Topical Crawling for Business Intelligence

Gautam Pant and Filippo Menczer**

Department of Management Sciences
The University of Iowa, Iowa City IA 52242, USA
email: {gautam-pant,filippo-menczer}@uiowa.edu

Abstract. The Web provides us with a vast resource for business intelligence. However, the large size of the Web and its dynamic nature make the task of foraging appropriate information challenging. General-purpose search engines and business portals may be used to gather some basic intelligence. Topical crawlers, driven by richer contexts, can then leverage on the basic intelligence to facilitate in-depth and up-to-date research. In this paper we investigate the use of topical crawlers in creating a small document collection that helps locate relevant business entities. The problem of locating business entities is encountered when an organization looks for competitors, partners or acquisitions. We formalize the problem, create a test bed, introduce metrics to measure the performance of crawlers, and compare the results of four different crawlers. Our results underscore the importance of identifying good hubs and exploiting link contexts based on tag trees for accelerating the crawl and improving the overall results.

1 Introduction

A large number of business entities — start-up companies and established corporations — have a Web presence. This makes the Web a lucrative source for locating business information of interest. A company that is planning to diversify or invest in a start-up would want to locate a number of players in the area of business. The intelligence gathering process may involve manual efforts using search engines, business portals or personal contacts. Topical crawlers can help in extracting a small but focused document collection from the Web that can then be thoroughly mined for appropriate information using off-the-shelf text mining, indexing and ranking tools.

Topical crawlers, also called focused crawlers, have been studied extensively in the past [6, 7, 3, 10, 2]. In our previous evaluation studies of topical crawlers, we found a similarity based Naive Best-First crawler to be quite effective [11, 12, 18]. However, the Naive Best-First crawler made no use of the inherent structure available in an HTML document. We also studied algorithms that attempt to identify the context of a link using a sliding window and a distance measure based on number of links separating a word from a given link. However, such approaches

** Current affiliation: School of Informatics and Computer Science Department, Indiana University. Email: fil@indiana.edu

often performed no better than variants of the Naive Best-First approach [12, 18]. In this paper we consider a crawler that identifies the context of a link using the HTML page's tag tree or Document Object Model (DOM) representation.

The problem of locating business entities on the Web maps onto the general problem of Web resource discovery. However, it has some features that distinguish it from the general case. In particular, business communities are highly competitive. Hence, it is unlikely that a company's Web page would link to a Web page of its competitor. A topical crawler that is aware of such domain level characteristic may utilize it to its advantage.

We evaluate our crawlers over millions of pages across 159 topics. Based on the number of topics and the number of pages crawled per topic, our evaluations are the most extensive in the currently available topical crawling literature.

2 The Problem

Searching for URLs of related business entities is a type of business intelligence problem. The entities could be related through the area of competence, research thrust, comparable nature (like start-ups) or a combination of such features. We start by assuming that a short list of URLs of related business entities is already available. However, the list needs to be further expanded. The short list may have been generated manually with the help of search engines, business portals or Web directories. An analyst may face some hurdles in expanding the list of relevant URLs. Such hurdles could be due to lack of appropriate content in relevant pages, inadequate user queries, staleness of search engines' collections, or bias in search engines' ranking. Similar problems plague information discovery using Web directories or portals. The staleness of a search engine's collection is highlighted by the dynamic nature of the Web [9]. Cho and Garcia-Molina [4] have shown, in a study with 720,000 Web pages and spanning over 4 months, that 40% of the pages in the `.com` domain changed every day. Hence, it is reasonable to complement traditional techniques with topical crawlers to discover up-to-date business information.

Since our focus is on studying the effect of different crawling techniques, we do not investigate the issue of ranking. We believe that ranking is a separate task best left until a collection has been created. In fact, many different indexing, ranking or text mining tools may be applied to retrieve information from the collection. Our goal is to find ways of crawling and building a small but effective collection for the purpose of finding related business entities. We measure the quality of the collection at various points during the crawl through precision-like and recall-like metrics that are described next.

