Crawling Deep Web Entity Pages

Yeye He*
Univ. of Wisconsin-Madison
Madison, WI 53706
heyeye@cs.wisc.edu

Dong Xin
Google Inc.
Mountain View, CA, 94043
dongxin@google.com

Venkatesh Ganti
Google Inc.
Mountain View, CA, 94043
vganti@google.com

Sriram Rajaraman
Google Inc.
Mountain View, CA, 94043
sriramr@google.com

Nirav Shah
Google Inc.
Mountain View, CA, 94043
nshah@google.com

ABSTRACT

Deep-web crawl is concerned with the problem of surfacing hidden content behind search interfaces on the Web. While many deep-web sites maintain document-oriented textual content (e.g., Wikipedia, PubMed, Twitter, etc.), which has traditionally been the focus of the deep-web literature, we observe that a significant portion of deep-web sites, including almost all online shopping sites, curate structured entities as opposed to text documents. Although crawling such entity-oriented content is clearly useful for a variety of purposes, existing crawling techniques optimized for document oriented content are not best suited for entity-oriented sites. In this work, we describe a prototype system we have built that specializes in crawling entity-oriented deep-web sites. We propose techniques tailored to tackle important subproblems including query generation, empty page filtering and URL deduplication in the specific context of entity oriented deep-web sites. These techniques are experimentally evaluated and shown to be effective.

Categories and Subject Descriptors: H.2.8 Database Application: Data Mining

Keywords: Deep-web crawl, web data, entities.

1. INTRODUCTION

Deep-web crawl refers to the problem of surfacing rich information behind the web search interface of diverse sites across the Web. It was estimated by various accounts that the deep-web has as much as an order of magnitude more content than that of the surface web [10, 14]. While crawling the deep-web can be immensely useful for a variety of tasks including web indexing [15] and data integration [14], crawling the deep-web content is known to be hard. The difficulty in surfacing the deep-web has inspired a long and fruitful line of research [3, 4, 5, 10, 14, 15, 17, 22, 23].

In this paper we focus on entity-oriented deep-web sites. These sites curate structured entities and expose them through search interfaces. Examples include almost all online shopping sites (e.g., ebay.com, amazon.com, etc.), where each entity is typically a product that is associated with rich structured information like item name, brand name, price, and so forth. Additional examples of entity-oriented deep-web sites include movie sites, job listings, etc. Note that this is to contrast with traditional document-oriented deep-web sites that mostly maintain unstructured text documents (e.g., Wikipedia, PubMed, etc.).

Entity-oriented sites are very common and represent a significant portion of the deep-web sites. The variety of tasks that entity-oriented content enables makes the general problem of crawling entities an important problem.

The practical use of our system is to crawl product entities from a large number of online retailers for advertisement landing page purposes. While the exact use of such entities content in advertisement is beyond the scope of this paper, the system requirement is simple to state: We are provided as input a list of retailers' websites, and the objective is to crawl high-quality product entity pages efficiently and effectively.

There are two key properties that set our problem apart from traditional deep-web crawling literature. First, we specifically focus on the entity-oriented model, because of our interest in product entities from online retailers, which are entity-oriented deep-web sites in most cases. While existing general crawling techniques are still applicable to some extent, the specific focus on entity-oriented sites brings unique opportunities. Second, a large number of entity sites (online retailers) are provided as input to our system, from which entity pages are to be crawled. Note that with thousands of sites as input, the realistic objective is to only obtain a *representative* content coverage of each site, instead of an *exhaustive* one. Ebay.com, for example, has hundreds of thousands of listings returned for the query "iphone"; the purpose of the system is not to obtain all iphone listings, but only a representative few of these listings for ads landing pages. This goal of obtaining representative coverage contrasts with traditional deep-web crawl literature, which tends to deal with individual site and focus on obtaining exhaustive content coverage. Our objective is more in line with the pioneering work [15], which also operates at the Web scale but focuses on general web content.

We have developed a prototype system that is designed specifically to crawl representative entity content. The crawling process is optimized by exploiting features unique to entity-oriented sites. In this paper, we will focus on describing important components of our system, including query generation, empty page filtering and URL deduplication.

Our first contribution is to show how query logs and knowledge bases (e.g., Freebase) can be leveraged to generate entity queries for crawling. We demonstrate that classical techniques for infor-

*Work done while author at Google, now at Microsoft Research.

WSDM'13, February 4–8, 2013, Rome, Italy.

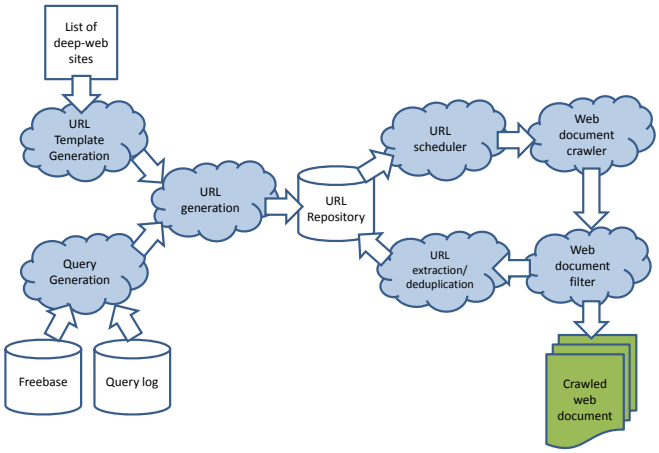


Figure 1: Overview of the entity-oriented crawl system

mation retrieval and entity extraction can be used to robustly derive relevant entities for each site, so that crawling bandwidth can be utilized efficiently and effectively (Section 5).

The second contribution of this work is a new empty page filtering algorithm that removes crawled pages that fail to retrieve any entities. This seemingly simple problem is nontrivial due to the diverse nature of pages from different sites. We propose an intuitive filtering approach, based on the observation that empty pages from the same site tend to be highly similar (e.g., with the same page layout and the same error message). In particular, we first submit to each target site a small set of queries that are intentionally "bad", to retrieve a "reference set" of pages that are highly likely to be empty. At crawl time, each newly crawled page is compared with the reference set, and pages that are highly similar to the reference set are predicted as empty and filtered out from further processing. This weakly-supervised approach is shown to be robust across sites on the Web (Section 6).

Additionally, we observe that the search result pages typically expose additional deep-web content that deserves a second round of crawling, which is an interesting topic that has been overlooked in the literature. In order to obtain such content, we identify promising URLs on the result pages, from which further crawling can be bootstrapped. Furthermore, we propose a URL deduplication algorithm that prevents URLs with near-identical results from being crawled. Specifically, whereas existing techniques use content analysis for deduplication which works only after pages are crawled, our approach identifies the semantic relevance of URL query segments by analyzing URL patterns, so that URLs with similar content that differ in non-essential ways (e.g., how retrieved entities are rendered and sorted) can be deduplicated. This approach is shown to be effective in preserving distinct content while reducing the bandwidth consumption (Section 7).

2. SYSTEM OVERVIEW

Deep-web sites	URL templates
ebay.com	*www.ebay.com/sch/i.html?**_nkw={query}**&_sacat=All-Categories*
chegg.com	*www.chegg.com/search/?**search_by={query}***
beso.com	*www.beso.com/classify?search_box=1&**keyword={query}***
...	...

Table 1: Example URL templates

In this section we explain each component of our system in turn at a very high level. The overall architecture of our system is illustrated in Figure 1.

