Design, Implementation and Test of a Flexible Tor-Oriented Web Mining Toolkit

Alessandro Celestini
Institute for Applied Computing (IAC-CNR),
Via dei Taurini 19, Rome, Italy
a.celestini@iac.cnr.it

Stefano Guarino
Institute for Applied Computing (IAC-CNR),
Via dei Taurini 19, Rome, Italy
s.guarino@iac.cnr.it

Abstract

Searching and retrieving information from the Web is a primary activity needed to monitor the development and usage of Web resources. Possible benefits include improving user experience (*e.g.* by optimizing query results) and enforcing data/user security (*e.g.* by identifying harmful websites). Motivated by the lack of ready-to-use solutions, in this paper we present a flexible and accessible toolkit for structure and content mining, able to crawl, download, extract and index resources from the Web. While being easily configurable to work in the "surface" Web, our suite is specifically tailored to explore the Tor dark Web, *i.e.* the ensemble of Web servers composing the world's most famous *darknet*. Notably, the toolkit is not just a Web scraper, but it includes two mining modules, respectively able to prepare content to be fed to an (external) semantic engine, and to reconstruct the graph structure of the explored portion of the Web. Other than discussing in detail the design, features and performance of our toolkit, we report the findings of a preliminary run over Tor, that clarify the potential of our solution.

***K*eywords** Dark Web, Tor Web graph

1 Introduction

As the Web has become the main means for information exchange and retrieval, a whole body of work focuses on gaining a better understanding of its content and shape, in order to improve usability and security. A few seminal papers [1, 2, 3] allowed drawing the first publicly known portrait of the "surface" Web, thus paving the way for further research which, over the last two decades, provided significant results: the optimization of search algorithms [4] and Web data extraction [5], the identification of efficient compressed representations [6, 7, 8], the proposal of suitable representation (*i.e.* graph based) models [9, 10], just to name a few. Web mining becomes an even more interesting/challenging task when the target includes the submerged Internet contents usually known as "deep" Web (estimated to sum up to more than 90% of the whole World Wide Web (WWW)), not crawled/indexed by traditional search engines. In general, while search engines are specifically tailored to respond to user queries with popularity as one of the key preferential criteria, the heterogeneous and unstructured nature of data available on the WWW calls for flexible Web mining tools able to extensively gather, organize, index and analyse Internet resources.

A recent research trend is especially focused on the subset of the deep Web usually called "dark" Web. This is the collection of web resources that exist on *darknets*, describable as overlay networks, which despite leaning on the public Internet require specific software, configuration or authorization to access. Among darknets, Tor (The Onion Router [1]) is probably the most known and used.

[1] https://www.torproject.org/

It is a communication network designed as a low-latency, anonymity-guaranteeing and censorship-resistant network, relying on an implementation of the so-called onion routing protocol [11]. Its servers, run by volunteers over the Internet, work as routers to allow Tor users to access the Internet anonymously, evading traditional network surveillance and traffic analysis mechanisms. Other than guaranteeing an anonymous access to normal websites, Tor allows running anonymous and untraceable Web servers, known as *hidden services*, that can only be accessed using a Tor-enabled browser. A hidden service is identified by its *onion url* and it is not associated with any (visible) IP address. Tor is able to interpret such urls and forward data packets to and from a hidden service, guaranteeing anonymity in both directions. Hidden services can offer various kinds of services, not just Web servers, without needing to reveal the location of the site. The main reason why both the research community [12, 13, 14, 15, 16] and society/media are increasingly showing interest in Tor is that a completely anonymous network represents the perfect breading ground for illegal activities: Tor hidden services have been accused to provide protection to terrorists [2], and are known to host marketplaces for drugs, weapons and pedo-pornography [17]. Mining the structure and content of the Tor dark web is a primary instrument to gain a better understanding of how Tor works and how it is being used, with a direct impact on both user experience and security.

Despite publicly available and effective crawlers do exist [18, 19, 20, 21, 22], we feel that the research community lacks a sufficiently general-purpose configurable solution for addressing (deep, dark) Web mining. The purpose of this paper is exactly to fill that gap, by presenting the design, implementation and test of a toolkit which can be easily used to automatically explore the Web. In our suite, Web crawling is made very "user-friendly" by letting a coordinator module control the spider, *i.e.* the software concretely responsible of jumping from site to site along hyperlinks. All collected data are extracted and indexed, and can be easily forwarded for content mining to any external semantic engine accessible through RESTful APIs. Additionally, our solution integrates a module that reconstructs the explored Web graph, thus allowing structural/topological analysis. We introduce our toolkit in a Tor-oriented guise, but it can be tuned to work on any subset of the Web, at the cost of a minor configuration effort.

To support the potential of our solution, we also report the highlights of a preliminary (yet wide) exploration of the Tor dark Web. These preliminary tests were run in the context of the "IANCIS" ISEC Project [3], and the presence of Expert System [4] as project partner gave us the access to their data analysis solutions. Specifically we had the opportunity of using their tool Cogito [5] as external semantic engine, as reported in Section 4. An extensive analysis of our findings lies beyond the scope of the present paper, whose focus is on design and implementation, and it is to be found in [23]. However, by making our toolkit available on demand and providing a preview of its capabilities, we are confident to give new impetus to the Web mining literature, and to pave the way for the extension of the body of work related to Tor and other similar darknets.

This paper is organized as follows: Section 2 presents a review of related work; Section 3 details the design and the performance of our framework; Section 4 discusses a set of preliminary findings that highlight the relevance of the proposed tool; finally, Section 5 reports a few concluding remarks and directions for future work.