3 The Test Bed

For our test bed, we need a number of *topics* and corresponding lists of related business entities. The hierarchical categories of the Open Directory Project

(ODP)[1] are used for this purpose. ODP provides us with a categorical collection of URLs that are manually edited and not biased by commercial considerations. Hence, we can find categories that list related business entities as judged by humans. To begin with, we identify categories in the ODP hierarchy that end with any of the following words: "Companies," "Consultants," or "Manufacturers." These categories serve as topics for our test bed. We collect only those categories that have more than 20 unique external URLs and we skip categories that have "Regional," "World" and "International" as a top-level or sub-category. After going through the entire RDF dump of ODP, 159 such categories are found. Next, we split each category's external URLs into two disjoint sets — *seeds* and *targets*. The set of seeds is created by picking 10 external URLs at random from a category. The remaining URLs make up the targets. The seeds are given to the crawlers as starting links while the targets are used for evaluation. The *keywords* that guide the crawlers are created by concatenating all the alphanumeric tokens in the ODP category name leaving out the most general category (Science, Arts etc.). We also concatenate the manual descriptions of the seeds and the targets and create a master *description* that would serve as another source for evaluations. An example of a topic with corresponding keywords, description, seeds and targets as extracted from an ODP category is shown in Table 1.

Each crawler is provided with a set of keywords and the corresponding seeds to start the crawl. A crawler is then allowed to crawl up to 10,000 pages starting from the seeds. The process is repeated for each of the 159 topics in the test bed. As a result, we may crawl more than one and a half million pages for one crawler alone. We use the following precision-like and recall-like measures to understand the performance of the crawlers:

– **Precision@N**: We measure the precision for a crawler on topic t after crawling N pages as:

$$precision@N_t = \frac{1}{N} \cdot \sum_{i=1}^{N} sim(d_t, p_i) \tag{1}$$

where d_t is the description of the topic t, p_i is a crawled page, and $sim(d_t, p_i)$ is the cosine similarity between the two. The cosine similarity is measured using a TF-IDF weighting scheme [17]. In particular, the weight of a term k in a given page p is computed as:

$$w_{kp} = \left(0.5 + \frac{0.5 \cdot tf_{kp}}{\max_{k' \in T_p} tf_{k'p}}\right) \cdot \ln\left(\frac{|C_t^*|}{df_k}\right) \tag{2}$$

where tf_{kp} is the frequency of the term k in page p, T_p is the set of all terms in page p, C_t^* is the set of pages crawled by all the crawlers for topic t at the end of the crawl, and df_k is the number of pages in C_t^* that contain the term k. After representing a topic description (d) and a crawled page (p)

[1] http://dmoz.org

as a vector of term weights ($\boldsymbol{v_d}$ and $\boldsymbol{v_p}$ respectively), the cosine similarity between them is computed as:

$$sim(d,p) = \frac{\boldsymbol{v_d} \cdot \boldsymbol{v_p}}{\| \boldsymbol{v_d} \| \cdot \| \boldsymbol{v_p} \|} \tag{3}$$

where $\boldsymbol{v_d} \cdot \boldsymbol{v_p}$ is the dot (inner) product of the two vectors. Note that the terms used for cosine similarity, throughout this paper, undergo stemming (using Porter stemming algorithm [15]) and stoplisting.
For a given N, we can average $precision@N_t$ over all the topics in the test bed to get the *average precision*@N. The latter can be plotted as a trajectory over time, where time is approximated by increasing values of crawled pages (N).

– **Target recall@N**: In the absence of a known relevant set, we treat the recall of the targets as an indirect indicator of actual recall (Figure 1). The $target\ recall@N_t$ for a crawler on a topic t can be computed at various points (N) during the crawl:

$$target\ recall@N_t = \frac{| T_t \cap C_t^N |}{| T_t |} \tag{4}$$

where T_t is the set of targets for topic t, and C_t^N is the set of N crawled pages. Again, we can average the recall (*average target recall*@N) at various values of N over the topics in the test bed.

The above evaluation metrics are a special case of a general evaluation methodology for topical crawlers [18]. We will perform one-tailed t-tests to understand, in statistical terms, the benefit of using one crawler over another. The null hypothesis in all such tests will be that the two crawlers under consideration perform equally well.

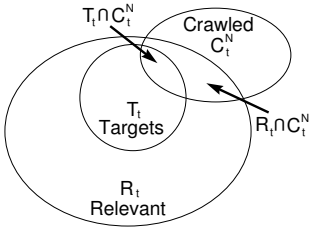

Fig. 1. $| T_t \cap C_t^N | / | T_t |$ as an estimate of $| R_t \cap C_t^N | / | R_t |$

4 Crawlers

We evaluate the performance of four different crawlers. Before we describe the crawling algorithms, it is worthwhile to illustrate the general crawling infrastructure that all the crawlers use.

