Topic-Sensitive Hidden-Web Crawling[*]

Panagiotis Liakos and Alexandros Ntoulas

National and Kapodistrian University of Athens
liakospanagiotis@gmail.com, antoulas@di.uoa.gr

Abstract. A constantly growing amount of high-quality information is stored in pages coming from the Hidden Web. Such pages are accessible only through a query interface that a Hidden-Web site provides and may span a variety of topics. In order to provide centralized access to the Hidden Web, previous works have focused on query generation techniques that aim at downloading all content of a given Hidden Web site with the minimum cost. In certain settings however, we are interested in downloading only a specific part of such a site. For example, in a news database, a user may be interested in retrieving only sports articles but no politics. In this case, we need to make the best use of our resources in downloading only the portion of the Hidden Web site that we are interested in. In this paper, we study how we can build a topically-focused Hidden Web crawler that can autonomously extract topic-specific pages from the Hidden Web by searching only the subset that is related to the corresponding category. To this end, we present query generation techniques that take into account the topic that we are interested in. We propose a number of different crawling policies and we experimentally evaluate them with data from two popular sites.

Keywords: Crawling, Hidden Web, Focused Crawling

1 Introduction

An ever-increasing amount of high-quality information on the Web today is accessible through Web pages pulling information from data sources such as databases or content management systems. Such pages are collectively called the Hidden Web because they are hidden from the search engine crawlers but are typically only accessible after issuing one or more queries to a search interface.

In most cases, the information existent in the Hidden Web is of high quality as it has been carefully reviewed, edited or annotated before being stored in a database or a content management system and may span a multitude of topics ranging from sports and politics to different medical treatments of a particular disease.

In order to facilitate the discovery of information on the Web, search engines and content-aggregation systems could greatly benefit from approaches that would allow them to collect and download the Hidden Web pages. Having information from the Hidden Web all in one place can be of great benefit to both users (as they can have a one-stop shop for their information needs) and for the search engines (as they can serve their users better).

[*] Partially supported by PIRG06-GA-2009-256603.

Since a Hidden Web site can be accessed by search engine crawlers only through a search interface, previous work has investigated ways of collecting the information from the Hidden Web sites. In most cases [17, 18, 5, 6, 11, 21, 22] previous work has focused on generating queries that are able to download *all* of (or as much as possible) a given Hidden Web site, with the minimum amount of resources spent (e.g. queries issued). For example, the technique in [17] iteratively issues queries to Hidden Web sites and can download about 90% of some sites using about 100 queries.

Although such approaches can work well for the cases where we are interested in doing a comprehensive crawl of a Hidden Web site, there are cases where we may be interested only in a specific portion of a Hidden Web site. For example, a search engine that specializes in traveling may be interested in picking only news articles that pertain to traveling from a general-purpose news database. A portal talking about politics may want to identify the politics-related articles from a blogs database and leave out the sports-related ones. Or, a mobile application focusing on night-life in San Francisco may want to pull only the related articles from all the postings on events in the wider Northern California.

In this paper, we study the problem of building a topic-sensitive Hidden Web crawler, that can automatically retrieve pages relevant to a particular topic from a given Hidden Web site. One way to achieve this goal would be to employ previously-developed techniques (e.g. [17, 18, 5]) to retrieve the majority of the Hidden Web site and then only keep the content that we are interested in. Since this approach may lead to downloading a number of pages that we are ultimately not interested in, it may also lead to a great waste in resources, measured in time, money, bandwidth or even battery life in the case of a mobile setting.

To this end, the goal of our crawler is to retrieve from a Hidden Web site as many pages related to a given topic as possible with the minimum amount of resources. Our main idea is to proceed in an iterative fashion issuing queries to the Hidden Web site that are very relevant to the topic that we are interested in.

In summary, this paper makes the following contributions:

– We formalize the problem of topically-sensitive Hidden Web crawling, i.e. downloading the pages from a Hidden Web site that pertain only to a given topic.
– We present an algorithm for performing topically-sensitive Hidden Web crawling. Our idea is to identify candidate keywords from the crawled documents that are relevant to the topic of interest.
– We propose a number of different policies that can be used to decide which of the candidate queries we can issue next. As we will show later in our experimental section some polices are much better in producing good keywords than others.
– We evaluate our algorithm using the different policies on two real *Hidden Web* sites and we showcase the merits of each of the policies.

2 Topic-Sensitive Hidden Web Crawling

At a high level, the goal of a general topic-sensitive (or focused) Web Crawler [9] is to retrieve pages from the Web that pertain to a given topic. The crawler has to evaluate the contents of each downloaded page to decide whether it is relevant to a particular topic.

