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 an almost endless resource of named entities, such as commercial products, people, books, and organizations. In this paper, we focus on those entities that have unique ids. An id is any string or number that distinguishes the entity in a globally unique way from other entities. For example, commercial products have ids in the form of GTIN codes. These are the numeric codes 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

But not just commercial products have ids. A surprisingly large portion of other entities also do. Companies have tax identification numbers; books have ISBN numbers; documents have document identifiers; chemical substances have ids in the form of CAS registry numbers, and so on. Quite frequently, Web pages that talk about these entities also mention their ids.

Goal. Our goal is to harvest these ids at large scale from the Web, together with the names of the entities that they identify. That is, our goal is to build a database that contains, in the example, (8806085725072, Samsung Galaxy S4). Using named entity recognition (NER), ids and entity names can be spotted in the pages. However, a page usually contains several entity names, and only one of them is usually the name of the entity in question. The challenge is thus to associate, with each id, the proper name for the entity. In the example, the challenge is to find that the correct name for the id “8806085725072” is “Samsung Galaxy S4” – and not “Samsung”, “VAT”, or “GT-I9295ZAADBT”.

It is far from trivial to associate the correct entity name to an id. First, Web pages contain usually dozens of entity names, so it is not clear which one corresponds to the id. In the example, “Samsung” is clearly an entity name, but not the correct one. Worse, some Web pages contain several ids and several entity names at the same time, so we must correctly match the ids and names on the page. The excerpt of Figure 1 is taken from a page that lists dozens of Samsung products.

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 it must accept entirely arbitrary pages, with possibly erroneous content, broken structure, or noisy information. This makes it impossible to rely on wrapper induction, or indeed on any predefined or learnable DOM tree structure. We have to be able to find the entity names in tables, in lists, as well as in plain unstructured text. These challenges come in addition to the usual difficulties such as non-standard HTML code, non-semantic markup (e.g., tables used for page layout), and creative tag combinations to arrange tabular information.

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

This allows us to extract millions of distinct entities from the Web, with an accuracy of 73% to 96% depending on the entity types. The result is a database of entity ids and names, with information about which pages mention which entities. The crucial advantage of our database is that every entity is guaranteed to be unique, so we can count distinct entities without being biased by duplicates. Thus, we can perform a detailed study of entities that exist on the Web: we can identify Web sites that are hubs for books or documents, we can build statistics about frequent first names of people, and we can determine which countries produce most products. We can trace producing countries, importing countries, and the flow of products from one to the other. In other words, we show not only how entities can be extracted from the Web, but also how they are distributed on the Web, and in the world.

Our core contributions are as follows:

• A generic algorithm for harvesting entity ids at large scale from Web pages and associating them with their entity name, independently of the page structure.

• A database of unique entities at Web scale, which contains millions of objects at an accuracy of 73–96%.

• Detailed statistics on products, documents, and people on the Web, computed using unique identifiers.

The paper is structured as follows. We first discuss related work in
Section 2. We then define our problem in Section 3 and our approach to solve it in Section 4. The details of Web page parsing are described in Section 5. We present our experiments in Section 6 and show in Section 7 some examples of analyses that can be conducted on our entity database. Section 8 concludes.

2. RELATED WORK

Our goal is to extract unique ids from Web pages and to associate, with each id, the correct entity name. We now survey related work about this goal.

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 (see [37], [13] for surveys). We rely on NER to spot ids and candidate names in Web pages. However, our focus is not on these methods. Rather, our focus is on associating, with each id, the correct name among several extracted name candidates. While NER can spot entity names, it cannot determine which entity name relates to which id, if the page contains several entity names. Our approach aims at solving this problem.

Wrapper induction. Wrapper induction [48], [15], [14], [18], [49], [10], [17], [8], [22] 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, because 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 [19] can deal with such variety, but targets the deep Web, while we target the surface Web.

Structured IE. A large suite of approaches (e.g., [2], [26], [27], [51]) aims at extracting information from structured sources. Some techniques use visual clues [38], [6]; others make use of the DOM structure of the Web pages [46]. Yet others make use of the schema [4]. Unlike these approaches, our method does not assume any particular structure in Web pages. It does not require that pages resemble each other, it does not need training data, and it does not assume a given schema. Rather, it works on both structured and completely 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 [24], [25]. The work by [25] succinctly mentions also manufacturer extraction from product titles. Supervised learning approaches have been proposed to update product catalogues using new offers [23], or to determine product prices [1]. These approaches, however, build on an existing catalog of products. Our goal, in contrast, is the creation of such a catalog.

Other work [36], [33], [21] handles product attribute extraction. [14] gives a method for discovering 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, as we do.

Knowledge bases. Recent work has led to the automated creation of large knowledge bases [49], [21], [11]. These contain many popular and important entities, but do not aim to be exhaustive. DBpedia, e.g., contains “only” a few ten thousand instances of the class Product, most of them being named ships. Our goal, in contrast, is to collect products systematically from the Web.

Web-scale databases. Several projects [42], [20], [7], [33], [50] 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 extract even from 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.

2.2 Entity Databases

There are several databases of unique Web entities.

