ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/284019225

OpinoFetch: a practical and efficient approach to collecting opinions on arbitrary entities

Article *in* Information Retrieval · October 2015

DOI: 10.1007/s10791-015-9272-0

CITATIONS
0

READS
34

2 authors, including:

Kavita Ganesan

19 PUBLICATIONS **384** CITATIONS

SEE PROFILE

All content following this page was uploaded by Kavita Ganesan on 23 November 2015.

The user has requested enhancement of the downloaded file.

OpinoFetch: A Practical and Efficient Approach to Collecting Opinions on Arbitrary Entities

Kavita Ganesan, University of Illinois at Urbana Champaign
ChengXiang Zhai, University of Illinois at Urbana Champaign

The abundance of opinions on the Web is now becoming a critical source of information in a variety of application areas such as business intelligence, market research and online shopping. Unfortunately, due to the rapid growth of online content, there is no one source to obtain a comprehensive set of opinions about a specific entity or a topic, making access to such content severely limited. While previous works have been focused on mining and summarizing online opinions, there is limited work on exploring the automatic collection of opinion content on the Web. In this paper, we propose a *lightweight* and *practical* approach to collecting opinion containing pages, namely review pages on the Web for arbitrary entities. We leverage existing Web search engines and use a novel information network called the FetchGraph to efficiently obtain review pages for entities of interest. Our experiments in three different domains show that our method is more effective than plain search engine results and we are able to collect entity specific review pages efficiently with reasonable precision and accuracy.

ACM Reference Format:
Kavita Ganesan and ChengXiang Zhai, 2014. OpinoFetch: A Practical Approach to Collecting Opinions for Arbitrary Entities *ACM* V, N, Article A (January YYYY), 24 pages.
DOI : http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

With the surge of Big Data capabilities, the abundance of opinions expressed by experts and ordinary users on the Web is now becoming vital to a wide range of applications. For example, market research tools may use opinions to determine if a product is worth developing or marketing. Business intelligence applications may use online opinions to understand what users like or dislike about a recently launched product. Another important usage is with online shopping sites where these sites can utilize existing opinions on the web to help users make purchase decisions. While there is a clear need for large amounts of opinion content about a topic or entity (e.g. person, product or business), access to such content is very limited as opinions are often scattered around the web and the web is inherently dynamic. Consider the task of collecting opinions about the *iPhone 5*; one can find related opinions on well known e-commerce sites such as Amazon.com and BestBuy.com, within popular review sites such as CNET.com and Epinions.com and on less mainstream sites such as Techradar.com and also on personal blog sites. It is clear that there is no central repository to obtain all the opinions about an entity or topic. Moreover, the set of sites that contain reviews about one entity may not contain reviews about another similar entity. This makes the task of developing computational techniques for *collecting online opinions* at a large scale a new and interesting research challenge with a pressing need.

Online opinions are present in a wide range of sources including user review websites (e.g. Yelp.com, Edmunds.com), personal and professional blog sites, micro-blog sites like Twittter, social networking sites like Facebook, comments on news articles and more. In this paper, we focus primarily on collecting any type of *user review* page (expert or non-expert) pertaining to an entity. This is because user reviews alone make-up a huge portion of the Web. For example, user generated reviews can be found in most e-commerce applications (e.g. Amazon.com, Walmart.com and eBay.com), specialized user review websites (e.g. Yelp.com), vertical search engines (e.g. Hotels.com) and online directories (e.g. Yellowpages.com). Thus, a comprehensive set of user reviews alone would be highly valuable to many opinion dependent applications.

Intuitively, the simplest way to collect online reviews, is simply to crawl the entire web and then identify opinion containing pages on entities of interest. While in theory this method seems feasible, in practice it is actually intractable as (1) visiting and downloading all pages on the web would be very time consuming and places high demands on network and storage resources and (2) it would

become very expensive to perform relevance classification on each page from the web. Thus, a more reasonable method to solving this problem would be to use a focused crawling strategy.

Existing focused crawlers [16] which are pre-dominantly supervised, are designed to crawl a comprehensive set of documents pertaining to a particular topic such as *'War in Iraq'* and *'Obamacare'*. Thus, the *type* of page or document (e.g. review page, news page, blog article, etc.) is not as important as the content of the page itself. Also, these focused crawlers are designed to collect many pages for a handful of topics thus the use of a supervised crawler to determine page relevance is reasonable. In contrast, the task of collecting review pages for a set of entities is slightly different since the target is essentially a particular type of page (i.e. review page) related to a specific type of entity (e.g. iPhone 5s) and the number of entities could span hundreds or thousands. Thus, a purely supervised approach would demand large amounts of training examples (for each entity) making existing focused crawlers not practical for the task of entity specific review collection.

With this, we propose **OpinoFetch**, a practical framework for collecting online reviews for *arbitrary* entities. Our key idea is to first obtain an initial set of candidate review pages for a given entity using an existing Web search engine. We then expand this list by exploring links around the neighborhood of the search results. We model all the collected pages and the relationship between the pages and entities, pages and sites, etc using a novel information network called the *FetchGraph*. This network serves as a data structure to help with efficient lookup of various statistics for entity relevance scoring and review page likelihood scoring and supports application specific querying and filtering of pages.

Compared to all previous work, the main advantage of our approach is that it assumes *no domain knowledge* and thus can work across any domain (e.g. hotels, cars, doctors, etc.). This is to provide a very general and practical approach to finding online opinions which is immediately usable in practice. Evaluation results in three different domains (i.e. attractions, hotels and electronics) show that our approach to finding entity specific review pages far exceeds Google search and we are able to find such review pages with reasonable *efficiency*, *precision* and *accuracy*. In summary, the contributions of this work are as follows:

(1) We propose a highly practical framework for discovering review pages of arbitrary entities leveraging existing Web search engines.
(2) We introduce FetchGraph, a novel information network for intuitive representation of complex crawl information and efficient lookup of key statistics.
(3) We create the first test set for evaluating this review page collection problem.
(4) We evaluate the proposed methods in three domains and show that our approach is capable of finding entity specific review pages efficiently with reasonable efficiency, precision and accuracy.

The source code of our work and evaluation set is available at http://kavita-ganesan.com/content/opinofetch

2. RELATED WORK

Over the last decade, the focus of most works on dealing with opinions has been on predicting polarity rating of opinions at different granularity (i.e. positive or negative at the document level, passage level or feature level) [18; 21; 14; 11; 10]. In more recent years, there has been several works focused on generating unstructured summaries of opinions [6; 7; 8] and also other types of textual summarization such as highlighting contradicting opinions in text [13]. While all these works attempt to help users better digest opinions, there is limited work on automatically collecting content that could potentially contain opinions about an entity or topic for a more comprehensive opinion analysis. Any form of opinion mining and summarization is currently performed on a small portion of opinion containing documents. The problem with relying on opinions from just one or two sources is data sparseness and source related bias which could result in inaccuracies in information presented to the end user. In this work, we propose a practical method to automatically crawl review pages on the Web (which typically contain an abundance of opinions) for an arbitrary set of entities.

The concept of focused crawling was introduced by Chakrabarti et. al. [2] where the goal is to download only web pages that are relevant to a query or set of topics. Rather than collecting and indexing all accessible web documents, a focused crawler analyzes its crawl boundary to find the URLs that are likely to be most relevant for the crawl, and avoids irrelevant regions of the web. While early focused crawling methods were based on simple heuristics [4; 9], most topical crawlers in literature are predominantly supervised machine learning based methods [2; 1; 5; 3; 15; 12]. There are some key commonalities amongst these topical crawlers. First, topical crawlers are primarily interested in the relevance of a page to a topic. While initially topical relevance was determined using simple heuristics, topical crawlers generally rely on classifiers that require labeled examples. Next, in most topical crawlers, URLs on pages considered 'relevant' are prioritized for further crawling.

While conceptually our task is similar to a traditional topical crawling task, there are some key differences that makes our task unique. In traditional topical crawling, relevance has mostly to do with how relevant a page is to the *topic of interest* regardless of the type of page. In our task however, the goal is to collect review pages for a specific set of entities (which could span hundreds), and thus relevance is about (1) if a candidate page is a review page and (2) if a candidate page is relevant to one of the target entities. Next, in order to collect reviews for arbitrary entities the approach should be general enough to work across domains. Most topical crawlers however are domain dependent as they are trained on data from the domain of interest.

While the work of Vural et. al [22] focuses on discovering any opinion containing documents on the Web, this work is closer to a general crawl where in crawling Web pages, the crawl is limited to sentiment containing pages using a URL prioritization strategy. In addition, unlike traditional topical crawling, in the work of Vural et. al, there is no need for an opinion containing page to be relevant to a specific topic or entity as long as the content is subjective. In our work however, we require that an opinion containing page is relevant to a desired set of entities. In addition, since we use the results of a search engine as a starting point, we focus on a *short crawl* rather than a long crawl as in Vural et. al.

3. A GENERAL AND PRACTICAL FRAMEWORK FOR REVIEW PAGE COLLECTION

Given a set of entities, $E = \{e_1, ..., e_n\}$, from any domain, $\mathcal{D}$, the goal of our task is to find a complete set of online review pages denoted as $R_i = \{r_{i1}, ..., r_{in}\}$, where R_i contains a set of relevant review pages about e_i. Entities refer to objects on which reviews are expressed. This can be *businesses* such as hotels and restaurants, *people* such as politicians and doctors and *products* such as smart phones and tablet computers. The domain is a broad category and the granularity of the domain (e.g. smart phones vs. all mobile devices) depends on the application need.

While opinions are present in a wide range of sources as explained in Section 1, in this paper, we focus specifically on collecting review pages that contain reviews about a single entity. We thus define a review page r_i as any dedicated page on the Web focused on reviewing a single entity $e_k \in E$. Such a review page can consist of one long review or several short reviews by different users. For example, for an iPhone 5s, review pages for that device may consist of (1) a blog article reviewing the iPhone 5s, (2) several pages on Amazon.com containing a series of reviews about the iPhone 5s and (3) a page from CNET.com containing an expert review about the iPhone 5s. Discussion on forums about *'Problems with iPhone 5s'* or *'How to use certain features of the iPhone 5s'* would not qualify as a review page as these pages while may contain spurts of opinions about the iPhone 5s, can also be intermingled with usage questions, responses and other non-relevant information and is not dedicated to reviewing the device. For the same reason, we also do not consider pages that contain reviews about two or more entities (e.g. iPhone 5s vs. Samsung Galaxy S5) as a potential review page.

