Snippet Matching for Click-Tracking Blocking

User Privacy for Search Engines

Author	Supervisor
Yana Kulizhskaya	Douglas Leith

School of Computer Science & Statistics
Trinity College, Dublin

A dissertation submitted to the University of Dublin, Trinity College in accordance with the requirements of the degree of Masters in Computer Science in the School of Computer Science & Statistics.

May 2017

Author's declaration

I, Yana Kulizhskaya, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

SIGNED: .... DATE: ....

Summary

User privacy on the web is a growing concern as more people make online activities a daily part of their lives. Search engines play a key role in enabling internet users to discover content and information on the web. With that, comes a fear that modern search engines know too much about their users and that they readily share this information with anyone that is willing to pay for it. While there is a wide range of privacy concerns associated with search engine use, this dissertation focuses on click-tracking. Click-tracking consists of search engines monitoring which web pages a user visits in order to allow for more effective targeted advertising. As it currently stands, a very limited amount of research has gone into prevention of click-tracking by search engines specifically.

This dissertation proposes the idea of search engine click-tracking blocking by a means of a URL resolution service. The URL resolution service is a third party service that would be able to resolve search result URLs, which is something that is currently done by the search engine itself and is what allows search engines to track user clicks. The dissertation acts as a stepping stone in the area of click-tracking blocking research and investigates the viability of the use of search result snippets to effectively infer search result URLs in a click-tracking blocking system. The dissertation consists of four main objectives: search result data collection, search result data analysis, snippet matching implementation for URL resolution and snippet matching evaluation.

The first step of the dissertation was to collect a search result dataset that was used to answer the main dissertation question of whether search result snippets can be used to infer search result URLs. Since no such dataset already existed, the Google search engine was scraped using publicly available datasets of real search engine queries. The scraped search result HTML pages were then parsed to extract the snippet information. A number of different types of search result data was collected to answer various research questions. Data was scraped from three geographical locations, the U.S., Australia and Ireland allowing for a more realistic dataset. The data collected in Ireland was scraped repeatedly over a period of five months to investigate the stability of snippets over time.

Once the search snippet dataset was collected, a snippet matching algorithm was implemented to investigate whether a query snippet could be correctly matched to the respective snippet in the snippet-URL repository. The collected snippets underwent a feature extraction process, consisting of stemming, stop word removal and tokenisation, for more efficient processing. The query snippet was matched to one of the snippet-URL pairs in the repository using the cosine similarity metric. In order to speed up the URL resolution of snippet queries, the algorithm was rewritten to be run on GPU cores in parallel. As part of the parallelisation process, the memory layout of the data was optimised and the algorithm code was restructured to better suit the GPU programming model.

The snippet matching algorithm was first evaluated on its error rate. The error rate for the entire dataset was relatively small, at 3.48%. The reason for such a low error rate was because the text of the collected snippets consisted of a highly diverse content, with almost half of all of the snippet features appearing once in the dataset. A large portion of the errors (approx. 70%) occurred because different URLs had identical search result snippets. In these cases, the difference in URLs was most commonly attributed to geographical location and user-specific HTTP GET parameters.

The algorithm was then evaluated on its robustness to change, i.e. whether minor modifications to the query snippet resulted in higher error rates. The results of the robustness evaluation indicated that snippet matching was resilient to input changes. The snippet matching algorithm produced error rates of below 10% when up to a third of the query snippet was removed or modified, meaning that snippet matching was effective in cases where the query snippets were updated versions of existing snippets as might be the case when dealing with real world data.

An examination of snippet stability over time was conducted by measuring the similarity of the snippet text taken at different points in time. The similarity of text was calculated using cosine similarity of the snippet word vectors. This examination concluded that snippets do not change significantly over time, with 93% of examined snippets not changing at all or not changing significantly. Most snippet changes occurred in search results that represented social media sites, news websites or sites containing listings such as job postings, for sale property or advertisements.

The overall conclusion of this dissertation is that search result snippets can be used to infer search result URLs with great accuracy, provided the search result is already known to the system. The fact that the snippets do not change significantly over time and that the snippet matching algorithm is resilient to input changes indicates that snippets are a great candidate for use in a URL resolution system, even when data is expected to change over time. However, additional research is needed to fully investigate the viability of the click-tracking blocking system as proposed by this dissertation.

Abstract

This dissertation proposes a solution to the privacy issues regarding user click-tracking by search engines by a means of a URL resolution service. The proposed solution would leverage the search result snippet data to be used for look up of the respective search result URL in a user-collected snippet-URL repository. In order to examine the viability of the proposed solution, the dissertation investigates whether search result snippets can be used to successfully infer search result URLs.

A search result dataset is collected from the Google search engine and a snippet matching algorithm is developed. An examination of the search snippet dataset indicates that snippets consist of a highly diverse content, making them easily identifiable. The error rate evaluation of snippet matching yields an error rate of 3.48%, indicating that search result snippets can be used for URL resolution with great accuracy, provided a priori knowledge of the snippet-URL mapping exists. Additional investigation into snippet stability and the snippet matching algorithm's resilience to modification of input further support the viability of a click-tracking blocking system that employs snippet matching for URL resolution as snippets are mostly stable over time and the matching process is robust to input change.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Dr. Douglas Leith, for his help and guidance during the course of this dissertation.

I would like to acknowledge the endless support of my boyfriend, Răzvan Damachi. His constant motivation and help in discussion and proof-reading of the thesis were invaluable.

Lastly, I would like to say thanks to my friends and family, who have been there for me over the course of the last five years, providing advice and encouragement along the way.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

User privacy on the web is a delicate topic that has been largely overlooked in the last number of years by the general public and the technology corporations alike. Various online businesses such as social media sites and online retailers exploit the personal information they are entrusted with by their users for profit. The collections of personal data these companies possess are often destined for advertisement agencies who use this information for effective marketing and targeted advertising. As it currently stands, web users have little say in how their data is processed and handled once disclosed to a website.

Worryingly, privacy statements of many large online companies remain vague regarding their use of user data. A study by Earp et al. [19] found that websites' privacy policy statements are typically not aligned with user privacy values, with websites placing emphasis on data collection practices and the users' priority being the transfer and sharing of their data with third parties. Additionally, a U.S.-based study by Turow et al. [32] concluded that 66% of the participants did not want their online data to be used for marketing purposes.

1.1 Motivation for Research Topic

When it comes to search engines, the issue of user privacy becomes increasingly more complex and critical. A person's interactions with search engines have the potential to not only provide a wealth of information about their personal identity but also uncover intimate details of their day-to-day life. Popular search engines can be regarded as omniscient observers, able to construct a complete and comprehensive profile for each of their users. Research conducted by Jones et al. [22] showed that simple classifiers trained on user search queries alone were able to predict age with an absolute error of 7 years and gender with 84% accuracy.

Search engines typically collect a wide range of data about their users. Privacy policy statements from the three largest search engines in the U.S. and the U.K. [5] [13], Google, Bing and Yahoo, state that the collected user data ranges from personal information such as names and credit card numbers to IP addresses, search queries and interactions with web content [10] [8] [6]. All three claim that this data is partly used for customisation of content. This claim is generally in line with user expectation, as previous research conducted by Preibusch [27] has shown that 2 out of 3 participants were happy to provide their click data to a search engine in return for an improved service.

1.1.1 Targeted Advertisement

Arguably, a more important reason for collection of personal data, particularly tracking of user clicks, by search engines is for revenue generation. Search engines commonly employ an advertisement model known as Pay-Per-Click (PPC). PPC allows website owners to charge advertisers a flat fee for each click on their ad.

In search engines PPC manifests into two categories of search results: sponsored and organic results. Organic search results are those that the search engine deems to best match the search query, while sponsored results are targeted advertisements relating to the search query. To select the set of sponsored results for a search query, the search engine considers the Cost-Per-Click value an advertiser is willing to pay and the probability of the user clicking on the ad, known as the Click-Through-Rate (CTR).

An advertisement's CTR is the ratio of times a user clicked on the ad vs. the number of times the ad was visible on the webpage. Therefore by tracking who clicked on what, a search engine is not only generating profit through targeted advertisement but also is able to leverage click data to get its users to click on more ads that get displayed.

1.1.2 Search Term Leaks

In addition to user clicks being auctioned off to advertisers, clicking on a search result link can result in leaks of personal information to third parties. Generally, search engines openly share search terms, clicks, users' genders and age groups with their advertisers in order to improve their advertisement campaigns. However, a user's search query can also be leaked through a HTTP header field to parties that have no connections to the search engine.

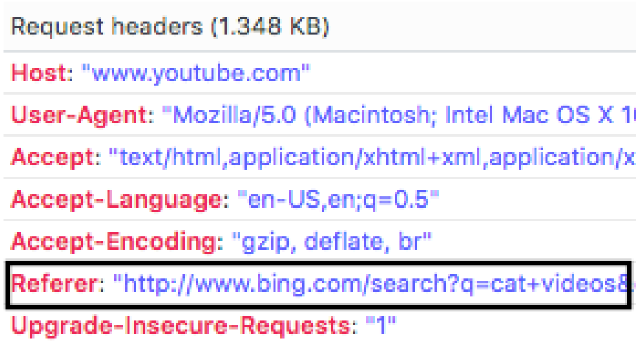
When a user enters a query into a search engine, the results for that query are fetched with a HTTP GET request. This means that the query terms are directly encoded into the search engine's results page URL. Examples of such encodings for Google, Bing and Yahoo are presented in Table 1.1 with the search query, "cat videos", highlighted in bold.

Search Engine	URL
Google	https://www.google.ie/#**q=cat+videos**
Bing	http://www.bing.com/search?**q=cat+videos**
Yahoo	https://ie.search.yahoo.com/search?**p=cat+videos**

TABLE 1.1. Examples of search query URL encoding.

Part of the HTTP request header is a field called 'referer'. Whenever a user navigates to a webpage by following a link on a website, this field is set by the browser to contain the URL of the outgoing webpage. In the context of search engines, when a user clicks on a search result the outgoing URL is the search engine's results page. Thus a user's search terms are shared with the website the user has clicked on. An example of this can be seen in Figure 1.1.

It is worthy to note that the 'referer' header field is only set when both websites use either HTTP or HTTPS, i.e. if a user is leaving a HTTP connection to go to a HTTPS-enabled website or vice versa, no search terms are leaked.



```
Request headers (1.348 KB)
Host: "www.youtube.com"
User-Agent: "Mozilla/5.0 (Macintosh; Intel Mac OS X 1
Accept: "text/html,application/xhtml+xml,application/x
Accept-Language: "en-US,en;q=0.5"
Accept-Encoding: "gzip, deflate, br"
Referer: "http://www.bing.com/search?q=cat+videos&
Upgrade-Insecure-Requests: "1"
```

FIGURE 1.1. Example of the 'referer' HTTP header field with search terms leaked.

In the recent years Google has made attempts to minimise search term leaks. In 2011, Google started blocking search term leaks to organic search results by leaving out HTTP GET parameters from URLs when setting 'referer' fields, however they continued to leak search terms to advertisers [23]. This issue was addressed again in 2015, when Google has stopped leaking search terms in 'referer' fields altogether [29].

However, while Google no longer leaks private data through protocol headers, they continue to allow advertisers to learn about the users that click on an ad via customisable

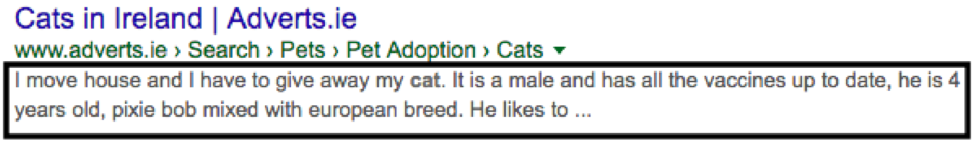
URL parameters [1]. These URL parameters are far more privacy-unfriendly than the HTTP 'referer' fields, as advertisers can now not only capture the search terms, but also details like user location, device type and time of day through search result clicks.

Unlike Google, Bing and Yahoo have yet to take steps towards minimisation of search term leaks.

1.2 Dissertation Objectives

The main goal of the dissertation is to investigate whether it is possible to build a tool to enhance user privacy in the context of search engines by the means of click-tracking blocking. As it currently stands, a feedback loop exists through user clicks in that the user is reliant on the search engine to resolve a search result to a URL. This loop allows search engines to monitor user behaviour and disclose user data to third parties.

A way to disrupt the feedback loop is to defer URL resolution to an independent service that is not operated by the search engine. This dissertation will examine whether it is possible to use text snippets[1] to infer the respective URL by using a repository of user-collected snippet-URL pairs. The overall idea involves a browser extension that would request search result URLs to be resolved on behalf of the user, updating the remote snippet-URL repository with previously unseen mappings.



Cats in Ireland | Adverts.ie

www.adverts.ie › Search › Pets › Pet Adoption › Cats ▾

I move house and I have to give away my cat. It is a male and has all the vaccines up to date, he is 4 years old, pixie bob mixed with european breed. He likes to ...

