some id types, assumption (2.) does not hold. For example, every personal email address belongs to one person, but one person can have several email addresses. We call such identifiers *pseudo-ids*. Our approach can also collect entities by their pseudo-ids, but it cannot guarantee the uniqueness of the collected entities in this case: for instance, the same person may appear in the constructed database multiple times under their various email addresses.

We assume that every entity has one or several *names*. A name is a human-readable string that identifies the entity intuitively.

Examples. The notion of ids and id types is a general one, so our approach can be applied to a variety of domains. Table 1 presents examples of id types. They cover both entities that are intrinsically Web-based, such as Web documents, but also a large number of real-world entities, such as chemicals, commercial products, vehicles, books, or magazines. The experiments in this paper will focus on the following id types, exemplified in Table 2:

- *Global trade item numbers* (GTINs) are identifiers for commercial products.
- *CAS numbers* are identifiers for chemical substances.
- *Digital object identifiers* (DOIs) identify electronic documents.
- *email addresses* as pseudo-ids for people.

Table 2: Examples for ids and entity names

Id type	Id	Entity name
GTIN	00068888883955	Pyramid PA305 100w Rack Mount Amplifier with Microphone Mixer
CAS	78123-16-7	N-benzyl-2-(2-methyl-1H-indol-3-yl)acetohydrazide
DOI	10.1037/a0024143	Cognitive niches: An ecological model of strategy selection.
Email	widom@cs.stanford.edu	Dr. Jennifer Widom

Problem statement. The input to our method is a set of *Web pages* obtained from a Web crawl. In addition, we are given an id type t and two NER modules for t: the *id validator* f_t^{id} and the *name finder* f_t^{name}. The id validator is a function that takes as input a string and returns *true* iff the string is an id of type t. The name finder is a function that, given an id of type t and a string, extracts possible candidate names for that id from the string. These can be single tokens or multi-words.

The output of our method is an *entity database* for type t, containing ids of type t with their associated names. In addition, we store the URLs of the Web pages where the entities were found.

4. APPROACH

4.1 Method Description

This section presents the three phases of our approach, which are implemented in the Map-Reduce framework. Section 5 gives details about Phase 1.

Phase 1. The first phase extracts ids and name candidates. Let us consider each page $p \in \mathscr{P}$. We first split the page into *records* $r_1, \ldots, r_n$, where each record is a region of p that contains exactly one id. For each record r_i, we extract the id id_i and all name candidates $name_{i,j}$, $j = 1, \ldots, m_i$ using f_t^{id} and f_t^{name} (see Section 5 for details). To account for differences in writing, we normalize all name candidates by upper-casing them, and by retaining only letters (alphanumeric characters for CAS). For each name candidate, we compute a score that indicates its likelihood of being the correct name for id_i. The result of this process is a table of the following form for each record r_i:

$$R1_i^p := \{\langle id_i, name_{i,j}, score_{i,j}, url(p)\rangle \mid j = 1, \ldots, m_i\}$$

Its rows contain the id, a name candidate for this id, a score for this name candidate, and the URL of the page. Note that the same name may occur multiple times in $R1_i^p$ (with the same score or with different scores) if the same name was extracted multiple times in r_i. The output of the first phase is then the union of these tables:

$$R1 := \bigcup_{p \in \mathscr{P}, r_i \in p} R1_i^p$$

Phase 2. The previous phase extracted all possible name candidates for all id occurrences. Hence, $R1$ is very noisy and contains many wrong candidates. For instance, some name candidates are not entity names, but rather descriptive elements, such as "Price", "see also", or "plastic". This problem could be reduced by using a better entity tagger f_t^{name}, but will ultimately always appear.

Since our corpus is large, we expect such non-specific names to appear uniformly over several ids, whereas the correct names will accumulate on one id. As an example, Figure 2 shows three real distributions of names across ids from our experiments (Section 6). The chemical name "amphetamine" appears 120 times with 10 different ids. But it appears 99 times for one of these ids (which is the correct id of amphetamine). The non-specific names "plastics" and "propionate" appear more uniformly with different ids.