Hence, the crawler aims at examining as small of a subset of the total search space as possible in order to avoid wasting resources by downloading pages that are irrelevant to the topic.

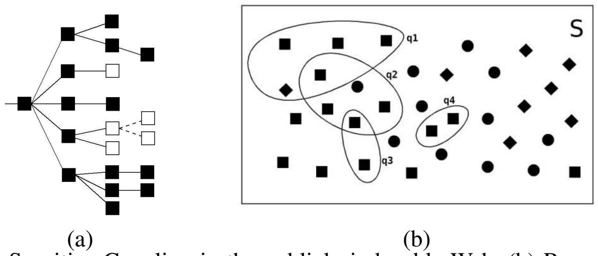


Fig. 1. (a) Topic-Sensitive Crawling in the publicly indexable Web, (b) Representing a Hidden Web site as a set

Figure 1a illustrates this process in the publicly indexable Web. Pages that are relevant to our topic of interest are represented with black boxes while the irrelevant ones are represented with white ones. The goal of the crawler is to follow only links with relevant pages, in order to use its resources efficiently. In the case of a Hidden-Web crawler however the pages are not directly accessible by following links as in [9]. Since the only access point that the crawler has is a search interface, it needs to identify good keywords that will return pages pertaining to a given topic.

The definition of what constitutes a "topic" may be different depending on the application at hand. For example, if we are interested in loosely collecting Web pages that are about football, then any page that contains the words "football", "quarterback", "touchdown", etc., may be relevant to the topic. Alternatively, in the cases when we are interested in discerning more accurately between topics (e.g. external affairs vs. internal politics) a more elaborate mechanism (e.g. an SVM-based classifier) may be used. Our work presented here does not depend on how exactly we detect whether a particular page belongs to a given topic. Instead, we make two generic assumptions regarding the topic definition that allows for a variety of different topic detection techniques to be employed. First, we generally assume that a topic can be described by a distribution of keywords. This is essentially the same assumption as most text classifiers are based on. Second, we assume that not all of the keywords in the distribution are known beforehand, as in this case the problem becomes trivial (i.e. the crawler simply needs to iterate over all the relevant keywords). We should note that not knowing all of the keywords is the typical practical scenario. In our football example above, we may know that "quarterback" or "touchdown" are good initial keywords but we may not necessarily know that "huddle" or "hailmary" are also good keywords for the crawler to use.[1] In our work, we typically provide the crawler with a small set of keywords that are relevant to begin with and the crawler discovers more keywords to use as it progresses through the Hidden-Web site.

[1] This scenario is also typical for the cases where one may want to perform a topic-sensitive crawling-by-example, i.e. provide an example document as input and have the crawler crawl documents that are similar to the input document.

At a high level, in order to access the pages from a Hidden Web site we typically need to apply the following steps:

1. First, submit a query through the interface provided by the Web Site, that characterizes what we want to locate.
2. Then we receive a result index page, that contains links and possibly additional information of the pages that matched the submitted query.
3. Finally, having identified a promising page in the results (we can skip pages already downloaded based on their URL), we follow the link and eventually visit the actual Web page.

2.1 A Topic-Sensitive Hidden Web Crawling Approach

As we discussed in the previous section, we gain access to the contents of a Hidden Web site only by issuing queries through a search interface and obtaining a result index page. Therefore, a Hidden Web crawler should automatically issue queries to the site and then retrieve the pages included in the results.

One of the biggest challenges in implementing such a crawler is that we need to pick the most appropriate terms that will retrieve the pages pertaining to the desired topic in the most effective way. If the crawler can pick terms that are very well-suited to the topic we are interested in, the crawler will be able to also retrieve pages that are of the given topic. On the other hand, if the crawler issues terms that are completely irrelevant, it will simply waste its resources by downloading pages that are of no interest to the users (and potentially also degrading the quality of the search engine that employs the crawler).

To illustrate, we assume that the crawler downloads pages from a Hidden Web site that contains a set of pages S (the rectangle in Figure 1b). These pages might cover a vast area of topics, most of which may potentially be irrelevant to our search. The results of each potential query q_i can be considered as a subset of S that contains the pages that the site would return as a result.

In practice, we are interested only in pages relevant to a specific topic (i.e. the squares in Figure 1b). To this end, in order to minimize the total cost of crawling, we need to discover queries that focus on the topical area of our interest, and in that way avoid the extra cost of issuing queries that return irrelevant results.

In order to select the best keywords, at each step of the algorithm, we examine the candidate keywords that we have retrieved from the collection so far and we pick one to be issued as our next query. We describe the keyword extraction and selection processes in section 2.2. In order to bootstrap our algorithm, we consider an initial description of the topic as a set of terms. This initial description of the topic can either be one or more terms that are relevant to the topic, or one or more documents relevant to the topic. After the first step, and after we have downloaded each batch of results based on the term that we queried, we augment our list of candidate terms to include terms that were newly found. This however, implies that we need to be able to determine whether the downloaded page pertains to the desired topic or not. For this step, we also employ a classifier as we will discuss in more detail in Section 2.3.