Our problem set-up allows for flexible adjustment of entities of interest according to the application needs and with this, the proposed framework would mainly focus on finding opinion pages about these target entities. This is to support a typical application scenario where a large number of reviews is needed for a specific set of entities (e.g. all hotels in the United States).

There are several challenges to solving this special task of review page collection for large number of *arbitrary* entities. The first challenge is finding review pages that match up to the entities of interest which can be from different domains. Typically, such a problem is solved using a completely supervised approach which would demand large amounts of training data. However, this is not practical as training data would be needed for each target entity and the list of entities can vary and can get quite large depending on the application.

The second problem is that there is no easy method for obtaining starting points for the crawl. Unlike commonly crawled domains such as news and blog domains where published RSS feeds are easily obtainable for use as starting points, obtaining an initial set of seed pages for review page collection is not as easy. If we used links from one specific review site as seeds, aside from being able to crawl all reviews from that site, there is no guarantee that this would take the crawler to other review sites. We also cannot rely on a fixed set of sites to obtain entity specific reviews because review containing sites are often incomplete. If one site has reviews for entity A this does not guarantee reviews for entity B even if they are closely related (e.g. reviews on iPhone 4s and iPhone 5).

To address these challenges, we propose a very general and *practical* review page collection framework capable of collecting review pages for *arbitrary* entities, by leveraging an existing Web search engine. The framework consists of the following steps:

Step 1. Obtaining an initial set of Candidate Review Pages (CRP): Given an entity e_k, we obtain an initial set of Candidate Review Pages (CRP) using a general Web search engine. This is used in conjunction with an *entity query*, a special query requesting review pages related to the target entity. σ_{search} controls the number of search results.

Step 2. Expanding the CRP list: We then expand the CRP list by exploring links around the neighborhood of each initial CRP, building a larger and more complete list of potential review pages. σ_{depth} controls how far into the neighborhood the exploration should happen. Intuitively, the more pages we explore, the more chances of recovering relevant review pages.

Step 3. Collecting entity related review pages: Next, all the pages in the expanded CRP list along with the initial CRPs are scored based on (1) *entity relevance* and (2) *review page likelihood*. Both these scores are used to eliminate irrelevant pages from the CRP list retaining a set of relevant review pages for each $e_k \in E_{\mathcal{D}}$. We will now expand on the details of each of these steps.

3.1. Obtaining Initial Candidate Review Pages

Since most content on the web are indexed by modern web search engines such as Google and Bing, review pages of all sorts of entities would also be part of this index. Also, modern search engines are very effective at finding pages about entities. Capitalizing on this fact, we can leverage web search engines to find the initial CRPs. We can do this by using information about the entity as the query (e.g. entity name + address or entity name + brand) along with biasing keywords such as *reviews* or *product reviews*. This is called the *entity query*. For example, the entity query for reviews of a hotel in Los Angeles would be similar to: *Vagabond Inn USC Los Angeles reviews*. The hope is that the results of such a query would consist of pointers to review pages about the target entity or have relevant review pages somewhere in the neighborhood.

We can thus treat the results of the search engine as an entry point to collecting a more complete and accurate set of pointers to entity related review pages. There are several advantages of doing this. First, search engines can help discover *entity specific review sites* as different sites would hold reviews for different subsets of entities even within the same domain. As an example, when we compare the search results for the query *iPhone 3g reviews* and *iPhone 4 reviews* on Google, we will find that there are sites that contain reviews for one of these products but not the other and vice versa. Next, since web search engines are effective in finding pages about entities, the task of matching reviews to an entity is already partially done by the search engine.

Since we use different search results for different entities, our task is more of a *hyper-focused data collection task*, as the search results are already quite 'close' to the relevant content. This is

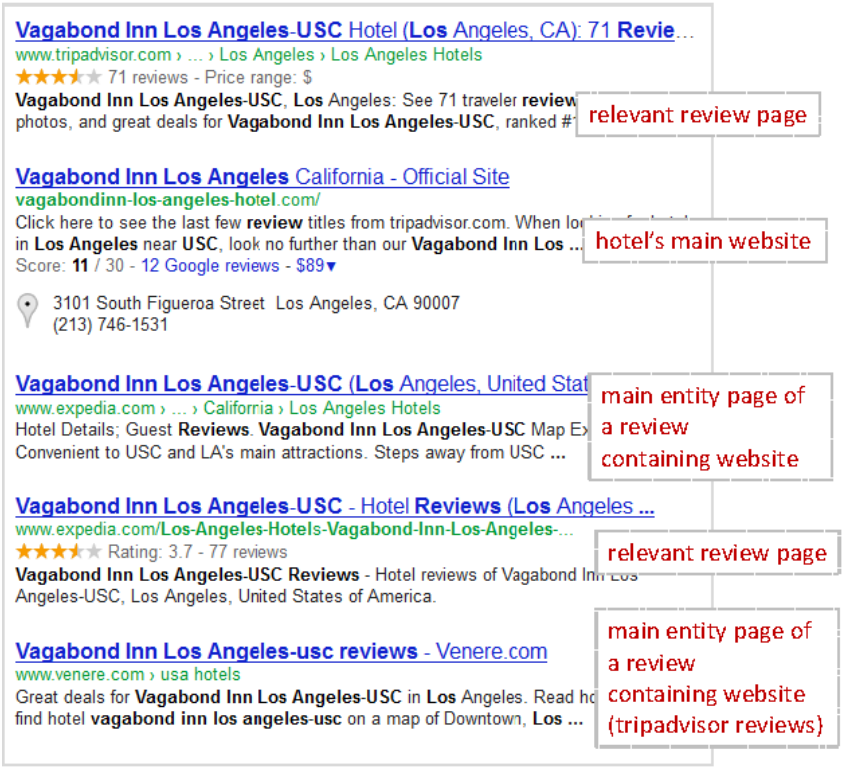

Fig. 1: Example search results from Google for the query *Vagabond Inn USC Los Angeles reviews*. Only two pointers are to actual review pages.

unlike traditional topical crawling where search results are used as a very general starting point of a long crawl. One may argue that the results of search engines alone are sufficient in finding entity related reviews. While some of the search engine returned pointers are valid links to review pages, there are many pointers that are not. In Figure 1, we show snapshot of results from Google for the query *Vagabond Inn USC Los Angeles reviews*. From this, we can see that only two out of five items point to valid review pages. The second item is pointer to the hotel's homepage. The third and fifth items point to sites that contain reviews about the hotel, but the link is to the main entity page rather than the actual review page. To address these problems, we propose to expand the CRP list.

3.2. Expanding CRP List

While it is possible to expand all URLs in a page for further exploration, this is a waste of resources as some URLs are known to be completely irrelevant (e.g. 'contact us' page and 'help' page). We propose a simple and effective URL prioritization strategy that attempts to bias the crawl path towards entity related pages. To achieve this, we measure the average cosine distance between (1) terms within the *anchor text* and the *entity query* and (2) *URL tokens* (delimited using special characters) and the *entity query*. Thus, the more the anchor text or URL resemble the entity query, the more likely that this page is relevant to the target entity. In each page, we can use the top N scoring links for further exploration until the chosen depth (σ_{depth}) is reached. N here can be a constant or a percentage of links. While more sophisticated strategies are possible, optimizing this step is not the focus of our work, we thus leave this as a future work.

3.3. Collecting Entity Related Review Pages

During the course of expanding the CRP list, we would naturally encounter many irrelevant pages to get to relevant ones. We thus need a method to eliminate the irrelevant pages. A page can thus be scored for (a) *review page likelihood* denoted by $S_{rev}(p_i)$ and (b) *entity relevance* denoted by $S_{ent}(p_i, e_k)$ where p_i is a page in the crawled set and e_k is the entity for which p_i appeared as a CRP. With this, to determine if a page p_i is relevant to an entity e_k, we need to check if $S_{rev}(p_i) > \sigma_{rev}$ and $S_{ent}(p_i, e_k) > \sigma_{ent}$ where σ_{ent} and σ_{rev} are two thresholds to be empirically set. While it may be possible to combine scoring of (a) and (b) into a single scoring approach, separating the scores provides more control on how each aspect is scored. Moreover, separating the scores would give the ability to give higher priority to one aspect than the other.

The proposed framework does not put any restriction on how to define the $S_{rev}(p_i)$ and $S_{ent}(p_i, e_k)$ scoring. Below we present several reasonable instantiations that we evaluate in our experiments. Since our emphasis is on the practicality of the approach and efficiency, we propose methods that require minimal supervision, ensuring flexibility (i.e. can work across entities, domains and languages) as well as accuracy.

3.3.1. Heuristics Based Review Page Classification. To determine if a page is a review page heuristically, we define a scoring function that utilizes a review vocabulary. Based on our observation that most review pages have similarities to a certain degree both in terms of structure and usage of words, we construct a review vocabulary consisting of the most common *review page indicating* words. For this, we manually obtained 50 different review pages from various web sources covering a wide range of domains such as electronics, software tools, doctors, hotels and others. We discarded common stop words from these pages using the stop word list from the Terrier Retrieval Toolkit [17]. We then rank each remaining term t in the combined vocabulary of the 50 review pages, denoted as R as follows:

$$Rank(t, R) = SiteFreq(t, R) * AvgTF(t, R) \tag{1}$$

where $SiteFreq(t, R)$ corresponds to how many web sources the term t occurred in and $AvgTF(t, R)$ is the sum of normalized term frequencies of a term t across all review documents that contain t. This is defined as:

$$AvgTF(t, R) = \frac{1}{n} \sum_{i}^{n} \frac{c(t, r_i)}{MaxTF_{r_i}} \tag{2}$$

In Equation 2, n is the total number of review documents containing t. With this, the more popular a term t is across sources and the higher its average term frequency, the higher the rank this term would have. This helps review page specific terms to emerge rather than domain specific words. Example of top terms from our review vocabulary are as follows: *review*, *helpful*, *services*, *rating*, *thank*, *recommend*. The top 100 terms and their corresponding weights, that is the $AvgTF(t, R)$ are included in the final review vocabulary. The $AvgTF(t, R)$, would act a weighting component in computing the $S_{rev}(p_i)$ score where we define $wt(t)$ as:

$$wt(t) = AvgTF(t, R) \tag{3}$$

To score a page for review page likelihood using the review vocabulary, we use the following formula:

$$S_{rev}(p_i) = \frac{\sum_{t \in V} log_2\{c(t, p_i)\} * wt(t)}{normalizer} \tag{4}$$

where t is a term from the defined review vocabulary, V and $c(t, p_i)$ is the frequency of t in p_i and $wt(t)$ is the importance weighting given to term t using the AvgTF(t,R) as defined in Equation 3. If we used only raw term frequencies, this may artificially boost the $S_{rev}(p_i)$ score even if only one of the terms in V was matched. Thus, we use log to scale down the frequencies. $wt(t)$ is important because it tells the scoring function which terms are better indicators of a review page. For example, terms like *review* and *rating* are better indicators than terms like *date*. Intuitively, a review page would have many of the terms in V with reasonably high term frequencies.