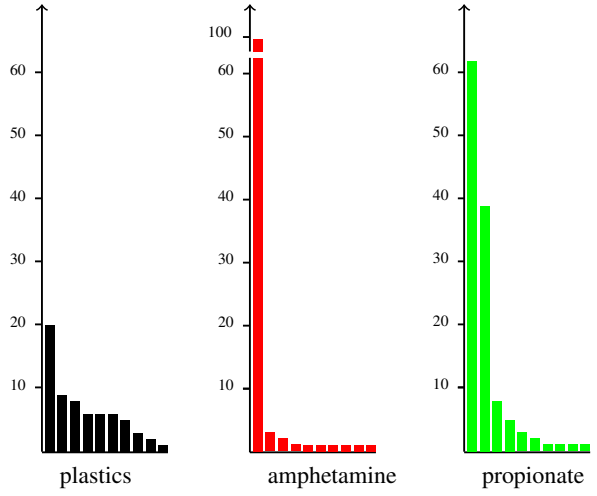

Figure 2: Frequency of occurrence of a name per id. The i^{th} bar indicates how many times the name occurs with its i^{th} id.

Our goal in Phase 2 is to identify names such as "amphetamine" that show a clear preference for one id. Formally, we count for each name n how often it appears with the id id in $R1$:

$$freq_n(id) := \left|\{\langle id', n', s', u'\rangle \in R1 \mid n' = n,\ id' = id\}\right|$$

Now, we attempt to identify names n whose distribution $freq_n$ shows a clear outlier. We experimented with several outlier detection methods, see our technical report [35]. In the end, the following technique worked best. Let id_n^1 and id_n^2 be the ids with the highest and second highest value for $freq_n(\cdot)$, breaking ties arbitrarily. The distribution $freq_n(\cdot)$ is said to have an outlier if id_n^1 appears in more than 30% of the cases, and at least 3 times more often than id_n^2. If a name appears with only one id, it is always considered an outlier.

$$freq_n(id_n^1) > 0.3 \times \textstyle\sum_i freq_n(i) \qquad freq_n(id_n^1) > 3 \times freq_n(id_n^2)$$

This technique is robust enough to work for all id types that we considered. Now, Phase 2 removes all names with no clear outlier:

$$R2 := \{\langle id_n^1, n, s, u\rangle \mid \langle id_n^1, n, s, u\rangle \in R1,\ n \text{ has outlier } id_n^1\}$$

The resulting table $R2$ contains names that are specific to one id.

Phase 3. While we have now filtered out the insufficiently specific names, entities may still have several names, some of which may be wrong. To remove less likely names, we pick, for each id, the name that appears most often. If the most frequent candidate names have the same frequency, we take the one with the highest score (ties on the score are resolved arbitrarily):

$$R3' := \{\langle id, n, s, u\rangle \mid \langle id, n, s, u\rangle \in R2,\ freq_n(id) = \max_{n'} freq_{n'}(id)\}$$

$$R3 := \{\langle id, n, u\rangle \mid \langle id, n, s, u\rangle \in R3',\ s \text{ maximal for this } id\}$$

The result $R3$ of Phase 3 is our final entity database: for every id, R_3 contains one name and the URLs of all pages where it was found.

4.2 Coverage

If we wanted to build a comprehensive database of all entity ids on the Web, we would have to parse all existing Web pages. However, in practice, we can only access a subset of pages that was found through crawling. Hence, our dataset is always incomplete. It may happen, for example, that we see the same entity id over and over again in our crawl, instead of seeing new ids that we could add to our collection. In the worst case, we could crawl half of the Web, but see only a small fraction of the distinct entities that exist.