FIGURE 1.2. Example of a search result snippet taken from Google for search query 'cat'.

The complete set of goals and considerations for the dissertation is outlined below:

1 **Collect a snippet dataset.**

The first goal of the dissertation is to collect a search result snippet dataset as, as far as is known, no such publicly available dataset currently exists.

Since Google operates one of the most popular search engines among the English-speaking populations, the snippet data will be collected from the Google search engine. As search engines could use varying techniques for snippet extraction,

[1] Snippets are short pieces of text that accompany a search result, see Figure 1.2.

it was considered to have been inappropriate to mix data from different search engines.

The data will be collected via automated web-scraping using a collected list of English search queries. While Google snippet data might not be representative of snippet data of other search engines, if an appropriate algorithm for snippet matching is developed, the dissertation research could be extended further to cater for users of other search engines.

Due to ethical reasons and time constraints, it was decided to use existing sources for search query sets instead of collecting search histories of individuals for later scraping. The underlying assumption of this research is that the search queries used in the dissertation resemble the types of search queries posed by real Google users and that the produced search results are in line with organically obtained search result pages.

2 Analyse the snippet dataset.
The second goal of the dissertation is to analyse the collected snippet dataset.

The snippet dataset will play a central role in the dissertation as it will be used to answer the main research question of whether text snippets could be used to infer search result URLs. An analysis of the data characteristics such as diversity and stability over time will provide clues about the best way to model the snippet data as well as an indication for the appropriate algorithmic approach to take for URL inference itself.

In particular, as part of this goal, it will be interesting to see:

- What the average length of a search result snippet is and how many words it contains.
- How diverse the snippet text is.
- Whether snippets remain stable over time or if there are categories of snippets that are more prone to change than others.

3 Develop a snippet matching algorithm.
The third goal is to design and implement an algorithm that is able to return the corresponding URL for a given query snippet. The snippet matching algorithm will work on the basis of matching of the query snippet to one of the snippets-URL pairs it knows about using a similarity metric.

Because web content and search engine implementations are constantly changing, the goal for the snippet matching algorithm is not to match exact snippet strings. Instead, the snippet matching algorithm should allow for some flexibility in the input while maintaining a reasonable error rate. The need for input flexibility might result from a slightly different word order or partially modified query snippet string.

In addition to the algorithm implementation, a feature extraction tool will be developed. In order to achieve computational efficiency for snippet matching, each of the snippet strings will be converted to a feature vector. The feature vectors should be constructed in such a way as to provide a concise, yet descriptive representation of the original string for optimal matching accuracy.

4 **Evaluate the snippet matching algorithm.**
Once the snippet matching algorithm is implemented, the next logical step is to examine its performance. The error rate of the algorithm is of most importance to this dissertation as it would allow to answer the dissertation question of whether search result snippets can be used to infer search result URLs. As part of this evaluation, the errors will also be investigated in a bid to improve the algorithm's performance.

Part of the evaluation will consist of examining of the robustness of the algorithm to change to input, i.e. if the algorithm is still able to infer the correct URL if the query snippet is modified as it might happen if the search engine's snippet extraction algorithm is changed or if the text reflected in the search result snippet is altered.

1.3 Dissertation Structure

In this chapter, the motivations for the research topic and a discussion of some of the existing problems with click-tracking in search engines are presented. A solution to these problems is proposed by a means of URL resolution by a third party service using search result data that is collected by search engine users. The four goals to be achieved during the course of this dissertation are set out. These goals will provide the answer to the dissertation question of whether search result snippets can be used to infer search result URLs.

Chapter 2 describes the state of the art in the field of user privacy in search engines and provides an overview of the tools that have been developed to combat the issues of click-tracking.

Chapter 3 outlines the techniques undertaken to collect the snippet dataset that is used throughout the dissertation. The chapter describes the sources of the search query datasets used for web scraping the search results. This chapter also provides an analysis of the dataset with a particular focus on snippet diversity.

Chapter 4 presents the feature extraction method and the resulting search snippet model. An overview of the snippet matching algorithm is provided, along with the text similarity metrics explored as part of the snippet matching process. A discussion of GPU parallelisation, the optimisation opportunities of the snippet matching code and their implementations follow.

In Chapter 5, the performance of the snippet matching algorithm is presented. The performance evaluation is performed based on the algorithm's error rate and its robustness to change. An examination of the stability of search result snippets over time is also outlined.

Chapter 6 provides some concluding thoughts on the dissertation as a whole, including a critical overview of the initial goals set out for the dissertation and the degree to which they were completed. Additionally, the chapter discusses a number of considerations for future research in this area.

Chapter 2

State of the Art

While search engine privacy concerns and their countermeasures is an active area of research, most of the publications in the field have been concentrated on prevention of user-profiling via search query aggregation. Tools like Track-Me-Not [31] have been developed to inject noise into search patterns of users to protect their identity. However, a limited number of attempts has been made at examining other privacy issues such as click-tracking or data leaks.

Search-related privacy-enhancing technologies generally come in two flavours. The first one are independent, feature-complete privacy-friendly search engines that claim to not compromise user privacy by not tracking user activity or storing search history. The second one are tools, most commonly web browser extensions, that operate over existing search engines and aim to tackle a specific privacy issue.

2.1 Privacy-Friendly Search Engines

2.1.1 DuckDuckGo

DuckDuckGo [9] is a search engine that claims to "not collect or share personal information". It enables internet users to locate information on the web through the DuckDuckBot, a DuckDuckGo web crawler, as well as other sources such as Wikipedia, Bing and Yahoo. DuckDuckGo does not profile users to provide personalised search results and generates its revenue mostly through non-targeted advertisement.

According to DuckDuckGo's privacy policy, they do not store IP addresses or log any other user-specific data such as user agent or search history. With respect to search term leaks, DuckDuckGo redirects HTTP requests in such a way as to hide the true referer URL from third party websites. DuckDuckGo provides a setting for enabling search via HTTP POST requests which do not leak search query terms. However, the redirection method is used by default due to the inconvenience of HTTP POST, as it breaks browser 'back' buttons and makes it difficult to share and bookmark search result page URLs.

2.1.2 Ixquick

Ixquick provides two privacy-friendly search engines, Ixquick [11] and StartPage by Ixquick [12]. Both operate as a proxy service over existing search engines to hide personally identifiable information such as IP addresses and browser cookies. By acting as a proxy, Ixquick and StartPage deny user profiling to the underlying search engines as these engines are unable to partition the aggregated data they receive.

Ixquick combines search results from a number of popular search engines it queries, ranking them based on the number of times the search result appeared in the combined search result set of the popular search engines. In contrast, StartPage only uses Google's services to provide search results to its users. Both engines are sustained through sponsored search results, however the privacy policies of Ixquick and StartPage state that they do not share or sell personal information. Ixquick and StartPage offer an additional proxy service to allow users to visit external sites without revealing their IP addresses. Similarly to DuckDuckGo, Ixquick and StartPage prevent search term leaks by using HTTP POST to serve search results.

2.2 Search Engine Browser Extensions

2.2.1 Click-Tracking Blocker

The only publication in the area of click-tracking blocking is the paper by Alberdeston et al. [14]. It proposed a browser plug-in that prevented Google, Bing and Yahoo from collecting click data from their users on both sponsored and organic search results. Alberdeston et al. analysed the methods used by the three search engines to allow for click data collection, noting that Google, Bing and Yahoo all tracked clicks with two-hop redirection through the search engine servers, although Bing did not track clicks on organic search results.

The two-hop redirection means that whenever a user clicks on a search result, the webpage would first redirect the user to a search engine's URL where the click will be recorded using additional HTTP parameters for tracking of the user ID, location and so on. After the redirection, the search engine forwards the user to the actual destination website. All of this happens at high speed and the user would not typically notice this behaviour.

Alberdeston et al. solved the problem of double redirection by modifying the search results page's HTML to strip out the redirection URLs encoded into the search result and replace them with the destination URLs. This was made possible due to the fact that the three search engines examined in the paper embedded the destination URL either into the redirection URL.

A possible limitation of the described technique is if the search engines begin obfuscating search result URLs by either hashing or encryption. This would mean that search result URL resolution would be reliant on the search engine redirect. While added encryption to URL resolution would hinder search engine performance, the risk of forced double redirects is not negligible as search engines' profits are fueled by click data.

2.2.2 Straight Google

Another browser extension called Straight Google, developed by Wang [33], exists that prevents click-tracking by URL replacement in webpage HTML. This extension focuses on Google services specifically and supports not only the search engine but other Google-owned products like YouTube, Google Maps and Gmail.

Straight Google removes all URL redirection in Google products, i.e. click-tracking and static redirects. Additionally it sets the 'do not track' HTTP field for all links clicked. However, the tool has not been updated since 2013 and a number of Straight Google's users claim that the extension no longer works with the new updates to Google services. There is no indication of whether the extension will be updated to accommodate these changes in the future.

Chapter 3

Data Collection & Analysis

The first step in examining whether search result snippets could be used for URL resolution was to create a dataset of real search result snippets and their respective URLs. Since it was decided to focus on the Google search engine for the purpose of this dissertation, a web scraper tool was built to scrape Google search result pages using a search query dataset gathered from a number of publicly available sources. After the scraping was complete, the snippet data was extracted from the web page HTML.

The resulting dataset was analysed using rudimentary statistical techniques for a better understanding of the size and diversity of the gathered search result snippets.

3.1 Search Query Dataset

The complete set of the search queries and their sources used in the scraping process is presented in Table 3.1.

A portion of the search query dataset was collected from soovle.com [4]. Soovle provides a list of most used search suggestions across a number of web search providers and is updated daily. Another source of search queries was the Text REtrieval Conference (TREC). TREC ran the Million Query Track in 2007, 2008 and 2009 aimed at ad-hoc retrieval algorithms on a large collection of documents [21]. The source of the queries for the Million Query track were logs of an unspecified popular internet search engine. Lastly, a small set of queries came from previous research conducted at Trinity College Dublin by Aonghusa and Leith [15] on the topic of personalised profiling. For this query set, a portion of queries came from Google Trends and the rest were generated to fit the research criteria.

Source	Size
soovle.com	1,124
Previous research	1,404
TREC'07	10,000
TREC'08	10,000
TREC'09	40,000
Total queries	62,528
Total unique queries	*62,286*

TABLE 3.1. Search query datasets for scraping.

3.2 Web Scraping

For scraping of the search result pages, Selenium WebDriver [3] for Python was used. Selenium WebDriver is a testing framework for web-based applications. It allows for easy browser process creation and interaction with web content through the created browser process. PhantomJS [20] was used in conjunction with Selenium. PhantomJS is a headless browser which enabled the scraper tool to process web content remotely at high speeds.

On start up, the scraper tool creates a PhantomJS process through the Selenium WebDriver. The browser establishes a connection with google.com and continuously sends it queries from the dataset from Section 3.1. The tool saves the source HTML of the first search results page for each of the queries. Additionally, the timestamp of when the page was saved is also recorded.

The scraper tool is set to sleep for a random number of seconds between the different actions it executes to retrieve search results for a query, e.g. entering of the query into the search bar or clicking the 'search' button. The total sleep time is anywhere between 10 and 26 seconds for a single query. This is done minimise the tool's impact on Google's resources as Google is known for blocking of IP addresses that produce high volumes of search requests.

The main goal of the search results scraping was to gather a realistic dataset of text snippets that would mimic a dataset that could have been collected by a number of real search engine users over a period of time. Hence, it was thought to be appropriate for the snippet dataset to contain search results from a number of different geographical locations. Since the collected search query set consisted of English search queries, it was decided to scrape text snippets for three English-speaking locations, Ireland, the U.S.A. and Australia. The Irish dataset was scraped from a virtual machine on the School of Computer Science & Statistics network, while the American and the Australian data came from Amazon Web Services instances located in Oregon and Sydney respectively.

Part of the dissertation objectives was to investigate whether the text snippet for a particular URL was likely to undergo significant changes over time. In order to gather a dataset for this purpose, the Irish scraper was run for four complete iterations of the search query set, once in November 2016, January 2017, February 2017 and March 2017. Thus, 62,528 * 4, or 250,112, Irish search results pages were scraped in total. In contrast, the American and the Australian scrapers completed one full run of the search query set each in January 2017.

An interesting consideration for URL inferral would have been to see how the snippet matching performed across different languages and whether the accuracy of the matching algorithm was affected by the language of the text snippet. Russian was chosen for this purpose due to the availability of Russian search query data. Bukvarix [7] provides lists of the top one million Russian queries collected monthly from yandex.ru, a popular Russian search engine. The query set used in the dissertation was collected between the 1st and the 11th of November 2016 and it was further sampled to produce 15,031 queries for scraping of Russian data. The Russian data was scraped in January 2017 on the virtual machine located in the College. However, in the end, a decision was made to put the this dataset aside due to the dissertation time constraints and the difficulty of processing data with a different character encoding.