URL template generation. At the top left corner the system takes a list of domain names of deep-web sites as input, and an example of which is illustrated in the first column of Table 1. The URL template generation component then crawls the home-pages of these sites, extracts and parses the web forms found on the home-pages, and produces URL templates. Example URL templates are illustrated in the second column of Table 1. Here the boldfaced "{query}" represents a wild-card that can be substituted by any keyword query (e.g., "iphone"); the resulting URL can be used to crawl deep-web content as if the web forms are submitted.

Query generation and URL generation. The query generation component at the lower left corner takes the Freebase [6] and query logs as input, outputs queries consistent with the semantics of each deep-web site (for example, query "iphone" may be generated for sites like amazon.com or ebay.com but not for tripadvisor.com).

Such queries can then be plugged into URL templates to substitute the "{query}" wild-card to produce final URLs, which will be stored in a central URL repository. URLs can then be retrieved from the URL repository and scheduled for crawling at runtime.

Empty page filter. It is inevitable that some URLs corresponding to previously generated queries will retrieve empty or error pages that contain no entity. Once pages are crawled, we move to the next stage, where pages are inspected to filter out empty ones. The process of filtering empty pages is critical (to avoid polluting downstream operations), but also non-trivial, for different sites indicate empty pages in disparate ways. The key insight here is that empty pages from the same site tend to be highly similar. So we intentionally retrieve a set of pages that are highly likely to be empty, and filter out any crawled pages from the same site that are similar to the reference set. Remaining pages with rich entity information can then be used for a variety of purposes.

URL extraction/deduplication. Additionally, we observe that a significant fraction of URLs on search result pages (henceforth referred to as "second-level URLs", to distinguish from the URLs generated using URL template, which are "first-level URLs") typically link to additional deep-web content. However, crawling all second-level URLs indiscriminately is wasteful due to the large number of second level URLs available. Accordingly, in this component, we filter out second-level URLs that are less likely to lead to deep-web content, and dynamically deduplicate remaining URLs to obtain a much smaller set of "representative" URLs that can be crawled efficiently. These URLs then iterate through the same crawling process to obtain additional deep-web content.

3. RELATED WORK

The aforementioned problems studied in this work have been explored in the literature to various extents. In this section, we will describe related work and discuss key differences between our approach in this work and existing techniques.

URL template generation. The problem of generating URL templates has been studied in the literature in different contexts. For example, authors in [4, 5] looked at the problem of identifying searchable forms that are deep-web entry points, from which templates can then be generated. The problem of parsing HTML forms for URL templates has been addressed in [15]. In addition, authors in [15, 20] studied the problem of assigning combinations of values to multiple input fields in the search form so that content can be retrieved from the deep-web effectively.

In our URL template generation component, search forms are parsed using techniques similar to what was outlined in [15]. However, our analysis shows that generating URL templates by enumerating values combination in multiple input fields can lead to an inefficiently large number of templates and may not scale to the number of websites that we are interested in crawling. As will be discussed in Section 4, our main insight is to leverage the fact that

for entity-oriented sites, search forms predominantly employ one text field for keyword queries, and additional input fields with good "default value" behavior. Our URL template generation based on this observation provides a tractable solution for a large number of potentially complex search forms without significantly sacrificing content coverage.

Query generation and URL generation. Prior art in query generation for deep web crawl focused on bootstrapping using text extracted from retrieved pages [15, 17, 22, 23]. That is, a set of seed queries are first used to crawl pages. The retrieved pages are analyzed for promising keywords, which are then used as queries to crawl more pages recursively.

There are several key reasons why existing approaches are not very well suited for our purpose. First of all, most previous work [17, 22, 23] aims to optimize coverage of *individual* sites, that is, to retrieve as much deep-web content as possible from one or a few sites, where success is measured by percentage of content retrieved. Authors in [3] go as far as suggesting to crawl using common stop words "a, the" etc. to improve site coverage when these words are indexed. We are in line with [15] in aiming to improve content coverage for a large number of sites on the Web. Because of the sheer number of deep-web sites crawled we trade off complete coverage of individual site for incomplete but "representative" coverage of a large number of sites.

The second important difference is that since we are crawling entity-oriented pages, the queries we come up with should be entity names instead of arbitrary phrases segments. As such, we leverage two important data sources, namely query logs and knowledge bases. We will show that classical information retrieval and entity extraction techniques can be used effectively for entity query generation. To our knowledge neither of these data sources has been very well studied for deep-web crawl purposes.

Empty page filtering. Authors in [15] developed an interesting notion of *informativeness* to filter search forms, which is computed by clustering *signatures* that summarize content of crawled pages. If crawled pages only have a few signature clusters, then the search form is *uninformative* and will be pruned accordingly. This approach addresses the problem of empty pages to an extent by filtering uninformative forms. However, this approach operates at the level of search form / URL template, it may still miss empty pages crawled using an informative URL template.

Since our system generates only one high-quality URL template for each site, filtering at the granularity of URL templates is likely to be ill-suited. Instead, our approach considers in this work filters at page level — it can automatically distinguishes empty pages from useful entity pages by utilizing intentionally generated bad queries. To our knowledge this simple yet effective approach has not been explored in the literature.

A novel page-level empty page filtering technique was described in [20], which labels a result page as empty, if either certain predefined error messages are detected from the "significant portion" of result pages (e.g., the portion of the page formatted using frames, or visually laid out at the center of the page), or a large fraction of result pages are hashed to the same value. In comparison, our approach obviates the need to recognize the significant portion of result pages, and we use content signature instead of hashing that is more robust against minor page differences.

URL deduplication. The problem of URL deduplication has received considerable attention in the context of web crawling and indexing [2, 8, 13]. Current techniques consider two URLs as duplicates if their content are highly similar. These approaches, referred to as *content-based URL deduplication*, proposes to first summarize page contents using content sketches [7] so that pages with similar content are grouped into clusters. URLs in the same cluster are then analyzed to learn URL transformation rules (for example, it can learn that `www.cnn.com/story?id=num` is equivalent to `www.cnn.com/story_num`).

Figure 2: A typical search interface (ebay.com)

In this paper, instead of looking at the traditional notion of page similarity at the content level, we view page similarity at the semantic level. That is, we view pages with entities from the same result set (but perhaps containing different portions of the result, or presenting with different sorting orders) as semantically similar, which can then be deduplicated. This significantly reduces the number of crawls needed, and is in line with our goal of obtaining representative content coverage given the sheer number of websites crawled.

Using semantic similarity, our approach can analyze URL patterns and deduplicate *before* pages are crawled. In comparison, existing content-based deduplication not only requires pages to be crawled first for content analysis, it would not be able to recognize the semantic similarity between URLs and would require billions of more URLs crawled.

Authors in [15] pioneered the notion of presentation criteria, and pointed out that crawling pages with content that differ only in presentation criteria are undesirable. Their approach, however, deduplicates at the level of search forms and cannot be used to deduplicate URLs directly.

4. URL TEMPLATE GENERATION

As input to our system, we are given a list of entity-oriented deep-web sites that need to be crawled. Our first problem is to generate URL templates for each site that are equivalent to submitting search forms, so that entities can be crawled directly using URL templates.

As a concrete example, the search form from ebay.com is shown in Figure 2, which represents a typical entity-oriented deep-web search interface. Searching this form using query "ipad 2" without changing the default value "All Categories" of the drop-down box is equivalent to using the URL template for ebay.com in Table 1, with wild-card "{query}" replaced by "ipad+2".

