Focused Crawling for Structured Data

Robert Meusel
Data and Web Science Group
University of Mannheim
Germany
robert@informatik.uni-mannheim.de

Peter Mika
Yahoo Labs
Barcelona, Spain
pmika@yahoo-inc.com

Roi Blanco
Yahoo Labs
Barcelona, Spain
roi@yahoo-inc.com

ABSTRACT

The Web is rapidly transforming from a pure document collection to the largest connected public data space. Semantic annotations of web pages make it notably easier to extract and reuse data and are increasingly used by both search engines and social media sites to provide better search experiences through rich snippets, faceted search, task completion, etc. In our work, we study the novel problem of crawling structured data embedded inside HTML pages. We describe *Anthelion*, the first focused crawler addressing this task. We propose new methods of focused crawling specifically designed for collecting data-rich pages with greater efficiency. In particular, we propose a novel combination of *online learning* and *bandit-based explore/exploit* approaches to predict data-rich web pages based on the context of the page as well as using feedback from the extraction of metadata from previously seen pages. We show that these techniques significantly outperform state-of-the-art approaches for focused crawling, measured as the ratio of relevant pages and non-relevant pages collected within a given budget.

Categories and Subject Descriptors

H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval – *Selection process*, *Relevance feedback*; I.2.6 [**ARTIFICIAL INTELLIGENCE**]: Learning

1. INTRODUCTION

The adoption of markup languages for structured data has reached considerable levels in recent years due to the increasing support by large consumers and publishers of web content. In early 2010, Facebook announced the so-called Open Graph Protocol (OGP) for marking up the header of HTML pages with simple structured data such as the type of the page and a thumbnail image. Later in 2010, the major web search engines – Bing, Google, Yahoo, followed by Yandex – have joined forces to provide a common vocabulary for more complex markup, the so-called schema.org vocabulary, which allows complex descriptions of the most common types of objects appearing in web pages (videos, reviews, recipes, addresses, personal profiles, product descriptions, etc.) As of today, large search engines are exploiting this markup for providing richer snippets, vertical search experiences such as Google's Recipe Search[1] as well as support for task completion such as checking in to a flight directly from the search results page (e.g. Yandex Islands[2]). Social media sites such as Facebook, Twitter and Pinterest primarily use the structured data extracted from web pages for richer displays of the content that is being shared.

CIKM'14, November 3–7, 2014, Shanghai, China.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2598-1/14/11 ...$15.00.
http://dx.doi.org/10.1145/2661829.2661902.

Recent data from Google shows that over 15% of web pages are using schema.org markup, containing over 25 billion descriptions of objects [13] and the number of pages with OGP markup is stipulated to be even larger [4]. Large search engines collect this data during their regular crawls of the Web, while social media sites extract it at the time an item is first shared. These commercial web collections are not publicly available however, despite the multitude of potential applications of these data outside of web search and social media. For example, small e-commerce sites can benefit from additional information about the products they are selling, such as specifications from the manufacturer's website, ratings and reviews from social media, alternative offers from other vendors, etc. Entirely new vertical search engines can be built by having comprehensive, up-to-date information from across the Web about all items of a given type. At the same time, current web crawler implementations and publicly available web collections such as the CommonCrawl[3] data are not targeted at maximizing the amount and value of the structured data in their collections.

In this paper, we introduce the first focused crawler specifically aimed at crawling structured data. As the field of crawlers is well-established, we build on existing methods to the largest extent possible, but design our crawler to maximize the value of the data collected as opposed to maximizing the number of pages crawled. In particular we devise and implement new crawling policies that learn the characteristics of data rich pages and web sites, and steer the crawler toward them. To achieve this, we propose a novel combination of online classification and a bandit-based page selection. The former approach overcomes the problem of absence of a-priori knowledge about whether a newly discovered page contains structured data or not. With the latter approach we address the problem of *exploitation* versus *exploration*, allowing the crawler to perform random walks and fetch pages potentially better than those already discovered.

This paper thus makes the following contributions:

1. To the best of our knowledge, we are first to introduce the idea of a crawler focusing on semantic data, embedded in HTML pages using markup languages as microdata, microformats or RDFa.

[1] `http://www.google.com/insidesearch/features/recipes/`
[2] `http://beta.yandex.com/`
[3] `http://commoncrawl.org/`

2. We show that state-of-the-art online classification approaches employed for topical-focused crawling can be adapted for this goal, being able to gather over 10% more relevant pages within the same budget than approaches making use of pre-trained static classifiers.

3. We introduce a new approach to focused crawling using a bandit-based selection process to overcome the problem of *exploitation* versus *exploration*.

4. We combine online classification and a bandit-based selection and show that we can further improve on online classification by 26%, and improve on a standard Breadth-First Search policy by 131%.

5. We demonstrate that our method can also be adapted in order to crawl for more fine-grained embedded semantic data.

6. We make the implementation of our crawler publicly available.[4]

The paper is structured as follows: In the following section we give a brief introduction into web crawling and focused crawlers. We also discuss related work in the areas of online learning and bandit-based selection. In Section 3, we describe our methodology and the corresponding implementation. We then outline our experiments and we present the outcomes in Section 5. Lastly, we describe future ideas and open challenges.

2. RELATED WORK

2.1 Web Crawlers and Focused Crawlers

The purpose of web crawling is to gather a collection of useful web pages as quickly and efficiently as possible, while providing at least the required features for respecting the limitations imposed by publishers (*politeness*) and avoiding traps (*robustness*). Dhenakarn and Sambanthan [10] provide a brief overview about the four policies whose combination is influencing the behavior of a web crawler. We inherit from an existing crawler implementation to define the policies for *re-visits*, *parallelization*, and *politeness*. We will mainly focus on implementing a novel *selection policy*, i.e. determining the order in which new URLs are discovered and processed. The selection policy of web crawlers typically use variations of the PageRank [26] algorithm with the aim to collect the most popular pages within the Web, as they are also more likely to be searched for. Even though by definition all crawlers aim to build a collection useful for a given purpose, *focused crawlers* as described in the literature target pages relevant to a particular topic. Focused crawlers were first mentioned by Menczer [22] who modeled the problem inspired by work on agents adapting to different environments. Later, Chakrabarti *et al.* coined the term focused crawler and introduced an approach using a pre-trained classifier to assign topic-labels to new URLs based on features which could be extracted from the URL itself [7]. Other classification features have been obtained using different NLP techniques [16, 17, 19, 29]. Furthermore, Diligent *et al.* used information collected using web search engines in order to gather additional features for classification [11]. Aggarwal *et al.* incorporated information gathered during crawling to steer the direction of the crawler and maximize the number of retrieved relevant pages [1]. They use features extracted from the content of the father of the page (i.e. the page where we found the link), retrieving tokens from unseen URL strings and features collected from sibling pages (i.e. whose URLs were discovered in the same page as the one to be crawled). After crawling a page, the probability of the different feature groups for a given *topic* is evaluated and the combined probability is used to update the priorities of unseen pages. Although this model makes use of features gathered during the crawling process, the probabilistic model needs to be manually adjusted beforehand, which Chakrabati *et al.* try to overcome when first introducing an online classification approach for focused crawling [6]. Chakrabarti *et al.* crafted two classifiers, one static, pre-trained from an upfront collected and tagged corpus, and one online, which was used to improve former decisions based on features extracted from the document object model, e.g. the anchor text in links of crawled pages. Four years later, Barbosa and Freire took on the main idea of incorporating information gathered during crawling to steer the crawler with an extended feature set [2]. Besides the context of the page where a URL was found, they made use of the graph-structure of web pages, for example by distinguishing between direct features retrieved from the father and the siblings of the page, which was later also used by Zheng *et al.* [33]. Although they incorporate information gathered during crawling, they only replace their classifier with an updated version in batches, solely employing newly gathered information and discarding formerly extracted information. Their results indicate that sequentially updated classifiers lead to higher rates of gathering web forms for certain topical domains. Umbrich *et al.* proposed a pattern-based approach to classify pages, in order to find specific media types in the Web [32]. Jiang *et al.* [15] used a similar method to learn URL patterns that lead to relevant pages in web forums.

