Noname manuscript No.
(will be inserted by the editor)

Focused Crawling for the Hidden Web

Panagiotis Liakos · Alexandros Ntoulas · Alexandros Labrinidis · Alex Delis

Received: date / Accepted: date

Abstract A constantly growing amount of high-quality information resides in databases and is guarded behind forms that users fill out and submit. The Hidden Web comprises all these information sources that conventional web crawlers are incapable of discovering. In order to excavate and make available meaningful data from the Hidden Web, previous work has focused on developing query generation techniques that aim at downloading all the content of a given Hidden Web site with the minimum cost. However, there are circumstances where only a specific part of such a site might be of interest. For example, a politics portal should not have to waste bandwidth or processing power to retrieve sports articles just because they are residing in databases also containing documents relevant to politics. In cases like this one, 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. We investigate how we can build a 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 area. In this regard, we present an approach that progresses iteratively and analyzes the returned results in order to extract terms that

P. Liakos
Univ. of Athens, 15784, Athens, Greece
E-mail: p.liakos@di.uoa.gr

A. Ntoulas
Zynga, San Fransisco, CA 94103
E-mail: ntoulas@gmail.com

A. Labrinidis
Univ. of Pittsburgh, Pittsburgh, PA 15260
E-mail: labrinid@cs.pitt.edu

A. Delis
Univ. of Athens, 15784, Athens, Greece
Tel.: +30-210-727.5242
Fax: +30-210-727.5214
E-mail: ad@di.uoa.gr

capture the essence of the topic we are interested in. We propose a number of different crawling policies and we experimentally evaluate them with data from four popular sites. Our approach is able to download most of the content in search in all cases using a significantly smaller number of queries compared to existing approaches.

Keywords Hidden Web · focused · crawling · topic-sensitive · query selection

1 Introduction

An ever-increasing amount of high-quality information on the Web today is accessible through Web pages which extract information from data sources such as databases or content management systems. The access to this information is guarded by search interfaces that generate requests. Therefore the information is hidden from conventional crawlers, which base their operation upon a static link structure of the Web and are incapable of discovering dynamically generated sites. For this reason, these pages are collectively termed the *Hidden Web* (or the *Deep Web*).

The amount of available information in the *Hidden Web* is believed to be at least an order of magnitude larger than the currently searchable WWW [6, 12, 7, 18]. Moreover, it is of higher quality than that available in ordinary web pages, as it has been carefully reviewed, edited or annotated before being stored in a database or a content management system. Furthermore, it presents a high degree of structure and may span a multitude of topics ranging from sports and politics to different medical treatments of a particular disease [14, 5].

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 content of *Hidden Web* sites. They will be able to clean, aggregate and make available data from several sources. Having information from the *Hidden Web* 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.

Since the data behind a *Hidden Web* site are reachable by search engine crawlers only through dynamically issued queries to its search interface, the database community has spent much effort investigating ways of digging them out. In most cases [21, 23, 2, 4, 13, 27, 28] 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., the queries issued. For example, the technique in [21] 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 the information buried

there. For example, a search engine that specializes in traveling may benefit from picking only news articles that pertain to traveling from a general-purpose news database. A portal regarding 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 area.

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 [21, 23, 2] to retrieve the majority of the *Hidden Web* sites and then keep only 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 depletion of server 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, whether that is quantified as a number of queries allowed or the available bandwidth. Our main idea is to issue queries to the *Hidden Web* site that are very relevant to the topic that we are interested in, proceeding in an iterative fashion. We discover terms for our queries from the documents which we have already crawled, after evaluating their closeness with our topic in search, and then use a ranking function to select the most promising ones. In that way, we manage to generate a set of keywords that are able to download the contents of a *Hidden Web* sit at a low cost.

In summary, this paper makes the following contributions:

- We formalize the problem of focused *Hidden Web* crawling, i.e. downloading the pages of a *Hidden Web* site that pertain only to a given topical area of interest.
- We present an approach for performing focused *Hidden Web* crawling. Our key idea is to identify candidate keywords from the crawled documents that are relevant to the topic of interest using repetitive feedback.
- We propose a number of different evaluation policies that can be used to decide which of the crawled documents may contain proper candidate queries that we can issue next. As we show in our experimental section our policies result in much better production of good keywords than the baseline approach.
- We experimentally evaluate our approach using the different policies on four real *Hidden Web* sites using two different metrics and we showcase the merits of each of the policies. Our crawler manages to retrieve the desired content in all tested setups using only a small part of the resources a generic *Hidden Web* crawler requires.

The rest of this paper is organized as follows. Section 2 reviews related work and provides the necessary background. Section 3 formalizes the problem investigated here and presents the details of our approach. A comprehensive

set of experiments using a wide dataset and different evaluation metrics is presented in Section 4 and Section 5 concludes this paper.

2 Related Work

Research on the Hidden Web has emerged during the last decade. In [23], Raghavan et al. introduced the problem of crawling the Hidden Web with a generic operation model of a task-specific Hidden Web crawler and a prototype implementation of this model, namely HiWE (Hidden Web Exposer). Their efforts focus mostly in overcoming the challenge of automatically parsing, processing and interacting with HTML forms. The visual layout of form elements, such as the distance between two input elements, is one of the methods used. Their approach requires human assistance in order to ensure the crawler issues queries relevant to the particular task, and they only consider search interfaces with multiple attributes forms. In [5], Bergholz et al. perform syntactic analysis to HTML forms in their approach of automatically identifying Hidden Web resources. Their crawler is domain-specific and is initialized with pre-classified documents and relevant keywords, i.e., they do not deal with the issue of automatic form filling. DeepBot [1] is very similar to HiWE. The visual layout is also used here, along with text similarity heuristics, so that Hidden Web site interfaces can be associated with domains and queries can then be executed on them. To overcome the complexities related to the client-side, such as JavaScript code and sessions, DeepBot uses Microsoft Internet Explorer's API. The queries issued are not produced automatically. After associating an interface with a domain, using the preconfigured domain definitions, the crawler begins a specific data collection task utilizing its corresponding predefined input terms. In [30], Zhang and Chang notice that query forms tend to reveal a "concerted structure". They hypothesize the existence of a *hidden syntax* that query interfaces follow, and build an ambiguous grammar and a best-effort parser to cope with it. Barbosa and Freire propose FFC [3] and ACHE [4], two frameworks aiming to automate the process of discovering *Hidden Web* entry points from which templates can then be generated. The latter, attempts to overcome two limitations of the former, which retrieved highly heterogeneous forms and required significant effort for tuning and training. Our work assumes that the entry point of a given *Hidden Web* site has already been discovered.

Google's approach on surfacing the Hidden Web is outlined in [18]. Good surfacing candidates are elected after probing forms and examining if the results retrieved are sufficiently distinct from each other. A comparison of their algorithm, named ISIT, with approaches that generate URLs for each entry in the Cartesian product of all input values of a form, or combine a smaller set of the available inputs - since forms often tend to be large - is provided, that shows that the proposed approach is highly effective. In order to generate input values *tf/idf* scores are used. The problem of automatically producing meaningful queries that can return large fractions of a document collection has

been examined numerous times. Barbosa and Freire [2] suggested that terms present in a document collection will be appropriate candidates for submission to the corresponding interface. Thus, they sample the collection to select a set of high-frequency keywords and use those to construct queries with high coverage. Among the issues they study is the usefulness of stop words in their queries, which tend to be extremely effective in some of the document collections examined but are ignored in others. In [21], a theoretical framework for analyzing the process of generating queries for a document collections as well as examining the obtained result is provided. In addition, the framework is applied to the problem of Hidden Web crawling and the efficiency of the approach is quantified. Three policies, namely *random*, *generic-frequency* and *adaptive* are examined. The first two use a set of words from a 5.5-million-Web-page corpus whereas the last one discovers new terms by utilizing the result pages. The results indicate that the *adaptive* policy, which is the most interesting one due to its automatic nature, leads to more effective crawling. Wu et al. [27] suggest that the goal of query selection algorithms is to find a *Weighted Minimum Dominating Set* in the corresponding attribute-value graph. Since this is a well known NP-hard problem, they propose a greedy link-based query selection method that traverses the database graph by following some "hub" nodes. Their approach can only function in the context of highly structured document collections, such as *DBLP* [1] and *IMDB* [2] that are used in their experimental setup. In [17] and later on in [26], Lu et al. incorporate sampling methods in an effort to select appropriate keywords and retrieve the entire database. They sample a database and then select terms that are efficient for this sampled part, using greedy methods. Their results suggest that the selected queries are effective for the full database as well. Our work differs from these efforts as we retrieve only portions of a Hidden Web site that are relevant to a specific topic, instead of simply maximizing the coverage of the underlying web database.