Algorithm 2.1

```
(1) WordCollection = di;
(2) while (available resources) do
      // extract the terms and build a WordCollection
(3)   if(cnt++ mod N == 0)
      T(WordCollection) = ExtractTerms(WordCollection);
      // select a term to send to the site
(4)   qi = SelectTerm(T(WordCollection));
      // send query and acquire result index page
(5)   R(qi) = QueryWebSite(qi);
      // download and evaluate the pages of interest
(6)   DownloadAndEvaluate(R(qi), pi, WordCollection);
    done
```

Fig. 2. Topic-Sensitive Crawling of a Hidden Web Site

Figure 2 shows the algorithm for a Topic-Sensitive Hidden Web Crawler. First, the Word Collection is initialized with an exemplary document, d_i. Step (3) of the algorithm is only issued once every N times –to reduce the number of required computations– and it extracts the best terms of the Word Collection. This is done by calculating the *TF/IDF* weight for every term of the collection. The process is explained in detail in Section 2.2. Step (4) picks the best of the terms that Step (3) extracted from the document, that has not been used so far, while step (5) uses this term to issue a query and retrieves the result index page. Finally, Step (6) downloads the Hidden Web pages that were included in the results and had not been previously downloaded and evaluates the contents of the results using one of the policies (p_i) presented in this work. The evaluation process of this step is responsible for the maintenance of the collection of words used for keyword extraction. This process depends heavily on the policy that is to be followed. The different policies are explained in Section 2.3. This process is repeated (Step (2)) until we use our available resources (e.g. number of queries allowed, storage, bandwidth, etc.)

2.2 Word Collection

In this section we provide the details of the Word Collection that serves as a pool of possible queries for our algorithm.

This pool of words is the document d_i used in Step (1) of the Algorithm 2.1. It consists of text that is related to the topic in search and is initialized with an exemplary document that describes that topic. So, if for instance we wanted to crawl for sports articles from a news site, we could provide the algorithm with an input document (or snippet, or a small set of keywords) that would consist of a few sport-related articles. However, the Word Collection cannot remain static during the execution of the algorithm for a variety of reasons.

First, the input document given to the algorithm during the initialization of the *Word Collection*, may not be enough for the extraction of all (or enough) terms that are needed for the retrieval of a sufficient amount of Web Pages. No matter how good that initial document may be in capturing the essence of the topic in search, it may manage to provide only a limited number of terms. Second, the initial *Word Collection* may be too specific, in a way that the terms extracted would not be general enough to capture the whole topical area of the document. For instance, if those sport-related articles mentioned earlier, were taken from a WNBA fan-site, the terms extracted from the *Word Collection* would retrieve results concerning women's sports and basketball. We are interested in matching the input document with a broad topic, in order to retrieve all the related Web Pages. Therefore, it is necessary to broaden our *Word Collection* during the execution of our algorithm. Finally, to successfully retrieve the maximum amount of Web Pages from a Hidden Web site, it is essential that we adapt to its terminology. For instance, we cannot retrieve but a subset of Web Site that indexes multilingual content if all the words in our *Word Collection* are English.

Therefore, it is clear that for effective Topic-Sensitive Hidden Web Crawling, the pool of words that is used for term extraction must be enriched continuously as well as adapted to the site in search. To face this issue, we exploited the contents of the results as potential parts of the *Word Collection*. Each result page is evaluated using one of the policies described in Section 2.3 and the contents of the ones relevant to the topic in search are added to the *Word Collection*.

In that way, the *Word Collection* is able to provide a sufficient amount of terms to be issued as queries. Furthermore, since the *Word Collection* is updated with content directly from the site in search, it can provide the algorithm with terms that not only are relevant to a specific topic, but have a higher significance for that site.

In order to effectively retrieve results, the queries picked from the Word Collection must be focusing on the topic that we are interested in. To successfully address this matter, in Step (4) we use the well-known *TF/IDF* term weighting system that measures accurately the general importance of a term in a collection.

2.3 Result evaluation policies

In this section we provide the details of the various evaluation policies that we employ in our work. These policies are used in order to decide if each page contained in the results is relative to the topic in search, and therefore will be helpful later. The contents of the pages that are considered in-topic are added to the *Word Collection*, and consequently take part in the keyword selection process.

