

Assignment 1
Introduction to Information Retrieval
CS734/834

John Berlin

September 22, 2016

Q.1 Question 1.2

Site search is another common application of search engines. In this case, search is restricted to the web pages at a given website. Compare site search to web search, vertical search, and enterprise search

Q.1 Answer

To compare the four searches I performed the first three using Google and base query “async redux”. The search results shown are limited to the first three in order to have presentable figures. For the site search I used the site *github.com* as the context for the base query is developing web pages in the React.js ecosystem. As seen in figure 1 the results are for libraries that do async actions in redux. The first two is from the redux library itself whereas the third is from a library that enhances that functionality.

The web searches top three results figure 2, featured two results from the online documentation of the library itself and one code example which the first result of the site search. These results were as expected as the query was missing the context given to search engine through the “site:github.com” as is done in site search. The context is key when comparing site search to web search as it effects the precision of these queries rather than recall. For example, I made this exact query recently when looking into how to better handle the async problems faced by WAIL¹ which is using flux (the predecessor to redux). I performed the web search first as I wanted to see in the documentation on how redux handled async operations rather than diving into the code bases of the libraries themselves. Google understood the likely hood of this happening for the base query and was precise in its top three results given the lack of context which is present in the site search.

Vertical search is done when only results of a certain type are desired which is another contextual based query. The results of this search, using Google Videos figure 3, in comparison to the site search can be described as combination of the web search and the site site search. The first two results of this query are a combination of all three results from the web search plus the first and second of the site search. The last result of the vertical search features content from the third result from the site search.

In order to compare enterprise search to site search I will use two examples: the first will be the classical example using my personal computers file search. The second will be using more targeted search on my personal computer within the context of the previous two comparisons. As I do not have a large corporate intranet to search I use *Desktop Search* in this example. Desktop search, as defined in our book, is the personal version of enterprise search [2, pp. 3]. I felt this is a correct substitution because the previous two comparisons used examples about how I personally used the them for WAIL. Furthermore if a team of developers was looking to find out where on their computers which libraries are currently being used for async operations figure 4(a) and then where in the repository is the functionality implemented figure 4(b).

Enterprise Search (*Desktop Search*) returned results that had the search term in them not necessarily the most relevant figure 4(a). The search term async can be found in directory names as well as many duplicates. The duplicates in these results are a product of the search being done on the file system where duplicates are acceptable. Unlike site search which returned unique results where duplicates are not as common. Whereas if you were to use the Linux utility *grep*, you can perform the same kind of query but limited to lines contained in the files in a specific part of the file system figure 4(b), much like site search. It is clear to see that enterprise search is a broader type of search yet is done a specific site(file system).

