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

Venky 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 search interfaces from some deep-web sites expose textual content (e.g., Wikipedia, PubMed, Twitter, etc), a significant portion of deep-web sites, including almost all shopping sites, curate structured entities as opposed to text documents. Crawling such entity-oriented content can be useful for a variety of purposes. We have built a prototype system that specializes in crawling entity-oriented deep-web sites. Our focus on entities allows several important optimizations that set our system apart from existing work. In this paper we describe important components in our system, each tackling a sub-problem including query generation, empty page filtering and URL deduplication. Our goal of this paper is to share our experiences and findings in building the entity-oriented prototype system.

1. INTRODUCTION

Deep-web crawl refers to the problem of surfacing rich information behind the web search interface of 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 [13, 18]. While crawling the deep-web can be immensely useful for a variety of tasks including web indexing [19] and data integration [18], crawling the deep-web content is known to be hard. The difficulty in surfacing the deep-web has thus inspired a long and fruitful line of research [3, 4, 5, 13, 18, 19, 20, 26, 27].

In this paper we focus on entity-oriented deep-web sites, that curate structured entities and expose them through search interfaces. This is in contrast to document-oriented deep-web sites that mostly maintain unstructured text documents (e.g., Wikipedia, PubMed, Twitter).

Note that entity-oriented deep-web sites are very common and represent a significant portion of the deep-web sites. Examples include, among other things, almost all online shopping sites (e.g., ebay.com, amazon.com, etc), where each entity is typically a product that is associated with rich information like item name, brand name, price, and so forth. Additional examples of entity-oriented deep-web sites include movie sites, job listings, etc. The rich structured content behind such deep-web sites are apparently useful for a variety of purposes beyond web indexing.

While existing crawling frameworks proposed for the general deep-web content are by and large applicable to entity-oriented crawling, the specific context of entity-oriented crawl brings unique opportunities and difficulties. We developed a prototype crawl system that specifically targets entity-oriented deep-web sites, by exploiting the unique features of entity-oriented sites and optimizing our system in several important ways. In this paper, we will focus on describing three important components of our system: query generation, empty page filtering and URL deduplication.

Our first contribution in the paper is a set of query generation algorithms that address the problem of finding appropriate input values for text input fields of the search forms. Our approach leverages two unique data sources that have largely been overlooked in the previous deep-web crawl literature, namely, query logs and knowledge bases like Freebase. We describe in detail how these two data sources can be used to derive queries semantically consistent with each site to retrieve entities (Section 4).

The second contribution of this work is an empty page filtering algorithm that removes crawled pages that contain no entity. We propose an intuitive yet effective approach to detect such empty pages, based on the observation that empty pages from the same site tend to be highly similar with each other (e.g., with the same error message). To begin with we first submit to each target site a small set of intentionally "bad" queries that are certain to retrieve empty pages, thus obtaining a small set of reference empty pages. At crawl time, each newly crawled page is compared with reference empty pages from the same site, and those pages that are highly similar to the reference empty pages can be determined as empty and filtered out from further processing. Our approach is unsupervised and is shown to be robust across different sites on the Web (Section 5).

Our third contribution is an URL deduplication algorithm that eliminates unnecessary URLs from being crawled. While previous work has looked at the problem of URL deduplication from a content similarity perspective after pages are crawled, we propose an approach that deduplicates based on the similarity of the queries used to retrieve entities, which can eliminate syntactically similar pages as well as semantically similar ones that differ only in non-essential ways (e.g., how retrieved entities are rendered and sorted). Specifically, we develop a concept of *prevalence* for URL

*Work done while author at Google

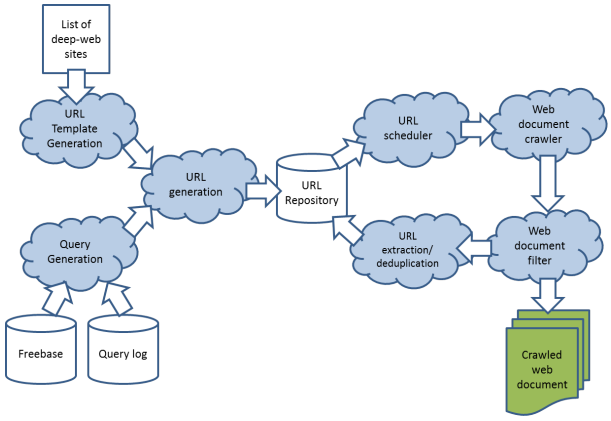


Figure 1: Overview of the entity-oriented crawl system

argument that can predict the relevance of URL arguments, and propose a deduplication algorithm based on prevalence. This approach is shown to be effective in preserving distinct content while greatly reducing the consumption of crawl bandwidth and system complexity (Section 6).

2. SYSTEM OVERVIEW

Deep-web site	URL template
ebay.com	*www.ebay.com/sch/i.html?_**nkw={query}**&_sacat=See-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

At a high level, the architecture of our system is illustrated in Figure 1. At the top left corner the system takes domain names of deep-web sites as input, as illustrated in the first column of Table 1. The URL template generation component then crawls the homepages of these websites, extracts and parses the web forms found on the homepage, and produces URL templates that correspond to the URLs if web forms are submitted, as illustrated in the second column of Table 1. Here the highlighted "{query}" represents a wildcard that can be substituted by any keyword query (e.g., "ipad+2") to retrieve relevant content from the site's backend database.

The query generation component at the lower left corner takes the Freebase and query log as input, outputs relevant queries matching the semantics of each deep-web site (for example, query "ipad 2" for ebay.com). The URL generation component can then plug the queries into the URL template to produce final URLs in a URL repository.

URLs can then be retrieved from the URL repository and scheduled for crawl. It is inevitable that some URLs correspond to empty pages (i.e., contain no entity). So after the pages are crawled, we move to the next stage, where the crawled pages are inspected and empty pages filtered. Remaining pages are the final output of the crawl system that can be used for a variety of entity-oriented processing.

We observe that a small fraction of URLs on the returned pages (henceforth referred to as "second-level URLs") typically link to additional deep-web content. However, crawling all second-level URLs indiscriminately is both wasteful and practically impossible given the large number of such URLs. In the next step, we filter out second-level URLs that are less likely to lead to deep-web

Figure 2: A typical search interface

content, and dynamically deduplicate remaining URLs to obtain a much smaller set of "representative" URLs that can be crawled efficiently. These URLs are then iterated through the same loop to obtain additional deep-web content. In practice a number of iterations are needed, as the crawler may fail to obtain the content due to server-side host-load restrictions and the URL scheduler can schedule any failed URLs to be re-crawled.

In the following, we will describe four important components in the system in turn, namely URL template generation, query generation, empty page filtering and URL deduplication.

