DSpin: Detecting Automatically Spun Content on the Web

Qing Zhang
University of California, San Diego
qzhang@cs.ucsd.edu

David Y. Wang
University of California, San Diego
dywang@cs.ucsd.edu

Geoffrey M. Voelker
University of California, San Diego
voelker@cs.ucsd.edu

***Abstract*—Web spam is an abusive search engine optimization technique that artificially boosts the search result rank of pages promoted in the spam content. A popular form of Web spam today relies upon automated spinning to avoid duplicate detection. Spinning replaces words or phrases in an input article to create new versions with vaguely similar meaning but sufficiently different appearance to avoid plagiarism detectors. With just a few clicks, spammers can use automated tools to spin a selected article thousands of times and then use posting services via proxies to spam the spun content on hundreds of target sites. The goal of this paper is to develop effective techniques to detect automatically spun content on the Web. Our approach is directly tied to the underlying mechanism used by automated spinning tools: we use a technique based upon immutables, words or phrases that spinning tools do not modify when generating spun content. We implement this immutable method in a tool called DSpin, which identifies automatically spun Web articles. We then apply DSpin to two data sets of crawled articles to study the extent to which spammers use automated spinning to create and post content, as well as their spamming behavior.**

I. Introduction

Web sites fervently compete for traffic. Since many users visit sites as a result of searching, sites naturally compete for high rank in search results using a variety of search engine optimization (SEO) techniques believed to impact how search engines rank pages. While there are many valid, recommended methods for performing SEO, from improving content to improving performance, some "black hat" methods use abusive means to gain advantage. One increasingly popular black hat technique is generating and posting Web spam using *spinning*.

Spinning replaces words or restructures original content to create new versions with similar meaning but different appearance. In effect, spinning is yet another means for disguising plagiarized content as original and unique. However, the spun content itself does not have to be polished, just sufficiently different to evade detection as duplicate content. The most common use of spinning in SEO is to create many different versions of a single seed article, and to post those versions on multiple Web sites with links pointing to a site being promoted. The belief is that these "backlinks", as well as keywords, will increase the page rank of the promoted sites in Web search results, and consequently attract more traffic to the promoted sites. Search engines seek to identify duplicate pages with artificial backlinks to penalize them in the page rank calculation, but spinning evades detection by producing artificial content that masquerades as original.

There are two ways content can be spun. The first is to employ humans to spin the content, as exemplified by the many spinning jobs listed on pay-for-hire Web sites such as Fiverr and Freelancer [26]. Although articles spun by humans might have better quality, an alternative, cheaper approach is to use automated spinning tools. For example, a typical job on Freelancer might pay as much as $2–$8 per hour for manually spinning articles [13], whereas others advertise automatically spinning and posting 500 articles for $5 [12]. Or spammers can simply purchase and use the tools themselves using popular tools such as XRumer [30], SEnuke, and The Best Spinner. For example, The Best Spinner sells for $77 a year. These article spinners take an original article as input, and replace words and phrases in the article with synonyms to evade copy detection; some can even rearrange sentences. Spammers using automated spinning tools can select an existing article and spin it hundreds or thousands of times with just a few clicks, and then use posting services via proxies to spam the spun content on hundreds of target sites (also available for purchase) all over the Web. Automated spinning appears to be a popular option for spammers: in a set of $427,881$ pages from heavily-abused wiki sites, of the English content pages 52% of them were automatically-generated spun articles.

The goal of this paper is to develop effective techniques to detect automatically spun content on the Web. We consider the problem in the context of a search engine crawler. The input is a set of article pages crawled from various Web sites, and the output is a set of pages flagged as automatically spun content. Although not necessarily required operationally, we also use clustering to group together articles likely spun from the same original text. This clustering enables us to study the behavior of spammers on two crawled data sets as they post spun content with backlinks as part of SEO campaigns to promote Web sites. Our approach is directly tied to the underlying mechanism used by automated spinning tools: we use precisely what the tools use to evade duplicate detection as the basis for detecting their output. As a result, we also explore in detail the operation of a popular automated spinning tool.

In summary, we believe our work offers three contributions to the problem of detecting spun content:

NDSS '14, 23-26 February 2014, San Diego, CA, USA

http://dx.doi.org/doi-info-to-be-provided-later

Spinning characterization. We describe the operation of *The Best Spinner* (TBS), purportedly the most popular automated spinning tool in use today. TBS enables spammers to select an input article, specify various settings for spinning (e.g., the frequency at which TBS substitutes words in the input article), generate an arbitrary number of spun output articles, and optionally validate that the spun content does not trigger detection by defensive services like the *CopyScape* plagiarism checker.

Spun content detection. We propose and evaluate a technique for detecting automatically spun content based upon *immutables*, words or phrases that spinning tools do not modify when generating spun content. We identify immutables using the tools themselves. The heart of an automated spinning tool like TBS is a so-called *synonym dictionary*, a manually-crafted dictionary used to perform word substitutions. Since tools operate locally, each instance has a copy of the dictionary (and, indeed, downloads the latest version on start) and we reverse engineer TBS to understand how to access its synonym dictionary. When examining an article, we use this dictionary to partition the article text into mutables (words or phrases in the synonym dictionary) and immutables. When comparing two articles for similarity, we then compare them primarily based upon the immutables that they share.

Behavior of article spammers. We implement this immutable method in a tool called DSpin, which, given a synonym dictionary as a basis, identifies automatically spun Web articles. We then apply DSpin to two data sets of crawled articles to study the extent to which spammers use automated spinning to create and post content, as well as their spamming behavior: the number of spun articles generated in a campaign, the number of sites targeted, the topics of the spun content, and the sites being promoted. For valid pages from abused wiki sites, DSpin identifies 68% as SEO spam, 32% as exact duplicates and 36% as spun content.

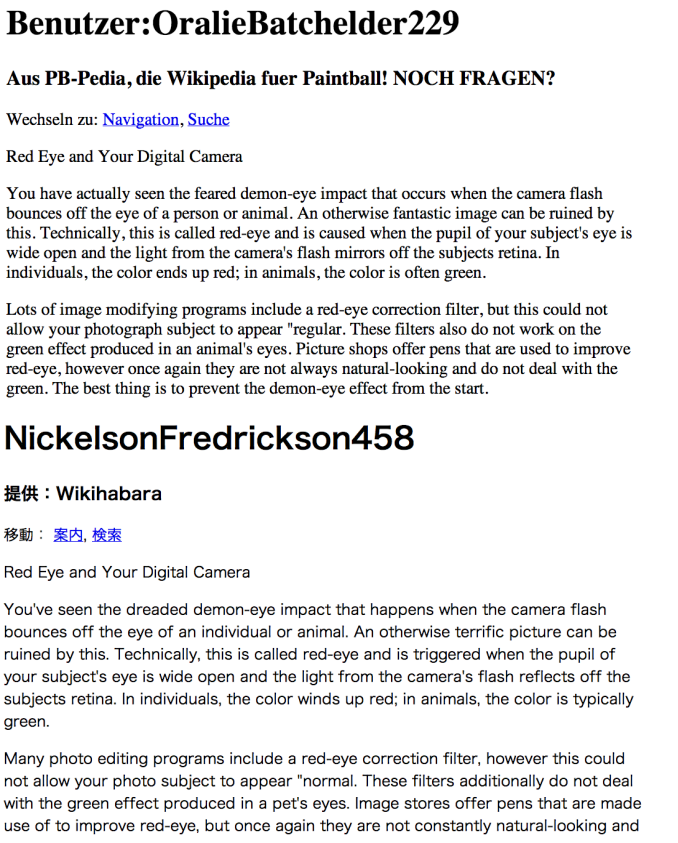
The remainder of this paper is structured as follows. Section II describes the role of spinning in Web spam SEO campaigns, and discusses related work in detecting Web spam and duplicate content on the Web. Section III describes the operation of the *The Best Spinner* and how we obtain its synonym dictionary. Section IV describes and evaluates a variety of similarity metrics for determining when two articles are spun from the same source, and motivates the development and implementation of the immutable method. In Section VI, we apply DSpin to examine spun content on two data sets of crawled articles from the Web. Section VII discusses how spammers might respond to the use of DSpin, and Section VIII concludes.

II. Background and Previous Work

As background, we first describe the role of spinning in black-hat SEO practices involving Web spam, and then discuss related work in both detecting Web spam and identifying near-duplicate content on the Web.

A. Spinning Overview

Search engine optimization (SEO) techniques seek to improve the page rank of a promoted Web site in search engine results, with the goal of increasing the traffic and



Benutzer:OralieBatchelder229

Aus PB-Pedia, die Wikipedia fuer Paintball! NOCH FRAGEN?

Wechseln zu: Navigation, Suche

Red Eye and Your Digital Camera

You have actually seen the feared demon-eye impact that occurs when the camera flash bounces off the eye of a person or animal. An otherwise fantastic image can be ruined by this. Technically, this is called red-eye and is caused when the pupil of your subject's eye is wide open and the light from the camera's flash mirrors off the subjects retina. In individuals, the color ends up red; in animals, the color is often green.

Lots of image modifying programs include a red-eye correction filter, but this could not allow your photograph subject to appear "regular. These filters also do not work on the green effect produced in an animal's eyes. Picture shops offer pens that are used to improve red-eye, however once again they are not always natural-looking and do not deal with the green. The best thing is to prevent the demon-eye effect from the start.

NickelsonFredrickson458

提供：Wikihabara

移動： 案内, 検索

Red Eye and Your Digital Camera

You've seen the dreaded demon-eye impact that happens when the camera flash bounces off the eye of an individual or animal. An otherwise terrific picture can be ruined by this. Technically, this is called red-eye and is triggered when the pupil of your subject's eye is wide open and the light from the camera's flash reflects off the subjects retina. In individuals, the color winds up red; in animals, the color is typically green.

Many photo editing programs include a red-eye correction filter, however this could not allow your photo subject to appear "normal. These filters additionally do not deal with the green effect produced in a pet's eyes. Image stores offer pens that are made use of to improve red-eye, but once again they are not constantly natural-looking and

Fig. 1. Example of articles spun from the same source and posted to different wiki sites as part of the same SEO campaign.

users visiting the site. There are many valid and officially recommended ways to improve search engine page rank by improving keywords, meta tags, site structure, site speed, etc. [16]. Indeed, an active service market of books, courses, companies, and conferences exists for optimizing one's Web presence. However, there is also a thriving underground industry that supports black-hat SEO, which can vary from violating recommended practices (e.g., keyword stuffing) to breaking laws (e.g., compromising Web sites to poison search results [22], [25], [35], [36]).