¹<https://github.com/N0taN3rd/wail>

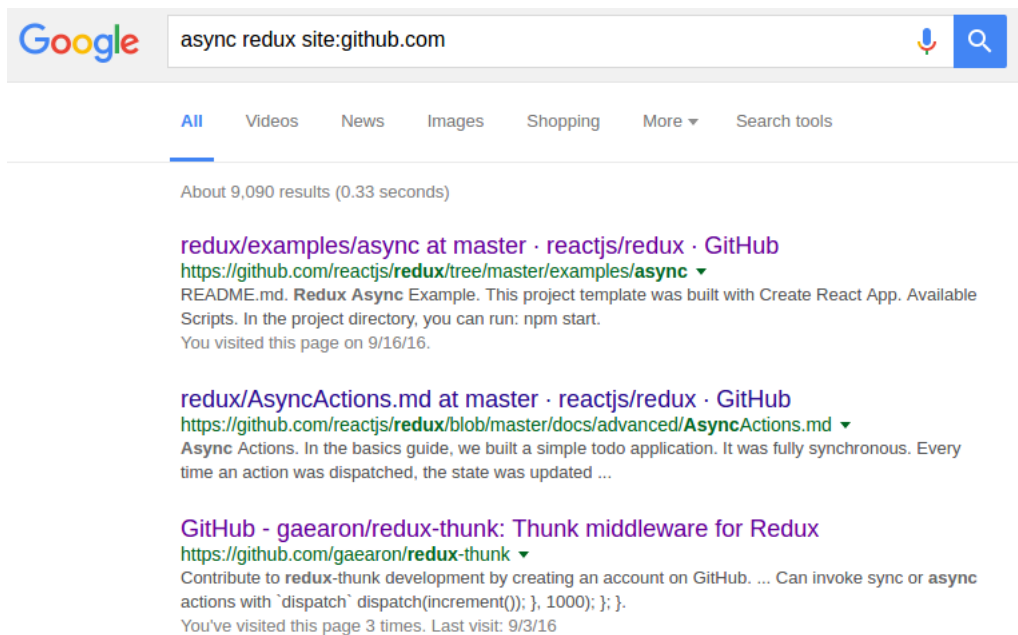


Figure 1: Site Search

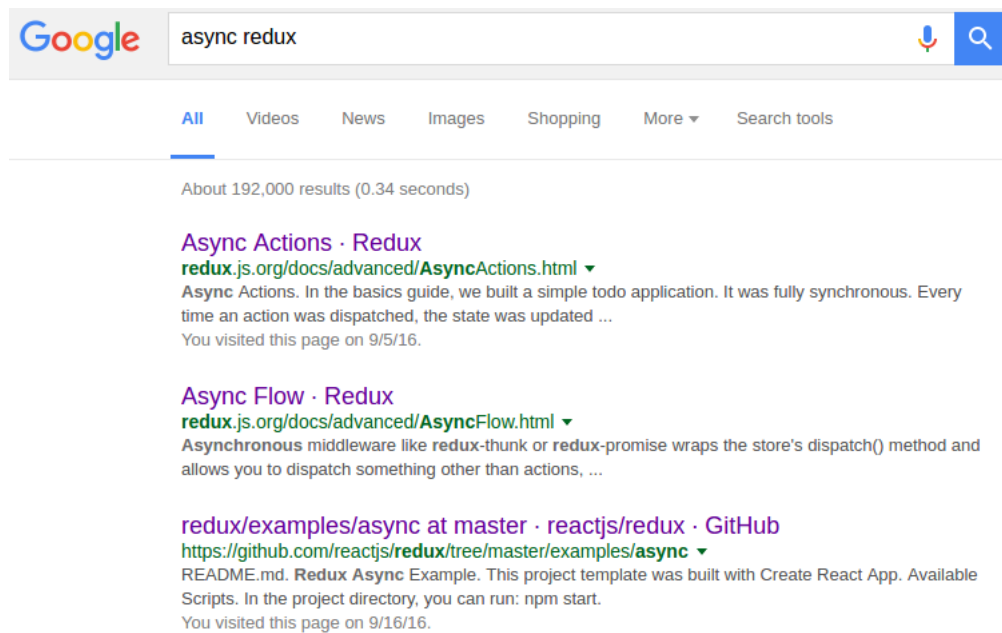


Figure 2: Web Search

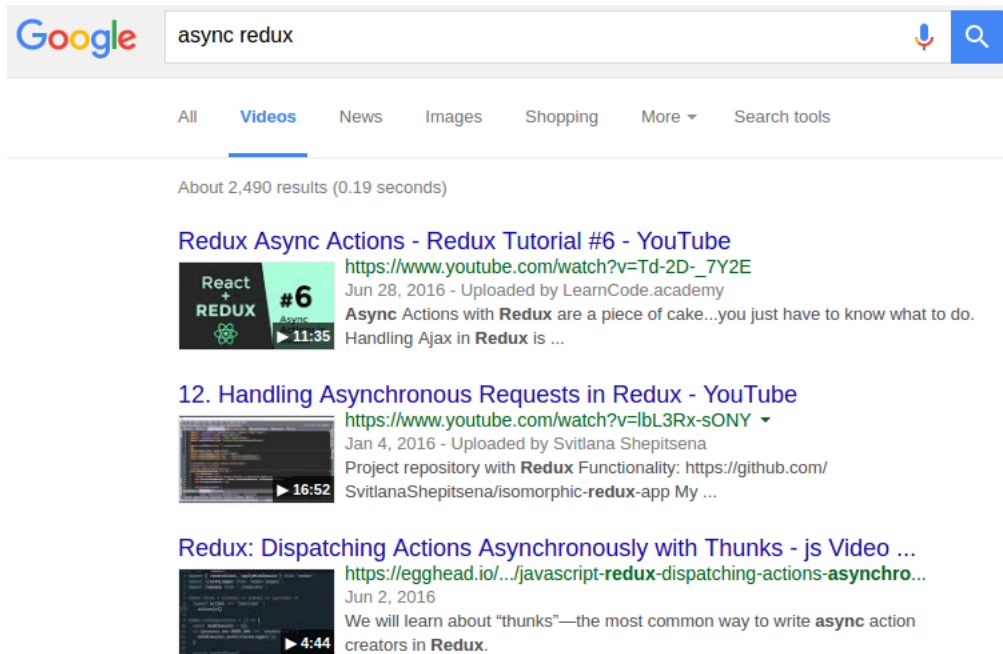
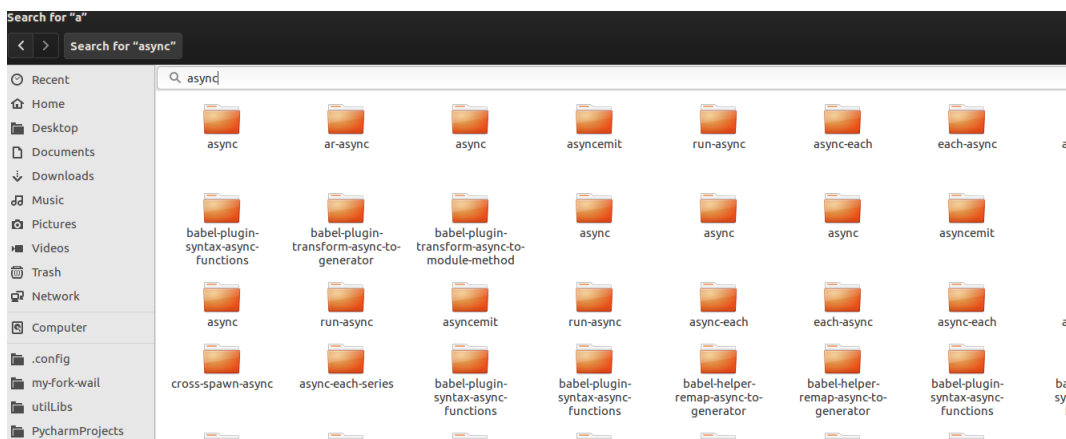
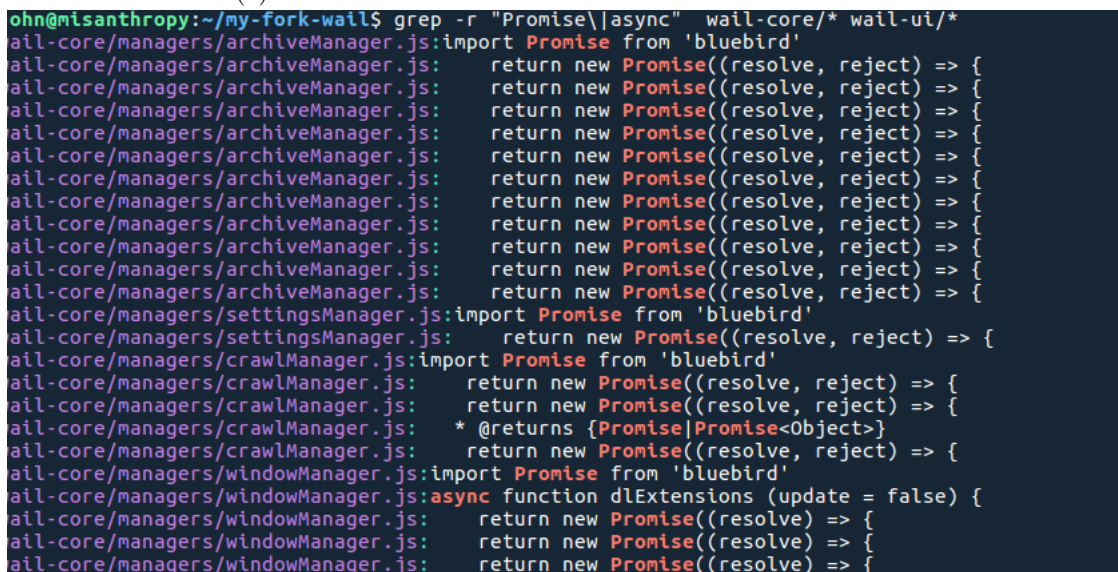