3.3 HTML Parsing

A single Google search results page typically contains around 10 organic search results, however this number can be lower for informational queries[1]. Depending on the search query sponsored results can also appear on the search results page, bringing the total number of search results from anywhere between 8 and 20.

The sponsored and organic search results appear in different sections of the search results page (see Figure 3.1) and different HTML elements are used to represent the two search result types.

To extract the snippet data from the HTML pages a Python library, BeautifulSoup 4 [28], was used. BeautifulSoup was created specifically for extraction of data from HTML pages by providing support for HTML tree traversal and element searching. For each

[1] An informational query is a query where the user is looking for a specific piece of information, e.g. "what's the capital of India" or "10 dollars to euro". For informational queries, Google often provides 'featured snippets' which are summarised website extracts that answer the user query directly.

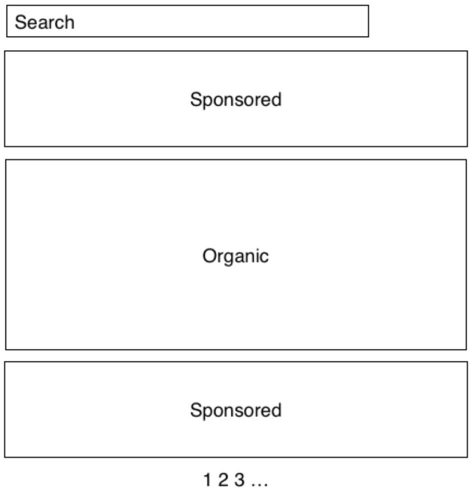


FIGURE 3.1. Search engine results page layout. The bottom sponsored section does not appear on all search results pages with sponsored content.

search result, the following information was collected (items in *italic* were taken directly from the HTML, while everything else was recorded as metadata during the scraping process):

- The search query for the search result.
- *The result URL.*
- *The result title.*
- *The result text snippet.*
- Timestamp of the result scrape.
- The location of where the result was scraped.

3.3.1 Organic Snippet Extraction

All of the organic search results are located in `<div>` elements inside a single `<div>` element of class `"srg"`. By searching for the element with the `"srg"` class attribute with BeautifulSoup it is easy to extract all of the search result elements. Figure 3.2 shows the structure of organic search results. The result URL is embedded into the `"href"` attribute of the heading element. The heading element also contains the result title. The actual snippet text is in a `<span>` element of class `"st"`. If the snippet text contains words from the search query, these words will be displayed in bold with additional `<b>` tags around them. During snippet extraction, words within the `<b>` tags were preserved in the snippet, but the tags themselves were removed.

```
<div class = "srg"> <!-- Organic search results group -->
  <div class = "g"> <!-- Expanded search result start -->
    <div class = "rc">
      <h3>
        <a href = $RESULT_URL$>
          $RESULT_TITLE$
        </a>
      </h3>
      <div class = "s">
        <span class = "st">
          $RESULT_SNIPPET$
        </span>
      </div>
    </div>
  </div> <!-- Expanded search result end -->
  <div class = "g"> ... </div>
  <div class = "g"> ... </div>
</div>
```

FIGURE 3.2. Organic search result structure with extraneous tags and attributes removed.

3.3.1.1 Robots.txt

Robots.txt is a text file that is used by website administrators to disallow web crawlers from crawling and indexing certain pages on the website's domain. This generally implies that webpages that do not allow web crawlers cannot be served as search results as the search engine does not know the content of the webpage.

However, webpages that cannot be crawled due to administrator restrictions may still show up as part of organic search results. This can happen when an external webpage that can be crawled references the uncrawlable document by linking to it. A search engine web crawler can then infer the uncrawlable webpage's content by examining the anchor text[2] and the surrounding text on the crawled page.

When a search engine returns an uncrawlable webpage as a search result, while it is able to provide the URL for the webpage, it cannot display a text snippet. In the case of Google, the following text string is provided instead of the snippet:

> *A description for this result is not available because of this site's robots.txt.*

The dataset collected contained **6,440** such snippets. These snippets were removed during the HTML parsing as they did not provide valuable data points for the snippet matching algorithm or the algorithm evaluation.

[2] Anchor text is the clickable text in a hyperlink.

3.3.2 Sponsored Snippet Extraction

Google's sponsored search results have a different structure from organic search results. They are not grouped into a `<div>` element, but instead they are in an ordered list, `<ol>`. The result title remains in the header element, however the header now contains two URLs. One URL is a Google URL and is probably used for click-tracking, while the second URL is the search result URL. The two URLs can be easily told apart as the click-tracking URL is set to be invisible with CSS styling.

```
<ol> <!-- Sponsored search results group -->
   <li class = "ads-ad"> <!-- Expanded search result start -->
      <h3>
         <a href = $TRACKING_URL$></a>
         <a href = $RESULT_URL$>
            $RESULT_TITLE$
         </a>
      </h3>
      <div class = "ellip"> $AD_TEXT$ </div>
      <div class = "ellip"> $AD_TEXT$ </div>
   </li> <!-- Expanded search result end -->
   <li class = "ads-ad"> ... </li>
   <li class = "ads-ad"> ... </li>
</ol>
```

FIGURE 3.3. Sponsored search result structure with extraneous tags and attributes removed.

Sponsored search results do not have text snippets as such, instead they contain a number of `<div>` elements all of which have the class attribute `"ellip"`. Inside these elements is what appears to be ad text such as advertiser slogans, short descriptions of available stock or descriptions of services offered. It does not seem like the text that accompanies sponsored search results is taken directly from the page pointed to by the result URL. Hence, the advertisement text was taken from the `<div>` elements and concatenated to manually create text snippets for sponsored search results.

3.4 Snippet Analysis

3.4.1 Dataset Size

The size of the snippet dataset for both sponsored and organic search results is presented in Table 3.2.

Snippet type	Size
Sponsored	8,264
Organic	3,237,000

TABLE 3.2. Dataset sizes for sponsored and organic snippets, where each data point is a snippet-URL pair.

Sponsored snippets account for a very small portion of the total dataset because the scraping process was not directed at specifically targeting advertisement content. As the snippets for sponsored and organic results are inherently different in that sponsored snippets are manually curated by advertisers and do not always reflect URL content (see Section 3.3), it would have been inappropriate to carry out the investigation on the combined dataset.

Instead, most of the research will focus exclusively on organic search results, unless explicitly stated otherwise. Due to the fact that the organic snippet dataset is significantly larger, it has a greater potential to provide statistically meaningful results and, arguably, is of more interest as the majority of search engine results are organic results.

A breakdown of collected snippet-URL pairs per country is presented in Table 3.3. The 'Total' column is the total number of snippet-URL pairs that was scraped on a particular server, while the 'Unique' column represents the number of unique (duplicates removed) pairs with scraped per location. The 'Combined dataset' row contains the number of snippet-URL pairs collected across all of the locations. The reason why the number of unique snippet-URL pairs is smaller than the total number of pairs for the U.S.A. and Australia is because, in a number of cases, different search queries produced the same search result.

Location	Total	Unique
Ireland	1,997,633	1,061,995
U.S.A.	628,987	541,376
Australia	610,380	548,331
Combined dataset	3,237,000	1,439,248

TABLE 3.3. Snippet dataset size per server location.

Snippets scraped with the Irish machine represent the largest share of the total dataset. This is because the Irish scraper was set to scrape four times as many queries as the Australian and the American servers to investigate how stable result snippets were over time. For this dataset it appears that the number of unique pairs is about half of the total number of pairs scraped. This could indicate that there might be some stability in the data over time or that a higher number of search queries produces the same search results for Irish locations.

The graph showing the snippet size distribution is presented in Figure 3.4. Figure 3.4 was constructed using 300,000 randomly sampled data points from the dataset. The vast majority of snippets in this sample are up to 330 characters in length and consist of up to 80 words. The number of words in this case is the number of tokens produced on splitting the snippet string on whitespace.

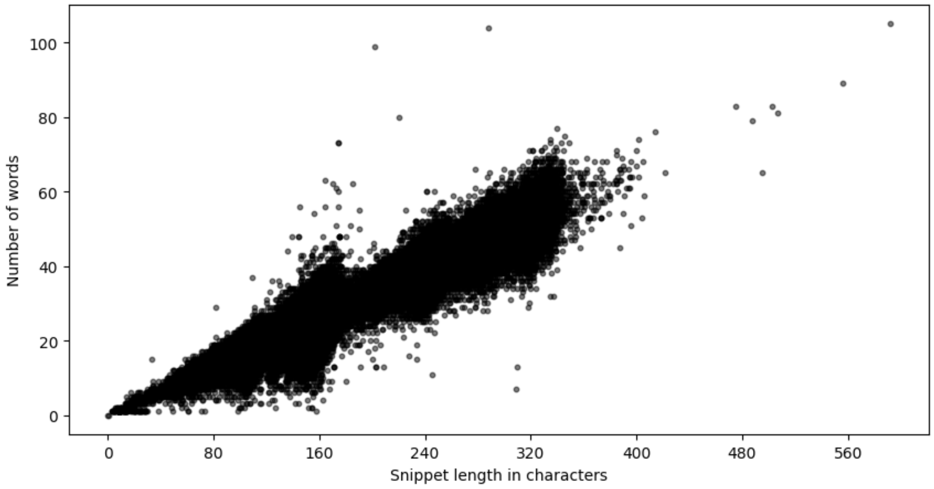


FIGURE 3.4. Distribution of snippet size.

3.4.2 Snippet Diversity

To examine snippet diversity, each of the snippets was pre-processed using the feature extraction techniques described in Section 4.1. This was done to strip away unnecessary information from the snippet such as punctuation and whitespace and increase the granularity of the data.

Only unique snippet-URL pairs were considered for the feature extraction pre-processing, which yielded **316,014** unique features for the dataset. The number of features in the dataset is quite significant as the Oxford English dictionary contains entries for approximately 170,000 words thought to be in current use.

To better understand why the number of features in the dataset was larger than the number of words in the English language, feature occurrence distribution was calculated and it is presented in Figure 3.5 (note the logarithmic scale of the x-axis). For every bin$_{[u..v]}$ in Figure 3.5, the height of the bin represents how many of the total number of features appear in at least u and at most v snippets. The histogram shows that the snippet dataset is quite diverse in terms of occurring features as the majority of features in the dataset occur in a small number of snippets. In fact, 154,942 (49%) of all features only appeared once in the entire dataset.

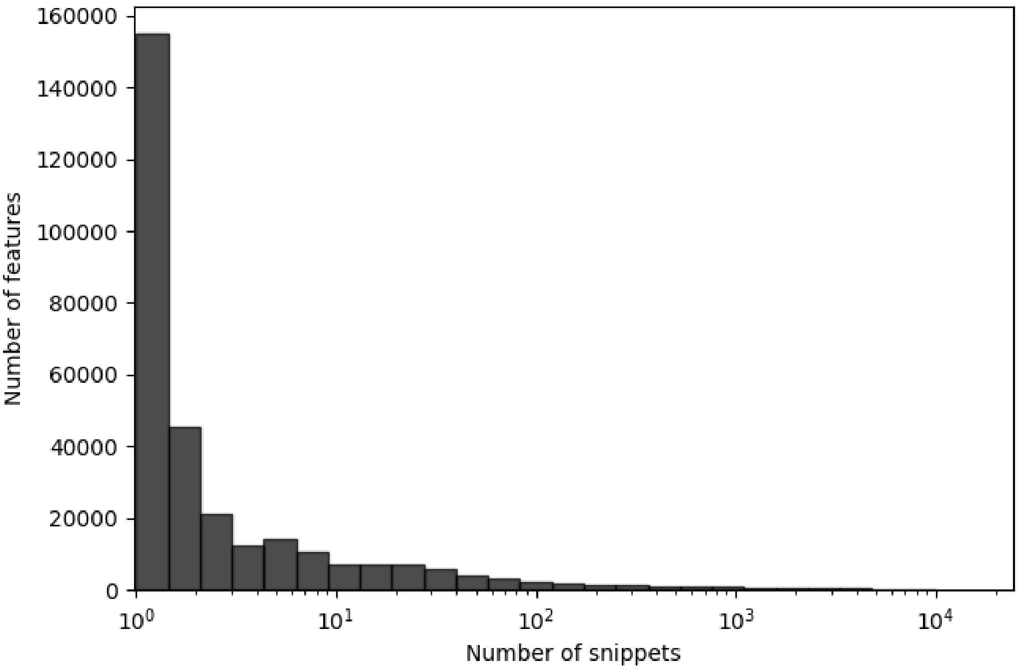


FIGURE 3.5. Distribution of feature occurrences. Size of each bin$_{[u..v]}$ = $\sum_{i=u}^{v} |f \in F : count(f) = i|$, where F is the snippet feature set.