3. URL TEMPLATE GENERATION

As input to our system, we are given a list of deep-web sites that are entity-oriented. The first problem in URL template generation is to locate the search form in each site. The form is then parsed to produce the URL template that is equivalent to a form submission when values are filled. As an example, observe that the search form from ebay.com in Figure 2 corresponds to a typical search interface. Searching this form using query q without changing the default value "All Category" of the drop-down box is equivalent to using the URL template for ebay.com in Table 1, with wildcard {query} replaced by q.

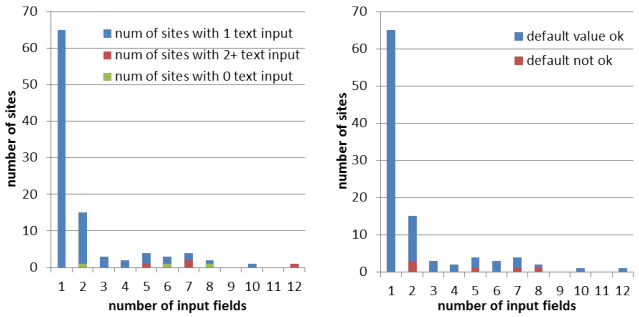
The general problem of generating URL templates has been studied in different context in the literature. For example, the authors in [4, 5] looked at the problem of identifying searchable forms that are deep-web entry points. In [19, 23], the problem of assigning appropriate values to the combination of input fields has been explored.

In principle, variants of these sophisticated techniques can be applied. Based on our observations of entity-oriented deep-web sites, however, we contend that in the context of these entity-oriented sites, templates can be generated in a much simpler manner.

Our first observation is that the search forms are almost always on the home page instead of somewhere deep in the site. We manually survey 100 sites sampled from the input sites used by our system,[1] only 1 of which (arke.nl) has the search form on a page one click away from the home page. This is not surprising — the search form is such an effective information retrieval paradigm that websites are only too eager to expose the search interface at prominent positions on their home pages. While traversing deep into each site may help to discover additional deep-web entry points, we find it sufficient to only extract the search forms on the homepage.

The second observation is that the search interfaces exposed by the entity-oriented sites are relatively simple. Overall, the use of text input fields (to accept keyword queries) is ubiquitous — filling appropriate queries for these text fields turns out to be challenging (to be discussed in Section 4). However, for all other input fields like drop-down boxes or radio buttons, the seemingly simplistic approach of using default values is just sufficient. As an intuitive example, the search interface of ebay in Figure 2 is fairly typical. It has one text input field to accept keyword queries, and an additional drop-down box to specify subcategories. When the selection defaults to "all category," all entities matching the keyword query

[1]We sample at the "organization" level, by treating all sites with the same name but different country suffix codes as one organization. For example, ebay.com and dozens of its subsidiaries operating in different countries (ebay.co.uk, ebay.ca, etc) are highly similar. They are treated as one organization, ebay, in our survey to avoid over-representation.



(a) num. of input text field (b) default value analysis

Figure 3: An analysis of search forms sampled from deep-web sites

will be retrieved.

To illustrate our point, Figure 3a plot the distribution of forms by their number of input fields, using forms extracted from 100 sites sampled from our input. The majority of the search forms are really simple — a full 80% of forms have only 1 or 2 input fields. Furthermore, almost all sites have exactly one text input field to accept keyword queries. For example, for all 1-input-forms (leftmost column), the only input field is the text input field. Overall, only 3% of sites have no text input (all of which are search interfaces for automobiles that only allow selecting model/year etc from a fixed set of values from drop-down boxes). An additional 4% has 2 text input fields (all in the airfare search vertical where an origin and a destination have to be specified).

For all other input fields like drop-down boxes or radio buttons, there typically exists a default value as in the example in Figure 2[1]. We then analyze whether using the default value in combination with appropriate queries can retrieve all possible entities. As shown in Figure 3b, using default values fails this task in only 6 out of the 100 sites surveyed, most of which again in the car/airfare search verticals. Although it can be argued that enumerating possible value combination from the drop-down boxes provide a specific subset of entities not possible to obtain if default values are used, as will be clear in Section 6, due to the prevalence of the use of *faceted search* paradigm in entity-oriented deep-web sites, crawling *second-level URLs* actually provides a more tractable way to retrieve a similar subset of entities than enumerating the space of all input value combinations in the search form.

To sum up, we observe that most search forms have at least 1 text input field. Filling its value from the virtually infinite space is an important challenge that we defer to the 'next section. At this template generation stage we only use the placeholder "{query}". For all other non-text input fields, the simple approach of using default values is sufficient. In principle, the few sites from niche verticals where the proposed template generation fails can still be handled using multi-input enumeration [19, 23] or data integration techniques [18].

Our implementation of URL template generation is built upon the parsing techniques developed in the pioneering work [19]. Since generating URL templates is not the focus of this work, we will skip the detail in the interest of space, and refer readers to [19] for details. We note that this approach comes with the standard caveat, that is it can handle HTML "GET" forms, but not the majority of HTML "POST" forms or javascript forms. It is our experience that overall, URL templates can be generated correctly for around 50% of the sites.

[1]Even when the default values are not displayed in browser, as typically is the case for date of departure/arrival fields in hotel booking vertical, default values can still be obtained when the HTML form source code are parsed.

4. QUERY GENERATION

After obtaining URL templates for each site, the next step is to fill relevant keyword query into the "{query}" wild-card in URL templates to produce final URLs. The challenge here is to come up with queries that match the semantics of the sites – crawling queries like "ipad 2" on tripadvisor.com does not make sense, and will most likely result in an empty/error page. The naive brute force approach of sending every known entity from some dictionary to every site is clearly inefficient and wasteful of the crawl bandwidth.

Prior art in query generation for deep web crawl mostly focus on bootstrapping using text extracted from the retrieve pages [19, 20, 26, 27]. That is, a set of seed queries are used first to crawl, the retrieved pages are analyzed for promising keywords, which are then used iteratively as queries to crawl more pages.

There are several key differences that set our approach apart from existing work. First of all, most of previous work [20, 26, 27] aims to optimize coverage of *individual* site, 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 more in line with [19] in aiming to improve content coverage of general sites on the Web. Because of the sheer number of deep-web sites on the Web we have to trade off complete coverage of individual site for incomplete but "representative" coverage of many more sites. Also observe that an implicit assumption used in previous work is that the "next page" link can always be reliably identified so that content that not shown on the first page can be crawled. While this is possible for a small number of sites, it is unrealistic in general for sites at large on the Web. In addition, we argue that crawling all possible "next pages" may not be necessary to start with, for the first page returned should already be representative for the query searched, exhaustively crawling all entities matching the same query may only bring marginal benefit. In our system we focus on producing a more diverse set of queries to crawl and contend that this is more beneficial to improving coverage than crawling all "next" links.