The policies that we consider in our work are the following:

– ***perfect:*** We use the categorization information directly from the site in search. Each document of *dmoz* and *Stack Exchange* is classified into topics and this policy utilizes this knowledge. Of course such information is not available in most cases. In our work, we will use this policy as a benchmark to determine how well the rest of the policies can do relative to this one that has perfect information regarding the topic that every result document belongs to.
– ***do-nothing:*** We accept all of the returned pages as in-topic. This policy updates the *Word Collection* with every page the crawler manages to discover. Since the first few terms are extracted from the input document it is expected that the first queries that will be submitted will be meaningful and so the corresponding result pages will have a high chance of being in-topic. However, since the results are never filtered it is expected that a lot of out-of-topic pages will find their way to the Word Collection and affect the term selection process significantly. Thus, this policy can also be used as a comparison point for the other policies.

– ***NaiveBayes:*** We use a Naive Bayes classifier that decides if a result is relevant or not to the topic in search. This policy examines the effects of using a topic classifier during the crawling process. The classifier needs to go through a training phase before it can be used. We feed the classifier with samples of documents belonging to a certain topic and then are able to test if other documents should be classified under this topic or not. Therefore, it is clear that this method can only be used for topics that the classifier is already trained for. Obviously, the better the classifier is trained the less erroneous pages will be added to the *Word Collection*.
– ***CosineSimilarity:*** We examine the cosine similarity of every result page with the initial example document and accept only a small percentage of the closest pages. In this way we ensure that the pool of words will be enriched only with terms closely related to the topic defined by our input. This policy is more flexible compared to the NaiveBayes policy in terms of adaptability as it requires no training. However, since this method depends heavily on the input document, we will examine how the quality of the input document affects performance in our experimental section.

3 Experimental Evaluation

3.1 Experimental Setup

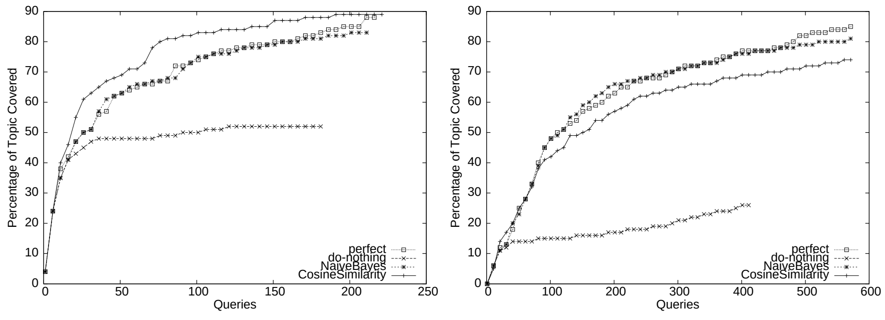
We start our experimental evaluation by presenting the datasets used in our experiments. Then, we demonstrate the performance of the proposed policies as described in Section 2.3 for a variety of topics. For the experiments we present here, we set variable N of Algorithm 2.1 to 7. That is, we update the *Word Collection* of Algorithm 2.1 every 7 queries. We experimentally tested different values for N and we found that 7 is essentially the breaking point after which we would observe a significant degradation in terms of the performance of our algorithms. Additionally, during the operation of the *CosineSimilarity* policy, we kept the top 1% of the returned documents.

For our experiments we used two different datasets, namely the Open Directory Project [1] and all the public non-beta *Stack Exchange* sites [2] contents. *Dmoz* indexes approximately 5 million links that cover a broad area of topics. Each link is accompanied with the site's title, a brief summary and its categorization. The links are searchable through a keyword-search interface and *dmoz* enforces an upper limit on the number of returned results (10,000 results). The Stack Exchange sites contain a total of 391,522 questions over twenty different topics. In order to further examine the performance of our algorithms, we enforced an upper limit of 1,000 results per query for this dataset as well. We considered the titles and summaries of each indexed link of the *dmoz* website and the questions of the *Stack Exchange* sites as documents, and allowed our policies to utilize them during the keyword extraction process.

For every topic we studied here, we needed an initial exemplary document to serve as a representative of that topic. In the following, we used random documents from the sites from the respective topics to serve as such exemplary documents. We do not report on how selecting a different initial document affects performance as we experimentally confirmed the findings of [17] that the selection of the first document does not significantly affect the returned results. However, we do study how the *number* of initial exemplary documents affects performance in Section 3.5.

3.2 Evaluation of the different policies over the same topic

We start our evaluation by studying the effectiveness of the policies proposed in Section 2.3 in retrieving pages of a given topic from *dmoz* and *Stack Exchange*. We show the fraction of documents returned for *dmoz* and *Stack Exchange* as a function of the queries issued in Figures 3a and 3b respectively. For *dmoz* we report results for the topic *Sports* while for *Stack Exchange* we report results for the topic *Wordpress*. The results over topics other than Sports and Wordpress, are very similar to figures 3a and 3b. We study the performance of our policies on additional topics in Section 3.4.