Table 1. A sample topic - keywords, description, seeds, targets

keywords	*description*	*seeds*	*targets*

Gambling Equipment Manufacturers	Gemaco Cards Manufacturer of promotional and casino playing cards in poker, bridge ... R.T. Plastics Supplier of casino value, roulette, poker and tournament chips ...	`http://www.gemacocards.com/`
		`http://www.rtplastics.com/`
		`http://www.ally.com/`
		`http://www.amtote.com/`
		`http://www.pokerchips.com/`
		...
			`http://www.agtco.com/`
			`http://www.cartamundi.com/`
			`http://www.barcrest.co.uk/`
			`http://www.ilts.com/`
			`http://www.ballygaming.com/`
			`http://www.tcsgroup.com/`
			`http://pidcgroup.com/`
			...

4.1 Crawling Infrastructure

The crawlers are implemented as multi-threaded objects in Java. Each crawler has many (possibly hundreds) threads of execution sharing a single synchronized *frontier* that lists the unvisited URLs. Each thread of the crawler follows a *crawling loop* that involves picking the next URL to crawl from the frontier, fetching the page corresponding to the URL through HTTP, parsing the retrieved page, and finally adding the unvisited URLs to the frontier. Before the URLs are added to the frontier they may be assigned a score that represents the estimated benefit of visiting the page corresponding to the URL. Some of the crawlers utilize the tag tree structure of the Web pages. They first *tidy*[2] the HTML page and then use an XML parser to obtain relevant information from the DOM structure. Tidying an HTML page includes both insertion of missing tags and reordering of tags in the page. The process is necessary for mapping the content of a page onto a tree structure with integrity, where each node has a single parent. We also enclose all the text tokens within `<text>...</text>` tags. This makes sure that all text tokens appear as leaves on the tag tree. While this step is not necessary for mapping an HTML document to a tree structure, it does provide some simplifications for the analysis. Figure 2 shows a snippet of an HTML page that is subsequently cleaned and converted into convenient XML format before being represented as a tag tree.

The current crawler implementation uses 75 threads of execution and limits the maximum size of the frontier to 70,000. Only the first 10KB of a Web page are downloaded. Note that during a crawl of 10,000 pages, a crawler may encounter more than 70,000 unvisited URLs. However, given that the average number of outlinks on Web pages is 7 [16], the maximum frontier size is not very restrictive for a crawl of 10,000 pages. Being a shared resource for all the threads, the frontier is also responsible for enforcing certain ethics that prevent the threads from accessing the same server too frequently. In particular, the frontier tries to enforce the constraint that every batch of D URLs picked from it are from D different server host names ($D = 50$). The crawlers also respect the Robot Exclusion Protocol.[3]

[2] http://www.w3.org/People/Raggett/tidy/
[3] http://www.robotstxt.org/wc/norobots.html

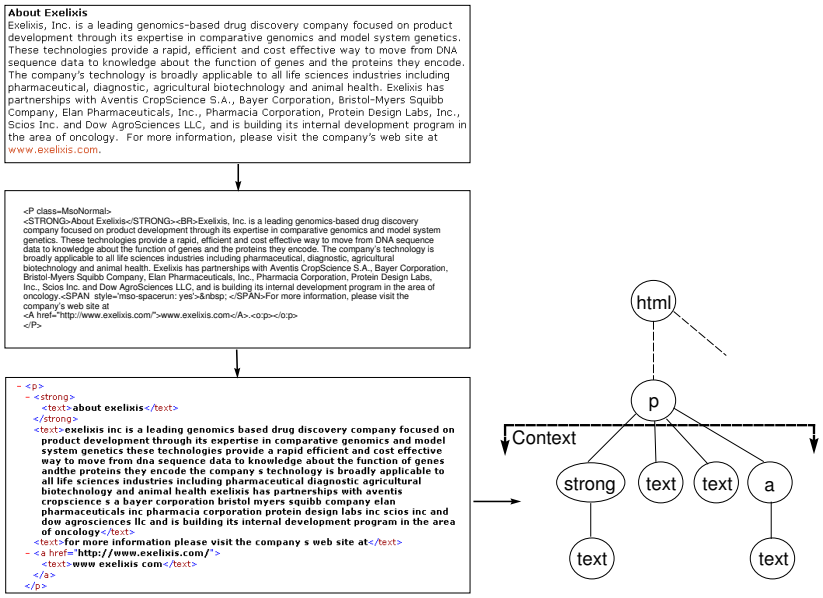

Fig. 2. An HTML snippet (top) whose source (center) is modified into a convenient XML format (bottom). The tag tree representation of the HTML snippet is shown on the right.