The main difference from this work with respect to mainstream focused crawling is that we are not aiming to perform topic-based classification, but rather looking at the value of web pages from the perspective of the data they contain. Web pages serving structured data have unique characteristics; structured markup is more common to particular types of pages, e.g. item detail pages, and favored by particular web sites, typically large dynamically generated sites serving certain types of content. Our target is also distinct from that of native *semantic web* crawlers that collect documents in RDF document formats, which follow *seeAlso* and *sameAs* references to related data items in order to discover new linked data sources and information. Two examples are *Slug* and *LDSpider* [12, 14]. These crawlers deal with the specific issues related to RDF data on the Web such as support for various native RDF formats, supporting various communication protocols etc. In contrast, our work focuses on structured data embedded inside HTML pages which has recently grown into a more popular way of exposing data on the Web. Recent studies have shown the increasing availability and diversity of data exposed this way [4, 24], offering a broad publicly available data source with large potential for various applications.

2.2 Online Learning and Explore/Exploit for Focused Crawling

State-of-the-art focussed crawlers partially make use of information gathered during crawling which is incorporated into the classification process in order to improve the accuracy of the prediction for unseen pages. In contrast to the aforementioned works, we propose an online learning method that continuously obtains feedback during crawling and incorporates it directly in an online classifier, rather than replacing the classifier from time to time. Such methods have been used before whenever data is available as stream [34] and the distribution of features within the data change over time [25].

[4] `http://webdatacommons.org/structureddata/anthelion`

Our underlying classification model will make use of all available feedback which can be successfully exploited for crawling for structured data, independently from its topic, and gather the largest number of relevant pages constrained to a given fetch budget.

Existing crawling policies implemented in the systems above focus largely on maximizing the immediate reward available to the crawler and lack in the discovery of new pages which potentially lead to more other relevant pages, but do not contain relevant information directly [9]. This problem can be described as the trade-off between *exploitation*, the crawling of pages where the expected value can be predicted with a high confidence and *exploration*, the search for new sources of relevant pages [21, 18, 28]. We address the issue of trading-off *exploitation* versus *exploration* by translating the problem of crawling into a bandit problem. We group newly discovered, not yet crawled pages by their corresponding host, each representing one bandit. During each iteration, where we want to select a new page to be crawled, we either select a page from a bandit, whose expected gain for a given objective function is maximal (exploitation) or select a page from a randomly chosen bandit (exploration). This approach was analyzed using synthetic data by Pandey *et al.* [27] and successfully applied by Li *et al.* in the context of news article recommendation [20]. To our knowledge, its value for focused crawling has not been established before.

3. METHODOLOGY

In the following, we are presenting the two general approaches to machine learning (online classification and bandit-based selection) that we adapt to the domain of focused crawling, and in particular to the task of collecting structured data from web pages.

3.1 Online Classification

Crawling pages that embody markup data can be cast as a focused crawling task, as their general aim is to devise an algorithm to gather as quickly as possible web pages relevant for a given objective function. Standard focused crawling approaches target pages that include information about a given topic, like sports, politics, events and so on. In our case, our primary objective function are pages which make use of specific markup standards, although there could be variants that narrow down this subset (see Section 5.3).

Focused crawlers make use of topic specific seeds and operate by training a classifier that is able to predict whether a newly discovered web page (before downloading and parsing its content) is relevant for the given target or not. Thus, it is mandatory to assemble a training set, find suitable topic seeds and learn a classifier *before* the crawling commences.

On the other hand, online learning approaches adapt the underlying model used for classification on the fly with new labeled examples. In the case of a crawler this would be suitable provided that it is possible to automatically acquire a label for a web page as soon as the content of the crawled page has been parsed. This approach is appealing because not only it is not necessary to create a training set in advanced but also the classifier adapts itself over time. In the case of the Web, where the distribution of single features is hard to predict it might happen that, while discovering larger amounts of pages the actual distribution differs strongly from the one of the training set. This adaptability is useful to ensure suitable classification results [25].

In order to predict the relevance of an unseen newly discovered page it is necessary to extract features which the classifier can take into account to make its prediction. We considered three major sources of features which are (partly) available for a web page before downloading and parsing it:

1. *the URL*, which can be handled using natural language processing (NLP) techniques to transform them into a feature vector.
2. *information coming from the parents of a page*, whose content has been already downloaded and the relevance for a given objective function is known.
3. *information coming from the siblings of a page*, meaning other pages which were found on the parent page, and whose relevance for a given target might already been known.

We note that these sources of features may become gradually available during the crawling process. We will always know the URL of candidate pages, but we might not have discovered *every* parent of a page; furthermore, information about siblings could not be available at all.

There are several possibilities to extract features from the URL of a page. In general the URL is split into tokens whenever there is a non alphanumeric character (punctuation, slashes and so on) and these tokens can be directly used as features of the page. In order to reduce the sparseness of the generated feature vectors, and potentially improve the accuracy of the classifier, it is possible to apply several pre-processing steps for the extracted tokens before finally transforming them into features. Among standard transformations, for example, we include removing tokens consisting of too few or too many characters. Another technique consists in mapping different spellings of a given token into its normalized version, or replacing tokens only made up by numbers with one static token representing *any* number.

Importantly, crawlers may not be aware of the range of different tokens (i.e., the dictionary) that can be extracted from the URL of newly discovered pages, which makes it difficult to use a predefined feature space for online learning. We overcome this problem by relying on the so-called *Hash-Trick* [30] and map all tokens into a fixed feature space.