2 Related Work

The Tor network has attracted significant attention from the research community, interested both in assessing its security with respect to de-anonymization attacks, and in understanding which threats a publicly available anonymous communication system could expose society to. One of the main research directions over Tor analysis consists in studying how Tor is used and what kind of contents its websites host, with the aim of understanding whether the charge of providing protection and anonymity to criminals is indeed well deserved. Spitters *et al.* [16] present a study focused on the analysis of Tor hidden services contents. They apply classification and topic model techniques to analyse the contents of over a thousand Tor hidden services, in order to model their thematic organi-

[2] http://www.bloomberg.com/news/articles/2014-10-15/how-anti-is-spies-fight-terrorism-with-digital-tools

[3] DG Home Affairs ISEC Programme 2013 - Project IANCIS "Indexing of Anonymous Networks for Crime Information Search", 2014-2016, GA n. HOME/2013/ISEC/AG/INT/4000005222 - www.iancis.eu

[4] http://www.expertsystem.com/

[5] http://www.expertsystem.com/cogito/

zation and linguistic diversity. Their results indicate that most hidden services in their data set exhibit illegal or controversial contents. Along the same line, Biryukov *et al.* [15] give an overview of Tor hidden services, studying 39824 hidden service descriptors collected in 2013 [13], and analysing and classifying their contents. At the time of the crawl they were able to connect to 7114 destinations, but half of them were excluded because inappropriate for classification, ending up with 3050 destinations. Their results show that resources devoted to criminal activities (drugs, adult content, counterfeits, weapons, etc.) constitute 44%, whereas the remaining 56% are devoted to a number of different topics: "Politics" (9%) and "Anonymity" (8%) are among the most popular. Biryukov *et al.* also estimate the popularity of hidden services by looking at the request rates for hidden service descriptors by clients. They find that, while the content of Tor hidden services is rather varied, the most popular hidden services are related to botnets. Even if the approach used by Biryukov *et al.* for hidden service descriptors collection cannot be reproduced (they exploited a bug of Tor, fixed in recent versions of the software) other authors presented similar studies. Owen and Savage [24] analysed hidden services' contents resorting to manual classification, while Biryukov *et al.* used topic modelling techniques. Other works focus their attention on specific subsets of Tor hidden services, like blogs, forum or marketplaces. An example is [25], where Soska and Christin present a study focused on Tor marketplaces showing interesting evolutionary properties of the anonymous marketplace ecosystem. The authors perform a specific Tor Web crawling, collecting data from 16 different marketplaces over more than two years, without focusing on a specific category of products. The results of their study suggest that marketplaces are quite resilient to law enforcement takedowns and large-scale frauds. They also evidence that the majority of the products being sold belong to drugs category.

Web crawling is a well-studied topic, and several successful crawlers' design have been proposed [18, 20, 22]. Nevertheless, research interest in the subject is still significant, with recent studies concentrating on focus and specialized crawlers. As a matter of fact, to extract specific information from Web sources like blogs, forums or websites it is usually necessary to design a customized crawler. An example is the BlogForever Crawler [26] presented by Blanvillain *et al.*, whose goal was to develop a crawler able to automatically extract information from blog posts (e.g. articles, authors, dates and comments). The crawler is actually a component of the BlogForever platform whose aim is to harvest, preserve, manage and reuse blog content. Zhao *et al.* [27] propose SmartCrawler a framework for efficient harvesting deep Web interfaces. Authors show that SmartCrawler is capable of achieving both wide coverage of deep Web interfaces and maintaining highly efficient crawling. Another example of special purpose crawler is TwitterEcho, an open source crawler proposed by Bošnjak *et al.* [28]. TwitterEcho is a Twitter crawler, which allows retrieval of Twitter data from a focused community of interest. The crawler is designed as distributed application enabling the crawling of high volumes of data, while respecting the limits imposed by Twitter. Authors state that TwitterEcho was designed to support academic researchers or analysts who wish to carry out research with focused Twitter data.

We observe that in general the crawling of particular types of websites, such as Tor's hidden services, requires the use of tailored approaches and customized tools. Specifically, we evaluated different alternatives for the development of our general-purpose crawler, including the implementation of a crawler prototype from scratch, and finally choosing BUbiNG [19] as the base for our crawler component. We additionally support the use of focused crawlers to improve data collection, with the possibility to merge data gathered resorting to different crawlers in a single dataset. This approach drove the design and implementation of our toolkit.

3 The Toolkit

We now present our toolkit, whose components and workflow are summarized in Figure 1. It consists of a core application for massive Web crawling, indexing and text mining, and of two external independent modules for customized/focused crawling and structure mining respectively [6].

[6] Researchers interested in using/testing our toolkit are invited to contact us. Other than to sharing the latest version, we will be available for support in all configuration aspects, including the integration of different semantic text analysis modules.

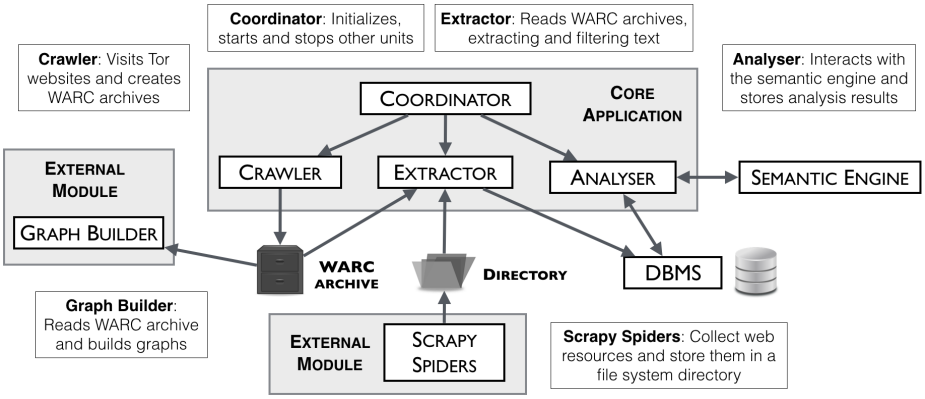

Figure 1: Components and workflow.

3.1 The Core Application

At the heart of the toolkit is a core application in charge of automatically exploring Tor websites and collecting their contents, while indexing and clustering gathered data. Based on an analysis of other applications oriented to the collection and analysis of data, we designed the core of our toolkit around four main components, all written in Java: a coordinator, a crawler, an extractor and an analyser. To manage the operations of the Java application two scripts are provided to start and stop its execution.

In the following, we describe the toolkit's core application more in detail: we start with an overview of its components and workflow, that prompts us to report on implementation and operational aspects; we then summarize all possible configuration options, pointing out the most important ones; finally, we focus on the crawler, discussing its design and functionalities.

3.1.1 Components and Workflow

The main process of the core application, that we developed from scratch, is the coordinator, responsible of organizing the operations of other units. When launching the tool, the coordinator is activated and it starts reading the configuration file and setting up the application. It checks the existence of the database and creates one if it does not exist. After initialization procedures, the coordinator activates the crawler, the process which concretely performs the task of visiting Tor websites.