Upon closer investigation, there appears to be a number of reasons why the dataset contains such a large number of infrequently occurring features:

- Non-English text

  Although English queries were used in the scraping process, a number of the scraped search results were either only partially in English or entirely in another language. This resulted in a number of non-English features.

  For example, feature *windschutzscheiben* was extracted from the following snippet:

  > 'MRA - Motorcycle Racing Accessories Verkleidungen, Scheiben, Hockersitze und Motorrad-Windschutzscheiben fur den Rennsport.'

  Although it would have been possible to add another pre-processing step to detect and remove non-English text from the dataset, it was decided that it was necessary to preserve the dataset in its original form. If the scraper was able to collect non-English search results while querying in English, there is a valid scenario where a real search engine user might be in a similar situation and might require a URL for a non-English snippet to be resolved. Therefore, the snippet matching algorithm must be able to deal with such requests to some degree instead of completely disregarding them.

- Misspellings

  Another portion of features extracted from the dataset were misspellings or typos of English words. The typos appear on the search result websites and the search engine just happened to include the misspelled content as part of the search result snippet.

  For example, feature *producemen* was extracted from the following snippet:

  > 'Merging clean, classic style with innovative on-trend design, Fossil producemen and women's wrist-watches which fuse traditional elegance with a youthful twist.'

- User names, website names and internet slang

  Another reason for a diverse number of features is the fact that often user names, website names and internet slang appear in search results. This is particularly common for snippets for social media sites such as Facebook or Twitter. While certain names occur in a large number of snippets, such as 'wikipedia' which is one of the top ten domains for the dataset (see Table 3.5) and is one of the 25 most common features, the majority of website and user names appear once or twice.

  For example, feature *noteworthygam* was extracted from the following snippet:

  > 'May 8, 2014 - Tomasz TGM. ... You can choose what you want the animation to be like, just simply comment below and I will make your idea come to life! ... NoteworthyGames - Clash of Clans 692,519 views.'

- Names of places and people

  Names of people and places in snippets also cause a large number of features in the dataset.

  For example, feature *halderman* was extracted from the following snippet:

  > 'At a price tag of $625,000, the property was sold through Halderman Real Estate Services by Girl Scouts of Tribal Trails Council. ACRES Land...'

The top ten features from the dataset are presented in Table 3.4. The most frequently occurring feature, "state", appears in 8.42% of the snippets in the dataset and by the sixth most frequent feature the frequency of occurrence is almost halved. This further suggests that the snippet data consists of highly diverse content.

Feature	No. Snippets
'state'	121,264
'servic'	88,770
'find'	86,968
'inform'	83,912
'home'	80,479
'free'	74,717
'review'	63,941
'like'	63,897
'includ'	63,756
'depart'	61,861

TABLE 3.4. Top ten snippet features.

Domain	No. URLs
wikipedia	55,664
facebook	43,478
youtube	36,092
google	26,282
indeed	14,744
tripadvisor	14,483
yelp	13,589
nih	12,576
amazon	11,121
ebay	8,684

TABLE 3.5. Top ten URL domains.

Unlike the snippet strings, the URLs represented in the dataset show some similarity. In total, **193,515** unique domains are presented in the snippet dataset. Out of those, 104,147 domains appeared once in the entire dataset. This means that 1,335,101 snippets are represented by just 89,368 URL domains.

The top 10 domains and their number of occurrences are given in Table 3.5. These include popular websites like Wikipedia, Facebook and YouTube. Surprisingly, 'google' is the fourth most common domain among the result URLs as Google search results seem to often refer to services such as Google Books, Google Maps and Google+.

Chapter 4

Snippet Matching

A basic approach to snippet matching could have been implemented as string comparison, where each query snippet string would have been compared to all of the snippet strings in the dataset and the URL of the closest matching snippet would have been considered to be the URL of the query snippet. This approach, however, would have been computationally infeasible in any real-world system, especially a system that is expected to process diverse text from across the web such as the one proposed by this dissertation.

Instead, each of the snippet strings was processed to extract a set of features that was representative of the content of the snippet to reduce the search space and enable faster matching. The feature extraction relied on a number of well-established text mining techniques such as stop word removal and tokenization. Following that, an efficient snippet matching algorithm was devised with the use of GPUs for additional computational speed-up.

4.1 Feature Extraction

Following the snippet diversity analysis in Section 3.4.2, it was clear that the dataset collected for the dissertation consisted of a diverse set of words. Hence, as much of the uniqueness of each snippet needed to be preserved during feature extraction in order to allow for the snippet matching algorithm to correctly distinguish between snippets.

As the goal of the snippet matching algorithm was to correctly identify the URL given a query snippet string, semantic similarity of snippets was not an important consideration for the algorithm. Instead, syntactic similarity of snippets was to be evaluated as part of the matching process.

To reduce the search space of the snippet matching algorithm and to allow the snippets to be matched according to their composition rather than semantic meaning, each snippet went through the following 5 steps in order to be converted from the text string to a feature vector. An example of the steps involved in feature extraction using a sample search result snippet from the dataset is provided in Table 4.1.

1. Punctuation & Digit Removal - The first step was to strip out any punctuation and other non-letter characters such as digits from the snippet string.

2. Tokenisation - The next step was to break up the snippet string into meaningful tokens. Due to the diversity of the snippets, the goal of this step was to preserve as many of the words from the snippet string as possible. The snippet string was split on whitespace. This created a token for each word in the snippet. During the tokenisation step, all of the characters were also converted to lower case to allow for easier matching later on.

3. Stop Word Removal - Stop words are the most common words in a language, e.g. 'the', 'have', 'are' or 'at' in the English language. For text mining purposes, these words are often considered to add noise to a text document as stop words rarely enhance contextual meaning and, in the case of document similarity, can result in a high number of false positives. Therefore, stop words are often removed in the pre-processing step.

   To remove stop words from the list of snippet tokens, the Natural Language Toolkit (NLTK) 3.0 library for Python [25] was used. NLTK provides a set of 153 stop words for the English language, consisting mostly of prepositions and pronouns.

4. Stemming - Following stop word removal, word stemming is performed. Stemming is a process by which inflected words are reduced to their stem or root form. For example, the words 'walked' and 'walker' can be stemmed to 'walk'. Stemming reduces the corpus size and simplifies the text matching process.

   For snippet stemming, NLTK's implementation of the Snowball Stemmer [26] for the English language was used.

5. Duplicate & Short Word Removal - Although typical word vector models preserve multiplicity of words (i.e. the frequency of word occurrence in a text), for the purpose of snippet matching token frequency is discarded and only the presence/absence of a token is recorded. Additionally, tokens of size three or less are removed from the set to reduce the corpus size.

4.2 Bag-of-Words Representation

The set of features extracted for each of the snippets is essentially a bag-of-words representation of that snippet. Bag-of-words models are commonly used in text mining and they represent a bag or a multiset of the words contained within a text string. Bag-of-words models do not maintain the grammatical structure of the text they represent nor do they preserve the word order.

As stated in Section 4.1, the frequency of words is disregarded during feature extraction, something that is typically preserved in bag-of-words models. The reasoning behind this decision was that the snippet text was relatively short and it would not have been worthwhile maintaining frequency counts as part of the model where most of the tokens only appear once.

Because the feature set size for this dataset is over 300,000 features, all of the snippets become very sparse feature vectors after feature extraction. This is undesirable as it takes up memory and makes any calculations difficult. In order to make the feature vectors more dense, it was decided to store the extracted snippet tokens directly in the feature vector, instead of storing presence/absence of every possible feature for every snippet. Additionally, to make computations faster, the tokens are hashed and the hashes are stored in a sorted order as integer operations are more efficient than string operations.

For example, for the following snippet string:

"The Caltrain Corridor Joint Powers Board, which owns and operates Caltrain ..."

a typical bag-of-words representation that assumes the following feature set:

['caltrain', 'bike', 'corridor', 'joint', 'powers', 'board', 'has', 'owns', 'operates']

would result in the feature vector:

[2, 0, 1, 1, 1, 1, 0, 1, 1]

In contrast, the implementation used in this dissertation would store the following as the snippet's feature vector:

sorted([h('caltrain'), h('corridor'), h('join'), h('power'), h('board'), h('own'), h('operate')])

Original Snippet	Yr Weather forecast for Dublin, Leinster (Ireland) Weather forecast for Dublin, Leinster (Ireland). Updated at 7:34. Next update .... Free weather data (Javascript- or XML-forecasts). Information about the place...
Punctuation & Digit Removal	Yr Weather forecast for Dublin Leinster Ireland Weather forecast for Dublin Leinster Ireland Updated at Next update Free weather data Javascript or XML forecasts Information about the place
Tokenization	[’yr’, ’weather’, ’forecast’, ’for’, ’dublin’, ’leinster’, ’ireland’, ’weather’, ’forecast’, ’for’, ’dublin’, ’leinster’, ’ireland’, ’updated’, ’at’, ’next’, ’update’, ’free’, ’weather’, ’data’, ’javascript’, ’or’, ’xml’, ’forecasts’, ’information’, ’about’, ’the’, ’place’]
Stop Word Removal	[’yr’, ’weather’, ’forecast’, ’dublin’, ’leinster’, ’ireland’, ’weather’, ’forecast’, ’dublin’, ’leinster’, ’ireland’, ’updated’, ’next’, ’update’, ’free’, ’weather’, ’data’, ’javascript’, ’xml’, ’forecasts’, ’information’, ’place’]
Stemming	[’yr’, ’weather’, ’forecast’, ’dublin’, ’leinster’, ’ireland’, ’weather’, ’forecast’, ’dublin’, ’leinster’, ’ireland’, ’updat’, ’next’, ’updat’, ’free’, ’weather’, ’data’, ’javascript’, ’xml’, ’forecast’, ’inform’, ’place’]
Duplicate & Short Word Removal	[’data’, ’inform’, ’place’, ’javascript’, ’ireland’, ’dublin’, ’forecast’, ’next’, ’free’, ’weather’, ’leinster’, ’updat’]

TABLE 4.1. Example of feature extraction for a search result snippet for the search query "weather".

4.3 Snippet Matching

All of the distinct snippet-URL pairs from the dataset were converted to bag-of-words representations to create a snippet-URL repository. The snippet-URL repository was to serve as a collection of user-gathered URL mappings against which the query snippets were compared (see Figure 4.1).

The reason for only using distinct snippet-URL pairs from the dataset was that including duplicate values in the repository would have made no impact on the algorithm's accuracy but it would have significantly slowed down the matching process because the algorithm would have needed to compare the query snippet against a larger set of mappings.

The snippet matching algorithm starts execution by receiving a search result snippet, i.e. the query snippet, for which the URL is not known. Feature extraction is performed on the query snippet to produce a feature vector for the snippet. The algorithm computes a match score between the query snippet and all of the snippets in the snippet-URL repository using a text similarity metric described below. Once all of the match scores are computed, the URL belonging to the snippet with the highest match score is returned as the URL the query snippet would resolve to. In cases where the highest match value is shared between more than one snippet in the repository, the URL that is returned is chosen at random.

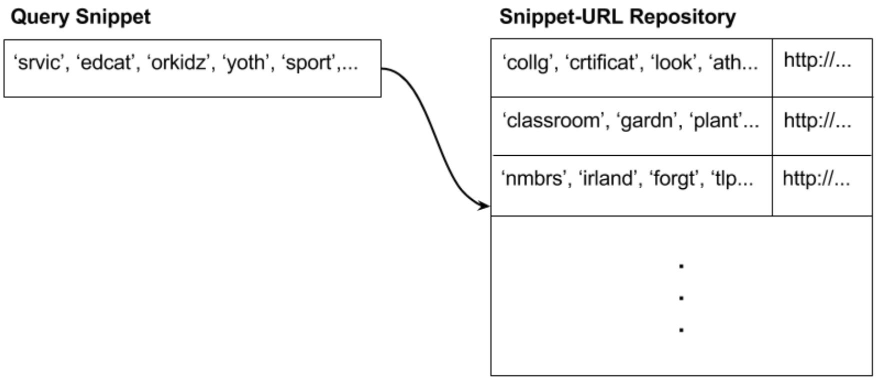

Query Snippet
'srvic', 'edcat', 'orkidz', 'yoth', 'sport',...
Snippet-URL Repository
'collg', 'crtificat', 'look', 'ath...
http://...
'classroom', 'gardn', 'plant'...
http://...
'nmbrs', 'irland', 'forgt', 'tlp...
http://...
.
.
.

FIGURE 4.1. The snippet matching algorithm. The query snippet is matched to one of the snippets in the snippet-URL repository and the URL for that snippet is returned as the URL the query snippet would resolve to.

At first, the text similarity metric for snippet matching was implemented as the Euclidean distance between the two feature vectors. It measures the straight-line distance between two points in Euclidean space. Text similarity using Euclidean distance of two feature vectors A and B is calculated as:

$$sim(A,B) = D(A,B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$

However, it soon became clear that Euclidean distance was not a suitable similarity metric for the snippet dataset. Due to the high dimensionality of the extracted feature vectors, the similarity values produced by the above equation for Euclidean distance were not meaningful.