This approach receives a list of pre-processed URL tokens previously split, $\{t\}$ and it creates a feature vector V with length k for the new page. First, it initializes every component with 0 values. Then, it maps each token t within the list to $x_t \in [0..(k-1)]$ using the hash-function described in Equation 1, where n is the number of characters of t, k is the number of selected hashes and $t[i]$ is the numeric value of the character at position i.[5] The corresponding position within the feature vector will then be updated $V(x_t) \leftarrow 1$.

$$x_t = \left| \left\lfloor \frac{\sum_{i=0}^{i<n} t[i] \cdot 31^{n-i+1}}{k} \right\rfloor \right| \qquad (1)$$

This way we can ensure that the number of features remains the same during the whole crawling process. Although the known drawback of hash-functions is the potential information loss whenever a collision happens, the described approach achieved good results in our case, when hashing tokens from URLs.

We also extract features from the parents and siblings of a page. These features are based on labels assigned to parent/sibling pages previously. For example, we introduce as a feature the number of parents/siblings labeled with the target class, and a binary feature representing the existence of at least one parent or sibling labeled with the target class.

The selection of features and classification algorithm will have a major influence on the final page selection performance. In order

[5]The numerator is equal to the *hashCode*-function implemented for string objects in Java.

Table 1: Results of feature and classification pre-experiments

classifier	attribute set	max accuracy	avg runtime per iteration (in ms)
HT	a	0.7656	54.1
HT	b	0.8165	1.2
HT	c	0.7431	56.3
NB	a	0.7146	4.0
NB	b	0.7710	0.9
NB	c	0.7147	2.0

to determine the most adequate combination of features, classifier and parameter configuration (number of hashes, classifier dependent settings, etc.), we ran a number of experiments on an independent development set. The dataset we compiled for these experiments was independently crawled and labels were acquired applying the method used in Section 4.2. The dataset consisted of $100K$ pages from over $1K$ different hosts with a balanced distribution of labels (same number of pages containing structured data and pages without any structured data). We randomly selected one page after the other, first letting the classifier predict the label and then training it with the real label.[6] We repeated this process ten times for each different configuration and measured both the overall accuracy of the classifier and the running time needed for classification and training of the whole dataset.

We experimented with two different online classification algorithms, namely Naive Bayes (also used by [6]) and Hoeffding Trees [34], and used three different major feature sets: (a) only tokens from the URL, (b) features from parents and (c) the combination of both. We engineered those features using a variety of configurations, for instance filtering by token length and replacing number tokens by the constant string *[NUMBER]*. Finally, we evaluated the performance of the classifiers with the tokens hashed into different number of features, ranging from $5K$ to $20K$.

Table 1 outlines the results of some of these experiments. We report the best performing configuration for each combination and omit the remaining results due to space limitations. In every case, ignoring tokens shorter than three characters and a replacement of numbers by a constant string worked the best. Hoeffding Trees (HT) performed overall better in comparison to Naive Bayes (NB) but needed up to 10-times more time to finish processing the whole dataset. In addition, we noticed that using $10K$ hashes produces the best results. Finally, it is worth remarking that adding sibling information into the feature set downgraded the performance consistently, so we excluded them in subsequent experiments.

In the following Section, we introduce the notion of bandit-based selection and explain how we combine online classification with the bandit-based approach.

3.2 Bandit Approach

A bandit-based selection approach estimates the relevance of a group of items for a given target, and performs this selection based on the expected gain (or relevance) of the groups. The bandit operates as follows: at each round t we have a set of actions $\mathcal{A}$ (or *arms*[7]), and we choose one of them $a_t \in \mathcal{A}$; then, we observe a reward $r_{a,t}$, and the goal is to find a policy for selecting actions such as the cumulative reward over time is maximized. The main idea is that the algorithm can improve its arm-selection strategy over time with every new observation. It is important to remark that the algorithm receives no feedback for unchosen arms $a \neq a_t$.

[6]This reflects the real operating mode of a crawler, except that a real crawler might have some delay in when feedback is available.

[7]Some bandits use *contextual* information as well [20].

An ideal bandit would like to maximize the expected reward $max_a\, E(r|a, \theta^*)$, where θ^* is the true (unknown) parameter. If we just want to maximize the immediate reward (exploitation) we need to choose an action to maximize $E(r|a) = \int E(r|a,\theta)p(\theta|D)d\theta$, where D is the past set of observations (a, r_a). However, in an exploration/exploitation setting we want to randomly select an action a according to its probability of being Bayes-optimal

$$\int \mathcal{I}\left[E(r|a,\theta) = \max_{a'} E(r|a',\theta)\right] p(\theta|D)d\theta \ , \qquad (2)$$

where $\mathcal{I}$ is the indicator function. In order to avoid computing this integral it suffices to draw a random parameter θ at each round t. One of the simplest and most straightforward algorithms is λ-greedy, where in each trial we first estimate the average payoff of each arm a. Then, with probability $1 - \lambda$, we choose the one with the highest payoff estimate $\hat{\theta}_{t,a}$ and with probability λ, we choose a random arm. In the limit, each arm will be tried infinitely often and the estimate $\hat{\theta}_{t,a}$ will converge to the true value θ_a. An adaptation of this straightforward algorithm is the usage of a decaying λ_t. This adaptation faces the problem of coming up with a large number of random selection when the estimated $\hat{\theta}_{t,a}$ is close to the true value θ_a. A decaying λ_t approaches 0 faster with each iteration. We will later employ a linear decaying factor, $\lambda_t = \lambda \cdot \frac{m}{t+m}$, where m is a constant.

In the case of our crawler, we will use our bandit-based approach to make a first selection of the host to be crawled. This is motivated by the observation that the decision to use structured data markup is performed at a host-level in most cases. Informally, we represent each host with a bandit that represents the value of all discovered pages belonging to this host. The available functions to calculate the score for a host and by this the estimated relevance for a target are diverse and described next. It is important to remark that selecting an arm (action) in this context would mean to select the host, which at a given point in time t has the highest expected value to include pages which are relevant for our target. Once we have selected the host, we follow by selecting a page from that host using the online classifier described in the previous Section.

Formally speaking, each host $h \in H^t$ represents one possible arm, which can potentially be selected by the bandit at an iteration t. Each h includes a list of all pages p belonging to this host. An action $a_t \in \mathcal{A}$ within our approach is then defined as the selection a host $h \in H^t$ based the estimated parameter θ_h at a given t and λ. In order to estimate θ_h for an arm, we can think about various different combination of available features. Next we will introduce the general approach and different functions to compute the score $s(h)$ using the following notation:

- $s(h)$ is defined as the score of the host h (or, in bandit notation, the expected reward $E(r|a, \theta^*)$).
- $C_{all,h}$ is the set of pages of h, which have already been crawled.
- $C_{good,h}$ (respectively $C_{bad,h}$) is the set of pages of h (not) belonging to the target class, which have already been crawled.
- R_h^t is the set of pages of h, which was already discovered but not yet crawled at iteration t. This means its part of the set of pages in the bandit representing h.
- $pred(p)$ is defined as the confidence value of p to belong to the target class, based on the used classification approach.