One popular abusive method of black-hat SEO is to post Web spam across many sites with "backlinks" to a promoted site. Such backlinks add perceived value when search engines calculate the page rank of a promoted site: conventional SEO wisdom holds that the more backlinks to a site, the higher its rank in search results. Search engines like Google have responded to such SEO techniques in updates to their page rank algorithm, such as Panda [32] and Penguin [4], which penalize pages with duplicate or manipulated content. Such algorithmic changes, however, rely on effective techniques to identify manipulated content. Thus, spammers have responded by making content manipulation harder to detect.

A popular form of Web spam today relies upon *spinning* to avoid duplicate detection. Spinning replaces words or phrases in an input article to create new versions with vaguely similar meaning but sufficiently different appearance to avoid plagiarism detectors. Figure 1 shows two examples of spun articles generated during the same SEO campaign that have

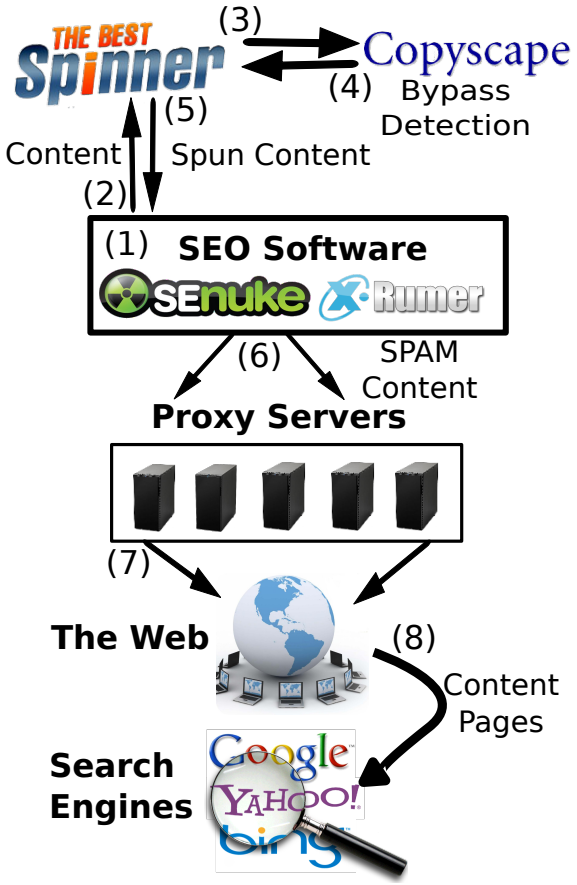


Fig. 2. The role of spinning in the workflow of abusive search engine optimization.

been posted to open wiki sites. We cut the pages short due to space constraints, but these pages both backlink to:

`http://evasivemosaic6837.wordpress.com/2012/11/28/how-to-locate-the-best-digital-camera/`

a site relating to cameras with links to adult webcam sites. Note that the spun content is in English, but has been posted to German and Japanese wikis.

Figure 2 shows the workflow of a spammer using spinning software to spam pages across the Web with backlinks to promote a particular site. The spammer uses an SEO software suite, such as *SEnuke* or *XRumer*, to orchestrate the SEO campaign. The SEO tool first helps automate the selection of original article content via an online directory of article collections, such as EzineArticles.com. Once the spammer has selected an article, often unrelated to the promoted site, the SEO tool sends the content to a content spinner such as *The Best Spinner*. The spinner changes words and phrases in the original article to generate repeated variations, and ensures that the spun content avoids triggering duplicate detection by submitting it to plagiarism services such as *CopyScape*. The spinner returns viable spun content back to the SEO software suite, which then spams target sites with the spun articles — typically via proxy services to obscure the spammer and to minimize the number of articles posted per IP address. Target lists and proxy services are heavily advertised by third-party sellers, and are easily integrated into the SEO tools.

As a result of this SEO campaign, search engine crawlers download the spun content across numerous sites. But, they cannot easily identify the spun articles as duplicate content since each article instance is sufficiently different from the others.

B. Article Spam Detection

Web spam taxonomies [17] typically distinguish between *content spam* and *link spam*, and article spinning can potentially fall in either.

The goal of content spam is to craft the content of a Web page to achieve high search engine rank for specific, targeted search terms. Techniques for detecting content spam pages include statistical methods incorporating features of the URL and host name, page properties, link structure, and revision behavior [14]; and more recently, statistical methods began incorporating features based upon a page's content, including page length, length of words, compressibility, phrase appearance likelihoods, etc. [28].

Additionally, other research focuses on techniques for detecting a specific form of *content spam* called "quilted pages", in which, the sentences and phrases that make up a page are stitched together from multiple sources on the Web [15] [27]. Fundamentally, these techniques work similarly — they begin with a large corpus of pages, split each page into overlapping n-grams, and detect a quilted page when a certain percentage of the n-grams from the candidate page are also found on other pages.

The *link spam* category distributes backlinks throughout spam pages to increase the page rank of a promoted site, rather than have the spam pages themselves appear higher in search results. Link spam has received substantial attention over the years. Since the primary value of link spam is the set of the backlinks they create, many approaches naturally focus on the link graph. Link farms, for example, form recognizable clusters of densely connected pages [37]. Other techniques such as TrustRank [18], ParentRank [38], and BadRank [21], [33] formalize a notion of Web page "reputation". Starting with training sets of good and bad pages, they propagate reputation scores along the link graph as part of the PageRank computation. Pages with resulting low reputation scores are considered spam and demoted in page rank.

Oftentimes, spun aritcles contain backlinks, which favor their classifcation as *link spam*. However, spun articles sometimes contain rich content that has been carefully modified from an original source, and so one might classify such articles as *content spam*. Given this dependence on the nature of the particular spinning campaign, classification should be made on a case-by-case basis.

C. Near-duplicate Document Detection

The description of Google's approach for near-duplicate document detection by Manku et al. [23] contains an excellent breakdown and discussion of the general body of work in the area. Generally speaking, the most common approaches for detecting near-duplicate documents on the Web use fingerprints to reduce storage and computation costs for performing what is naively an $n \times n$ comparison problem. The classic work is by

Broder *et al.* who developed an approach based on *shingles* [7]. Shingles are n-grams in a document hashed to provide a fixed storage size and to make comparisons and lookups efficient. Sketches are random selections of shingles that provide a fixed-size representation for each document. Similar documents share the majority of the shingles in their sketches with each other, enabling the identification of documents that share most of their content while allowing for minor differences. This approach enables a graph representation for similarity among pages, with pages as nodes and edges between two pages that share shingles above a threshold. The graph yields clusters of similar documents that either share an edge or have a path of edges that connect them.

Subsequent work has explored a variety of enhancements and variations to this baseline approach. Broder *et al.* in their original work proposed "super shingles", essentially shingles of shingles, to further condense and scale implementations. I-Match calculates inverse document frequencies for each word, and removes both the very infrequent and the very common words from all documents [10]. It then computes one hash for each remaining document, and those documents with identical hashes are considered duplicates of each other. Rather than choosing shingles randomly, SpotSigs [34] refines the selection of fingerprints to chains of words following "antecedents", natural textual anchors such as stop words. Two documents are then similar if their Jaccard similarity scores for their sets of word chains are above a configurable threshold.

With simhash, Charikar described a hashing technique for fingerprinting documents with the attractive property that hashes for similar documents differ by only a few bits [9]. Henzinger combined this technique with shingling and explored the effectiveness of such a hybrid approach on a very large Google corpus [19]. Subsequently, Manku *et al.* developed practical techniques to optimize the simhash approach with an operational context in mind, and demonstrated their effectiveness on the largest document corpus to date, Google's crawl of the Web [23].

While we use the fingerprinting work as a source of inspiration for DSpin, and borrow some implementation techniques (Section V-C), DSpin addresses a fundamentally different problem. These approaches identify near-duplicate content, which by design automated spinning tools specifically aim to avoid.

III. The Best Spinner

The goal of this work is to understand the current practices of state of the art software spinning tools as a basis for developing better detection techniques. This section examines the functionality of the most popular spinning suite, The Best Spinner (TBS), and leverages this examination to create a scalable technique for detecting spun content in the wild. In particular, we reverse engineer TBS to gain access to its synonym database, and later use the database to identify words likely unchanged by spinning software.

A. The Best Spinner

The first step of this study is to understand how articles are spun, and we start by examining how spinners work. There are multiple vendors online that provide spinning services. To select a service, we browsed underground SEO forums such as BlackHatWorld.com and selected The Best Spinner (TBS). The blackhat SEO forums frequently mention TBS as the de-facto spinning tool, and notably other popular SEO software such as SEnuke and XRumer [30] provide plugins for it.

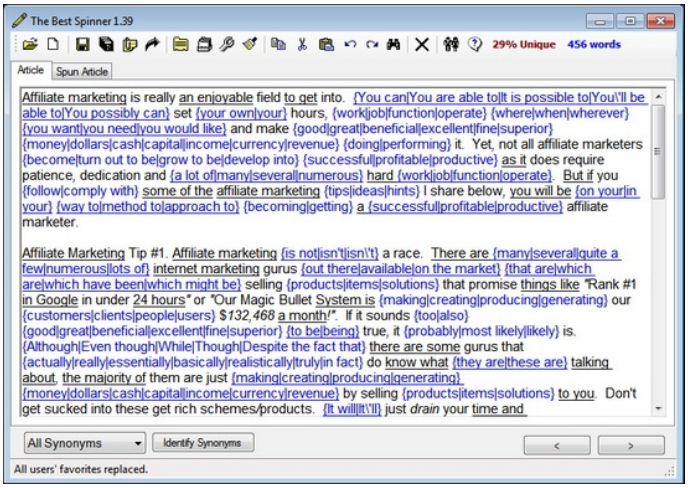


Fig. 3. The Best Spinner

We downloaded TBS for $77, which requires registration with a username and password. TBS requires credentials at runtime to allow the tool to download an updated version of a synonym dictionary. We installed TBS in a virtual machine running Windows 7. The application itself appears similar to any word processing application. Once the user loads a document, TBS generates a "spintax" for it. Spintax is a format where selective sets of words are grouped together with alternate words of similar meaning. During the actual spinning process, TBS replaces the original words with one of the alternates. Each set of words, including the original and synonym words, are enclosed in curly braces and separated by a "|".