The crawler carries out a breadth-first search of Tor websites and stores all retrieved resources in a WARC archive. A list of known hidden services, the *root set*, is provided to the crawler to determine the data collection starting points. When the archive reaches a threshold size, the coordinator stops the crawler. The threshold size of the archive is a configuration parameter, which can be modified in the configuration file. Once the crawler stops, the coordinator moves the WARC archive and creates a new empty file, which will be used by the crawler at its next activation, then the extractor starts.

The extractor is a multi-thread process that concurrently reads and extracts text from resources contained in the WARC archive (and/or in a file system directory, see Section 3.1.2). The extractor reads the WARC archive, extracts texts from collected data and stores them in a database. To extract text from digital documents, the extractor uses the Apache Tika API [7]. Apache Tika is a project of the Apache Software Foundation, providing a java toolkit able to detect and extract metadata and text from over a thousand different file types. The extractor only stores texts from web pages that replied successfully during the crawling (HTTP status code "200 OK"), and it filters extracted texts according to specific configuration options (*e.g.* on a language basis). All data are stored in a document-oriented database, in which each web resource is stored as a document. For each web resource the extractor stores: (i) the HTTP response header, (ii) the WARC record header, (iii) the metadata provided by Tika, (iv) the extracted text, and (iv) the language of the document's

[7] https://tika.apache.org

text. Specifically, we use ArangoDB [8], a multi-model, open-source, NoSQL database with flexible data models for documents, graphs, and key-values. It supports ACID transactions if required and provides a SQL-like query language or JavaScript extensions. Once the extractor terminates its operations, the coordinator activates the analyser.

The analyser is a multi-thread process that concurrently reads documents from the database and sends their texts to the semantic engine, for analysis. For each document the analyser prepares a RESTful request, containing its text and language, and sends the request to the engine. The engine sends back analyses results, which are again stored by the analyser in the initial database, together with other document's information. Once all documents have been analysed, the coordinator stops the analyser.

In our test, reported in Section 4, the analyser relies on Cogito, a multi-language semantic engine developed by Expert System, which can understand the meaning in context within unstructured text. Cogito is able to find hidden relationships, trends and events, transforming unstructured data into structured information. Through several analyses, it identifies three different types of entities (people, places and companies/organizations), categorizes documents on the basis of several taxonomies and it is able to extract entity co-occurrences. Since Cogito is a proprietary software, it cannot be included in any freely available version of our toolkit. Yet, our analyser can be used out-of-the-box with any semantic engine providing RESTful APIs. Embedding the analyser with other options for text mining is our primary goal for further development of the toolkit.

3.1.2 Configuration

The application behaviour can be set up by modifying a Java properties file, named "config.properties" (see Table 1 for a list of all available configuration parameters). First of all, through the configuration file it is possible to determine what operations the application will perform, by selecting any subset of the three building blocks, *i.e.* crawling, extraction, and analysis. If the extractor is enabled, either or both of two supported modalities can be selected namely extraction from WARC and extraction from file. In the first modality the extractor uses a WARC file, in the latter it uses a file system directory to retrieve resources to elaborate.

A number of options can be specified through the configuration file, including: (i) the number of threads used by the extractor and the analyser agents; (ii) the threshold size of WARC archives generated by the crawler – when that size is reached, the archive is passed to the extractor and a new archive is created by the crawler; (iii) the directory used by the crawler to store data; (iv) the name of the database to be created or used to store documents; (v) the name of the database's collections used by the extractor and the analyser agents; (vi) the directory used by the extractor to retrieve file – in case the extraction from file modality has been activated. Moreover, it is possible to activate the file system data storage option, *i.e.* data produced by crawler, extractor and analyser can be stored locally in the file system, keeping in mind that a copy of the same data is stored in the database by default. Finally, the configuration file contains the address and other parameters used to contact the RESTful web service exposed by the semantic engine. The settings concerning the RESTful web service allow the use of different engines whether remote or local, for the analysis of textual data.

3.1.3 The Crawler

We evaluated different alternatives for the development of the crawler. In particular, we implemented a crawler prototype from scratch and evaluated it against three main existing candidates: Apache Nutch [9] [29], Heritrix [10] [21] and BUbiNG [19]. Considering several criteria, such as performance, configurability, extensibility and supportability, we found BUbiNG to be the most appropriate choice as the base for our crawler component. BUbiNG is a high-performance, scalable, distributed, open-source crawler, written in Java, and developed by the Laboratory for Web Algorithmics (LAW) at the Computer Science Department of the University of Milan.

Significant efforts were needed for the integration of BUbiNG within our toolkit, so as to turn it into an application's component under the control of the coordinator. Moreover, we needed to enable BUbiNG operating in Tor instead of the surface Web. Whereas, by default, BUbiNG presents a set

[8] https://www.arangodb.com

[9] http://nutch.apache.org

[10] https://webarchive.jira.com/wiki/display/Heritrix

Table 1: List of all available configuration parameters (file 'config.properties').

Parameter	Description
extractorThreads	number of threads used by the extractor
analyserThreads	number of threads used by the analyser
dbUser	username used to access the database
dbPassword	password used to access the database
dbName	name of database used to store documents
extractedDocCollection	name of database's collection used to store extracted text
analysedDocCollection	name of database's collection used to store analysed text
crawling	enable/disable crawling operation
extraction	enable/disable extraction operation
analysis	enable/disable analysis operation
crawlerDir	directory used by crawler to create WARC files
extractWARC	enable/disable extraction from WARC, requires 'extraction=true'
extractFile	enable/disable extraction from file, requires 'extraction=true'
fileDir	directory containing files to extract, requires 'extractFile=true'
languageFilter	a comma separated list of languages, only texts in these languages are stored and analysed
warcSizeThreshold	threshold size of WARC archive, when reached crawler is stopped and extraction agent is started
storeFilesExtraction	enable/disable storage of texts extracted on file system
dirFilesExtraction	directory used to store texts extracted, requires 'storeFilesExtraction=true'
storeFilesAnalysis	enable/disable storage of analysis results on file system
dirFilesAnalysis	directory used to store analysis results, requires 'storeFilesAnalysis=true'
storeFilesCrawling	enable/disable storage of crawled data on file system
dirFilesCrawling	directory used to store crawled data, requires 'storeFilesCrawling=true'
engineHost	host name of RESTful web service
enginePath	invoked endpoint of RESTful web service
engineKey	key used to invoke RESTful web service
engineKeyFieldName	name of the form field used to store 'engineKey'

of threads that perform DNS requests, in our application we avoid these requests and send them directly to a HTTP proxy. We chose to use privoxy [11], after testing other alternatives. In particular, we decided not to use polipo [12], that is often used in combination with Tor, because it is no longer maintained and currently seems not able to manage correctly the format of some HTTP responses. However, any HTTP proxy configured to use Tor can be used. Through the crawler configuration file, that is a Java properties file named 'crawler.properties', it is possible to specify which HTTP proxy the crawler must use. Several other parameters can be set through that configuration file, including: (i) the number of threads the crawler will use for its operations (parsing, dns resolution, fetching);

[11] https://www.privoxy.org

[12] https://www.irif.fr/ jch/software/polipo/

(ii) which resources are collected from websites (*e.g.* html pages, media file, digital documents); (iii) network timeouts used when contacting websites; (iv) where collected date will be stored; (v) how and whether to manage cookies; (vi) the delays between requests to the same website. For a complete list of configuration parameters we refer the reader to BUbiNG's documentation [13]. With our toolkit we provide a standard crawler configuration, tested for Tor network, thus editing that file is not needed for standard usage.