The second important difference is that the techniques and data sources we use are very different. Instead of using text extracted from the crawled pages, we leverage two important data sources, namely (1) query log, filtered and cleaned using entity-oriented approaches, and (2) manually curated knowledge base like Freebase [7]. To our knowledge neither of these two has been studied in the deep-web crawl literature for query generation purposes. We will discuss each approach in turn in the following sections.

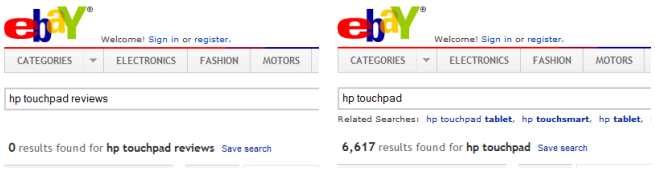
4.1 Query logs based query generation

Query log refers to the information of keyword queries searched on search engines (e.g., Google), and URLs that users finally clicked on among all links that are displayed. Conceptually query log makes a good candidate for query generation in deep web crawls — queries with high number of clicks to a certain site is a clear indication of the relevance between the query and the site, thus submitting the query through the site's search interface for deep-web crawl makes intuitive sense.

For our query expansion purposes, we used about 6 months' worth of query log from Google. We normalize information in the query log to the following standard form $< keyword\_query,$

Deep-web sites	sample queries from query log
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 log



(a) search with "hp touchpad reviews"

(b) search with "hp touchpad"

Figure 4: An example of Keyword-And based search interface

$url\_clicked, num\_times\_clicked >$.

4.1.1 Query Log Filtering

While query log in general constitutes a good source of information for query generation, not all queries searched on search engines (henceforth referred to as "search engine queries") make a good candidate for entity-oriented deep-web crawls (referred to as "site search queries").

The first observation is that there are a certain percentage of "navigational queries", where users have a clear destination in mind, but rather than manually typing in the URLs, they use search engines to get redirected to the site of interest. Examples include "ebay uk", "barnesandnoble locations", etc. Such navigational queries do not correspond to deep-web entities, crawling using such queries in URL templates are clearly not ideal.

As a heuristic to exclude navigational queries, we only consider queries that are clicked for at least 2 pages in the same site, each for at least 3 times. This leaves us with a total of about 19M unique query/site pairs. Table 2 illustrates example site/query pairs.

4.1.2 Query Log Cleaning

Query cleaning motivation. After filtering out navigational queries, we observe that the remaining queries are rich in nature, but too noisy to be used directly to crawl deep-web sites. Specifically, queries in the query log tend to contain extraneous tokens in addition to the central entity of interest, while it is not uncommon for the search interface on deep-web sites to expect only entity names as queries. Figure 4 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 4a), while searching using only the entity name "HP touchpad" retrieves 6617 such products (Figure 4b).

The problem illustrated in Figure 4 is not isolated when search engine queries are used for entity-oriented crawls. There are two sides of the problem, on the one hand, a significant portion of search engine queries contain extraneous tokens in addition to entity mentions. On the other hand, many search interfaces on deep-web sites only expect clean entity queries.

On the query side, it is actually common for search engine queries to include extraneous information. We observe at least three categories of such queries. First, many search engine queries aim to retrieve information about certain aspects of the entity of interest. For example, "HP touchpad review", "price of chrome book spec" where "review" and "price of" specify aspects of the entity of interest. Second, some tokens in the search engine queries can be navigational rather than informational. Examples include "touchpad ebay" or "wii bestbuy", where ebay and bestbuy only specify sites to which users want to be directed and have nothing to do with the entity. Lastly, it is common for users to frame queries using natural language questions, e.g., "where to buy iPad 2", "where to stay in new york".

On the other hand, many deep-web sites only expect clean entity queries. Conceptually, the entity oriented search interface on deep-web sites fits into the "keyword search over structured database" paradigm, e.g., [1, 6, 10, 15]. However, in practice this tends to be implemented relatively simply. For example, it is our observation that variants of the simple Keyword-And mechanism are commonly used across different sites (e.g., ebay.com, overstock.com, nordstrom.com, etc). In Keyword-And based search, *all* tokens in the query have to be matched in a tuple before the tuple can be returned. As a result, when search engine queries that contain extraneous tokens are used no matches may be retrieved (Figure 4b). Even if the other conceptual alternative, Keyword-Or is used, the presence of extraneous tokens can still promote spurious matches that are less desirable.

In contrast, the modern search engines are much more specialized in answering keyword queries, by leveraging sophisticated ranking functions e.g., [8, 24]. Accordingly, search engines are much better in answering noisy keyword queries than a typical deep-web site (hardly a surprise comparing the amount of efforts put into the few big search engines, and the individual efforts of maintaining a deep-web site).

This mismatch in how keyword queries are handled in search engine and deep-web sites underlines the fact that search engine queries are ill-suited for deep-web crawls directly.

Query pattern aggregation. In order to use search engine queries for deep-web crawl purposes, we propose to 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 the name of the (ever-growing) entities on the Web, and their possible name variations, abbreviations or even typos commonly seen in query log. On the other hand, the diverse nature of the query log only makes the it more valuable, for it captures a wide variety of entities along with their name variations that closely matches the content that can be crawled from the Web.

Instead of identifying entity mentions from queries directly, we propose to first find common patterns in the query that are clearly not entity related. We observe that people tend to frame queries in certain fixed ways (e.g., "where to stay in", "reviews", etc). Inspired by an earlier work on entity extraction [21] we propose to detect such patterns by aggregating pattern occurrences in the query log.

Stated more formally, given a keyword query $q = (t_1, t_2, ..., t_n)$ that consists of n tokens, we want to segment it into three subsequences, a (possibly empty) prefix p, an entity mention $e = (t_i, t_{i+1}, ...t_j)$, where $1 \leq i \leq j \leq n$, and a (possibly empty) suffix s, such that p and s are not relevant to the central entity e.

To identify entity mentions and segment queries, we obtained a dump of the Freebase data [7] — a manually curated repository that contains about 22M entities. We then find the maximum length subsequence in each search engine query that matches Freebase entities as an entity mention. If there are more than one match with

the same length, all such matches are preserved as candidates.

After entity mentions are extracted from the query, the remainders are treated as query prefix/suffix. We aggregate distinct prefix/suffix across the query log to obtain frequent patterns. The most frequent patterns are likely to be irrelevant to specific entities because it is unlikely for so many queries to search for the same entities.