TBS permits the user to adjust a number of parameters when generating the spintax:

Frequency: This parameter determines the spinning frequency. The options are every word, or one in every second, third, or fourth word. Typically, a lower number increases the frequency of replacements within the document; when selecting every third word, TBS tries to replace one in every three words (phrases can also be replaced, so the frequency is not exact). The manual and tutorial videos for TBS suggest that spammers should change at least one of every four words. The reason given is that popular duplicate detection tools, such as CopyScape, use a shingle size of at least four because shingle sizes of three or less have too many false positives. The TBS manual recommends setting the default parameter between every second and fourth word.

Remove original: This parameter removes the original word from the spintax alternatives. In effect, it ensures that TBS always chooses an alternate word to replace the original. For example, if the spintax for the word "Home" is:

$$\{Home|House|Residence|Household\}$$

then with *Remove Original* set it would be:

$$\{House|Residence|Household\}$$

Auto-select inside spun text: This is a check box parameter that, when selected, spins already spun text. This feature essentially implements nested spins, effectively increasing the potential set of word replacements.

In addition to these parameters, the user may also manually change the spintax by hand.

B. Reverse Engineering TBS

The core of TBS is its ability to replace words and phrases with synonyms. Since TBS selects synonyms from a custom synonym dictionary, the synonym dictionary is the foundation of article spinning. For this study, we obtained access to the dictionary by reverse engineering how the tool obtains it.

During every startup, TBS downloads the latest version of the synonym dictionary. We found that TBS saves it in the program directory as the file `tbssf.dat` in an encrypted format. By inspection of the encrypted database, we found that it is also obfuscated via base64 encoding. Since the TBS binary is a .NET executable, we were able to to reverse-engineer the binary into source using a .Net decompiler from Telerik; Figure 4 shows the portion of the code responsible for downloading and encrypting the synonym dictionary. It downloads the synonym dictionary using an authentication key, *GlobalVarsm._unique*, which is assigned at runtime during login by making the following request using the login credentials:

```
http://thebestspinner.com/?action=app_
login&email=email&password=password
```

Emulating TBS's sequence of steps, we queried the server to obtain the key, and directly downloaded the synonym dictionary using the key mimicking the behavior of TBS. We then `xor`ed it with the downloaded database, procuring the synonym dictionary in text format. As of August 2013, the decrypted synonym dictionary is 8.4 MB in size and has a total of 750,114 synonyms grouped into 92,386 lines. Each line begins with the word or phrase to replace, followed by a set of words of similar meaning separated by "|".

Note that the synonym dictionary does not have a one-to-one mapping of words. If word 'b' is a synonym of 'a', and word 'c' is a synonym of word 'b', there is no guarantee that word 'c' is in the synonym group of word 'a'. This artifact increases the difficulty of article matching in the following sense. If word a in article A is transformed into a_1 in article A_1 and a_2 in A_2, we are unable to compare a_1 and a_2 directly in the synonym dictionary; i.e., if we lookup a_1 in the synonym dictionary, a_2 is not guaranteed to be in the set of synonym of a_1.

C. Controlled Experiments

We now perform a controlled experiment to compare different similarity methods for detecting spun content.

To explore the effects of its various configuration parameters, we use TBS to generate spun articles under a variety of parameter settings. We downloaded an article from EzineArticles.com, a popular online article directory. The article consists of 482 words on the topic of mobile advertising. To exercise possible use case scenarios, we vary the spinning configurations of *Max synon* (3 and 10) and *Frequency* (1–4) during data set generation. We also toggle the *Auto-select inside* and *Remove original* parameters. Each configuration generates five spun articles in the test data set. We configure TBS to spin *Words and phrases* as opposed to *Phrases only* to maximize the variation between spun articles. We also add a control group to this data set where we randomly pick five different articles from EzineArticles unrelated to the spun article. As a baseline, the pairwise word overlap of the articles in this control set averages 26%.

Frequency	Max Synon 3	Max Synon 10	Max Synon 3 Auto-Select	Max Synon 3 Rm. Orig.
4th	84.0	79.0	83.0	78.0
3rd	79.0	73.0	76.0	70.0
every other	70.0	63.0	69.0	61.0
all	49.0	37.0	69.0	35.0

TABLE I. TABLE SUMMARIZING THE PERCENT OF OVERLAP BETWEEN THE ORIGINAL AND SPUN CONTENT.

To get a sense of the extent to which spinning modifies an article, we calculate the percentage of the article that remains unmodified for each configuration. We calculate this percentage by taking the number of words that overlap between the spun and original article, and dividing by the size of the spun article. We compute this ratio across all five spun articles for each configuration and report the average in Table I, leading to four observations. First, increasing the *Max Synon* parameter from three to ten causes more text (5–12%) to be spun. Second, the *Auto-Select* parameter has little impact on spun articles with minor changes for *Frequency* settings from 4th to "every other" with an average difference of 1.7%. However, when the *Auto-select* is set, there is no difference between setting *Frequency* from every other to all. Third, the *Remove original* option causes more text to be spun for all frequency settings ranging from 6–14%. Last, as expected, the *Frequency* parameter directly affects the amount of spun text, causing as much as 34% more text to be spun.

Using this training set, we next evaluate how well different algorithms perform.

IV. SIMILARITY

Determining whether two articles are related by spinning is a specialized form of document similarity. As with approaches for near-duplicate document detection, we compute a *similarity score* for each pair of articles. The unit of comparison differs depending on the technique. We first describe how it is defined generally, followed by the details of how it is defined for each technique. Table II summarizes the results.

A general comparison for the similarity of two sets, A and B, is defined by the classic Jaccard Coefficient:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

The straightforward application of the Jaccard Coefficient (JC) is to take all the words from the two documents, A and B, and

```
this.status.Text = "Downloading latest synonym file...";
Application.DoEvents();
stream = webClient.OpenRead(string.Concat("http://thebestspinner.com/?action=app_syn_db&uniq=", GlobalVars.m_uniq));
BinaryReader binaryReader = new BinaryReader(stream);
byte[] numArray = binaryReader.ReadBytes(9999999);
stream.Close();
string str = uTF8.GetString(numArray);
this.status.Text = "Unpacking...";
Application.DoEvents();
string str1 = str;
str = this.xorstring(ref str);
this.status.Text = "Parsing synonyms...";
Application.DoEvents();
this.ParseSyns(ref str);
str = GlobalVars.TripleDESEncode(str1, GlobalVars.m_key);
if (str.Length > 1000000)
{
    StreamWriter streamWriter = new StreamWriter(string.Concat(Path.GetTempPath(), "\\tbssf.dat"));
    streamWriter.Write(str);
    streamWriter.Close();
}
```

Fig. 4. Source code for downloading and encrypting the synonym dictionary in TBS.

compute the set intersection over the set union across all the words. Identical documents have a value of 1.0 and the closer this ratio is to 0, the more dissimilar A is from B. We base our definition for when two spun articles are similar on the Jaccard Coefficient. The ideal case is to have a technique that produces a Jaccard Coefficient as close to 1.0 as possible for two documents that are spun from the same source document, and to have a low Jaccard Coefficient for articles that are different.

A. Methods Explored

Given the Jaccard Coefficient metric, one needs to decide how to compute the intersection and size of two documents. There is significant degree of freedom in this process. For instance, one may choose to compare paragraphs, sentences, words, or even characters. Shingling and parts-of-speech represent two popular bases of comparison [11].

1) Shingling: Shingling is a popular method for identifying near duplicate pages. We implement shingling by computing shingles, or n-grams, over the entire text with a shingle size of four. We pick this size because the longer the n-gram, the more probable that it will be over-sensitive to small alterations as pointed out by [5], especially in the case of spun content. The shingling procedure operates as a sliding window such that the 4-gram shingle of a sentence "a b c d e f" is the set of three elements "a b c d", "b c d e", and "c d e f". Therefore, the unit of comparison for shingling is a four-word tuple. The metric for determining the Jaccard Coefficient with shingling is:

$$\frac{shingles_N(A) \cap shingles_N(B)}{shingles_N(A) \cup shingles_N(B)} \quad (2)$$

where the intersection is the overlap of shingles between two documents. As expected from Table II, shingling ranks two articles that are derived from the same input with a relatively low similarity between 21.1–60.7%. Since spinning replaces one out of every N words, as long as the frequency of word spinning occurs as often as the N-gram shingle then the intersection of shingles will be quite low. Although useful for document similarity, it is not useful for identifying spun content given the low similarity scores. (Plagiarism tools are believed to use some form of shingling, and spinning is designed to precisely defeat such tools.) Shingling ranks some spun content as low as non-spun articles in the control group.

2) Parts-of-speech: Parts-of-speech identifies spun content based on the article's parts-of-speech and sentence structure. The intuition is that if a substitution is made with a synonym, it would be substituted with the same part of speech. We implement this technique by identifying the parts-of-speech for every word in the document using the Stanford NLP package [2]. We pass the entire document, one sentence at a time, to the NLP parser. For each sentence, the NLP parser returns the original sentence with parts-of-speech tags for every word. We strip the original words, and use the parts-of-speech lists as the comparison unit. A document with N sentences therefore has N lists, each list containing parts-of-speech for the original sentence, and the corresponding Jaccard Coefficient is defined as:

$$\frac{pos(sentences(A)) \cap pos(sentences(B))}{pos(sentences(A)) \cup pos(sentences(B))} \quad (3)$$

When experimenting with articles spun with TBS, words are not necessarily being replaced with words of the same parts-of-speech. Furthermore, TBS can replace single words with phrases, and phrases comprised of multiple words can be spun into a single word. Table II reflects these observations, showing very low similarity scores ranging from 20.8% to 38.8%. Further, the similarity scores for spun content are nearly indistinguishable from the control group of different articles. Hence, we also do not consider this technique promising.