Ipeirotis et al. [14] propose a technique of classifying databases through probing. They train a rule-based document classifier using a set of preclassified documents, count the number of matches the database achieves for each query they issue, and use this number along with the classifier to categorize the database. By following this approach, no retrieval of documents from the database at all is needed for the categorization. In a later work [13], Ipeirotis and Gravano create content summaries of web-accessible text databases. They use focused probing to query the database, retrieve a sample and eventually categorize it. They also present an approach of locating the most topically-specific databases for a given query. Their results indicate that the proposed techniques offer very effective content-summary construction and database selection. This work is however more relevant to generic *Hidden Web* crawling, since the categorization they perform is for the whole database. Yang et al. [29], introduce the idea of using a "query document", from which phrases regarding

[1] `www.informatik.uni-trier.de/~ley/db/`

[2] `www.imdb.com/`

a certain topic can be extracted, in order to retrieve relevant content from a database. These phrases are complemented with extra information with the use of external resources such as Wikipedia. In order to rank candidate phrases they employ a *td/idf* based scoring. Their approach targets the retrieval of a small number of relevant documents of high quality information as opposed to gathering the thematically close documents in their entirety.

Chakrabarti et al. [8] study ways to selectively search for pages on a specific set of topics that represent a relatively narrow segment of the Web. A comprehensive 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. The former learns to recognize relevance from examples embedded in a topic taxonomy whereas the latter identifies topical vantage points on the Web. However, their methodology is only applicable on the publicly indexable web, since hyperlinks are not used at all on the Hidden Web. Since then, many similar efforts have been proposed, but again only for the surface web. Diligenti et al. [9] attempt to improve the efficiency with which content related to a certain category can be found, by modeling the links and the content of documents that are closely linked with target pages. In [20], associations between web pages and predefined categories are identified by using term-classification rules compiled by machine learning algorithms. These approaches however, deal with topic-sensitive crawling of the publicly indexable Web, whereas we target the *Hidden Web*.

3 Topic-sensitive Hidden Web crawling

At a high level, the purpose of a generic focused (or topic-sensitive) web crawler [8] is to collect pages from the Web that pertain to a given topic. To do so, an evaluation of the content of each downloaded page during crawling is necessary, in order to decide whether the page is relevant to a particular topic or not. In the former case, its out-links are possibly good candidates whereas in the latter, its descendants will most likely be a waste of resources. Hence, a topic-sensitive crawler aims at examining as small of a subset of the total search space as possible, by discarding pages irrelevant to the topic of the search.

An illustration of this process for the publicly indexable Web is provided in Figure 1. Colored boxes represent pages with content relevant to our topic of interest, whereas white boxes represent irrelevant ones. To maximize its efficiency and use the available resources wisely, a topic-sensitive crawler has to follow only those paths of the figure that lead to colored boxes.

The case is very different however for the *Hidden Web* and its dynamically generated sites, so the method described in [8] cannot be employed. In order to retrieve the content of such a site, one has to issue queries to its entry point, i.e., a search interface with one or more fields.

To access pages from a *Hidden Web* site we typically need to apply the following steps:

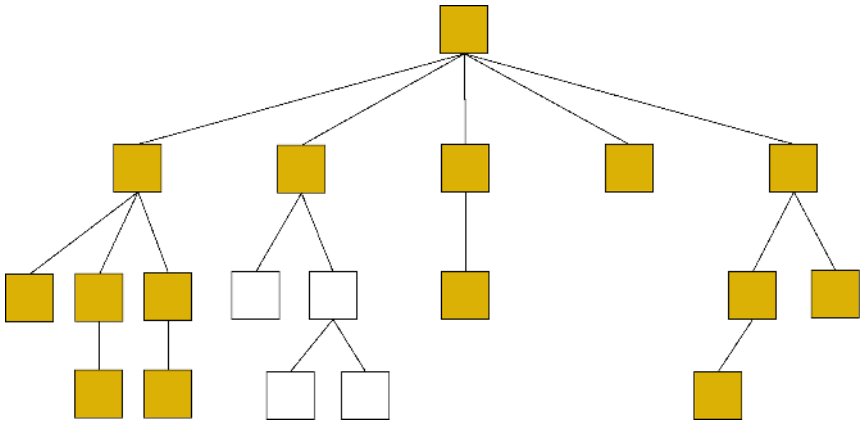

Fig. 1: Topic-sensitive crawling in the publicly indexable Web.

1. First, submit a query through the interface provided by the site, that characterizes what we want to locate. The most common search interface is a text search box, such as the one illustrated in Figure 2a. This is the interface a site provides to its users and enables them to enter a search term and perform a full text-search in its underlying index. Other sites may choose to offer more advanced search capabilities using a structured query interface with a subset of the content attributes, e.g., title, author and date.
2. Then, we receive a result index page, that contains links and possibly additional information of the pages that matched the submitted query. The results of a query submitted in this site are illustrated in Figure 2c. As it is shown, the results usually are sorted in order of relevance with the term in search, and are also paginated in an effort to minimize the bandwidth used for each request the server accepts.
3. Finally, after identifying a promising page in the results, we are able to follow the link and visit the actual Web page.

Therefore, it is evident that crawling the *Hidden Web* requires the submission of queries that will surface the dynamically generated content of each site. Our approach aims at crawling in a topic-sensitive manner so we must first elaborate on what constitutes a "topic", since the definition may be different depending on the application at hand. If we are interested in loosely collecting Web pages concerning football for instance, then any page that contains the words *"football"*, *"quarterback"* and *"touchdown"* may be relevant to the topic. Alternatively, in cases when we are interested in discerning more accurately between topics, e.g., external affairs vs. internal politics, a more elaborate mechanism using a category network may be used [25].

Our work does not depend on how exactly we detect whether a particular page belongs to a given topic. Instead, we make two assumptions regarding the topic definition that allow for a variety of different topic detection techniques

(a)

(b)

(c)

Fig. 2: The search interface of a *Hidden Web* site (a), a result page for the query 'Barack' (b), and a relevant page from the results (c).

to be employed. First, we generally assume that a topic can be described by a distribution of keywords. This is essentially the same assumption that most text classifiers are based on, and other topic-sensitive crawlers adopt [29]. Second, we assume that not all of the keywords in the distribution are known beforehand, as in this case the problem becomes trivial since the crawler would simply have to iterate over all of 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 *"fumble"* are also good keywords for the crawler to use. 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, by choosing to feed off of pages that pertain to the topic in search and discard those that do not.

3.1 A Topic-sensitive Hidden Web crawling approach

Due to the dynamically generated nature of the *Hidden Web* content, the task of a corresponding crawler is to automatically submit queries to a site and retrieve pages by following the links 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, it will also be able to retrieve pages that belong to the topic at hand. If instead, the crawler issues queries with irrelevant terms, it will only manage to waste processing nodes and bandwidth, downloading pages that are of no interest to users. Moreover, this potentially will degrade the quality of the search engine that employs the crawler.