Since, we perform a sum of weights over all terms in V, the $S_{rev}(p_i)$ value can become quite large for highly dense review pages and this would make it difficult to set thresholds. To overcome this problem $S_{rev}(p_i)$ is normalized with a $normalizer$. We evaluate two normalization options where with the first option we explore several self adjustable normalizers based on overall review densities. For the second option, we propose a fixed normalizer which acts as a saturation point where when scores exceed a certain level, the scores would automatically qualify these pages as potential review pages. We now describe each of these normalizers:

Self Adjustable 1: SiteMax Normalizer: If a particular site is densely populated with reviews, then many of the review pages within this site would have high review relevance scores. Similarly, if a site contains limited reviews, then its likely that many of the review pages would have low review relevance scores. Thus, if we normalize the raw $S_{rev}(p_i)$ using the maximum $S_{rev}(p_i)$ score from the site that it originates from, the score of a true review page would always be high regardless of density of the review site.

Self Adjustable 2: EntityMax Normalizer: In many cases, if an entity is highly popular, the user reviews on that entity would accordingly be abundant. Similarly, if an entity is not so popular, then the amount of reviews on that entity would also be limited. This is usually true across websites. For example, reviews on the *iPhone* is abundant regardless of the review containing site. By using the maximum $S_{rev}(p_i)$ score of all pages related to a particular entity as a normalizer, a review page of an unpopular entity would still receive a high score because the maximum $S_{rev}(p_i)$ score would not be very high.

Self Adjustable 3: GlobalMax Normalizer: When there is a limited number of collected pages for a given entity (e.g. uncommon entities such as a particular doctor) the EntityMax normalizer could become unreliable. Also, when there is only one page collected from a particular site (e.g. a blog page), the $S_{rev}(p_i)$ score using the SiteMax normalizer would be artificially high as it is normalized against itself. To help with cases where the EntityMax or SiteMax normalizers are unreliable, we can use the maximum $S_{rev}(p_i)$ score based on all collected pages.

Fixed Value Normalization (FixedNorm): To set a fixed value $normalizer$, we essentially looked at the highest scoring pages in our entire crawl set using two different review vocabulary sources. Even though our crawl set spans different domains and we used two different review vocabularies, most of the highly populated review pages had a unnormalized $S_{rev}(p_i)$ score above 100 and the highest scoring review pages had an unnormalized $S_{rev}(p_i) > 200$. We empirically selected 150 as the $normalizer$ value. Note that using a smaller value (e.g. 120) did not show any significant difference in performance. However, using a larger value like 200 affected precision. With this $normalizer$ value, Equation 4 now becomes:

$$S_{rev}(p_i) = \frac{\sum_{t \in V} log_2\{c(t, p_i)\} * wt(t)}{150} \tag{5}$$

3.3.2. Language Modelling Approach to Review Page Classification (RLM). Another review page classification option that we propose is to extend the concept of the review vocabulary proposed in Section 3.3.1 and build a *review language model* and a *background language model*. We refer to

this as method as ***RLM***. The idea here is that if a page scores highly according to a review language model compared to a background model, then the page can be considered a review page. Otherwise, the page is considered a non-review page. This is analogous to a Naive Bayes classifier with the exception that we try an additional smoothing technique in addition to the traditional Laplace smoothing used with Naive Bayes (also known as add-one smoothing).

In essence, we have two classes. One is the 'review page' class, denoted by C_r and the other is a 'background' class denoted by C_b. To construct the review language model for C_r we expanded the review pages used for the construction of the review vocabulary to 100 review pages. The contents of each of these review pages were used to construct a unigram review language model. For the background language model on the other hand, we obtained another 100 pages that were non-review pages. These consisted of news articles, forum pages, unrelated blog articles, tweets and etc. Given these two language models, we then compute the probability that a given page, p_i in the collection was generated by one of these models. Formally, given a page p_i, the probability that this page was generated by C_r or C_b is computed as follows:

$$P(C_k|p_i) \propto \prod_{t_u \in p_i} P(t_u|C_k) * P(C_k) \tag{6}$$

where, $C_k \in \{C_r, C_b\}$ and t_u is a term from p_i. Thus, if $P(C_r|p_i) > P(C_b|p_i)$, then $S_{rev}(p_i) = 1$ (p_i is a review page) otherwise $S_{rev}(p_i) = 0$ (p_i is a non-review page). Since not all terms within a page would appear in the language models, we use two different smoothing techniques. Specifically, we use Laplace smoothing (RLM_Laplace) as well as Dirichlet smoothing (RLM_Dirichlet) to deal with unseen terms instead of assigning a probability of '0' [23]. The benefit of using a language modelling approach to review page classification is that we are actually re-using state-of-the-art information retrieval based scoring methods. In addition, we can also use different smoothing techniques as seen fit by the system. Note that on our test set consisting of about 200 test cases with a mix of review and non-review pages, the RLM approach achieves precision of 0.92 and recall of 0.80 with Dirichlet smoothing and a precision of 0.94 and recall of 0.75 with Laplace smoothing.

3.3.3. Entity Relevance Scoring. A review page is often related to a single entity. Even though there may be review pages comparing two or more entities together, we are only interested in review pages that are centered around a single entity. Also, even though a page p_i may be a review page, it may not necessarily be relevant to the target entity e_k of interest. To eliminate such irrelevant pages, we need an *entity relevance* score measuring how relevant p_i is to e_k. We explore several entity relevance scoring strategies based on on some of the observations we have made about commonalities in review pages:

TitleOnly: We have noticed that since a typical review page is centered around one entity, the page title tends to contain terms describing the entity. Thus, we can compute the similarity of the *entity query* to the *title of a page* to determine entity relevance.

URLOnly: We have also observed that most URLs pointing to review pages, are often very descriptive, containing full or partial description of the entity. For example, the URL for a hotel review on Tripadvisor is as follows: *http://www.tripadvisor.com/Hotel_Review-g60950-d74362-Reviews-BEST_WESTERN_PLUS_Royal_Sun_Inn_Suites-Tucson_Arizona.html*. As can be seen, this URL has the full description of the entity. Based on this observation, we can thus compute the similarity of the *entity query* to the *URL of a page*.

ContentsOnly: Within the contents of review pages, there are often times mentions about the entity itself. For example, for a Nikon D1500 camera, a reviewer may say '*the Nikon D1500 is worth the buy, and I would certainly recommend it*'. This is evidence that the item being reviewed is the Nikon D1500. When there are multiple mentions of that sort, we get more and more evidence of relevance to a particular entity. Thus, the closer the similarity between the *entity query* and the

contents of a page the more relevant the page is to the entity of interest.

MaxURLTitle: In an unexpected circumstance, it could happen that the title of a review page may not contain description about the target entity or could be empty or the URL may in fact not be a descriptive path. Thus, we define another measure that takes the maximum score between **URLOnly** and **TitleOnly** where in the absence of one component, there is a score from the 'backup' component.

For the similarity measure of **URLOnly** and **TitleOnly**, we use a modified form of *Jaccard* [19] similarity. Jaccard is used for comparing the similarity and diversity of sample sets and is defined as the size of the intersection divided by the size of the union of the sample sets. We modify the Jaccard formula by changing the denominator to reflect how much of the entity query terms are captured by the page title or URL instead of using the union of terms between the entity query and page title or URL. This is because, in some cases, the page title or URL terms can be rather verbose containing a mix of irrelevant terms along with relevant terms resulting in an unfair intersection to union ratio. For more reliable scoring, we use only the entity query terms as the denominator. With this, the entity relevance of a page p_i with entity e_k, $S_{ent}(p_i, e_k)$ using the TitleOnly or URLOnly scoring is defined as follows:

$$S_{ent}(p_i, e_k)_{TitleOnly} = \frac{|T_{title_{p_i}} \cap T_{EQ}|}{|T_{EQ}|} \tag{7}$$

$$S_{ent}(p_i, e_k)_{URLOnly} = \frac{|T_{url_{p_i}} \cap T_{EQ}|}{|T_{EQ}|} \tag{8}$$

where T_{title} is a set containing page title tokens, T_{url} is a set containing all the URL tokens (delimited using special characters) and T_{EQ} is a set containing all the terms within the entity query. The Jaccard variation is ideal for these two texts because we are interested in measuring similarity between two pieces short texts. Also, since the modified Jaccard similarity score also ranges from $[0-1]$ this makes it easy to set the σ_{ent} cutoff.

For measuring the similarity between the entity query and the contents of a page for ContentsOnly scoring, we use the *Cosine similarity* measure [20] which is more appropriate for matching a query with the contents of a page as it takes term frequency into account. While one can utilize more advanced information retrieval based similarity measures such as Okapi BM25, we selected cosine because it provides a score range from $[0-1]$, making it ideal for threshold setting. With this, the entity relevance of a page p_i with entity e_k, $S_{ent}(p_i, e_k)$ for ContentsOnly scoring is computed as follows:

$$S_{ent}(p_i, e_k)_{ContentsOnly} = \frac{\vec{EQ} \cdot \vec{T_{p_i}}}{||\vec{EQ}||\ ||\vec{T_{p_i}}||} \tag{9}$$

where $\vec{EQ}$ is a vector of terms with corresponding counts from the *entity query* and $\vec{T_{p_i}}$ is a term vector with corresponding counts based on the *contents of a page*, p_i. A page p_i is relevant to an entity e_k if $S_{ent}(p_i, e_k) > \sigma_{ent}$.