4.2 Breadth-First Crawler

Breadth-First is a baseline crawler for our experiments. The crawl frontier is a FIFO queue. Each thread of the crawler picks up the URL at the front of the queue and adds new unvisited URLs to the back of it. Since the crawler is multi-threaded, many pages are fetched simultaneously. Hence, it is possible that the pages are not fetched in an exact FIFO order. In addition the ethics enforced in the system, that prevent inundating a Web server with requests, also make the crawler deviate from strict FIFO order. The crawler adds unvisited URLs to the frontier only when the size of the frontier is less than the maximum allowed.

4.3 Naive Best-First Crawler

The Naive Best-First crawler treats a Web page as a bag of words. It computes the cosine similarity of the page to the given keywords and uses it as a score ($link\_score_{\text{bfs}}$) of the unvisited URLs on the page. The URLs are then added to the frontier that is maintained as a priority queue using the scores. Each thread picks the best URL in the frontier to crawl, and inserts the unvisited URLs at appropriate positions in the priority queue. The $link\_score_{\text{bfs}}$ is computed using Equation 3. However, the keywords replace the topic description, and the vector representation is based only on term frequencies. A TF-IDF weighting scheme is

problematic during the crawl because there is no a priori knowledge about the distribution of terms across crawled pages.

As in the case of the Breadth-First crawler, a multi-threaded version of Naive Best-First crawler with ethical behavior does not crawl in an exact best first order. Due to multiple threads it acts like a Best-N-First crawler where N is related to the number of simultaneously running threads. Best-N-First is a generalized version of the Naive Best-First crawler that picks N best URLs to crawl at a time. We have found certain versions of the Best-N-First crawler to be strong competitors in our previous evaluations [14, 12, 18]. Note that the Naive Best-First crawler and the crawlers to follow keep the frontier size within its upper bound by retaining only the best URLs based on the assigned scores.

4.4 DOM Crawler

Unlike the Naive Best-First crawler, the DOM crawler tries to make use of the structure that is available in an HTML page through its tag tree representation. Figure 2 shows the tag tree representation of an HTML snippet. In its general form, the DOM crawler treats some node that appears on the path from the root of the tag tree to a link as an *aggregation node* of the link. All the text in the sub-tree rooted at the aggregation node is then considered a context of the link. In the current implementation of the DOM crawler we have set the immediate parent of a link to be its aggregation node. While the choice of the parent as the aggregation node may seem arbitrary, we note that it is no more arbitrary than picking up W words or bytes around a hyperlink and treating them as the link's context as has been done before [7, 1]. In an ideal situation we would like to identify the optimal aggregation node for a given page and topic. This problem is being studied [13] but it is beyond the scope of this paper. In our example from Figure 2, all the text (at any depth) under the <p> paragraph tag forms a context for the link `http://www.exelixis.com`. After associating a link with its context, the crawler computes the cosine similarity between the context and the given keywords. The measure is called the *context_score*. In addition, the crawler computes the cosine similarity between the page in which the link was found and the keywords, which corresponds to $link\_score_{\text{bfs}}$. The final $link\_score_{\text{dom}}$ associated with a link is computed by:

$$link\_score_{\text{dom}} = \alpha \cdot link\_score_{\text{bfs}} + (1 - \alpha) \cdot context\_score$$

where α weighs the relative importance of the entire page content vs. the context of a link. It is set to 0.25 to give more importance to the *context_score*. The frontier is a priority queue based on $link\_score_{\text{dom}}$. In practice, due to the reasons mentioned for the other crawlers, the individual pages are not fetched in the exact order of priority.

Note that the Naive Best-First crawler can be seen as a special case of the DOM crawler in which the aggregation node is set to the root of the DOM tree (the <html> tag), or where $\alpha = 1$.