Our general approach is to group all newly discovered pages into the corresponding host. To select a new page, we first use the bandit algorithm to identify the host of the page selecting the one with the current highest score or one random (depending on the value of λ_t). From this selected host, we take the page with the highest confidence for the target class. This process is depicted in Algorithm 1. Note that the bandit is unable to use single pages as *arms*, given that we only need to retrieve them once and the feedback loop would be rendered useless. We also classify per-host pages to prioritize them after a host is selected for crawling. (A pure bandit-based approach would select a random page from within the host).

```
Data: Initial back-off probability $\lambda$, initial seed set $R_h$, decaying
      factor $m$
$\lambda_t \leftarrow \lambda, C_{bad,h} \leftarrow \emptyset, C_{good,h} \leftarrow \emptyset \; \forall h \in R_h$
for $t \leftarrow 1$ to $T$ do
    Draw uniformly a random number $n \in [0..1]$
    if $n > \lambda_t$ then
        for $h \in H^t$ do
            if $|R_h^t| > 0$ then
                Compute the score $s(h)$
            end
        end
        Select host $h = \mathrm{argmax}_{h \in H^t}\, s(h)$
    else
        Select a random host h where $|R_h^t| > 0$
    end
    $p \leftarrow h = \mathrm{argmax}_{p' \in R_h}\, pred(p')$
    crawl p and observe reward $r_{h,t}$
    if $r_h = 1$ then
        add $p$ to $C_{good,h}$
    else
        add $p$ to $C_{bad,h}$
    end
    update $H$ and $R_h$ with new $p^*$, $h$ retrieved from $p$
    for $\forall$ new $h$ do
        $C_{bad,h} \leftarrow \emptyset, C_{good,h} \leftarrow \emptyset$
    end
    $\lambda_t \leftarrow \lambda \cdot \frac{m}{t+m}$
end
```

Algorithm 1: Adapted general K-armed Bernoulli λ-greedy Bandit for focused crawling, with a linear decaying factor.

We now define several functions to compute the score $s(h)$.
Negative Absolute Bad function, where the score of a host is the negative number of already crawled pages not belonging to the target class of this host $s(h) = -|C_{bad,h}|$
Best Score function, where the score of a host is defined by the maximal confidence for the target class of one of its containing pages $s(h) = \max_{p \in h} \mathrm{pred}(p) \; \forall p \in R_h$
Success Rate function, where the score of a host is defined by the ratio between the number of pages crawled, belonging to the target class and those not belonging to this class. The ratio is initialized with prior parameters α and β which we set both to 1: $s(h) = (C_{good,h} + \alpha)/(C_{bad,h} + \beta)$.
Thompson Sampling function, where the score of a host is defined as a random number, drawn from a beta-distribution with prior parameters α and β. This function is based on the *K-armed Bernoulli bandit* approach introduced by Chapelle *et al.* [8] and described in algorithm 1. In this case we take as the score at iteration t the random draw $s(h)$ =Beta$(C_{good,h} + \alpha, C_{bad,h} + \beta)$. We initialized the prior with 1.
Absolute Good · **Best Score** function, where the score is the product of the absolute number of already crawled relevant pages: $|C_{good,h}|$ and the *best score function*.
Thompson Sampling · **Best Score** function, where the score is the product of the *thompson sampling function* and the *best score function*.
Success Rate · **Best Score** function, where the score is the product of the *success rate function* and the *best score function*.

Note that the reward depends on the target function of the bandit; in general we assign a positive reward only if the page crawled contains some form of markup data, but the process works similarly for other different objective functions (see Section 5.3).

4. EXPERIMENTAL SETUP

In this Section we describe the architecture and the process flow implementing the methodology discussed in Section 3.[8] Next, we introduce the dataset employed for our evaluation, and the different experiments performed.

4.1 System Architecture & Process Flow

We implemented our solution as part of a full-fledged web crawler, although our component is modular and can be integrated into other existing systems.

As input the application takes a queue of newly discovered, already filtered URLs[9] – named *input queue* Q_I. The output of the application is another queue where the URLs are ordered by the expected relevance for a given target – called *ready queue* Q_R.

URLs coming from Q_I are internally grouped by their host $h \in H$. Whenever a host h is selected, it is enqueued into the ready host queue Q_H. Note that Q_H can include the same h multiple times, whereas Q_I and Q_R consist of a list of unique p pages. Beside this, the application orchestrates several sub-processes:

- A *URL input handler* P_{input} that takes the next URL from Q_I and adds it into its corresponding host h.
- A *URL output handler* P_{output} that selects a URL from Q_H to be crawled, based on the targeting function and puts it into Q_R
- A *bandit-based host handler* P_{bandit} that selects the next h based on a given function and inserts it into Q_H.
- An *online classifier* $P_{classifier}$ that classifies new URLs based on a given set of parameters and the target function.[10]

Figure 1 illustrates the flow throughout our approach. The crawling process starts with a number of initial seed pages (0), which are fed into Q_I. Then, P_{input} pulls the first page p from Q_I. Before adding p into the corresponding h, the page is classified by $P_{classifier}$. In the online setting, $P_{classifier}$ starts off with an empty model as no training data (pages) are available so far. Whenever $|H| \neq 0$ and $\exists h \in H : |R_h^t| > 0$, P_{bandit} selects one host h based on the given $s(h)$ and λ (1). The selected h is inserted into Q_H and hosts in Q_H are processed by P_{output}. For each host, the URL with the highest confidence for the target class is selected and pushed into Q_R (2). The reordered pages are now ready to be handled by other components of the crawler. After downloading (3) and parsing (4) the page, the newly found links are added into Q_I (5). In addition, the label of the crawled pages is returned as feedback to $P_{classifier}$ which updates its classification model (6).

This component is fully distributed in nature, which in practice means that processes operate independently and some of them work faster than others. We optimized all the underlying processes in order to maximize the system throughput, this is, to minimize the probability that Q_R gets empty and the crawler has to wait for new

[8]The source code of the crawler is available at `http://webdatacommons.org/structureddata/anthelion`.
[9]By filtering we mean the removal of duplicate and unwanted pages (like certain file extensions like videos, images, etc.).
[10]We use the *MOA Java library* 2012.08 from `http://moa.cms.waikato.ac.nz` introduced by Bifet *et al.* [3].

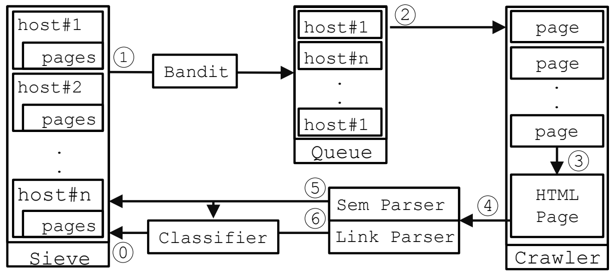

Figure 1: The architecture of Anthelion

pages. Additionally, we implemented a mechanism to delay the process P_{bandit} whenever the crawler is busy, as it might occur a slight delay in receiving the feedback for the action a_t when the system calculates the score for a_{t+1}.