(a) Topic Sports of *dmoz* (b) Topic Wordpress of *Stack Exchange*
Fig. 3. Results of the four policies over two datasets

For the topic *Sports dmoz*, contains a total of 90,693 pages. As Figure 3a shows, all four policies behaved identically for the first seven queries, where only the initial document is being used for keyword extraction. *Do-nothing*'s performance was enormously affected after those first few queries, when the *Word Collection* was updated with all the returned results, since as the name of the policy implies, no action was taken to filter them. As a consequence, *Do-nothing* performed badly, as expected. Policies *Perfect* and *NaiveBayes* behaved similarly with the first one managing to retrieve a slightly bigger percentage of *Sports*-related documents. This can be explained from the fact that the two policies had a high percentage of common queries issued (28%). After 210 queries, the *Perfect* policy retrieved 86.43% of the total relevant documents, while the *NaiveBayes* policy managed to collect 83.42% of them.

The *CosineSimilarity* policy managed to outperform all three of them by retrieving 89.66% of the documents categorized under the Topic *Sports* after issuing about 180 queries. 19% of the terms used for query submission were common with the *Perfect* policy and 20% of them were common with the *NaiveBayes* policy. This implies that *CosineSimilarity* does a better exploration of the keyword space compared to the other policies and it can find keywords leading to more relevant documents earlier compared to the Perfect policy which takes about 220 queries to achieve the same performance.

We also retrieved all the documents of the *Stack Exchange* sites relevant to the *Wordpress* publishing platform. There was a total of 17,793 questions categorized under this topic. The results are illustrated in figure 3b. After 572 queries, the *Perfect* policy retrieved 85% of the total relevant documents, the *NaiveBayes* policy retrieved 81% and the *CosineSimilarity* policy managed to collect 71% of them. The significant amount of more queries needed for the retrieval of documents from this dataset is explained by the much smaller upper limit of returned documents per query.

3.3 Queries issued and topic accuracy

In order to investigate more closely the performance of our policies we further examined the actual queries issued by each policy. We present a sample of queries issued by each policy together with the precision achieved (i.e. fraction of in-topic documents) in Figure 4.

One thing that we should note is that there is a lot of overlap in the results of each term. Although every policy manages to discover meaningful terms that return a good number of results, a large portion of them has been already discovered by previous queries. Additionally, the *Do-nothing* policy is the most successful in finding "popular" terms. Most of the terms illustrated in Figure 4 returned the maximum of 10,000 results, while the average after 210 queries was 8,650. This is due to the fact that the Word Collection of this policy was eventually enriched with terms from every possible topic of the Open Directory Project. The *NaiveBayes* policy was second with 7,927, the *Perfect* policy third with 7,530 and the *CosineSimilarity* policy last with 7,426 results per query.

Using the *dmoz* categorization information for every downloaded document, we also measured the precision of the different policies as shown in Figure 4. Overall, the *Perfect* policy was the most successful one since it is allowed to use the class information. Of course, in practice such information is not available and thus this policy serves as a benchmark in terms of classification precision. The *Perfect* policy achieved a 68.2% precision on average.

The *Do-nothing* policy chooses to accept every document it downloads as relevant to the topic in search, so its errors are equal to the total number of links retrieved minus the links that were actually in-topic. It's accuracy was 7.8% on average. The classifier used in the *NaiveBayes* policy on the other hand, proved to be very effective and achieved a 64.4% precision on average. However, the impact of all errors is not the same for our algorithm. An in-topic document that is classified as irrelevant is not added to the Word Collection and does not affect the term extraction process. On the other hand, an irrelevant document that "sneaks" its way into the Word Collection, may cause the selection of inappropriate terms for query submission. For the *NaiveBayes* policy 45% of the documents added to the Word Collection, were actually categorized under another topic in the *dmoz* classification taxonomy.

Finally, the *CosineSimilarity* achieved a 58.6% precision on average. However, this did not affect the policy in a negative way. The retrieved documents, despite the fact that they belong to a different topic, had very high cosine similarities with the input document, so naturally, the Word Collection was not altered in an undesirable way. As a result, the *CosineSimilarity* policy outperformed the other policies in terms of recall as we showed in the previous section.