Fortunately, this is unlikely to happen if we assume the crawl is sufficiently random. To show this, we focus on the set $\mathscr{W}$ of pages on the entire Web that mention at least one entity of type t, and we consider $\mathscr{P}' := \mathscr{P} \cap \mathscr{W}$ the set of the input pages $\mathscr{P}$ that mention some entity of type t. We write α for $|\mathscr{P}'|/|\mathscr{W}|$, and assume that $\mathscr{P}'$ is a subset of $\alpha|\mathscr{W}|$ pages drawn uniformly at random in $\mathscr{W}$.

We call $\mathscr{E}$ the set of all different entities of type t appearing in $\mathscr{W}$, and $\mathscr{E}' \subseteq \mathscr{E}$ those appearing in $\mathscr{P}'$. Intuitively, $\mathscr{E}'$ are the entities of $\mathscr{E}$ that we can extract from our sample. We assume that some entity of $\mathscr{E}$ occurs in strictly more than one page of $\mathscr{P}$, and claim:

THEOREM 1. *For any fixed $0 < \alpha < 1$, the expected value of $|\mathscr{E}'|$ over draws of $\mathscr{P}'$ is strictly greater than $\alpha|\mathscr{E}|$.*

In other words, if we crawl a random subset of 50% of all Web pages that mention entities, then we can expect to see more than 50% of all entities in our sample. The result is proven in our technical report [35]. Intuitively, we show that, for any entity $e \in \mathscr{E}$, the probability of obtaining e is $\geq \alpha$, which can be seen by choosing one arbitrary page p where e occurs and noticing that the probability of drawing p is at least α.

5. IMPLEMENTATION

We now discuss the detailed implementation of Phase 1 of our method from Section 4, where we parse a Web page to extract records, ids, and name candidates.

5.1 Parsing Web Pages

Requirements. The Web pages that we consider are written in HTML, which in theory can be parsed to a DOM tree that represents the structure of the page. In practice, a large number of HTML documents on the Web are ill-formed. In addition, the structure of the DOM tree does not necessarily correspond to the page structure as seen by a human. For example, if a page contains several H1 tags, then a human sees several sections. The DOM tree, however, contains just one parent node with alternating H1-nodes and text-nodes as children. If the page discusses different entities, then it is likely that each entity falls within one H1-dominated block. The DOM tree, however, does not make this directly apparent. More generally, many websites use HTML markup in a non-semantic way which leaves little information in the DOM tree about the relationship between the elements in the page.

Our method is inspired by MDR [41], but applies more generally to Web pages without tabular structures. Most importantly, it is robust enough to work on Web pages without a proper DOM tree, and simple enough to run at Web scale on arbitrary input.

Frame trees. We segment the HTML page p into a *frame tree* (the name of which is not related to frames in HTML documents). A frame tree looks like a DOM tree, but contains additional nodes for blocks that are introduced by *separators*. A separator is a tag that starts a new paragraph, such as H1,...,H6, HR, BR, sequences of BR, and P. Any opening HTML tag starts a new frame, and any matching closing tag closes the current frame. If there is no closing tag in a scope of a parent frame, then the current frame is closed when the parent frame is closed. Additionally, every separator starts a new frame that ends at the next occurrence of a separator of equal or higher weight, or at the end of a parent frame.

```
function parse(HTML document d, parent tag t):
// d is passed by reference so recursive calls may modify it
FrameTree result ← ⟨tag: t, content: [ ]⟩
if t is self-closing then return(result)
while |d| > 0
    if d starts with tag t'
        if t' is closing
            if t' closes t
                remove t' from d
                break
            end if
            if t' ⊁ t
                remove t' from d
                continue
            end if
            break
        end if
        if t ⊁ t' then break
        remove t' from d
        if t' is separator
            f ← [ parse(d, t') ]
            f.appendAll(parse(d, t'*).content)
            result.content.append(⟨tag: t'*, content: f⟩)
        else
            result.content.append(parse(d, t'))
        end if
    else
        read and remove text s from d
        if s is not whitespace then
result.content.append(s)
    end if
end while
return result
```