4.5 Hub-Seeking Crawler

The Hub-Seeking crawler is an extension of the DOM crawler that tries to explore potential hubs during its crawl. Since the seed URLs are assumed relevant to the topic, the crawler determines that a page that links to many of the *seed hosts* is a good hub. Seed hosts are fully qualified host names for Web servers, such as `www.igt.com`, of the seed URLs. A page points to `www.igt.com` if it points to any page from the host. The crawler assigns a *hub_score* to each page based on the number of seed hosts it points to. It then combines the *hub_score* with the $link\_score_{\text{dom}}$ to get a $link\_score_{\text{hub}}$ for each link on a given page. We would like *hub_score* to have the following properties:

- It should be a non-negative increasing function for all values of $n \geq 0$, where n is the number of seed hosts.
- It should have extremely small or zero value for $n = 0, 1$ (to avoid false positives). The score for $n = 2$ should be relatively high as compared to the average $link\_score_{\text{dom}}$. The event that a page points to two different business entities in the same area is relatively unlikely and hence must be fully exploited.
- The scores must lie between 0 and 1 so that they can be compared against $link\_score_{\text{dom}}$.

While there is an infinite number of functions that satisfy the above properties, we use the one described in Figure 3.

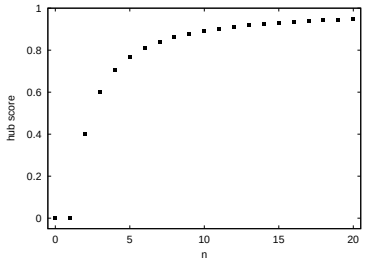

Fig. 3. Hub scores based on $hub\_score(n) = \frac{n \cdot (n-1)}{1+n^2}, n = 0, 1, 2, \ldots$.

Note that the Equation in Figure 3 relies on the fact that the seed URLs are relevant. It is *ad hoc* and lacks the sophistication of methods such as Kleinberg's algorithm [8] to identify hubs and authorities. However, it is easy to compute in real time during the crawl using the link information from a single page. The $link\_score_{\text{hub}}$ that is used to prioritize the frontier queue is calculated as:

$$link\_score_{\text{hub}} = max(hub\_score, link\_score_{\text{dom}}). \tag{5}$$

5 Performance

Figures 4 (a) & (b) show the performance of all the crawlers described above on the test bed. We note that on average the Hub-Seeking crawler performs better than the DOM Crawler which in turn performs better than the Best-First crawler for most part of the 10,000 page crawls. The baseline Breadth-First crawler performs the worst as expected. The above observation is true for both of the performance metrics described in section 3. We note that the performance of the Hub-Seeking crawler is significantly better than the Naive Best-First crawler on *average target recall*@10000 (Figure 4 (b)). In fact a one tailed t-test to check the same generates a p-value of 0.004. While the Hub-Seeking crawler outperforms the Best-First crawler even on the *average precision*@10000 metric (Figure 4 (a)), a corresponding one tailed t-test gives a p-value of 0.105. Hence, the benefit is not as significant for precision. However, this means that while building a collection that is at least as topically focused as the one built by the Naive Best-First crawler, the Hub-Seeking crawler will harvest a greater number of the relevant pages of other business entities in the area.