Figure 3 presents a *Hidden Web* site as a set of pages S which constitute the crawler's search space. These pages might cover a vast area of topics, most of which may possibly be irrelevant to our search. We represent pages of different topics using different shapes and colors in Figure 3. The results of each potential query q_i can be considered as a subset of S, which contains the pages that the site would produce as a result.

In practice, we are only interested in downloading pages relevant to a specific topic, e.g., the squares in Figure 3).

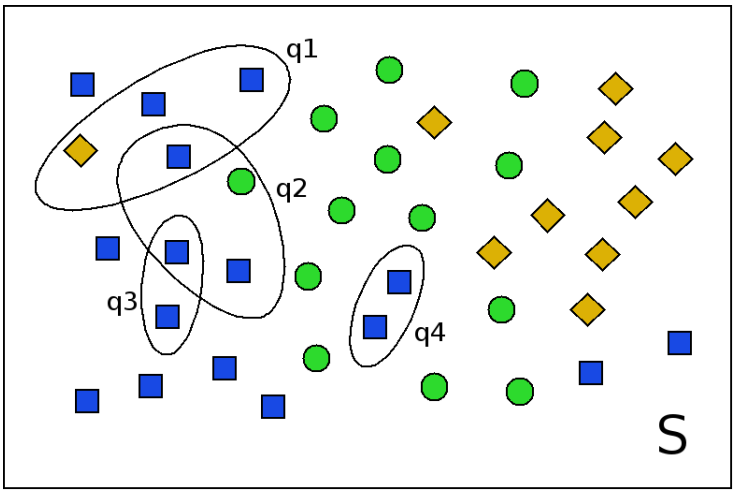

Fig. 3: Representing a Hidden Web site as a set.

Algorithm 1 A Topic-Sensitive *Hidden-Web* Crawler

```
WordCollection = d_i;                                              (1)

while (available resources) do
    // extract the terms and build a WordCollection
    if (cnt++ mod N) == 0 then
        T(WordCollection) = ExtractTerms(WorldCollection);         (2)
    end if
    // select a term to send to the site
    q_i = selectTerm(WordCollection);                              (3)
    // send query and acquire result index page
    R(q_i) = submitAndDownload(q_i);                               (4)
    // download and evaluate the pages of interest
    update(WordCollection);                                        (5)
end while
```

Algorithm 1 outlines our approach for a Topic-Sensitive *Hidden Web* crawler. In order to bootstrap the algorithm, we consider an initial description of the topic, which can either be one or more terms, or one or more documents that are relevant to the topic. This occurs in the first step of the algorithm, where the *Word Collection* is initialized with an exemplary document, d_i. The next step (2) extracts terms from the *Word Collection*, and is issued periodically, in order to reduce the number of required computations. The extraction is done by calculating the *tf/idf* weight for every term of the collection. The role of the *Word Collection* in our approach is explained in detail in Section 3.2. Step (3) picks the best of the terms that were extracted in the previous step and has not been used this far, while Step (4) uses this term to issue a query and retrieves the result index page. Finally, Step (5) downloads the *Hidden*

Web pages that were included in the results and had not been previously downloaded. Moreover, it 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 and thus, it depends heavily on the policy that is to be followed. The different policies are explained in Section 3.3. Depending on our limitations regarding time, storage or bandwidth, we can restrict the number of the algorithm's iterations. For example, we can stop its execution after submitting a fixed number of queries or reaching a particular amount of retrieved documents.

3.2 Word Collection

In this section we outline how we initialize and maintain the *Word Collection* that serves as a pool of candidate queries for our algorithm.

This pool initially comprises the terms of a "query document" d_i which serves as an example of the topical area we are interested. More thoroughly, this is a document which consists of text that is close to the topic in search and essentially, describes this topic. Thus, if for instance we wanted to crawl for sports articles from a news site, we could provide the algorithm with a "query 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, 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 can only manage to provide 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 sites that index multilingual content if all the words in our *Word Collection* are in English.

It becomes clear that for effective Topic-Sensitive *Hidden Web* Crawling, the pool of words that is used for term extraction must be enriched continuously and adapt to the site in search. To address this issue, we exploit 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 3.3 and the

contents of the ones relevant to the topic in search are added to the *Word Collection*. In this regard, the *Word Collection* gets enriched with plenty of appropriate terms that can 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 particular site.

In order to select the most appropriate keywords from the terms of the *Word Collection*, we use the *tf/idf* term weighting system, which addresses the issue of measuring the general importance of a term in a collection and allows us to distinguish those terms that are characteristic of our topic in search. A rare term that occurs frequently in a document is often more useful than a common term which appears with similar frequency in the document. This property is utilized by *tf/idf* both accurately as well as effectively [24]. Suppose a term t occurs $n_{t,p}$ times in a web page p which has a total of N_p terms. Then the term frequency (tf) of t in this page is $tf(t,p) = \frac{n_{t,p}}{N_p}$. Now, suppose that in a total of D web pages, term t occurs in d_t of them. The inverse document frequency (idf) of t is $idf(t) = log(\frac{D}{d_w})$. The *tf/idf* weight is given by the product of these two measures: $tf/idf(w,p) = tf(w,p) \times idf(w)$.

Figure 4 illustrates how the *World Collection* gets enriched during the execution of our approach. The words that already exist there are sorted according to their *tf/idf* score, and the best one not used so far is picked (*debate*) for submission. By issuing a query with this term, we retrieve new results containing words that are possibly helpful for the remaining of the process. After evaluating the retrieved content, some words that are likely to be of assistance are appended to the *Word Collection* (presidential), while some that are probably irrelevant are discarded (fox). The decision is based on one of the result evaluation policies that are detailed in Section 3.3.

3.3 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 relevant to the topic in search, and therefore will be helpful later. The content of the pages that are considered in-topic are added to the *Word Collection*, and consequently take part in the keyword selection process.

We examine the following policies:

- ***Perfect:*** We use the categorization information directly from the site in search. Each document of the web sites that are used in our experimental evaluation is classified into topics, so this policy takes advantage of 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.

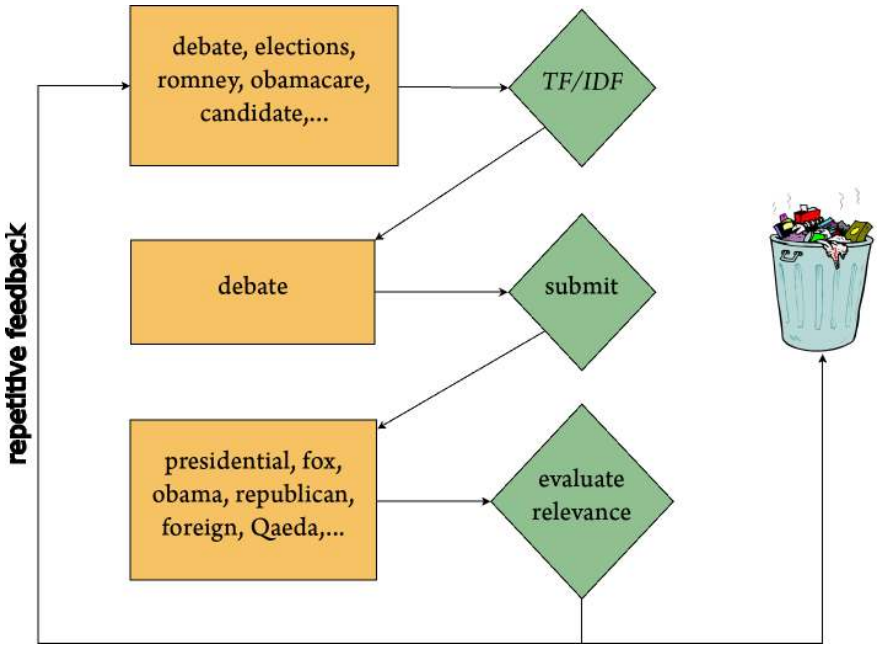

Fig. 4: The process of maintaining and enriching the *Word Collection*.