Figure 3: Vertical Search



(a) Classic



(b) Targeted

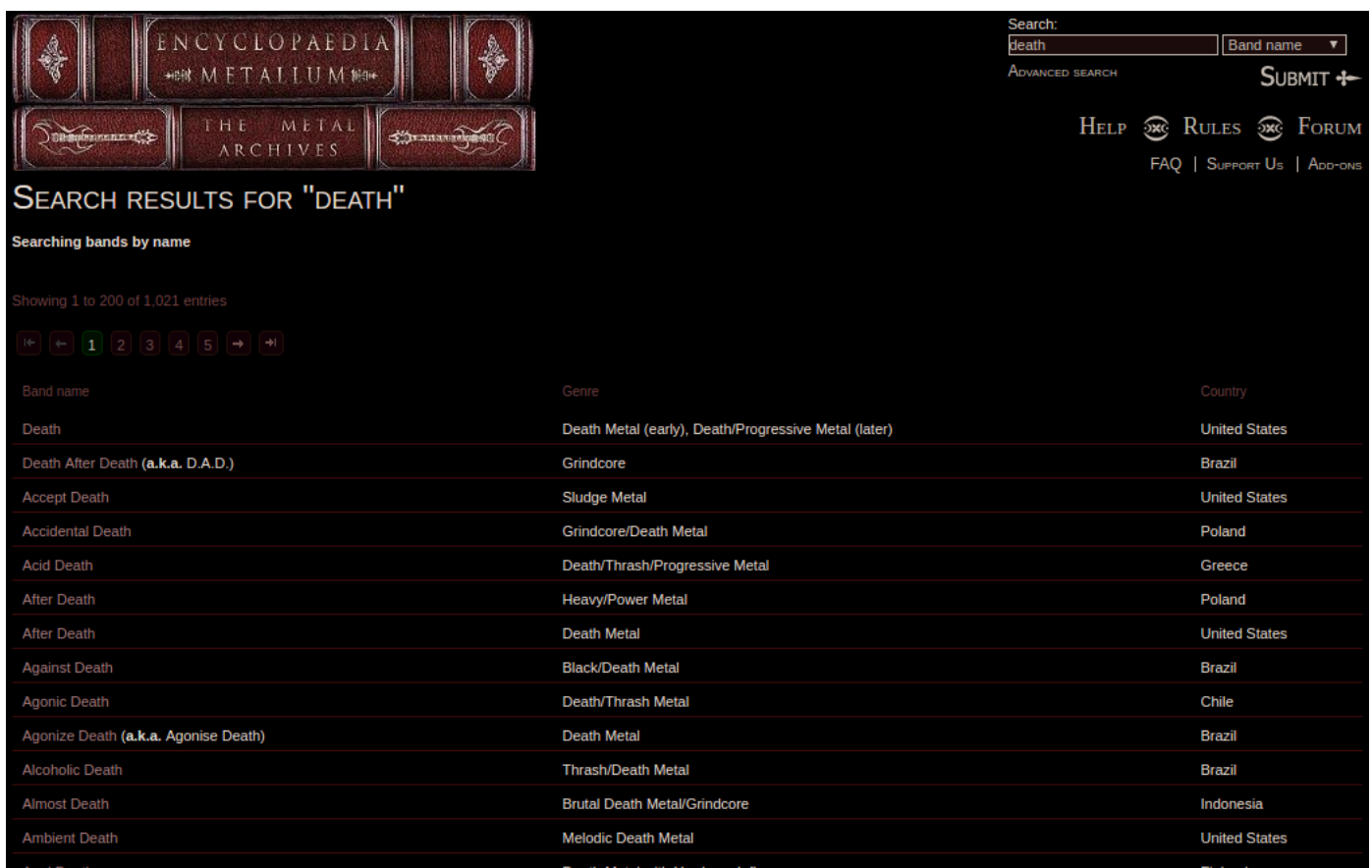
Figure 4: Enterprise (Desktop) Search

Q.2 Question 1.4

List five web services or sites that you use that appear to use search, not including web search engines. Describe the role of search for that service. Also describe whether the search is based on a database or grep style of matching, or if the search is using some type of ranking.

Q.2 Answer

The first website I use that features database style matching is metal-archives.com. Metal archives is a site which allows for you search for a metal band(s) by name, genre, album and many more. You can tell it uses database style search by the search interface it uses and by examining the query string used to produce Figure 5.



The screenshot shows the Metal Archives website with a search bar at the top right containing the text 'death'. Below the search bar, the text 'SEARCH RESULTS FOR "DEATH"' is displayed. A table of search results follows, listing bands, their genres, and their countries. The table is titled 'Searching bands by name' and shows 'Showing 1 to 200 of 1,021 entries'. The table has three columns: 'Band name', 'Genre', and 'Country'. The first row is 'Death', 'Death Metal (early), Death/Progressive Metal (later)', and 'United States'. The second row is 'Death After Death (a.k.a. D.A.D.)', 'Grindcore', and 'Brazil'. The third row is 'Accept Death', 'Sludge Metal', and 'United States'. The fourth row is 'Accidental Death', 'Grindcore/Death Metal', and 'Poland'. The fifth row is 'Acid Death', 'Death/Thrash/Progressive Metal', and 'Greece'. The sixth row is 'After Death', 'Heavy/Power Metal', and 'Poland'. The seventh row is 'After Death', 'Death Metal', and 'United States'. The eighth row is 'Against Death', 'Black/Death Metal', and 'Brazil'. The ninth row is 'Agonic Death', 'Death/Thrash Metal', and 'Chile'. The tenth row is 'Agonize Death (a.k.a. Agonise Death)', 'Death Metal', and 'Brazil'. The eleventh row is 'Alcoholic Death', 'Thrash/Death Metal', and 'Brazil'. The twelfth row is 'Almost Death', 'Brutal Death Metal/Grindcore', and 'Indonesia'. The thirteenth row is 'Ambient Death', 'Melodic Death Metal', and 'United States'. The fourteenth row is 'Anal Death', 'Death Metal with Hardcore Influences', and 'Finland'.

Band name	Genre	Country
Death	Death Metal (early), Death/Progressive Metal (later)	United States
Death After Death (a.k.a. D.A.D.)	Grindcore	Brazil
Accept Death	Sludge Metal	United States
Accidental Death	Grindcore/Death Metal	Poland
Acid Death	Death/Thrash/Progressive Metal	Greece
After Death	Heavy/Power Metal	Poland
After Death	Death Metal	United States
Against Death	Black/Death Metal	Brazil
Agonic Death	Death/Thrash Metal	Chile
Agonize Death (a.k.a. Agonise Death)	Death Metal	Brazil
Alcoholic Death	Thrash/Death Metal	Brazil
Almost Death	Brutal Death Metal/Grindcore	Indonesia
Ambient Death	Melodic Death Metal	United States
Anal Death	Death Metal with Hardcore Influences	Finland

Figure 5: Metal Archives Death

The query string "search?searchString=death&type=band_name" shows this site is talking to its REST backend. REST or REpresentational State Transfer is a means to talk to a web service most commonly backed by a database. As seen in Figure 5 I am very happy to see that the Floridian death metal pioneers *Death* are at the top of the list for the band name search death. But sadly Stormtrooper of Death (S.O.D) are not in the top results which is a Scott Ian (Anthrax) side project.

The second is github which features a mix of database search with grep style searching. Figure 6 shows the result of search for quick sort on github. The default search results are repositories with the search terms contained in there name and or in the Readme.md contained in the repo itself hence grep. Along with the repository search you can also search code i.e code lines for the presence of your search terms. Further solidifying the declaration of mixed is the ability to do “git grep” to search for a term across all branches in your repository and that the query string used to produce Figure 6 “search?utf8=%E2%9C%93&q=quick+sort” indicates another RESTful style web api.

The screenshot shows the GitHub search interface. At the top, there's a navigation bar with 'Pull requests', 'Issues', and 'Gist'. Below it, a search bar contains 'quick sort' and a 'Search' button. The main content area displays 'We've found 1,184 repository results'. On the left, there's a sidebar with filters for 'Repositories' (1,184), 'Code' (7,855,406), 'Issues' (100,775), 'Wikis' (10,010), and 'Users' (1). Below this is a 'Languages' section with a bar chart showing counts for Java (301), C++ (214), C (115), JavaScript (75), Python (68), C# (61), Objective-C (29), Ruby (29), PHP (16), and HTML (15). The main results list shows three repositories: 'allmycode/sort' (C++), 'cave7/sort' (C), and 'ippeikino/Quick-Sort' (C). Each result includes the repository name, a snippet of the README, the language, star count, fork count, and the last update date. At the bottom, there are links for 'Advanced search' and 'Cheat sheet'.

