6

WEB SCRAPING AND CRAWLING

LEARNING OBJECTIVES

The goals of Chapter 6 are to help you to do the following:

1. Define the main techniques for **web crawling**.
2. Explore available software packages for automatically collecting textual data from webpages,
3. Compare web crawling and **web scraping** techniques.
4. Compare tools and supporting material available for web crawling and scraping techniques.

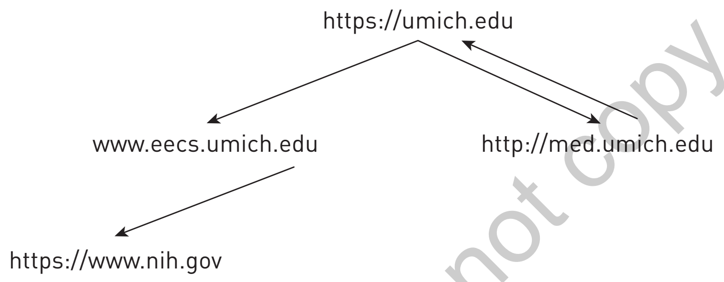
INTRODUCTION

In this chapter, we survey two categories of tools that are critically important for acquiring large amounts of digital textual data: web scraping and web crawling. Both can be accomplished with off-the-shelf software or in programming environments like Python or R. Crawling and scraping can potentially save huge amounts of time if the alternative is to manually scrape data from webpages. But they require an investment of time to learn how to use them, and this requires some background knowledge about the structure of the World Wide Web.

The web—a common abbreviation for the World Wide Web—consists of billions of interlinked hypertext pages. These pages contain text, images, videos, or sounds and are usually viewed using web browsers such as Firefox or Internet Explorer. Users can navigate the web either by directly typing the address of a webpage (the **URL**) inside a browser or by following the links that connect webpages between them.

The web can be visualized as a typical example of a graph, with webpages corresponding to vertices in the graph and links between pages corresponding to directed edges. For instance, if the page https://umich.edu includes a link to the page www.eecs.umich.edu and one to the page http://med.umich.edu and the later page, in turn, links to the page of the National Institutes of Health (https://www.nih.gov) and back to the https://umich.edu page, it means that these four pages form a subgraph of four vertices with four edges, as is illustrated in Figure 6.1.

FIGURE 6.1 ■ Sample Web Graph



In addition to "traditional" webpages, which account for a large fraction of the data that we currently find online, today's web also includes a number of other data sources, such as sites with user-contributed content (e.g., Wikipedia, *HuffPost*), social media sites (e.g., Twitter, Facebook, Blogger), deep web data (e.g., data stored in online databases such as https://www.data.gov), or e-mail (e.g., Gmail, Hotmail). While some of these sources may not be publicly available (for instance, e-mail is by definition private, and so are a number of Facebook profiles), they still represent data in digital format that account for online traffic.

There are many challenges that come with the use of web data, many of which are highlighted throughout this book. In addition, there are also challenges that are associated with crawling such data, which we address in this chapter.

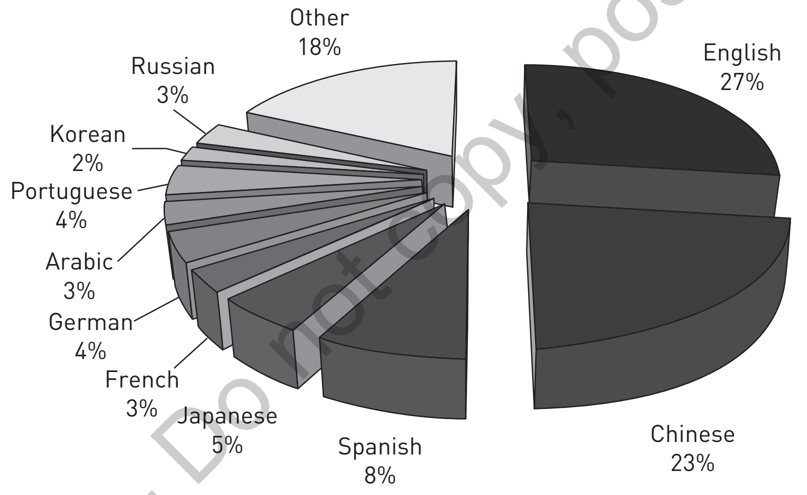
WEB STATISTICS

While the size of the web is generally considered to be unknown, there are various estimates concerning the size of the indexed web—that is, the subset of the web that is covered by search engines. Web statistics compiled in 2014 by www.geekwire.com suggested 5 million terabytes of data online, out of which approximately 20% is textual data.