B. The Immutable Method

The key to the immutable method is to use the reverse-engineered synonym dictionary to aid in identifying spun content. In this method, we extract the text of a page and separate each article's words into those that appear in the synonym dictionary, *mutables*, and those that do not, *immutables*. We then focus entirely on the list of immutable words from two articles to determine if they are similar. Since the immutables are not spun, they serve as ideal anchors for detecting articles spun from the same input (as well as the input article itself). We exclude links in this analysis, as links can vary from one

	Max Synon 3				Max Synon 10				Max Synon 3 Auto-Select				Max Synon 3 Removed Original				Control
experiment	4th	3rd	other	all	4th	3rd	other	all	4th	3rd	other	all	4th	3rd	other	all	control
shingles	50.1	47.5	30.5	23.2	40.9	31.9	25.3	21.1	54.9	40.7	38.0	30.3	60.7	44.9	35.1	24.9	20.1
POS	31.5	34.1	24.0	22.1	24.3	22.9	22.3	21.0	36.5	25.1	30.9	24.0	38.8	26.3	24.2	20.8	20.1
immutables	93.5	92.4	94.6	80.2	97.7	92.5	86.6	78.3	96.6	94.4	93.4	93.4	96.6	94.5	90.8	74.9	27.8
mutables	90.4	90.0	86.0	82.3	86.0	83.0	79.3	77.0	91.5	89.0	88.7	87.6	93.0	89.3	87.6	82.6	59.4

TABLE II. AVERAGE MATCH RATE FOR SHINGLING, PARTS OF SPEECH, AND IMMUTABLES UNDER DIFFERENT SETTINGS FOR TBS.

spun version to another.[1] We treat each immutable as unique; a page has a set of unique immutables instead of a list of immutables. We differentiate between duplicate immutables by adding a suffix. With immutables, the Jaccard Coefficient is defined as:

$$\frac{immutables\,(A) \cap immutables\,(B)}{immutables\,(A) \cup immutables\,(B)} \tag{4}$$

Applying the immutable method to the training data set, Table II shows that using immutables to compute the Jaccard Coefficient results in ratios well above 90% for most spun content when using recommended spinning parameters. Under the most challenging parameters, spinning every word and/or removing the original mutable words, the immutable method still produces a similarity score as high as 74.9%. Furthermore, unlike the previous methods, it scores spun content with a high value while scoring articles that are different in the control group with a low coefficient of 27.8%. It thus provides a clear separation between spun and non-spun content.

The reason this technique does not produce a 100% Jaccard Coefficient is due to the behavior of spinning in which both words and phrases can be spun. We use a greedy implementation that scans the document from beginning to end. For every word, we see if the word is in the synonym dictionary, and if it is, we mark it as mutable. If not, we look up the word combined with zero to five subsequent words to see if the word or phrase is present in the synonym dictionary. Due to the greedy nature of this implementation, we may inadvertently mark mutable phrases as a series of immutable words. For example if {a,b,c} is a phrase, and both {a} and {a,b,c} are in the synonym dictionary, then we mark only {a} as mutable while marking {b} and {c} as immutable. However, Table II indicates that this greedy approach still produces very good results for detecting spun content.

In the control experiment, the synonym dictionary used to spin content and detect content are the same. In practice, when examining content on the Web, the synonym dictionaries may differ between the times of spun content generation and detection. To gauge the rate of change in the synonym dictionary over time for TBS, we downloaded two versions of the synonym dictionary 138 days apart and measured the overlap between the two. We found that 94% of the words in the synonym dictionary stayed the same, indicating that the degree of change in the dictionary is small.

One benefit of using the immutable method is that, in addition to its accuracy, it also greatly decreases the number of bytes needed for comparison by reducing the representation of each article by an order of magnitude. The average ratio of the number of immutables versus the number of total words in the original document is 6%. We disregard content that has one immutable or less, as one immutable would not provide enough confidence to determine whether two articles are related via spinning. The reduction in size of our data (Section V-A) is illustrated in Figure 5. The figure shows that more that 90% of pages we evaluate have an 80% reduction in the number of words that needs to be compared versus the original. More than 65% of the content has a 90% reduction. We find similar ratios in our *GoArticles* data set.

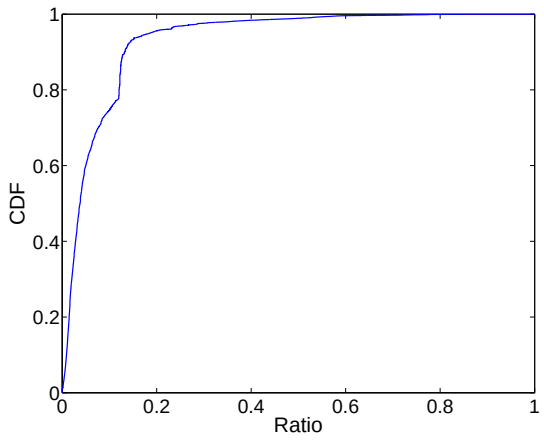


Fig. 5. Ratio of original document word count versus immutable word count

C. Verification Process

To further test the immutable method, we generated a 600-article test data set. We select five articles from five different popular article directories, and one directory containing five articles randomly selected from Google news. We spin each article 20 times using the bulk spin option in TBS. We selected the word replacement frequency of one out of every three words as suggested by the TBS spinning tutorial [3]. We apply the immutable method on this data set, and all the spun content is identified and correctly matched together with the original source.

Although the immutable method produces very accurate classification of spun content in our experimental data set, it is agnostic towards analyzing mutable content. Since the mutable words can account for 80% or more of the text on a page, ignoring them completely can cause false positives in cases where foreign text or symbols will bias two otherwise different pages to be identified as spun. To address this concern, we add another layer of verification to the immutable technique, which we call the mutable verifier. At a high level, the mutable

[1] Kolari *et al.* studied identifying blog spam via examining link based features [20].

verifier cross-references the mutable words, words that appear in the synonym dictionary, among two pages.

The mutable verifier computes the overlap of the mutables in the following steps:

- First, it sums all the words that are common between the two pages, and adds it to the total overlap count.
- Second, it compares the first level synonyms. It computes the synonyms of the remaining words from one page and determines if they match the words of the other page, and vice versa. Matches are added to the overlap count.
- Third, it computes the second level synonyms by taking the synonyms of the synonyms of the remaining words and comparing them in a similar fashion to step two.

As with immutables, the score the mutable verifier produces is the overlap over the union of the mutable words. We use a overlap threshold of 70%.

The mutable verifier rates spun content in our training data set as detailed in Table II. It produces a high rate for spun content with scores between 77% to 93% for spun pages, and a similarity score of 59.4% for non-spun pages. We do not employ the mutable verifier on the entire data set because it has a much higher overhead, as the algorithm compares all the words in the documents for two levels of synonyms. Instead, we rely on this to verify content which our immutable method identifies as spun to filter out false positives. Since the mutable verifier only runs in the verification phase, it only needs to take two documents at a time, enabling easy parallelization.

V. Methodology

Given a means of detecting the similarity of two potentially spun articles, we implement the immutable method in the Hadoop framework. We first discuss how we acquire two data sets from domains that are known to have spam. We then sanitize the data sets sanitized for the purpose or article comparison, removing foreign text, link spam, invisible text, and exact duplicates. Then we optimize duplicate detection to better scale to larger data set sizes.

A. Data Sets

We evaluate the immutable method on two data sets. The first is a set of crawled wiki pages actively targeted by spammers. The second is a popular article directory, GoArticles.com, which spammers use as both a source of articles for spinning as well as a target for spamming.

1) Spammed Wikis: The *wiki* data set is a collection of wiki article spam that we collected over a month between December 1, 2012 through December 31, 2012. The wikis themselves are benign, but their permissions leave them open for abuse by spammers. Figure 1 shows a typical example of spam posted as a wiki article.

To populate the *wiki* data set, we use a set of wikis discovered to have article spam. We identified this set of wikis by purchasing a Fiverr job offering to create 15K+ legitimate backlinks.[2] At the end of the job, the seller returned a list of URLs (not as legitimate as advertised) pointing to wiki article spam for inspection. From this list, we identified 797 distinct wikis apparently targeted by a wide range of spammers.

On an hourly basis, we then crawled the recent posts on each of the wikis, the majority of which are spam articles. Because the wikis all use the MediaWiki [24] platform, we use the following URL:

```
http://mediawikiexample/&Special:
RecentChanges&limit=500&hideminor=0
```

to fetch links for the 500 most-recent changes to the wiki. Note that we ignore any recent changes that do not occur within the hour to avoid fetching content that overlaps with previous crawls. We crawl 55K pages on average per hour, and in total we crawl 37M pages for December 2012.

2) GoArticles: The *GoArticles* data set is a collection of articles from GoArticles.com, a large article directory with over 4.6M articles. An article directory is a Web site in which users post articles on various topics, e.g., business, health and law. Some high quality article directories even pay authors for their contributions. In general, article directories forbid duplicate and spun content. The goal of the article directory is to attract readers and to make money from advertising. These directories enable users to submit unique articles and embed links to other relevant Web sites. Authors may use these pages to generate backlinks to their own personal sites.

We select this article directory for several reasons. First, GoArticles.com is one of the top search results returned by Google for the query "article directory". Furthermore, we observe that the site is targeted by members of the black hat SEO forums [1], who are lured by the fact that the site allows users to build backlinks as "dofollow" that can affect search engine page rankings. Often, sites that allow users to create backlinks in the form of HTML anchors can specify that all links created by users should be labeled "nofollow", indicating to search engines that they should disregard the link when ranking the linked page. The use of "nofollow" therefore acts as a deterrent to spamming a site with SEO backlinks. In the SEO vernacular, links not labeled as "nofollow" are considered "dofollow" and, as such, sites that allow them are highly prized by article spammers.

We populate the *GoArticles* data set by first enumerating 1M unique URLs pointing to articles on the site, and then crawling each article pointed to by the URL. To enumerate URLs, we take advantage of the URL format followed by the site, shown below, in which *title* refers to the title of the article and *id* refers to a seven-digit unique identifier for the article:

```
http://goarticles.com/<title>/<id>/
```

We found that the site ignores the *title* field and returns the article that matches the *id*. As a result, crawlers can fetch articles between *id* ranges while using random strings for the *title*. In addition, the site assigns *id*s to articles in chronological order. Thus, we can fetch all articles for a time period if we

[2]Fiverr is an online community where services are offered for hire at some predetermined cost.

know the corresponding *id* for the start and end of an interval of time.

Using this technique, we crawl over 1M articles over a month from the *id* range 6M–7M, which corresponds to articles posted between January 2012 to May 2013. We crawl at a rate of 2.7 URLs per second to limit the load on the site. This data set overlaps in time with the *wiki* data set that spans the month of January of 2013.

B. Filters

Before analyzing the posted articles for spun content, we apply several pre-filters to the data set. These pre-filters remove pages that lack substantive English text. These pages would likely be filtered during a search engine page rank algorithm. We apply each filter in sequence on both data sets to reduce the overall processing required.

Visible text: We remove all pages that do not contain any visible text on the page. We detect visibly blank pages by first stripping all HTML tags. If no visible text remains, we remove the page from the data set.

Content tag: The pages in each set share a common HTML tag that embeds the content of interest. For pages in the *Wiki* data set, the tag is a `div` labeled "bodyContent". For pages from *GoArticles*, the tag is a `div` with "class=article". If a crawled page lacks the appropriate tag, such as index page, we remove the page from the data set.