Algorithm 1: Building a frame tree

Algorithm 1 parses an HTML document recursively into a frame tree. It is called initially with a dummy parent tag DOC, and relies on a containment relation $\succ$ on tags, so that $t \succ t'$ if a tag t can contain a tag t'. For example, DOC$\succ$HTML$\succ$BODY$\succ$DIV. This order can be derived from the HTML grammar. In addition, we introduce an artificial tag t^* for every separator tag t. For example, we introduce the tag H1*, which will be the label for a frame that consists of a H1-header and the following text. We extend $\succ$ to cover also these tags. For example, a H1-frame can contain H2-frames: H1$^*\succ$H2*. The algorithm yields a tree of frames, whose leaf nodes are text nodes. We call these nodes *text frames*. Algorithm 1 will produce frame trees even if the page is not fully standards-compliant. Our technical report [35] shows an example for the algorithm.

5.2 Extracting Records

Once Algorithm 1 has produced a frame tree for an input Web page, we must find *records* in this tree. A record r of type t in a page $p \in \mathscr{P}$ is the largest subtree rooted at some node of the frame tree of p that contains a single id of type t, using f_t^{id} to identify which text frames of the frame tree are ids. For this, a text frame must correspond exactly to one id, with no surrounding text. We call r a *detail record* if it is the only record of p; otherwise, it is a *free record*. Intuitively, detail records occur when the entire page deals with the entity, and free records typically belong to listings of entities, e.g., lists or tables, or free-floating descriptions of entities.

Table 3: Total number of items, accuracy, and recall after each phase

	GTIN items			CAS items			DOI items			Email items		
	Num.	Acc.	Rec.	Num.	Acc.	Rec.	Num.	Acc.	Rec.	Num.	Acc.	Rec.
Phase 1	3,929,312	38%±8%	60%	241,602	76%±6%	80%	1,167,810	52%±8%	50%	13,625,860	90%±6%	63%
Phase 2	2,550,703	76%±8%	48%	235,779	86%±5%	76%	1,038,950	68%±9%	45%	13,625,860	90%±6%	63%
Phase 3	2,550,703	82%±7%	50%	235,779	96%±3%	78%	1,038,950	73%±8%	47%	13,625,859	90%±6%	63%

Table 4: Richest sources for entities of various entity types

Product sources	Items
www2.loot.co.za	304,431
www.books-by-isbn.com	50,683
gtin13.com	26,834
en.wikipedia.org	21,873
www.buchhandel.de	18,264

Chemical sources	Items
www.chembuyersguide.com	129,211
www.chemnet.com	22,061
www.lookchem.com	12,354
www.seekchemicals.com	7,326
www.tradingchem.com	4,769

Document sources	Items
wwwtest.soils.org	20,635
www.plosone.org	19,261
www.citeulike.org	13,491
www.astm.org	10,020
bja.oxfordjournals.org	9,030

Once all records in a page have been identified, the function f_t^{name} is applied to all text subframes of each record. This yields a set of candidate names per record. We score each candidate by its negative distance from the id, and by taking into account the tags that surround the candidate (e.g., H1 or B). Details of how the components of this score (and alternative scores) perform can be found in our experimental evaluation in our technical report [35]. The ids together with their candidate names and scores make up the table $R1$ of our Phase 1.

5.3 Discussion

Our approach deals correctly with many common structures in HTML documents that refer to entities. If, for example, each row of an HTML table contains an id, then each row will become a record in Algorithm 1. Similarly, for HTML lists, if each item in an HTML list contains an id, then each item will become a record. This holds no matter whether the page uses the TABLE tag, the UL tag, or in fact any other tag.

