Introduction to Web Science

Assignment 5

Prof. Dr. Steffen Staab
staab@uni-koblenz.de

René Pickhardt
rpickhardt@uni-koblenz.de

Korok Sengupta
koroksengupta@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Koblenz-Landau

Submission until: November 30, 2016, 10:00 a.m.
Tutorial on: December 2, 2016, 12:00 p.m.

Please look at the lessons 1) **Dynamic Web Content** & 2) **How big is the Web?**

For all the assignment questions that require you to write code, make sure to include the code in the answer sheet, along with a separate python file. Where screen shots are required, please add them in the answers directly and not as separate files.

Team Name: QUEBEC:

1. Daniel Kostic
2. Stefan Vujovic
3. Igor Fedotov

1 Creative use of the Hyptertext Transfer Protocol (10 Points)

HTTP is a request response protocol. In that spirit a client opens a TCP socket to the server, makes a request, and the server replies with a response. The server will just listen on its open socket but cannot initiate a conversation with the client on its own.

However you might have seen some interactive websites which notify you as soon as something happens on the server. An example would be Twitter. Without the need for you to refresh the page (and thus triggering a new HTTP request) they let you know that there are new tweets available for you. In this exercise we want you to make sense of that behaviour and try to reproduce it by creative use of the HTTP protocol.

Have a look at `server.py`[1] and `webclient.html` (which we provide). Extend both files in a way that after `webclient.html` is servered to the user the person controlling the server has the chance to make some input at its commandline. This input should then be send to the client and displayed automatically in the browser without requiring a reload. For that the user should not have to interact with the webpage any further.

1.1 webclient.html

```
1: <html>
2: <head>
3:         <title>Abusing the HTTP protocol - Example</title>
4: </head>
5: <body>
6:         <h1>Display data from the Server</h1>
7:         The following line changes on the servers command line
8:         input: <br>
9:         <span id="response" style="color:red">
10:                 This will be replaced by messages from the server
11:        </span>
12: </body>
13: </html>
```

1.2 Hints:

- This exercise is more like a riddle. Try to focuse on how TCP sockets and HTTP work and how you could make use of that to achieve the expected behaviour. Once you have an idea the programming should be straight-forward.
- The Javascript code that you need for this exercise was almost completely shown in one of the videos and is available on Wikiversity.

[1] you could store the code from http://blog.wachowicz.eu/?p=256 in a file called `server.py`

- In that sense we only ask for a ”proof of concept” nothing that would be stable out in the wilde.
  - In particular, don’t worry about making the server uses multithreading. It is ok to be blocking for the sake of this exercise.
- Without use of any additional libraries or AJAX framework we have been able to solve this with 19 lines of Javascript and 11 lines of Python code (we provide this information just as a way for you to estimate the complexity of the problem, don’t worry about how many lines your solution uses).

Answer
To implement this, the client side javascript is sending a request for a special path, where no file can be found. The server is modified to check for this path in an exception handler, and send user input if the path is ”webclient”. Client side function invokes itself over and over again after each response is received.

```
function loadResponse() {
      var xhttp = new XMLHttpRequest();
      xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
         document.getElementById("response").innerHTML = this.responseText;
         loadResponse();
        }
      };
      xhttp.open("GET", "webclient", true);
      xhttp.send();
    }
    loadResponse();
```

Server side modification:

```
except Exception as e:  # in case file was not found, generate 404 page
        # If this "file" is webclient, send back user input
    if file_requested == "www/webclient":
        response_headers = self._gen_headers(200)
        response_content = input("Send message to client\n")
        response_content = response_content.encode()
    else: # requested file is not found
        print("Warning, file not found. Serving response code 404\n", e)
        response_headers = self._gen_headers(404)

        if (request_method == 'GET'):
            response_content = b"<html><body><p>Error 404: File not found</p><p>Python HTTP server</p></body></html>"
```

2 Web Crawler (10 Points)

Your task in this exercise is to "crawl" the `Simple English Wikipedia`. In order to execute this task, we provide you with a mirror of the Simple English Wikipedia at `141.26.208.82`.