The web is estimated to include more than 600 million web servers and 2.4 billion web users, with about 1 billion Facebook users and 200 million Twitter users (http://royal.pingdom.com/2013/01/16/internet-2012-in-numbers). Estimates of the number of e-mails come from www.radicati.com, which suggests that 154 billion e-mails are sent daily, of which more than 60% are spam.

An interesting statistic refers to the proportion of languages used on the web. Information collected by www.internetworldstats.com in 2015 showed the distribution of language use as illustrated in Figure 6.2.

FIGURE 6.2 ■ Top Ten Languages on the Web in 2015



Source: Internet World Stats, http://www.internetworldstats.com/stats7.htm

WEB CRAWLING

Web crawling is the process of building a collection of webpages by starting with an initial set of URLs (or links) and recursively traversing the corresponding pages to find additional links. A collection built this way can be used, for instance, to perform text mining (see Chapter 15), opinion analysis (see Chapter 14), text classification (see Chapter 13), or any other process that requires textual data.

Processing Steps in Web Crawling

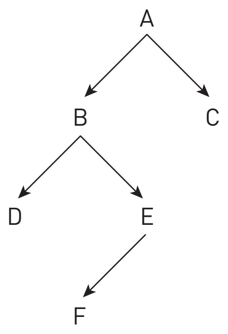
A crawler typically performs the following steps:

1. The crawler creates and maintains a list of URLs to be processed. This list is initially seeded with some manually selected URLs, and it is then iteratively grown into a large set of URLs.
2. The crawler selects a URL from the list (see below for selection strategies), marks it as "crawled," and it fetches the webpage from that URL. The page is processed, and links and content are extracted. This processing can be as simple as just extracting links using a regular expression (regex) that matches all the occurrences of tags such as <a href="http://. . . .">, followed by removal of all the HTML tags to obtain the content of the page. At times, a more sophisticated processing may be required—for instance, when the links also include relative links or links to fragments or when the content of a page includes entire sections devoted to advertisements or other content that needs to be removed.
3. If the content has already been seen, it is discarded. If not, it is added to the collection of webpages to be further processed (e.g., indexed, classified).
4. For each URL in the new set of URLs identified on the page, a verification is made to ensure that the URL has not been seen before, that the page exists, and can be crawled. If all these filters are passed, the URL is added to the list of URLs at Step 1, and the crawler goes back to Step 2.

Traversal Strategies

An important aspect of any crawler is its web **traversal strategies**. As mentioned before, the web is a graph and therefore different graph traversal algorithms can be applied. One way of traversing the web is called breadth-first, where, given one webpage, we first collect and process all the pages that can be reached from URLs on that page before we move on to other pages. The second way of traversing the web is called depth-first, where, given one webpage, we extract one URL from that page, collect and process the page that can be reached from that one URL, extract again one URL on the page we just processed, and so on until we reach a dead end. Only then do we backtrack and process additional URLs on the pages we have just visited. For instance, Figure 6.3 shows a simple web graph, along with the order of page traversal, starting with Page A, for each of these two strategies.

FIGURE 6.3 ■ Web Traversal Strategies



Breadth-first traversal order: A, B, C, D, E, F
Depth-first traversal order: A, B, D, E, F, C

Crawler Politeness

Most websites have a clear crawling policy which states what **crawlers** can or cannot traverse them and which parts of the site can be crawled. There are two main ways of indicating a crawling policy. The most commonly used one is robots.txt, which is a file that is placed at the root of the website (e.g., www.cnn.com/robots.txt). This file can include a list of crawlers (or agents) that are disallowed for specific portions of the site. For instance, the following content of robots.txt indicates that all the crawlers are disallowed from traversing pages found under/tmp or /cgi-bin; the exception is BadBot, which is disallowed from the entire site:

```
User-agent: *
Disallow: /tmp/
Disallow: /cgi-bin/
User-agent: BadBot
Disallow: /
```

Another way of providing a crawling policy is through meta tags included in the HTML of individual pages. There is a special meta tag called robots, which can be used with combinations of values for two aspects: index or noindex (allow or disallow this webpage to be crawled) and follow or nofollow (allow or disallow the crawler to follow links on this webpage). For instance, a webpage could have a robots meta tag as follows:

<meta name="robots" content="index,nofollow">

It states that this page can be crawled, but links on the page cannot be followed.