Repository	Language	Stars	Forks	Last Updated
allmycode/sort	C++	2	8	Sep 5, 2012
cave7/sort	C	3	3	Sep 12, 2012
ippeikino/Quick-Sort	C	2	4	Jun 20, 2011
carlocombate/Quick	C	0	0	Feb 10

Figure 6: Github Quick Sort

npmjs.com is another service I use to look when looking for Nodejs packages. Figure 7 shows the results when searching for react packages which produces the query string search?q=react. Npmjs.com is the web site for npm, node package manager, which is a service that manages node libraries much like github. It is clear to see that npmjs.com is database search.

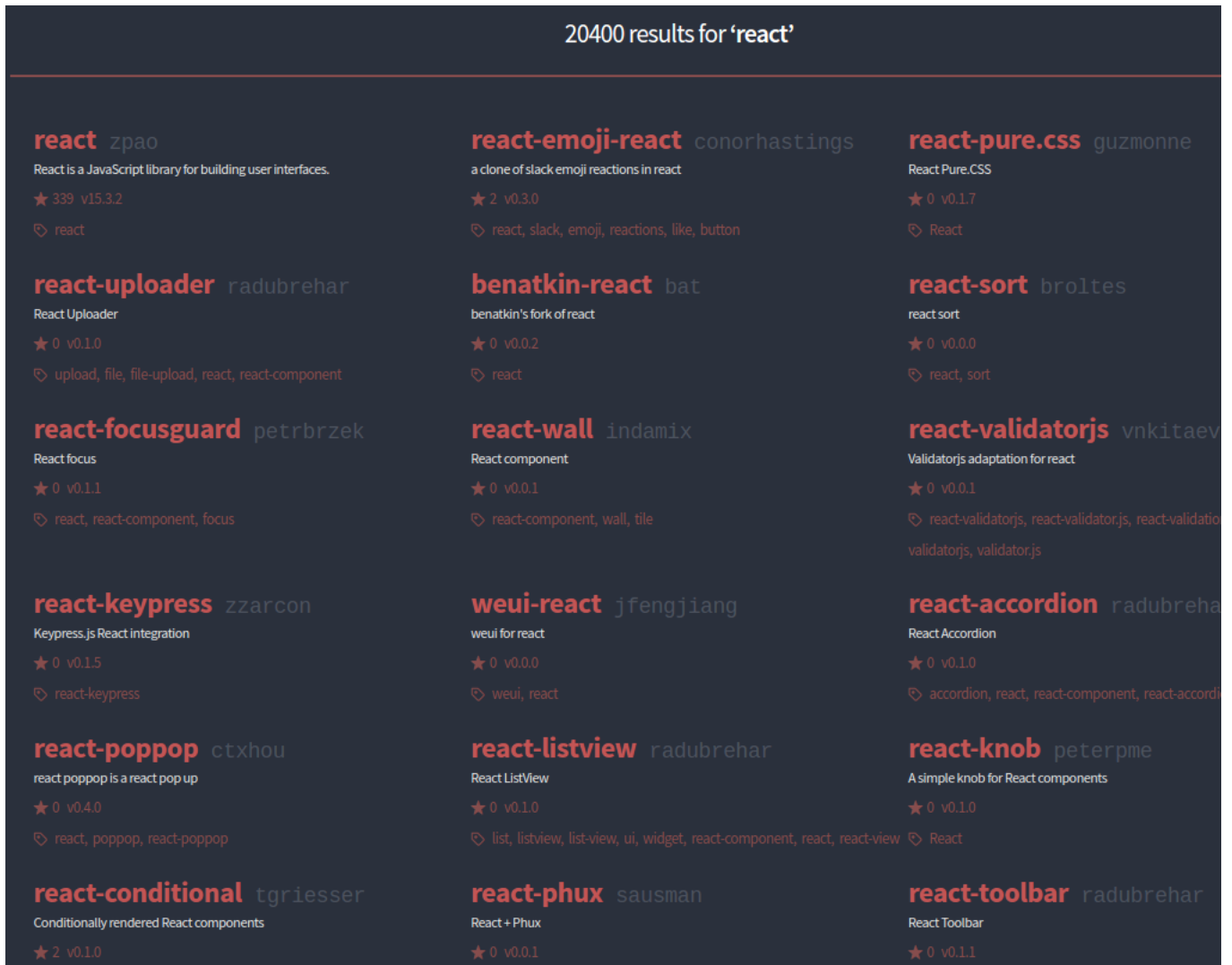


Figure 7: NPM React

Another database style searched website I use is tex.stackexchange.com which features the power of stackoverflow but for L^AT_EX related issues. Figure 8 shows the results when searching for questions tagged with bibliographies sorted by frequency. The query string that goes along side this search is “questions/tagged/bibliographies?sort=frequent” and once again indicates the search was executed against a database.

The screenshot displays the Tex Stack Exchange interface. At the top, there's a header with the TeX logo and navigation tabs: Questions, Tags, Users, Badges, and Unanswered. Below this, the 'Tagged Questions' section is active, showing a list of questions sorted by frequency. The first question is 'Question mark or bold citation key instead of citation number' with 97 votes and 2 answers. The second is 'bibtex vs. biber and biblatex vs. natbib' with 280 votes and 5 answers. The third is 'Biblatex/biber fails with a strange error about missing recode data.xml file' with 134 votes and 2 answers. The fourth is 'Using BibTeX to make a list of references without having citations in the body of the document?' with 85 votes and 1 answer. On the right side, there's a sidebar showing '582 frequent questions tagged' with the tag '{bibliographies}' and a list of related tags: {bibtex}, {biblatex}, {citing}, {natbib}, {subdividing}, {biber}, {urls}, {lyx}, {beamer}, and {apa-style}.

Figure 8: Tex Stack Exchange Bibliographies

Finally my most frequented and favorite database probably grep search style website bandcamp. Figure 9 shows the results when searching for my favorite sub-genre of punk which is power violence. The query string that accompanies this result is “tag/power-violence”. The results of this query are somewhat disappointing as the band ACxDC dominates the results. They are not a bad band but are the like the Metallica of power violence but eclipse bands like Chiens, whose only recognition in the top results is the split with ACxDC. Great bands like Insect Warfare and Apartment 213 still made it in the top results tho. But the greatest sadness in this is that the power violence powerhouse Gets Worse is no where to be seen. They have been pumping out eps at a steady rate since 2012 all of which blow away ACxDC in terms of the ferocity of their raw and gritty sound continually perfected with each release.