4. FETCHGRAPH: A NOVEL INFORMATION NETWORK FOR REVIEW CRAWLING

In the previous section, we outlined a very general approach with a reasonable instantiation to finding entity related review pages using several lightweight and practical approaches. While the specific ideas proposed can be implemented in a multitude of different ways, our goal is to make the approach usable in practice. With heavy usage of network bandwidth and page parsing and processing, often times crawlers can become extremely inefficient and would not work for the desired

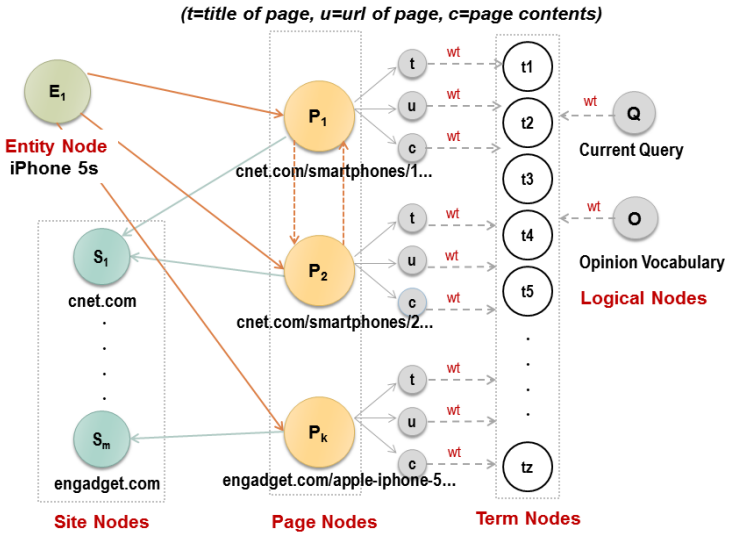

Fig. 2: Example FetchGraph for a single entity - *iPhone 5*. Dashed edges indicate compound edges. Gray nodes indicate logical (conceptual) nodes. Term nodes represent terms in the combined vocabulary of the data collection task (e.g. page content and URL tokens).

number of entities. We thus define two key aspects of usability in practice: (1) Efficiency of the algorithm - which is usually affected by the inability to store and lookup key statistics quickly and (2) Access to crawl information - for post crawl access and querying.

With this, we propose a novel data structure called the FetchGraph, which provides fast access to all sorts of crawl statistics and models complex relationships between different components in a data collection problem, enabling post crawl access and querying. Figure 2 shows an example of a FetchGraph for the problem of collecting review pages for a single entity, *iPhone 5*. Note that this is a partially complete graph used as an example. The first thing we would notice from Figure 2 is that the FetchGraph provides an intuitive view of the entire data collection problem as it models the complex relationships between the different components (e.g. entities, pages, sites, terms). Each component is represented by a simple node in the graph and relationships are modeled using edges. This graph can be used as an index as well as a lookup data structure. For example, to obtain the term frequency of a word in a given page, we would follow the path of looking for a specific *page node* $\rightarrow$ *content node* $\rightarrow$ *term node* which would give the frequency, without the need to repeatedly compute term frequencies for a given page. Another example is if the client application wants to only consider review pages from the top 10 most densely populated review sites. We can easily short list such pages by considering only sites with the highest cumulative $S_{rev}(p_i)$ scores, easily computed using the FetchGraph.

The key steps in our instantiation of the review page collection framework include (1) finding initial CRPs, (2) expanding the CRP List, (3) growing the FetchGraph and (4) pruning the FetchGraph to eliminate irrelevant pages. The first two steps are independent of the FetchGraph and are explained in Section 3. In the next few sub-sections, we will discuss the components of the FetchGraph, how to grow the FetchGraph and show examples of how to use it for computing $S_{rev}(p_i)$ and $S_{ent}(p_i, e_k)$.

4.1. Components of the FetchGraph

Typically, in a focused web page collection problem we would have a set of *web pages* where each of these pages originate from a specific *site*. Each web page is related to a specific topic or *entity* and at the very core, the web pages and its URL are made up of a set of *terms*. With this, we define 5 core node types of the FetchGraph: (1) entity nodes, (2) page nodes, (3) term nodes (4) site nodes and (5) logical nodes. In addition, we also have two application specific logical nodes: (6) OpinVocab node and (7) query node.

In formal terms, we denote entity nodes as $E_{\mathcal{D}} = \{e_i\}_{i=1}^{n}$ where $\mathcal{D}$ represents a domain type, page nodes as $P = \{p_i\}_{i=1}^{k}$, term nodes as $T = \{t_i\}_{i=1}^{z}$, site nodes as $S = \{s_i\}_{i=1}^{m}$ and logical nodes as L_X, where X represents the type of logical node. Figure 2 graphically illustrates how these nodes are connected.

A logical node is a conceptual node encapsulating a sub-component, a concept or a cluster of information. With this node, we can easily add semantics to the FetchGraph. The OpinVocab node, $L_{OpinVocab}$ is a logical node that encapsulates all terms in the review vocabulary (as constructed in Section 3.3.1). $L_{OpinVocab}$ would thus have edges to all relevant term nodes with edges holding the weight contribution of each term in the vocabulary. The query node, $L_{Query_{e_k}}$ is a logical node encapsulating the entity query for each entity e_k. To model the contents of an entity query, there is an edge from $L_{query_{e_k}}$ to all relevant term nodes. A page is made up of several sub-components (e.g. URL, title, contents). $L_{URL_{p_i}}$ represents the URL node, $L_{Title_{p_i}}$ represents the Title node and $L_{Content_{p_i}}$ represents the content node. These three logical nodes link to term nodes to model term composition. As we will see later, all of these logical nodes are actually used in relevance scoring.

4.2. Growing the FetchGraph

Algorithm 1 outlines the construction of the FetchGraph for review page collection. We start with the set of CRPs collected for a given entity e_k. For any incoming page, we check if the page already exists in the graph (based on page URL). The time complexity of this is O(1) as we do a hash lookup based on the page URL. If a page is an existing page, then only the ownership of the page to entity e_k is established. Entity ownership is revised if the $S_{ent}(p_i, e_k)$ score is larger for the current entity than the existing one (line 3-5).

Algorithm 1 $GrowFetchGraph(CRPList_{e_k}, e_k, G)$

1: **for** $CRP \in CRPList_{e_k}$ **do**
2: $p_i \leftarrow GetPageNode(CRP, G)$
3: **if** $pageExists(p_i, G)$ **then**
4: $S_{ENT}(p_i, e_k) = ComputeEntityRel(p_i, e_k)$
5: $UpdateEntityOwnership(p_i, e_k, S_{ENT}(p_i, e_k))$
6: **else**
7: **if** NOT $isNearDuplicatePage(p_i, G)$ **then**
8: $AddNode(p_i, L_{URL_{p_i}})$
9: $AddNode(p_i, L_{Content_{p_i}})$
10: $AddEdge(L_{Title_{p_i}} \rightarrow T)$ ▷ page title is made up of terms
11: $AddEdge(L_{URL_{p_i}} \rightarrow T)$ ▷ URL tokens is made up of terms
12: $AddEdge(L_{Content_{p_i}} \rightarrow T)$ ▷ content is made up of terms
13: $S_{rev}(p_i) = ScoreRevRel(p_i)$ ▷ review page likelihood scoring
14: $S_{ent}(p_i, e_k) = ScoreEntRel(p_i, e_k)$ ▷ entity relevance scoring
15: $AddEdge(e_k \rightarrow p_i)$ ▷ an entity owns the page
16: $AddEdge(p_i \rightarrow s_k)$ ▷ page is part of a site
17: **else**
18: $AddToDuplicateList(p_i, G)$
19: **end if**
20: **end if**
21: **end for**

If a page does not already exist in the graph, a check is done to see if the page is a near duplicate page to an existing page (line 7). This usually happens when there are multiple URLs linking to the same page. If a page is a near duplicate, then this page is added to the duplicate list of the existing page (line 18). Otherwise, a new node p_i, representing this new page is created. We will not discuss how near duplicates are detected since this is not the focus of our work and we also turn this feature 'off' during evaluation. Next, the page related logical nodes described in the previous section are created. These nodes are added to the parent page, p_i (line 8-9). Then, these logical nodes are linked to relevant term nodes based on the textual content within these components. For each component, there will be one edge for each unique term and the term frequencies are maintained at the edge level (line 10-12).

Once a page is added to the FetchGraph, the next step is to score the new page in terms of $S_{rev}(p_i)$ and $S_{ent}(p_i, e_k)$ to determine review page likelihood and entity relevance (line 13-14). Note that for the heuristics based approach to $S_{rev}(p_i)$ scoring with self-adjustable normalizers, only the unnormalized scores are initially computed. The RLM approach will return a value of '0' (non-review page) or '1' (review page) as the score.

After scores have been computed, other relationships in the FetchGraph are established. An edge from the entity node to the page node is added to indicate entity ownership (line 15). Entity ownership of a page depends on which entity the page appeared as a CRP. We also link the page with the site that it originates from (line 16).

Once all CRPs for all entities have been added to the FetchGraph, the $S_{rev}(p_i)$ score for the heuristics based approach with self-adjustable normalizers is normalized using dependency information from the graph. Then the graph is pruned based on the $S_{rev}(p_i)$ and $S_{ent}(p_i, e_k)$ relevance scores . This leaves us with a set of high confidence relevant review pages. Although pruning here is done at the very end, it can also be performed periodically as the graph grows.

4.3. Computing $S_{rev}(p_i)$ using FetchGraph

In Section 3.3.1, we presented several methods for scoring a page to determine review page likelihood using a heuristics based approach with several normalization strategies as well as a review language modelling approach (RLM). We will now explain how to compute the heuristics based $S_{rev}(p_i)$ scores which uses a review vocabulary. This review vocabulary can be easily modeled in the FetchGraph as described earlier where the terms and their contributing weights are encapsulated by $L_{OpinVocab}$.