The exact technique that parses search forms is developed based on techniques proposed in [15], which we will not discuss in details in the interest of space. However, our experience with URL template generation leads to two interesting observations worth mentioning.

Our first observation is that for entity-oriented sites, the main search form is almost always on home pages instead of somewhere deep in the site. The search form is such an effective information retrieval paradigm that websites are only too eager to expose them. A manual survey suggests that only 1 out of 100 randomly sampled sites does not have the search form on the home page (www.arke.nl). This obviates the need to use sophisticated techniques to locate search forms deep in websites (e.g., [4, 5]).

The second observation is that in entity-oriented sites, search forms predominantly use one main text input fields to accept keyword queries (a full 93% of sites surveyed have exactly one text field to accept keyword queries). At the same time, other non-text input fields exhibit good "default value" behavior (94% of sites out of the 100 sampled are judged to be able to retrieve entities using default values without sacrificing coverage).

Since enumerating values combination in multiple input fields

Deep-web sites	sample queries from query logs
ebay.com	cheap iPhone 4, lenovo x61, ...
bestbuy.com	hp touchpad review, price of sony vaio, ...
booking.com	where to stay in new york, hyatt seattle review, ...
hotels.com	hotels in london, san francisco hostels, ...
barnesandnobel.com	star trek books, stephen king insomnia, ...
chegg.com	harry potter book 1-7, dark knight returns, ...

Table 2: Example queries from query logs

(e.g., [15, 20]) can lead to an inefficiently large number of templates and may not scale to the number of websites that we are interested in crawling, we leverage aforementioned observation to simplify URL template generation by producing one template for each search form. Specifically, only the text field is allowed to vary (represented by a placeholder "{query}") while others fields will take on default values, as shown in Table 1. In our experience this provides a more tractable way to generate templates than the previous multi-value enumeration approach that works well in practice. We will not discuss details of template generation any further in the interest of space.

5. QUERY GENERATION

After obtaining URL templates for each site, the next step is to fill relevant keyword query into the "{query}" wild-card to produce final URLs. The challenge here is to come up with queries that match the semantics of the sites. A dictionary-based brute force approach that sends every known entity to every site is clearly inefficient – crawling queries like "ipad" on tripadvisor.com does not make sense, and will most likely result in an empty/error page.

We utilize two data sources for query generation: query logs and knowledge-bases. Our main observation here is that classical techniques in information retrieval and entity extraction are already effective in generating entity queries.

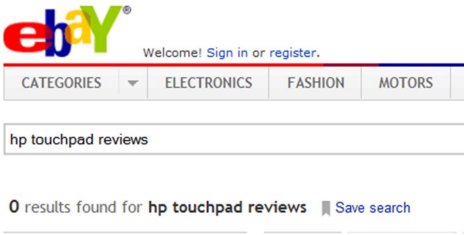
5.1 Entity extraction from query logs

Query logs refer to keyword queries searched and URLs clicked on search engines (e.g., Google). Conceptually query logs make a good candidate for query generation in deep-web crawls — queries with high number of clicks to a certain site is an indication of the relevance between the query and the site, submitting such queries through the site's search interface for deep-web crawl thus makes intuitive sense.

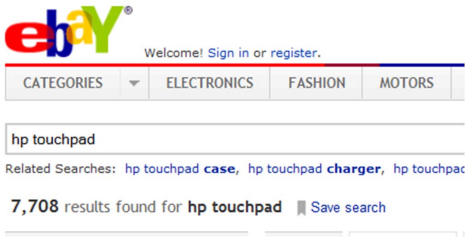
We used Google's query logs with the following normalized form $< keyword\_query, url\_clicked, num\_times\_clicked >$. To filter out undesirable queries (e.g., navigational queries), we only consider queries that are clicked for at least 2 pages in the same site, for at least 3 times.

Although query logs contain rich information, it is also too noisy to be used directly for crawling. Specifically, queries in the query logs tend to contain extraneous tokens in addition to the central entity of interest. However, it is not uncommon for the search interface on deep-web sites to expect only entity names as queries. Figure 3 serves as an illustration of this problem. When feeding a search engine query "HP touchpad reviews" into the search interface on deep-web sites, (in this example, ebay.com), no results are returned (Figure 3a), while searching using only the entity name "HP touchpad" retrieves 6617 such products (Figure 3b).

This issue above is not isolated. On the one hand, search engine queries typically contain tokens in addition to entity mentions, which either specify certain aspects of entities of interest (e.g., "HP touchpad review", "price of chrome book spec"), or are simply natural language fragments (e.g., "where to buy iPad 2", "where to stay in new york"). On the other hand, many search interfaces only expect clean entity queries. This is because a significant portion of



(a) search with "hp touchpad reviews"



(b) search with "hp touchpad"

Figure 3: An example of Keyword-And based search interface

entity sites employ the simple Keyword-And mechanism, where *all* tokens in the query have to be matched in a tuple before the tuple can be returned (thus the no match problem in Figure 3b). Even if the other conceptual alternative, Keyword-Or is used, the presence of extraneous tokens can promote spurious matches and lead to less desirable crawls.

We reduce the aforementioned problem to entity extraction from query logs. Or to view it the other way, we clean the search engine queries by removing tokens that are not entity related (e.g., removing "reviews" from "HP touchpad reviews", or "where to stay in" from "where to stay in new york", etc.).

In the absence of a comprehensive entity dictionary, it is hard to tell if a token belongs to (ever-growing) entity names and their name variations, abbreviations or even typos. At the same time, the diverse nature of the query logs makes it all the more valuable, for it captures a wide variety of entities and their name variations.

Inspired by an influential work on entity extraction from query logs [18], we first identify common patterns in query logs that are clearly not entity related (e.g., "reviews", "specs", "where to stay in" etc.) by leveraging known entities. Query logs can then be "cleaned" to extract entities by removing such patterns.

Specifically, we first obtained a dump of the Freebase data [6] — a manually curated repository with about 22M entities. We then find the maximum-length subsequence in each search engine query that matches Freebase entities as an entity mention. The remaining tokens are treated as entity-irrelevant prefix/suffix. We aggregate distinct prefix/suffix across the query logs to obtain common patterns ordered by their frequency of occurrences. The most frequent patterns are likely to be irrelevant to entities and need to be cleaned.

EXAMPLE 1. *Table 2 illustrates the sample queries with mentions of Freebase entity names underlined. Observe that this entity recognition this way is not perfect. For example, the query "where to stay in new york" for booking.com has two matches with Freebase entities, the match of "where to" to a musical release with that name, and the match of "new york" as city name. Since both matches are of length two, we obtain the false suffix "stay in new york" (with an empty prefix) and the correct prefix "where to stay in" (with an empty suffix), respectively. However, when all the prefix/suffix in the query logs are aggregated, the correct prefix "where*

Deep-web sites	sample queries from query logs
ebay.com	iPhone 4, lenovo, ...
bestbuy.com	hp touchpad, sony vaio, ...
booking.com	where to, new york, hyatt, seattle, review, ...
hotels.com	hotels, london, san francisco, ...
barnesandnobel.com	star trek, stephen king, ...
chegg.com	harry potter, dark knight, ...

Table 3: Example entities extracted for each deep-web site

to stay in" occurs much more frequently and should clearly stand out as a entity irrelevant pattern.