4.2 Dataset

In line with the related work, we employ a static dataset for our experiments in order to isolate ourselves from changes in page content and the web graph, as well as other factors such as the availability of web page hosts. All the datasets we use in the following experiments have been extracted from the publicly accessible dataset provided by the Common Crawl Foundation. This dataset consists of over 3.8 billion web documents, where over 3.5 billion belong to the type *text/html* gathered in the first half of 2012 [31]. We used two derived sub-datasets of this original crawl to create the final datasets for our experiments: First, given that we require the structure of the crawled part of the WWW, we use the web graph dataset which was extracted by the WebDataCommons team, and it is described by Meusel *et al.* [23]. The data consists of 3.5 billion URLs with over 128 billion hyperlinks connecting them.[11] From this dataset we extracted a subset of around 5.5 million web pages which are reachable from one root URL. This was randomly selected from the URLs retrieved by crawling the pages of the *Open Directory Project*.[12] The dataset includes $455\,848$ different hosts.

Second, given that we also need to know which pages in our extracted subsets is relevant to our objective function, we use the *structured data set*, also extracted by the team of WebDataCommons from the Common Crawl data set. This dataset was created by parsing the HTML code of the crawled pages for the three markup standards Microdata, Microformats and RDFa using *Any23*[13] and includes, among others, the number of embedded structured data for each page [4]. From this dataset we extracted all web pages which are also present in our 5.5 million subset containing (a) at least one structured data statement and (b) more than four statements embedded using Microdata. From (a) we acquired 1.5 million pages, which comprise 27.4% of the whole 5.5 million sub-dataset. From (b) we acquired $179\,383$ pages, which is 3.25% of the whole 5.5 million sub-dataset.

With the subset and the structured information retrieved in (a) we will run most of the experiments to evaluate our approach for the general task of gathering efficient structured data from the Web. With the subset and the structured information retrieved in (b) we will run a secondary experiment and show that our approach is adaptable to different objectives in the area of structured data crawling.

[11]`http://webdatacommons.org/hyperlinkgraph`
[12]`http://dmoz.org`
[13]`http://any23.apache.org`

4.3 Experiments Description

Our first series of experiments aims to validate that our approach can effectively steer a crawling process toward web pages that embed any kind of structured data (Dataset a).

In the first step we compare our approach to a standard Breadth-First Search (BFS) approach and the typical approach to building focused crawlers for specific topics, i.e. using a static classifier. We run our implementation on the described dataset with a static classifier which we initially trained with $100K$, $250K$ and $1\,000K$ pages, and in comparison we run several crawls that incorporate online classifiers. In a second step we determine which scoring function for the hosts leads to the highest number of crawled pages that are relevant to our target function. We run several crawls incorporating different scoring functions for the bandit-selection, turning off the greedy component of the algorithm ($\lambda = 0$). Likewise, we selected the page with the highest confidence score from the bandit-chosen host. In a next step, we try different static values for λ to report the influence of the randomness for the best performing scoring functions. Additionally we will show the effect of a decaying λ_t using different decaying factors m.

In a second series of experiments, we change the objective for our crawling function. Therefore, we have defined relevant pages as those that embed structured data within its HTML code regardless of its kind and quantity. We now narrow down further this definition in order to measure the adaptability of our techniques to different objective functions. We want to reward only pages that embed at least five statements using the markup standard Microdata (Dataset b). Pages using Microdata typically use the schema.org vocabulary and provide more complex descriptions of the information present in the page. The number of statements is a rough quality criteria in that we filter out pages that provide only minimal detail. As an example, a movie page that contains at least five statements might include the facts that: (1) this page describes a movie, (2) the movie has the title *Se7en*, (3) the movie has a rating of 8.7, (4) was published in 1995 and (5) this information was maintained by *imdb.com*. Finally, we analyze the runtime of the different scoring functions. This is an important consideration because crawling is essentially a matter of resources, and it might happen that the crawler requires an unacceptably large time budget in order to select a new page being crawled. We are aware that this consideration depends on the crawling strategy and the implemented policies, which have been optimized consciously.

4.4 Metrics

The main objective of focused crawlers is to maximize the number of relevant pages gathered while minimizing the number of not relevant pages which are downloaded and parsed during the crawl. In order to evaluate the effectiveness of our approach, we use a *precision* measure that reports on the ratio of retrieved relevant pages to the total number of pages crawled. A page is considered to be relevant when it supports the objective crawling function, this is, whether the page contains structured data or not at all.

5. RESULTS

In the following we will present the results of the experiments described before. We used the same dataset (a) and the same initial seeds for each experiments for them to be comparable, except for Section 5.3 where we used dataset (b). In addition, as a large number of our experiments depends on sampling – especially those that test different bandit functions – we repeated each experiments up to five times and reported the average. The curves in the drawings within the result section are calculated using the smoothing spline method.

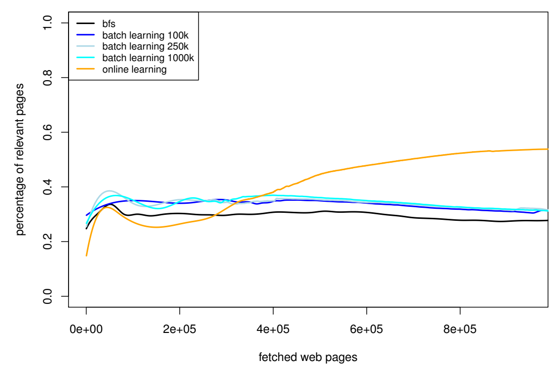

Figure 2: Percentage of relevant fetched pages during crawling comparing batch and online Naive Bayes classification

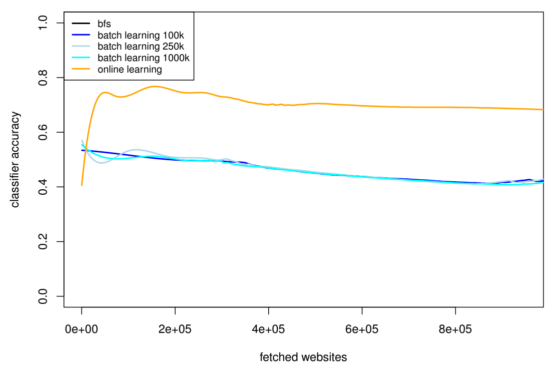

Figure 3: Development of the classification accuracy of batch and online Naive Bayes learning during crawling

5.1 Offline versus Online Classification

Static classification has been a dominant method for focused crawling. Our first set of experiments compared the performance of batch with online leaning in our domain of interest. We ran different crawls using pre-trained classification models learned on a subset of $100K$, $250K$ and $1\,000K$ randomly selected pages. Figure 2 shows the number of relevant retrieved pages of static approaches (blue lines). The orange lines show the ratio of relevant pages gathered by a crawler equipped with an adaptive online model which was trained completely from scratch during the crawling process. In addition we include the data series (black line) representing a pure breadth-first search approach (BFS).