WEB SCRAPING

There are several available Internet-based methods for collecting ("scraping") document collections for social science research. There are several traditional methods of acquiring text data for research, including transcribing interviews (e.g., Bamberg, 2004) and downloading text files from digital archives, including news archives such as LexisNexis (https://www.lexisnexis.com) and Access World News (www.newsbank.com/libraries/schools/solutions/us-international/access-world-news) and digital archives of historical documents (Jockers & Mimno, 2013). As there are a number of useful resources available to help with organizing interview transcripts and with finding and working with digital archives, in this chapter we focus only on newer Internet-based methods for creating corpora from scratch. The first of these methods is known as web scraping, which involves using commercial software and, as needed, programming languages such as Python to identify and download text from one or more pages or archives within a single website. Web crawling involves using programming languages to identify and download text from a large number of websites. While both web scraping and web crawling are powerful tools for data acquisition, they should ideally be used only within a project with a logical and practical research design (see Chapter 5).

SOFTWARE FOR WEB CRAWLING AND SCRAPING

While researchers with programming backgrounds will generally prefer to use Python or another programming language over commercial web scraping software, commercial scraping software is reasonably easy to use for nonprogrammers, and it may at times have technical advantages over Python and other languages used for web scraping.

Web scraping software is designed to recognize different types of content within a website and to acquire and store only the types of content specified by the user. So, for instance, web scraping software allows a user to search a newspaper website and save only the names of article authors or to search a real estate website and save only the prices, addresses, or descriptions of listed properties. The software works by running *scripts* written by the user. The scripts tell the software on which webpage to start, what kind of text to look for (e.g., text with a certain font size or formatting), what to do with the text that is found, where to navigate next once text is saved, and how many times to repeat the script. Saved text data can be downloaded in a convenient file form such as a comma-separated values (CSV) file or a Microsoft Excel spreadsheet. Although web scraping software is very powerful and relatively user friendly, sites with complex structures often lead scraping software to stall or freeze. Luckily there are many videos and online support forums available for software users.

There are several useful freeware and commercial software products available on the market, and instructional videos for most of these are easy to find on YouTube and other Internet video services or in the manual pages available on Linux via the *man* command. In our own research, we have used *Lynx*, which is a very simple Linux-based command-line browser that allows for automatic processing of a page, including link and content extraction, and the Linux command *wget,* which allows for recursive crawling with a prespecified depth of the crawl. We have also used *Helium Scraper* (www.heliumscraper.com).

SOFTWARE AND DATA SETS FOR WEB SCRAPING AND CRAWLING

Helium Scraper is an affordable and convenient scraping package with good online and YouTube support (www.heliumscraper.com).

Outwit Hub is another scraper similar in concept to Helium (https://www.outwit.com/products/hub).

FMiner is a scraping package that includes some advanced data extraction features (www.fminer.com).

Mozenda is a comprehensive cloud-based package of scraping tools designed for business applications (www.mozenda.com).

RapidMiner is a "data science platform" with web scraping tools (https://rapidminer.com).

Visual Web Ripper is an affordable, dedicated web scraping package (www.visualwebripper.com).

Import.io offers a powerful suite of data extraction tools (https://www.import.io).

Beautiful Soup is a Python library for extracting data out of HTML files (https://pypi.python.org/pypi/beautifulsoup4).

Lynx and wget are online commands available in almost any Unix/Linux environment, which can be used for direct download of webpages.

Conclusion

Web scraping and crawling allows for direct access to the raw data available on websites. While scraping typically targets the collection of the text from a page or a site, crawling also involves the identification of new pages to scrape, for the purpose of, for example, building a large collection of web data. This chapter briefly overviewed the main approaches for web scraping and crawling and also gave pointers to a number of programming or off-the-shelf tools that can be used to facilitate this process.

Key Terms

Crawlers 79	Traversal strategies 78	Web crawling 75
Scrapers 82	URL 76	Web scraping 75

Highlights

- Web crawling is the process of building a collection of webpages by starting with an initial set of URLs or links and recursively traversing the corresponding pages to find additional links.
- Web scraping generally involves using commercial software and, as needed, programming languages such as Python to identify and download text from one or more pages or archives within a single website.

Discussion Questions

- How do researchers choose whether to use a web crawler or a web scraper?
- What are some challenges in using crawlers and **scrapers** on websites with complex structures, such as many interlinked pages, graphics, and video?
- Considering your plans for your own research project (its topic, research questions, and at least a preliminary research design), what is the ideal tool for acquiring the data you need? More realistically, given real-world time and resource constraints, what is the best available web crawling or web scraping tool?