If each row of a table corresponds to an entity, then the entity names will usually all be in the same column. Conversely, a column that always contains the same word is unlikely to contain an entity name. As our approach is generic, Phase 1 extracts all candidate names agnostically. Then, Phase 2 will remove globally frequent names. Thus, Phase 2 will have the same effect as discarding frequent words from table columns, just that it operates on a global table. Finally, Phase 3 will choose the most frequent name for an id. Thus, Phase 2 and Phase 3 together act like the TF-IDF mechanism in information retrieval.

6. EXPERIMENTS

Setup. We performed experiments on the English portions of the ClueWeb09 and ClueWeb12 Web crawl. In total, our 35 TB corpus contains 1.2 billion Web pages. We evaluated our approach for the GTIN, CAS, and DOI id types, as well as the pseudo-id *email*[1]. We used simple NER modules that basically use regular expressions (see our technical report [35] for details). The extraction process took 10 hours on a Hadoop cluster of 10 nodes, with the total capacity of 80 map-reduce tasks, amounting to an average of about 3,000 pages, or 100 MB, processed per second and per node.

Quality. To verify the quality of our name extraction, we produced a gold standard set of ids and entity names. For this purpose, we randomly sampled, for each type, 200 occurrences of ids in pages from our Web corpus. We annotated each id manually with its correct name in the page. Then, we compared the output of each phase to this gold standard. For the first two phases, as there are multiple name candidates per id, we pick one name at random for each id, to simulate a guess. Table 3 shows our results, providing Wilson intervals [9] to ensure their statistical significance. We considered each id, and compared the assigned name to the correct name from the gold standard. Accuracy is the proportion of correct names. Recall is the proportion of correct names that was correctly assigned.

As we can see, Phase 1 cannot find all entities that have an id. It also has a low accuracy, because it produces many name candidates, one of which is chosen at random for the evaluation. This is essentially what a naive **baseline** algorithm would do: it would extract all candidate names on a Web page, and assign one name at random to each id on that page. As we see, the performance is mediocre, with a accuracy of 38% for GTINs.

However, in our approach, Phase 2 and 3 filter out the erroneous names, increasing the accuracy up to 96%. The recall varies through the phases, since the set of candidate names per id shrinks, which may increase or decrease the recall. In the end, our method assigns the correct name for 83% of the products, 96% of the chemical substances, 73% of the documents, and 90% of the emails.

Quantity. As Table 3 shows, our database contains 2,550,703 products, 235,779 chemical substances, 1,038,950 documents, as well as 13,625,859 email addresses. 55.6% of our products are books. The other products include things as diverse as office supply items, DVDs, or USB cables. While email addresses are pseudo-ids, and we cannot guarantee uniqueness in this case, all other ids are unique. This means that our numbers provide a lower bound for the number of chemical substances, unique documents, books, and commercial products on the Web. Our database is also orders of magnitude larger than other public databases.

To estimate the coverage of our database, we compared our data to the YAGO KB [34], focusing on books, whose ids (ISBN codes) are known to YAGO. YAGO contains 11,271 books with an ISBN. Of these, 1,662 appear in our database. We assume that we missed some of the YAGO books because Wikipedia, the source of YAGO, is not necessarily entirely in our corpus; furthermore, our cleaning phases may remove correct book candidates. By contrast, our database contains 1.4 million books that are unknown to YAGO.

Extensions. In the technical report [35], we showcase extensions and applications of our approach. These include extending the extractors to other id types, extracting entity attributes (such as molecular formulae for chemical substances), or merging our database with the YAGO KB [34] as an input to the PATTY system [26], to discover more typed textual patterns.

[1]Since many people share the same name, we skipped Phase 2 for email addresses.

Table 5: Common person names

Family names		Given names		Full names	
Smith	84,376	John	238,446	John Smith	1,969
Johnson	55,277	David	207,931	David Smith	1,484
Brown	46,499	Michael	155,880	John Doe	1,371
Jones	45,322	Mark	117,755	Michael Smith	990
Williams	43,492	Robert	109,814	David Brown	899

Table 6: Top companies by prod.