- ***Do-nothing:*** We accept all of the returned pages as in-topic. This policy updates the *Word Collection* with the contents of 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 which 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 content will find its way to the Word Collection and worsen the term selection process significantly. Thus, this policy can also be used as a comparison point for the other policies.
- ***Classifier based policies:*** This family of policies examines the effects the presence of classifier has to 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*. We examine the effect the presence of three different type of classifiers have:
  - ***NaiveBayes:*** The *NaiveBayes* classifier assumes that all attributes of a data set are independent of each other. This assumption, which in most cases is false, does not prevent the classifier from achieving high classification accuracy, while it is also ideal for domains with a

large number of attributes, since it simplifies the learning phase. Text classification is such a domain and simple Bayesian classifiers have been proven to be surprisingly successful when classifying text [10].
- ***SMO:*** Sequential Minimal Optimization (SMO) is an algorithm used for training Support Vector Machines (SVMs), whose solution involves a large quadratic programming (QP) optimization problem. SMO breaks this problem into a series of smallest QP problems which are solved analytically and manages to speed up and simplify the training of an SVM [22].
- ***J48:*** This policy creates a *C4.5* decision tree using *J48*, an open source Java implementation in the weka data mining tool [11]. Decision tree algorithms use a training set to build tree data structures that can be used to classify new cases. Each internal node of the tree structure contains a test based on one of the attributes of the set, the result of which determines which branch should be followed. *C4.5* provides high classification accuracy and outperforms other main-memory algorithms as far as speed is concerned [16].

- ***CosineSimilarity:*** We examine the cosine similarity of every result page with the initial exemplary document and accept only a small percentage of the closest ones. In that way, we ensure that the pool of words will be enriched only with terms closely related to the topic defined by our input. The cosine similarity is:

$$similarity(A, B) = cos(\theta) = \frac{A * B}{||A|| * ||B||}$$

where A and B are the *tf/idf* vectors of the two documents that are compared. This policy is clearly superior to the *NaiveBayes* policy in terms of adaptability, since it requires no training. However, since this method depends heavily on the query document, it is important to examine the effect of the quality of the latter in its decision making.

4 Experimental Evaluation

We start our experimental evaluation by presenting the datasets used in our experiments and the choices we made regarding the values of the crawler parameters. Then, we proceed with an extensive set of experiments on the performance of the proposed policies, discussed in Section 3.3. We first examine the effectiveness of the crawler using the total number of queries issued as a metric. Next, we consider the total pages downloaded during the crawling process, both relevant and irrelevant, as an evaluation criterion. Then, we investigate the behavior of the crawler when utilizing only specific parts of results to generate new queries. In Section 4.5, we present the actual queries issued during crawling and the precision achieved. After that, we present results for all of our policies for a variety of topics to evaluate them without domain bias. Next,

we test the effect the size of the "query document" has on the *CosineSimilarity* policy and examine if any improvement can be achieved by exploiting the *NOT* operator. Finally, we compare our results with that of a generic *Hidden Web* crawler.

4.1 Datasets used and calibration of key crawler parameters

In order to evaluate our method we used a variety of large human-generated datasets which are listed below:

- The Open Directory Project (*dmoz*) [3], a multilingual open content directory of World Wide Web links. The listings of the site are grouped into categories (which may include various subcategories) depending on their topic. *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). We considered the titles and summaries of each indexed link of the *dmoz* website as documents.
- The public non-beta *Stack Exchange* sites [4] contents: Stack Exchange is a group of sites covering many different fields that offer the opportunity to ask and answer questions related to the topic each site covers. 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. We considered the titles and questions of the *Stack Exchange* sites as documents.
- The New York Times Annotated Corpus [5] which contains over 1.8 million articles written and published by the New York Times between January 1, 1987 and June 19, 2007. These articles are manually summarized and tagged by library scientists. The tags applied concern the topic of the article - among other things - and use a controlled vocabulary that is applied consistently across articles. The titles of each article along with its synopsis was used in our experimental setup. Since each article may have more than one tags concerning its topic, we consider an article belongs to all of them. For this dataset we imposed a limit of 10,000 returned results per query.
- A collection of Geo-coded Tweets [6]. We used over 28 million of the most recent tweets of the dataset and since there was no topical taxonomy provided, we considered each tweet's country of origin as its topic. In order to enhance the process of discovering the country of origin and examine the

[3] `http://www.dmoz.org`

[4] `http://stackexchange.com/`

[5] The New York Times Annotated Corpus, Linguistic Data Consortium, Philadelphia, `http://catalog.ldc.upenn.edu/LDC2008T19`

[6] `http://istc-bigdata.org/index.php/our-research-data-sets/`

behavior of our crawler when utilizing structured information, we considered only terms included in the 'city' field of each document as potential submission terms. The tweet itself, its sender and its city of origin were considered as documents and we did not impose a limit on the return results per query.

For each data collection studied here, we used an initial exemplary document to serve as a representative of the topic in search. In the following, we used random documents from the respective topics to serve as "query documents". We do not report on how selecting a different initial document affects performance as we experimentally confirmed the findings of [21], that the selection of the first document does not significantly affect the returned results. However, we do study how the *size* of initial query documents affects performance in Section 4.7.

Furthermore, the presence of a text corpus is necessary, in order to perform the *tf/idf* measurements and extract the best keywords for each topic. For this purpose we used random documents from the four aforementioned datasets.

We used Apache Lucene [19] to index and query over the datasets with the *Standard Analyzer* and the default stop words filter.

For our experiments, we set variable N of Algorithm 1 to 7. That is, we update the *Word Collection* of Algorithm 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, after the submission of every query, for future use in the term-extraction process. We opted for this specific 1% in order to hold approximately the same number of returned documents as the other competing policies.

4.2 Using number of queries as a metric

In this section, we report results for the topic *Sports* of *dmoz* and for the topic *Wordpress* of *Stack Exchange* for which we measured the amount of individual queries it took to download pages from.

For the topic *Sports*, *dmoz* contains a total of $90,693$ pages. As Figure 5 shows, all six 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* started to utilize all the returned results for enrichment, since as the name of the policy implies, no action was taken to filter them. As a consequence, *Do-nothing* performed badly overall, as expected. Policies *Perfect* and *NaiveBayes* behaved quite 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 led to 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%