In the end, cosine similarity was used to compute the match scores. Cosine similarity is based on the calculation of the cosine of the angle between two non-zero vectors. It is measured as:

$$sim(A,B) = cos(\Theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

where A and B are two non-zero vectors.

4.4 GPU Performance Improvements

As the snippet-URL repository contained around 1.5 million snippet-URL pairs, calculating the highest match score between all of the repository pairs and the query snippet sequentially was not computationally feasible for any significant number of queries. It was decided to take advantage of GPU programming which offers significant computational speedups through parallelisation to enable faster processing of snippet queries.

```
def get_url(query_snippet):

  for snippet in repository.snippets: # can parallelise
     calc_match(query_snippet,snippet)

  highest_match = get_highest_match() # can parallelise
  url = repository.get_url(highest_match)
  return url
```

FIGURE 4.2. Snippet matching algorithm pseudo-code.

Figure 4.2 shows the pseudo-code of the snippet matching algorithm. The algorithm contains two lines that can be easily parallelised, highlighted with a comment. The first line is the loop that calculates the match scores between the query snippet and the snippets in the repository. Because the match score calculation is only dependent on the current snippet and is independent of any other calculations, the loop can be parallelised with each snippet being processed by a separate thread. The second line that can be optimised is the selection of the highest match score. The highest match score can be selected using a parallel reduction, where groups of values are compared in parallel and the highest values from each group are combined into groups for the next reduction round until just one value remains.

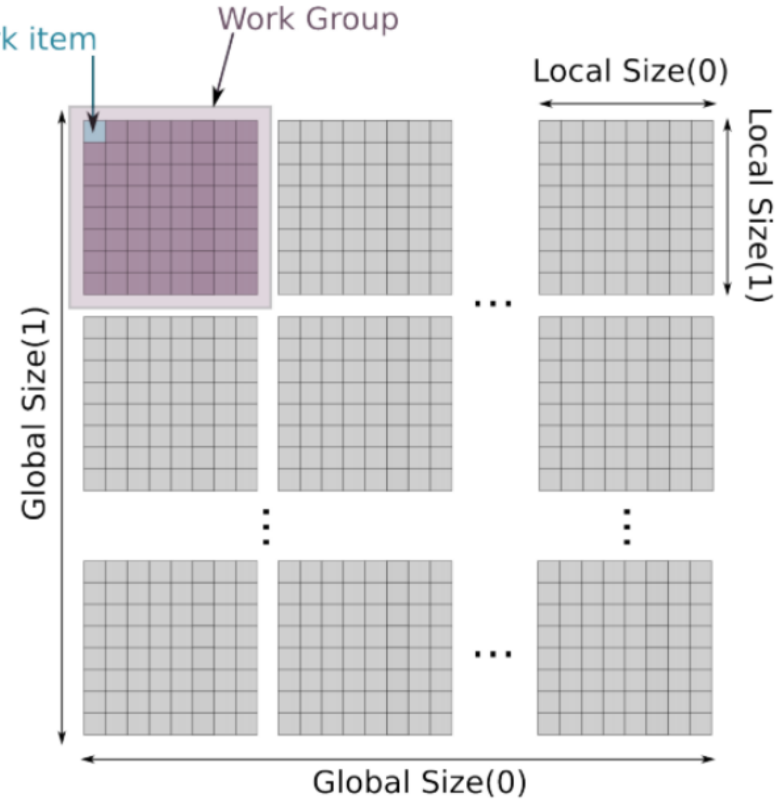


FIGURE 4.3. Conceptual GPU architecture (diagram from Tompson and Schlachter [30]).

4.4.1 GPU Architecture

The GPU available for use in this dissertation was the NVIDIA GeForce GTX 1080 graphics card with 2,560 CUDA cores[1], where one CUDA core is a small computing unit

[1] CUDA (Compute Unified Device Architecture) is parallel computing platform and a programming model for general purpose GPU programming developed by NVIDIA [2].

capable of execution in parallel.

Traditionally, CPUs were designed to optimise the performance of a single thread by reducing latency of memory accesses with large caches and by providing branch prediction and out-of-order instruction execution. The GPU architecture takes a different approach to achieve computational efficiency. GPUs are special-purpose processors that maximise throughput by hiding computational latency with parallel execution of hundreds of slower threads. While GPUs were initially created for use in graphical rendering, in recent years they have become popular for more general, computation-intensive tasks such as video processing or matrix operations.

The GPU architecture resembles the Single-Instruction-Multiple-Data (SIMD) model where each program instruction is executed over a number of data elements simultaneously. All of the processing to be carried out by a GPU is known as 'units of work'. Each unit of work is processed as a 'work item'. N work items are are processed by a 'work group'. The distribution of work items to be processed by the GPU across work groups can either be specified by the programmer or configured by the GPU itself. Figure 4.3 demonstrates the conceptual architecture of a GPU.

An interesting point to note is that a work group shares the program counter during parallel execution. This means that if the executed program contains branching instructions such as for loops or if statements, all of the different branches will be executed sequentially. For example, if three out of ten work items take the 'if' branch of an if statement, the seven other work items will be stalled until the all of the instructions in the 'if' branch are executed. The three work items will then be stalled while the 'else' branch is executed.

A GPU's memory is organised into a hierarchy to support efficient access. GPUs have a large DRAM memory that serves as the global memory for the entire processor. The DRAM is indexed by the programmer and each index represents a work item. Work groups operate over a contiguous block of work items by loading the DRAM memory chunk into a smaller, local memory that is only accessible to the particular work group. The local memory is also used for communication between work items of a work group. Additionally, each work item has a small private memory for stack-allocated variables.

4.4.2 Open Computing Language

Open Computing Language (OpenCL) provides an API for heterogeneous computing environments consisting of CPUs, GPUs and other hardware accelerators. PyOpenCL library [24], which provides Python bindings for OpenCL programming, was used to parallelise the snippet matching algorithm.

Figure 4.4 presents the OpenCL platform model. OpenCL assumes a host, typically the CPU, and a number of compute devices which can either be GPUs or CPUs. The

OpenCL model specifies that a compute device consists of a number of compute units, i.e. GPU work groups, which are in turn composed of processing elements, i.e. work items. The host script is tasked with the parallel execution set up, including the specification of the kernel code. A kernel is a small C program that is run by all of the processing elements in parallel.

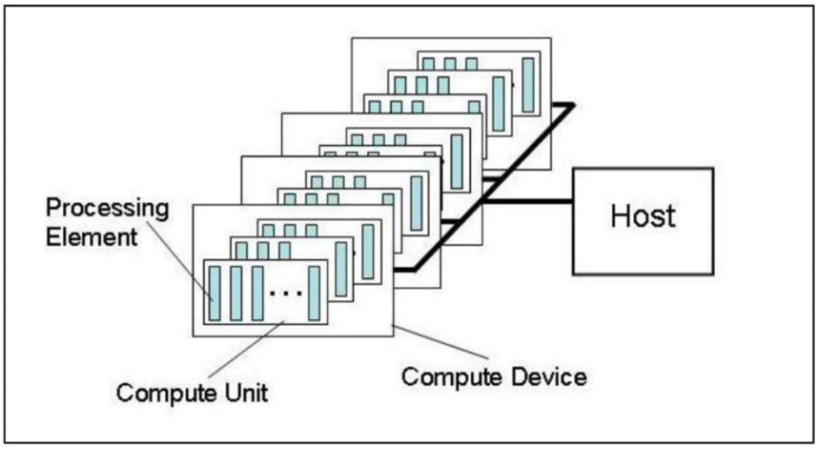


FIGURE 4.4. OpenCL platform model (diagram from Tompson and Schlachter [30]).

The host script also takes care of the data load and retrieval to/from the GPU DRAM prior and after the kernel execution. OpenCL provides APIs for creating memory buffers on the host and handles data transport between the devices. For memory buffers only one dimensional arrays of bytes are accepted, meaning that it is the programmer's responsibility to ensure the correct layout of the data in memory. A typical work flow of the host script involves parallel set up, data transfer, kernel invocation and transfer and processing of the results of the kernel execution.

Inside the kernel code, the programmer has access to the work group and the work item IDs (referred to as global and local size in Figure 4.3) which allow for correct identification of the memory chunk that is to be processed as a work item. The kernel is also able to synchronise work items in a work group with memory barriers, however synchronisation across different work groups is not possible.

4.4.3 Match Score Calculation Parallelisation

As described previously, the match score calculations were easily parallelisable as they were computed independently of each other. This was achieved with a kernel program where a work item was the computation of the match score between a query snippet and a candidate snippet from the repository.

```
int query_i = 0;
int candidate_i = 0;

while (query_i < q_size && candidate_i < c_size) {
   if (query_snippet[query_i] == candidate[candidate_i]) {
      // cosine calculation
      query_i++;
      candidate_i++;
   }
   else if (query_snippet[query_i] < candidate[candidate_i]) {
      // cosine calculation
      query_i++;
   }
   else {
      // consine calculation
      candidate_i++;
   }
}
```

FIGURE 4.5. Sequential match score calculation in O(n).

In the sequential version of the code, the bag-of-words representations of the snippets in the snippet-URL repository consisted of a sorted list of hashed features present in the snippet string. This allowed for iteration over the two feature vectors for cosine similarity calculation in linear time (pseudo-code is provided in Figure 4.5). The sequential code consisted of a number of conditional branching, which was undesirable for the purposes of GPU parallelisation. A significant number of branching instructions in kernel code reduces the benefits of the GPU parallelisation as mentioned in Section 4.4.1.

In order to reduce the number of branches in the kernel, the query snippet representation had to be restructured from a dense vector of hashed features to a sparse vector of all of the features in the dataset. This resulted in a 316,014 element vector for the query snippet, where features that were present in the query snippet were marked as '1' and the rest of the elements were '0'. The snippets in the snippet-URL repository were also restructured to contain the indices of the feature set instead of the feature hashes as shown in Figure 4.6.

This meant that the algorithm now had two different representations for query and repository snippets. The query snippets became sparse vectors, similar to a typical bag-of-words model, while the snippets in the repository continued to be dense vectors that now indexed the feature set and, subsequently, the query snippet.

As the repository snippets remained to be dense vectors, they had to be padded to a uniform length in order to be processed on the GPU. The vectors were padded with zeros, with the first vector element representing the true length of the vector. For maximum

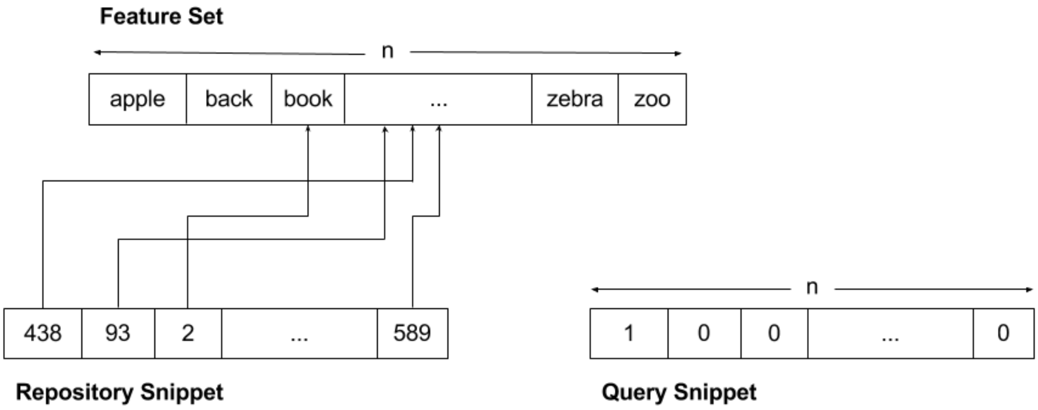


FIGURE 4.6. Snippet feature vector structure for parallelisation.

efficiency, the repository snippets were sorted by their true length in the GPU memory buffer to allow for snippets of similar length to be processed by one work group and thus minimise work item branching stalls.

Because of the new query snippet structure, the cosine similarity metric for match score calculation was no longer suitable as it would have required iterations over the entire feature set for all work items, as well as requiring some conditional branching. Instead, the match score was calculated as a number of matching features between the query snippet and the repository snippet as shown in Figure 4.7. This change to the similarity metric meant that the kernel now contained a single branch instruction in the for loop and because snippets of similar lengths were stored near each other in memory, stalls on this branch instruction were reduced.

```
int match_score = 0;

for (int i = 0; i < c_size; i++) {
   match_score += query_snippet[candidate[i]];
}
```

FIGURE 4.7. Parallel match score calculation in O(n).