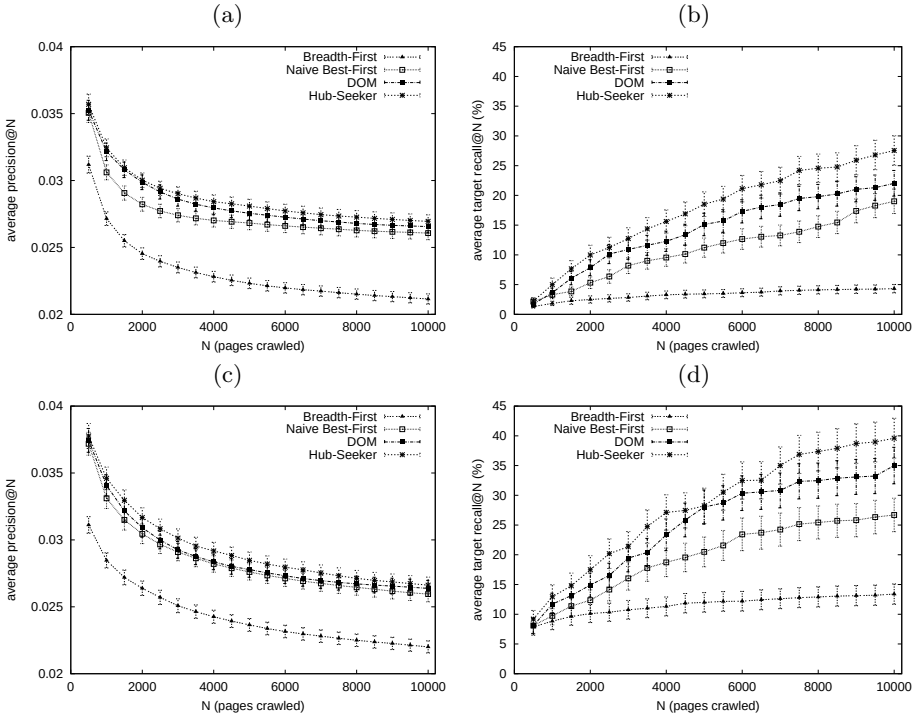

Fig. 4. Performance of the crawlers: (a) *average precision*@N and (b) *average target recall*@N using the ODP links as seeds; (c) *average precision*@N and (d) *average target recall*@N using the augmented seeds. The error bars correspond to ± 1 standard error.

6 Improving the Seed Set

A search engine can assist a topical crawler by sharing the more global Web information available to it. We use the Google API[4] to automatically find more effective seeds for the crawler. The Hub-Seeking crawler assumes that a page that points to many of the seed hosts is a good hub. We use a similar idea to find good hubs from a search engine in order to better seed the crawl. Using the Google API, we find the top 10 back-links of all the test bed seeds for a topic. We then count the number of times a back-link repeats across different seeds. The count is used as a *hub_score*. The process is repeated for all the 159 topics in the test bed. Only the top 10 back-links in accordance with the *hub_score* are kept for each topic. If the count for a back-link is less than 2 then the link is filtered out. Also, to make sure that the problem is non-trivial, we must avoid hubs that are duplicates of the ODP page that lists the seeds and the targets. For this purpose, we do not include any back-link that is from the domain `dmoz.org` or `directory.google.com` (a popular directory based on ODP data). Furthermore, to screen unknown ODP mirrors, any back-link page that has higher than 90% cosine similarity to the the topic description is filtered out. A back-link page is first projected into the space of terms that appear in the topic description, and then the cosine similarity is measured in a manner similar to the Naive Best-First crawler. The projection is necessary to ignore extra text (in addition to the ODP data) that may have been added by a site. Due to the filtering criterion and limited results through the Google API, it is possible that we do not find even a single hub for certain topics. In fact, out of the 159 topics, we find hubs for only 94 topics. We use the hubs thus obtained along with the original test bed seeds to form an *augmented seed* set. The augmented seeds are used to start the crawl for each of the 94 topics.

Figures 4 (c) & (d) show the performance of the four crawlers using the augmented seed set. We first notice that the order of performance (on both metrics) among the crawlers remains the same as that for the original seeds. However, the performance of the DOM crawler in addition to the Hub-Seeking crawler is significantly better than the Naive Best-First crawler on $average\,target\,recall$@10000 (Figure 4 (d)). One tailed t-tests to verify the same yield p-value of 0.024 for the DOM crawler and 0.002 for the Hub-Seeking crawler. Another important observation to make is that all the crawlers better their performance as compared to the experiments using the original seeds for most part of the 10000 page crawls (cf. Figure 4 (a),(b) and Figure 4 (c),(d)). However, $average\,precision@N$ plots (with augmented seeds) show steeper downhill slopes leading to slightly poorer performance towards the end of the crawls for all but the Breadth-First crawler (Figure 4 (c)). The benefit of augmented seeds is more prominent when we look at $average\ target\ recall@N$ plots (cf. Figure 4 (b) and (d)). We perform one-tailed t-tests to check the hypothesis that the availability of augmented seeds (that include hubs) significantly improves the performance of the crawlers based on $average\ target\ recall$@10000. We find that for all the crawlers the null hy-

[4] http://www.google.com/apis/

pothesis (no difference with augmented seeds) can be rejected in favor of the alternative hypothesis (performance improves with augmented seeds) at $\alpha = 0.015$ (in most cases even lower) significance level. We conclude that there is strong evidence that the availability of "good" hubs during the crawl does improve the performance of the crawlers. By providing the hubs, the search engine assists the crawlers in making a good start that affects their overall performance.