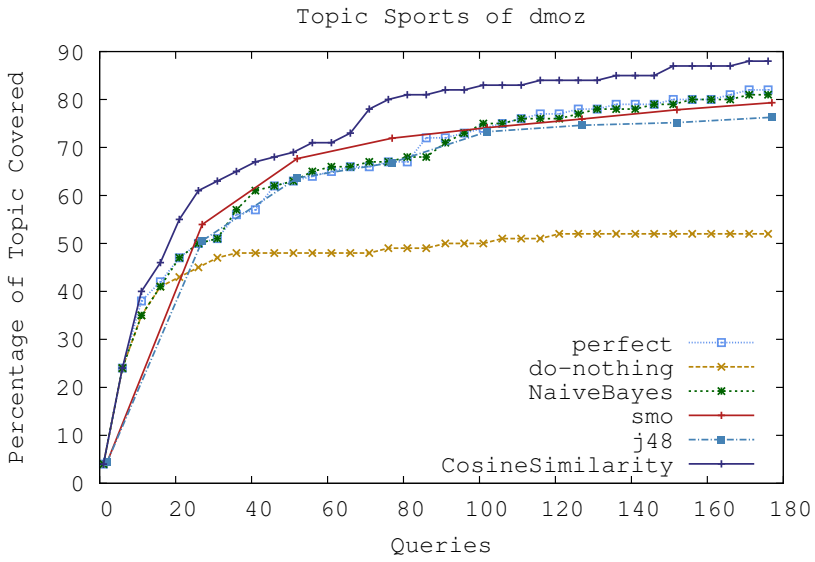

Fig. 5: Results of the six policies for topic Sports of *dmoz*

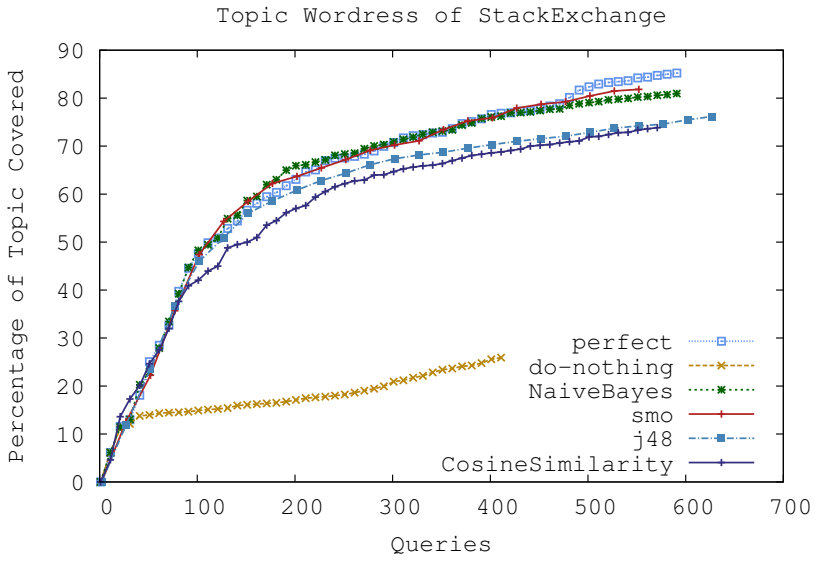

Fig. 6: Results of the six policies for topic Wordpress of *Stack Exchange*

of them. The other two classifier based policies, namely *smo* and *j48* performed slightly worse.

The *CosineSimilarity* policy managed to outperform all five other policies 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* did a better exploration of the keyword space compared to the other policies and it found keywords leading to more relevant documents earlier, compared to the *Perfect* policy which took 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 6. After 564 queries, the *Perfect* policy retrieved 85% of the total relevant documents, the *smo* 82%, the *NaiveBayes* 81%, and the *j48* and *CosineSimilarity* policies managed to collect 74% of them. The *Do-nothing* policy behaved well below those numbers again. 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 we enforced in this dataset.

4.3 Using pages downloaded as a metric

In this section we report results for the topic *US Election Campaign 2004* of the *NYT Corpus* for which we measured the amount of total pages downloaded, i.e., both relevant and irrelevant, during the crawling process. This topic contains a total of 36,388 articles that are related to the 2004 American Presidential Elections. The results are presented in Figure 7 where it is shown that retrieving 70% of the documents releated to the 2004 campaign required the download of 277,146, 462,262, and 473,195 pages for the policies *Perfect*, *CosineSimilarity* and *j48*, respectively. The other two classifier-based policies behaved a little worse with 493,649 pages for *smo* and 573,439 for *NaiveBayes*. *Do-Nothing* collected half of the relevant documents only after downloading 524,857 pages totally.

4.4 Utilizing specific attributes

Here, we report results for crawling over the *Geo-coded Tweets* with a query document consisted entirely of tweets from *France*. For this dataset, we followed an alternative approach and utilized only specific parts of the results, i.e., we enriched that *Word Collection* using the values of the attribute 'city'. The results are presented in Figure 8. We can see that after 350 queries all five policies that evaluate the results managed to retrieve between 74% and 77% of the relevant pages. The reason for almost identical behavior of our policies lies in the fact that only the part of the results concerning the city of the retrieved

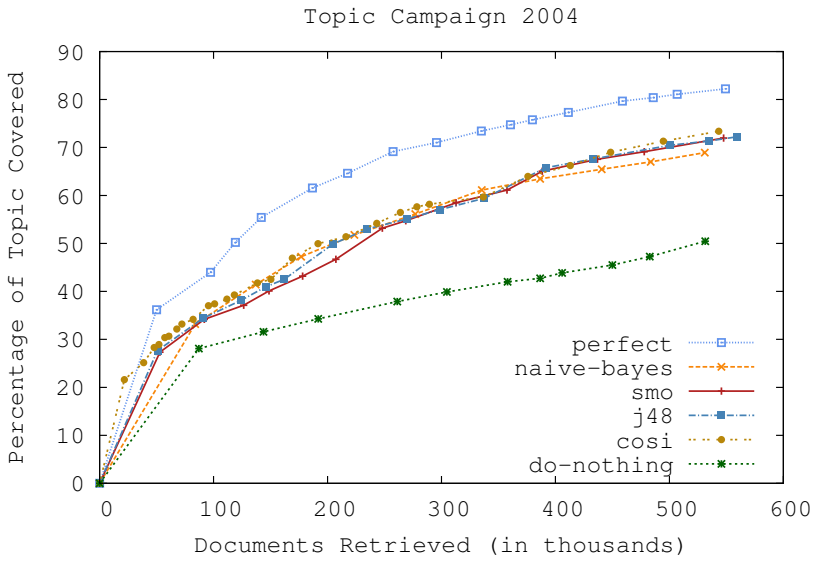

Fig. 7: Results of the six policies for topic Campaign-2004 of *NYT Corpus*

tweet was added to the *Word Collection*. This led to very similar sets of terms that were selected for all policies, which naturally resulted to practically exact performance.

4.5 Queries issued and topic precision

In order to investigate more closely the performance of our policies, we further examined the actual queries issued when using each one of them. We present a sample of queries for each policy together with the precision achieved, i.e., the fraction of in-topic documents,in Figure 9.

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 9 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 $7,530$, the *smo* policy fourth with $7,488$, the *CosineSimilarity* policy fifth with $7,426$, and the *j48* policy last with $7,401$ results per query.

Using the *dmoz* categorization information for every downloaded document, we also measured the precision of the different policies as shown in

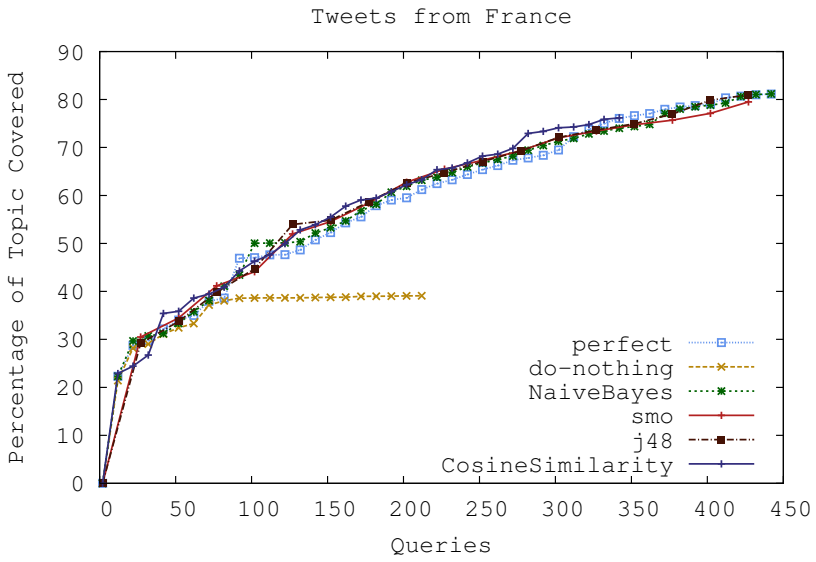

Fig. 8: Results of the six policies for country *France* of the *Geo-coded Tweets*

Figure 9. 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 *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.