EXAMPLE 1. *Table 2 illustrate the sample queries with mentions of Freebase entity names underlined. Observe that this is a rather rough entity recognition. First false matches can occur. For example, the query "where to stay in new york" for booking.com has two matches with Freebase entities, the less expected match of "where to", which according to Freebase is the name of a musical release, and the match of "new york" as city name. Since both matches are of length two, both are preserved, resulting the false suffix "stay in new york" and the correct prefix "where to stay in", respectively. However, when all the prefix/suffix in the query log are aggregated, "where to stay in" clearly stands out as it is much more frequent.*

In addition, Freebase may not contain all possible entities. For example in the query "hyatt seattle review" for booking.com, the first two tokens refer to the Hyatt hotel in Seattle, which however cannot be found in Freebase. Instead it will produce three matches for each token, "hyatt" the hotel company, "seattle" the location, and "review" an unexpected match to a musical album. Accordingly the generated prefixes/suffixes include "seattle review", "hyatt", "review", and "hyatt seattle". This nevertheless works fine as after aggregation only the suffix "review" is frequent enough to be used to clean the query into "hyatt seattle".

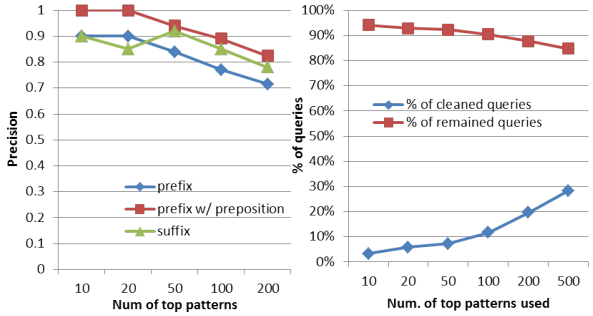
Top prefix	Top prefix with preposition	Top suffix
how	lyrics to	lyrics
watch	pictures of	download
samsung	list of	wiki
download	map of	torrent
is	history of	online
which	lyrics for	video
free	pics of	review
the	lyrics of	mediafire
best	facts about	pictures

Table 3: Top 10 common patterns

We list the top 10 most frequent prefix and suffix patterns in Table 3. We further observe that the presence of preposition in the prefix is a good indication that the prefix is not relevant to any entity. Patterns so produced are listed in the second column. The same trick however does not apply straightforwardly to suffix, for suffix with preposition are mostly referring to locations like "in us", "in uk", etc, that are useful in retrieving exact matches and are treated as an integral part of entity mentions that should not be dropped.

In Table 3 patterns that are relevant to the entity (and are thus mislabeled) are underlined. It is clear from the table that most patterns found this way are indeed not related to specific entities. Removing such patterns allows us to obtain clean and diverse entity names ranging from song/album names ("lyrics", "lyrics to", etc), location/attraction names ("pictures of", "map of", "where to stay in" etc), to numerous variety of product names ("review", "price of", etc).

In Figure 5a we summarize the precision of the patterns so produced. We manually label the top patterns as correct or incorrect, depending on whether the patterns are related to entities or not, and evaluate the precision for top 10, 20, 50, 100 and 200 patterns. Not surprisingly, the precision decreases as more number of patterns are included.



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

Figure 5: Removing top patterns

Figure 5b shows the impact of removing top patterns, defined as the percentage of queries in the query log that contain top patterns. At top 200 about 19% of the queries will be cleaned using our approach. In addition the total number of distinct queries reduces by 12%, because after cleaning some queries become duplicate with existing queries. In general this query size reduction is positive, as it reduces number of unnecessary crawls that may arise if patterns (e.g., "review", "price of") are not cleaned.

4.2 Freebase based query expansion

While the query log provides a diverse set of seed entities, its coverage for each site can be dependent on the site's popularity as well as the item's popularity (recall that number of clicks is used to to predict the relevance between the query and the site, which is affected by both the site popularity and item popularity). Even for really popular sites the coverage of the queries generated from query log are typically not exhaustive (e.g., all possible city names for a travel site, for example).

While the coverage provided by the query log is still limited, we observe that there exists manually curated entity repository, e.g., Freebase, that maintains entities in certain domains with very high coverage (comprehensive lists of known cities, books, car models, movies, etc). Certain categories of entities, 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 on travel sites (e.g., tripadvisor.com, booking.com), housing sites 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. We explore the problem of matching deep-web sites with Freebase entities in this section.

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 4: Freebase domains used for query expansion

Deep-web sites	sample queries from query log
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 5: Example entities extracted for each deep-web site

From a top-down perspective, Freebase data is organized as follows. On the highest level Freebase data are grouped into the so-called "domains", or categories of relevant topics, like automotive, book, computers, etc, in the first column in Table 4. Under each domain, there is a list of relevant "types", each of which consists of manually curated data instances that can be thought of as a relational table. For example, the domain film contains top types including film (list of film names), actor (list of actor names) and performance (which actor performed in which film relation).

Although Freebase data are in general of high quality, some domains in Freebase (e.g., chemistry ontology, or wikipedia articles) are not as widely applicable for deep-web crawl purposes. In our experiments in this section we will focus on 10 domains that are of wide interests, as listed in Table 4.

Recall that we can already extract Freebase entities from the query log. Table 5, for example, contains lists of entities extracted from the the sample queries in Table 2. Thus, for each site, we can effectively obtain a list of relevant Freebase entities as seeds. Using these seed entities that are indicative of site semantics, we measure relevance between Freebase "types" and sites.

While there exist multiple ways to model the relevance ranking problem, we view this as an information retrieval problem. We treat the multi-set of Freebase entity mentions for each site (each row in Table 5) as a document, and the list of entities in each Freebase type as a query. Both of the document and the query can be represented using a feature vector model, and the classical *term-frequency, inverse document frequency* (TF-IDF) ranking in information retrieval can then be applied straightforwardly.

DEFINITION 1. *[24] Let $D = \{D_i\}$ be the set of documents, and Q be the query. In the **vector space model**, the query Q is represented as a weight vector q*

$$q = (w_{1,q}, w_{2,q}, ...w_{t,q}),$$

and each document is also represented as a weight vector d_i

$$d_i = (w_{1,i}, w_{2,i}, ...w_{t,i}),$$

*where each dimension in the vector represents a unique entity from a Freebase type. The **relevance score** between query Q and document D_i can be modeled as the cosine similarity of the two vectors q and d_i.*

$$sim(q, d_i) = cos\theta = \frac{q \cdot d_i}{||q||\ ||d_i||}$$

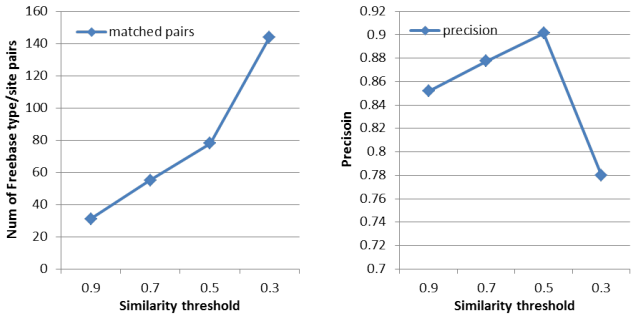
DEFINITION 2. *[24] In **term-frequency, inverse-document-frequency** (TF-IDF) weight scheme, each token weight $w_{t,d}$ in document/query is weighted as a product of term-frequency, $tf(t,d)$, and inverse-document-frequency, $idf(t)$,*

$$w_{t,d} = tf(t,d) * idf(t),$$

where $tf(t,d)$ is the number of occurrences of t in document d, and $idf(t)$ is the inverse of document frequency, $idf(t) = log\frac{|D|}{|\{d:t\in D\}|}$