7 Related Work

Various facets of Web crawlers have been studied in depth for nearly a decade (e.g.,[6, 7, 3, 10]). Chakrabarti *et al.* [3] used a distiller within their crawler that identified good hubs using a modified version of Kleinberg's algorithm [8]. As noted earlier we use a much simpler approach to identify potential hubs. Moreover, the authors did not show any strong evidence of the benefit of using hubs during the crawl. In a more recent work Chakrabarti *et al.* [2] used the DOM structure of a Web page in a focused crawler. However, their use of DOM trees is different from the present idea of using the aggregation node to associate a link to a context. In particular, their method uses the DOM tree in a manner similar to the idea of using text tokens in the "vicinity" of a link to derive the context. In contrast, the aggregation node explicitly captures the tag tree hierarchy by grouping text tokens based on a common ancestor.

Measuring the performance of a crawler is a challenging problem due to the non-availability of relevant sets on the Web. Nevertheless, researchers use various metrics to understand the performance of topical crawlers (e.g.,[5, 3, 11]). A study by Menczer *et al.* [11] on the evaluation of topical crawlers looks at a number of ways to compare different crawlers. A more general framework to evaluate topical crawlers is presented by Srinivasan *et al.* [18].

8 Conclusions

We investigated the problem of creating a small but effective collection of Web documents for the purpose of locating related business entities by evaluating four different crawlers. We found that the Hub-Seeking crawler that identifies potential hubs and exploits the intra-document structure significantly outperforms the Naive Best-First crawler based on estimated recall. The Hub-Seeking crawler also maintains a better estimated precision than the Naive Best-First crawler. Since Web pages of similar business entities are expected to form a competitive Web community, recognizing neutral hubs that link to many of the competing entities is important. We do see a positive effect in performance through identification of hubs both at the start of the crawl and during the crawl process.

In the future we would like to explore the performance of the Hub-Seeking crawler on more general crawling problems. As noted earlier, the issue of finding the optimal aggregation node is being investigated [13].

Acknowledgments

Thanks to Robin McEntire, Valdis A. Dzelzkalns, and Paul Stead at GlaxoSmithKline for their valuable suggestions. This work has been supported in part through a summer internship by GlaxoSmithKline R&D to GP, and by the NSF under CAREER grant No. IIS-0133124 to FM.

References

1. S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource list compilation by analyzing hyperlink structure and associated text. In *WWW7*, 1998.
2. S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *WWW2002*, Hawaii, May 2002.
3. S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific Web resource discovery. In *WWW8*, May 1999.
4. J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *VLDB 2000*, Cairo, Egypt.
5. J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks*, 30(1–7):161–172, 1998.
6. P. M. E. De Bra and R. D. J. Post. Information retrieval in the World Wide Web: Making client-based searching feasible. In *Proc. 1st International World Wide Web Conference*, 1994.
7. M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur. The shark-search algorithm — An application: Tailored Web site mapping. In *WWW7*, 1998.
8. J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
9. S. Lawrence and C.L. Giles. Accessibility of information on the Web. *Nature*, 400:107–109, 1999.
10. F. Menczer and R. K. Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the Web. *Machine Learning*, 39(2–3):203–242, 2000.
11. F. Menczer, G. Pant, M. Ruiz, and P. Srinivasan. Evaluating topic-driven Web crawlers. In *Proc. 24th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2001.
12. F. Menczer, G. Pant, and P. Srinivasan. Topical web crawlers: Evaluating adaptive algorithms. *To appear in ACM Trans. on Internet Technologies*, 2003. http://dollar.biz.uiowa.edu/~fil/Papers/TOIT.pdf.
13. G. Pant. Deriving Link-context from HTML Tag Tree. In *8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2003.
14. G. Pant, P. Srinivasan, and F. Menczer. Exploration versus exploitation in topic driven crawlers. In *WWW02 Workshop on Web Dynamics*, 2002.
15. M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
16. S. RaviKumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the Web graph. In *FOCS*, pages 57–65, Nov. 2000.
17. G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.
18. P. Srinivasan, F. Menczer, and G. Pant. A general evaluation framework for topical crawlers. *Information Retrieval*, Submitted, 2003. http://dollar.biz.uiowa.edu/~fil/Papers/crawl_framework.pdf.