Another potential problem is that Freebase may not contain all possible entities. For example in the query "hyatt seattle review" for booking.com, the first two tokens "hyatt seattle" refer to the Hyatt hotel in Seattle, which however is absent in Freebase. Using Freebase entities "hyatt" (a hotel company), and "seattle" (a location) will be recognized separately. However, with prefixes/suffixes aggregation, the suffix "review" is so frequent across the query logs such that it will be recognized as an entity-irrelevant pattern. This can be used to clean the query to produce entity "hyatt seattle".

Our experiments using Google's query log (to be discussed in Section 8) will show that this simple approach of entity extraction by pattern aggregation is effective in producing entity queries.

5.2 Entity expansion using knowledge-bases

While query logs provide a good set of initial seed entities, its coverage for each site depends on the site's popularity as well as the item's popularity (recall that the number of clicks is used to predict the relevance between the query and the site). Even for highly popular sites, there is a long tail of less popular items which may not be captured by query logs.

On the other hand, we observe that there exists manually curated entity repositories (e.g., Freebase), that maintain entities in certain domains with very high coverage. For example. Freebase contains comprehensive lists of city names, books, car models, movies, etc. Such categories, if matched appropriately with relevant deep-web sites, can be used to greatly improve crawl coverage. For example, names of all locations/cities can be used to crawl travel sites (e.g., tripadvisor.com, booking.com), housing sites (e.g., apartmenthomeliving.com, zillow.com); names of all known books can be useful on book retailers (amazon.com, barnesandnoble.com), book rental sites (chegg.com, bookrenter.com), so on and so forth. In this section, we consider the problem of expanding the initial set of entities using Freebase.

Recall that we can already extract Freebase entities from the query logs for each site. Table 3, for example, contains lists of entities extracted from the sample queries in Table 2. Thus, for each site, we need to bootstrap from these seed entities to expand to Freebase entity "types" that are relevant to each site's semantics.

We borrow classical techniques from information retrieval: if we view the multi-set of Freebase entity mentions for each site as a document, and the list of entities in each Freebase type as a query, then the classical *term-frequency, inverse document frequency* (TF-IDF) ranking can be applied.

For each Freebase type, we use TF-IDF to produce a ranked list of deep-web sites by their similarity scores. We then "threshold" the sorted list using a relative score. That is, we include all sites with scores above a fixed percentage, τ, of the highest similarity score in the same Freebase type as matches. Empirically results in Section 8 show that setting $\tau = 0.5$ achieves good results and is used in our system. This approach is significantly more effective than other alternatives like Cosine or Jaccard Similarity [21], with precision reaching 0.9 for $\tau = 0.5$.

6. EMPTY PAGE FILTERING

Once the final URLs are generated, pages can be crawled in a fairly standard manner. The next important issue that arises is to filter empty pages with no entity in them, in order to avoid polluting downstream pipelines. However, different sites can display disparate error messages, from textual messages (e.g., "sorry, no items is found", "0 item matches your search", etc.), to image-based error messages. While such messages are easily comprehensible for humans, it is difficult to detect automatically across all different sites. The presence of dynamically generated ads content further complicates the problem of detecting empty pages.

We develop a page-level filtering approach that filters out crawled pages that fail to retrieve any entities. Our main observation is that empty pages from the same site are typically extremely similar to each other, while empty pages from different sites are normally very different. Ideally we should obtain "sample" empty pages for each deep-web site, with which newly crawled pages can be compared. To do so, we generate a set of "background queries", that are long strings of arbitrary characters that lack any semantic meanings (e.g., "zzzzzzzzzzzz", or "xyzxyzxyzxyz"). Such queries, when searched on deep-web sites, will almost certainly generate empty pages. In practice, we generate N (10 in our experiments) such background queries in order to be robust against the rare case where a bad "background query" accidentally matches some records and produces a non-empty page. We then crawl and store the corresponding "background pages" as the reference set of empty pages. At crawl time, each newly crawled page is compared with background pages to determine if the new page is actually empty.

Our content comparison mechanism uses a signature based page summarization techniques also used in [15]. The signature is essentially a set of tokens that are descriptive of the page content, but also robust against minor differences in page content (e.g., dynamically generated advertisements). [1] We then calculate the Jaccard Similarity between the signature of the newly crawled page and the "background pages", as defined below.

DEFINITION 1. *[21] Let S_{p_1} and S_{p_2} be the sets of tokens representing the signature of the crawled page p_1 and p_2. The **Jaccard Similarity** between S_{p_1} and S_{p_2}, denoted $Sim_{Jac}(S_{p_1}, S_{p_2})$, is defined as $Sim_{Jac}(S_{p_1}, S_{p_2}) = \frac{S_{p_1} \cap S_{p_2}}{S_{p_1} \cup S_{p_2}}$*

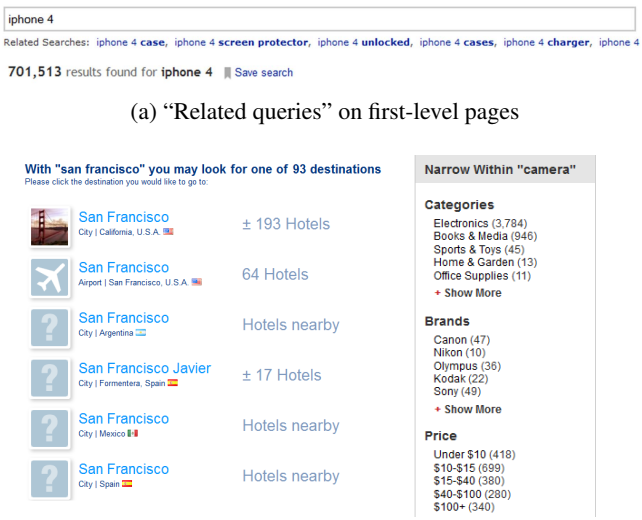
The similarity scores are averaged over the set of N "background pages", and if the average score is above certain threshold θ, we label the newly crawled page as empty. As we will show in experiments, this approach is very effective in detecting empty pages across different websites (with an overall precision of 0.89 and a recall of 0.9).

7. SECOND-LEVEL CRAWL

7.1 The motivation for second level crawl

We observe that the first set of pages crawled using URL templates often contain URLs that link to additional deep-web contents. In this work, we refer to the first set of pages obtained through URL templates as "first-level pages" (because they are one click away from the homepage), and those pages that are linked from first-level pages as "second-level pages" (and the corresponding URLs "second-level URLs"). There are at least a few common cases in which crawling second-level pages can be useful.

[1] Our signatures are generated using a proprietary method also used in [15], the details of which is beyond the scope of this paper. In principle well-known content summarization techniques like [7, 16] can be used in place.



(a) "Related queries" on first-level pages

(b) Disambiguation pages

(c) Faceted search

Figure 4: Examples for which second-level crawl is desirable

In the first category, when a query is searched, an additional set queries relevant to the original query are displayed. This is known as query expansion [19], which aims to help users to reformulate their queries. Figure 4a is a screen-shot of such an example. When the original query "iphone 4" is searched, the returned page displays queries related to the original query, like "iphone 3gs", "iphone 4 case", etc. Since these queries are maintained and suggested by individual sites, they can in most cases lead to valid deep-web content, thus improving content coverage.

Figure 4b shows another scenario for second-level crawl. In this example, when the query "san francisco" is searched, a disambiguation page is returned containing multiple cities with that name. Following second-level URLs on the disambiguation page is needed in order to expose rich deep-web content (lists of hotels in this case).