Products. The UPC Database (http://www.upcdatabase.com) contains 1.6M ids of commercial products, but is not available for download. GTIN13.com (http://gtin13.com) and Smoopa (http://www.smoopa.com/) are other databases of ids, but no information on their content is freely available. Amazon and other booksellers store large numbers of ISBN codes, and some search engines may rely on a catalog of products, but those databases are not available for download, because having a product repository is a competitive advantage. Our open methodology and dataset, in contrast, can be used freely by any vendor to improve coverage.

Documents. The International DOI Foundation (http://www.doi.org/) assigns identifiers to text documents upon request. There are 84M document ids. However, the foundation does not provide a central search capability across all DOI names, and the data cannot be downloaded.

Chemistry. 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 for only 7,900 substances.

People. There are some commercial Websites that scrape the Web for personal data (e.g. http://www.yasni.com/), however, they do not provide a downloadable dataset of people. Social networks, likewise, harvest personal data, but do not make them available for public use.

The main advantage of our database that it is freely available, while most existing databases are commercial. Furthermore, our method is a general technique that can apply to any entities that have unique identifiers, whereas existing approaches are domain-specific.

3. PROBLEM STATEMENT

Our goal is to build a database of unique entity ids from Web pages, and to associate to each id a human-readable name. We now formally define our notion of ids and the problem that 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 follow the same syntax, and that refer to entities of the same domain. For example, ISBNs are always sequences of 10-13 digits, and they are used to identify books. CAS numbers are sequences of 8 digits that identify chemical substances. The only assumptions that we make is that no two entities can have the same id (e.g., one ISBN cannot refer to two different books), and that one entity can only have one id in one type (e.g., a book can only have one ISBN number, but it can have also a GTIN code, because GTINs are a different id type that happen to include books).

In some cases, we cannot make the assumption that entities have only one id in a type. For example, every personal email address belongs to one person, but one person can have several personal email addresses. In this case, we call the identifier a pseudo-id. 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 multiple times in the con-
Our goal is to figure out which candidate belongs to the string. The input to our method is a set of real examples of entities of these types, obtained from our data. The problem is that we can expect from it.

Table 1: Id types

<table>
<thead>
<tr>
<th>Id type</th>
<th>Entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISBN</td>
<td>Books</td>
</tr>
<tr>
<td>GTIN</td>
<td>Products</td>
</tr>
<tr>
<td>CAS</td>
<td>Chemicals</td>
</tr>
<tr>
<td>DOI</td>
<td>Documents</td>
</tr>
<tr>
<td>VATIN</td>
<td>Companies</td>
</tr>
<tr>
<td>BIC</td>
<td>Banks</td>
</tr>
<tr>
<td>NSN</td>
<td>Military products</td>
</tr>
<tr>
<td>ISIN</td>
<td>Stocks</td>
</tr>
<tr>
<td>VIN</td>
<td>Vehicles</td>
</tr>
<tr>
<td>GRID</td>
<td>Digital recordings</td>
</tr>
<tr>
<td>ISAN</td>
<td>Audiovisual material</td>
</tr>
<tr>
<td>Pub#</td>
<td>US Patents</td>
</tr>
<tr>
<td>ILU</td>
<td>Containers</td>
</tr>
<tr>
<td>MESH</td>
<td>Chemicals</td>
</tr>
<tr>
<td>OMIM</td>
<td>Diseases</td>
</tr>
<tr>
<td>ICD-10</td>
<td>Diseases</td>
</tr>
<tr>
<td>Email</td>
<td>People/organizations (pseudo-id)</td>
</tr>
<tr>
<td>IBAN</td>
<td>People/organizations (pseudo-id)</td>
</tr>
<tr>
<td>Phone</td>
<td>People/organizations (pseudo-id)</td>
</tr>
</tbody>
</table>

The first phase extracts ids and name candidates. Let \(\mathcal{P}\) be a function that, given an id of type \(t\), and two NER modules for \(t\): the id validator \(f^\text{id}_t\) and the name finder \(f^\text{name}_t\). The id validator is a function that takes as input a string and returns true if 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.

The problem is that \(f^\text{id}_t\) will sometimes extract multiple ids from the same page, and \(f^\text{name}_t\) will typically extract a large number of name candidates. Our goal is to figure out which candidate belongs to which id, and what is the best name for which id. This is particularly difficult if a page talks about several entities, and thus contains several ids and several names. Even if the page just talks about one entity, it typically contains dozens if not hundreds of name candidates. As we will show, the correct name can be selected with high precision by leveraging the uniqueness property of ids at large scale.

The output of our method is an entity database for type \(t\), which contains ids of type \(t\), and associates each id with the name of the entity – much like Table 2. In addition, we store with each entity the URLs of the Web pages where the entity was found.

4. APPROACH

We first describe our approach, and next discuss the coverage that we can expect from it.

4.1 Method Description

In all of the following, we assume given a set \(\mathcal{P}\) of Web pages, an id type \(t\), and NER modules \(f^\text{id}_t\) and \(f^\text{name}_t\) for that type. Our approach proceeds in 3 phases. We first describe our algorithm at a high level, deferring the implementation of Phase 1 to Section 5 and more details to the appendix.