The performance numbers of static-based classification are slightly higher than the number of BFS. Remarkably, online learning is able to increase notably the amount of relevant pages crawled after $400K$ fetches. At the end of the crawl, the adaptive approach is able to collect $539K$ relevant pages whereas the best static one (trained with $250K$ examples) fails to collect $200K$ of those. This trend is similar with Hoeffding Trees, although the difference in performance diminishes when the model is trained with $1M$ pages. Still, we note there is a decreasing performance rate for static classification approaches. The online learner also underperforms on the first half of the crawl. This is because the model is empty at the beginning and needs to be trained in subsequent iterations. On the other hand, static models have a slight edge at the beginning which is due to their knowledge advantage.

Figure 3 reports the accuracy over time of the classifiers present in Figure 2. The x-axis shows the number of fetched websites, whereas the y-axis describes the ratio of correctly classified to crawled pages. The saturated accuracy of the static classification approaches ranges between 0.55 and 0.45 where the adaptive model reaches 0.7 in the long run.

5.2 Evaluation of Different Bandit Functions

We now look into the interplay of bandit algorithms and the different functions to calculate the expected value of a host h (presented in Section 3). This first analysis will not include any randomness ($\lambda = 0$), to observe the real impact of the different setups, and compare them against a random selection, a BFS approach and the a pure online classification based selection (*best score function*).

Figure 4 shows the percentage of relevant pages retrieved during the crawling of one million pages. Firstly, all tested functions lead to higher number of retrieved relevant pages than the a pure random selection (black line) or a BFS (grey line). Furthermore, except for

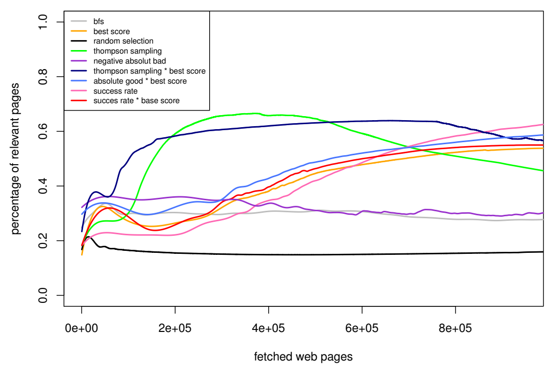

Figure 4: Percentage of relevant fetched pages during crawling comparing different bandit functions ($\lambda = 0$)

the Thompson Sampling based selection (TS) all the scoring functions outperform online classification on its own (and therefore using static classification approaches). The highest performance rate is achieved by the *success rate function*, which simply measures the ratio between relevant and non-relevant pages for a host. Here we are able to fetch around $628K$ relevant pages out of one million. The three combinations of *best score functions* with (a) TS, (b) *absolute good* and (c) *success rate* yield the second best results. Regarding the TS based functions, we see a sharp increase in relevant pages retrieved at early stages of the crawl. This decreases toward the end of the measurement series ending up gathering between $550K$ and $600K$ relevant pages. In comparison the other mentioned functions present a positive trend toward the end of the series.

Having identified the best performing scoring functions we now want to focus on the *explore / exploit* problem. We run the best performing bandit-based selection functions, namely *absolute good* in combination with the *best score*, the *success rate* and the combination of *success rate* and *best score* and measure the impact of different values λ. We tested the named functions, using different fixed values of λ (we report on the best ones) and compared to the corresponding gathering rate without any random selection. We did not consider the TS approach, as it already includes an element of randomness through sampling from a beta-distribution [8].

Figure 5 shows the impact of different λ values for our three selected functions. We can state that the usage of a random factor in the cases of the best score and the *absolute good function* fails to increase the number of crawled relevant pages. Regarding the functions that include *best score*, using a fixed λ greater than zero reduces the number of relevant pages.

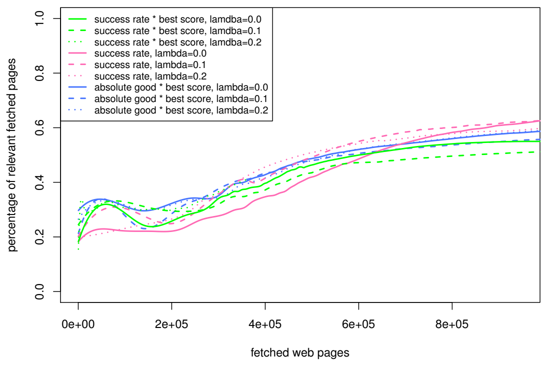

Figure 5: Percentage of relevant fetched pages during crawling pages comparing best performing bandit functions with different λ values

The above result may suggest that λ greater than zero may not be beneficial. Figure 6 zooms into the first $400K$ crawled pages and shows that there is a positive impact of including a random factor $\lambda > 0$, lifting the relevant page rate from 0.3 to 0.4. However, this effect diminishes when the amount of crawled pages reaches $1M$.

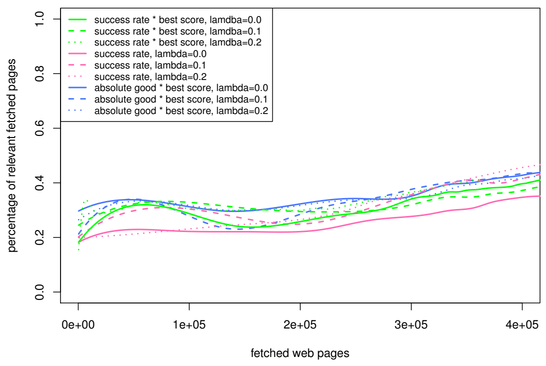

Figure 6: Percentage of relevant fetched pages during crawling of first $400K$ pages comparing best performing bandit functions with different λ values

The above results support our initial intuition that a decaying lambda may provide the best results overall. We now compare the performance of linear decaying functions for λ (described in section 3), with a fixed $m = 10K$ (value learned on an independent development set). Figure 7 shows the number of crawled relevant pages of the *success rate function* for the static and decaying λs. In addition to the already used $\lambda = 0.2$ we also show the results of a larger $\lambda = 0.5$ in order to increase the randomness and potentially learn more in the earlier stages of the crawl. Results show a positive impact of a decaying λ for the percentage of fetched relevant pages, achieving the maximum amount of relevant pages ($673K$). The positive effect is especially noticeable with $\lambda = 0.5$ – with no decaying factor, one out of two page selections are random (yielding the worst results), however when the decaying factor comes into play this negative effect disappears in the long run.