You can start crawling from `http://141.26.208.82/articles/g/e/r/Germany.html` and you can use the `urllib` or `doGetRequest` function from the last week's assignment.

Given below is the strategy that you might adopt to complete this assignment:

1. Download `http://141.26.208.82/articles/g/e/r/Germany.html` and store the page on your file system.
2. Open the file in python and extract the local links. (Links within the same domain.)
3. Store the file to your file system.
4. Follow all the links and repeat steps 1 to 3.
5. Repeat step 4 until you have downloaded and saved all pages.

2.1 Hints:

- Before you start this exercise, please have a look at Exercise 3.
- Make really sure your crawler doesn't follow external urls to domains other than `http://141.26.208.82`. In that case you would start crawling the entire web
- Expect the crawler to run about 60 Minutes if you start it from the university network. From home your runtime will most certainly be even longer.
- It might be useful for you to have some output on the crawlers commandline depicting which URL is currently being fetched and how many URLs have been fetched so far and how many are currently on the queue.
- You can (but don't have to) make use of breadth-first search.
- It probably makes sense to take over the full paths from the pages of the Simple English Wikipedia and use the same folder structure when you save the html documents.
- You can (but you don't have to) speed up the crawler significantly if you use multithreading. However you should not use more than 10 threads in order for our mirror of Simple English Wikipedia to stay alive.

Answer:
Crawler implemented using breadth first search. All visited links are saved in a set, and a python deque is used for a queue. For every valid link a node is constructed, and the node knows its parent, all children and URL. Different ways and methods for scraping

have been tested, initially wget and beautifulsoup, but were abandoned and used urllib and simple string parsing instead (proved to be faster).
Node class:

```
class Node(object):
    total_web_pages = 1
    total_num_links = 0
    links_in_page = []
    int_link = []
    ext_link = []

    def __init__(self, _parent, _node_url):
        self.parent = _parent
        self.node_url = _node_url
        self.node_children = []
        if _parent is None:
            self.level = 0
        else:
            self.level = _parent.level+1
        self.link_num = 0
```

Helper class, saving results and making graphs

```
import json
import numpy as np
from Node import Node


def write():
        with open("data.txt", 'x') as f:
                data = {'total_web_pages': Node.total_web_pages,
                'total_num_links': Node.total_num_links,
                'links_in_page': Node.links_in_page,
                'int_link': Node.int_link,
                'ext_link': Node.ext_link}
                obj = json.dumps(data)
                f.write(obj)


def load():
        with open ("data1.txt", 'r') as f:
                data = f.read()
                obj = json.loads(data)
                Node.total_web_pages = obj['total_web_pages']
                Node.total_num_links = obj['total_num_links']
                Node.links_in_page = obj['links_in_page']
                Node.int_link = obj['int_link']
                Node.ext_link = obj['ext_link']
                print("Total web pages: {}".format(Node.total_web_pages))
                print("Total link number: {}".format(Node.total_num_links))
                print("Average links per page: {}".format(np.mean(Node.links_in_page)))
                print("Median links per page: {}".format(np.median(Node.links_in_page)))

if __name__ == "__main__":
        import crawl
        load()
        crawl.make_hist(Node.links_in_page)
        crawl.make_scat(Node.int_link, Node.ext_link)
```