The rest of the policies depend on the quality of the results that are used for the *Word Collection* enrichment, and thus the number of evaluation errors they commit. 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. The same stands for 48% of the ones added with the *j48* policy, and 31% of the ones added with the *smo* policy.

Finally, the *CosineSimilarity* used mostly documents belonging to topics different than *Sports* (72.9%). However, this did not affect the policy in a negative way. The retrieved documents, despite this fact, had very high cosine similarities with the query document, so naturally, the *Word Collection* was not altered in an undesirable way. As a result, the *CosineSimilarity* policy

No	Term	Precision
1	results	42.83%
2	statistics	43.61%
3	roster	71.36%
10	men	27.26%
15	scores	27.89%
20	players	30.62%
25	hockey	41.85%
30	tennis	7.43%
40	rugby	14.43%
60	sport	3.82%
100	competition	12.28%

(a) *Perfect*

Term	Precision
results	42.83%
statistics	43.61%
roster	71.36%
schedules	10.31%
church	0.00%
coaching	17.89%
methodist	0.00%
beliefs	10.11%
stellt	0.00%
bietet	0.00%
nach	0.00%

(b) *Do-nothing*

No	Term	Precision
1	results	42.83%
2	statistics	43.61%
3	roster	71.36%
10	schedules	10.31%
15	standings	67.96%
20	baseball	38.29%
25	records	8.38%
30	membership	1.52%
40	county	0.39%
60	fc	5.45%
100	standing	26.66%

(c) *NaiveBayes*

Term	Precision
results	42.83%
statistics	43.61%
roster	71.36%
basketball	46.36%
scores	43.09%
records	31.96%
field	38.15%
race	29.50%
squad	26.95%
swimming	20.31%
division	13.17%

(d) *smo*

No	Term	Precision
1	results	42.83%
2	statistics	43.61%
3	roster	71.36%
10	player	30.54%
15	coaching	21.86%
20	sports	17.59%
25	players	28.98%
30	calendar	23.13%
40	united	19.24%
60	junior	16.64%
100	standing	11.57%

(e) *j48*

Term	Precision
results	42.83%
statistics	43.61%
roster	71.36%
tables	5.61%
player	24.36%
players	33.70%
hockey	44.08%
baseball	32.20%
race	14.56%
conference	1.73%
competitive	11.61%

(f) *CosineSimilarity*

Fig. 9: Terms issued when using the different policies.

outperformed the other policies in terms of recall as we showed in the previous section.

4.6 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. This allows us to examine the performance of our crawler when retrieving documents belonging to both well and ill-defined topics of the same *Hidden Web* site. We used the following categories: Computers (103,336 documents), Recreation (91,931 documents), Shopping (87,507 documents), Society (218,857 documents) and Sports (90,639 documents).

Figure 10 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 topics in discussion using the *NaiveBayes* policy are presented in Figure 11. 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. The other two classifier-based approaches, namely *smo* and *j48*, produced slightly and significantly worse results than *NaiveBayes*, respectively. Figures 12 and 13 show their performance respectively.

Figure 14 depicts the 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. The *CosineSimilarity* policy performed slightly better than the *Perfect* policy in 4 of the 5 topics examined, *Computers* being the only exception. Additionally, it out-scored the classifier based policies 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.