Company	Prefix	Products
Bernat	0057355	1,116
Panasonic	0037988	929
Lion	0023032	927
Nikon	0018208	829

Table 7: Top email domains

Domain	Addresses
gmail.com	304,236
yahoo.com	290,292
hotmail.com	281,498
aol.com	259,769

7. ANALYSES

Our dataset is a huge resource of Web objects, which can give rise to different analyses – much in the spirit of Culturomics [25]. The following experiments illustrate this.

7.1 Resources

Our dataset can identify Internet domains that are particularly rich in different types of Web entities. Table 4 shows the best data sources that we could determine for email addresses, chemicals, and documents. This analysis could help steer information extraction approaches to target domains that are rich in the desired items.

We also computed the most frequent email providers occurring in the email addresses that we collected (Table 7). Addresses come mostly from Gmail, followed by Yahoo! and Hotmail, which are indeed the top three email providers, as determined by [36].

7.2 People

Common names. Our extraction found 13 million email addresses with an associated person name. Since email addresses are only *pseudo-ids*, we may not conclude that we found 13 million people. However, we can still identify the most common given names and family names on the Web (Table 5). The popular first names that we found correspond roughly to the frequent English names. For instance, our top 50 male given names cover 43 of the top 50 male given names of the US 1990 census data [37].

Gender. By intersecting our set of given names with the US census data about common female and male first names, we extended the analysis to find the gender of the people on the Web. For the 331 unisex names we attributed both genders proportionally to their gender priors, based on the name frequency statistics. We find that women are slightly under-represented: out of 11.6 million names in our database whose gender we could identify, only 47% were female. 1,990,290 names were not recognized as American names.

7.3 Commercial Products

Company names. The first 4-7 digits of the GTIN are a company identifier. Since there is no publicly available database to map these prefixes to company names, we did as follows. We assume that product titles often contain the company name. We grouped the GTINs by their 4-digit prefix and computed the word that was most frequent within a group, but infrequent outside the group. A manual analysis on a random sample of 80 products shows that this method can indeed identify the company name (or at least part of it) correctly in 83% of the cases, with a Wilson score interval of $\pm 7\%$. Table 6 shows the companies with the most products.

Table 8: Countries by production

Country	#Products	Country	GDP (trillion)
USA	1,024,219	USA	14.99 US$
UK	59,542	China	7.20 US$
Germany	26,949	Japan	5.87 US$
Japan	17,353	Germany	3.60 US$
France	12,845	France	2.77 US$

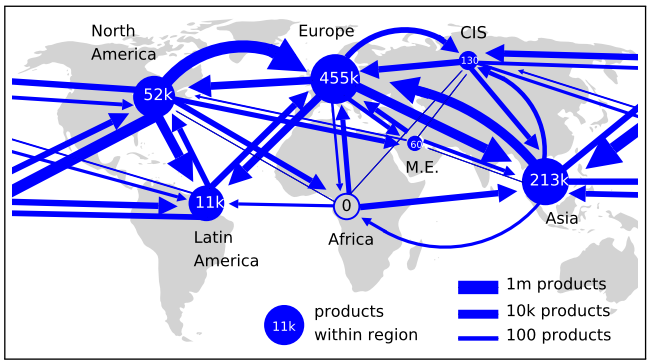

Figure 3: Intra- and inter-regional trade (log-scale)

Countries. The first 3 digits of a GTIN identify a country of production. For this analysis, we extracted the GTINs from the entire ClueWeb corpus (not just the English one). We then calculated the number of unique items produced in different countries (Table 8). This ranking has a remarkable overlap with the list of countries by GDP provided by the World Bank [39].