Word count: We discard small pages from analysis. Pages with a small number number of words often result in false positives with similarity algorithms. From manually examining article pages, we find it rare that spammers post spun article content using small articles. Unlike manual spinning, there is no incentive in automated spinning to favor small pages. Consequently, although small pages might contain spam, in our experience, little of that spam is generated using spinning tools. We find more short posts contain unintelligible words interlaced with lots of links. From experimentation, we find that a threshold of 50 words was a good balance between false positives and negatives.

Link density: We also discard pages with an unusually high link density. Again, we find that spammers using spun article pages typically do not litter the page with large numbers of links in the spun text, perhaps to avoid other detection algorithms that specifically look for such features. Thus we discard any page that has a link density of every fifth word or higher.

Foreign text: Since we reverse engineer only the English synonym dictionary from TBS, we only evaluate the immutable method on pages with mostly English text. (Otherwise, effectively every word is labeled as an immutable.) We use a language detector library [31] to identify the language of each page, and discard any page not identified as English. Since there is nothing inherent about the immutable method that ties it to English, we believe it would readily extend to other languages given access to synonym dictionaries for those languages.

C. Inverted Indexing

A naive pairwise comparison of two documents leads to $O(n^2)$ comparisons, which is infeasible for processing data at scale. Therefore, we implement the immutable method using inverted indexes, similar to the method described in [6]. For every immutable in the text, we generate a pair:

$$< id, immu > \tag{5}$$

The *id* is a unique index corresponding to an article, and *immu* is an immutable that occurs in *id*. We differentiate duplicate immutables by marking each with a unique suffix number to simplify document comparison. Next, we perform a "group by" on the immutables:

$$< immu, group < ids >> \tag{6}$$

Each group represents all document *id*s that contain the immutable. We decompose each group into a pairwise *id* key and a "1". Each pairwise *id*, $id_i : id_j$, indicates a single shared immutable between two documents. Each group with N *id*s therefore has $N^2/2$ pairs. The key-value pairs appear as:

$$< id_i : id_j, 1 > \tag{7}$$

Last, we "group by" $id_i : id_j$ and count the number of ones, yielding:

$$< id_i : id_j, count > \tag{8}$$

The count represents the total number of immutables that overlap between id_i and id_j. This format is also convenient for finding the total number of immutables in the original document. For a document id_i, the number of its immutables is given by $id_i : id_i$, namely the number of total overlapping immutables an article has with itself. From the list of pairs $< id_i : id_j, count >$, we calculate the *similarity score* between each two pages. We set the threshold for the *similarity score* to be 75%. We determine this threshold from the training data set, in which the lowest *similarity score* we found for any cluster was 74.9%.

D. Clustering

The id pairs can be transformed into clusters to convey more information. We cluster duplicate content, near duplicate content and spun content as detailed in Section VI both for filtering and also for assessing the behavior of spam campaigns.

To transform pairs into clusters, we use a graph representation where each page is a node and each pair has an edge in the graph. Each connected subgraph represents a cluster. We first build the graph using pairs or edges as input and the ids as nodes. For each node, we traverse all reachable nodes using breadth-first search and mark every node traversed as visited. We continually process unvisited nodes until every node has been visited. The results are disjoint clusters of ids.

E. Exact Duplicates and Near Duplicates

Advice posted in SEO forums advises against posting identical content with backlinks across multiple sites since search engines are capable of detecting such duplicate content [23]. In practice, we find that spammers continue to post identical content. By default, the various similarity algorithms discussed in Section IV would detect exact duplicates as spun content. Since our focus is spun content, we separate exact duplicates from the analysis of spun content in experiments.

We identify exact duplicates by generating a hash over each page in the form of an MD5 sum: two articles are identical if their MD5 sums match. We add this sum to the immutables list for each page. When analyzing a two-page pair, we first determine if the two pages are identical. This is done by determining if their *similarity score* is 100%; the MD5 sums must match to obtain this value, since the immutables include this hash. Anything less than 100% implies that the pages are not exact duplicates. If the pages are not exact duplicates, we adjust the *similarity score* calculation to account for the extra MD5 sum by subtracting two from the denominator, and recompute the *similarity score* using this modified version. This method gives us the same *similarity score* as if the hash sums are not added.

We also attempt to identify near duplicates and to separate them from our spinning analysis. We achieve this using the mutable verifier. In the results, we find articles with a 100% mutable match, but with mismatching MD5 sums. Manual examination of examples shows that these are near duplicate pages where the text on the two pages are identical, with minor differences in the links or spacing, which we characterize as near duplicates in our data set. Recall in Table II, no spun page has a mutable similarity score of 100% match. Thus we separate these pages from the spun page analysis as well.

F. Hardware

We run our experiments on a shared cluster with 24 physical nodes running Fedora Core 14. Each node has a single Xeon X3470 Quad-Core 2.93GHz CPU and 24 GB of memory. The experiments are written in Java, and run as a combination of Hadoop 1.1.2 and Pig 0.11.1 jobs.

VI. Spinning in the Wild

We next use DSpin to identify spun content in the two crawled data sets. We examine the amount of spun content, characteristics of the clusters of spun pages including size, content, and number of sites targeted, and finally look at the relationship between the wiki sites and the article directory for spinning and spamming.

A. Volume

First, we report the total volume of spun content found using the immutable method for both the *wiki* and *GoArticles* data sets. For each data set, we apply filters that, in sequence, remove articles that have no visible text (visible), no expected content tag (body), insufficient word count (wc), too many links (link), or a language other than English (english) as discussed in Section V-B. After applying filters, we obtain all page pairs that match according to the immutable method.

We divide these pairs into exact and non-exact duplicates. We form clusters of the exact duplicates, and use the smallest alpha-numeric ID to represent each duplicate cluster. Next, we cluster all non-duplicate pairs. During this clustering phase, we remove all IDs found in the duplicate clusters with the exception of the smallest alpha-numeric ID.

After removing exact duplicates, we validate each edge (an edge represents a match between two pages) with the mutable verifier discussed in Section IV-C. The mutable verification process removes unmatched pairs, identifies near-duplicate pairs, and removes them as well. Figure 6 shows the size of the data set after each step of filtering for the *wiki* and *GoArticles* data sets, as well as applying the mutable filter. Finally, the last column reports the total number of spun pages found.

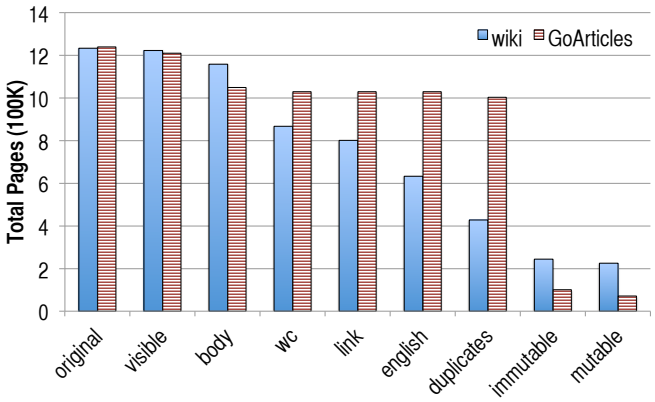


Fig. 6. Effect of applying filters on the *wiki* and *GoArticles* data sets. The last column shows the final number of spun pages identified.

Wiki data set: The *wiki* data set contains the first crawl of each day over a month, yielding $1,233,595$ initial pages of the 37M total. To make data processing time reasonable, we did not use all crawled pages in the analysis. And we chose one of the crawls each day to get a longer-term sample over time (rather than many of the crawls per day). Each of the filters has a notable impact on the data set (e.g., there are many empty, tiny, or non-English postings), leaving $632,966$ after filtering as our baseline set of pages. Of these, $205,085$ are duplicates and removing them yields $427,881$ pages with substantive content. Of those, the immutable method labels $244,107$ pages as potentially spun. After applying the mutable verifier, which verifies the pages and removes near duplicates, DSpin identified a total of $225,070$ spun pages.

GoArticles data set: We crawled $1,239,700$ *GoArticles* pages over the time period from January 2012 to May 2013. Most of the pages have substantive content, so the filters affect this data set less. As an article directory, GoArticles has much less spam on it than the abused wikis. Applying the filters and removing exact duplicates leaves $1,003,317$ pages. Of those, the immutable method labels $100,181$ as likely spun. With the mutable verifier and near duplicates removed, DSpin identified $71,876$ spun pages.

In summary, after filtering the wiki pages DSpin identified a majority of the pages, 68.0%, as SEO spam. Of these, 32.4% are exact duplicates and 35.6% are spun content. The *GoArticles* data set has drastically less spun content (7.0%) than the *wiki* data set. These results align with expectations. The *GoArticles* data set is a legitimate article directory that purportedly tries to filter spam content, whereas the *wiki* data set contains sites known to be targets for spammers. From this point forward we will focus our discussion on the spun content only.

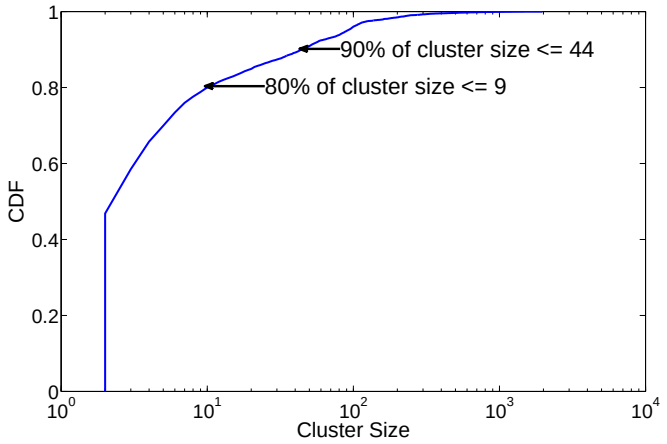


Fig. 7. CDF of cluster sizes for the *wiki* data set.

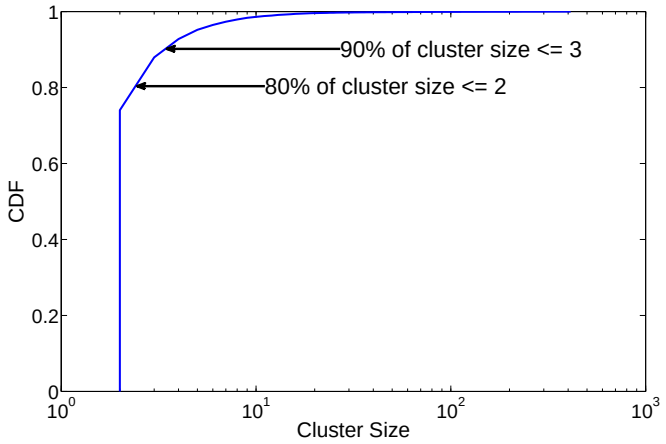


Fig. 8. CDF of cluster sizes for the *GoArticles* data set.