No	Term	Precision	Term	Precision	Term	Precision	Term	Precision
1	results	42.83%	results	42.83%	results	42.83%	results	42.83%
2	statistics	43.61%	statistics	43.61%	statistics	43.61%	statistics	43.61%
3	roster	71.36%	roster	71.36%	roster	71.36%	roster	71.36%
10	men	27.26%	schedules	10.31%	schedules	10.31%	tables	5.61%
15	scores	27.89%	church	0.00%	standings	67.96%	player	24.36%
20	players	30.62%	coaching	17.89%	baseball	38.29%	players	33.70%
25	hockey	41.85%	methodist	0.00%	records	8.38%	hockey	44.08%
30	tennis	7.43%	beliefs	10.11%	membership	1.52%	baseball	32.20%
40	rugby	14.43%	stellt	0.00%	county	0.39%	race	14.56%
60	sport	3.82%	bietet	0.00%	fc	5.45%	conference	1.73%
100	competition	12.28%	nach	0.00%	standing	26.66%	competitive	11.61%

(a) *Perfect* (b) *Do-nothing* (c) *NaiveBayes* (d) *CosineSimilarity*

Fig. 4. Queries issued by the different policies.

3.4 Comparison of policies under different topics of dmoz

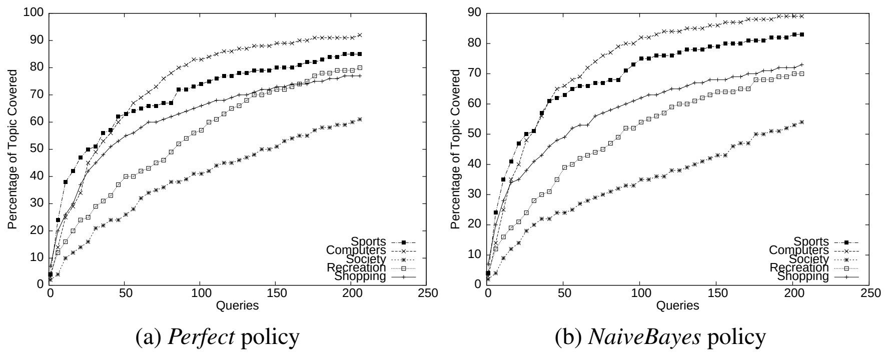
In this section, we present the performance of the policies *Perfect*, *NaiveBayes* and *CosineSimilarity* over five different topical areas belonging to the classification taxonomy *dmoz* uses. To this end we used the following topics: Computers (103,336 documents), Recreation (91,931 documents), Shopping (87,507 documents), Society (218,857 documents) and Sports (90,639 documents).

Figure 5a illustrates the behavior of our approach for these five different categories of *dmoz* using the *Perfect* policy. Topics *computers* and *sports* proved to be the easiest to retrieve while *society* needed a significantly bigger amount of queries to surpass the 80% barrier. This is due to the fact that *society* is a much larger category compared to the rest. More specifically, topic *Computers* returned 92.17% of the total documents after 210 queries. Topics *Sports* and *Recreation* discovered 86.43% and 80.76%, respectively, with the same amount of queries. Finally for the topics *Shopping* and *Society* the policy collected 77.82% and 62.06%, respectively.

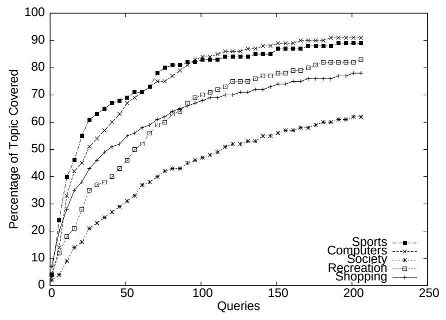
The results for the said topics using the *NaiveBayes* policy are presented in Figure 5b. This policy is performing slightly lower than *Perfect* for each of the five topics. The ordering of the topics is, however, a little different, since the *NaiveBayes* policy behaved very poorly for the topic *Recreation*, which was ranked fourth below the topic *Shopping*. More explicitly, after 210 queries, topics *Computers* collected 89.81% of the documents, topic *Sports* 83.42%, topic *Shopping* 73.34%, topic *Recreation* 70.86% and topic *Society* 54.86%. This is due to the fact that *Recreation* is a much broader topic than the rest and thus *Perfect* can benefit more by knowing the topic of the documents beforehand.

In Figure 5c we observe similar results for the *CosineSimilarity* policy. Topics *Computers* and *Sports* were again first with 91.84% and 89.66%, respectively, after issuing 210 queries. Topic *Recreation* collected 83.55% of the related documents, while topics *Shopping* and *Society* returned 79.02% and 62.97%, respectively, after the same amount of queries.

At a high level, the *CosineSimilarity* policy performed slightly better than the *Perfect* policy in 4 of the 5 topics examined, *Computers* being the only exception. Addi-



(a) *Perfect* policy (b) *NaiveBayes* policy



(c) *CosineSimilarity* policy

Fig. 5. Comparison of polices on different topics.

tionally, it out-scored the *NaiveBayes* policy for every one of the five different topics. Topic *Recreation* behaved better than *Shopping* after the 87th query, as it did with the *Perfect* policy after the 162nd query.