Global trade. While the GTIN indicates where a product was produced, the top level domain of a page where the product appears probably identifies a country where the product is sold. This allows us to trace product exports (Figure 7.3). The export patterns that we extracted are comparable to the true flow of products between the regions, as estimated by the WTO [40] (see our technical report [35] for details).

8. CONCLUSION

In this paper, we have shown how to systematically harvest entities with unique ids from the Web. By making use of the properties of ids, we could extract entity names with an accuracy of 73–96%, yielding a database of 13 million email addresses with their name, 235 thousand chemical substances, 1 million documents, 1.4 million books, and 1.1 million other commercial products. We believe that this dataset is the first public database of canonicalized items of this size. We have shown possible uses of this database by conducting a number of analyses, and expect that many more exciting experiments can be conducted with our data.

We believe that our methodology is applicable not just to the types of entities that we worked with in this paper, but also to a broad range of other entities, which will make the Web ever more semantic and more useful. All data and analyses are available at: `http://resources.mpi-inf.mpg.de/d5/ibex`.

Acknowledgements. We are grateful to Pierre Senellart for his useful feedback. This work has been partly funded by the Télécom ParisTech Research Chair on Big Data and Market Insights, as well as by the Labex DigiCosme (project ANR-11-LABEX-0045-DIGICOSME) operated by the French ANR as part of the program "Investissement d'Avenir" Idex Paris-Saclay (ANR-11-IDEX-0003-02).

9. REFERENCES

[1] R. Agrawal and S. Ieong. Aggregating Web offers to determine product prices. In *KDD*, 2012.

[2] A. Arasu and H. Garcia-Molina. Extracting structured data from Web pages. In *SIGMOD*, 2003.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A nucleus for a Web of open data. In *ISWC*, 2007.

[4] A. Bakalov, A. Fuxman, P. P. Talukdar, and S. Chakrabarti. Scad: Collective discovery of attribute values. In *WWW*, 2011.

[5] M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open Information Extraction from the Web. In *IJCAI*, 2007.

[6] R. Baumgartner, S. Flesca, and G. Gottlob. Visual Web information extraction with Lixto. In *VLDB*, 2001.

[7] P. Bohannon, N. Dalvi, Y. Filmus, N. Jacoby, S. Keerthi, and A. Kirpal. Automatic Web-scale information extraction. In *CIKM*, 2012.

[8] M. Bronzi, V. Crescenzi, P. Merialdo, and P. Papotti. Extraction and integration of partially overlapping Web sources. In *VLDB*, 2013.

[9] L. Brown, T. Cai, and A. Dasgupta. Interval Estimation for a Binomial Proportion. *Statistical Science*, 16(2), 2001.

[10] A. Carlson, J. Betteridge, R. C. Wang, E. R. H. Jr., and T. M. Mitchell. Coupled semi-supervised learning for information extraction. In *WSDM*, 2010.

[11] C. Chang, M. Kayed, M. Girgis, and K. Shaalan. A survey of Web information extraction systems. *TKDE*, 18(10), 2006.

[12] W. W. Cohen, M. Hurst, and L. S. Jensen. A flexible learning system for wrapping tables and lists in HTML documents. In *WWW*, 2002.

[13] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large Web sites. In *VLDB*, 2011.

[14] N. Dalvi, R. Kumar, and M. A. Soliman. Automatic wrappers for large scale Web extraction. In *VLDB*, 2011.

[15] N. Derouiche, B. Cautis, and T. Abdessalem. Automatic extraction of structured Web data with domain knowledge. In *ICDE*, 2012.

[16] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *NCAI*, 2000.

[17] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang. DIADEM: Thousands of websites to a single database. In *VLDB*, 2014.

[18] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Kruepl, and B. Pollak. Towards domain-independent information extraction from Web tables. In *WWW*, 2007.

[19] R. Ghani, K. Probst, Y. Liu, M. Krema, and A. Fano. Text mining for product attribute extraction. *SIGKDD Explor. Newsl.*, 8(1), 2006.