Phase 1. The first phase extracts ids and name candidates. Let us consider one page \(p \in \mathcal{P}\). 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 \(i\) and all name candidates \(\text{name}_{i,j}\) for \(i\). To account for differences in writing, we normalize all name candidates by upper-casing them, and by retaining only letters (alphanumeric characters for CAS). Each name candidate also comes with a score that indicates its likelihood of being the correct name for \(i\). We discuss different scoring models and our final choice in Appendix A.

The result of this process is a table of the following form for each record \(r_i\):

\[
R^p_i := \{ (i, j, \text{name}_{i,j}, \text{score}_{i,j}, \text{url}(p)) | j = 1, \ldots, m \}
\]

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 \(R^p_i\) (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:

\[
R_1 := \bigcup_{p \in \mathcal{P}} \bigcup_{r \in p} R^p_i
\]

Again, \(R_1\) will contain several name candidates for the same id, extracted from the same page or from different pages. It may also contain the same name multiple times. The idea is that the subsequent phases will filter out the erroneous names.

Phase 2. The previous phase has just extracted all possible name candidates for all id occurrences. Hence, \(R_1\) is very noisy and contains a large number of wrong candidates. For instance, some name candidates are not entity names, but rather descriptive elements, such as “Price”, “see also”, or “plastic”.

Since our corpus of Web pages 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 3 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.

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
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.

name $n$ how often it appears with the id $id$ in R1:

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

Now, we attempt to identify names $n$ whose distribution $freq_n$ shows a clear outlier. We experimented with several outlier detection methods, which we detail in Appendix B. In the end, the following technique works best. Let $id_n^1$ and $id_n^2$ be the ids with the highest and second highest value for $freq_n(i)$, breaking ties arbitrarily. The distribution $freq_n(i)$ is said to have an outlier if $id_n^3$ appears in more than 30% of the cases, and at least 3 times more often than $id_n^2$.

$$freq_n(id_n^1) > 0.3 \times \sum_i freq_n(i)$$

$$freq_n(id_n^2) > 3 \times freq_n(id_n^2)$$

This technique is robust enough to work across the board for all id types that we considered. If a name appears with only one id, it is always considered an outlier. Now, Phase 2 removes all names that show no clear outlier:

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

This leaves us with a table $R2$ that 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 the 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):

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

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

Phase 3 leaves us with a table $R3$ that contains, for every id, a single name, and the URLs of all Web pages where this entity was found. This is our final entity database.

### 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 only have access to a subset of pages that was found through crawling. Hence, our dataset is necessarily 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 $\mathcal{W}$ of pages on the entire Web that mention at least one entity of type $t$, and we consider $\mathcal{P} := \mathcal{P} \cap \mathcal{W}$ the set of the input pages $\mathcal{P}$ that mention some entity of type $t$. We write $e$ for $\mathcal{P} \cap \mathcal{P}$, and we assume that $\mathcal{P}$ is a subset of $\alpha |\mathcal{W}|$ pages drawn uniformly at random in $\mathcal{W}$.

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

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

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.

**Proof.** Call $x$ the expected value of $|\mathcal{E}'|$. For each entity $e \in \mathcal{E}$, define a random variable $E_e$ which is 1 if $e$ occurs in $\mathcal{P}$, and 0 otherwise. It follows that $|\mathcal{E}'| = \sum_{e \in \mathcal{E}} E_e$, as the number of entities occurring in $\mathcal{P}$ is the sum, for each entity, of 0 if it occurs in $\mathcal{P}$ and 1 otherwise. Now, by linearity of expectation, $x$ is the sum of the expected values of the $E_e$, and, as $E_e \in \{0,1\}$, this means that $x = \sum_{e \in \mathcal{E}} E_e \times q_e$, where $q_e$ is the probability that $E_e = 1$.

We now show that $q_e > \alpha$ for all $e \in \mathcal{E}'$. Indeed, let us fix $e \in \mathcal{E}'$, and let $p_e \in \mathcal{P}$ be an arbitrary page where $e$ occurs. Let $E'_e$ be a random variable which is 1 if $p_e$ occurs in $\mathcal{P}$, and 0 otherwise, and $q'_e$ be the probability that $E'_e = 1$. By definition $E'_e = 1$ implies that $E_e = 1$, so that $q_e > q'_e$. Now, the variable $E'_e$ can be modeled as a hypergeometric distribution with the following parameters: the number of draws is $\alpha |\mathcal{W}|$ (corresponding to $|\mathcal{P}|$), the number of