3.5 Impact of input document size

The *CosineSimilarity* policy depends heavily on the input document, since it does not use it to only extract the first few queries to be issued, but to evaluate the result documents retrieved as well. To this end, we examine the impact of the size of the initial document on the behavior of this policy. The other three policies use the initial document only for the first step of the process, so they are not affected as much by the size of the initial document.

Figure 6 illustrates the results for *CosineSimilarity* under three different input documents of variable size, while retrieving documents from *dmoz* under the *Computers* category. As we can see, as we limit the input size, the performance deteriorates. However, even for a very small initial document we can still get reasonably good results. More specifically, using an input document that consists of 1,000 titles and summaries of links indexed by *dmoz*, *the CosineSimilarity* policy retrieved 91.14% of the relevant documents after 190 queries. In comparison, with only 100 titles and summaries, the policy discovered 87.77% of the documents with the same number of queries. With an input document of 50 titles and summaries, it retrieved a sizeable 86.67% of them.

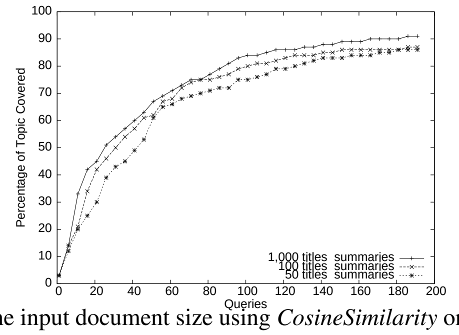


Fig. 6. Impact of the input document size using *CosineSimilarity* on topic *Computers*

3.6 A high-level comparison to generic Hidden Web crawling

In [17], an experiment on crawling all of the *dmoz* site's documents is presented. It takes about 700 queries to retrieve a little over 70% of its contents and after that we would have to analyze the pages to identify which ones belong to the topic at hand. The *CosineSimilarity* policy proposed in this paper was able to download 70% of topics *Sports* and *Computers* after 52 and 60 submitted queries respectively. Therefore, we can significantly reduce the cost of crawling when we are interested in a specific topic by using our techniques.

4 Related Work

Research on the Hidden Web has emerged during the last years. In [18], Raghavan et al. first introduced the problem of Hidden Web crawling by presenting an architectural model for a Hidden Web crawler. Their efforts in this work, mainly focus on learning Hidden-Web query interfaces. The potential queries are either provided manually by users or collected from the query interfaces. Similar work is presented in [6] where some new strategies for effective discovery of Hidden Web forms are presented, while [14] discusses the approach used from Google in filling such Web forms.

In [5, 17, 21] the problem of automatically producing meaningful queries that can return large fractions of a document collection is examined. [17] provides a theoretical framework for analyzing the process of generating queries for a document collection as well as examining the obtained result. In addition, the framework is applied to the problem of *Hidden Web crawling* and the efficiency of the approach is quantified. In [5], a number of methods for building multi-keyword queries is presented and experimentally evaluated to show that a large fraction of a document collection can be returned. Wu et al. [21] focus on the core issue of enabling efficient Web Database crawling through query selection and propose a theoretical framework that transforms the database crawling procedure into a graph-traversal problem; in the latter, the proposal is based on following up "relational" links. [20] developed a new set covering algorithm to address the problem of web database crawling and in [13], the incremental web database crawling problem is examined. Our work differs from the above efforts as we retrieve only portions of a *Hidden Web* site that are relevant to a specific topic.

Chakrabarti et al. studied ways to selectively search for pages relevant to a predefined set of topics [9]. A complete process for discovering topic-specific resources from the Web is presented. A classifier and a distiller are used for the evaluation of crawled pages based on their hypertext. However, this methodology is only applicable on the publicly indexable web, since hyperlinks are not used at all on the Hidden Web. Since then, there has been many similar work made [16, 10, 15, 4], again for the publicly indexable web. In [7] the authors present a method for determining the topic of a given page based on keywords found in its URL. In our case, we opted to also use content from the page since we download such content anyway in order to get access to keywords that will be used to update our Word Collection.

In [23], Yang et al. propose a method for retrieving documents relevant to a query document. Candidate phrases are extracted from an input document and a score is assigned to each one of them, using two different policies. The first is a linear combination of the total *TF/IDF* score of all the terms of the phrase and their degree of coherence. The second is based on the probability of pairs of terms occurring in the same document (mutual information based). Our goal is different in that we aim at retrieving as many documents as possible around the topic of the input document. In addition, we proceed iteratively and learn from previously downloaded documents in order to determine our queries.