4.4.4 Highest Match Selection Parallelisation

Once all of the match scores are computed in parallel, the algorithm has to select the highest match score from 1.5 million values. A common technique for doing this in

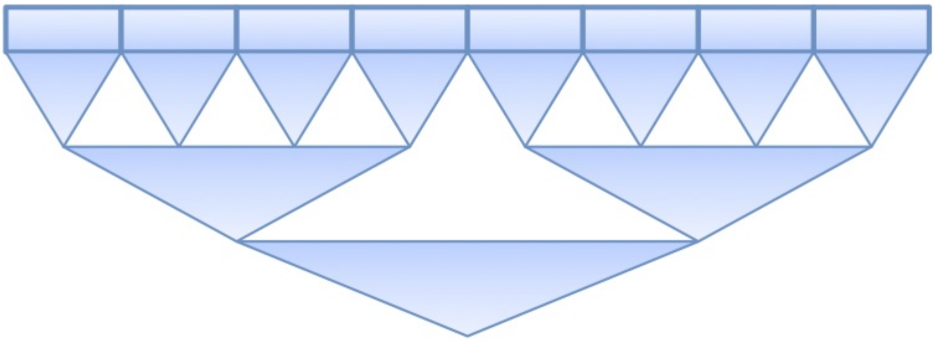


FIGURE 4.8. Parallel reduction structure (diagram taken from Catanzaro [18])

parallel is known as reductions. Reductions take an array of values and 'reduce' them to a single value. In the case of the snippet matching algorithm, the goal is to reduce all of the match scores to just the highest score.

Reductions are performed by groups of threads, or work groups in the case of GPUs, as shown in Figure 4.8. Each work group takes N work items, one for each thread, and reduces them to one work item for the entire work group. The reduction factor is directly related to the specifications of the GPU used for the parallel computation. The NVIDIA GeForce GTX 1080 graphics card used in this dissertation was capable of running up to 1024 threads per work group, meaning that by performing a single reduction operation on the vector of match scores its size could be reduced from 1.5 million to 1,464 elements.

In order to ensure equal division of work between the work groups and the threads, the match score vector was padded to the closest power of two before being processed, as suggested by Catanzaro [18]. The padded portion of the vector consisted of elements of -1 as the valid range of match score values was $[0, len(query\_snippet)]$. This ensured that a padded value was never selected as the highest match score.

The pseudo-code for the reduction kernel is given in Figure 4.9. The first step for the threads is to move the relevant data from the global GPU memory to a local vector that is shared between all threads in the work group. The kernel must ensure that all of the threads have copied their work item to local memory before proceeding to avoid race conditions.

The reduction can be performed with two different memory access patterns, as shown in Figure 4.10. The first one is on the left-hand side of Figure 4.10. This pattern involves each thread processing two neighbouring elements of the vector and overwriting the fist of the two with the larger value. While this provides fast memory access on the first iteration of the for loop, any subsequent iterations are problematic as the neighbouring elements move further apart in the vector. This causes memory conflicts and the benefit of parallelisation is lost as data needs to be moved around for each computation.

```
void reduction() {
  local_vector[thread_id] = thread_id < work_group_size ?
      global_vector[thread_id] : -1;
  sync_threads();

  for (int offset = work_group_size / 2; offset > 0; offset >>=1) {
      if (thread_id < offset) {
          int a = local_vector[thread_id + offset];
          int b = local_vector[thread_id];

          if (b < a) {
              move_a_into_b();
          }
          sync_threads();
      }
  }

  if (thread_id == 0) {
      write_result(local_vector[0]);
  }
}
```

FIGURE 4.9. Highest value selection reduction kernel pseudo-code.

The right-hand side of Figure 4.10 shows the memory access pattern used in the snippet matching algorithm. For this pattern, the values are written to a contiguous block at the beginning of the vector. This means that with each iteration the distance between the compared elements decreases, which results in less memory conflicts overall.

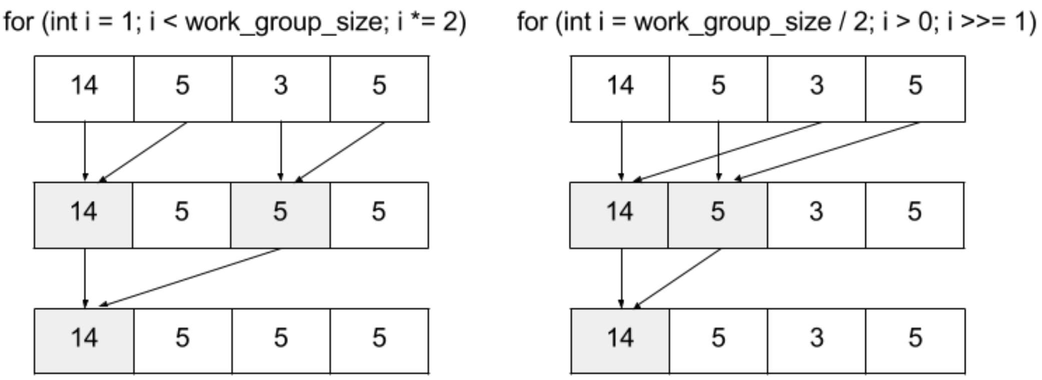

for (int i = 1; i < work_group_size; i *= 2)
14
5
3
5
14
5
5
5
14
5
5
5
for (int i = work_group_size / 2; i > 0; i >>= 1)
14
5
3
5
14
5
3
5
14
5
3
5

FIGURE 4.10. Parallel memory access patterns.

Another thread syncronisation is required at the end of each iteration of the loop to ensure the correct order of execution. Once the local portion of the match score vector is reduced, the thread with the ID of 0 has the highest match value for the work group and is delegated with writing it back to global GPU memory.

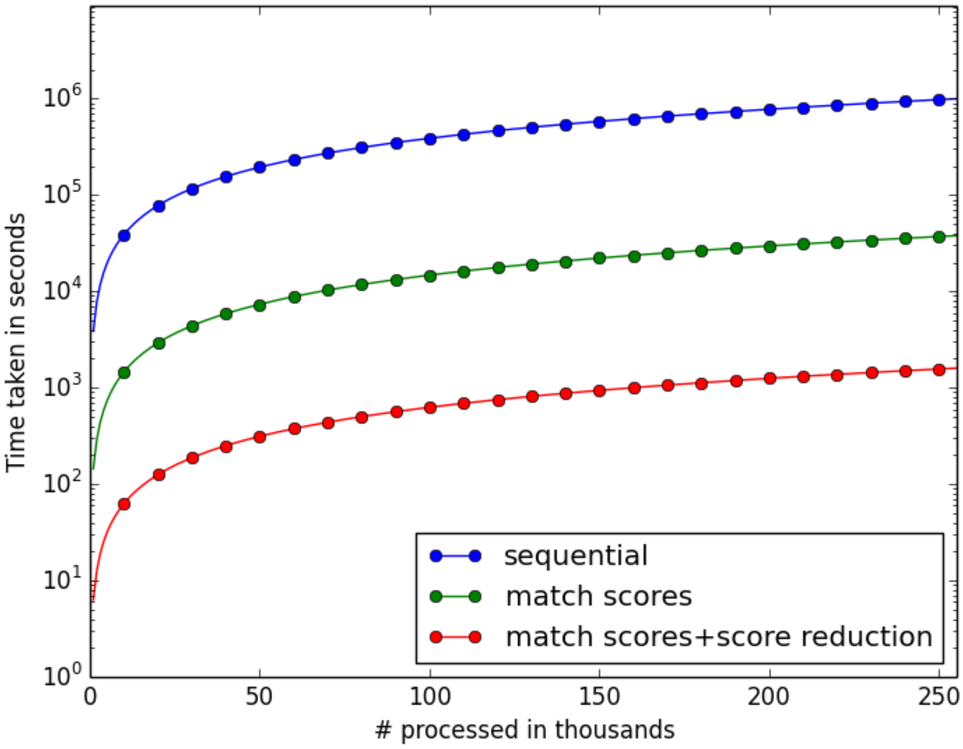


FIGURE 4.11. Performance difference between sequential and parallel algorithm.

The performance differences between the different algorithm implementations can be seen in Figure 4.11. The y-axis in Figure 4.11 is a logarithmic scale. The blue line shows the performance of the sequential snippet matching algorithm, where processing of 1,000 queries on average took 3,870 seconds. The green line represents the performance of the snippet matching algorithm with just the match score calculations computed in parallel, with an average of 144 seconds per 1,000 queries. The red line shows the final version of snippet matching algorithm with both the match score calculation and the highest match score selection computed in parallel. This version of the algorithm took approximately 6 seconds to process 1,000 queries.

CHAPTER 5

SNIPPET MATCHING RESULTS & EVALUATION

The snippet matching evaluation process was based on an investigation of how often the snippet matching algorithm returned the wrong URL while knowing the correct mapping for the query snippet. Additionally, the reasons behind the occurrence of errors were examined to better understand whether any changes needed to be made to the implementation of the snippet matching algorithm to reduce the error rate and improve the algorithm's performance.

Another important aspect of the snippet matching evaluation, presented in this section, was the robustness of the algorithm to changes to the query snippet and whether it could still perform well when given modified input. Finally, a stability investigation was also carried out to examine whether search result snippets were likely to undergo significant change over time and whether older snippets could be used to infer URLs for newer versions of the same snippet.

5.1 Algorithm Performance

The main performance measure that was of interest to this dissertation was the error rate of the algorithm. The goal of the dissertation was to investigate whether search result snippets could be used to infer search result URLs and evaluating the algorithm in terms of its error rate would have provided an answer to the dissertation question.

In order to correctly evaluate the error rate of the algorithm, only the snippets that were already part of the snippet-URL repository could be used as query snippets. This was because the basic underlying assumption of the dissertation was that the algorithm would only be run on previously collected data. Hence, evaluating performance based on unseen data would have been inappropriate. For that reason the collected snippet dataset was used as the query snippet set as well as the snippet-URL repository. Essentially, the snippet dataset was queried over itself and the error rate was measured. For this evaluation, only distinct snippet-URL pairs were used as query snippets. Including non-distinct pairs would have resulted in an inaccurate error rate.

The parallel (GPU-based) implementation of the snippet matching algorithm was used in the performance evaluation for efficiency reasons. The algorithm was queried was queried using distinct snippets from the dataset, with the highest matching snippet URL returned as the URL for the query snippet. The returned URL was compared to the true URL of the query snippet and the number of wrong URL predictions was recorded.

5.1.1 URL Equality

A key part of the error rate calculation was determining whether the query snippet URL was equal to the URL returned by the snippet matching algorithm. URL equality is not as straight forward as performing string comparison, as non-identical URLs might still point to the same web content, while subtle differences in other URLs can result in extremely different web pages. Examples of URL differences include:

- The top-level-domain
- The protocol used
- The connection port
- Letter case
- GET parameters

Overall, URL equality for non-identical URLs cannot be guaranteed unless the differences of the two URLs are semantically equivalent. For example, URL domains are case-insensitive [17] meaning that "google.com" and "gOOGle.com" are the same URL. Any other part of the URL is treated as case-sensitive and the same assumption about URL equality no longer holds.

For URL equality testing, Python-URL-Tools [16] was used. This library provides a URL comparison method that breaks the URL string into individual components and performs equality tests on these components. As part of the comparison, the library strips the connection ports from URLs, removes empty GET parameters and sorts them alphabetically. The library does not consider URLs that are only different in the protocol they use (HTTP vs HTTPS) as equal. However, this was something that was useful for URL equality in snippet matching and so this extra check was added alongside the library comparison method.

5.1.2 Error Rate

The error rate for the snippet matching algorithm is presented in Table 5.1. As mentioned in Section 3.4.1, the sponsored and the organic search results were evaluated separately due to the fact that they are generated in different ways. The error rate for the organic search result dataset (3.48%) is double of that for the sponsored search result dataset (1.74%). This is somewhat expected as the organic dataset has approximately 1.5 million snippets, whereas the sponsored dataset is much smaller, at around 8,000 snippets.

Snippet Type	Error Rate
Organic	3.48%
Sponsored	1.74%

TABLE 5.1. Snippet matching error rate per snippet type.

However, it is surprising that the error rate for the sponsored search results is so small. As mentioned by Souvey [29], Google provides the advertisers with a number of ways to customise sponsored search result URLs to include user-specific parameters such as user location or device type. Hence, the expectation for the sponsored search results would be that the search result URL would be tailored to the particular searching event, which should result in a higher error rate during snippet matching. A possible explanation for such a low error rate is that the sponsored search result dataset is too small for any statistically meaningful results. Although a portion of the 1.74% error rate was due to mismatched URL parameters, there was not enough of a representative sample collected that would have covered all of the possible ranges of advertiser URLs to thoroughly investigate this hypothesis.

Location	Error Rate
Ireland	4.00%
Ireland (sampled)	2.37%
U.S.A.	0.63%
Australia	0.71%

TABLE 5.2. Snippet matching error rate per server location.