Second-level crawls are also desirable for sites that employ the "faceted search/browsing" paradigm [11]. In faceted search, returned entities are presented in a multi-dimensional, faceted manner. In the example shown in Figure 4c when "camera" is searched, a "multi-faceted" entity classification is returned along with a large set of results, to allow users to drill-down using additional criteria (e.g., category, brand, price etc.). Conceptually, this is equivalent to adding an additional predicate to the original entity retrieval query, which amounts to a new query. Accordingly, such URLs are also desirable for further crawling.

We also note that crawling second-level URLs using our simplified URL template with default values can achieve similar effects as exhaustively template generation by enumerating all possible values combinations (Section 4). In this example website in Figure 4c where query "camera" is searched with default category "All-categories", the second-level URL for category "Electronics" actually corresponds to searching "camera" with sub-category "Electronics" selected.

Furthermore, while the previous multi-field enumeration approach would search for a query with all value combination e.g., searching for "camera" with the (inconsistent) sub-categories "pets" or "furnitures" that would lead to empty pages, our approach is data driven — typically faceted search links are exposed only when there exists deep-web content matching the searching criteria (category "pets" will not display when "camera" is searched). As a result, our approach is more likely to retrieve content successfully.

www.buy.com/sr/searchresults.aspx?qu=gps&sort=4&from=7&mfgid=-652&page=1
www.buy.com/sr/searchresults.aspx?qu=gps&sort=5&from=7&mfgid=-652&page=1
...
www.buy.com/sr/searchresults.aspx?qu=gps&sort=4&from=8&mfgid=-652&page=1
www.buy.com/sr/searchresults.aspx?qu=gps&sort=4&from=9&mfgid=-652&page=1
...
www.buy.com/sr/searchresults.aspx?qu=gps&sort=4&from=7&mfgid=-652&page=2
www.buy.com/sr/searchresults.aspx?qu=gps&sort=4&from=7&mfgid=-652&page=3
...
www.buy.com/sr/searchresults.aspx?qu=gps&sort=4&from=7&mfgid=-1755&page=1
www.buy.com/sr/searchresults.aspx?qu=gps&sort=5&from=7&mfgid=-1755&page=1
...
www.buy.com/sr/searchresults.aspx?qu=tv&sort=4&from=7&mfgid=-1001&page=1
www.buy.com/sr/searchresults.aspx?qu=tv&sort=5&from=7&mfgid=-1001&page=1
...

Table 4: Duplicate cluster of second-level URLs

7.2 URL extraction and filtering

While *some* second-level URLs are desirable, not all second-level URLs should be crawled for efficiency as well as quality reasons. First, there are on average a few dozens second-level URLs for each first-level page crawled. Crawling all these second-level URLs becomes very costly at large scale. Furthermore, a significant portion of these second-level URLs are in fact entirely irrelevant with no deep-web content (many URLs are static and navigational, for example browsing URLs, login URLs, etc.), which need to be filtered out.

We filter URLs by using *keyword-query arguments*. Keyword-query argument is the URL argument immediately prior to the "{query}" wild-card in URL templates. For example, in Table 1, "_nkw=" is the keyword-query argument for ebay.com. Similarly we have "search_by=" for chegg.com, and "keyword=" for beso.com. The presence of the keyword-query argument in a given domain is in general indicative that the page is dynamically generated with keyword predicates and is thus likely to be deep-web related. We observe that filtering URLs by keyword-query arguments significantly reduces the number of URLs — typically by a factor of 3-5 — while still preserves desirable second-level URLs that lead to deep-web content.

7.3 URL deduplication

We observe that after URL filtering, there are groups of URLs that are different in their text string but really lead to similar or nearly identical deep-web content. In this section we propose to deduplicate second-level URLs to further reduce the number of URLs that need to be crawled.

To take a closer look at second-level URLs, we use URLs extracted from buy.com in Table 4 as an example to illustrate. Dynamical URLs generated by deep-web form submission generally follow the W3C URI recommendations [1], where the part of URL string after "?" is the so-called *query string*, and each component separated by "&" (or ";") is a *query segment* that consists of a pair of *argument* and *value* connected by an "=".

The observation here is that each query segment typically corresponds to a query predicate. Take the first URL in Table 4 as an example, the query segment "qu=gps" indicates that returned entities should contain the keyword "gps". "Sort=4" specifies that the list of entities should be sorted by price from low to high (where 4 is an internal encoding for that sorting criterion); "from=7" is an internal tracking parameter to record which URL was clicked that leads to this page; "mfgid=-652" is a predicate that selects manufacturer Garmin (where -652 is again an internal encoding), and finally "page=1" retrieves the first page of matching entities (typically each page only presents a limited number of entities, thus not all results can be displayed on one page). While the exact URL encodings of query strings vary wildly from site to site, such map-

Figure 5: Second-level URLs from different sorting criteria

pings from query segments to logical query predicates generally exist across different sites.

In this particular example, if this URL query string is to be written in SQL, it would correspond to the query below:

```
SELECT * FROM db
WHERE description LIKE '%gps%'
AND manufacturer = 'Garmin'
ORDER BY price DESC
LIMIT 20; -- number of entities per page
```

The second URL in Table 4 differs from the first one in the segment "sort=5", which sorts entities by release date. They actually correspond to different sorting tabs in the result page, as illustrated in Figure 5, and there exist many additional sorting criteria, each of which corresponds to a second-level URL.

Recall that we aim to recover a representative content coverage of each site — obtaining exhaustive coverage for a large number of site is unrealistic and also unnecessary. With that in mind, we only need to crawl one of these two example URLs (and other similar URLs with different sorting criteria) discussed above by "deduplicating" them — after all, crawling entities with similar properties but different sorting criteria only produce marginal benefit.

Similarly, the third and fourth URL in Table 4 differ from the first URL in the segment "from=". This is only for internal tracking purposes so that source of the click can be identified. While the URL strings are different in the "from=" part, the content they lead to are identical and thus also need to be deduplicated.

Lastly, the fifth and sixth URL in the Table differ from the first URL in the "page=" segment. This is to retrieve different portion of the result set as the number of entities presented on each page tends to be limited. Again with the goal of recovering representative content coverage, we would want to avoid iterating over the complete result set by crawling all results that span different pages with the use of URL deduplication.

Given this requirement, existing content-based URL deduplication (e.g., [2, 8, 13]) is clearly insufficient. To see why this is the case, consider the first and second URL in Table 4 that retrieves the same result set but use different sorting criteria. Since the result set can span multiple pages, a different sorting order can produce a totally different result page. Existing techniques that analyze content similarity will not be able to recognize the semantic similarity between these two pages. Similarly content based deduplication will treat the fifth and sixth URL as distinct pages instead of semantic duplicates, thus wasting significant crawling bandwidth. In this paper we propose an approach that analyzes URL argument patterns and deduplicates URLs even before any pages are crawled.

7.3.1 Pre-crawl URL deduplication

In this work we propose to analyze the patterns of second-level URLs before they are crawled, and use a new definition to capture both content similarity as well as the similarity in the semantics of queries that are used to retrieve deep-web content.

First, we categorize query segments into three groups, (1) *selection segments* are query segments that correspond to selection predicates and can affect the set of result entities (e.g., "qu=gps" and "mfgid=-652" in the example URLs discussed above, which are essentially predicates in the `where` clause of the SQL query); (2) *presentation segments* are query segments that do not change the result set, but only affect how the set of entities are presented (e.g., "sort=4" or "page=1" in the example URL); and lastly, (3) *content irrelevant segments* are query segments that have no immediate impact on the result entities (e.g., the tracking segment "from=7").