Browse best-sellers, new releases, artist recommendations, vinyl, and more.

power violence

[browse all tags](#)

related tags: [fastcore](#) [thrashcore](#) [grind core](#) [grind](#) [powerviolence](#) [crust punk](#) [grindcore](#) [thrash](#) [crust](#) [sludge](#) [hardcore punk](#) [noisecore](#) [hardcore punk](#) [hardcore](#) [doom](#) [screamo](#) [punk](#) [harsh noise](#)

[best-selling](#) [new arrivals](#)

<p>The Oracles of Death EP ACxDC</p>	<p>I blight worms</p>	<p>PSYCHO 001_INSECT ... Psychocontrol records</p>	<p>Antichrist Demoncore ACxDC</p>	<p>METH DRINKER / DEA... Drop Out</p>
<p>Paper Cage Rape Revenge</p>	<p>Postcard Flexi ACxDC</p>	<p>Split with Disparo ACxDC</p>	<p>Split 5" with Chiens ACxDC</p>	<p>Preview of UK Tour Tap... ACxDC</p>
<p>ACxDC The Second Coming</p>	<p>ACxDC he had it coming</p>	<p>apartment 213 cleveland power violence</p>	<p>DEMO 2016</p>	<p>Antichrist Demoncore</p>

Figure 9: Bandcamp Power Violence

Q.3 Question 3.7

Write a program that can create a valid sitemap based on the contents of a directory on your computer's hard disk. Assume that the files are accessible from a website at the URL `http://www.example.com` . For instance, if there is a file in your directory called `homework.pdf` , this would be available at `http://www.example.com/homework.pdf` . Use the real modification date on the file as the last modified time in the sitemap, and to help estimate the change frequency.

Q.3 Answer

The code for this question can be seen in the source code listing 1. The program utilizes three libraries which made this question easy: `PyFilesystem` [4] for its combination of listing the contents of a directory and file stats in a single function call, `Arrow` [1] the python version of `Momentjs` and `Yattag`[5] pythonic document creation. Using python's argument parser library to get the directory for which to create the sitemap, the program lists the contents of the directory and if the name of the file is acceptable, determined by “-dots” (include . files or not) argument create a new xml node where the last modified time is converted into utc and finally after the files of the directory have been read output the created site to stdout. Listing 1 shows the output of code listing 1 when run on a directory containing the wiki for WAIL.

Listing 1: Sitemap of the Wiki for WAIL

```
<urlSet xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.example.com/Using+Wail.md</loc>
    <lastmod>2016-08-03T03:32:46.439265+00:00</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/Home.md</loc>
    <lastmod>2016-08-03T03:10:43.711074+00:00</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/Getting+Started.md</loc>
    <lastmod>2016-08-03T03:38:06.393983+00:00</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/Screen-Shots.md</loc>
    <lastmod>2016-08-03T00:59:45.379063+00:00</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/FAQ.md</loc>
    <lastmod>2016-08-03T00:59:45.379063+00:00</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/images/misc.png</loc>
    <lastmod>2016-08-02T18:40:20+00:00</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/images/wailSideBar.png</loc>
    <lastmod>2016-08-02T18:38:14+00:00</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/images/services.png</loc>
    <lastmod>2016-08-02T18:40:04+00:00</lastmod>
  </url>
```

```

</url>
<url>
  <loc>http://www.example.com/images/frontpage.png</loc>
  <lastmod>2016-08-02T18:38:48+00:00</lastmod>
</url>
<url>
  <loc>http://www.example.com/images/heritrixRunningCrawlOptions.png</loc>
  <lastmod>2016-08-02T18:41:04+00:00</lastmod>
</url>
<url>
  <loc>http://www.example.com/images/wayback.png</loc>
  <lastmod>2016-08-02T18:39:04+00:00</lastmod>
</url>
<url>
  <loc>http://www.example.com/images/configureNewCrawl.png</loc>
  <lastmod>2016-08-02T18:39:44+00:00</lastmod>
</url>
<url>
  <loc>http://www.example.com/images/heritrix.png</loc>
  <lastmod>2016-08-02T18:39:22+00:00</lastmod>
</url>
<url>
  <loc>http://www.example.com/images/settings.png</loc>
  <lastmod>2016-08-02T18:40:44+00:00</lastmod>
</url>
</urlSet>

```

```

from urllib.parse import quote_plus as quote
from argparse import ArgumentParser
from arrow import Arrow
from fs.osfs import OSFS
from yattag import Doc, indent
from shared_code import FullPaths, is_dir

```

```

dots = False

```

```

def accept(name, moreDots=dots):

```

```

    if moreDots:
        return True
    else:
        return not name.startswith('.')

```

```

if __name__ == '__main__':

```

```

    parser = ArgumentParser(description="Create a sitemap of a directory on your local file system",
        prog='sitemap',
        usage='%(prog)s [options]')
    parser.add_argument('-dir', '--directory', help='directory to use', action=FullPaths, type=is_dir)
    parser.add_argument('-dots', help='include dot files', action='store_true')
    args = parser.parse_args()
    dir = OSFS(args.directory)
    dots = args.dots
    dirs = []
    skipFirst = True
    doc, tag, text = Doc().tagtext()
    with tag('urlSet', xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"):
        for name, stats in dir.listdirinfo(files_only=True):
            if accept(name):

```

```

        with tag('url'):
            with tag('loc'):
                text('http://www.example.com/%s' % quote(name))
            with tag('lastmod'):
                text(str(Arrow.utcfromtimestamp(stats['modified_time'].timestamp()))))
    for aDir in dir.walkdirs():
        if skipFirst:
            skipFirst = False
            continue
        if accept(aDir[1:None]):
            for name, stats in dir.listdirinfo(path=aDir, files_only=True):
                if accept(name):
                    with tag('url'):
                        with tag('loc'):
                            text('http://www.example.com%s/%s' % (aDir, quote(name)))
                        with tag('lastmod'):
                            text(str(Arrow.utcfromtimestamp(stats['modified_time'].timestamp()))))

    dir.close()
    result = indent(
        doc.getvalue(),
        indentation=' ' * 2,
        newline='\r\n'
    )

    print(result)

```

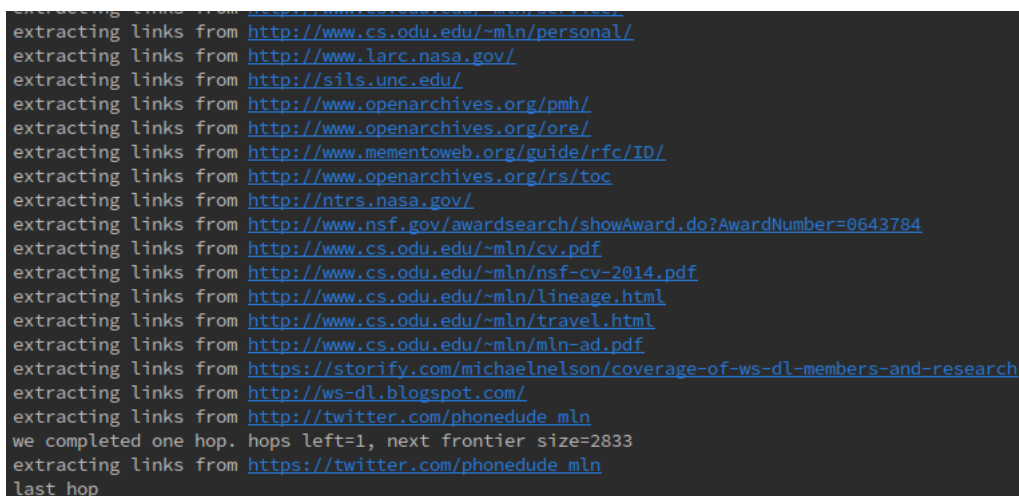
Source Code 1: Python Directory Sitemap Generation

Q.4 Question 3.9

Write a simple single-threaded web crawler. Starting from a single input URL (perhaps a professor's web page), the crawler should download a page and then wait at least five seconds before downloading the next page. Your program should find other pages to crawl by parsing link tags found in previously crawled documents.