Formally, given a page node p_i, its content node, $L_{Content_{p_i}}$ and the OpinVocab node $L_{OpinVocab}$, let T_c be all term nodes connected to $L_{content_{p_i}}$ and let T_{ov} be all term nodes connected to $L_{OpinVocab}$. For simplicity we refer to $L_{content_{p_i}}$ and $L_{OpinVocab}$ as L_c and L_{ov}. With this, the unnormalized $S_{rev}(p_i)$ score with respect to the FetchGraph is computed as follows:

$$S_{rev}(p_i) = \sum_{t \in T_c \cap T_{ov}} log_2[wt(L_c \rightarrow t)] * wt(L_{ov} \rightarrow t)$$

where $L_c \rightarrow t$ and $L_{ov} \rightarrow t$ refer to the connecting edges from L_c to t and L_{ov} to t.

To normalize the raw $S_{rev}(p_i)$ scores using the self-adjustable normalizers, we have several options as proposed in Section 3.3.1. Given P as all pages in the FetchGraph, let P_{e_k} be all pages connected entity node e_k and let P_s be all pages from a particular site s. With this the normalized scores are defined as follows:

GlobalMax :

$$S_{rev_{GM}}(p_i) = \frac{S_{rev(p_i)}}{\max_{p \in P}(S_{rev(p)})}$$

EntityMax :

$$S_{rev_{EM}}(p_i) = \frac{S_{rev(p_i)}}{\max\limits_{p \in P_{e_k}}(S_{rev(p)})}$$

SiteMax :

$$S_{rev_{SM}}(p_i) = \frac{S_{rev(p_i)}}{\max\limits_{p \in P_s}(S_{rev(p)})}$$

SiteMax + GlobalMax:

$$S_{rev_{SM+G}}(p_i) = 0.5 * S_{rev_{SM}}(p_i) + 0.5 * S_{rev_{GM}}(p_i)$$

EntityMax + GlobalMax:

$$S_{rev_{EM+G}}(p_i) = 0.5 * S_{rev_{EM}}(p_i) + 0.5 * S_{rev_{GM}}(p_i)$$

The subscripts GM, EM and SM represent GlobalMax, EntityMax and SiteMax normalization respectively.

The benefit of using the FetchGraph for this computation task is that a page is loaded into memory only once for construction of the FetchGraph. All other statistics right from term frequencies of page related terms to determining site specific pages (for SM computation) or entity specific pages (for EM computation) can be directly accessed from the FetchGraph without a separate inverted indexes or data structure.

4.4. Computing $S_{ent}(p_i, e_k)$ using FetchGraph

In Section 3.3.3, we propose to compute $S_{ent}(p_i, e_k)$ based on the similarity between the page URL, page Title and page contents with the entity query. Since a given page can appear in the CRP list of different entities, we will be computing the $S_{ent}(p_i, e_k)$ scores for the same page with different entities. Therefore, there will be repeated access to URL and page related terms. To manage the URL terms and page contents without re-parsing and re-tokenizing it each time, we can maintain a table in memory with the URL of a page as the key and the list of terms as the entries. While this approach will work very well for a small number of pages, as the list of URL's grow (as more and more pages are crawled), the table will become huge as we maintain separate lists of tokens for each page and the terms can be repetitive across lists. In the FetchGraph however, there are no duplicate terms as we maintain one unique node for a given term. The terms within a URL is modeled using edges to relevant term nodes. Similarly, the terms within the page title and page contents are also modeled using edges to relevant term nodes. With this, the growth of the graph is much more manageable (we show later that the FetchGraph's growth is linear to the number of pages).

Given an entity node e_k and page node p_i, where p_i is connected to e_k, the $S_{ent}(p_i, e_k)$ with respect to the FetchGraph using the TitleOnly or URLOnly scoring is computed as follows:

$$S_{ent}(p_i, e_k)_{TitleOnly} = \frac{|T_{Title} \cap T_Q|}{|T_Q|}$$

$$S_{ent}(p_i, e_k)_{URLOnly} = \frac{|T_{URL} \cap T_Q|}{|T_Q|}$$

where T_{URL} contains all term nodes connected to $L_{URL_{p_i}}$ (logical node representing the URL), T_{Title} contains all term nodes connected to $L_{Title_{p_i}}$ and T_Q contains all term nodes connected to $L_{Query_{e_k}}$ (logical node representing the entity query).

5. EXPERIMENTAL SETUP

We evaluate our proposed OpinoFetch framework in terms of precision, recall, $F_{0.4}$ score and accuracy. We also give insights into efficiency and provide examples of application related queries that

the FetchGraph can answer. For our evaluation, we use three different domains - *electronics*, *hotels* and *attractions* where each of these domains is quite different from one another.

Dataset and Queries. The electronics domain is highly popular with reviews in a variety of sources ranging from personal blog sites to expert review sites. The hotels domain while not as popular as electronics, has abundance of reviews on well known travel sites such as Hotels.com and Tripadvisor. The attractions domain is least popular on the web and the available reviews in each source is often incomplete. Even on big sites like Tripadvisor, the reviews in the attractions domain only covers a small portion of all available attractions. In our dataset, we have a total of 14 entities for which reviews are to be collected (5 hotels; 5 electronics; 4 attractions).

In finding the initial set of CRPs (details in Section 3.1), we use Google as the general Web search engine. For the *entity query* issued to the search engine, we use a descriptive query consisting of the *entity name*, *address* (if location specific) and the term *reviews*. Thus, for an attraction such as Universal Studios in Florida the resulting query will be *Universal Studios, Orlando Florida Reviews*. Throughout our evaluation, we primarily use the top 50 Google results for each entity query.

Evaluation Measures. As our task is more of a *hyper-focused data collection task*, the actual pages that need to be collected are already close to the starting points. Thus, the difficulty is in finding the actual relevant content around the vicinity of these starting points. With this, we focus on a short range crawl rather than a long crawl. We show later that distant URLs yield much lower gains in recall. Topical crawlers are usually evaluated by *harvest rate* which is the ratio between number of relevant and all of the pages retrieved [16; 2]. While it is interesting to see shifts in precision over number of retrieved pages for a long crawl, this is not so interesting for a short crawl where the number of pages crawled per entity is not very large. Thus, we measure *precision*, *recall*, $F_{0.4}$ score as well as *accuracy* after the FetchGraph has been pruned. We used $F_{0.4}$ to give more weight to precision.

For a given entity, e_k, let us assume that $\sum PredictedPositive(e_k)$ refers to the number of crawled pages for e_k that the system considers relevant. From this set, let us assume that $\sum TruePositive(e_k)$ refers to the subset of pages that are actually relevant according to the gold standard. Precision would thus be computed as:

$$Precision(e_k) = \frac{\sum TruePositive(e_k)}{\sum PredictedPositive(e_k)}$$

Assuming, $\sum RelPages(e_k)$ refers to the entire set of relevant pages in our gold standard for entity e_k, recall is thus computed as:

$$Recall(e_k) = \frac{\sum TruePositive(e_k)}{\sum RelPages(e_k)}$$

Given the precision and recall, the $F_{0.4}$ which gives slightly more weight to precision is computed as:

$$F_{0.4}(e_k) = (1 + 0.4^2)\frac{Precision(e_k) * Recall(e_k)}{(0.4^2 * Precision(e_k)) + Recall(e_k)}$$

Finally, accuracy is computed as:

$$Accuracy(e_k) = \frac{\sum TruePositive(e_k) + \sum TrueNegative(e_k)}{\sum RetrievedPages(e_k)}$$

where $\sum TrueNegative(e_k)$ refers to the number of pages that were correctly pruned and $\sum RetrievedPages(e_k)$ refers to the total number of crawled pages for e_k.

Defining true recall for this task is extremely difficult as there is no mechanism to obtain all relevant review pages about an entity from the web, nor is it easy to crawl the entire web and identify review pages pertaining to the entities in our evaluation set. Given that most pages on

the web are indexed by well known search engines, we approximate recall by constructing a gold standard judgment set that looks deeper into entity specific search results. Specifically, to construct our gold standard judgments, for each entity query in our evaluation set, we explore the top 50 results and follow links up to a depth of 3 in order to build a link repository. We then ask human judges (through crowdsourcing) to judge if a given URL points to a review page for the named entity. We had a total of 57,154 unique (entity query + URL) pairs which called for 171,462 judgment tasks using 3 human judges for each task. The majority voting scheme was used as the final judgment. To control the quality of the judgments, we introduced 50 gold standard judgment questions where for every few judgment tasks presented to the workers there will be a hidden gold standard question. If a worker misses too many of these gold standard tasks, then the contribution of this worker will be excluded.

While the constructed gold standard judgments can provide a good estimate of precision and recall, the actual precision and recall is actually higher. This is because, first, there are many URLs with minor differences that point to the same content. The precision and recall would be higher if we capture all of these URLs even though in reality, capturing at least one is equally good. Also, as our judgment set can include pages in other languages and it could be easy for a human to judge if some of the pages in other languages contain reviews or not. Our framework will most likely prune pages that are non-English and this further lowers precision and recall artificially. Eliminating duplicates for each URL in this judgment set and filtering based on language would be expensive and is unnecessary because ultimately what matters is the relative improvement in performance.

Baseline. Since there is no other relevant work that has explored the collection of entity specific review pages, we do not have a similar competing method for comparison. We thus use Google results as a baseline as these search results are deemed relevant to the entity query and are 'close' to the actual information need.

During evaluation, we turned off several extended features: We turned off the *duplicate elimination* module so we do not tie duplicate pages together; We place no restrictions on the type of URLs followed as there could be many file types that can be potentially eliminated; We also do not force crawling of English only content to enable future work in other languages and also to asses which methods are less sensitive to language.

6. RESULTS

By default, the following are the settings used throughout our evaluation unless otherwise mentioned. Number of search results, $\sigma_{search} = 50$; CRP expansion depth, $\sigma_{depth} = 1$; Entity scoring method: URLOnly; Review scoring method: Heuristics based with EntityMax+GlobalMax normalization; We set $\sigma_{rev} = 0.2$ and $\sigma_{ent} = 0.5$ as they perform the best for the combination of *URLOnly* and *EntityMax+GlobalMax* scoring methods.

Run Name	Description
OpinoFetch_RV_FixedNorm	Heuristics based review page scoring with fixed normalization
OpinoFetch_RV_EM+GM	Heuristics based review page scoring with entity max and global max normalization
OpinoFetch_RLM_Laplace	Review language model based review page classification with laplace smoothing
OpinoFetch_RLM_Dirichlet	Review language model based review page classification with dirichlet smoothing

Table II: Description of runs with different $S_{rev}(p_i)$ scoring methods.