We then define two URLs as **semantic duplicates** if the corresponding selection queries have the same set of selection segments. More specifically, if queries corresponding to two URL strings return the same set of entities, then irrespective of how the entities are sorted or what portion of result set are presented, the two URLs are considered to be duplicates to each other. We can alternatively state that two URLs are considered as semantic duplicates if they differ only in content irrelevant segments or presentation segments.

While the reason of disregarding "content irrelevant segments" are straightforward, the rationale behind ignoring *presentation segments* goes back to our goal of obtaining "representative coverage" for each site. Exhaustively crawling the complete result set in different sorting orders provides marginal benefits; crawling one page for each distinct set of selection predicates is deemed sufficient.

EXAMPLE 2. *We use example URLs in Table 4 to illustrate our definition of semantic duplicates. The first group of URLs all correspond to the same selection predicates (i.e., "qu=gps" and "mfgid=-652") but differ only in content irrelevant segments ("from="), or presentation segments ("sort=", "page="). Crawling any one URL from this group will provide representative coverage.*

On the other hand, URLs from the first group and second group differ in selection segment "mfgid=?", where "mfgid=-652" represents "Garmin" while "mfgid=-1755" is for "Tomtom". The selection queries would retrieve two different sets of entities, thus should not be considered as semantic duplicates.

It can be seen that our *semantics-based URL deduplication* is a more general notion that goes beyond simply content-based similarity. Our approach is not based on any content analysis. Rather, it hinges on the correct identification of the categories of query segments by URL pattern analysis.

On the high level, our approach is based on two key observations. First, search result pages in the same site are likely to be generated from the same template and are thus highly *homogeneous*. That is, for the same site, the structure, layout and content of result pages share much similarity (which include deep-web URLs embedded in result pages that we are interested in).

Second, given page homogeneity, we observe that almost *all* result pages in the same site share certain presentation logics or other content-irrelevant functionalities. In the example of URLs extracted from buy.com we discussed above, almost all result pages can sort entities by price, or advance to the second page of entities in the result set. Each page also implements the site-specific click-tracking functionality. These presentation logics translates to the same presentation segments ("sort=4" and "page=2"), and content-irrelevant segments ("from=7"), respectively, which can be found in almost every result page.

With these observations, we propose to take all URLs embedded in a result page as a unit of analysis. We then aggregate the frequency of query segments and identify segments that commonly occur across many different pages in the same site. The fact that these query segments in almost all pages indicates that they are not specific to the input keyword query, and are thus likely to be either presentational (sorting, page number, etc.), or content irrelevant (internal tracking, etc.).

On the other hand, selection segments, like manufacturer name ("mfgid=-652" for "Garmin") in the previous example, are much more sensitive to the input queries. Only when queries related to GPS are searched, will the segment representing manufacturer "Garmin" ("mfgid=-652") appear. Pages crawled for other entities

(e.g., laptops or cameras or furnitures) are likely to contain a different set of query segments for manufacturers. A specific query segment (argument/value pair) is likely to exist on *some*, but not *all* crawled pages.

To capture this intuition we define a notion of prevalence at the query segment (argument/value pair) level and also at the argument level.

DEFINITION 2. *Let $\mathcal{P}_s$ be the set of search result pages from the same site s, and $p \in \mathcal{P}_s$ be one such page. Further denote $D(p)$ as the set of query segments (argument-value pairs) in second-level URLs extracted from p, and $D(\mathcal{P}_s) = \cup_{p \in \mathcal{P}_s} D(p)$ as the union of all possible segments.*

*The **prevalence** of an argument-value pair (a, v), denoted as $r(a, v)$, is $r(a, v) = \frac{|\{p | p \in \mathcal{P}_s, (a,v) \in D(p)\}|}{|\mathcal{P}_s|}$.*

*The **prevalence** of an argument a, denoted as $r(a)$, is the average prevalence value of argument-value pairs with the same argument a, or $r(a) = \frac{\sum_{(a,v) \in D(\mathcal{P}_s)} r(a,v)}{|\{(a,v) | (a,v) \in D(\mathcal{P}_s)\}|}$.*

Intuitively, the prevalence of an argument-value pair specifies the ratio of pages from site s that contain the argument-value pair in the second-level URLs. For example if the URL with the argument-value pair "sort=4" that sorts items by price exist in 90 out of 100 result pages from buy.com, its prevalence is 0.9. The prevalence of an argument is just the average over all possible values of this argument (the prevalence of argument "sort=", for example, is averaged from "sort=1", "sort=2", etc.). The average prevalence score at argument level is a more robust indicator of the prevalence of an argument.

Since query segments with arguments that have a high prevalence score tend to be either content irrelevant, or presentational, we set a threshold score θ, such that any argument with prevalence higher than θ are considered to be *semantically-irrelevant* (for example if "sort=" has a high enough prevalence score, all URLs that differ only in query segments "sort=?" can be deduplicated, because crawling the same set of entities with all possible sorting orders are deemed as unnecessary). On the other hand, if an query segment's argument has prevalence lower than θ it is assumed to be a selection segment, which is relevant to the result set of entities.

After identifying semantically-irrelevant arguments, second-level URLs from the same site can then be partitioned by disregarding these irrelevant arguments into different buckets, as in Table 4. URLs in the same bucket are treated as semantic duplicates, and only one URL in the same partition needs to be crawled[2].

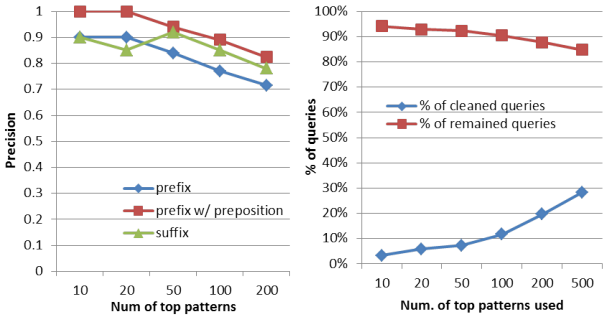
To sum up, our deduplication algorithm takes second-level URLs on the same result page as a unit of analysis instead of analyzing URLs individually. This has the advantage of providing more context for analysis and producing robust prediction through aggregation. Note that our analysis is possible because result pages returned from the search interface tend to be homogeneous. Web pages in general are much more heterogeneous and this page-oriented URL deduplication may not work well in a general web crawl setting.

8. EXPERIMENTS

In this section, we will discuss key experimental results for the query generation, empty page filtering, and URL deduplication, respectively.

Query extraction from query logs. Recall that in Section 5.1, we first identify frequent patterns that are likely to be entity irrelevant using pattern aggregation, which are then used to extract entities from query logs.

[2]Note that we pick one random URL to crawl rather than removing irrelevant query segments, because there are cases in which removing query segments breaks the URL and lead to error pages.



(a) Impact of removing top patterns (b) Top pattern precision

Figure 6: Removing top patterns

In this experiment, we used 6 month's worth Google's query logs, and entities in Freebase as seed entities. In order to evaluate whether patterns produced by our approach is truly entity-irrelevant or not, we asked a domain expert to manually label top 200 patterns, as correct predictions (irrelevant to entities) or incorrect predictions (relevant to entities), by looking at 5 sample queries with and without each pattern. We do not observe much ambiguities since only top patterns were labeled.