Domain:type name	matched deep-web sites
Automotive:model_year	stratmosphere.com, ebay.com
Book:book_edition	christianbook.com, netflix.com, barnesandnoble.com, scholastic.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, agoda.com, apartmenthomeliving.com, ...
Music:track	netflix.com, play.com, musicload.de
TV:tv_series_episode	netflix.com, cafepress.com
Wine:wine	wineenthusiast.com

Table 6: Matched domains for top Freebase types



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

Figure 6: Effects of different score threshold

We observe that TF-IDF based relevance ranking is more effective than other similarity models like simple Cosine similarity or Jaccard Similarity [25], due to the existence of false matches of Freebase entities that have very common names. For example, when Cosine similarity is used (without tf-idf) there is a very high similarity between the deep-web site "1800flowers.com" and the Freebase type "citytown" (names of cities). The reason, as it turns out, is because a high number of queries associated with "1800flowers.com" contain words like "flowers", "love", "baskets" etc. These very common words surprisingly coincide with cities named "flowers", "love", "baskets" that are treated as matches. Without using the IDF to penalize such common terms and promote infrequent terms like "Zaneville" (which is a much more indicative location name), precision of matches produced using simple cosine similarity is low.

For each Freebase type as input, we can use TF-IDF to produce a ranked list of deep-web sites by their similarity scores. In this list of sites sorted by similarity, we need to "threshold" the list, and use all sites with scores higher than the threshold as matches. Since the similarity scores for different Freebase types are not directly comparable, setting a constant threshold score across multiple Freebases types are not possible. We use a "relative thresholds" by threshold at a fixed percentage of the highest similarity score for each Freebase type (for example, if the highest score of all sites for Freebase type model_year is 0.1, a relative threshold of 0.5 means that any site with score higher than 0.05 will be picked as matches).

In order to explores the effects of using different thresholds, we manually evaluate the 5 largest Freebase types (by total number of entities) in all 10 experimented domains. Figure 6a shows the total number of matched Freebase-type / site pairs and Figure 6b illustrates the matching precision. In order not to overstate the precision of the matching algorithm, we ignore matches for sites that span multiple product categories (ebay.com, nextag.com, etc), by treating such matches as neither correct nor incorrect. As we can see, while the number of matched pairs increases as threshold

decreases, there is a significant drop in matching precision when threshold decreases from 0.5 to 0.3. Empirically a threshold of 0.5 is used in our system.

Table 6 illustrates example matches between Freebase types and deep-web sites. Incorrect matches are underlined, and only matches for the largest Freebase type in each experimented domain are listed in the interest of space. Sites are sorted by their similarity score and a threshold of 0.5 is used.

5. EMPTY PAGE FILTERING

Once the final URLs are generated and pages crawled, we need to filter empty pages with no entity in them. Deep-web sites typically display error messages like "sorry, no items matching your criteria is found", or "0 item matches your search" when empty page are returned. While such error messages are easy for human to identify, it can be challenging for a program to automatically detect all variants of such messages across different sites.

Authors in [19] developed a novel 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 since it works at the granularity of search form / URL template, it may still miss empty pages crawled using an informative URL template.

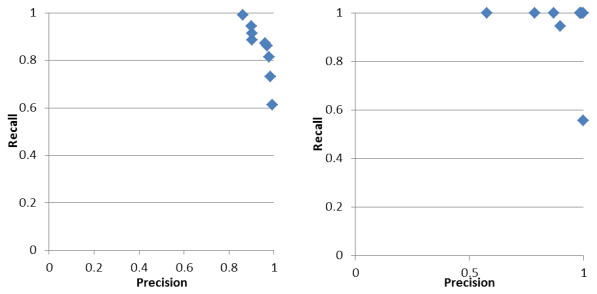
Since the search forms for entity-oriented sites are observed to be predominately simple (Section 3), we typically have only one template for each site. Filtering at the granularity of URL templates are thus ill-suited. On the other hand, it is inevitable that some queries in the diverse set of generated queries will fail to retrieve any entities. Filtering at the granularity of page is thus desirable.

Our main observation for page-level filtering is that empty pages in the same site are extremely similar to each other, while empty pages from different sites are disparate. 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 randomly permuted characters that lack any semantic meanings (e.g., "zzzzzzzzzzzzz", or "xyzxyzxyzxyz"). Such queries, when searched on deep-web sites, will almost certainly generate empty pages. In practice, we generate N (typically 10) such background queries to be robust against the rare case where a bad query can accidentally retrieve some results. We then crawl and store the corresponding "background pages". At crawl time, each newly crawled page is compared with the "background pages" to determine if the new page is actually empty.

Our content comparison mechanism is based on the effective page summarization called signature developed in [19]. The signature is essentially a set of tokens that are descriptive of the page content, and robust against minor differences in the page (e.g., dynamic advertising content). We then calculate the Jaccard Similarity using the signature of the newly crawled page and the "background pages", as defined below.

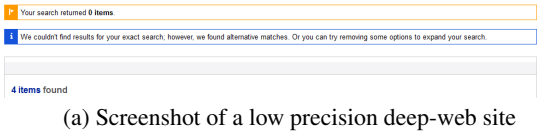
DEFINITION 3. *[25] Let S_{p_1} and S_{p_2} be the sets of tokens representing the signature of the crawled page p_1 and p_2. The **Jaccard Similarity** between S_{p_1} and S_{p_2}, denoted $Sim_{Jac}(S_{p_1}, S_{p_2})$, is defined as* $Sim_{Jac}(S_{p_1}, S_{p_2}) = \frac{S_{p_1} \cap S_{p_2}}{S_{p_1} \cup S_{p_2}}$

The similarity scores are then averaged over the set of "background pages", and if the average score is above certain threshold θ, we can label the crawled page as empty.