	Recall			Precision			$F_{0.4}$ Score		
Run/Search Results Size	**10**	**30**	**50**	**10**	**30**	**50**	**10**	**30**	**50**
Google	0.0153	0.0210	0.0230	0.4941	0.4013	0.3776	0.093	0.115	0.121
OpinoFetch_RV_FixedNorm	0.0959	0.1093	0.1148	**0.7439**	**0.6003**	**0.5927**	0.385	0.371	0.377
OpinoFetch_RV_EM+GM	**0.0965**	**0.1132**	**0.1160**	0.7407	0.5942	0.5922	**0.386**	**0.375**	**0.378**
OpinoFetch_RLM_Laplace	0.0681	0.0791	0.0789	0.6449	0.5650	0.5619	0.297	0.306	0.305
OpinoFetch_RLM_Dirichlet	0.0843	0.0991	0.1007	0.6375	0.5696	0.5550	0.335	0.344	0.342

Accuracy			
Run/Search Results Size	**10**	**30**	**50**
Google	0.4941	0.4013	0.3776
OpinoFetch_RV_FixedNorm	**0.7982**	**0.7866**	**0.7825**
OpinoFetch_RV_EM+GM	0.7946	0.7798	0.7814
OpinoFetch_RLM_Laplace	0.6062	0.6908	0.7016
OpinoFetch_RLM_Dirichlet	0.6340	0.7006	0.6999

Table I: Performance at different search sizes. OpinoFetch_RV_FixedNorm has the highest overall accuracy and precision. OpinoFetch_RV_EM+GM has the highest recall and $F_0.4$ scores. Table II provides a brief description of the different OpinoFetch runs.

Baseline (Google Search) vs. OpinoFetch. In Table I, we report performance comparison of OpinoFetch (with different review page scoring methods) and Google at different search result sizes. Table II provides a brief description of the different OpinoFetch runs. Based on Table I, we see that the precision of Google search is low even though the number of search results is not very large (between 10 - 50). This shows that many of the search results that come up as part of the *entity query* issued are not direct pointers to review pages or are completely irrelevant to the entity query. Then, we can also see that with Google results there is limited gain in terms of recall even with increasing number of search results. This goes to show that while there exists many relevant pages in the vicinity of the search results, Google search does not capture these pages. One may argue that an entirely different query would improve these results. However, general search engines like Bing and Google serve typical users who want results fast and tend to use less descriptive queries than what was used in our evaluation. Therefore, we expect the search results using our entity query to be more accurate compared to a non-descriptive query (e.g. *Universal Studios Reviews* which shows mixed results between the one in Florida and Hollywood).

From Table I, we can also see that the performance of OpinoFetch based on the $F_{0.4}$ score alone is significantly better than plain Google search. The recall steadily improves when more and more search results are used since we perform CRP expansion as explained in Section 3.2. This shows that there is a lot of relevant content around the vicinity of the search results and our approach that looks for such relevant content is effective in that we are able to identify a lot of these relevant review pages. As we pointed out in Section 5, the actual recall and precision values would be higher if we discounted redundancies (almost identical URLs) and language barriers (pages in different languages other than English).

Performance comparison of different $S_{rev}(p_i)$ scoring methods. In Section 3.3, we proposed several methods for scoring a page to determine its likelihood of being a review page. Based

on Table I, we can see that the heuristics based scoring method with the use of a normalizer (OpinoFetch_RV_FixedNorm and OpinoFetch_RV_EM+GM) performs better than the review language modelling (RLM) approach (OpinoFetch_RLM_Laplace and OpinoFetch_RLM_Dirichlet) in terms of precision, recall as well as accuracy. Upon further investigation, we noticed that the RLM approach does not do well in classifying pages where there is a mix of user reviews along with other general description (e.g. general hotel information) and thus classifying relevant review pages as non-review pages. Unlike RLM, since the heuristics based approach does not use all words in a page to determine if a page is a review page or not, the presence of general description, advertising information, links to other pages and etc. are less distracting to this approach. The RLM approach also tends to work better on densely populated review pages and makes many mistakes when classifying review pages with just a few reviews. The heuristics approach leveraging a review vocabulary is less sensitive to shorter review pages except when there are no reviews at all or where there are several one liners.

Another interesting observation that we can see from Table I, is that the use of an adjustable normalizer (EM+GM) while provides better recall, based on the $F_{0.4}$ scores, we can see that we can achieve comparable performance just by using a fixed normalizer (FixedNorm) as described in Section 3.3.1. Thus, if computing the EntityMax and GlobalMax values becomes computationally expensive for an application, using the FixedNorm normalizer would work just as well.

Best normalization strategy for heuristics based $S_{rev}(p_i)$ scoring. To determine the best normalization strategy for heuristics based $S_{rev}(p_i)$ scoring, we look into the precision and $F_{0.4}$ scores using different types of normalizers across $\sigma_{rev} \in \{0.1, 0.3, 0.5\}$. We set $\sigma_{ent} = 0$ to turn off pruning based on entity relevance and set $\sigma_{depth} = 1$. The results are summarized in Table III. First, notice that all normalizers except SiteMax improve precision of the results over no pruning. Next, we see that the self-adjustable normalizer methods that incorporate EntityMax have higher levels of precision than the ones that incorporate SiteMax. This is reasonable, because a popular site like Tripadvisor would cover entities from different domains (e.g. hotels and attractions). Thus, the maximum the $S_{rev}(p_i)$ score from such a site may be too high for sparsely populated domains such as attractions resulting in unreliable normalized scores. As can be seen from the standard deviation of the SiteMax normalizer, the spread of the scores is the highest with many entities having a score of '1' as well as '0'. EntityMax uses the maximum score of pages related to one entity and thus the score gets adjusted according to entity popularity. Interestingly, EntityMax+GlobalMax performs slightly better than EntityMax in terms of precision likely because we also use the global maximum which boosts the scores of densely populated review pages and reduces the scores of sparsely populated ones.

Based on Table III, we can also see that the *FixedNorm* normalization strategy yields the highest precision and $F_{0.4}$ scores and is comparable to the scores of *EntityMax+GlobalMax* in terms of recall and accuracy. This shows that the fixed normalization value used in Equation 5 acts as an effective saturating point where pages that exceed the normalizer values are immediately classified as review pages and pages that have a low value are pruned away by the σ_{rev} threshold.

Which $S_{ent}(p_i, e_k)$ scoring method is most effective? To understand which $S_{ent}(p_i, e_k)$ scoring method is most effective, we ran our algorithm at different σ_{ent} thresholds, with the *OpinoFetch_RV_FixedNorm* run. We set $\sigma_{search} = 50$ and $\sigma_{rev} = 0.2$. The results at different σ_{ent} thresholds are reported in Table IV. Notice that the best $F_{0.4}$ score achieved is with the TitleOnly scoring method (at $\sigma_{ent} = 0.7$), followed by MaxURLTitle (at $\sigma_{ent} = 0.7$) and URLOnly (at $\sigma_{ent} = 0.5$). On average however, MaxURLTitle has the highest $F_{0.4}$ scores across σ_{ent} thresholds indicating that in the absence of scores from one component (e.g. empty title or URL), using scores from the backup component is indeed effective. This is further evident from the standard deviation of scores, where the score dispersion with the use of MaxURLTitle is tighter compared to just using TitleOnly or URLOnly. This is because entities that were previously getting a score of '0' are now getting a more appropriate score reducing the dispersion of scores. Overall, PageContents has the worst performance with the highest performance being at $\sigma_{ent} = 0.2$. This makes sense,

	Precision	Recall	$F_{0.4}$ Score	Accuracy	Std. Dev.	Improvement in precison over not pruning
FixedNorm	**0.536**	0.098	**0.332**	0.678	0.232	32.769%
EM+GM	0.532	0.098	0.330	0.677	0.229	31.663%
EM	0.532	0.099	0.331	0.678	0.231	31.742%
SM+GM	0.434	0.123	0.322	0.569	0.233	7.402%
SM	0.398	**0.129**	0.309	0.502	0.304	-1.593%

Table III: Average precision, recall, $F_{0.4}$ score and accuracy across $\sigma_{rev} \in \{0.1, 0.3, 0.5\}$ with different normalizers. EM=EntityMax; SM=SiteMax; GM=GlobalMax, Fixed=Fixed Normalization. Table also shows standard deviation of the $S_{rev}(p_i)$ scores and % improvement in precision over not pruning pages.

	$F_{0.4}$					Std. Dev.
$S_{ent}(p_i, e_k)$ **Scoring Method/**σ_{ent}	**0.2**	**0.5**	**0.7**	**0.9**	**Average**	
TitleOnly	0.3640	0.3729	**0.3890**	0.2837	0.3524	0.3649
URLOnly	0.3678	**0.3783**	0.3772	0.2915	0.3537	0.3891
MaxURLTitle	0.3541	0.3729	**0.3889**	0.3034	**0.3548**	0.3578
PageContents	**0.3511**	0.1028	0.0376	0.0001	0.1229	0.1974

Table IV: $F_{0.4}$ scores and standard deviation of different $S_{ent}(p_i, e_k)$ scoring methods at different levels of σ_{ent}. *OpinoFetch_RV_FixedNorm* was used for $S_{rev}(p_i)$ scoring.

because there can be many misleading anchor texts and words about other entities resembling the entity query, resulting in false positive word matches.

Is one domain harder than another? Although collecting review pages is a fairly generic task for all entities, the difficulty in collecting reviews in one domain can be quite different from another. In our evaluation, we have observed that collecting reviews from the *attractions* domain was most difficult with lowest precision and recall as shown in Figure 3. One reason for this is because we have observed a lot more ambiguity in the attractions domain compared to the electronics or hotels domain. For example, one of our entities in the attractions domain is *Disneyland Park Anaheim California*. Based on our investigation, we noticed several other entities that carry a similar name. Examples are review pages related to *Space Mountain Disneyland Park* (a fun ride) and *Fairfield Inn Anaheim Disneyland Park Resort* (a hotel) all of which could yield false positives. One potential way to improve this is to determine the type of entity the page is related to (e.g.'hotel', 'attraction', 'electronics') and use that in the scoring. This is something we would like to explore as part of our future work.