Crawler code:

```
import os
import time
import traceback
from collections import deque
from urllib.error import HTTPError
from urllib.parse import urlparse, urljoin
from urllib.request import urlopen
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gaussian_kde
import wget
from bs4 import BeautifulSoup
import helper
from Node import Node

visited = set([])


def download_urllib(url):
    parsed_url = urlparse(url)
    if parsed_url.path.find('.') == -1:
        name = 'index.html'
        file_path = parsed_url.path
        print("Filename not found!!!")
    else:
        name = parsed_url.path.split('/')[-1]
        file_path = parsed_url.path.split('/')[:-1]
        file_path = "/".join(file_path)
    try:
        response = urlopen(url)
        data = response.read()
        text = data.decode('utf-8')
        if not os.path.exists("junk/" + file_path):
            os.makedirs("junk/" + file_path)
        f = open("junk{}/{}".format(file_path, name), mode="x", encoding="utf-8")
        f.write(text)
        f.close()
        return text
    except UnicodeDecodeError:
        # traceback.print_exc()
        print("Failed decoding file: " + name)
        return False
    except HTTPError:
        # traceback.print_exc()
        return False
    except Exception:
        # traceback.print_exc()
        return False


def get_links_from_text(file):
    urls = []
    internal = 0
    external = 0
    file = file.split('\n')
    for line in file:
        while line.find('<a') != -1:
            a_start = line.find('<a')
            whole_tag = line[a_start + 2:]
```

```
60              a_tag = whole_tag[:whole_tag.find('>')]
61              if a_tag.find("href") == -1:
62                  line = line[a_start + 2:]
63                  continue
64              href = a_tag[a_tag.find('href="') + 6:]
65              path = href[:href.find('"')]
66              if path.find("http") == 0:
67                  external += 1
68              else:
69                  internal += 1
70              urls.append(path)
71              line = line[a_start + 4:]
72      Node.int_link.append(internal)
73      Node.ext_link.append(external)
74      Node.total_num_links += len(urls)
75      Node.links_in_page.append(len(urls))
76      return urls
77
78
79  def bs_soup(text):
80      urls = []
81      internal = 0
82      external = 0
83      soup = BeautifulSoup(text, 'lxml')
84      for a_tag in soup.find_all('a'):
85          path = a_tag.get('href')
86          if path is not None:
87              if path.find("http") == 0:
88                  external += 1
89              else:
90                  internal += 1
91              urls.append(path)
92      Node.int_link.append(internal)
93      Node.ext_link.append(external)
94      Node.total_num_links += len(urls)
95      Node.links_in_page.append(len(urls))
96      return urls
97
98
99  def download(url):
100     parsed_url = urlparse(url)
101     # parsed_url = url
102     if parsed_url.path.find('.') == -1:
103         name = 'test.html'
104     else:
105         name = parsed_url.path.split('/')[-1]
106     try:
107         # final_link = parsed_url.scheme+ "://" + parsed_url.netloc + parsed_url.path
108         filename = wget.download(url, "junk/"+name)
109         # print(filename)
110         return filename
111     except Exception:
112         # print("*****LINK FAILED*****\n"+url)
113         # traceback.print_exc()
114         # raise Exception("Link failed :" + url)
115         return False
116
117
118 def get_links(filename):
119     urls = []
120     internal = 0
121     external = 0
```

```
122     try:
123         with open(filename, mode='r', encoding="utf-8") as file:
124             # with open(filename+'.txt', mode='w') as helper:
125                 for line in file:
126                     while line.find('<a') != -1:
127                         a_start = line.find('<a')
128                         whole_tag = line[a_start + 2:]
129                         a_tag = whole_tag[:whole_tag.find('>')]
130                         if a_tag.find("href")==-1:
131                             line = line[a_start+2:]
132                             continue
133                         href = a_tag[a_tag.find('href="') + 6:]
134                         path = href[:href.find('"')]
135                         if path.find("http") == 0:
136                             external += 1
137                         else:
138                             internal += 1
139                         urls.append(path)
140                         line = line[a_start+4:]
141                         # helper.write(path + '\n')
142         Node.int_link.append(internal)