Table 5 lists the top 10 most frequent prefix and suffix patterns we produced. The presence of preposition in the prefix is also good indication that the prefix is not relevant to any entity so such patterns is also aggregated in the second column. In this table, patterns that are relevant to entities (and are thus mislabeled) are underlined. It is clear from the table that most patterns found this way are indeed not related to entity mentions. Removing such patterns allows us to obtain a cleaner set of entity names ranging from song/album names ("lyrics", "lyrics to", etc.), location/attraction names ("pictures of", "map of", "where to stay in" etc.), to various product names ("review", "price of", etc.).

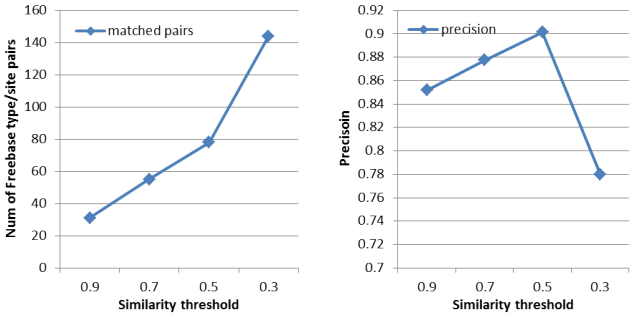
Top prefix	Top prefix with prep.	Top suffix
how	lyrics to	lyrics
watch	pictures of	download
samsung	list of	wiki
download	map of	torrent
is	history of	online
which	lyrics for	video
free	pics of	review
the	lyrics of	mediafire
best	facts about	pictures

Table 5: Top 10 common patterns

In Figure 6a we summarize the precision for top 10, 20, 50, 100 and 200 patterns. Not surprisingly, the precision decreases as more number of patterns are included.

Figure 6b shows the percentage of queries in the query logs that contain top patterns, thus illustrating the impact of query cleaning using top patterns. At top 200 about 19% of the queries will be cleaned using our approach. The total number of distinct queries also reduces by 12%, for after cleaning some queries become duplicate with existing ones. This reduces the number of unnecessary crawls, and also improves the quality of the resulting pages (when extraneous information like "review", "price of" is included spurious matches are promoted and the search results are less clean accordingly).

Entity expansion using Freebase. In Section 5.2, we discussed entity expansion using Freebase using extracted seed entities. At



(a) Num. of matched pairs (b) Precision of matched pairs

Figure 7: Effects of different score threshold

the highest level Freebase data are grouped into "domains", or categories of relevant topics, like automotive, book, computers, etc., in the first column in Table 6. Within each domain, there is a list of "types," each of which consists of manually curated entities with names and relationships. For example, the domain film contains types including film (list of film names), actor (list of actor names) and performance (which actor performed in which film relation).

Domain name	Top types	# of types	# of instances
Automotive	trim_level, model_year	30	78,684
Book	book_edition, book, isbn	20	10,776,904
Computer	software, software_comparability	31	27,166
Digicam	digital_camera, camera_iso	18	6,049
Film	film, performance, actor	51	1,703,255
Food	nutrition_fact, food, beer	40	66,194
Location	location, geocode, mailing_address	167	4,150,084
Music	track, release, artist, album	63	10,863,265
TV	tv_series_episode, tv_program	41	1,728,083
Wine	wine, grape_variety_composition	11	16,125

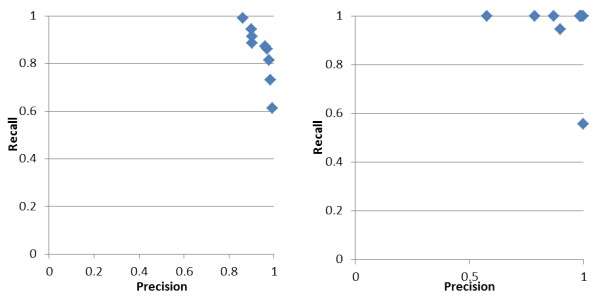
Table 6: Freebase domains used for query expansion

Domain:type name	matched deep-web sites
Automotive:model_year	stratmosphere.com, ebay.com
Book:book_edition	christianbook.com, netflix.com, barnesandnoble.com
Computer:software	booksprice.com
Digicam:digital_camera	rozetka.com.ua, price.ua
Food:food	fibergourmet.com, tablespoon.com
Location:location	tripadvisor.com, hotels.com, apartmenthomeliving.com
Music:track	netflix.com, play.com, musicload.de
TV:tv_series_episode	netflix.com, cafepress.com
Wine:wine	wineenthusiast.com

Table 7: Sample Freebase matches (incorrect ones are underlined)

Since not all Freebase domains are equally applicable for deep-web crawl purposes (e.g., chemistry ontology), for human evaluation purposes we only focus on the 5 largest Freebase types in 10 widely applicable domains listed in Table 6. We also restrict our attention in 100 high-traffic online retailers that we are most interested in. We asked a domain expert to manually label, for each pair of <website, Freebase type> result, whether the matching is correct. That is, whether entities in the Freebase type can be used to retrieve valid product entities from the matching website. In order not to overestimate the matching precision, we intentionally ignore matches for general-purpose sites that span multiple product categories (ebay.com, nextag.com, etc.), by considering such matches as neither correct nor incorrect.

Figure 7a shows the total number of matched Freebase-type / site pairs and Figure 7b illustrates the matching precision. As we can see, while the number of matched pairs increases as the threshold decreases, there is a significant drop in matching precision when



(a) Vary score threshold (b) Using different sites

Figure 8: Precision/recall of empty page filtering. (8a): Each dot in the graph represents precision/recall of a different score threshold; (8b): Each dot represents results from a different website.

threshold decreases from 0.5 to 0.3. Empirically a threshold of 0.5 is used in our system.

Table 7 illustrates example matches between Freebase types and deep-web sites. Top 3 matches of the largest type in each Freebase domain are listed (in some cases only 1 or 2 matches are above the relative threshold). Overall this produces good quality matches to Freebase types, which in turn greatly improves crawling coverage.

Empty page filtering. To evaluate the effectiveness of our empty page filtering approach (Section 6), we randomly selected 10 deep-web sites from a list of high-traffic sites (namely, booking.com, cdiscount.com, ebay.com, ebay.com.uk, marksandspencer.com, nordstrom.com, overstock.com, screwfix.com, sephora.com, tripadvisor.com), and manually identified their respective error messages (e.g., "Your search returned 0 items" is the error message used by ebay.com). This manual approach, while accurate, does not scale to a large number of websites. It does, however, enables us to build the ground truth — any page crawled from the site with that particular message is regarded as an empty page (negative instances), and pages without such message are treated as non-empty pages (positive instances). We can then evaluate the precision and recall of our algorithm, where precision and recall are defined as

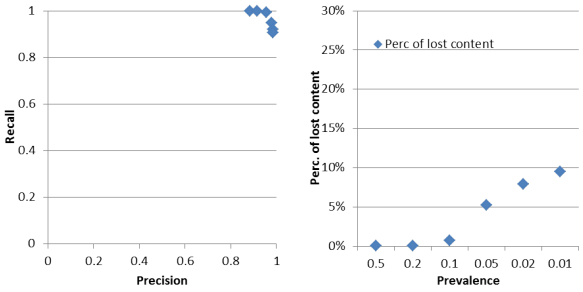
$$precision = \frac{|\{\text{pages predicted as non-empty}\}| \cap |\{\text{pages that are non-empty}\}|}{|\{\text{pages predicted as non-empty}\}|}$$

$$recall = \frac{|\{\text{pages predicted as non-empty}\}| \cap |\{\text{pages that are non-empty}\}|}{\{|\text{pages that are non-empty}|\}}$$

Figure 8a shows the precision/recall graph of empty page filtering when varying the threshold score θ from 0.4 to 0.95. We observe that setting threshold to a low value, say 0.4, achieves high precision (predicted non-empty pages are indeed non-empty) at the cost of significantly reducing recall to around only 0.6 (many non-empty pages are mistakenly labeled as empty because of the low threshold). At threshold 0.85 the precision and recall are 0.89 and 0.9, respectively, which is a good empirical setting that we used in our system.