How many levels to explore? By default, in our evaluation we use $\sigma_{depth} = 1$. Now, we look into how much improvement we see in terms of recall by following links at different σ_{depth} levels. We fix $\sigma_{search} = 50$ and compare gain in terms of recall at different search depths. The results are shown in Figure 4. Notice that we gain the most in terms of recall by just analyzing links within the search results ($\sigma_{depth} = 1$) and as we follow links that are further away from the search results, the gain in recall keeps dropping. While the search results itself may not be direct pointers to review pages, there are actually many relevant review pages that are close to the search results and as these links are discovered, recall significantly improves. On the other hand, as the crawler digs deeper and deeper, the relevance of the links followed to the target entity (i.e. entity query) declines and therefore the gain in recall is also much lower. Thus, the best crawl depth is $\sigma_{depth} = 2$ as crawling further does not improve recall significantly.

Also notice that the attractions domain gains the most in terms of recall at every level. This is because reviews in this domain are sparse and any additional links followed yields more review pages compared to just the search results which had very low precision and recall to start with.

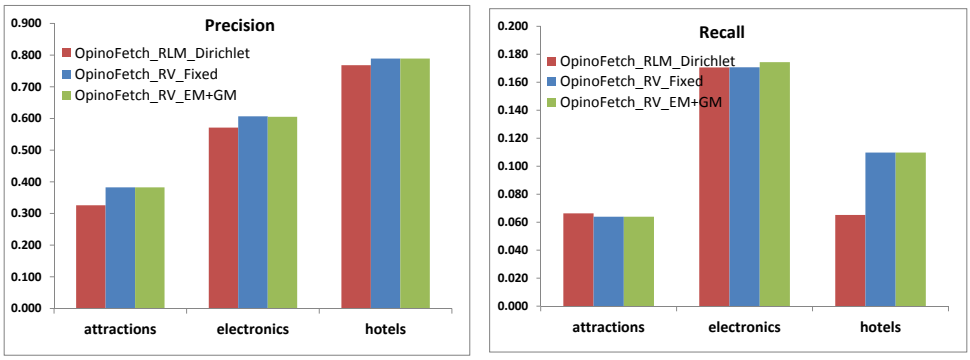

Fig. 3: Precision and recall of different OpinoFetch runs in different domains with $\sigma_{search} = 50$, $\sigma_{depth} = 1$ and $\sigma_{ent} = 0.5$.

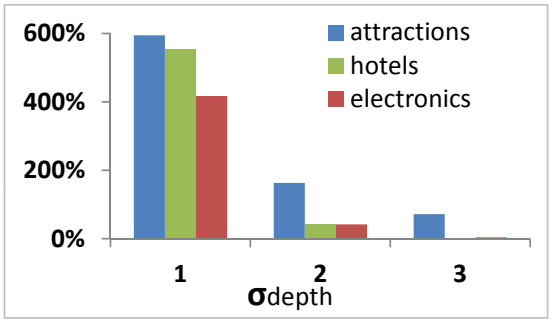

Fig. 4: Gain in recall at different depths using OpinoFetch.

6.1. Site Coverage

One could argue that it is possible to obtain reviews about all entities in a particular domain (e.g. hotels, electronics, etc) just by crawling a few major opinion sites. However, based on our observation, even entities within the same domain can have a very different set of review sources and thus just a handful of opinion sites would not cover all reviews about an entity. We would thus like to show that OpinoFetch can reach out to long-tail reviews that we would not be able to obtain by just crawling a few major opinion sites. We refer to this analysis as site coverage.

For the site coverage analysis, we run OpinoFetch (with a crawl depth of 2) using the top 100 search results from Google for 4 entities within the electronics domain. We then compile a list of sites for URLs deemed relevant by OpinoFetch for each of the 4 entities. In creating the review site list, we eliminate all redundancies and normalize international sites and sub-domains (e.g. asia.cnet.com, reviews.cnet.com and www.cnet.com would be converted into cnet.com). With this, we have a unique list of review sites for each entity. Given this list, we categorize all sites that appear in the top 20 search results of each entity as major opinion sites. Since users typically only look at the first few pages of the search results, Google tends to rank all the sites deemed relevant and important before other 'less important' sites. Thus, this strategy of considering sites that appear in the top 20 search results as the major opinion sites is reasonable. All other sites found using OpinoFetch are then regarded as long-tail sites.

Table V shows the distribution of sites (long-tail vs. major sites) found using OpinoFetch for the 4 entities. From this, we can see that in all cases, more than 50% of the relevant review pages are from long-tail sites. This goes to show that there are a lot of reviews that exist in a variety of

Entity Type	Major Sites	Long Tail Sites	#Relevant Sites
Apple iPhone 64GB 4S	28.26%	71.74%	46
Garmin Nuvi 205	25.00%	75.00%	20
HP Touchpad Tablet 16GB	32.35%	67.65%	34
Nikon D5100	18.84%	81.16%	69

Table V: Distribution of major opinion sites vs. long tail sites. Note that all sites are unique accounting for sub-domain differences, internationalization and any form of redundancies.

Entity Type	Major Sites	Long Tail Sites
Nikon D5100	target.com reviews.bestbuy.com ebay.com costco.com consumerreports.org pcmag.com	cameras.pricedekho.com club.dx.com digital-photography-school.com kenrockwell.com nikondslrtips.com photographylife.com
HP Touchpad 16G Tablet	reviews.officemax.com newegg.com computershopper.com engadget.com expertreviews.co.uk pcworld.co.uk wired.com	webosnation.com anandtech.com pcpro.co.uk pocket-lint.com forum.tabletpcreview.com winnipeg.kijiji.ca tabletconnect.blogspot.com

Table VI: Example of major opinion sites and long tail review sites for Nikon D5100 camera and HP Touchpad 16GB Tablet.

different sources than just the major opinion sites. Also, note that the number of sites containing relevant reviews about the entities are very different even though they all are electronics. Table VI shows example of specific sites for 2 of the 4 entities. An example page from *kenrockwell.com* is http://www.kenrockwell.com/nikon/d5100.htm for the Nikon D5100 camera. Another example is http://www.webosnation.com/review-hp-touchpad from *webosnation.com* for the HP Touchpad 16GB Tablet. Both these sites contain personal reviews on the corresponding products which will be a value add when aggregated with reviews from other sources.

6.2. Failure Analysis

To better understand reasons as to when our system fails (i.e. reasons for false positives and false negatives), we did a closer analysis on our best run ($S_{rev}(p_i)$ scoring using *OpinoFetch_RV_FixedNorm* and $S_{ent}(p_i, e_k)$ scoring using *MaxURLTitle*). We use $\sigma_{rev} = 0.2$ and $\sigma_{ent} = 0.7$. We report details of our analysis in Table VII.

The first point to note based on this table is that we have a higher number of false positives (262) compared to false negatives (138). This shows that the system does a better job at recognizing pages that are truly relevant but has a harder time eliminating irrelevant pages.

False positive analysis. Within the false positives, based on Table VII, notice that the mean $S_{ent}(p_i, e_k)$ score is 0.946 with a standard deviation of 0.102 indicating the system is confident that the pages are entity relevant. Also, if we look at the mean $S_{rev}(p_i)$ score notice that the score of 0.243 is slightly above the set threshold of $\sigma_{rev} = 0.2$. Upon closer look at the false positives, we found that in most cases, pages are considered to be entity relevant when in fact they are not, with a high score of $S_{ent}(p_i, e_k)$. This explains the high mean $S_{ent}(p_i, e_k)$ value for the false positives. As an example, one of our entity queries in our dataset is '*apple iphone **64 gb** 4s product review*'. Some review pages about the *'iPhone 4s **32 gb**'* and *'iPhone 4s **16 gb**'* yield a high $S_{ent}(p_i, e_k)$ score. This is due to the minor differences in the entity type description present either in the title or URL

	$S_{rev}(p_i)$		$S_{ent}(p_i, e_k)$	
	Mean	**Std. Dev.**	**Mean**	**Std. Dev.**
False Positives	0.445	0.196	0.946	0.102
False Negatives	0.243	0.313	0.849	0.233

	# False Negative/False Positive Pages		
	Review Page? $(S_{rev}(p_i) > \sigma_{rev})$	**Entity Relevant?** $(S_{ent}(p_i, e_k) > \sigma_{ent})$	**Total**
False Positives	262	262	262
False Negatives	39	99	138

Table VII: Analysis of best run ($S_{rev}(p_i)$ scored with *OpinoFetch_RV_FixedNorm* and $S_{ent}(p_i, e_k)$ scored with *MaxURLTitle*. Threshold values $\sigma_{rev} = 0.2$ and $\sigma_{ent} = 0.7$. The first table shows the distribution of scores under false positive and false negative categories. The second table shows proportion of false positive pages vs. false negatives.

	Reasons	**# False Negatives with** $S_{ent}(p_i, e_k) < \sigma_{ent})$	**# False Negatives with** $S_{rev}(p_i) < \sigma_{rev})$
Reason1	review in other languages	0	48
Reason2	limited reviews or review terms	0	40
Reason3	crawled content issues	0	1
Reason4	judgment set issues	3	10
Reason5	insufficient info or overload in page title or URL	36	0
Total		**39**	**99**

Table VIII: False Negative Analysis

	Reasons	**# False Positives**
Reason6	Ambiguous entity names	256
Reason7	Looks like a review page	86

Table IX: False Positive Analysis

during TitleOnly and URLOnly scoring. From Table IX, we see that 256 of the review pages were not entity relevant due to ambiguity and only 86 review pages had the issue of resembling a review page (when it actually was not). To deal with the bigger problem of ambiguity in entity names, we can enforce a scoring scheme where certain words in the entity query should be mandatory. Otherwise these pages should receive a $S_{ent}(p_i, e_k)$ score of '0'. We believe that for the problem of pages resembling review pages, we can easily handle the cases with borderline review scores by eliminating pages based on page length (very short pages are less likely to contain textual reviews).