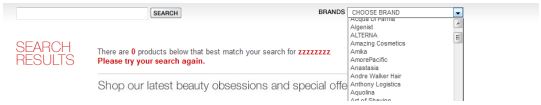


(a) Vary score threshold (b) Distribution of individual site

Figure 7: Precision/recall of empty page filtering



(a) Screenshot of a low precision deep-web site



(b) Screenshot of a low recall deep-web site

Figure 8: Empty page filtering analysis

To evaluate the effectiveness of the empty page filtering approach, we randomly selected 10 deep-web sites and manually identified their respective error messages (e.g., "Your search returned 0 items" is the message used on ebay.com). This enables us to build the ground truth — any page crawled from the site with that particular message are regarded as empty pages (negative instances), and pages without such message are treated as non-empty pages (positive instances). We can then evaluate using precision and recall, where precision is defined as

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

and recall is defined as

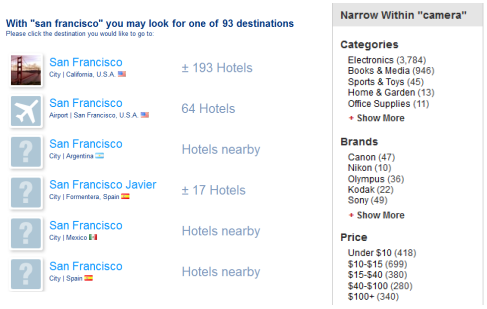
$$recall = \frac{|\{\text{pages predicted as non-empty}\} \cap \{\text{pages that are non-empty}\}|}{\{|\text{pages that are non-empty}|\}}$$

Figure 7a 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 use in our system.

Figure 7b 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 one site with low precision (around 0.55) and another with low recall (around 0.58), respectively.

The cause of the anomalous performance of the two sites are actually quite interesting. For the low precision site, for which our algorithm mistakenly labeled empty pages as non-empty, when a query matches no item in the back-end database, the original query is automatically reformulated into a related query so that some alternative matches can be produced. This is illustrated in Fig-

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



(b) Disambiguation on first-level pages

(c) Faceted search on first-level pages

Figure 9: Motivation for second-level crawl

ure 8a. Because the error message "your search returned 0 items" is present, the page is marked as empty by the ground-truth. However, because the page still returns some meaningful content (the alternative matches), the signature produced for the page is significantly different from the signature of background empty pages crawled using bad queries. As a result, many of such pages are labeled as "non-empty" by our algorithm although the ground-truth claims such pages to be empty.

The low recall problem observed on the other site, however, arises due to a completely different reason. The search result page of this particular site, as shown in Figure 8b, is very unique in that it contains a drop-down box that lists all available product brands to facilitate user browsing, regardless of whether the returned page is empty or not. These information are very specific and unique such that they dominates the signature produced for the page, even when a few items are actually retrieved. As a result, the signatures of many pages that only have a few items are similar to that of background pages, thus resulting in a low recall.

6. SECOND-LEVEL CRAWL AND URL DEDUPLICATION

6.1 The motivation for second level crawl

We analyze the first set of pages obtained using URL templates, for which we call the "first-level pages" because they are directly obtained through the search interface (one-level beneath the homepage). we observe that there are many additional URLs on the first-level pages that link to other desirable deep-web content, which are just 1-click away. Crawling these additional deep-web URLs exist on the first-level pages, henceforth referred to as "second-level URLs/pages", is highly desirable. First, we categorize three common cases where crawling second-level pages can be useful.

In the first category, when a keyword query is searched, a list of other frequently searched queries relevant to the original query are displayed. This is known as query expansion [22] in the literature that aims to help users to reformulate queries more easily. Figure 9a is a screenshot of such an example site. When the original query "iphone 4" is searched, the returned page displays queries related to the original query, like "iphone 4 unlocked", "iphone 3gs", "iphone 4 case", etc. Since these query suggestions are maintained and provided by the site, they provide a reliable way to discover other relevant deep-web pages and to improve content coverage.

Figure 9b shows the second type of sites for which second-level crawl can be useful. In this type of sites, a disambiguation page is often returned first when a query is searched. Only after following appropriate URLs on the disambiguation page are rich deep-web content revealed. In the example, when "san francisco" is searched, all cities in the world with that name appear, and URL of each city can lead to the real content, which in this case a list of hotels.

Second-level crawls are also desirable for the third type of sites, as illustrated in Figure 9c. These sites employ a very common search paradigm called "faceted search/browsing" [14], in which returned entities are presented in a multi-dimensional, faceted manner. Multiple classification criteria are displayed, oftentimes on the right-hand side on the result page, to allow users to drill-down using different criteria. In this example when "camera" is searched, in addition to returning a (large) set of entities, a "multi-faceted" entity classification is also presented as in Figure 9c. URLs exposed by the faceted search interface allow users to narrow down by category, brand, price etc — conceptually equivalent to placing an additional predicate on entity retrieval query to produce a subset of entities. These URLs are desirable targets for further crawl, as they bring representative coverage for a potentially large set of returned results.

In addition, we observe that some of drill-down in second-level URLs exposed by the multi-faceted search are actually equivalent to submitting search forms with appropriate values of drop-down boxes filled in. Recall that in URL template generation in Section 3, we take a simplifying approach of using default values (e.g., "All-categories") from drop-down boxes instead of enumerating all possible values (subcategories "Electronics", "Furniture", ...). In this example of query "camera", the URL for category "Electronics" in the faceted search interface is equivalent to searching "camera" using the search form, with sub-category "Electronics" selected in drop-down box. From that perspective, crawling second-level URLs can be equivalent to enumerating all possible values from drop-down boxes.

What makes crawling second-level URLs more attractive than the alternative of enumerating all possible value combination in the search form is the potential savings in number of crawl attempts. Observe that the second-level URLs are typically produced according to the queries searched, that is, URLs for mismatched value combinations that retrieve no entity (for example query "camera" under the category of "Furniture") will not be generated and need not be crawled. In comparison, if all possible values in the search form are to be exhaustively enumerated there is no way to know beforehand if certain value combination retrieves no results, thus a large number of crawl attempts may be wasted (for example it is shown in [18] that exhaustive enumeration yields 32 million form submissions for a car search website, which is a number greater than the total number of cars for sale in US).

6.2 URL extraction and filtering

Even though *some* second-level URLs on the first-level pages are desirable, not all second-level URLs should be crawled, due to efficiency as well as quality concerns.

First of all, for each first-level page, the number of second-level URLs that can be extracted ranges from dozens to a few hundreds. For example, in our experiment the number of second-level URLs extracted from a batch of 35M first-level pages are over 1.7 billion, which is clearly too many to be crawled efficiently. Scaling-out using clusters of machines does not help, as the bottleneck lies in the site-specific host-load restriction, which limits the number of crawls permitted per second without overloading the server.

More importantly, not all second-level URLs are equally desir-

able. For example, there typically exists a URL for each entity returned on the result page that links to a page with detailed description of of the item. Such detailed item pages are less desirable from a cost/benefit perspective. Their information already exist on the first-level pages; furthermore, each such crawl only obtain one entity instead of a list of entities. There also exist many second-level URLs entirely irrelevant to deep-web entities, for example, catalog browsing URLs of the site, member login URLs, etc. None of these URLs are good candidates for second-level crawls.