Q.4 Answer

The solution to this question is a modified version of the solution submitted for assignment 1 for cs532 and the source can be seen in source code listing 2. The modifications made to the original were minor instead of looking for links that resolve to pdf files the code now extracts all links from a uri that resolves to a 200 and puts them into a queue the frontier_gen function. The first uri or seed is placed in the queue first and the first call to frontier_gen is made. Then after waiting for five sections the next uri is popped from the queue and the links contained in it are added. I have added an additional part to this assignment which is a stopping criteria hops. Hops is the number links away from seed url and the reason for this stipulation can be seen in figure 10. After the first hope the frontier has reached a size of 2833 uris waiting to be processed after visiting links one hope away from the seed url <http://cs.odu.edu/~mln>. I also did this from experience gained while working with Heritrix which will download the entire internet if it is not given explicit hop bounds. The current hop the program is at is determined by decrementing the frontier count gotten at the start of next hop by taking the length of the queue. When it reaches zero the last uri for the final_hop has its links extracted and the remaining uris are printed.



```
extracting links from http://www.cs.odu.edu/~mln/personal/
extracting links from http://www.larc.nasa.gov/
extracting links from http://sils.unc.edu/
extracting links from http://www.openarchives.org/pmh/
extracting links from http://www.openarchives.org/ore/
extracting links from http://www.mementoweb.org/guide/rfc/ID/
extracting links from http://www.openarchives.org/rs/toc
extracting links from http://ntrs.nasa.gov/
extracting links from http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0643784
extracting links from http://www.cs.odu.edu/~mln/cv.pdf
extracting links from http://www.cs.odu.edu/~mln/nsf-cv-2014.pdf
extracting links from http://www.cs.odu.edu/~mln/lineage.html
extracting links from http://www.cs.odu.edu/~mln/travel.html
extracting links from http://www.cs.odu.edu/~mln/mln-ad.pdf
extracting links from https://storify.com/michaelnelson/coverage-of-ws-dl-members-and-research
extracting links from http://ws-dl.blogspot.com/
extracting links from http://twitter.com/phonedude_mln
we completed one hop. hops left=1, next frontier size=2833
extracting links from https://twitter.com/phonedude_mln
last hop
```

Figure 10: Frontier Size

```
import re
import requests
from urllib.parse import urljoin
from bs4 import BeautifulSoup
from collections import deque
from argparse import ArgumentParser
import time

reg_s = "((([A-Za-z]{3,9}:(?!(\\|/)?)(?:[\\-;:&=\\+\\$,\\w]+@)?[A-Za-z0-9\\.\\-]+|\" + \\
    \"(?:www\\.|[\\-;:&=\\+\\$,\\w]+@)[A-Za-z0-9\\.\\-]+)((?:\\|/[\\+~%\\|\\.|\\w\\-]*)?)\" + \\
    \"\\??(?:[\\-\\+&=%@\\.\\w]*)#?(?:[\\.|\\|/\\\\\\w]*)?)\"")
# my standard url regex found a while ago
url_re = re.compile(reg_s, re.IGNORECASE)

relative = re.compile(\"^(?!www\\.|(?:(http|ftp)s?://|\\[A-Za-z]:\\\\\\\\|/|/)(#[a-zA-Z0-9\\-\\_]+)).*\")
```

```

def gen_frontier(uri, session, q, seen):
    """
        Simplistic frontier generation function
        Download the current uri and if it resolves to a 200
        extract all the a elements.
        If we have seen the link contained in an href of the current
        a element skip it otherwise add it to our queue.
        A sanity check uri regex is utalized
        if it fails check if the uri is relative
        otherwise skip it
    """
    print('extracting links from %s'%uri)
    r = session.get(uri) # type: requests.Response
    ctype = r.headers.get('Content-Type', 'why you no tell me').lower()
    if r.ok and 'text/html' in ctype:
        try:
            s = BeautifulSoup(r.text, 'html5lib')
        except:
            # just because if this fails there are problems
            s = BeautifulSoup(r.text, 'html5lib')
        all_a = s.find_all('a', href=True)

        for link in map(lambda a: a['href'], all_a):
            if link not in seen:
                # print("extracting links from %s" % uri)
                if url_re.match(link):
                    print('found %s'%link)
                    q.append(link)
                    seen.add(link)
                else:
                    if relative.match(link):
                        print('found relative %s' % link)
                        fulllink = urljoin(r.url, link)
                        if fulllink not in seen :
                            q.append(fulllink)
                            seen.add(fulllink)
                    else:
                        print("The input uri %s failed to pass my regex " % link, relative)
            else:
                print("We have a link that does not resolves to an ok or is not text/html: %s, %d, %s" % (uri,
                    ↪ r.status_code, ctype))

if __name__ == '__main__':
    parser = ArgumentParser(description="Single Threaded Crawler", prog='crawler', usage='% (prog)s
    ↪ [options]')
    parser.add_argument('-s', '--seed', help='seed to start crawling', type=str,
    ↪ default='http://cs.odu.edu/~mln')
    # ok I am adding this to the program as having worked with Heritrix an unbounded crawler will
    ↪ download the internet
    parser.add_argument('-hops', help='how many hops from the start should the crawler crawl',
    ↪ type=int,
    ↪ default=2)
    args = parser.parse_args()
    useragent = 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:44.0) Gecko/20100101 Firefox/44.01'
    hops = args.hops
    q = deque()
    # add the initial seed url to queue
    q.append(args.seed)
    # simple attempt at avoiding crawler traps i.e. links that reference the page containing them
    seen = set()

    with requests.Session() as session:
        session.headers.update({'User-Agent': useragent})

```

```

uri = q.popleft()
seen.add(uri)
# extract all uris from the seed and add it to seen
gen_frontier(uri, session, q, seen)
# get the number of uris in the frontier for hop 1
frontierCur = len(q)
# continue crawling until we are at the last hop
while hops > 0:
    time.sleep(5)
    uri = q.popleft()
    frontierCur -= 1
    # vist this uri and extract its uris
    gen_frontier(uri, session, q, seen)
    # when the frontier for the current hop has been exhausted
    if frontierCur == 0:
        # get the new frontier size
        frontierCur = len(q)
        # decrement hops as we are now one curHop+1 away from the seed uri
        hops -= 1
        print('we completed one hop. hops left=%d, next frontier size=%d' % (hops,
            ↪ frontierCur))
# we are now at the last hop exhaust the queue
print("printing uris at hope %d away from the seed"%args.hops)
while True:
    try:
        uri = q.popleft()
        print('uri gotten from last hop'%uri)
    except IndexError:
        break

```

Source Code 2: Python Single Threaded Crawler

Q.5 Question 3.14

Write a program to generate simhash fingerprints for documents. You can use any reasonable hash function for the words. Use the program to detect duplicates on your home computer. Report on the accuracy of the detection. How does the detection accuracy vary with fingerprint size?