For what concerns the crawling process, in BUbiNG a pool of software agents are responsible for both exploring the Web and collecting (and partially elaborating) the data. Such agents work in parallel, each handling in turn several threads, and by default implementing a *breadth-first-search*. Due to Tor's high volatility, the results of the crawling process are theoretically susceptible to fluctuations based on the order in which links are followed. Yet, we did not have any reason to prefer one order over another, and we therefore did not modify this setting. Summing up, in our experiments BUbiNG was used as follows:

- A predetermined set of hidden services, the *root set*, is inserted in an *url list*.
- The first *fetching thread* available extracts the first onion url from the list and exports the content of the corresponding hidden service in main memory.
- The first *parsing thread* available analyses that content aiming at extracting new onion urls to visit.
- The new onion urls found are passed to a *sieve* able to verify whether those urls were already visited before.
- If so, the urls are discarded, otherwise they are added in the tail of the url list.
- *Fetching threads* attempt to contact each url for a maximum of three times, after that the url is considered not available.

3.2 External Modules

Alongside of the core application, our toolkit comprehends two external modules which can be launched independently: a set of Scrapy Spiders, written in Python, and a Graph Builder, written in C. The spiders enhance the crawling process by permitting focused crawling, supporting semi-automated procedures, and offering anti-detection settings. The Graph Builder supports the reconstruction of the graph associated to the crawling process, enabling topological studies of the explored portion of the Web.

3.2.1 Scrapy Spiders

Besides BUbiNG, we developed and integrated in our toolkit a set of customized spiders which can be managed as modules and used to boost the crawling process. Specifically, our spiders were written relying on Scrapy [14], a Python framework for crawling websites and extracting data. Using ad-hoc spiders besides the main crawler allows for focused crawling which supports a semi-automated (*i.e.* human assisted) procedure needed to gain access to hidden contents requiring a login procedure or a captcha solver. Moreover, through configuration settings, we are able to choose a breadth-first or depth-first search strategy for the crawling procedure. A further customization required for our spiders consists in avoiding crawling detection, which may be implemented by our targets. To that purpose, we included configuration settings concerning the robots exclusion protocol, network timeouts and (again) delays between requests to the same website. Furthermore, we programmed our spiders to visit only specific sections of targeted websites and to carry out only legal and not suspicious actions. Finally, the data collected by Scrapy Spiders are stored as files in a directory and are integrated in the core application via the extractor module. Indeed, through configuration settings, the extractor unit can be instructed to retrieve files from system directories. The Scrapy Spiders module can be used as a template to create other spiders with the same framework, or as an example of how to integrate data collected by other crawlers.

[13] http://law.di.unimi.it/software/bubing-docs/overview-summary.html

[14] https://scrapy.org

3.2.2 Graph Builder

The Graph Builder is written in C and supports the reconstruction of the graphs associated to the crawling process, using the WARC archive created by BUbiNG. To parse HTML file and extract links we use the mythml library [15]. The Graph Builder module is a multi-thread application, it takes as input the name of the WARC file to read and the number of threads to use to build the graphs. For each WARC file three directed graphs are created by the module, a page graph, a host graph and a service graph, which are represented as list of edges. In the page graph an edge exists between page A and page B if there is a least a link from page A to page B. In the host graph an edge exists between host A and host B if there is a least a page of host A that links to a page of host B. The service graph is the higher level of grouping, in which each node represents an hidden service, which is identified by a 16 character string (base32 encoded). In this graph an edge exists between service A and service B if there is a least a page of service A that links to a page of service B. Each graph is represented by two files that are written as output by the module: a ".index" and a ".edges" file. The ".index" file contains the id of each graph's node with the corresponding url, while the ".edges" file contains the list of graph's edges represented as id pairs.

3.3 Usage

To manage the execution of the core application we provide three scripts, which are used to start, stop and reload the configuration file. Moreover, the core application handles the following POSIX signals: SIGTERM, SIGHUP and SIGINT. If the application receives a SIGINT or SIGTERM it activates the stop procedure, if it receives a SIGHUP it schedules the reload of configuration file. The other two modules, namely the Graph Builder and the Scrapy Spiders, have to be started separately. The user can decide to use any subset of modules and operations provided by the toolkit. In particular, the Graph Builder should be activated only at the end of the crawling procedure, whereas the Scrapy Spiders can be activated whenever needed. For Scrapy Spiders the only requirements are: the activation of the core application and the activation of the extraction-from-file modality. In case these requirements are not met, data collected by spiders won't be extracted, analysed and stored in the database by the core application.

4 Testing the Toolkit

To assess the performance and the potential of our toolkit, we tested its ability to mine the contents and structure of the Tor Web. In this section, we report relevant findings of this preliminary, yet wide, exploration run. More details can be found in [23], where we thoroughly investigate the topology of the Tor Web graph, the semantics of extracted texts, and their mutual correlation. In the following, we first briefly recap what the Tor Web is and how it works, in order to both motivate and contextualize the successive analysis.

4.1 Tor Principles

The expression *Tor Web* refers to the network made of all Tor's hidden services, connected through hyperlinks. This network is not to be confused with the network of Tor's *relays*, *i.e.* routers managed by volunteers over the Internet upon which Tor's anonymity relies, through multiple layers of encryption that are stripped off one by one along the route that connects source to destination [11]. To contact a hidden service, a client needs to obtain the service's *onion url* (a 80-bit excerpt of the SHA-1 hash digest of its public key), which is used to download a signed *descriptor*, containing the service's public key and a list of *introduction points*. Communication between client and hidden service takes place through secure circuits to a commonly known relay, known as *rendezvous point*. The rendezvous point is chosen by the client and communicated to the hidden service via one of the introduction points. The security of Tor relies on cryptography at three different levels: (i) encryption for data privacy within Tor, (ii) authentication between clients and relays, and (iii) integrity/authenticity of the list of relays, stored by special nodes of the network called *directory authorities*, endowed with their own directory signing key.