Figure 8b plots the precision/recall of individual deep-web site for empty page filtering. Other than a cluster of points at the upper-right corner, representing sites with almost perfect precision/recall, there is only one site with relative low precision and another one with relative low recall.

URL deduplication. To understand the effectiveness of our semantic URL deduplication technique (Section 7.3), we used the same set of 10 entity sites used in empty page filtering, and manually label all URL arguments above the threshold 0.01 as either semantically relevant or irrelevant for deduplication purposes. Note that we cannot afford to inspect all possible arguments, because websites can typically use a very large number of arguments in URLs. For example, we found 1471 different arguments from overstock.com, 1243 from ebay.co.uk, etc. Furthermore, ascertaining semantic relevance of arguments that appear very infrequently can



(a) Argument level results (b) URL level recall

Figure 9: Precision/recall of URL deduplication. (9a): Each dot represents argument precision/recall at different prevalence value; (9b): Each dot represents lost URL at different prevalence value.

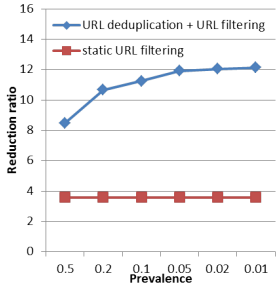


Figure 10: Reduction ratio of URL deduplication

be increasingly hard. As a result we only evaluate arguments with prevalence score of at least 0.01, and identified a total of 152 arguments that are semantically relevant.

Figure 9a shows the precision/recall of URL deduplication at the argument level. Each data point corresponds to a threshold at a different value that ranges from 0.01 to 0.5. Recall that our prevalence based algorithm predicts an argument as irrelevant if its prevalence score is over the threshold. This predication is deemed correct if the argument is manually labeled as irrelevant (because it is presentational or content-irrelevant). At threshold 0.1, our approach has a precision of 98% and recall of 94%, respectively, which is a good empirical value we use for our crawl system.

The second experiment in Figure 9b shows the recall at URL level. An argument mistakenly predicted as irrelevant by our algorithm will cause URLs with that argument to be incorrectly deduplicated. In this experiment, in addition to using all arguments manually labeled as relevant in the ground truth, we treat unlabeled arguments with prevalence lower than 0.1 as relevant (which we cannot manually verify however due to the sheer size of such infrequent arguments). We then evaluate the percentage of URLs mistakenly deduplicated (percentage of the loss in content that should have been crawled) due to the mis-prediction. The graph shows that at 0.1 level, only 0.7% of URLs are incorrectly deduplicated.

Finally, Figure 10 compares the reduction ratio of second-level URLs between our approach and the simpler approach of using URL filtering only (which filters out static URLs and URLs without the keyword-query segment, e.g., _nkw= for ebay.com). As can be seen, URL filtering alone accounts for a reduction ratio of 3.6. Our approach of using semantic URL deduplication on top of URL filtering achieves a roughly 10 fold reduction in the number of URLs, which is 2.3 to 3.4 times more reduction than using URL filtering alone, depending on the prevalence threshold. This amounts to significant saving given that the number of second-level URLs extracted from results pages are on the order of billions.

9. CONCLUSION

In this work we develop a prototype system that focuses on crawling entity-oriented deep-web sites. We leverage characteristics of these entity sites, and propose optimized techniques that improve the efficiency and effectiveness of the crawl system.

While these techniques are shown to be useful, our experience points to a few areas that warrant future studies. For example, in the template generation, our parsing approach only handles HTML "GET" forms but not "POST" forms or javascript forms, which reduces site coverage. In query generation, although Freebase-based entity expansion is useful, certain sites with low traffic or diverse traffic do not get matched with Freebase types effectively using query logs alone. Utilizing additional signals (e.g., entities bootstrapped from crawled pages) for entity expansion is an interesting area. Efficiently enumerate entity query for search forms with multiple input fields is another interesting challenge.

Given the ubiquity of entity-oriented deep-web sites and the variety of uses that entity-oriented content can enable, we believe entity-oriented crawl is a useful research effort, and we hope our initial efforts in this area can serve as a springboard for future research.

10. REFERENCES

[1] HTML 4.01 Specification, W3C recommendations. http://www.w3.org/addressing/url/4_uri_recommentations.html.

[2] Z. Bar-yossef, I. Keidar, and U. Schonfeld. Do not crawl in the dust: different urls with similar text. In *Proceedings of WWW*, 2006.

[3] L. Barbosa and J. Freire. Siphoning hidden-web data through keyword-based interfaces. In *Proceedings of SBBD*, 2004.

[4] L. Barbosa and J. Freire. Searching for hidden web databases. In *Proceedings of WebDB*, 2005.

[5] L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *Proceedings of WWW*, 2007.

[6] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of SIGMOD*, 2008.

[7] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *Proceedings of WWW*, 1997.

[8] A. Dasgupta, R. Kumar, and A. Sasturkar. De-duping urls via rewrite rules. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, Proceedings of KDD, 2008.

[9] J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. In *Proceedings of SIGIR*, Proceedings of SIGIR, 2009.

[10] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the deep web. *Commun. ACM*, 50, 2007.

[11] M. A. Hearst. UIs for faceted navigation recent advances and remaining open problems. In *Proceedings of HCIR*, 2008.

[12] A. Jain and M. Pennacchiotti. Open entity extraction from web search query logs. In *Proceedings of ICCL*, 2010.

[13] H. S. Koppula, K. P. Leela, A. Agarwal, K. P. Chitrapura, S. Garg, and A. Sasturkar. Learning url patterns for webpage de-duplication. In *Proceedings of WSDM*, 2010.

[14] J. Madhavan, S. R. Jeffery, S. Cohen, X. luna Dong, D. Ko, C. Yu, and A. Halevy. Web-scale data integration: You can only afford to pay as you go. In *Proceedings of CIDR*, 2007.

[15] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's deep web crawl. In *Proceedings of VLDB*, 2008.

[16] G. S. Manku, A. Jain, and A. D. Sarma. Detecting near-duplicates for web crawling. In *Proceedings of WWW*, 2007.

[17] A. Ntoulas. Downloading textual hidden web content through keyword queries. In *JCDL*, 2005.

[18] M. Paşca. Weakly-supervised discovery of named entities using web search queries. In *Proceedings of CIKM*, 2007.

[19] Y. Qiu and H.-P. Frei. Concept based query expansion. In *Proceedings of SIGIR*, 1993.

[20] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. Technical report, Stanford, 2000.

[21] P.-N. Tan and V. Kumar. *Introduction to Data Mining*.

[22] Y. Wang, J. Lu, and J. Chen. Crawling deep web using a new set covering algorithm. In *Proceedings of ADMA*, 2009.

[23] P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma. Query selection techniques for efficient crawling of structured web sources. In *Proceedings of ICDE*, 2006.