<table>
<thead>
<tr>
<th>Id type</th>
<th>Id</th>
<th>Entity name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTIN</td>
<td>00968888883955</td>
<td>Pyramid PA305 100w Rack Mount Amplifier with Mixer</td>
</tr>
<tr>
<td>GTIN</td>
<td>09783540442820</td>
<td>Machine Translation: From Research to Real Users</td>
</tr>
<tr>
<td>CAS</td>
<td>78123-16-7</td>
<td>N-benzyl-2-(2-methyl-1H-indol-3-yl)acetohydrazide</td>
</tr>
<tr>
<td>CAS</td>
<td>67011-42-1</td>
<td>3-acetamido-5-(hexanoylamino)-2,4,6-triiodo-benzoic acid</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:widom@cs.stanford.edu">widom@cs.stanford.edu</a></td>
<td>Dr. Jennifer Widom</td>
</tr>
</tbody>
</table>

| Table 2: Examples for ids and entity names |

<table>
<thead>
<tr>
<th>Id type</th>
<th>Id</th>
<th>Entity name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTIN</td>
<td>00968888883955</td>
<td>Pyramid PA305 100w Rack Mount Amplifier with Mixer</td>
</tr>
<tr>
<td>GTIN</td>
<td>09783540442820</td>
<td>Machine Translation: From Research to Real Users</td>
</tr>
<tr>
<td>CAS</td>
<td>78123-16-7</td>
<td>N-benzyl-2-(2-methyl-1H-indol-3-yl)acetohydrazide</td>
</tr>
<tr>
<td>CAS</td>
<td>67011-42-1</td>
<td>3-acetamido-5-(hexanoylamino)-2,4,6-triiodo-benzoic acid</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:widom@cs.stanford.edu">widom@cs.stanford.edu</a></td>
<td>Dr. Jennifer Widom</td>
</tr>
</tbody>
</table>

| Table 2: Examples for ids and entity names |

<table>
<thead>
<tr>
<th>Id type</th>
<th>Id</th>
<th>Entity name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTIN</td>
<td>00968888883955</td>
<td>Pyramid PA305 100w Rack Mount Amplifier with Mixer</td>
</tr>
<tr>
<td>GTIN</td>
<td>09783540442820</td>
<td>Machine Translation: From Research to Real Users</td>
</tr>
<tr>
<td>CAS</td>
<td>78123-16-7</td>
<td>N-benzyl-2-(2-methyl-1H-indol-3-yl)acetohydrazide</td>
</tr>
<tr>
<td>CAS</td>
<td>67011-42-1</td>
<td>3-acetamido-5-(hexanoylamino)-2,4,6-triiodo-benzoic acid</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:widom@cs.stanford.edu">widom@cs.stanford.edu</a></td>
<td>Dr. Jennifer Widom</td>
</tr>
</tbody>
</table>
successes is 1 (corresponding to the page \( p_e \)), and the population size is \(|\mathcal{W}|\). Indeed, \( E'_p \) is the number of successes (here, 0 if \( p_e \) is not drawn and 1 if it is drawn) when drawing \(|\mathcal{W}|\) pages, without replacement, from the \(|\mathcal{W}|\) possible pages. Hence, from the probability mass function of the hypergeometric distribution, we deduce that \( q'_e = \frac{(|\mathcal{W}|-1)}{|\mathcal{P}_1|-1} / \frac{|\mathcal{W}|}{|\mathcal{P}_1|} \), which simplifies to \( \frac{|\mathcal{P}_1|}{|\mathcal{W}|} \), that is, \( q'_e = \alpha \), so that \( q_e \geq \alpha \).

What is more, letting \( e_0 \in \mathcal{E} \) be an entity that occurs in strictly more than one page of \( \mathcal{P} \), we have \( q_{e_0} > q'_e \). Indeed, if \( p'_e \) is a page different from \( p_{e_0} \), where \( e_0 \) occurs, there is a draw of \( |\mathcal{P}| \) where we have \( E'_{e_0} = 0 \) but \( E'_e = 1 \), namely, any draw where \( p'_e \neq p_{e_0} \), but \( p_{e_0} \notin \mathcal{P} \). Hence, as these additional draws have non-zero probability, \( q_{e_0} > q'_e \).

Now, putting it together, we have that \( x = \sum_{e \in \mathcal{E}} q_e > \sum_{e \in \mathcal{E}} q'_e \), as the bound is strict for the term with \( e_0 \). By the above, this simplifies to \( x > \alpha |\mathcal{E}| \).

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 can in theory be parsed to a DOM tree that represents the structure of the page. However, while HTML defines tags with nesting syntax and semantics, neither of them is always respected by Web site creators. In fact, a large number of HTML documents on the Web cannot be properly parsed into a DOM tree because they are not well-formed.

In addition, the structure of the DOM tree does not correspond to the page structure as seen by a human. For example, if a page contains several \( \text{H1} \) tags, then a human sees several sections. The DOM tree, however, contains just one parent node with \( \text{H1} \)-nodes and text-nodes in alternation as children. If the page discusses different entities, then it is likely that each entity falls within one \( \text{H1} \)-dominated block (see Section A for experiments on this). 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.

This problem is well-known, and it is addressed by work on Web page segmentation. Most approaches, however, render the page visually \( [28, 20, 48, 10] \), including layout or style sheet analyses. This is too expensive in our scenario, where we need a rough and rapid segmentation of pages at Web scale. Other approaches use machine learning \([52, 46]\), but there is no training data in our scenario. Another method, MDR \([46]\), can detect tabular structures in Web pages. Our method is inspired by MDR, but more generally applies 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 unrelated 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 \( \text{H1} \)..., \( \text{H6} \), \( \text{HR} \), \( \text{BR} \), sequences of \( \text{BR} \), and \( \text{P} \). 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.