B. False Positives

We examined the false positive rate of the clustered data set. To this end, we randomly sampled 99 clusters from both the *GoArticles* and *wiki* data sets. For each cluster, we chose two pages at random, inspected them visually, and determined if the two articles appear to be spun from a common source. Using this methodology, we examined a total of 396 articles and found no evidence of false positives, where a false positive is defined as two articles that appear in the same cluster but are unrelated.

However, we find some clusters contain articles with mostly duplicate content. These texts are mostly the same with differences in adding a title and/or footer. In the *wiki* dataset, 27 out of 72 article pairs are mostly duplicates; while in the *GoArticles* dataset, 14 out of 85 samples are mostly duplicates.

Further, manual inspection of the cluster samples reveals that there were some differences between spun articles. Some spun articles appear mechanically spun, as the sentences read awkwardly or have semantic deficiencies. Other articles appear to have benefited from human intervention, as the sentences read smoothly and are meaningful. We consider the problem of classifying spun content into machine and human categories to be an interesting challenge for future work.

C. Cluster Sizes

SEO folklore holds that the more backlinks a promoted page has, the more effective the SEO campaign. Next we measure the scale at which spammers post spun content, presumably reflecting the scale at which they perceive backlinks are necessary for effective SEO. In particular, we look at the distribution of cluster sizes for both the *wiki* data set and the *GoArticles* data set, shown in Figures 7 and 8. Each cluster represents the set of spun pages generated from a given source page. We find a total of $12,783$ clusters from the *wiki* data set compared with $27,141$ clusters from the *GoArticles* data set. Even though the *wiki* data set has more spun pages, it has fewer clusters. This results from more spun pages per cluster.

From Figure 7, a majority of spun pages resides in clusters of size 600 pages or less for the *wiki* data set. But most clusters are small: over 80% of the clusters have nine postings or fewer. And as a single site, cluster sizes are even smaller on *GoArticles*. Figure 8 highlights this point by annotating that 80% of clusters have two pages, and 90% of clusters have three pages or less.

Topical Words (*Wiki*)	Freq.	Topical Words (*GoArticles*)	Freq.
business	0.110%	business	0.120%
internet	0.100%	windshield	0.110%
company	0.095%	online	0.110%
insurance	0.087%	glass	0.110%
online	0.086%	auto	0.094%
marketing	0.085%	company	0.090%
information	0.072%	car	0.084%
car	0.054%	internet	0.084%
weight	0.053%	information	0.062%
body	0.053%	offer	0.061%

TABLE III. FREQUENT TOPICAL WORDS IN SPUN CONTENT.

D. Content

If spinning campaigns manipulate a promoted page's rank via backlinks from spun pages with seemingly viable, non-duplicated content, understanding the content of such spun pages can indicate the affiliations of spun pages. To this end, we calculate word frequencies on spun content found in the *GoArticles* and *wiki* data sets. We extract frequent words, and then remove any words that are not a noun, a verb, or an object. Table III shows the top ten topical words, where a topical word suggests the topic of an article.

Overall, most of the popular words appear to relate to sales and services, some to particular markets (automobile, weight loss). This is no surprise since a major incentive to increasing a page's search rank would be to encourage more traffic to a Web site and increase sales opportunities.

E. Domains

Next we characterize the relationship between spun articles and domains. If spammers post spun articles on pages within a single domain, then the total set size considered for duplicate detection algorithms would be greatly reduced. However, if spammers post spun pages across many domains, then detection techniques would have to account for this behavior. We evaluate this relationship for the *wiki* data set as all pages from the *GoArticles* data set exist in the same domain.

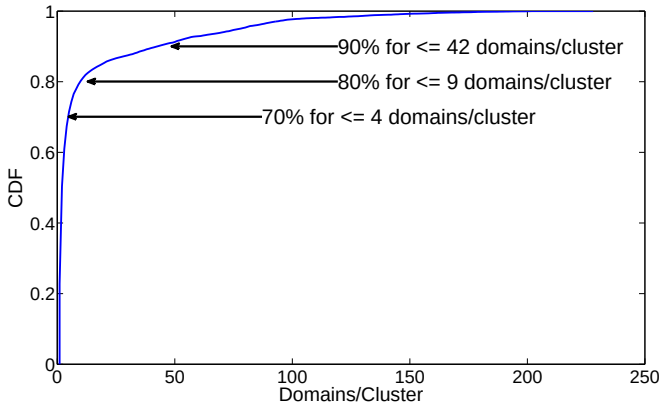


Fig. 9. CDF of the number of domains in clusters for the *wiki* data set.

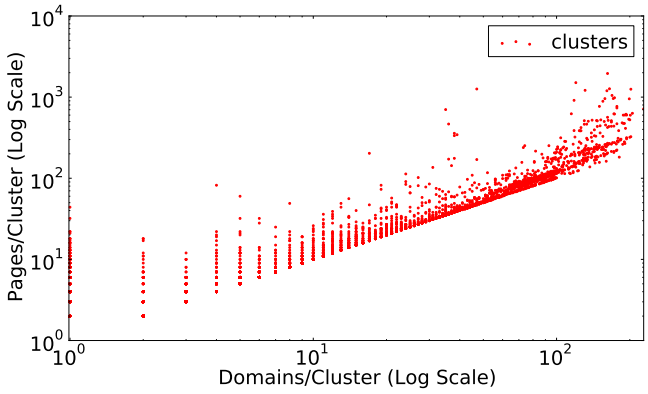


Fig. 10. Number of pages per cluster versus number of domains per cluster for the *wiki* data set.

1) Spun Content Across Domains: We map each page to its domain, and compute the number of distinct domains that appear within each cluster. Figure 9 displays a CDF of the number of domains per cluster. The CDF shows that although some clusters contain only one domain, the average cluster spans across 12 ± 27 domains, with some clusters spanning across as many as 228 domains. This result provides evidence that spammers target multiple domains when posting spun content, instead of a single site.

Further, Figure 10 shows the relationship between the number of domains in a cluster and the number of pages in the cluster. It indicates a strong, positive correlation between larger scale spinning campaigns and a larger number of targeted domains. For instance, the largest spinning campaign that spans only a single domain has 44 pages, while the largest spinning campaign that spans 228 domains contains 716 pages.

2) Spun Content per Domain: Spammers may target some domains more heavily than others, for numerous reasons. In Figure 11 we calculate the percentage of spun content per domain, for all domains that contain spun content. For each domain, we normalize the quantity of spun content to the total number of pages from that domain. The bulk of the distribution are when domains have 15%–65% spun content, showing that, when a wiki is targeted for spamming, it is targeted heavily.

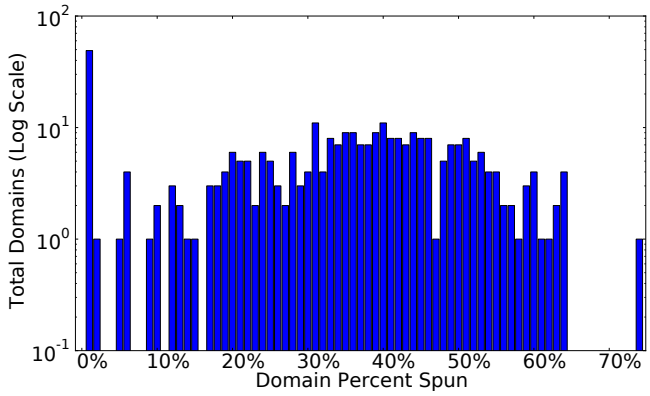


Fig. 11. Percent of spun content per domain normalized by total URLs in domain for the *wiki* data set.

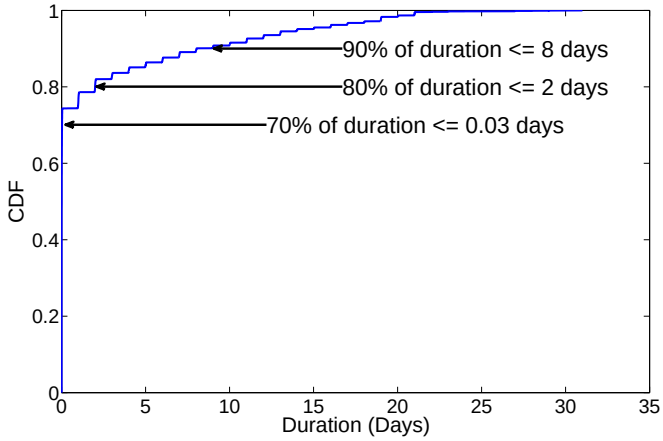


Fig. 12. CDF of duration in days of a spinning-based spam campaign in the *wiki* data set.

F. Timing

This section examines spammer behaviour in terms of the rate and duration at which spammers post spun articles during a spinning campaign. It is possible that spammers may want to post spun content over a longer duration of days, so as to avoid raising suspicion about a rapid deluge of spun content. However, a long-term spamming campaign may not be as effective for SEO as a short-term one. We first describe how we scrape the timing information for each spinning campaign in the two data sets, and then discuss the results.

1) Wiki: Most wiki pages contain a last modified tag, and we use this tag to extract the time of post. However, the tag text itself does not have a uniform template, and many are in languages other than English. We scrape this field and use regular expressions to parse the date for common templates in most languages. Some pages contain either no timing information or a language not supported by the parser. After we cluster pages of spun content, we parse $145,885$ pages out of $225,070$ that contain timing information. We find the minimum and maximum date in each cluster to compute the duration in days for each spam campaign.

Figure 12 shows a CDF of the duration in days of the spinning-based spam campaigns in the *wiki* data set. Nearly

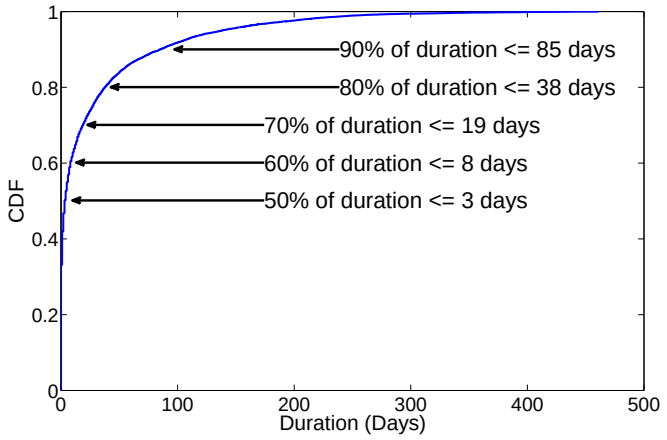


Fig. 13. CDF of the duration in days of a spinning-based spam campaign in the *GoArticles* data set.