Developing suitable tools to explore and analyse the Tor Web is of primary importance for a number of reasons. First, searching through the Tor Web is not possible using standard approaches, since

[15] https://github.com/lexborisov/myhtml

Tor's hidden services are not indexed by traditional search engines like Google or Bing. Specific Tor search engines exist, accessible through the surface Web (*e.g.* Ahmia [16], Onion.link [17]), or through Tor only (*e.g.* DuckDuckGo [18], TORCH [19]), but they are not likewise reliable, mostly because Tor is very volatile and not as connected as the surface Web. Even the size of the Tor Web can be hardly estimated, despite a few crawling attempts [16, 15, 25]. Although the analysis of the surface Web graph has flourished in the past, to the best of our knowledge no similar result/work for the Tor Web exists, and consequently little or no information over the structure of the Tor hidden services network is known to date. A few attempts at classifying the content of Tor hidden services can be found in the literature [16, 15, 25], but each presents limitations related to either the scale or the scope of the crawling, or to the techniques – mostly, topic model-based – used to mine the collected textual data. Both novel instruments and further studies are needed to fully characterize the Tor Web and to collect as much information as possible on its structure and content, which is instrumental in reaching a clear understanding of its functioning and of the habits of its users (hidden service's owners and clients).

4.2 Data Collection

As shown by past attempts [30, 15], exploring the entire Tor Web is not feasible/practical for a number of reasons: it can be unstructured, or volatile/temporary; furthermore, relaying timeouts often occur. Additionally, many hidden services could deliberately try to limit their visibility (*e.g.* by not advertising themselves). We therefore argue that a complete scan of the Tor Web is only possible supporting the crawler with some sort of brute force searcher, able to continuously feed the url list of the crawler with new hidden services found by participating to Tor (*e.g.* to collect hidden service descriptors), or by trying to access random onion urls. We followed a viable alternative, used in related studies [25], consisting in looking at the portion of the Tor Web that is accessible from the surface Web. Specifically, our root set was composed of a (large) set of hidden services obtained merging together results of both standard (*e.g.* Google) and Tor-specific (*e.g.* Ahmia) search engines, other than onion urls advertised on a few Tor wikis and link directories, like "The Hidden Wiki" page [20]. Notably, our root set contains the complete list of hidden services indexed by Ahmia at the time our crawling activity started. This choice can be read as follows: we focus on the portion of the Tor Web that is accessible to "common" users [21], only leaving out the most isolated and (probably) most volatile hidden services (not very interesting, at least from a structural point of view) and thus measuring the size of the "public" part of the Tor Web.

We ran the crawler for about six weeks. Since preliminary tests showed that the latency of the Tor network is much higher than expected, we set the connection and socket timeout to 360 seconds. At the end of the process our dataset contained 1119048 records, thus the crawler tried to connect to 1119048 urls. However, the actually visited urls were 918885, 824324 of which replied with a success HTTP status code (200), whereas the other 94561 replied with a 3xx status code [22]. All remaining connection attempts ended with an error code: 96816 urls replied with a 4xx status code, and 103347 urls with a 5xx status code. The numbers of our crawling are summarized in Table 2. Let us underline that these numbers do support the quality of both our root set and the crawling process: the amount of hidden services we were able to analyse is comparable to those used in previous similar attempts in the literature [16, 15].

4.3 Structure Mining

By feeding the Graph Builder module of our platform with all data gathered in the crawling phase, we were able to reconstruct the graph of the explored portion of the Tor Web. The outer border of the Tor network, *i.e.* pages of the surface Web that link to or are linked by Tor pages, is not reached

[16] https://ahmia.fi

[17] https://onion.link

[18] http://3g2upl4pq6kufc4m.onion

[19] http://xmh57jrzrnw6insl.onion

[20] wikitjerrta4qgz4.onion

[21] Our notion of a common user includes even advanced users that know how to run a crawler, but not users who are given the url of a hidden service by the owner of that service.

[22] A status code 3xx is related to Web redirection (`https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`).

Table 2: Crawling results

Number of pages	918885
Number of domains/subdomains	5420
Number of hidden services	5144
Total number of hyperlinks	20446513
Number of bidirectional hyperlinks	2483366

Table 3: The host graphs

Graph	**Nodes**	**Edges**	**CCs (GCS)**		**Diameter**	**Max Degree**	
Undirected	5420	64379	1 (5420)		4	4759	
Directed	5420	65716	WCCs 1 (5420)	SCCs 4514 (670)	∞	In-Degree 303	Out-Degree 4751

(W/S)CCs = (Weakly/Strongly) Connected Components – GCS = Giant Component Size

by the crawler, and is therefore not included in the graph (but this behaviour can be easily adjusted modifying the configuration file of the crawler). Quite naturally, it is possible to define three different graphs: (i) a *page* graph, whose nodes are visited pages and whose edges are hyperlinks between pages; (ii) a *host* graph (HG), where pages belonging to the same host (*i.e.* the same Tor domain or subdomain) are grouped into a single node, and all hyperlinks among pages of two different groups collapse into a single edge; (iii) a *service* graph, analogous to the host graph, except that grouping occurs at the hidden service level. For our structural analysis, we focused on the host graph, which we believe better synthesizes the connectivity and navigability of the Tor Web. We considered both its *directed* (the direction of each edge is of course induced by the corresponding hyperlink) and its *undirected* version, which exhibits significant differences. Table 3 summarizes the characteristics of the two graphs.