**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 \( \text{DOC} \), 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, \( \text{DOC} \succ \text{HTML} \succ \text{BODY} \succ \text{DIV} \). 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 \( \text{H1}* \), which will be the label for a frame that consists of a \( \text{H1} \)-header and the following text. We extend \( \succ \) to cover also these tags. For example, a \( \text{H1} \)-frame can contain \( \text{H2} \)-frames: \( \text{H1}* \succ \text{H2}* \). 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.

**Example.** Consider the following HTML document:

```
<body>
<h1>Samsung Galaxy S4</h1>
<p>Id: &lt;b&gt;8806085725072</b&gt;
<h2>Accessories</h2>
<h2>Galaxy S4 Charging Cable</h2>
4047443213525
</body>
```

This document is not correct HTML: there is no \( \text{HTML} \) or \( \text{head} \) element, closing tags are missing, etc. Yet, Algorithm 1 is able to parse this document, yielding the frame tree of Figure 3 (omitting the dummy \( \text{DOC} \) node). The figure shows ids in bold and the names that should be extracted in italics.
5.2 Extracting Records

Algorithm 1 has produced a frame tree for an input Web page. We now show how to find records in this tree. A record $r$ of type $t$ in a page $p \in \mathcal{P}$ is a subtree rooted at some node of the frame tree of $p$ that contains a single id of type $t$. We say that $r$ is a detail record if it is the only record of $p$; otherwise, we call it a free record. Intuitively, detail records occur when the entire page is concerned about the entity, and free records typically belong to listings of entities, such as lists or tables, or possibly free-floating descriptions of entities.

We find records on a page as follows. We apply the id validator $\text{id}$ to all text frames of the frame tree, and mark the frames that are valid ids. If the page contains exactly one id, we have a detail record, which is just the root frame of the entire page. If the page contains several ids, we traverse the frame tree in a depth-first search. As soon as we find a subframe that contains exactly one id, we construct a free record from that subframe. In our example in Figure 3, we would find two free records, rooted at each H1* node.

Once all records in a page have been identified, the function $\text{name}$ is applied to all text subframes of each record. This yields a set of candidate names per record. In our example in Figure 3, we would extract the candidate names for the first id, and the candidates "Accessories" and "Galaxy S4 Charging Cable" for the second id. The ids together with their candidate names make up the table R1 of our Phase 1.

5.3 Discussion

We remark that 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. The same is true for HTML lists: if each item in an HTML list contains an id, then each item will become a record. The same goes for any other repetitive structure, no matter which tag is used to delimit the items; conversely, a TABLE tag that does not actually describe a table will not confuse the algorithm. This makes the algorithm robust enough to run on arbitrary Web sites.

The flipside is that Algorithm 1 cannot rely on structural rules which usually hold about HTML tables. For instance, if each row of a table corresponds to an entity, one could expect that entity names will 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, it extracts all candidate names alogistically. We rely on Phase 2 to clean up the result of this extraction by filtering out the globally frequent names.
In order to estimate the potential of our approach, we could extend the attribute-finder to extract also attributes of the entities. The center of the Wilson interval, 93%, is the estimated ratio, which may differ from the percentage of correct evaluations.

### 6.2 Extensions

**Attribute extraction.** Entities can have certain attributes. Commercial products, e.g., can have a price, and chemicals have a molecular formula. To show that our approach could in principle be extended to extract also the attributes of an entity, we consider the extraction of molecular formulae for chemical substances (e.g., “Cd5Cl(PO4)3”). We build a regular expression that accepts any sequence of names and digits, where each name has to be a valid chemical element name from the periodic table. This yields an attribute-finder, which works analogously to a name finder, and which runs through all 3 phases. With this approach, we could extract 1,662 chemical formulae. We picked a random sample of 50 values, and checked them manually. We obtain an accuracy of 93% ± 6%.

**Other id types.** In order to estimate the potential of our approach for other types of ids, we implemented the name finders and id checkers for ICD-10 disease codes, OMIM disease codes, VATIN company tax codes for France, and MESH chemical codes. On our corpus, the algorithm returned 2,418 distinct MESH chemicals, 7,521 OMIM diseases, and 240 VATIN companies. On the French part of ClueWeb 2009, in contrast, we found 2,233 distinct VATIN companies. These numbers make us believe that our approach can be extended to different ids and to different corpora.

**Extrinsic application.** Our data can be seen as a basic knowledge base (entities, labels, ids, types) and thus can be used in different scenarios employing KBs. As a proof of concept we merged our corpus, the algorithm returned 2,661 distinct ICD-10 diseases, checkers for ICD-10 disease codes, OMIM disease codes, VATIN company tax codes for France, and MESH chemical codes. On our corpus, the algorithm returned 2,418 distinct MESH chemicals, 7,521 OMIM diseases, and 240 VATIN companies. On the French part of ClueWeb 2009, in contrast, we found 2,233 distinct VATIN companies. These numbers make us believe that our approach can be extended to different ids and to different corpora.

### 7. ANALYSES

Our dataset is a huge resource of Web objects, which can give rise to different analyses – much in the spirit of Culturomics [29]. The following experiments illustrate this.