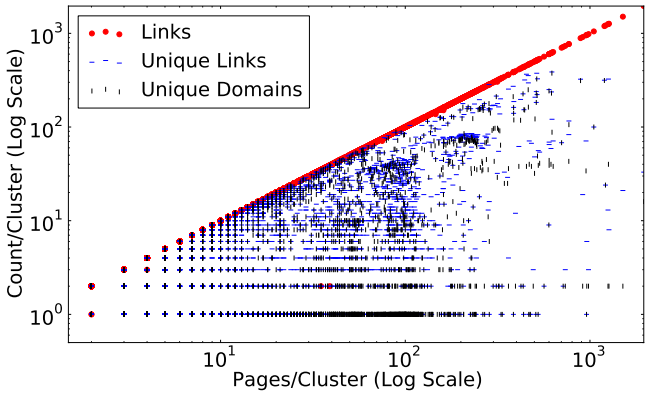


Fig. 14. A scatter plot of links, unique links, and unique domains versus the size of each spun cluster in the *wiki* data set.

75% of these campaigns last for less than a day. These results confirm intuition. When spammers use automated tools to post spun content, it is a short-term interactive process. Given these results, the timing behavior of these campaigns might be a useful additional feature in clustering spun pages into a campaign, although we have yet to explore it.

2) GoArticles: Since all the pages in the *GoArticles* data set are from a single site, extracting their post times is straightforward. Figure 13 shows a CDF of the duration in days of the spinning-based spam clusters in the set. The time scales are substantially longer than the *wiki* data set. More than 40% of the clusters have durations of at least a week, and 20% were a month.

One explanation for these long durations is that the clusters we are identifying are not actually from the same spam campaign. For the most part, the pages in a cluster are related by spinning, but either one of the pages was used as a seed, or the GoArticles site was spammed at different times by different campaigns (perhaps sharing a seed page from another site).

G. Backlinks

A spinning campaign may contain backlinks to promote a site through black hat SEO. There are two general strategies to backlinking used by campaigns. In one strategy, spammers can place the same backlink on every spun page. Here it is straightforward to confirm that pages belong to the same campaign by verifying that they promote the same site. However, there is another strategy, where spammers create spinning campaigns with backlinks pointing to a variety of sites to hinder spamming detection and prevent penalization of the search engine ranking of a single monetization site.

To examine this scenario, we scrape the backlinks from every page within a cluster and create a map of pages to a set of backlinks. Figures 14 and 15 show the number of links, unique links, and unique domains versus the number of pages per each cluster for the *wiki* and *GoArticles* data sets, respectively.

1) Wiki: First, we observe that the number of links per cluster closely tracks the number of pages per cluster (see Figure 14). Links occur on $99.97\% \pm 1.41\%$ of pages per cluster on average. This data corroborates that spinning campaigns contain backlinks.

Second, the number of unique links per cluster trends below the number of links. On average, unique links occur on $59.48\% \pm 34.27\%$ of the pages per cluster. Only two clusters that contain more than 100 pages contain all unique links. However, many clusters that are smaller than 100 pages have all unique links. Taken together, the unique links data suggests that most spinning campaigns promote fewer sites than the number of spun pages, but some spinners craft unique backlinks for each spun page.

Third, we consider the number of unique domains pointed to by backlinks versus the cluster size of the article. The intention here is to capture incidents where a clever spammer uses unique backlinks to avoid detection, but still needs to point to a common domain. For clusters less than 100 pages, most unique backlinks also point to unique domains (unique domains occur on $55.53\% \pm 33.49\%$ of such cluster pages on average). However, as the cluster size reaches at least 100 pages, the number of unique domains trends smaller than the number of unique backlinks (unique domains occur on $15.51\% \pm 19.90\%$ of such cluster pages on average). This result indicates that the larger spinning campaigns tend to target a much smaller set of domains compared to the number of pages in the campaign, such that unique domains occur on $53.82\% \pm 34.00\%$ of cluster pages on average.

2) GoArticles: The *GoArticles* data set shares many of the same trends as the *wiki* data set, but at smaller scale (see Figure 15). Clusters containing less than 10 pages have an equivalent number of links, unique links, and unique domains in 33.70% of cases. However, clusters with at least 10 pages tend to have several duplicate links and domains (only 7.10% of clusters with at least 10 pages have an equivalent number of links, unique links, and unique domains). This behavior suggests different strategies, with larger spinning campaigns generally targeting a smaller set of unique backlinks and domains than the number of pages in the campaign, although a few larger clusters contain all unique links and domains.

H. GoArticles as Seed Pages

Finally, we examine the relationship between spammed wiki pages and the article directory. We look at the possibility

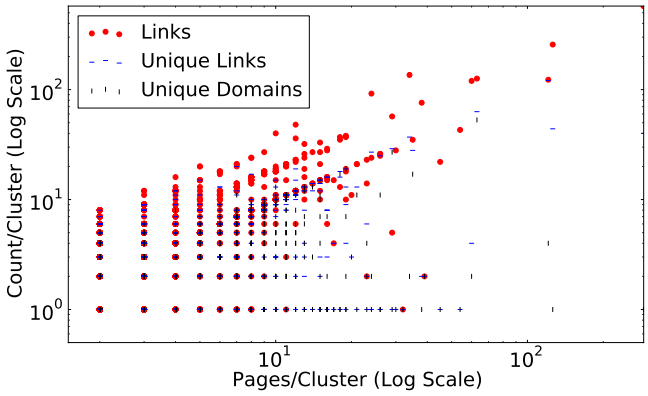


Fig. 15. A scatter plot of links, unique links, and unique domains versus the size of each spun cluster in the *GoArticles* data set.

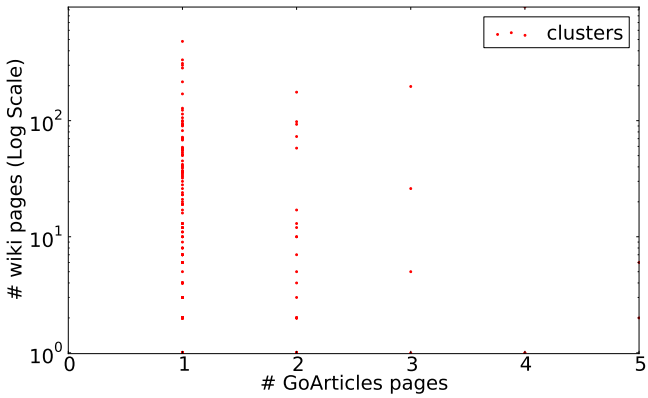


Fig. 16. Composition of clusters containing pages from both the *wiki* and *GoArticles* data sets.

that spammers use *GoArticles* as seed pages for spinning, and then post the spun content on wiki pages. We explore this question by examining the crossover of the two data sets.

We combine the month crawl of the *wiki* data set with the *GoArticles* data set. If these two data sets are disjoint, then every cluster of spun content would only contain articles from one data set. In this case, the spun articles from the *wiki* and *GoArticles* sites likely represent separate spam campaigns. On the other hand, if spun clusters cross data set boundaries, this overlap would indicate a spam campaign observed across both data sets. If clusters do exhibit this behavior, one explanation is that spammers share the same pool of spun articles and post spun content ubiquitously across domains. Alternatively, spammers may take content from article directories, and use it to seed generation of spun content.

We look at two metrics in the overlap, the number of pages from *GoArticles* and *wiki* in each cluster, and the times at which the *GoArticles* and *wiki* pages are posted to the sites.

We found 229 clusters that contain both *GoArticles* and *wiki* pages. Figure 16 plots the number of *GoArticles* (x-axis) and *wiki* (y-axis) pages that occur for each cluster. The trend indicates that the majority of cross domain clusters contain many wiki pages (31.6 on average), compared with just 1.2 on average for *GoArticles* (82% of these clusters contain just

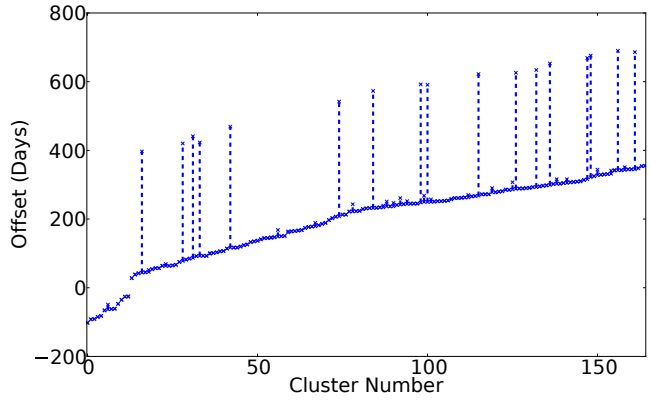


Fig. 17. Time difference between *wiki* and *GoArticles* post times in overlapping clusters with one page from *GoArticles* and the remaining pages from the *wiki* set.

one). Furthermore, the low number of pages from *GoArticles* in these clusters is independent of the number of *wiki* pages in the clusters.

Timing is another indication. If spammers are taking articles from article directories and using them as seeds for spinning, the posting times from the article directory should precede the times on the wiki pages; otherwise, the wiki and article directory post times will often be uncorrelated with each other. Figure 17 explores the posting times in detail. For each cluster containing one *GoArticles* page and at least one *wiki* page on the x-axis, it shows two points on the y-axis connected by a line. The bottom point shows the earliest posting time of a *wiki* page in the cluster relative to the posting time of the *GoArticles* page, and the top point similarly shows the relative posting time of the latest posted *wiki* page.

Most of the wiki pages (92%) are posted well after the *GoArticles* page, and often by a very large gap of weeks to months. In these cases, one conclusion is that the spammers responsible for this crossover behavior are taking articles from article directories, spinning them, and posting them on wiki pages. There are a small number of exceptions, though. The clusters with negative y values are when the post time in *GoArticles* precedes the *wiki* pages. For these, we conclude that the spammers used seed articles from somewhere else, and *GoArticles* was simply another target site for posting spun content.