4.7 Impact of query 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

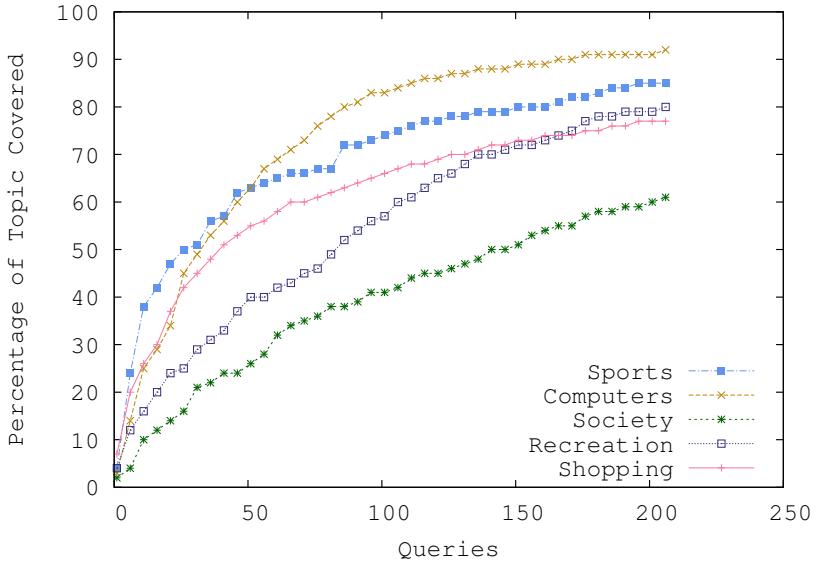

Fig. 10: *Perfect* policy on different topics of *dmoz*

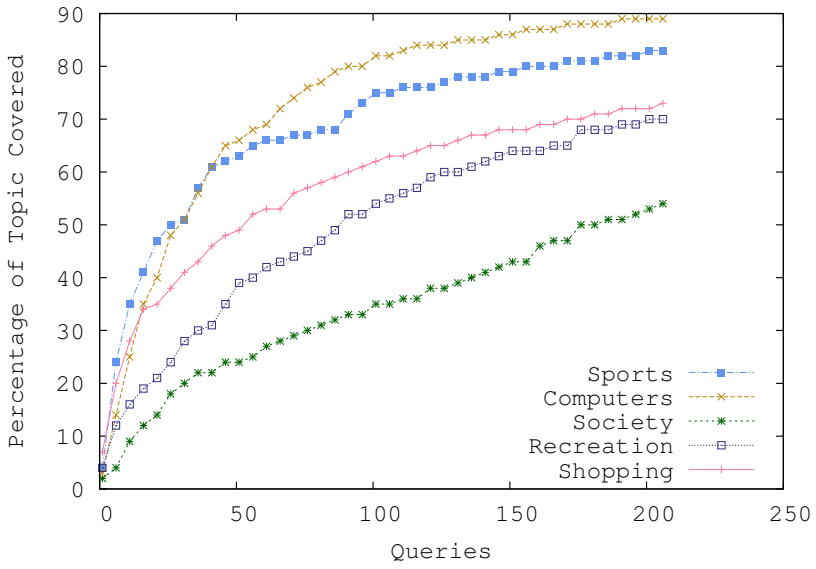

Fig. 11: *NaiveBayes* policy on different topics of *dmoz*

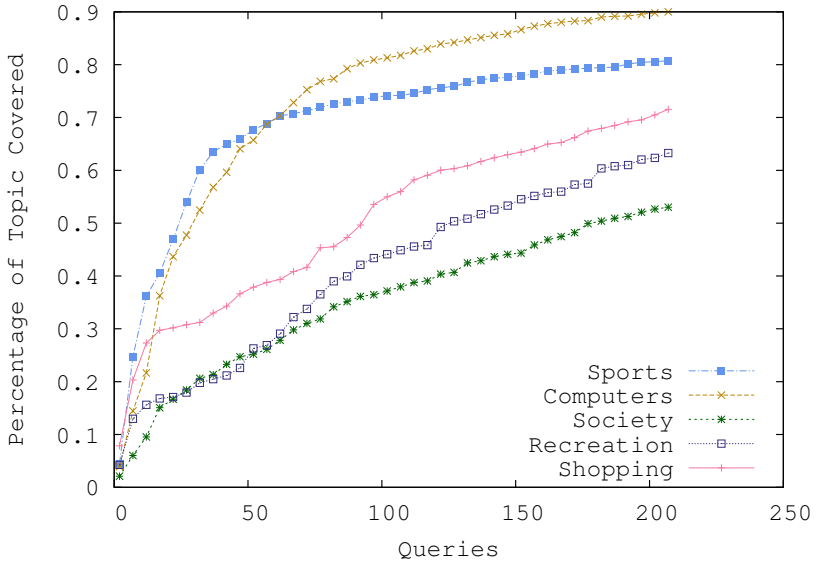

Fig. 12: *smo* policy on different topics of *dmoz*

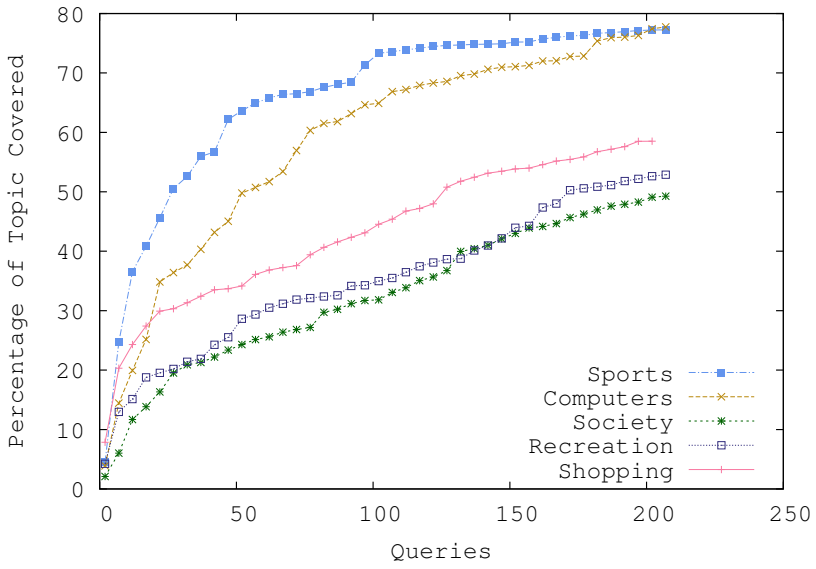

Fig. 13: *j48* policy on different topics of *dmoz*

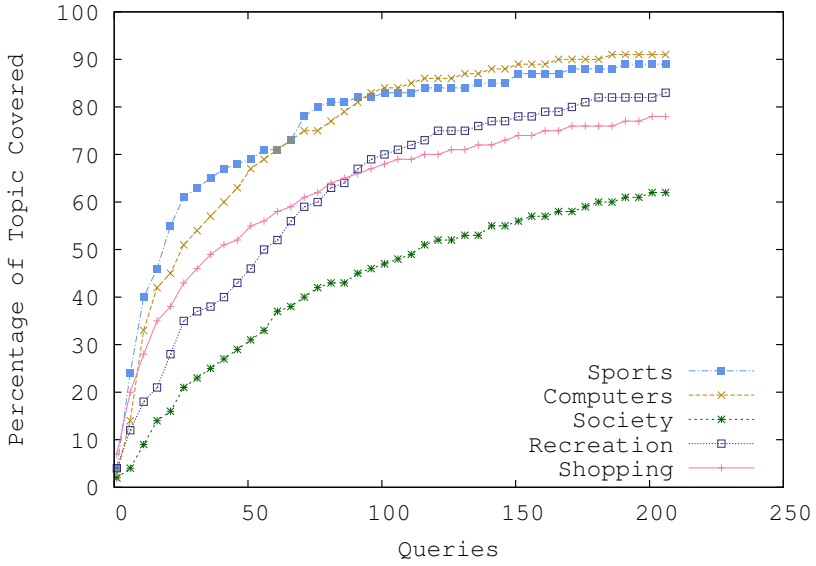

Fig. 14: *CosineSimilarity* policy on different topics of *dmoz*

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 15 illustrates the results for *CosineSimilarity* under three different sized "query documents", while crawling for *Computer* related documents from *dmoz*. We see that as the size limits the performance worsens. However, even for a very small "query document" we can still get very 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. Therefore, the *CosineSimilarity* policy behaves fairly good even with relatively small "query documents".

4.8 Impact of the NOT operator

The upper limit that *dmoz* enforces on the number of returned results plays a significant role in the performance of our approach. We can only retrieve a subset of the documents that a query matches and have to submit new queries to retrieve the rest of them. In order to deal with this issue, we examined the use of the *NOT* operator as part of our queries. For every term that has

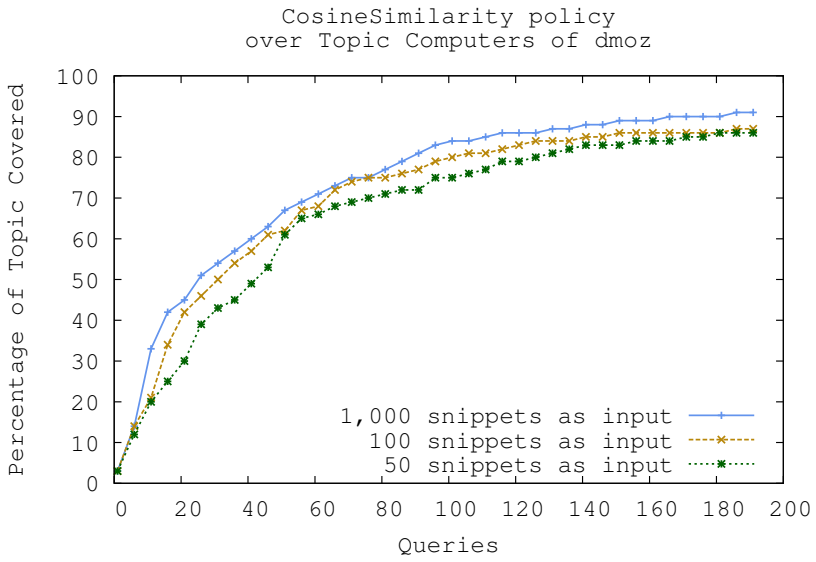

Fig. 15: Impact of the input document size using *CosineSimilarity* on topic *Computers*

returned a number of documents smaller than the upper limit, we can be sure that we have retrieved every document that contains it. Thus, by excluding this term from all future submissions, using the *NOT* operator, we can avoid downloading the same content and retrieve results that otherwise would be unreachable due to the maximum results limitation.

It is clear however, that we cannot apply the same policy for terms that have returned the maximum number of links, because in that way we would exclude from our search the rest of their matching documents. Since a lot of the terms our policies generate do actually reach that limit, it was expected that the impact of the *NOT* operator would not be significant. We tested this approach with the *Perfect* and *CosineSimilarity* policies for the topic *Sports* of *dmoz* and noticed little improvement for the former and very limited for the latter. The results are illustrated in Figure 16

4.9 A high-level comparison to generic Hidden Web crawling

In order to demonstrate the effectiveness of our approach, we compare our results with that of generic *Hidden Web crawler*. In Section 4.6, we presented the results our policies had over five different topics of *dmoz*. The *CosineSimilarity* policy, was able to download 70% of the content relevant to *Sports* and *Computers* after 52 and 60 submitted queries respectively. The rest of our policies behaved quite similarly. In [21], an experiment on crawling all of