Figures 2 and 3 provide a snap-shot of the in- and out-degree distributions of the directed host graph, focusing, for the sake of clarity, on the most relevant parts of the plots. From Figure 2 we see that, despite the two distributions having a somewhat similar qualitative behaviour, the out-degree is significantly more squashed at 0. Specifically, $\sim 72\%$ of the hosts have out-degree 0, *i.e.*, no outgoing hyperlinks, and adding up hosts with out-degree 1 we reach $\sim 90\%$ of the whole graph, while hosts with out-degree 2 or 3 are one order of magnitude rarer than those with out-degree 1. On the other hand, having in-degree 0 or 1 is far less likely for a host, which is reasonable if we think of how the host graph is built based on the page graph. The distance between the two distributions is notable up to degree ~ 30; after that the two trends become very comparable. Overall, Figure 2 suggests that most hosts have only a few inbound connections, and do not provide any link to any other host. This is further confirmed by the joint in-out-degree distribution, depicted in Figure 3, that shows that almost half of the hosts concurrently have out-degree 0 and in-degree ≤ 10. Among other things, these figures underline the importance that specific hosts (most likely, link directories) have in the topology of the Tor Web graph, providing links to a very large number of other hosts with very small in-degree and no outgoing edges, in a *star*-like structure.

If we consider the volatility of the Tor network, and that even the surface Web has been shown to present similar properties [1], it is not surprising that the network of Tor hosts is very disconnected if represented as a *directed* graph. However, in Figure 4 we see that the undirected version of the graph is significantly more connected than its directed counterpart. In particular, zooming in on degrees ≤ 100, we see that no node has degree 0 (in fact, there is only one giant connected component), and that degrees 1, 2 and 3 together account for only $\sim 10\%$ of the whole graph.

Finally, to have a better understanding of the importance of single hosts in the Tor network, in Figure 5 we plot the statistical distribution of the normalized Betweenness Centrality (BC), comparing the directed and undirected host graph. The BC score of a node is proportional to the number of shortest paths passing through it; roughly speaking, it quantifies the importance of that node in the overall connectivity of the graph and in the way information flow through it. Although some slight differences emerge, the overall trend is the same for the two graphs: there are a few very

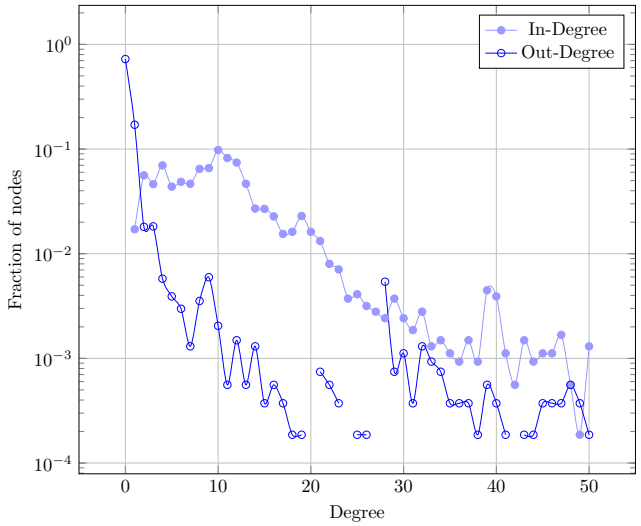

Figure 2: In-degree and out-degree distributions for the directed host graph

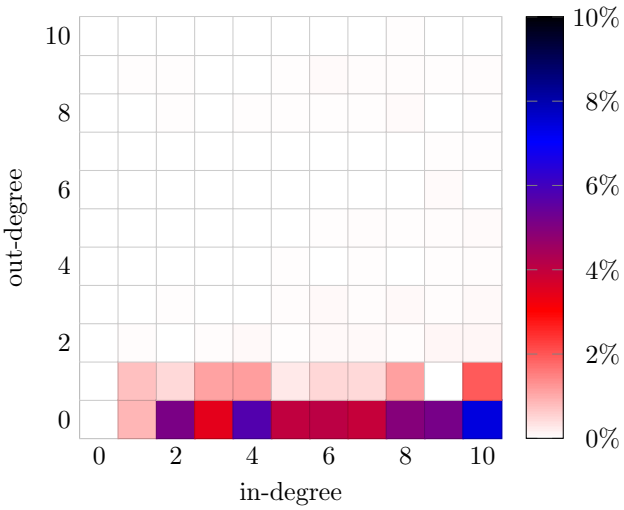

Figure 3: Joint in-out-degree distribution for the directed host graph

central nodes, surrounded by a vast majority of peripheral nodes. For the computation of the BC we resorted to a multi-GPU implementation [31, 32].

4.4 Content Mining

In order to evaluate the content of all collected data, we made use of the Cogito semantic engine, that can be integrated into our platform thanks to its RESTful APIs, and used to analyse all textual data collected [23]. Cogito's ability to understand what a text is about relies on a customizable semantic network called *Sensigrafo*, and on a disambiguation engine called *Essex*. Both modules strongly depend on the specific taxonomy used to model concepts of interest. In our case, the taxonomy was defined within the scope of the "IANCIS" ISEC Project, with categories chosen so as to cover a wide range of topics, including, but not limited to, illegal activities often associated with Tor. These categories, together with an overall score that quantifies their measured relevance in our dataset, are reported in Table 4. Specifically, in our framework Cogito was configured to return, for each page p, a value $S_p[i] \in [0, 1]$ that quantifies to which extent the content of p can be associated to the

[23] We focused on English text, but Cogito's taxonomy can be extended to other languages.