#### 7.1 Resources

Our dataset can identify Internet domains that are particularly rich in different types of Web entities. In Table 4, we show the best data sources that we could determine for email addresses, chemical, and documents. This analysis could help steer information extraction approaches to target domains that are rich in the desired items. Most notably, amazon.com is not among the most common domains. We assume that, if it were added to the crawl, it would multiply the number of products that we would find.

We also computed the most frequent email providers in our data (Table 7). Our email addresses come mostly from Gmail, followed by Yahoo! and Hotmail. These are indeed the top three email providers, as determined by the Techspot magazine [40].

#### 7.2 People

**Common names.** Our extraction found 13 million email addresses with an associated person name. However, email addresses are only pseudo-ids: a person can have several email addresses, and so
we may not conclude that we found 13 million people. However, we may assume that the number of email addresses that a person owns is independent of their name. Therefore, we can compute the most common given names and the most common family names on the Web (Table 5). The popular first names that we found correspond roughly to the frequent English names. Our top 50 male given names cover 43 of the top 50 male given names of the US 1990 census data [41]. We also mined the most common complete names, with “John Smith” being the single most common name.

**Gender.** We extended our analysis to finding the gender of the people on the Web. By intersecting our set of given names with the US census data about common female and male first names, we can assign a gender to each person name. For the 331 unisex names, we attributed both genders proportionally, based on the name frequency statistics, to take advantage of any gender priors on them.

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 identify the company that produced the product. Unfortunately, there is no publicly available database that maps these prefixes to company names. Therefore, we resorted to the following method. We assume that the product titles often contain the company name. We grouped the GTINs by their 4-digit prefix. Then, we 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 ±7%. Table 6 shows the top companies.

**Countries.** The first 3 digits of a GTIN identify the country of production of a product. To conduct this analysis, we extracted the GTINs from the entire ClueWeb corpus (not just the English one). We calculated the number of unique items produced in different countries, and show the top countries in Table 8. If we compare the ranking to the list of countries by GDP (as provided by the World Bank [43]), we find a remarkable overlap. Our top 5 countries cover 4 out of the 5 countries with highest GDP. To investigate this similarity further, we built a vector that contains, for each country, the number of products that we could find. We built another vector that contains, for each country, its GDP. We find a cosine similarity of 79% between these vectors. We take this as an indication that the GTINs can serve as a proxy of the productivity of a country.

**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 which countries export to which other countries. We grouped the countries by the regions that the World Trade Organization (WTO) uses [44]. We took again the GTINs from the entire ClueWeb, and plotted the trade flow on a map (Figure 4).

<table>
<thead>
<tr>
<th>Country</th>
<th>#Products</th>
<th>Country</th>
<th>GDP (trillion)</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>1,024,219</td>
<td>USA</td>
<td>14.99 US$</td>
</tr>
<tr>
<td>UK</td>
<td>59,542</td>
<td>China</td>
<td>7.20 US$</td>
</tr>
<tr>
<td>Germany</td>
<td>26,949</td>
<td>Japan</td>
<td>5.87 US$</td>
</tr>
<tr>
<td>Japan</td>
<td>17,353</td>
<td>Germany</td>
<td>3.60 US$</td>
</tr>
<tr>
<td>France</td>
<td>12,845</td>
<td>France</td>
<td>2.77 US$</td>
</tr>
</tbody>
</table>

Table 8: Countries by production

**Figure 4: Intra- and inter-regional trade (log-scale)**

“CIS” stands for the Commonwealth of Independent States, which consists of the former Soviet Republics. The size of the circles corresponds to the number of products produced in and advertised within one region. The size of the arrows corresponds to the number of products produced in one region and advertised in another one. The scale is logarithmic. We found no products produced in Africa and advertised in Africa. The other quantities correspond roughly to what one would expect: Europe, North America, and Asia are the dominant exporters.

We compared our analysis to the true flow of products between the regions, as estimated by the WTO [45]. For each region, we construct a vector that contains the number of exported products for each target region. We compare this vector to the vector of exported merchandise in US$ from the WTO. We computed the cosine similarity of the vectors for each region, and found remarkable overlap. The similarity is lowest (49%) for the Middle East. We hypothesize that this is due to the fact that the Middle East exports many commodities that are not sold by GTIN. For Europe, Asia, and the CIS, however, the values are over 95%.

### 8. CONCLUSION

In this paper, we have shown how to harvest entities with unique ids systematically from the Web. By making use of the properties of ids, we could extract entity names with an accuracy of 73–96%. This allowed us to create 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 that contains so many such items in a canonicalized manner. We have shown possible uses of this database by conducting a number of analyses, which include the frequent names of people, and the flow of trade in the world. We 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 picked here as examples, but also to a broad range of other entities. It should be possible to extract banks, companies, audiovisual material, or possibly even social network ids. This will make the Web more and more semantic, and thus help making the Internet ever more useful.

We have made all data and analyses available online at [http://resources.mpi-inf.mpg.de/d5/ibex](http://resources.mpi-inf.mpg.de/d5/ibex) and will make it available publicly should the paper be accepted.
A. SCORING NAME CANDIDATES

We now discuss several design alternatives for the scoring model used in Phase 1. We compare the performances of the design alternatives on test data. For this, we use the same experimental setup as in our final experiments (see Section 6).