[20] P. Gulhane, R. Rastogi, S. H. Sengamedu, and A. Tengli. Exploiting content redundancy for Web information extraction. In *VLDB*, 2010.

[21] A. Kannan, I. Givoni, R. Agrawal, and A. Fuxman. Matching unstructured product offers to structured product specifications. In *KDD*, 2011.

[22] H. Köpcke, A. Thor, S. Thomas, and E. Rahm. Tailoring entity resolution for matching product offers. In *EDBT*, 2012.

[23] A. Kopliku, M. Boughanem, and K. Pinel-Sauvagnat. Towards a framework for attribute retrieval. In *CIKM*, 2011.

[24] W. Y. Lin and W. Lam. Learning to extract hierarchical information from semi-structured documents. In *CIKM*, 2000.

[25] J.-B. Michel et al. Quantitative analysis of culture using millions of digitized books. *Science*, 331(6041), 2011.

[26] N. Nakashole, G. Weikum, and F. M. Suchanek. PATTY: A taxonomy of relational patterns with semantic types. In *EMNLP-CoNLL*, 2012.

[27] H. Nguyen, A. Fuxman, S. Paparizos, J. Freire, and R. Agrawal. Synthesizing products for online catalogs. *PVLDB*, 4(7), 2011.

[28] Z. Nie, Y. Ma, S. Shi, J. Wen, and W. Ma. Web object retrieval. In *WWW*, 2007.

[29] T. Pham and K. Nguyen. A simhash-based scheme for locating product information from the Web. In *SoICT*, 2011.

[30] K. Probst, R. Ghani, M. Krema, A. Fano, and Y. Liu. Semi-supervised learning of attribute-value pairs from product descriptions. In *IJCAI*, 2007.

[31] D. Putthividhya and J. Hu. Bootstrapped named entity recognition for product attribute extraction. In *EMNLP*, 2011.

[32] S. Sarawagi. Information Extraction. *Foundations and Trends in Databases*, 2(1), 2008.

[33] K. Simon and G. Lausen. ViPER: augmenting automatic information extraction with visual perceptions. In *CIKM*, 2005.

[34] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A core of semantic knowledge. In *WWW*, 2007.

[35] A. Talaika, J. Biega, A. Amarilli, and F. M. Suchanek. Harvesting entities from the web using unique identifiers - IBEX. Technical report, Telecom ParisTech, 2015. `http://suchanek.name/work/publications/ibex2015tr.pdf`.

[36] Techspot. Gmail finally overtakes Hotmail as world's top email service. `http://techspot.com/news/50678-google.html`, 2012. Accessed: 2014-11-07.

[37] United States Census Bureau. US census. `http://www.census.gov/data/data-tools.html`, 1990. Accessed: 2013-10-01.

[38] P. Venetis, A. Halevy, J. Madhavan, and et al. Recovering semantics of tables on the Web. In *VLDB*, 2011.

[39] World Bank. GDP (current US$). `http://data.worldbank.org/indicator/NY.GDP.MKTP.CD`. Accessed: 2013-10-01.

[40] World Trade Organization. International trade statistics: World trade developments. `https://www.wto.org/english/res_e/statis_e/its2012_e/its12_world_trade_dev_e.pdf`, 2012. Accessed: 2014-11-07.

[41] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *WWW*, 2005.

[42] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. In *WWW*, 2005.

[43] S. Zheng, R. Song, J. R. Wen, and C. L. Giles. Efficient record-level wrapper induction. In *CIKM*, 2009.

[44] J. Zhu, Z. Nie, X. Liu, B. Zhang, and J. Wen. StatSnowball: a statistical approach to extracting entity relationships. In *WWW*, 2009.

[45] J. Zhu, Z. Nie, J. R. Wen, B. Zhang, and W. Y. Ma. Simultaneous record detection and attribute labeling in Web data extraction. In *SIGKDD*, 2006.