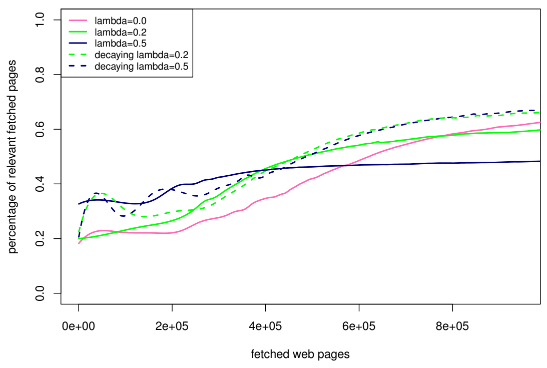

Figure 7: Percentage of relevant fetched pages during crawling comparing the success function with decaying and static λ

The results in this Section are summarized in Table 2. The results show a 10% improvement of the best performing method for online classification (Naive Bayes) over the best performing result for static classification (HoeffdingTree with $1\,000K$ training set) and a 26% improvement of the best combined bandit-based approach (Success Rate with decaying $\lambda = 0.5$) on top of online classification alone.

5.3 Adaptability to more Specific Structured Data Crawling Tasks

In this experiment we change the focus of our crawler and reward only pages with at least five statements using microdata.

We compare the BFS approach and the best score function with the best performing configuration from the former section: (1) *absolute good* $\cdot$ *best score* $\lambda = 0.0$, (2) *success rate* $\cdot$ *best score* $\lambda = 0.1$, (3) *success rate* $\lambda = 0.2$ and (4) *success rate* with decaying $\lambda_t = 0.5$ and $m = 10K$. Figure 8 shows the percentages of fetched relevant pages for the first one million crawled pages.

From the figure we can observe that all tested functions perform remarkably better than the BFS approach. The overall achieved rates are around five times smaller than the rates we reached for the more general objective function. However, now the amount of relevant pages among all the ones in the crawl is around eight times lower (0.04 vs. 0.27). In addition, after crawling one million pages, the bandit functions also outperform the online classification based selection strategy. Like in the previous experiment using a success rate based function tend to gather the highest number of relevant pages, with the *success rate function* with $\lambda = 0.2$ reaching a percentage of relevant crawled pages of 0.12 in the first million crawled pages. In comparison, online classification based selection ends up with a ratio of 0.08. Finally, in this experiment a decaying λ performed comparably to using a fixed λ value.

5.4 Runtime

In the previous experiments we have shown that the combination of online classification and a bandit-based approach leads to a higher percentage of relevant crawled pages for both tested objectives. We now assess what is the processing overhead incurred by our classification approaches and the current implementation for page selection. This time is critical, as when comparing to a BFS approach, we cannot venture to drop below the average processing

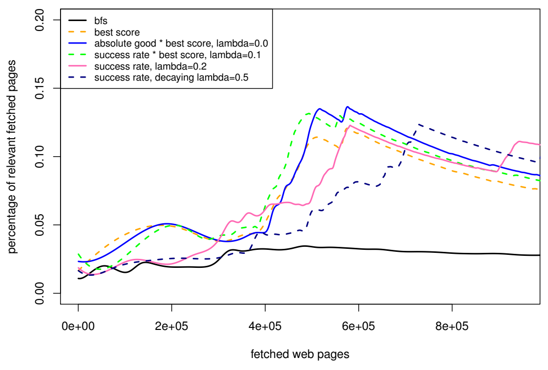

Figure 8: Percentage of relevant fetched pages during crawling aiming for pages with at least five Microdata statements.

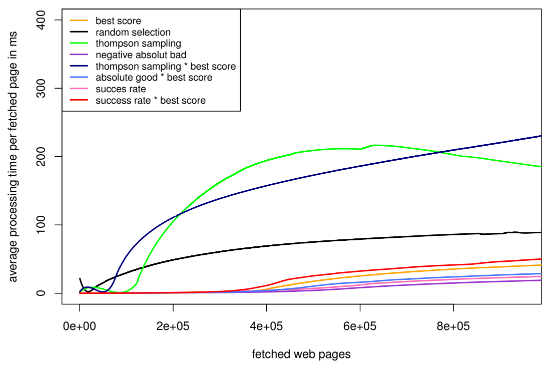

Figure 9: Average processing time to select one page over time

Table 2: Overview of percentage of crawled relevant pages after one million crawled pages

Selection Strategy	Percentage of Relevant Pages
Random	0.159
BFS	0.291
Naive Bayes (100K Training Set)	0.312
Naive Bayes (250K Training Set)	0.316
Naive Bayes (1 000K Training Set)	0.311
Naive Bayes (Online)	0.534
HoeffdingTree (100K Training Set)	0.408
HoeffdingTree (250K Training Set)	0.381
HoeffdingTree (1 000K Training Set)	0.482
HoeffdingTree (Online)	0.512
Thompson Sampling ($\lambda = 0.0$)	0.452
Thompson Sampling · Best Score ($\lambda = 0.0$)	0.562
Negative Absolute Bad ($\lambda = 0.0$)	0.300
Absolute Good · Best Score ($\lambda = 0.0$)	0.589
Success Rate ($\lambda = 0.0$)	0.628
Success Rate · Best Score ($\lambda = 0.0$)	0.550
Success Rate ($\lambda = 0.1$)	0.628
Success Rate ($\lambda = 0.2$)	0.600
Absolute Good · Best Score ($\lambda = 0.1$)	0.558
Absolute Good · Best Score ($\lambda = 0.2$)	0.590
Success Rate (decaying $\lambda_t = 0.2$)	0.662
Success Rate (decaying $\lambda_t = 0.5$)	0.673

time to crawl and parse a page as in this case the crawler process would have to wait for our selection.

The theoretical time which is needed to select one page for crawling is mainly influenced by four factors: (1) the number of hosts, as the bandit needs to go through all of them on each iteration, (2) the runtime of the scoring function for the hosts, (3) the selection of the final page from the selected host, which depends on the number of pages per crawl that are ready to crawl (4) the time to add the feedback to the system – including training the classifier and updating internal scores. In terms of a random selection (2) and (4) are omitted.

Figure 9 shows the average time in milliseconds for the bandit approaches presented before to determine the next page to be crawled. To make results comparable we also include the fully random selection approach. We can observe, that scoring functions not making use of the Thompson Sampling, where internally a beta-distribution needs to be calculated perform better than a pure random selection. The average time to selection one page range below 50ms for the dataset we used in our experiments. The two functions, making use of a beta-distribution need up to 300ms to select one page. Looking deeper into these functions we noticed that the creation of the beta-functions and the selection of the random value needs over 75% of the whole processing time.

In order to estimate the overhead of including our selection policies into a fully-fledged system one needs full measurements of the standard crawling cycle: establishing a connection, downloading the page, parsing and extracting new links. Taking a broad general estimate from a existing BFS crawler *Ubi-Crawler* [5], which needs $800ms$ to fully process one page per thread, our *selection* policy would incur in an overhead of less than 10%, as we need not more that $50ms$ for page selection. In comparison to this, we would boost the percentage of crawled relevant pages by factor three.