The error rate of the snippet matching algorithm when evaluated using the snippet data from a single location, i.e. search results from either Ireland, the U.S. or Australia alone, is presented in Table 5.2. From the table it is evident that both the U.S. and Australia have comparable error rates, while the search results scraped from an Irish IP address have a slightly higher error rate of 4%. Both the American and the Australian datasets are similar in size, at about 550,000 snippets each. The Irish dataset is roughly double the size of the American/ Australian datasets, which could be part of the reason why it has a higher error rate. The higher rate could also be attributed to the fact that the Irish dataset contains snippets collected over time, where not all of the earlier collected snippets are identical to the later snippets. The Irish dataset was randomly sampled to produce a subset of 550,000 snippets and the error rate in this case was 2.37%.

5.2 Sample Errors & Discussion

An interesting observation about the error rate for the combined dataset of the organic snippets from all locations, i.e. 3.46%, is that **67.51%** of the errors occurred because the two different URLs had identical search result snippets, meaning that no possible algorithm would have been able to identify the correct URL for all of these queries deterministically.

5.2.1 Identical Snippets

As identical snippets were the cause of the majority of errors in this evaluation, it was important to examine why different URLs were likely to share search result snippets and whether these were truly different URLs or whether these errors were a result of poor URL equality tests.

The errors resulting from identical snippets had to be manually examined as there was no way to automate the process. Out of the 35,048 errors that occurred due to identical snippets, 100 errors were randomly chosen for manual inspection.

The inspected errors were classified into 5 categories outlined below. For each category, the number of errors examined belonging to that category is given along with examples of the errors from the dataset. A single error could have been classified as belonging to more than one category as some errors were a result of a combination of categories as explained below:

1. Unrelated domains with identical content: 1

   There was one examined error that occurred because two websites had published the same news article. This caused the search engine to display identical search result snippets for the two search results.

Query URL	Matched URL
http://www.lexology.com/library/detail.aspx?g=25221701-41ac-428e-9606-beb06f55f2fc	https://www.pbwt.com/robert-p-lobue/antitrust-update-blog-2/ny-district-ct-monopolization-claims-alleging-manipulation-electricity-prices-proceed-against-barclays/

2. Similar URLs on the same domain: 31

   In some cases, the errors were URLs that were visually similar and retrieved the same content, but were different enough that this difference was not detectable by a script.

Query URL	Matched URL
https://www.excellusbcbs.com/wps/portal/xl/individual-family-coverage/health-plans/child-health-plus/	https://www.excellusbcbs.com/wps/portal/xl/inp/healthplans/childhealthplus/
https://www.hud.gov/hudportal/HUD?src=/states/washington/renting	http://www.hud.gov/local/index.cfm?state=wa&topic=renting

3. Different, but related domains: 4

   A small portion of examined errors was due to URLs pointing to different subdomains of the same website or different, but related domains.

Query URL	Matched URL
https://en.wikipedia.org/wiki/Oregon_Department_of_Revenue	https://en.m.wikipedia.org/wiki/Oregon_Department_of_Revenue
https://www.occ.gov/publications/publications-by-type/other-publications-reports/approach-to-fba-supervision.pdf	https://www.occ.treas.gov/publications/publications-by-type/other-publications-reports/approach-to-fba-supervision.pdf

4. Varied GET parameters: 40

   These types of errors involved almost identical URLs that were only different in the URL parameters. In some cases the different parameters made no impact to the retrieved content, while others did have an effect on the rendered page.

Query URL	Matched URL
http://www.va.gov/directory/guide/state.asp?STATE=NC	https://www.va.gov/directory/guide/state.asp?dnum=ALL&STATE=NC
https://twitter.com/senjohnmccain	https://twitter.com/senjohnmccain?lang=en

5. Geographical location: 51

   Errors belonging to this category resulted from the data being collected in different geographical locations. Errors in this category included different top-level domains and location parameters. Because of the location parameters, this category and the *Varied GET parameters* category are not disjoint.

Query URL	Matched URL
http://www.msn.ie/	http://www.msn.com/
http://www.sigmaaldrich.com/catalog/ product/aldrich/108154?lang=en& region=AU	http://www.sigmaaldrich.com/catalog/ product/aldrich/108154?lang=en& region=US

5.2.2 Other Errors

The errors that did not have identical search result snippets occurred primarily due to poorly represented feature vectors.

For example, one of such errors occurred because the following snippet:

"Toys for sale in Ireland. Buy and sell Toys on DoneDeal.ie. Toys For Sale in Ireland - DoneDeal.ie"

was wrongly matched to:

"Pigs for sale in Ireland. Buy and sell Pigs on DoneDeal.ie. Pigs For Sale in Ireland - DoneDeal.ie"

During feature extraction, 'toys' would have been stemmed to 'toy' and 'pigs' to 'pig'. As both of these are three letters long, both words would have been removed during the last feature extraction step, making the feature vectors identical despite the different snippet strings.

Similarly, the query snippet:

"PowerPoint Presentation. 2000. 2001. 2002. 2003. 2004. 2005. 2006. 2007. 2008. 2009. 2010. 2011. 2012. 2013. 2014. 2000. 2001. 2002. 2003. 2004. 2005. [PPT]PowerPoint Presentation - Code.org"

was matched to the following snippet:

"PowerPoint Presentation2000-2005 (2000s). PowerPoint Presentation. PowerPoint Presentation. PowerPoint Presentation. PowerPoint Presentation. [PPT]PowerPoint Presentation"

In this case, a wrong URL was matched to the query snippet because numbers were not represented as features and the frequency counts were not preserved.

For the two examples and other similar errors, the errors rate could have been reduced further by relaxing the feature extraction process to allow for retention of more features. However, this would mean a much larger feature set and a longer compute time.

5.3 Snippet Matching Robustness

In order to test for the algorithm's robustness to modification of input, three types of input changes could have been considered: insertion, i.e. addition of features to the feature vector, deletion, i.e. the removal of features, and replacement of features. Testing of snippet matching robustness to removal and replacement of features was chosen as insertion was not considered to be an interesting input change for the snippet dataset. Insertion of features would have preserved the original snippet string and a meaningful test in this case would have required a carefully constructed experiment where the snippets were naturally similar to each other. This was not the case for the snippet dataset, as discussed in Section 3.4.2.

The evaluation of snippet matching robustness was carried out in two phases. The first phase tested the algorithm's performance when the query snippets (contained in the snippet-URL repository) underwent feature removal. The second phase evaluated the algorithm's performance when query snippet features were replaced with other features from the dataset.

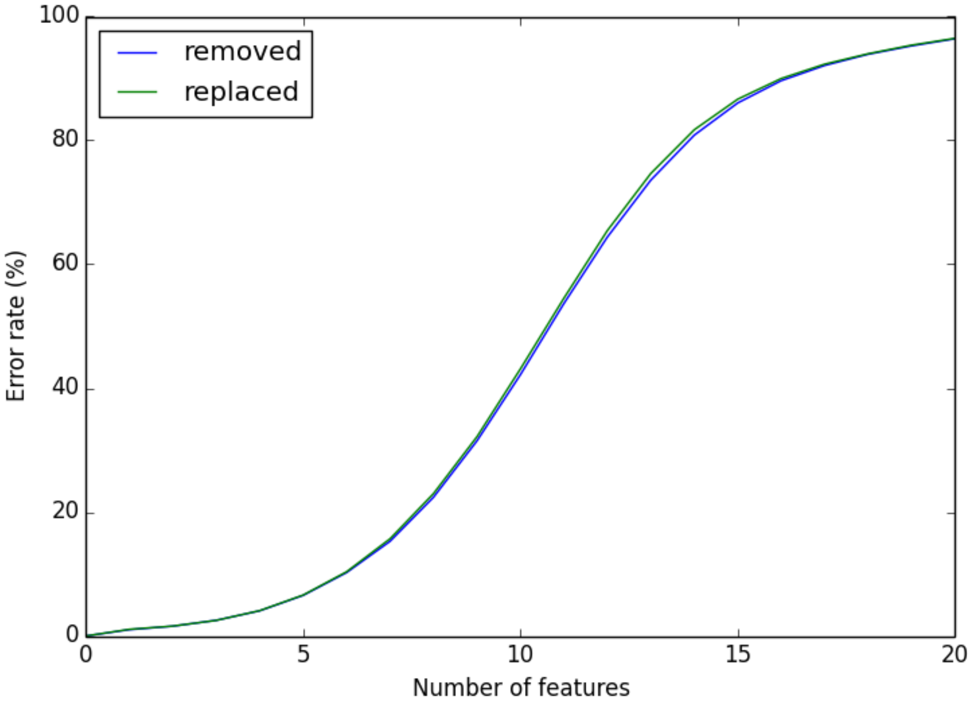


FIGURE 5.1. Snippet matching robustness evaluation results.

Both phases involved randomly sampling a set of 100,000 snippet-URL pairs from the entire snippet dataset. The error rate for this set was calculated. For the first phase, each of the query snippets had a random feature removed and the error rate was recalculated. This process was repeated until up to twenty random features had been removed from each of the query snippets. For phase two, instead of feature removal, a random feature was replaced with a uniformly randomly chosen feature from the feature set. For both phases random sampling was repeated ten times and averages were taken to minimise the sampling bias.

The results of the robustness evaluation are presented in Figure 5.1. From Figure 5.1, it is clear that while feature removal results a marginally smaller error rate, both feature removal and feature replacement have a similar impact on the performance of snippet matching. The error rate in both cases remains below 10% for up to five features removed or replaced. The majority of snippets in the dataset contain around fifteen features, meaning that the algorithm can tolerate replacement or removal of up to a third of the snippet at minimal cost to the error rate. After six features, the error rate grows exponentially until it begins to level off at around seventeen features, when only the longer query snippets still contain some original information.

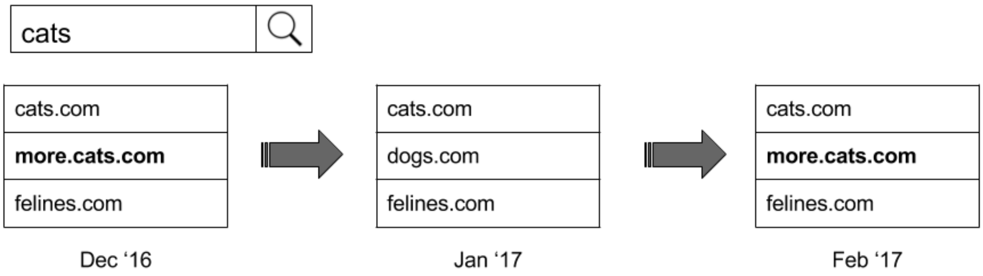


FIGURE 5.2. Example of possible search result sets returned for the search query "cats" requested at different points in time. Note that the domain "more.cats.com" does not appear as a search result in January.

5.4 Snippet Stability

The Irish scraper had scraped Google using the search query set four times over the course of five months in order to collect search result snippet data over time for the snippet stability evaluation. The search queries and the scraping times were the only aspects of the scraping process that were controllable by the scraper. The set of the returned results and their ranking was out of the scraper’s control, meaning that relatively few search results were consistently returned by the search engine during the entire scraping process (see Figure 5.2). As a result, the snippet stability could only be evaluated on a subset (59.5%) of the total number of snippets represented in the dataset because some of the search results were only seen once over the five months.

Each of the search results in each of the search result sets from Figure 5.2 can be thought of as a snippet data point. To construct a representation of each of the search snippets over time, snippet data points were aggregated based on the search result URL and the search query. The search query was used for aggregation because it is possible for the same URL to be returned as a search result with two different snippets if significantly different search queries are used. The aggregated data points were then sorted according to the scraping timestamp to provide an overview of snippet change over time.

Figure 5.3 shows the distribution of the number of snippet data points that were collected per snippet. Although Google had been scraped four times, a small number of snippets had five data points collected over time. This was due to the fact that the search query set had a small number of duplicate queries.

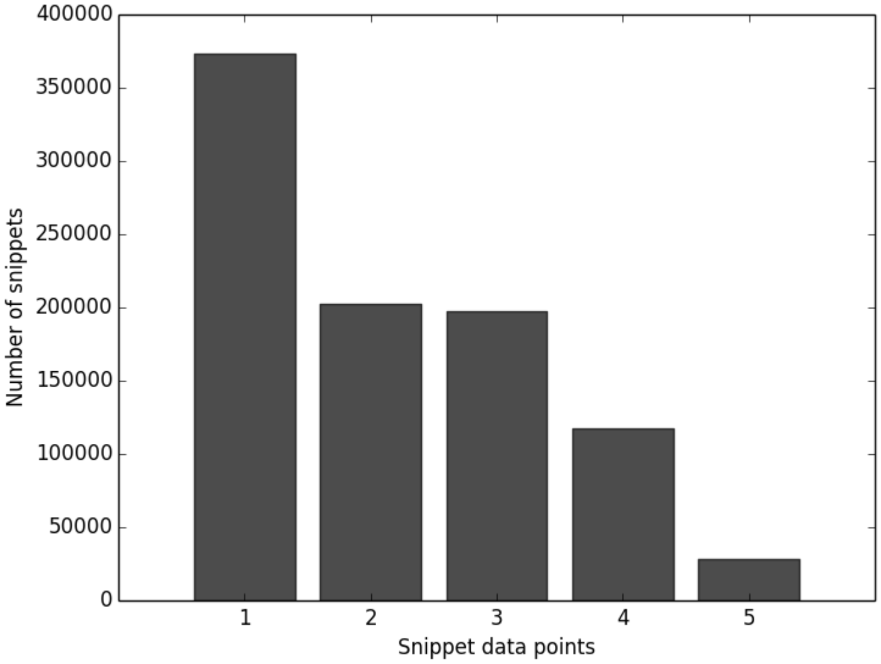


FIGURE 5.3. Distribution of snippet data point sizes collected by the Irish scraper.