Q.5 Answer

In order to answer this question I used six files with following naming scheme: *name.ext*, *not_name.ext* and *name_same.ext*. The first set of three used the syllabus for this course for the base and similar files with the not similar file being the assignment description for websci assignment 1. The second set uses the L^AT_EX source submitted for websci assignments 1 and 9. The second set's base and same file is assignment 9 source and the not similar file is the source for assignment 1. These files can be found in the directory *similarDocs* which accompanies this report.

The hashing function used for the words or tokens as I refer to them in Source Code 3 is md5. Md5 is used as it has the property of given an input it will produce the same hash on consecutive usage of the function on the same input. Before calculation of the hash for each token the documents text was sanitized by removing the punctuation and tokens that were found to be in a stop word list provided by *NLTK* [3] or less than two characters long. For all tokens that passed the filtering criteria their weights were calculated as the frequency in which they appeared, for example if a token appeared in the document twice it was assigned the the weight of two. After the token weights were calculated each of the tokens were hashed. Md5 in python can return the digest of the hash in two forms: the digest itself or an hexadecimal representation of the digest. The hex digest was used as it is the quickest to transform into a binary string. The conversion from hex to binary is as follows: for each character in the hexadecimal representation of the digest, determine its ordinal value and represent it as a binary value.

Three cases were used to test the effectiveness of this implementation of simhash. The first used a bit length of 8 which was used in the example found in the book [2, pp. 64] as a base line. The results Figure 11 show that simhash was able to determine the duplicates syllabus_same.txt, syllabus.txt and websci_assignemnt9.tex, websci_assignemnt9_same.tex both of which had similarity scores of 1.0. The duplicates in this test when compared to the "same" files both had scores of 0.8944. This seems odd but will be explained in the next test.

```
Comparing files contained in the directory /home/john/Documents/cs834-f16/assignments/a1/code/similarDocs
-----
using bit len of 8
syllabus.txt not_syllabus.txt 0.894427191
syllabus.txt syllabus_same.txt 1.0
websci_assignemnt9.tex not_websci_assignemnt9.tex 0.894427191
websci_assignemnt9_same.tex websci_assignemnt9.tex 1.0
websci_assignemnt9_same.tex not_websci_assignemnt9.tex 0.894427191
not_syllabus.txt syllabus_same.txt 0.894427191
```

Figure 11: Simhash similarity using bit length of 8

The second test used the minimum length of the bit string produced over all terms by md5. Table 1 shows the maximum hash length produced by md5 for a document to help get an idea of the size of the hashes when considering it's role in the similarity calculations.

As seen in table one the lengths of the hash can get large. Which in turns leads to the conclusion to be that the differences in the hashes must not be in the most significant bits in the binary representation of the hash. It must be noted that since the L^AT_EX files used share markup terms this would lead to them having similar maximum hash lengths even after punctuation has been removed. To compensate for this case the minimum hash length was used for the number of bits in the similarity computation as seen in Figure 12.

The minimum hash length is close to for both document types 196 syllabus and 197 websci but the

file	max hash length
websci.assignemnt9.tex	213
not_websci.assignemnt9.tex	213
syllabus.txt	210
not_syllabus.txt	211

Table 1: Max bit length of terms produced by md5

```

Using bit length determined by the length of the md5 hash
Comparing files contained in the directory /home/john/Documents/cs834-f16/assignments/a1/code/similarDocs
-----
for ext .txt the bit length is 196
for ext .tex the bit length is 197
-----
syllabus.txt not_syllabus.txt 0.832787675578
not_syllabus.txt syllabus_same.txt 0.832787675578
websci_assignemnt9.tex not_websci_assignemnt9.tex 0.811393038308
websci_assignemnt9_same.tex not_websci_assignemnt9.tex 0.811393038308
syllabus.txt syllabus_same.txt 1.0
websci_assignemnt9.tex websci_assignemnt9_same.tex 1.0

```

Figure 12: Simhash similarity using min md5 length

similarities for the non-similar files are more distance at 0.8327 and 0.8113 respectively. This confirms the assumption that the significant digits of the hash do not play as great of a role for large hash lengths.

The final test uses the random value generated as describe previously with lower bounds of 10 and upper bound of 20, the results are seen in Figure 13. Once again the significant bits come into play with the similarity between non-similar files. The syllabus files were the only group to have the distinction made for different files with score of 0.875 whereas the websci files no difference could be determined.

```
Generating random bit length using floor(random.uniform(low,high))
Comparing files contained in the directory /home/john/Documents/cs834-f16/assignments/a1/code/similarDocs
-----
using bit len of 18
syllabus.txt syllabus_same.txt 1.0
syllabus.txt not_syllabus.txt 0.875
not_websci_assignemnt9.tex websci_assignemnt9.tex 1.0
not_websci_assignemnt9.tex websci_assignemnt9_same.tex 1.0
syllabus_same.txt not_syllabus.txt 0.875
websci_assignemnt9_same.tex websci_assignemnt9.tex 1.0
Process finished with exit code 0
```