6. CONCLUSIONS AND FUTURE WORK

This paper introduces Anthelion - the first focused crawler targeting web pages containing markup standards for embedding structured data. Anthelion combines a bandit-based selection strategy for web pages with online classification to steer a web crawler towards relevant pages. The current implementation, which is publicly available, is designed to replace the *selection policy* of existing crawlers. We have shown that the use of online classification, in comparison to static classifiers, can achieve better results in this domain being able to collect over 10% higher numbers of relevant pages for a given objective function. Furthermore, our results show that grouping pages based on their host and making use of features shared by this group empowers the selection strategy for pages and improves considerably the resulting percentage of relevant crawled pages. We demonstrated that a bandit-based selection strategy, in combination with a decaying learning rate (decaying λ) overcomes the *explore/exploit* problem during the crawling process. Our results show that it is possible to increase the percentage of relevant crawled pages in comparison to a pure online classification-based approach by 26% (see Table 2).

Narrowing the focus of our crawler to web pages using Microdata to embed richer structured data (where we can extract at least 5 statements) we have shown that our approach can gather 66% more relevant pages within the first million than a pure online classification based approach. In general, estimating the expected value of an host using the *success rate function* in combination with always selecting the pages with the highest confidence for the target class tends to lead to the best results. Going beyond *precision* considerations, we have analyzed the runtime performance needed by the current implementation of our approach to select relevant pages for crawling and showed that we need, in average, 50ms to select a new page. The results presented in this paper demonstrate that a focused crawler using a bandit-based selection with online classification is capable of effectively gathering web pages embedding structured data. The two techniques we have described allow for potential improvements. Based on the idea of Lie *et al.* [20] we want to explore the usage of a contextual bandit approach where the results of one bandit can influence the gain of the other bandits. We could also consider extending the classifier to multi-class problems, which would account for graded relevance of crawled pages. We have shown that this approach can be adapted to more fine grained objective functions in this domain, although we want to face further objectives when gathering rich structured data using Microdata. A natural extension of our work would be to take into further consideration the quality of the data being crawled, e.g. considering the extent to which the data conforms to the pre-defined schema or to some model of the expected value of attributes. We could also enable our crawler to answer complex queries while crawling. For example, a used car dealer may be interested in only schema.org *Offer* instances that are about cars and relate to older models of certain brands, with a given price, etc. On the other side, we are also interested if our approach can help to increase the percentage of relevant pages in the *standard* focused crawling task, when searching topic related web pages.

7. REFERENCES

[1] C. C. Aggarwal, F. Al-Garawi, and P. S. Yu. Intelligent crawling on the world wide web with arbitrary predicates. In *Proc. WWW'01*, New York, NY, USA, 2001.

[2] L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *Proc. WWW'07*, New York, NY, USA, 2007.

[3] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 11, 2010.

[4] C. Bizer, K. Eckert, R. Meusel, H. Mühleisen, M. Schuhmacher, and J. Völker. Deployment of rdfa, microdata, and microformats on the web – a quantitative analysis. In *ISWC 2013*. Springer Berlin Heidelberg, 2013.

[5] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. UbiCrawler: a scalable fully distributed Web crawler. *Software-Practice and Experience*, 34(8), July 2004.

[6] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *Proc. WWW'02*, New York, NY, USA, 2002.

[7] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: New approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16), 1999.

[8] O. Chapelle and L. Li. An empirical evaluation of thompson sampling. In *Advances in Neural Information Processing Systems*, 2011.

[9] A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins. The discoverability of the web. In *Proc. WWW '07*. ACM, 2007.

[10] S. Dhenakaran and K. T. Sambanthan. Web crawler–an overview. *IJCSCN*, 2011.

[11] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, M. Gori, et al. Focused crawling using context graphs. In *VLDB*, 2000.

[12] L. Dodds. Slug: A semantic web crawler. *Jena User Conference 2006*, 2006.

[13] R. V. Guha. Schema.org update. `http://events.linkeddata.org/ldow2014/slides/ldow2014_keynote_guha_schema_org.pdf`, April 2014.

[14] R. Isele, J. Umbrich, C. Bizer, and A. Harth. LDSpider: An open-source crawling framework for the web of linked data. In *Proc. ISWC '10 –Posters and Demos*, 2010.

[15] J. Jiang, N. Yu, and C.-Y. Lin. Focus: Learning to crawl web forums. In *Proc. WWW'12*, New York, USA, 2012. ACM.

[16] M.-Y. Kan. Web page classification without the web page. In *Proc. WWW Alt. '04*, New York, NY, USA, 2004. ACM.

[17] M.-Y. Kan and H. O. N. Thi. Fast webpage classification using url features. In *Proc. CIKM '05*, 2005.

[18] G. Kane and M. Alavi. Information technology and organizational learning: An investigation of exploration and exploitation processes. *Orginazation Science*, 18(5), September-October 2007.

[19] T. Lei, R. Cai, J.-M. Yang, Y. Ke, X. Fan, and L. Z. 0001. A pattern tree-based approach to learning url normalization rules. *WWWW*, 2010.

[20] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. *CoRR*, abs/1003.0146, 2010.

[21] J. G. March. Exploration and exploitation in organizational learning. *Orginazation Science*, 2, March 1991.

[22] F. Menczer. ARACHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery. In *Proc. ICML'11*, San Francisco, CA, 1997. Morgan Kaufmann.

[23] R. Meusel, S. Vigna, O. Lehmberg, and C. Bizer. Graph structure in the web - revisited: a trick of the heavy tail. In *Proc. of WWW'14 companion*, pages 427–432. International World Wide Web Conferences Steering, 2014.

[24] P. Mika and T. Potter. Metadata statistics for a large web corpus. In *LDOW*, 2012.

[25] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, and F. Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 2012.

[26] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Tech. Report 1999-66, Stanford InfoLab, Nov 1999.

[27] S. Pandey, D. Chakrabarti, and D. Agarwal. Multi-armed bandit problems with dependent arms. In *ICML '07*, 2007.

[28] G. Pant, P. Srinivasan, F. Menczer, et al. Exploration versus exploitation in topic driven crawlers. In *WWW02 Workshop on Web Dynamics*. Citeseer, 2002.

[29] A. Puurula. Scalable text classification with sparse generative modeling. In *PRICAI 2012: Trends in Artificial Intelligence*. Springer, 2012.

[30] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan. Hash kernels for structured data. *J. Mach. Learn. Res.*, 10, Dec. 2009.

[31] S. Spiegler. Statistcs of the common crawl corpus 2012. Technical report, SwiftKey, June 2013.

[32] J. Umbrich, M. Karnstedt, and A. Harth. Fast and scalable pattern mining for media-type focused crawling. *KDML '09*, 2009.

[33] S. Zheng, P. Dmitriev, and C. L. Giles. Graph based crawler seed selection. In *Proc. WWW '09*, New York, USA, 2009.

[34] I. Zliobaite, A. Bifet, B. Pfahringer, and G. Holmes. Active learning with evolving streaming data. In *ECML/PKDD*, 2011.