Snippet stability was defined as snippet similarity between two consecutive data points for a snippet, i.e. if the snippet remained similar between two scraping events, it was considered to be stable. To measure snippet similarity, the cosine similarity metric was used (see Section 4.3 for the cosine similarity formula). To get the vectors for cosine similarity calculations, the snippet strings were split on whitespace to produce feature vectors as as much of the original snippet string as possible had to be preserved to get accurate results.

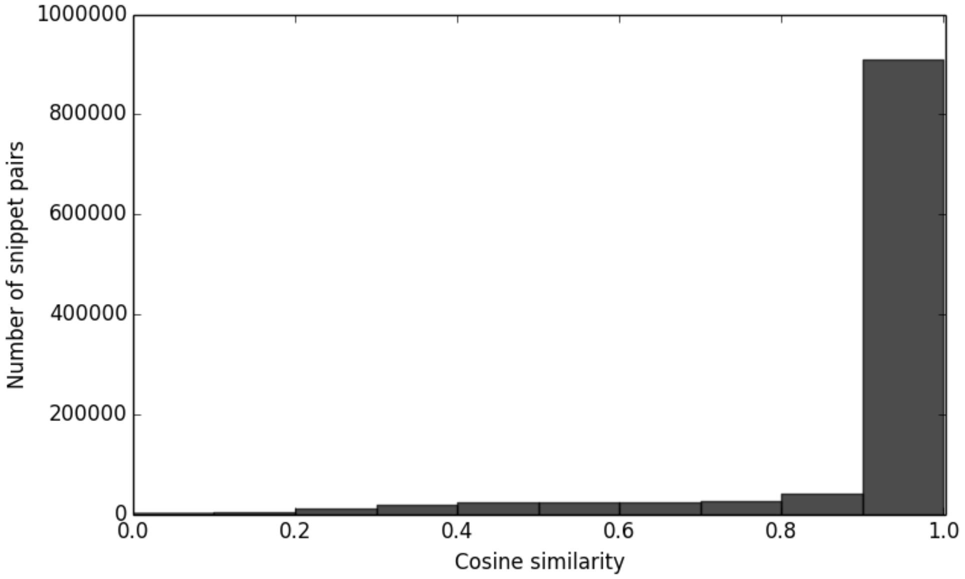


FIGURE 5.4. Cosine similarity of snippet data points.

In total, 920,962 snippet pairs where analysed. The results of the similarity measure between two consecutive snippet data points is shown in Figure 5.4. A cosine similarity value of 1 indicates that the two vectors are maximally similar, while maximally dissimilar vectors have a cosine similarity of 0. 86.79% of the examined snippet pairs had a cosine similarity value of between .9 to 1 (with 79.32% having a similarity value of .99+[1]). This means that the vast majority of snippets did not change at all or did not change significantly over time.

Snippets that were most likely to change over time came from the following types of web pages:

- News media homepages.
- Listings web pages such as property/jobs/advertisement.
- Forum threads.
- Social media such as Facebook and Pinterest.

[1] Cosine similarity of .99+ was used as a number of cosine similarity calculations resulted in floating point errors.

Chapter 6

Conclusion

The main goal of the dissertation was to investigate whether search result snippets could be used to infer search result URLs in the context of a click-tracking blocking system. The results of the dissertation showed that search result snippets can be effectively used for URL inference in such a system. The low error rate of snippet matching, combined with the algorithm's robustness to change and the stability of snippets over time indicate that snippets provide a uniquely identifiable description of their respective URLs.

However, further research will be necessary to expand on the investigation carried out in this dissertation in order to create a complete end-to-end solution to the click-tracking problem. This required research includes click-tracking blocking system design and implementation, a storage hierarchy design for efficient lookup and a scalable snippet matching algorithm.

6.1 Achievements & Limitations

The first objective of the dissertation was to collect a search result snippet dataset. This was completed with scraping of the Google search engine using publicly available search query sets from three geographical locations. The scraping resulted in a sizable dataset of over three million search results that were used to answer the dissertation questions. The search queries used in the scraping process came from sources like Google Trends and Soovle, which list real search queries that are currently popular on the internet. This implies that the collected dataset should be somewhat representative of a dataset that could be gathered by examining real user search patterns and that any conclusions reached in this dissertation are applicable to the real world data. A limitation of such approach is that, while a representative dataset could be collected, the frequency of search query usage is not known, e.g. whether a large number of search engine users consistently look up a small number of queries. Knowing search frequencies would help optimise snippet matching in terms of memory layout and the resolution of search results based on popularity or demand.

The second dissertation goal was to analyse the snippet data to gather insights into the data characteristics for modeling of the snippet text and the implementation of the snippet matching algorithm. The results of snippet analysis indicated that the snippet text was highly diverse, as a lot of features were unique to individual snippets. Preserving as much of the uniqueness of the snippet was key to the successful implementation of the snippet matching algorithm. Investigating snippet stability over time showed that generally search result snippets were unlikely to change over time, with the majority of changes coming from snippets representing social media sites, forums and news media homepages. High snippet stability is of benefit to the click-tracking blocking system, as the latter will not need to constantly update the snippet-URL mappings to provide accurate results.

The third objective was to create a snippet matching algorithm. Based on the snippet analysis, a feature extraction method was developed to convert the collected search result snippets to feature vectors to be processed by the algorithm. Parallelisation techniques helped to speed up the snippet matching process. A key limitation of the snippet matching algorithm is the fact that the algorithm will not scale well as the number of snippet-URL pairs in the repository grows. While the algorithm performed well for the purpose of this dissertation, it will need to be optimised for use in a real click-tracking blocking system. Hence, further research based on real user data would be advantageous to understanding of user search patterns and would aid the development of a scalable snippet matching algorithm.

Lastly, the snippet matching algorithm was evaluated on its error rate to see whether search result snippets could be used to infer search result URLs. The error rate produced by the evaluation was small, at 3.48% for organic search results and 1.74% for sponsored search results, implying that search snippets were a good indication of the search result URLs. Additionally, snippet matching showed robustness to input changes which, together with the high stability of snippets, indicates that snippet matching could be successfully applied to real world data where snippets might undergo minor changes over time. In hindsight, more efforts could have been made to scrape a larger collection of sponsored search results to produce a more meaningful evaluation, as the scraped dataset of 8,264 sponsored snippets is too small for any conclusive results.

6.2 Future Work

This dissertation only serves as a proof-of-concept and significantly more research is required to create a fully-functional click-tracking blocking system. One of the main tasks for future research would be a re-design of the snippet matching algorithm to allow it to scale well with a growing number of snippet-URL pairs.

A possible, user privacy-enhancing solution to the scaling issue would be to allow the user of the click-tracking blocking system to maintain a small, local cache of snippet-URL pairs. Then, every time a search result URL needs to be resolved, a local lookup can be performed on the user's machine which can be escalated to the remote server if the snippet cannot be found in the local cache. By maintaining a local cache, user privacy would be preserved as the remote server will never know exactly what data is being requested by which user.

To allow the snippet matching algorithm to scale, as well as enabling local cache storage, the snippet-URL data will need to be partitioned and categorised in a meaningful way, be it alphabetically, by search result popularity or by search result topics. A further study examining possible partitions of snippet data is therefore required.

BIBLIOGRAPHY

[1] Google AdWords Support - Use ValueTrack parameters with your tracking template. https://support.google.com/adwords/answer/2375447.

[2] NVIDIA CUDA Compute Unified Device Architecture. http://developer.download.nvidia.com/compute/cuda/1.0/NVIDIA_CUDA_Programming_Guide_1.0.pdf, 2007.

[3] Selenium WebDriver. http://www.seleniumhq.org/projects/webdriver/, 2016.

[4] Soolve - Top Internet Keywords. http://soovle.com/top/, 2016.

[5] StatCounter - Top 5 Search Engines in the United States. http://gs.statcounter.com/#desktop-search_engine-US-monthly-201608-201608-bar, 2016.

[6] Yahoo Privacy Policy. https://privacy.microsoft.com/en-us/privacystatement/, 2016.

[7] Bukvarix. http://www.bukvarix.com/top-popular-keywords.html, 2016.

[8] Bing Privacy Policy. https://privacy.microsoft.com/en-us/privacystatement/, 2017.

[9] DuckDuckGo. https://duckduckgo.com/, 2017.

[10] Google Privacy Policy. https://www.google.com/policies/privacy/, 2017.

[11] Ixquick. https://www.ixquick.eu, 2017.

[12] StartPage by Ixquick.
https://www.ixquick.com/, 2017.

[13] Market share held by the leading search engines in the United Kingdom.
https://www.statista.com/statistics/280269/market-share-held-by-search-engines-in-the-united-kingdom/, 2017.

[14] R. Alberdeston, E. Dondyk, and C. C. Zou.
Click-Tracking Blocker: Privacy Preservation by Disabling Search Engines' Click-Tracking.
Globecom 2014 - Communication and Information System Security Symposium, pages 570 – 575, 2014.

[15] P. Mac Aonghusa and D. Leith.
Don't let Google know I'm lonely.
ACM Transactions on Privacy and Security, 19(1):3:1–3:25, 2016.

[16] R. Baier.
Python URL Tools.
https://github.com/rbaier/python-urltools, 2014.

[17] T. Berners-Lee, W3C/MIT, R. Fielding, Day Software, L. Masinter, and Adobe Systems.
RFC 3986: Uniform Resource Identifier (URI): Generic Syntax.
http://www.rfc-editor.org/rfc/rfc1654.txt, 2005.

[18] B. Catanzaro.
OpenCL Optimization Case Study: Simple Reductions.
http://developer.amd.com/resources/articles-whitepapers/opencl-optimization-case-study-simple-reductions/, 2010.

[19] J. B. Earp, A. I. Anton, L. Aiman-Smith, and W. H. Stufflebeam.
Examining Internet Privacy Policies Within the Context of User Privacy Values.
IEEE Transactions on Engineering Management, 52(2):227 – 237, 2005.

[20] A. Hidayat.
PhantomJS.
http://phantomjs.org/, 2016.

[21] J. Allan and B. Carterette and B. Dachev and J. A. Aslam and V. Pavlu and E. Kanoulas.
Million Query Track 2007 Overview.
Text REtrieval Conference (TREC) 2007, 2007.

[22] R. Jones, R. Kumar, B. Pang, and A. Tomkins.
"I Know What You Did Last Summer" - Query Logs and User Privacy.
Proc. 16th Conf. Information and Knowledge Management, pages 909 – 914, 2007.

[23] E. Kao.
Making search more secure.
https://googleblog.blogspot.ie/2011/10/making-search-more-secure.html, 2011.

[24] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih.
PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation.
Parallel Computing, 38(3):157–174, 2012.

[25] E. Loper and S. Bird.
NLTK: The Natural Language Toolkit.
Proc. ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics, 1:63–70, 2002.

[26] M. F. Porter.
Snowball: A language for stemming algorithms.
http://snowball.tartarus.org/texts/introduction.html, 2001.

[27] S. Preibusch.
The Value of Web Search Privacy.
IEEE Security & Privacy, 13(5):24 – 32, 2015.

[28] L. Richardson.
BeautifulSoup.
https://www.crummy.com/software/BeautifulSoup/, 2017.

[29] C. Souvey.
Referer changes for ad clicks.
http://googleadsdeveloper.blogspot.ie/2015/09/referrer-changes-for-ad-clicks.html, 2015.

[30] J. Tompson and K. Schlachter.
An Introduction to the OpenCL Programming Model.
http://cims.nyu.edu/ schlacht/OpenCLModel.pdf, 2012.

[31] V. Toubiana, L. Subramanian, and H. Nissenbaum.
TrackMeNot: Enhancing the privacy of Web Search.
Computing Research Repository, abs/1109.4677, 2011.

[32] J. Turow, J. King, C. J. Hoofnatle, A. Bleakly, and M. Hennessy.
Americans Reject Tailored Advertising and Three Activities that Enable It.
http://dx.doi.org/10.2139/ssrn.1478214, 2009.

[33] K. Wang.
Straight Google.
http://userscripts-mirror.org/scripts/show/121261, 2013.