Figure 13: Simhash similarity using random bits

```
import os
import hashlib
import re
from math import floor
from fs.osfs import OSFS
from collections import Counter, defaultdict
from nltk import word_tokenize
from argparse import ArgumentParser
from string import punctuation
from random import uniform
from nltk.corpus import stopwords as noInclude
from scipy.spatial.distance import cosine
from itertools import combinations
from shared_code import *

noPunc = re.compile('[%s]' % re.escape(punctuation))
stopWords = set(noInclude.words('english'))
translate_table = dict((ord(char), ' ') for char in punctuation)

def doc_features_hash(text, bits):
    # sanitize the document by removing all punctuation
    clean = noPunc.sub(' ', text)
    # generate the token weights (frequency)
    c = Counter()
    for token in word_tokenize(clean):
        # exclude stopwords and single letter tokens
        if token not in stopWords and len(token) >= 2:
            c[token.lower()] += 1
    # generate the hashes by md5
    hashes = {}
    md5 = hashlib.md5()
    for token in c.keys():
        md5.update(token.encode('utf-8'))
        # http://stackoverflow.com/questions/26685067/represent-a-hash-string-to-binary-in-python
        the_hash = ''.join(format(ord(i), 'b') for i in md5.hexdigest())
        hashes[token] = the_hash[:bits]
    # return the token weights and hashes
    return c, hashes

def doc_features_hash_rand(text):
```

```

"""
This version differs slightly
Here we take the minimum hash value of all tokens
as the length of the hash for all tokens
The token length initially for all tokens
is the length of the md5 hash transformed to a
binary string
"""
clean = noPunc.sub(' ', text)
c = Counter()
for token in word_tokenize(clean):
    if token not in stopWords and len(token) >= 2:
        c[token.lower()] += 1
hashes = {}
hash_lens = []
md5 = hashlib.md5()
for token in c.keys():
    md5.update(token.encode('utf-8'))
    the_hash = ''.join(format(ord(i), 'b') for i in md5.hexdigest())
    hashes[token] = the_hash
    hash_lens.append(len(the_hash))
hash_len = min(hash_lens)
return c, {token: hashes[token][:hash_len] for token in hashes.keys()}, hash_len

def gen_ith_of_v(i, hashes, weights):
    # generate the ith entry of vector V
    # accumulate values for the ith position in V
    toSum = []
    for token, the_hash in hashes.items():
        val = the_hash[i]
        weight = weights[token]
        if val == '1':
            toSum.append(1 * weight)
        else:
            toSum.append(-1 * weight)
    # sum the accumulated values resulting in the ith value of V
    return sum(toSum)

def file_fingerprint(file, rand_bits, bits):
    with open(file, 'r') as content_file:
        content = content_file.read()
    if rand_bits:
        feature_weights, feature_hashes, hash_len = doc_features_hash_rand(content)
        bits = hash_len
    else:
        feature_weights, feature_hashes = doc_features_hash(content, bits)
    fingerprint = []
    # generate the fingerprint
    for i in range(0, bits):
        # get the ith value of v
        ith_val = gen_ith_of_v(i, feature_hashes, feature_weights)
        # determine the ith value of the fingerprint
        if ith_val > 0:
            fingerprint.append(1)
        else:
            fingerprint.append(0)
    return fingerprint

def cosine_sim(hash1, hash2):
    # piggy back on scipy cosine distance
    # cosine similarity is 1 - cosine_distance
    return 1 - cosine(hash1, hash2)

```

```

def dir_sim_rand_helper(file_fps, combos):
    """
        As the simhashes between files of different types could
        potentially contain different lengths. This helper
        finds the minimum hash length for a common file extension.
        Then uses this length when checking for similarity
        between documents of same extension
    """
    common_ext = defaultdict(list)
    for f in file_fps.keys():
        ext = os.path.splitext(f)[1]
        common_ext[ext].append(len(file_fps[f]))
    real_len = {ext: min(common_ext[ext]) for ext in common_ext.keys()}
    for ext, bit_len in real_len.items():
        print('for ext %s the bit length is %d' % (ext, bit_len))
    print('-----')
    for k, v in combos:
        kext = os.path.splitext(k)[1]
        vext = os.path.splitext(v)[1]
        if kext == vext:
            rlen = real_len[kext]
            print(k, v, cosine_sim(file_fps[k][:rlen], file_fps[v][:rlen]))

def dir_sim(dirp, randombits, bits):
    """
        Finds the similarity between documents of the same extension
        contained in the given directory path.
        If we are using random bits i.e md5 length the helper is used.
        For each file contained in the directory:
        get its fingerprint and generate the comparison combinations
        then for each combination who's extension matches
        print the similarity between them
    """
    if bits is not None:
        print('using bit len of %d' % bits)
    dir = OSFS(dirp)
    file_fps = {}
    for fname in dir.listdir(files_only=True):
        file_fps[fname] = file_fingerprint(os.path.join(dirp, fname), randombits, bits)
    dir.close()
    combos = list(map(dict, combinations(file_fps.items(), 2)))
    if bits is None:
        dir_sim_rand_helper(file_fps, combos)
        return
    for k, v in combos:
        kext = os.path.splitext(k)[1]
        vext = os.path.splitext(v)[1]
        if kext == vext:
            print(k, v, cosine_sim(file_fps[k][:bits], file_fps[v][:bits]))

if __name__ == '__main__':
    parser = ArgumentParser(description="Generate Simhash of a document", prog='docSimhash.py',
                             usage='% (prog)s [options]')

    mode = parser.add_mutually_exclusive_group(required=True)
    mode.add_argument('-f', '--file', help='file to use', action=FullPaths, type=is_file)
    mode.add_argument('-d', '--dir', help='directory to look for duplicates', action=FullPaths,
                      type=is_dir)

    file_comp = mode.add_mutually_exclusive_group()
    file_comp.add_argument('-fc', '--filecomp', help='compare two files', action="store_true")
    file_comp.add_argument('-f1', '--file1', help='file one to use', action=FullPaths, type=is_file)
    file_comp.add_argument('-f2', '--file2', help='file two to use', action=FullPaths, type=is_file)

```

```

bit_mode = parser.add_mutually_exclusive_group(required=True)
bit_mode.add_argument('-b', '--bits', help='number of bits to use', type=int)
bit_mode.add_argument('-md5b', '--md5bits', help='number of bits to use determined by md5 hash
↳ length',
                        action="store_true")
bit_mode.add_argument('-randb', '--randbits', help='generate random bit length using value
↳ lowend highend', type=int, nargs='+')
args = parser.parse_args()
print()
randy = False
if args.randbits is not None:
    print("Generating random bit length using floor(random.uniform(low,high))")
    args.bits = floor(uniform(args.randbits[0], args.randbits[1]))

if args.md5bits is True:
    print("Using bit length determined by the length of the md5 hash")
    randy = True

if args.dir is not None:
    print('Comparing files contained in the directory %s' % args.dir)
    print('-----')
    dir_sim(args.dir, randy, args.bits)
else:
    if args.filecomp:
        file1_fp = file_fingerprint(args.file1, randy, args.bits)
        file2_fp = file_fingerprint(args.file2, randy, args.bits)
        print("The simhash similarity between %s and %s is %d" % (
            args.file1, args.file2, cosine_sim(file1_fp, file2_fp)))
    else:
        result = file_fingerprint(args.file, randy, args.bits)
        print("the simhash fingerprint for %s is %s" % (args.file, ''.join(str(fi) for fi in
            result)))

```

Source Code 3: Simhash Generator and Document Similarity

References

- [1] Arrow. Arrow - Better dates & times for Python. <https://github.com/crsmithdev/arrow>.
- [2] B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Pearson, 1 edition, 2 2009. ISBN 9780136072249.
- [3] NLTK. NLTK – the Natural Language Toolkit – is a suite of open source Python modules, data sets and tutorials supporting research and development in Natural Language Processin. <https://github.com/nltk/nltk>.
- [4] PyFilesystem. PyFilesystem provides a simplified common interface to a variety of different filesystems, such as the local filesystem, zip files, ftp servers etc. <https://github.com/PyFilesystem/pyfilesystem>.
- [5] yattag. Yattag is a Python library for generating HTML or XML in a pythonic way. <https://github.com/sirex/yattag>.