In [3], a focused crawler for the Hidden Web is presented. Its main focus is on extracting information from labels of form elements, to associate Hidden Web sites to topic domains and to analyze the forms to find ways to execute queries automatically. However, the queries to be issued are not produced automatically, but are picked from predefined resources. Bergholz et al. have also addressed this matter in [8]. However, as the case was before, they use predefined sets of keywords named “querying resources” in order to retrieve results from the collections it discovers. In [12], Ipeirotis et al. attempted to categorize Hidden Web Databases by probing them with queries and evaluating the amount of the results. They used pre-classified documents to train a rule-based document classifier and transformed the rules it produced, to queries for use. Similarly, [19] presents a technique called informed probing where potentially available metadata (e.g. title, or meta tags) from the entry page to a Hidden Web site can be used as probe keywords in order to categorize the Hidden Web sources.

5 Conclusion and Future Work

We examined how we can build a Topic-Sensitive Hidden Web Crawler that, given an exemplary document, can effectively retrieve those documents that are relevant to a certain topic. We presented an algorithm for a Topic-Sensitive Hidden Web Crawler and proposed and experimented with a variety of policies deployed to help us measure the relevance of the returned documents with the searched topic. Our experimental evaluation indicates that our suggested algorithm has great potential for harnessing topic-specific documents.

In the future, we plan to exploit diverse query formulations in order to further reduce the overheads involved in the process, to experiment with additional Hidden Web sites that may freely provide their data, and finally, to explore techniques that may incrementally retrieve recently updated content.

References

1. The Open Directory Project `http://www.dmoz.org`.
2. Stack Exchange `http://stackexchange.com/`.
3. M. Álvarez, J. Raposo, A. Pan, F. Cacheda, F. Bellas, and V. Carneiro. Deepbot: a focused crawler for accessing hidden web content. In *Proc. of Int. Workshop on Data Enginering Issues in E-commerce and Services*, DEECS '07, NY, USA, 2007.
4. N. Angkawattanawit and A. Rungsawang. Learnable crawling: An efficient approach to topic-specific web resource discovery, 2002.
5. L. Barbosa and J. Freire. Siphoning hidden-web data through keyword-based interfaces. In *Proceedings of SBBD*, Brazil, 2004.
6. L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *Proceedings of the WWW Conference*, NY, USA, 2007.
7. E. Baykan, M. Henzinger, L. Marian, and I. Weber. Purely url-based topic classification. In *Proceedings of the WWW Conference*, Madrid, Spain, 2009.
8. A. Bergholz and B. Chidlovskii. Crawling for domain-specific hidden web resources. In *Proceedings of WISE*, DC, USA, 2003.
9. S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. In *Proceedings of the WWW Conference*, NY, USA, 1999.
10. M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *Proceedings of VLDB*, CA, USA, 2000.
11. P. G. Ipeirotis and L. Gravano. Distributed search over the hidden web: hierarchical database sampling and selection. In *Proceedings of VLDB*, Hong Kong, 2002.
12. P. G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: categorizing hidden web databases. *SIGMOD Rec.*, 30:67–78, 2001.
13. W. Liu, J. Xiao, and J. Yang. A sample-guided approach to incremental structured web database crawling. In *Proceedings of ICIA*, Harbin, China, 2010.
14. J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's deep web crawl. In *Proceedings of VLDB*, Auckland, New Zealand, 2008.
15. F. Menczer, G. Pant, and P. Srinivasan. Topic-driven crawlers: Machine learning issues. *ACM TOIT, Submitted*, 2002.
16. S. Noh, Y. Choi, H. Seo, K. Choi, and G. Jung. An intelligent topic-specific crawler using degree of relevance. In *IDEAL*, volume 3177 of *Lecture Notes in Computer Science*, pages 491–498, 2004.
17. A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden web content through keyword queries. In *Proceedings of JCDL*, NY, USA, 2005.
18. S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proceedings of VLDB*, San Francisco, CA, USA, 2001.
19. S. Seshadri and B. F. Cooper. Routing queries through a peer-to-peer infobeacons network using information retrieval techniques. *IEEE TPDS*, 18:1754–1765, 2007.
20. Y. Wang, J. Lu, and J. Chen. Crawling deep web using a new set covering algorithm. In *Proceedings of the ADMA Conference*, Berlin, Heidelberg, 2009.
21. P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma. Query selection techniques for efficient crawling of structured web sources. In *Proceedings of the ICDE*, Washington, DC, USA, 2006.
22. W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *Proceedings of SIGMOD*, NY, USA, 2004.
23. Y. Yang, N. Bansal, W. Dakka, P. Ipeirotis, N. Koudas, and D. Papadias. Query by document. In *Proceedings of WSDM*, NY, USA, 2009.