**Score design.** In Phase 1, each pair of an entity id and an entity name receives a real-valued score. The score can also be NIL, meaning that the candidate name is completely removed from the list. To evaluate different design alternatives for scoring models, we compared them against a gold standard constructed in the following way. We first ran just the id validator $f^i_D$ for all id types $i$ on all text frames of all pages of our corpus. We sampled a set of 200 id occurrences randomly for each type (different from those of Section 5). We manually inspected the Web pages where these ids occurred, and extracted the correct entity name for each id by hand.

Now, to evaluate the performance of a scoring model against this gold standard, we run Phase 1 of our extraction on all pages of $\mathcal{P}$, and rank for every id all the candidate names by decreasing score, breaking ties arbitrarily. (Note that the scoring model may also remove a candidate entirely from the list.) Our goal is to design a scoring model that retains the correct name candidates, and ranks them closer to the top. Formally, we compute the precision at rank $i$ as the proportion of ids with non-empty rankings where the correct name is among the first $i$ ranked candidates. The recall at rank $i$ is the proportion of ids where the correct name is among the first $i$. We tried different scoring models. Table 9 shows the results for GTINs, choosing for each scoring model the rank that maximises the precision (the results for the other types are comparable).

**Models for detail records.** We now present the various scoring models that we evaluate, looking only at detail records. Our first scoring model, random, assigns the same value 1 to all candidates. Thus, the precision and recall reflect what a random assignment of extracted entity names to ids would produce. Predictably, the performance is not very impressive, and the maximum precision of 86% is achieved only at rank 285 (we do not achieve a precision of 100% because the NER modules are not perfect).

Our next scoring model, title, removes a name if the name does not occur in the title tag of the page (and assigns 1 otherwise). This decreases recall slightly, but improves the rank with maximal precision drastically, to 17. We observed that the name of an entity occurs before the id in the large majority of cases. Hence, our next model, order, removes a candidate if it appears after the id in the HTML file (and assigns 1 otherwise). This decreases the recall only slightly, but improves the rank with maximal precision to 187. The table shows that a combination of the order and title feature improves the rank with maximal precision even to 6.

<table>
<thead>
<tr>
<th>Type</th>
<th>Scoring model</th>
<th>Prec.</th>
<th>Rec.</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detail</td>
<td>random</td>
<td>86%</td>
<td>25%</td>
<td>275</td>
</tr>
<tr>
<td>Detail</td>
<td>title</td>
<td>71%</td>
<td>16%</td>
<td>17</td>
</tr>
<tr>
<td>Detail</td>
<td>order</td>
<td>84%</td>
<td>24%</td>
<td>187</td>
</tr>
<tr>
<td>Detail</td>
<td>order + title</td>
<td>70%</td>
<td>16%</td>
<td>6</td>
</tr>
<tr>
<td>Detail</td>
<td>order + distance</td>
<td>84%</td>
<td>24%</td>
<td>78</td>
</tr>
<tr>
<td>Detail</td>
<td>order + distance + style4</td>
<td>68%</td>
<td>15%</td>
<td>3</td>
</tr>
<tr>
<td>Detail</td>
<td>order + distance + style4 + title</td>
<td>75%</td>
<td>10%</td>
<td>1</td>
</tr>
</tbody>
</table>

| Free       | first3        | 80%   | 29%  | 9    |
| Free       | first3 + style| 85%   | 29%  | 9    |

Figure 5: Number of ids with which each particular name occurs in a log-log scale. A point $(x, y)$ means that there are $y$ names that occur with $x$ ids.

Through manual inspection, we found that name candidates that are closer to the id are more likely to be the correct name. Hence, our next scoring model, distance, scores a name by the negative distance between the name and the id. If, e.g., there are 5 name candidates that lie between the current candidate and the id, then the score will be -5. Combining order with distance (by using the distance score, and removing a candidate if order says so) has a very positive effect on the rank with maximal precision.

We also observe that the tags that contain the name candidate play an important role. For example, a headline $H1$ is most likely the entity name, even if it is far away from the id. Hence, our next scoring model assigns a score of 1 for “hiding tags” such as SMALL or STRIKE, a score of 2 for plain text, 3 for “highlighting tags” such as B or I, and 4 for headers such as H1. Any finer scale of styles did not seem to have any additional effect. We found that we achieve the best results if we remove all candidates that do not have a tag score of 4. This is what the scoring model style4 does. By combining all of these scoring models, we achieve very good precision already at rank 1. This combined model is what we will use for detail records.

**Models for free records.** The title feature cannot be applied to free records. However, a variant of order, which restricts the extraction to the first 3 text frames of each record, yielded good precision (shown as first3). The style score, likewise, helps. If we combine these two scoring models (by using the full style score as described above, and by removing any candidates beyond the 3rd text frame), we obtain very good precision at a very small rank already. This is the scoring model that we use in the experiments for free records.

B. OUTLIER DETECTION

We now discuss our choice of how to detect outliers in Phase 2. Our goal is to exclude names that are associated with many different ids, as we expect them to be general words rather than entity-specific names. Like in Appendix A, we rely on experiments to identify the best design choices.