In view of this, we filter URLs by only considering URLs that contain the argument for "{query}" wild-card in URL templates. In Table 1 for example, the arguments are "_nkw=" for ebay.com, "search_by=" for chegg.com, "keyword=" for beso.com, etc. This filtering stems from the observation that for all three categories of desirable second-level URLs discussed above, the content of the second-level pages are still generated using keyword queries — either with a new keyword search relevant to the original query (category 1), or with the same keyword search but some additional filtering predicates (category 2 and 3). Filtering URLs by query argument turns out to be effective that can significantly reduce the number of URLs while still preserving desirable second-level URLs. The reduction ratio is typically between 3 to 5, — for example, the 1.7 Billion second-level URLs extracted from the experimental batch of 35M first-level pages are reduced down to around 500 million.

6.3 URL deduplication

6.3.1 Deduplication objective

Ideally, the filtered set of second-level URLs can still be further reduced to best utilize crawl bandwidth. In this section we propose deduplicate URL to achieve further reduction.

Traditionally two URLs are considered duplicates if the content of the pages are the same or highly similar [2, 11, 17]. We call these approaches **content-based URL deduplication**. Our proposed definition of duplicates captures content similarity as well as semantic similarity of the entity retrieving queries.

Specifically, recall that the mechanism of dynamically generating deep-web content corresponds to an entity selection query. Take the URL from buy.com in the first row of Table 7 as an example. The part of string after "?" is called *query string*. Each component deliminated by "&" is a *query segment* that consists of a pair of CGI *argument* and *value*. Each query segment typically corresponds to a predicate. For example, the query segment "qu=gsp" requires the entity to contain keyword "gps". "Sort=4" specifies that the list of entities should be sorted by price from low to high; "from=7" is for internal use so that the site can track which URL was clicked to lead to this page; "mfgid=-652" is a predicate that selects only manufacturer Garmin; and finally "page=1" retrieves the first page of entities that match the criteria. If this query string is to be written in SQL, it would look like the query below

*Select * From db*
Where description like '%gps%', manufacturer = 'Garmin'
Order by price desc
Limit 20;

While the exact representation and the internal encoding of the query string varies wildly from site to site, the concept of query string generally holds across different sites.

Our definition of URL duplicates is simply such that if URLs correspond to selection queries with the same set of selection predicates, i.e., entities returned are the same, then irrespective of how the items are sorted or what portion of matched entities are presented, these URLs are considered to be duplicates to each other.

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

Table 7: Duplicate cluster of second-level URLs

We refer to query segments that have no effect on the page content as *content irrelevant segments* (e.g., the tracking parameter "from=?"), and segments that only affect how the retrieved set of entities are presented as *presentation segments* (e.g., the sorting criteria "sort=?"). An alternative way to state the our definition then is that all URLs that differ only in content irrelevant segment or presentation segment are duplicates.

EXAMPLE 2. *As an example, URLs in the same group in Table 7 are duplicates URLs by our definition. The first group of URLs, for example, all corresponds to the same selection query discussed above. They only differ in content irrelevant segments, like tracking parameters ("from="), or presentation segments like sorting criteria ("sort="), and page number ("page="). These URLs are considered as duplicates and we only need to crawl one URL from each group as a result.*

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

The decision of disregarding "content irrelevant segments" are straightforward. The rationale behind treating *presentation segments* as irrelevant goes back to our overarching goal of obtaining "representative coverage" for each site. Because again our objective is not to obtain complete coverage of individual site. Instead, we aim at representative coverage — crawling one page for each new conceptual selection query is sufficient. Exhaustively crawling all pages for the same selection query that differ only in how items are presented only provides marginal benefits. URLs that differ in presentation segments are treated as "semantic duplicates" as a result. Accordingly, our goal of deduplication is **semantic-based URL deduplication** — it subsumes the traditional content-baesd URL deduplication by capturing content similarity as well as semantic similarity.

6.3.2 Related work

The problem of URL deduplication has received considerable attention in the context of web crawling [2, 11, 17]. This line of work propose to first analyze the content sketch [9] to group highly similar page into duplicate clusters. URLs in the same duplicate cluster are then processed using data mining techniques to learn various URL transformation rules (e.g., `cnn.com/mony/whatever` is equivalent to `money.cnn.com/whatever`, or `domain.com/story?id=nu` is equivalent to `domain.com/story_num`).

Given our stronger definition of URL duplicates, deduplication using page content analysis clearly won't work. Specifically, two queries with the same selection predicates but different presentation criteria can lead to very different page content. For example, if there are a large number of matched items, using different sorting criteria, or different number of items per page, etc can generate a totally different page. In addition, these techniques are post-crawl deduplication, whereas our proposed technique works without crawling the actual page content.

Authors in [19] 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, works at the granularity of search forms and cannot be used to deduplicate URLs directly.

6.3.3 *Pre-crawl URL deduplication*

Our deduplication scheme is not based on any content analysis. As a matter of fact, URLs are deduplicated even before pages are crawled, a significant departure from existing post-crawl approaches. Our approach is based on two key observations. First, search result pages dynamically generated from the same deep-web site are *homonogenous*. That is, the structure, layout and content of result pages from the same site share much similarity. It thus makes sense to use all URLs from the same page as a unit of analysis, in addition to analyzing each URL individually.

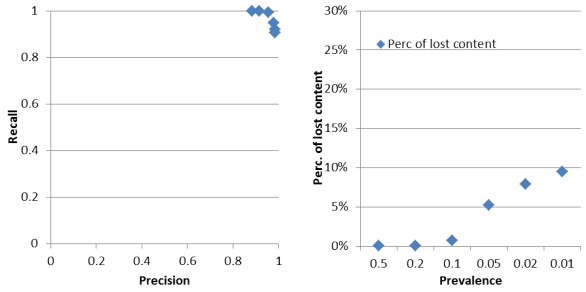
Secondly, given the homogeneity of the search pages from the same deep-web site, we observe that if the same URL query segment (i.e., a pair of argument/value, like "sort=4") appears very frequently across many pages from the same site, it tends to be either presentation segment or content irrelevant segment. Take the typical second-level URLs extracted from buy.com in Table 7 as an example. One could expect that *all* result pages returned should share certain presentation logics and contain similar URLs — for example all pages would contain some URLs that allow items to be sorted by price (with query segment "sort=4"), URLs that advance to a different page ("page=3"), etc. In addition, if some pages have URLs embedded with query segments for click source tracking ("from=7"), then since result pages are generated in a homogeneous manner, most likely other pages will also contain URLs with the same tracking query segments. The presence of these query segments in almost all pages indicates that they are not specific to the input keyword query, thus likely to be either presentational (sorting, page number, etc), or content irrelevant (internal tracking, etc). On the other hand, certain query segments, like manufacturer name or subcategory, are sensitive to the input queries used. For example only when query related to gps are searched, will the segment representing manufacturer "Garmin" ("mfgid=-652") or "Tomtom" ("mfgid=-652") appear. Pages crawled with other entity queries are likely to contain a different set of query segments for different manufacturers. In this case, a specific query segment for manufacturer names is likely to exist on *some*, but not *all* crawled pages.