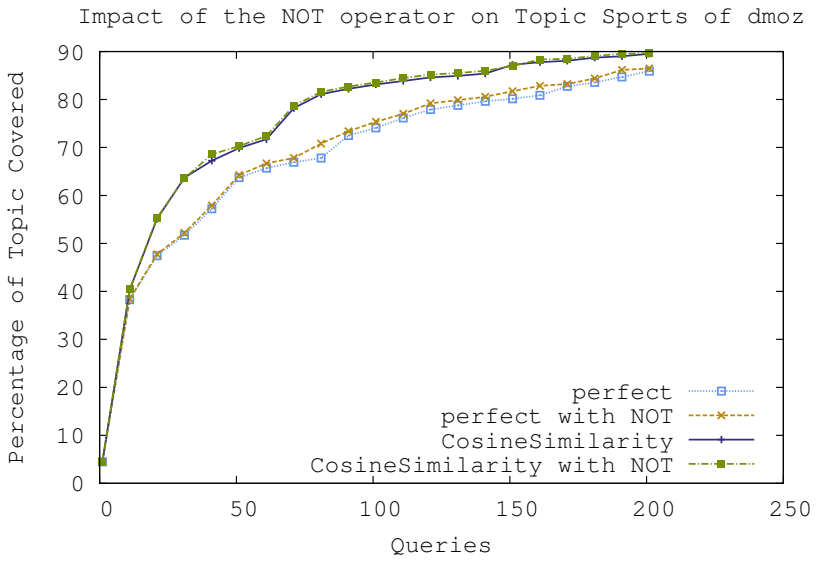

Fig. 16: Impact of the NOT operator using policies *Perfect* and *CosineSimilarity* on topic *Sports* of *dmoz*

the *dmoz* site's documents is presented. It takes about 700 queries to retrieve a little over 70% of its contents, and after that one would have to analyze the pages to identify which ones belong to the topic at hand. Therefore, it is clear that by generating those queries that are likely to retrieve pages that pertain to a certain topic, we managed to reduce the resources needed to crawl a *Hidden Web* site in a focused manner.

5 Conclusion and Future Work

We examined how we can build a focused *Hidden Web* crawler that, given a query document, can retrieve effectively those documents that are relevant to a certain topic. By avoiding the download of irrelevant pages, we limit the crawling requirements in terms of both hardware and network resources. Our approach uses the *tf/idf* weighting system to extract appropriate terms. We also proposed and evaluated a number of policies that measure the relevance of the returned documents with the topic in search. Our experimental evaluation indicates that our suggested algorithm has great potential for harnessing topic-specific documents. In the context of our work, we managed to successfully retrieve the majority of the documents related to a number of topics from four *Hidden Web* sites, by issuing a significantly smaller amount of queries than what it would be required to retrieve the site in its entirety.

In the future, we plan to examine if diverse query formulations will further reduce the overheads in the process. Moreover, we will continue with adjusting our approach to handle more complex query interfaces. Finally, another potential direction for future work is to build a crawler for the *Hidden Web* that focuses on downloading recently updated content. In that way, the crawler will achieve to further cut down its requirements in resources, since it will be able to avoid downloading the same documents over and over again.

Acknowledgements This work has been partially supported by *SocWeb*, *iMarine*, and *Sucre FP7 EU* projects. A preliminary version of the work appeared in the *Proc. of the 13th Int. Conf. on Web Information Systems Engineering* [15].

References

1. 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 the 3rd Int. Workshop on Data Enginering Issues in E-commerce and Services (EC)*, pages 18–25, San Diego, CA, June 2007.
2. L. Barbosa and J. Freire. Siphoning hidden-web data through keyword-based interfaces. In *SBBD*, pages 309–321, Distrito Federal, Brasil, October 2004.
3. L. Barbosa and J. Freire. Searching for hidden-web databases. In *Proc. of the 8th International WebDB*, pages 1–6, Baltimore, MD, June 2005.
4. L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *Proc. of the 16th Int. Conf. on World Wide Web (WWW)*, pages 441–450, Banff, Canada, May 2007.
5. A. Bergholz and B. Chidlovskii. Crawling for domain-specific hidden web resources. In *Proc. of the 4th Int. Conf. on Web Information Systems Engineering (WISE)*, pages 125–133, Roma, Italy, December 2003.
6. M. K. Bergman. The deep web. surfacing hidden value. *The Journal of Electronic Publishing*, 7(1):1–17, August 2001.
7. M. J. Cafarella, J. Madhavan, and A. Halevy. Web-scale extraction of structured data. *SIGMOD Rec.*, 37(4):55–61, Mar. 2009.
8. S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. In *Proc. of the 8th Int. Conf. on World Wide Web (WWW)*, pages 1623–1640, Toronto, Canada, May 1999.
9. M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB)*, pages 527–534, Cairo, Egypt, September 2000.
10. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, Nov. 1997.
11. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.
12. B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the deep web: A survey. *Communications of the ACM*, 50(5):94–101, 2007.
13. P. G. Ipeirotis and L. Gravano. Distributed search over the hidden web: hierarchical database sampling and selection. In *Proc. of the 28th Int. Conf. on Very Large Data Bases (VLDB)*, pages 394–405, Hong Kong, China, August 2002.
14. P. G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: categorizing hidden web databases. *SIGMOD Rec.*, 30:67–78, May 2001.
15. P. Liakos and A. Ntoulas. Topic-sensitive hidden-web crawling. In *Proc. of the 13th Int. Conf. on Web Information Systems Engineering (WISE)*, pages 538–551, Paphos, Cyprus, November 2012.

16. T.-S. Lim, W.-Y. Loh, and Y.-S. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Mach. Learn.*, 40(3):203–228, Sept. 2000.
17. J. Lu, Y. Wang, J. Liang, J. Chen, and J. Liu. An approach to deep web crawling by sampling. In *Proc. of the 2008 IEEE / WIC / ACM Int. Conf. on Web Intelligence, (WI)*, pages 718–724, Sydney, NSW, Australia, December 2008.
18. J. Madhavan, D. Ko, Ł. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's deep web crawl. *Proc. VLDB Endow.*, 1(2):1241–1252, Aug. 2008.
19. M. McCandless, E. Hatcher, and O. Gospodnetic. *Lucene in Action, Second Edition*. Manning Publications Co., Greenwich, CT, USA, 2010.
20. 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.
21. A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden web content through keyword queries. In *Proc. of the 5th ACM/IEEE-CS Joint Conf. on Digital libraries (JCDL)*, pages 100–109, Denver, CO, June 2005.
22. J. C. Platt. Advances in kernel methods. chapter Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.
23. S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB)*, pages 129–138, Roma, Italy, September 2001.
24. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, 1986.
25. P. Schonhofen. Identifying document topics using the wikipedia category network. In *Proc. of the 2006 IEEE/WIC/ACM Int. Conf. on Web Intelligence (WI)*, pages 456–462, Hong Kong, China, December 2006.
26. Y. Wang, J. Lu, and J. Chen. Crawling deep web using a new set covering algorithm. In *Proc. of the 5th Int. Conf. on Advanced Data Mining and Applications (ADMA)*, pages 326–337, Beijing, China, August 2009.
27. P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma. Query selection techniques for efficient crawling of structured web sources. In *Proc. of the 22nd Int. Conf. on Data Engineering (ICDE)*, pages 47–, Atlanta, GA, April 2006.
28. W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *Proc. of the 2004 ACM SIGMOD Int. Conf. on Management of data*, pages 95–106, Paris, France, June 2004.
29. Y. Yang, N. Bansal, W. Dakka, P. Ipeirotis, N. Koudas, and D. Papadias. Query by document. In *Proc. of the 2nd ACM Int. Conf. on Web Search and Data Mining (WSDM)*, pages 34–43, Barcelona, Spain, February 2009.
30. Z. Zhang, B. He, and K. C.-C. Chang. Understanding web query interfaces: Best-effort parsing with hidden syntax. In *Proc. of the 2004 ACM SIGMOD Int. Conf. on Management of Data*, pages 107–118, Paris, France, June 2004.