VII. Discussion

Automated spinning is directly a response to defensive pressure: near-duplicate document detection by services like Google [4], [32]. As shown in Section IV, spinning undermines shingle-based approaches and plagiarism detectors like CopyScape. If services deploy defenses like DSpin against automated spinning, spammers will similarly respond to such pressure to evade detection. Likely responses fall into changing how the tool spins content, where it spins the content, or how much of the content it spins.

One spammer response might be to change the dictionary frequently. As noted in Section IV-B, TBS does not currently change its synonym dictionary much over time: 94% of the

words in the dictionary remained the same over four months. If it were to change the dictionary more frequently, it would have to change the set of words it spun (the mutable set) and/or the set of synonyms used for replacement. Since these changes would likely result in a smaller dictionary at any particular time, it might reduce the quality of spun content — although likely still sufficient for the goals of posting spun articles for link spam. DSpin is insensitive to some degree of dictionary variation since it does not have to be precise; it is encouraging that DSpin could identify automatically spun content in the wild that was generated over a year and presumably by tools other than TBS. However, if DSpin did need to precisely track frequent changes to the synonym dictionary over time, then the resulting burden would be the ease or difficulty of obtaining updated dictionaries.

Currently tools like TBS download the dictionary locally to produce spun content, making the dictionary vulnerable to automated download and analysis. Instead, tools could compute spun content remotely so that clients do not have access to the dictionary. Remote computation will increase cost, but the availability of cheap compromised hosts [8] will unlikely undermine the profibility of spinning tools. To counter, defenses like DSpin could reverse-engineer immutables by generating spun content and examining the output for invariant text, much like techniques for inferring botnet email spam templates [29]. By using such inferrence a spinning detector would then also automatically infer the synonym dictionary, obviating the need to obtain it or update it over time.

Another set of questions concerns the generality of DSpin in terms of other spinning tools, manual spinning, and scale. For any dictionary-based spinning tool, in principle the immutable method in DSpin should be able to detect their spun output. The human-generated spun articles DSpin does detect (Section VI-B), it likely does so coincidentally. However, manual spinning likely still has immutable words that can be used as anchors, and may be vulnerable to inference across a set of samples [29]. We have not experimented with other spinning tools or human-generated spun content, though, and both remain open questions for future work.

In terms of scale, the workload we used to evaluate DSpin is modest and an immediate question is whether DSpin could detect spun content at Internet scale. Given that Google has scaled its systems for detecting near-duplicate content to the entire Web [23], we believe that there is no fundamental reason why the immutable method in DSpin could not as well. In one sense, the immutable method in DSpin is analogous to near-duplicate detection (detecting duplicated immutables), yet requires less data to compare since DSpin only compares immutables — a much smaller fraction of any article (Section IV-B). In fact, DSpin might be readily incorporated into Google's simhash-based approach by using immutables as the features in document hashes. As a result, scale is of less concern than how spammers might respond to evade spinning detection.

VIII. Conclusion

In this paper we have proposed a method for detecting automatically spun content on the Web. In this method, we use the synonym dictionary that spinning tools use to create spun content as a filter, reducing crawled pages to a much smaller set of "anchor" words that remain unchanged and are common among all of the spun pages generated from the same seed page. We then implement this method in a tool called DSpin that operates on sets of crawled Web pages to identify spun content. Using controlled experiments, we show that DSpin successfully identifies spun content, and moreover clusters spun content according to the set of spun pages generated together. We then apply DSpin to two crawled data sets, a set of wiki pages known to be targets of Web spam and a popular article directory.

With DSpin, we find that spinning is a popular spamming technique on certain sites. A significant amount of the posted content on the wikis is not only Web spam, but also spam that consists of automatically spun content. We also find that a portion of the clusters of spun content span both the wikis and the article directory, with evidence that spammers do use article directories as seed pages.

Acknowledgements

We are grateful to Richard Strong for insightful comments and detailed feedback on the paper, Chris Grier for his assistance and support with Hadoop, Steve Checkoway and David Kohlbrenner for insights on reverse-engineering the TBS dictionary, and Damon McCoy for proxy support during crawling. We also thank the anonymous reviewers for their valuable feedback. This work was supported in part by the Office of Naval Research MURI grant N000140911081, by National Science Foundation grant NSF-1237264, and by generous research, operational and/or in-kind support from the UCSD Center for Networked Systems (CNS).

References

[1] Blackhat SEO Forum. http://www.blackhatworld.com/, 2013.

[2] Natural Language Processing Software. http://nlp.stanford.edu/, 2013.

[3] The Best Spinner User Guide. http://thebestspinner.com/TBSUsersGuide.pdf, 2013.

[4] A. Aders. What You Need to Know About Google's Penguin Update. http://www.inc.com/aaron-aders/what-you-need-to-know-about-googles-penguin-update.html, 2012.

[5] A. Z. Broder. On the resemblance and containment of documents. In *In Compression and Complexity of Sequences (SEQUENCES '97)*, pages 21–29. IEEE Computer Society, 1997.

[6] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *Selected papers from the sixth international conference on World Wide Web*, pages 1157–1166, Essex, UK, 1997. Elsevier Science Publishers Ltd.

[7] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntatic Clustering of the Web. SRC Technical Report 1997-015, Digital Equipment Corporation, July 1997.

[8] J. Caballero, C. Grier, C. Kreibich, and V. Paxson. Measuring Pay-per-Install: The Commoditization of Malware Distribution. In *Proceedings of the USENIX Security Symposium*, San Francisco, CA, August 2011.

[9] M. S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the 34th ACM STOC Conference*, May 2002.

[10] A. Chowdhury, O. Frieder, D. Grossman, and M. McCabe. Collection Statistics for Fast Duplicate Document Detection. *ACM Transactions of Information Systems (TOIT)*, 20(2), April 2002.

[11] M. Elhadi and A. Al-Tobi. Use of text syntactical structures in detection of document duplicates. In *ICDIM*, pages 520–525. IEEE, 2008.

[12] Example Job for Automatically Spinning. http://fiverr.com/virtualgirl2010/spin-1-article-and-make-it-50-to-60-percent-unique.

[13] Example Job for Manual Spinning. http://www.freelancer.com/projects/Editing-Articles/Article-spinner-sentence-level.html.

[14] D. Fetterly, M. Manasse, and M. Najork. Spam, Damn Spam, and Statistics: Using statistical analysis to locate spam web pages. In *Proceedings of the 7th WebDB Workshop*, June 2004.

[15] D. Fetterly, M. Manasse, and M. Najork. Detecting Phrase-Level Duplication on the World Wide Web. In *Proceedings of the ACM SIGIR Conference*, August 2005.

[16] Google's Search Engine Optimization Starter Guide. https://static.googleusercontent.com/external_content/untrusted_dlcp/www.google.com/en/us/webmasters/docs/search-engine-optimization-starter-guide.pdf, 2010.

[17] Z. Gyöngyi and H. Garcia-Molina. Web Spam Taxonomy. In *Proceedings of 1st AIRWeb Workshop*, May 2005.

[18] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating Web Spam with TrustRank. In *Proceedings of the 30th VLDB Conference*, August 2004.

[19] M. Henzinger. Finding Near-Duplicate Web Pages: A Large-Scale Evaluation of Algorithms. In *Proceedings of the 29th ACM SIGIR Conference*, August 2006.

[20] P. Kolari, A. Java, T. Finin, T. Oates, and A. Joshi. Detecting spam blogs: a machine learning approach. In *proceedings of the 21st national conference on Artificial intelligence - Volume 2*, AAAI'06, pages 1351–1356. AAAI Press, 2006.

[21] T. G. Kolda and M. J. Procopio. Generalized BadRank with Graduated Trust. Technical Report SAND2009-6670, Sandia National Laboratories, October 2009.

[22] L. Lu, R. Perdisci, and W. Lee. SURF: Detecting and Measuring Search Poisoning. In *Proceedings of the ACM CCS Conference*, October 2011.

[23] G. S. Manku, A. Jain, and A. Das Sarma. Detecting Near-duplicates for Web Crawling. In *Proceedings of the 16th WWW Conference*, May 2007.

[24] MediaWiki. http://www.mediawiki.org.

[25] T. Moore, N. Leontiadis, and N. Christin. Fashion Crimes: Trending-Term Exploitation on the Web. In *Proceedings of the ACM CCS Conference*, October 2011.

[26] M. Motoyama, D. McCoy, K. Levchenko, S. Savage, and G. M. Voelker. Dirty Jobs: The Role of Freelance Labor in Web Service Abuse. In *Proceedings of the USENIX Security Symposium*, San Francisco, CA, August 2011.

[27] M. Najork. Detecting quilted web pages at scale. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 385–394, Portland, Oregon, USA, 2012.

[28] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting Spam Web Pages Through Content Analysis. In *Proceedings of the 15th WWW Conference*, May 2006.

[29] A. Pitsillidis, K. Levchenko, C. Kreibich, C. Kanich, G. M. Voelker, V. Paxson, N. Weaver, and S. Savage. Botnet Judo: Fighting Spam with Itself. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2010.

[30] Y. Shin, M. Gupta, and S. Myers. The Nuts and Bolts of a Forum Spam Automator. In *Proceedings of the 4th USENIX LEET Workshop*, March 2011.

[31] N. Shuyo. Language Detection Library for Java Software. https://code.google.com/p/language-detection/, 2013.

[32] A. Singhal and M. Cutts. Finding More High-Quality Sites in Search. http://googleblog.blogspot.com/2011/02/finding-more-high-quality-sites-in.html, 2011.

[33] M. Sobek. PR0 — Google's PageRank 0 Penalty, 2002.

[34] M. Theobald, J. Siddharth, and A. Paepcke. SpotSigs: Robust and Efficient Near Duplicate Detection in Large Web Collections. In *Proceedings of ACM SIGIR*, July 2008.

[35] D. Y. Wang, S. Savage, and G. M. Voelker. Cloak and Dagger: Dynamics of Web Search Cloaking. In *Proceedings of the ACM CCS Conference*, October 2011.

[36] D. Y. Wang, S. Savage, and G. M. Voelker. Juice: A Longitudinal Study of an SEO Campaign. In *Proceedings of the NDSS Symposium*, February 2013.

[37] B. Wu and B. D. Davison. Identifying Link Farm Spam Pages. In *Proceedings of the 14th WWW Conference*, May 2005.

[38] B. Wu, V. Goel, and B. D. Davison. Propagating Trust and Distrust to Demote Web Spam. In *Proceedings of the MTW Workshop*, May 2006.