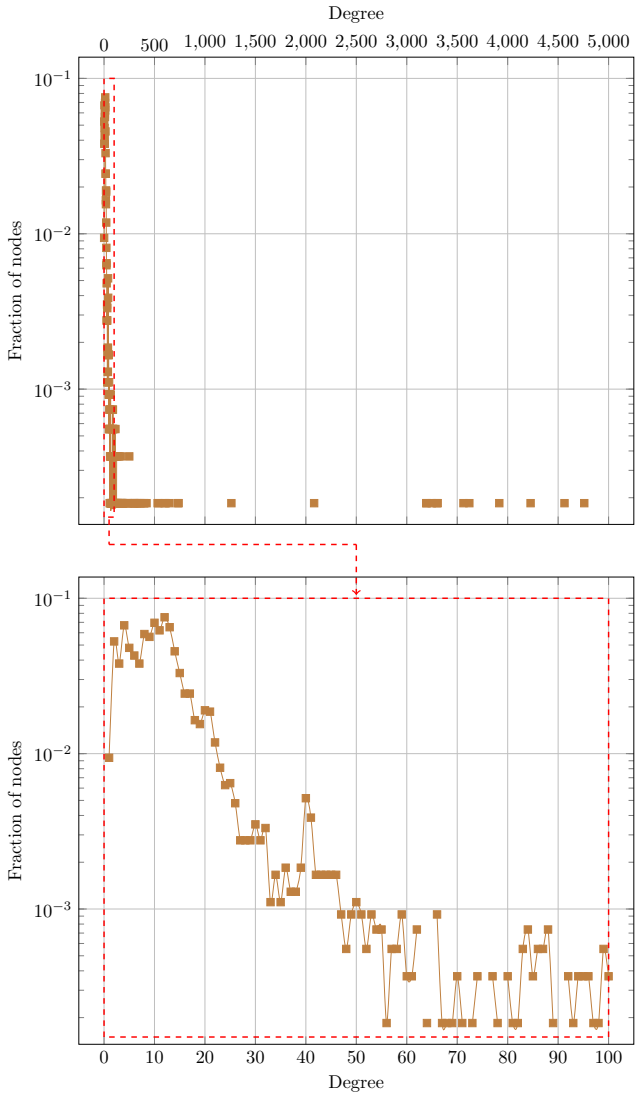

Figure 4: Degree distribution for the undirected host graph

i^{th} category of our taxonomy. Putting together scores $S_p[i]$ for all i yields a *semantic vector* S_p of size n ($n = 17$ in our case) that describes the "position" of page p with respect to the taxonomy. The overall score of category i reported in Table 4 is obtained averaging $S_p[i]$ over all pages p whose semantic vector is not all null, *i.e.* whose content is somewhat relevant with respect to at least one category of our taxonomy. What emerges from Table 4 is that "Information System" is by far the most relevant category in the dataset, probably due to its wide scope, followed by "Social Network", which captures the abundance of forums and markets where Tor users interact. Not surprisingly, topics related with cyber criminality are also significantly relevant, as well as "Illicit Drugs" and "Terrorism". Conversely, "Child Pornography" related content appeared in our dataset less frequently than one could imagine, based on previous works.

Finally, the content of two pages p_1 and p_2 can be compared based on the *cosine similarity* of the two corresponding semantic vectors, $cosine(S_{p_1}, S_{p_2})$, defined as the cosine of the angle between S_{p_1} and S_{p_2}. If $cosine(S_{p_1}, S_{p_2}) = 0$ the two vectors are orthogonal and the two pages have nothing in common. Conversely, if $cosine(S_{p_1}, S_{p_2}) = 1$ the two pages can be considered fully equivalent, at least with respect to our taxonomy. We computed the pairwise cosine similarity of the semantic vectors associated to all pages of our dataset with the goal of assessing their semantic uniformity. Figure 6 shows the statistical distribution of the cosine similarity between any two pages of our

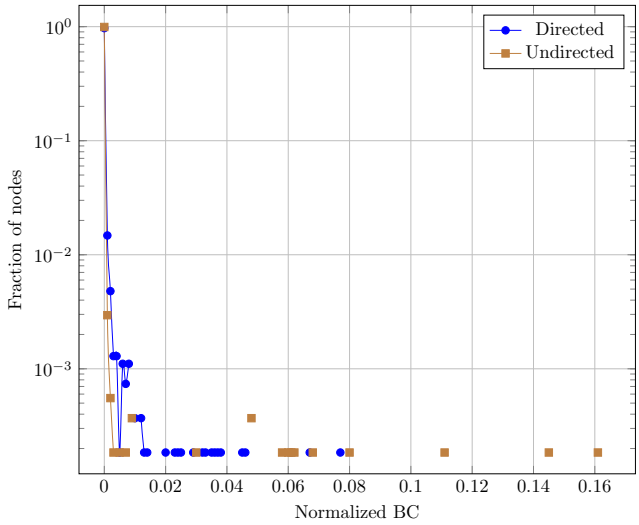

Figure 5: Normalized BC distribution for the host graphs

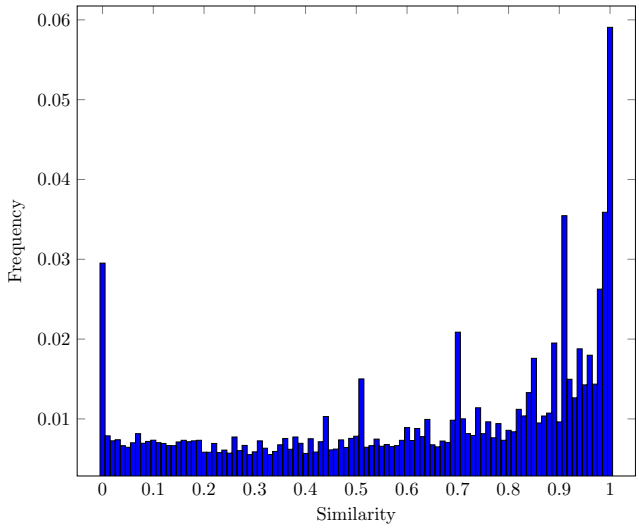

Figure 6: Statistical distribution of the cosine similarity among pages of our crawl

crawling. In most cases $cosine(S_{p_1}, S_{p_2}) \approx 1$ or $cosine(S_{p_1}, S_{p_2}) = 0$, which suggests that most semantic vectors are significantly unbalanced (*i.e.* the corresponding page can be associated with only a small subset of the categories considered), so that any two vectors are either very similar or completely unrelated. This fosters the intuition that the Tor Web is very topic-oriented, with most hidden services focusing on a specific topic, and only a few hubs, mostly marketplaces, forums, wikis, or link directories, that can be related to several different topics.

5 Conclusions

In this paper, we presented a novel Web mining toolkit, designed with the aim of providing a wide-scope, flexible, easy-to-use instrument to thoroughly explore (portions of) the Web, collecting and analysing data, and reconstructing the graph structure of the crawled network. Motivated by the increasing attention paid by both researchers and society to submerged and anonymous Web servers and resources, we tailored our toolkit to operate on the Tor dark Web, but we left full control to the user for configuring/adjusting/modifying its behaviour.

Table 4: Relevance scores of the categories of our taxonomy in crawled pages

Category	Score
Information System	0.65872592
Social Network	0.16532867
Cyber Security	0.13046941
Illicit Drugs	0.09255990
Terrorism	0.07622640
Weaponary	0.05358337
Cyber Deception	0.05026633
Fraud	0.03810798
Cyber Attack	0.03700734
Murder	0.01545559
Child Pornography	0.01166300
Rape	0.00722616
Racism	0.00467250
IT Services Companies	0.00149265
Media Companies	0.00121687
Arms Trafficking	0.00066599
Gambling	0.00024497

Other than describing in details our design and implementation choices, we reported the results of a test run executed over three months, exploring the portion of the Tor Web that can be reached starting from the surface Web. Our findings, analysed more in depth in [23], provide a fascinating portrait of the world's most famous darknet, clarifying the effectiveness and potential of our toolkit. By making our suite available on demand, we expect to pave the way for further research able to provide meaningful insights on the characteristics of the most hidden and intriguing sides of the Web.