We first ran Phase 1 on the entire corpus, and collected the ids with their name candidates. Figure 5 shows the number of ids that
each particular name occurs with on a log-log scale. Most names occur with only one id. However, there are names that occur with multiple ids, and not all names that occur with multiple ids are noise. For example, the names “plastics”, “amphetamine”, and “propionate” each occur with 10 ids. Figure 2 shows how often these names occur with each of their ids. Only “amphetamine” is a correct name for a chemical substance. It is clearly associated to one id, with the other occurrences being noise. “plastics” and “propionate” are more uniformly distributed across the ids, and are not names for chemical ids. Our goal is to distinguish a correct name such as “amphetamine” from the incorrect names.

**Schemes.** There are many different methods to detect outliers. These include the Z-test, the Grubbs test, the Dixon test, and the Box-plot rule (see, e.g., [12, 23] for surveys). We looked into all four methods, but none of them is applicable in our setting. The reason is that we need to find an outlier already for a very small set of distinct ids (on average 11) without any knowledge of the underlying probability distribution.

Thus, we formalized our intuitive definition of an outlier. An outlier for a name is the id that has the highest frequency, such that its frequency is above a minimal threshold and there is no second id with comparable frequency. This means that an id is an outlier if there is no second id, or if:

$$f_i > p n \land i \neq f_j$$

Here $f_k$ is the frequency of the id with the rank $k$, $n$ is the total number of id occurrences, and $i \in \mathbb{N}$ and $p \in [0, 1]$ are parameters. We now describe how we chose those parameters.

**Parameter tuning.** To find these parameters, we used again the sample of id occurrences from Appendix A. Each id in this sample has been mapped manually to the correct name, giving us a set of correct names. By contrast, we manually selected 100 general names per id type from the output of Phase 1 (for example, “Pet supplies” from the list of GTIN name candidates), giving us a set of incorrect names. We considered all ids for which the sample names occurred, giving us, for every correct and incorrect name, a number of id occurrences, and $p$ is the total number of id occurrences. For every correct name, we measured the precision and recall when classifying good and bad names. In general, precision increases with growing $i$ and $p$, and recall decreases. Our focus is on precision, and hence we required the precision to be at least 95%. By varying $i$ and $p$, we found that the combination $i = 3$ and $p = 20$ is a sweet-spot, which achieves a precision of 95% and a recall of 25%.

**C. NER MODULES**

We describe here one possible implementation of the NER modules $f_{\text{id}}$ and $f_{\text{name}}$ from Section 4 for $t \in \{\text{GTIN, CAS, Email, DOI}\}$. These modules are the input of our method, and any NER modules can be used in place of the ones described here.

**GTINs.** A GTIN contains 14 digits: one digit for the packaging level, 3 digits for the country, 4-7 digits for the company, and the remaining digits for the product. The last digit is a check digit. $f_{\text{id}}$ checks the sequence, and validates the check digit. $f_{\text{name}}$ is a validator that accepts the input string, if it starts with a letter or number, contains a word of at least 4 characters, and contains no more than 250 characters. Books have GTINs that start with “978”. To avoid that we take the author of a book as its title, $f_{\text{GTIN}}$ aggressively rejects candidates that look like author names or lists of author names. If the candidate string contains a given name or a single-letter abbreviation, or if more than one third of the tokens in the string are commas, the validator rejects it. We compiled a dictionary of first names from the 1990 US Census [31] and the Balie project [30], from which we removed some common words (such as “China”).

**CAS.** A CAS number consists of 3 parts, separated by hyphens, where the last part is a check digit. Hence, $f_{\text{id}}$ checks the syntactic form of the id, and validates the check digit. $f_{\text{name}}$ is a validator that accepts the input string if it contains a word of at least 4 characters and is not more than 250 characters long. It rejects a candidate if it contains a chemical formula or any character other than alphanumeric characters, brackets, quotes and hyphens.

**Email.** $f_{\text{id}}$ checks whether the input consists of a local part, followed by the @-sign and a domain name. $f_{\text{name}}$ has to recognize person names such as “John Smith”, “Smith, John”, and “Dr. John Smith”. The literature has developed sophisticated approaches to this end [7]. Here, we use a simple name finder that scans the input string and returns all substrings that follow the pattern “first middle last” or “last, first middle”. Here, last and middle match any, possibly hyphenated, capitalized word. $f_{\text{name}}$ maps each combination of first names from our dictionary of first names. Since we were only interested in personal email addresses and not in organizations, Web administrators, or service providers, $f_{\text{name}}$ returns only person names that overlap with the email address.

**DOIs.** $f_{\text{id}}$ just verifies whether the id follows the pattern of a numeric prefix followed by a slash and a sequence of characters. Document titles are often not marked up separately, but occur in plain text. Therefore, $f_{\text{id}}$ has to search candidates for the document title in the input string. There is some work on this task (known as bibliographic reference parsing, e.g., [7]), but these approaches can only extract from homogeneously formatted lists of references. In our case, in contrast, the names appear as an arbitrary sub-sequence of plain text. $f_{\text{name}}$ splits the string by the separators .-??. It accepts a substring as a candidate, if it contains at least 4 words. As with book titles, we exclude author names.

**D. REFERENCES**