False negative analysis. In the case of false negatives instead, based on Table VII, notice that while most pages are considered entity relevant (94/131) only 35 were considered review pages when in fact the remaining 96 pages are also review pages. This shows that in most of the false negative cases, actual review pages are not having a high enough $S_{rev}(p_i)$ score to qualify them as review pages. This is caused by several reasons. The first reason is that some of these review pages are in other languages other than English. While OpinoFetch_RV_FixedNorm can correctly classify some of the review pages in other languages (especially those that have a mix of English), it is unable to handle non-English pages from specific websites. In addition, review pages that were almost empty (i.e. no reviews or one or two short lines) also had a low $S_{rev}(p_i)$ score resulting in these pages being pruned. Table VIII shows that 88 of the false negatives, were caused either by review being

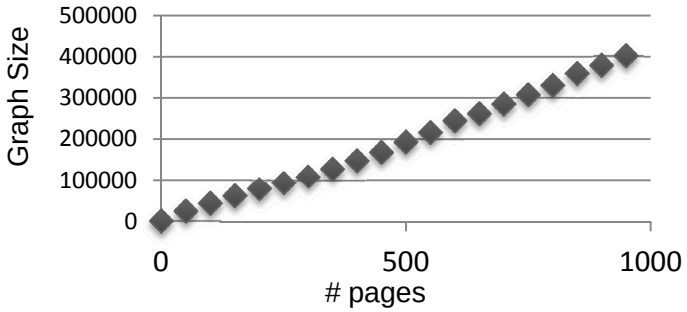

Fig. 5: Growth of FetchGraph with respect to number of pages collected.

	+FetchGraph	**-FetchGraph**
$S_{rev}(p_i)$ **(unnormalized)**	0.085ms	8.62ms
EntityMax Normalizer	0.056ms	4.39s

Table X: Average execution time in computing $S_{rev}(p_i)$ & EntityMax normalizer with and without the FetchGraph.

in other languages or limited reviews within the page. We however believe that this is not a critical miss as even if such a page was not correctly classified, there is limited loss in information.

In summary, the more pressing problem in our method is dealing with the false positive cases where we need to disambiguate entities that are almost similar.

6.3. Time and Memory Analysis

The OpinoFetch framework is developed in Java. For all experiments we use a 2x 6-core @ 2.8GHz machine with 64GB memory.

One key advantage of using the FetchGraph is its ability to keep most information related to the data collection problem encapsulated within a single heterogenous network. This can range from representing information about entities to individual terms within the data collection vocabulary. With this, it is quite possible for the network to grow too large too fast and not fit in memory for processing. In Figure 5, we can see that the FetchGraph's growth is actually linear to number of pages collected and this is without any code related optimization or special filters (which can decrease overall nodes created). If we added 1 million pages to the FetchGraph and assume that each node and edge are represented by objects of size 50 bytes (base object is 8 bytes), the resulting FetchGraph would be approximately 20GB, which is still manageable in memory and would only reduce in size with various optimizations.

Another advantage of using the FetchGraph is efficiency in information access. When we need access to dependency information (e.g. in computing normalizers for $S_{rev}(p_i)$) or repeated access to various statistics (e.g. page related term frequencies), it is not possible to obtain such information easily or efficiently without a proper data structure. Due to the versatility of the FetchGraph, once a page gets added to this information network, it becomes easy to access all sorts of information from the network.

Table X shows execution time of computing the unnormalized $S_{rev}(p_i)$ score and execution time for computing the EntityMax normalizer using the FetchGraph and without it using a combination of lists and in memory tables (averaged across all domains). It is clear that even to compute the unnormalized $S_{rev}(p_i)$ it would be quite expensive to repetitively compute and recompute these scores without any supporting data structure. This becomes worse when we normalize the scores as seen in the time to compute the EntityMax normalizer without the FetchGraph. The execution time utilizing the FetchGraph is notably lower as the page is only loaded into memory once and all other statistics can be obtained by accessing the FetchGraph directly. While it is feasible to use a database

Electronics (3.92 ms)			Attractions (2.60 ms)		
Site	$S_{rev}(p_i)$	**# Ent.**	**Site**	$S_{rev}(p_i)$	**# Ent.**
Amazon	57.90	5/5	Yelp	41.65	4/4
Bestbuy	19.48	5/5	Tripadvisor	32.76	4/4
Ebay	20.71	5/5	Yahoo! Travel	1.92	4/4
Cnet	17.94	4/5	Rvparkreviews	14.21	2/4
Digitaltrends	5.37	2/5	Virtualtourist	6.24	2/4
Techradar	5.83	2/5	Igougo	5.06	2/4

Table XI: Snapshot of results for the query *select PopularSites(10) from FectchGraph(D) order by EntityCount*; *D=Electronics* and *D=Attractions*. $S_{rev}(p_i)$ represents the cumulative $S_{rev}(p_i)$ score for the site.

for some of these tasks, the FetchGraph is an in memory data structure and thus is much faster than accessing the database especially when large joins are expected. Also, since we can separate the data collection problem (e.g. by domain), we only need to load the required networks into memory.

6.4. Sample Query & Results

One of the important uses of the FetchGraph is to answer application related questions. Assuming we have a special query language to query the FetchGraph, one interesting question is: *What are the popular review sites in a given domain?* This query is quite typical for business intelligence applications that perform analysis on subsets of data. Using the FetchGraph this information can be obtained by ranking the sites based on *indegree* information and cummulative per site $S_{rev}(p_i)$ scores which would result in popular and densely populated review sites to emerge at the top.

Table XI shows a snapshot of results requesting top 10 popular sites for the *electronics* and *attractions* domain. In total, there were 77 sites for attractions and 100 for electronics (without any duplicate elimination). First, it is obvious that the list of review websites vary greatly from domain to domain. Then, we also see that not all sites within a given domain contain reviews for all the entities. This is intuitive as some sites may be very specific to a subset of entities (e.g. only cell phones) or some sites may contain incomplete directory listings or product catalogs. The more striking fact is that all this information (including score aggregation and ranking) can be obtained very quickly from the FetchGraph (3.29 ms for electronics and 2.60 ms for attractions).

7. CONCLUSION

In this paper, we proposed a highly practical framework for collecting online opinions namely reviews for *arbitrary entities*. We leverage the capabilities of existing Web search engines and a new general information network called the FetchGraph to *efficiently* discover review pages for arbitrary entities.

Our evaluation in three interesting domains show that we are able to collect *entity specific review pages* with reasonable precision and accuracy without relying on large amounts of training data or sophisticated Named Entity Recognition tools. We also show that our approach performs significantly better than relying on just plain search engine results. Our analysis clearly shows that the FetchGraph supports efficient storage and lookup of complex crawl information and we also demonstrate how the FetchGraph can be queried to answer interesting application questions.

Compared with existing approaches in topical crawling, our approach is *practically oriented*, *domain independent* and is *weakly supervised* and is thus immediately usable in practice. The proposed FetchGraph is also highly flexible and can be extended to different data collection problems such as collecting news articles about specific topics, forum pages about specific questions and etc.

Although our experiments clearly demonstrated the effectiveness of OpinoFetch, our results also reveal that our system struggles in disambiguating entities that are almost identical. In the future, we plan to address this problem by using 'early pruning' rules to eliminate pages that do not satisfy the initial constraints prior to applying the proposed scoring methods. We also intend to introduce an

iterative approach where we use our current approach to obtain training examples to automatically build a mini entity-specific classifier.

Our source code and evaluation set is available at http://kavita-ganesan.com/content/opinofetch.

REFERENCES

Soumen Chakrabarti, Kunal Punera, and Mallela Subramanyam. Accelerated focused crawling through online relevance feedback. In *Proceedings of WWW '02*, 2002.

Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. In *Proceedings of the WWW '99*, 1999.

Hsinchun Chen, Yi-Ming Chung, Marshall C Ramsey, and Christopher C Yang. A smart itsy bitsy spider for the web. *Journal of the American Society for Information Science, Special Issue on AI Techniques for Emerging Information Systems Applications*, 1998.

P. De Bra, G.J. Houben, Y. Kornatzky, and R. Post. Information retrieval in distributed hypertexts. In *Proceedings of the 4th RIAO Conference*, 1994.

Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *Proceedings of the 26th International Conference on VLDB*, VLDB '00, 2000.

Kavita Ganesan, ChengXiang Zhai, and Jiawei Han. Opinosis: A graph based approach to abstractive summarization of highly redundant opinions. In *Proceedings of COLING '10*, Beijing, China, 2010.

Kavita Ganesan, ChengXiang Zhai, and Evelyne Viegas. Micropinion generation: An unsupervised approach to generating ultra-concise summaries of opinions. In *Proceedings of the WWW '12*, 2012.

Shima Gerani, Yashar Mehdad, Giuseppe Carenini, Raymond T. Ng, and Bita Nejat. Abstractive summarization of product reviews using discourse structure. In *Proceedings of the EMNLP '14*.

M. Hersovici, M. Jacovi, Y.S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur. The shark-search algorithm. an application: tailored web site mapping. *Computer Networks and ISDN Systems*, 30, 1998.

Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of KDD '04*, 2004.

Minqing Hu and Bing Liu. Mining opinion features in customer reviews. In *Proceedings of AAAI '04*, 2004.

Judy Johnson, Kostas Tsioutsiouliklis, and C. Lee Giles. In Tom Fawcett and Nina Mishra, editors, *ICML*.

Hyun Duk Kim and ChengXiang Zhai. Generating comparative summaries of contradictory opinions in text. In *Proceedings of the CIKM '09*, 2009.

Yue Lu, ChengXiang Zhai, and Neel Sundaresan. Rated aspect summarization of short comments. In *Proceedings of the 18th international conference on World wide web*, 2009.

A. McCallum, K. Nigam, J. Rennie, and K. Seymore. A machine learning approach to building domain-specific search engines. Proc. AAAI Spring Symposium on Intelligent Agents in Cyberspace, 1999.

B. Novak. A survey of focused web crawling algorithms. SKIDD, 2004.

I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A high performance and scalable information retrieval platform. In *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*, 2006.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? Sentiment classification using machine learning techniques. In *Proceedings of EMNLP '02*, 2002.

Raimundo Real and Juan M. Vargas. The Probabilistic Basis of Jaccard's Index of Similarity. *Systematic Biology*, 45, 1996.

Amit Singhal. Modern information retrieval: A brief overview. 2001.

Benjamin Snyder and Regina Barzilay. Multiple aspect ranking using the good grief algorithm. In *Proceedings of HLT-NAACL '07*, pages 300–307, 2007.

A. Gural Vural, B. Barla Cambazoglu, and Pinar Senkul. Sentiment-focused web crawling. In *Proceedings of the CIKM '12*, 2012.

ChengXiang Zhai. *Statistical Language Models for Information Retrieval*.

View publication stats