The only element of our toolkit that we cannot release is the semantic engine used in our test run, a proprietary software. Even though our analyser can interact with any engine providing RESTful APIs, implementing our own text mining tool to replace Cogito is the primary direction for future extensions of our toolkit. Luckily, the literature flourishes with compelling ideas [33, 34] which can be developed into algorithms and applications for addressing communal desiderata of Web content mining, such as extracting topics/clusters and assessing document similarities.

References

[1] JonM. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and AndrewS. Tomkins. The web as a graph: Measurements, models, and methods. In Takano Asano, Hideki Imai, D.T. Lee, Shin-ichi Nakano, and Takeshi Tokuyama, editors, *Computing and Combinatorics*, volume 1627 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 1999.

[2] Raymond Kosala and Hendrik Blockeel. Web mining research: A survey. *SIGKDD Explor. Newsl.*, 2(1):1–15, June 2000.

[3] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans M. Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35:66–71, 2002.

[4] Soumen Chakrabarti, Amit Pathak, and Manish Gupta. Index design and query processing for graph conductance search. *The VLDB Journal*, 20(3):445–470, June 2011.

[5] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. Web data extraction, applications and techniques: A survey. *Knowledge-Based Systems*, 70:301 – 323, 2014.

[6] Paolo Boldi and Sebastiano Vigna. The webgraph framework i: Compression techniques. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 595–602, New York, NY, USA, 2004. ACM.

[7] Francisco Claude and Gonzalo Navarro. Fast and compact web graph representations. *ACM Trans. Web*, 4(4):16:1–16:31, September 2010.

[8] Francisco Claude and Susana Ladra. Practical representations for web and social graphs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 1185–1190, New York, NY, USA, 2011. ACM.

[9] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D Sivakumar, Andrew Tomkins, and Eli Upfal. Stochastic models for the web graph. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 57–65, 2000.

[10] Anthony Bonato. A survey of models of the web graph. In Alejandro López-Ortiz and AngèleM. Hamel, editors, *Combinatorial and Algorithmic Aspects of Networking*, volume 3405 of *Lecture Notes in Computer Science*, pages 159–172. Springer Berlin Heidelberg, 2005.

[11] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Usenix Security Symposium*, 2004.

[12] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. Stegotorus: A camouflage proxy for the tor anonymity system. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 109–120, New York, NY, USA, 2012. ACM.

[13] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for tor hidden services: Detection, measurement, deanonymization. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 80–94, Washington, DC, USA, 2013. IEEE Computer Society.

[14] Daniel Arp, Fabian Yamaguchi, and Konrad Rieck. Torben: A practical side-channel attack for deanonymizing tor communication. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 597–602, New York, NY, USA, 2015. ACM.

[15] Alex Biryukov, Ivan Pustogarov, Fabrice Thill, and Ralf-Philipp Weinmann. Content and popularity analysis of tor hidden services. In *Distributed Computing Systems Workshops (ICDCSW), 2014 IEEE 34th International Conference on*, pages 188–193, June 2014.

[16] Martijn Spitters, Stefan Verbruggen, and Mark van Staalduinen. Towards a comprehensive insight into the thematic organization of the tor hidden services. In *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint*, pages 220–223, Sept 2014.

[17] Monica J. Barrat. Silk road: Ebay for drugs. *Addiction*, 107(3):683–683, 2012.

[18] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice and Experience*, 34(8):711–726, 2004.

[19] Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. Bubing: Massive crawling for the masses. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, pages 227–228, 2014.

[20] Allan Heydon and Marc Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

[21] Gordon Mohr, Michael Stack, Igor Ranitovic, Dan Avery, and Michele Kimpton. An introduction to heritrix an open source archival quality web crawler. In *In IWAW'04, 4th International Web Archiving Workshop*. Citeseer, 2004.

[22] Christopher Olston, Marc Najork, et al. Web crawling. *Foundations and Trends® in Information Retrieval*, 4(3):175–246, 2010.

[23] Massimo Bernaschi, Alessandro Celestini, Stefano Guarino, and Flavio Lombardi. Exploring and analyzing the tor hidden services graph. *ACM Transactions on the Web*, page To appear, 2017.

[24] Gareth Owen and Nick Savage. Empirical analysis of tor hidden services. *IET Information Security*, 10(3):113–118, 2016.

[25] Kyle Soska and Nicolas Christin. Measuring the longitudinal evolution of the online anonymous marketplace ecosystem. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, pages 33–48, 2015.

[26] Olivier Blanvillain, Nikos Kasioumis, and Vangelis Banos. Blogforever crawler: techniques and algorithms to harvest modern weblogs. In *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*, page 7. ACM, 2014.

[27] Feng Zhao, Jingyu Zhou, Chang Nie, Heqing Huang, and Hai Jin. Smartcrawler: A two-stage crawler for efficiently harvesting deep-web interfaces. *IEEE Transactions on Services Computing*, 9(4):608–620, 2016.

[28] Matko Bošnjak, Eduardo Oliveira, José Martins, Eduarda Mendes Rodrigues, and Luís Sarmento. Twitterecho: a distributed focused crawler to support open research with twitter data. In *Proceedings of the 21st International Conference on World Wide Web*, pages 1233–1240. ACM, 2012.

[29] Rohit Khare, Doug Cutting, Kragen Sitaker, and Adam Rifkin. Nutch: A flexible and scalable open-source web search engine. *Oregon State University*, 1:32–32, 2004.

[30] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining light in dark places: Understanding the tor network. In Nikita Borisov and Ian Goldberg, editors, *Privacy Enhancing Technologies*, volume 5134 of *Lecture Notes in Computer Science*, pages 63–76. Springer Berlin Heidelberg, 2008.

[31] Flavio Vella, Giancarlo Carbone, and Massimo Bernaschi. Algorithms and heuristics for scalable betweenness centrality computation on multi-gpu systems. *CoRR*, abs/1602.00963, 2016.

[32] Massimo Bernaschi, Giancarlo Carbone, and Flavio Vella. Scalable betweenness centrality on multi-gpu systems. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '16, pages 29–36, New York, NY, USA, 2016. ACM.

[33] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[34] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.