To capture this intuition we first define the notion of prevalence at the argument-value pair level and at the argument level.

DEFINITION 4. *Let $\mathcal{P}_s$ be the set of search result pages from the same deep-web site s, and $p \in \mathcal{P}_s$ be one such page. Further denote $D(p)$ as the set of argument-value pairs (query segments) in second-level URLs extracted from p, and $D(\mathcal{P}_s) = \cup_{p \in \mathcal{P}_s} D(p)$ as the set of all possible argument-value pairs from pages in site s.*

*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 argument a, denoted as $r(a)$, is defined as $r(a) = \frac{\sum_{(a,v) \in D(\mathcal{P}_s)} r(a,v)}{|\{(a,v) | (a,v) \in D(\mathcal{P}_s)\}|}$.*



(a) Argument level precision/recall (b) URL level recall

Figure 10: Precision/recall of empty page filtering

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 "sort=", for example is averaged from "sort=1", "sort=2", etc).

The aggregated prevalence score at argument level produces a robust prediction of the prevalence of the argument. Because argument with 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 irrelevant (if "sort=" has a high enough prevalence score, all "sort=?" are treated as irrelevant). Second-level URLs from the same site can then be partitioned by disregarding any query segment with relevance higher than θ, as in Table 7. URLs in the same partition are treated as semantic duplicates, and only one URL in the same partition needs to be crawled[1].

To evaluate the effectiveness of our URL deduplication, we randomly sample 10 deep-web sites, and manually label all arguments above threshold 0.01 as either relevant or irrelevant for deduplication. 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, in the experimental batch of 35M documents alone, there are 1471 different arguments from overstock.com, 1243 from ebay.co.uk, etc. Furthermore, ascertaining semantic meaning and the relevance of arguments that appear very infrequently can be increasingly hard. As a result we only evaluate arguments with prevalence score of at least 0.01.

Figure 10a shows the precision/recall of 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 predict 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 setting we use for our crawl system.

The second experiment in Figure 10b 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.

[1]Note we pick one URL in each partition to crawl instead of aggressively removing irrelevant query segments to normalize URLs, because there exist cases where removing query segments can invalidate the URL altogether.

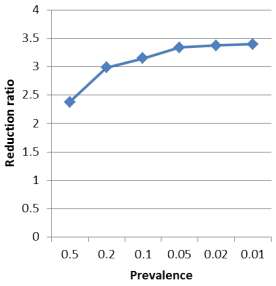


Figure 11: Reduction ratio of URL deduplication

cated. 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. We then evaluate the percentage of URLs that will be mistakenly deduplicated (loss of content that can could have been crawled) due to misprediction. The graph shows that at 0.1 level, only 0.7% of URLs are incorrectly deduplicated.

Finally Figure 11 shows the number of second-level URLs that can be deduplicated when using the proposed approach. As can be seen, the reduction ratio varies from 2.3 to 3.4, depending on prevalence threshold. Note that since the number of second-level URLs are in the order of magnitude of billions, our deduplication approach represents significant savings in crawl traffic.

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.

7. CONCLUSION

In this work we develop a prototype system that focuses on crawling entity-oriented deep-web sites. Focusing on entity-oriented sites allow us to optimize our crawl system by leveraging certain characteristics of these entity sites. Three such optimized components are described in detail in this paper, namely, query generation, empty page filtering and URL deduplication. Given the ubiquity of entity-oriented deep-web sites and the variety of possible entity-oriented processing using their content, developing further techniques to optimize entity-oriented crawl can be a useful direction for future research.

8. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[2] Z. Bar-yossef, I. Keidar, and U. Schonfeld. Do not crawl in the dust: different urls with similar text. In *In Proc. 15th WWW*, pages 1015–1016. ACM Press, 2006.

[3] L. Barbosa and J. Freire. Siphoning hidden-web data through keyword-based interfaces. In *In SBBD*, pages 309–321, 2004.

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

[5] L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 441–450, New York, NY, USA, 2007. ACM.

[6] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *In ICDE*, pages 431–440, 2002.

[7] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, pages 1247–1250, 2008.

[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international conference on World Wide Web 7*, WWW7, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[9] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *The sixth international conference on World Wide Web*, pages 1157–1166, Essex, UK, 1997. Elsevier Science Publishers Ltd.

[10] E. Chu, A. Baid, X. Chai, A. Doan, and J. Naughton. Combining keyword search and forms for ad hoc querying of databases. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 349–360, New York, NY, USA, 2009. ACM.

[11] 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*, KDD '08, pages 186–194, New York, NY, USA, 2008. ACM.

[12] J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 267–274, New York, NY, USA, 2009. ACM.

[13] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the deep web. *Commun. ACM*, 50:94–101, May 2007.

[14] M. A. Hearst. Uis for faceted navigation recent advances and remaining open problems.

[15] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *In VLDB*, pages 670–681, 2002.

[16] A. Jain and M. Pennacchiotti. Open entity extraction from web search query logs. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 510–518, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[17] 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 the third ACM international conference on Web search and data mining*, WSDM '10, pages 381–390, New York, NY, USA, 2010. ACM.

[18] 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 *In Proc. of CIDR-07*, 2007.

[19] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's deep web crawl. In *Proc. VLDB Endow.*, volume 1, pages 1241–1252. VLDB Endowment, August 2008.

[20] A. Ntoulas. Downloading textual hidden web content through keyword queries. In *In JCDL*, pages 100–109, 2005.

[21] M. Paşca. Weakly-supervised discovery of named entities using web search queries. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 683–690, New York, NY, USA, 2007. ACM.

[22] Y. Qiu and H.-P. Frei. Concept based query expansion. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '93, pages 160–169, New York, NY, USA, 1993. ACM.

[23] S. Raghavan and H. Garcia-molina. Crawling the hidden web. In *In VLDB*, pages 129–138, 2001.

[24] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. In *INFORMATION PROCESSING AND MANAGEMENT*, pages 513–523, 1988.

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

[26] Y. Wang, J. Lu, and J. Chen. Crawling deep web using a new set covering algorithm. In *Proceedings of the 5th International Conference on Advanced Data Mining and Applications*, ADMA '09, pages 326–337, Berlin, Heidelberg, 2009. Springer-Verlag.

[27] P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma. Query selection techniques

for efficient crawling of structured web sources. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 47–, Washington, DC, USA, 2006. IEEE Computer Society.

APPENDIX