143         Node.ext_link.append(external)
144         Node.total_num_links += len(urls)
145         Node.links_in_page.append(len(urls))
146         return urls
147     except UnicodeDecodeError:
148         traceback.print_exc()
149         print("Failed decoding file: " + filename)
150         return urls
151 
152 
153 def get_children_from_urls(urls, parent):
154     nodes = []
155     for ur in urls:
156         if ur in visited:
157             continue
158         node = Node(parent, ur)
159         nodes.append(node)
160     return nodes
161 
162 
163 def remove_file_name(full_url):
164     if full_url.find('.') == -1: return full_url
165     return '/'.join(full_url.split('/')[:-1])
166 
167 
168 def make_hist(x):
169     x = [s for s in x if s <= 150]
170     n, bins, patches = plt.hist(x, bins=30, normed=0, facecolor='blue', alpha=0.75)
171     plt.xlabel('Number of links')
172     plt.ylabel('Number of pages')
173     plt.title('Distribution of links')
174     plt.axis([0, 150, 0, 8000])
175     plt.grid(True)
176 
177     plt.show()
178 
179 
180 def make_heatmap(x,y):
181     xy = list(zip(x,y))
182     xy = filter(lambda p: p[0]<400 and p[1]<200, xy)
183     xy = list(map(list, zip(*xy)))
```

```
    x = xy[0]
    y = xy[1]
    xy = np.vstack([x, y])
    z = gaussian_kde(xy)(xy)

    fig, ax = plt.subplots()
    plt.axis([0, 400, 0, 200])
    ax.scatter(x, y, c=z, s=100, edgecolor='')
    plt.show()

def make_scat(x,y):
    # x = [s for s in x if s < 400]
    # y = [s for s in y if s < 200]
    plt.axis([0, 400, 0, 200])
    plt.scatter(x,y)
    plt.show()


if __name__ == "__main__":
    start_time = time.time()
    start_url = "http://141.26.208.82/articles/g/e/r/Germany.html"
    start_url = "http://localhost/simple/articles/g/e/r/Germany.html"
    # start_url = "http://141.26.208.82/index.html"
    file_name = download(start_url)
    init_node = Node(None, start_url)
    init_node.node_children = get_children_from_urls(get_links(file_name), init_node)

    # init_node.node_children = list(set(init_node.node_children))

    visited.add(start_url)
    queue = deque([])
    queue.extend(init_node.node_children)

    try:
        while queue:
            # print("Size of queue {}".format(len(queue)))
            if Node.total_web_pages % 1000 == 0:
                print("Total web pages crawled: {}".format(Node.total_web_pages))
            if Node.total_web_pages == 150000:
                break
            current_node = queue.popleft()
            # print("Level of node is: {}".format(current_node.level))
            if current_node.node_url.find('http') == 0 or current_node.node_url.find('#') == 0:
                continue
            # link = urljoin(start_url, link)
            current_node.node_url = urljoin(current_node.parent.node_url, current_node.node_url)
            link = current_node.node_url
            if link in visited:
                continue
            else:
                # filename = download(link)
                filename = download_urllib(link)
                if not filename:
                    continue
                else:
                    Node.total_web_pages += 1

                current_node.node_children = get_children_from_urls(get_links_from_text(filename), current_node)
                queue.extend(current_node.node_children)
                visited.add(link)
    except KeyboardInterrupt:
        pass
```

```
246     helper.write()
247     end_time = time.time()
248     print("\nResults:\n")
249     print("Time taken to finish: {}".format(end_time-start_time))
250     print("Total web pages: {}".format(Node.total_web_pages))
251     print("Total link number: {}".format(Node.total_num_links))
252     print("Average links per page: {}".format(np.mean(Node.links_in_page)))
253     print("Median links per page: {}".format(np.median(Node.links_in_page)))
254     # print("Internal and external links found: {}, {}".format(Node.int_link, Node.ext_link))
255     make_hist(Node.links_in_page)
256     make_scat(Node.int_link, Node.ext_link)
```

3 Web Crawl Statistics (10 Points)

If you have successfully completed the first exercise of this assignment, then please provide the following details. You may have to tweak your code in the above exercise for some of the results.

3.1 Phase I

1. Total Number of *webpages* you found.
2. Total number of links that you encountered in the complete process of crawling.
3. Average and median number of links per web page.
4. Create a *histogram* showing the distribution of links on the crawled web pages. You can use a bin size of 5 and scale the axis from 0-150.

3.2 Phase II

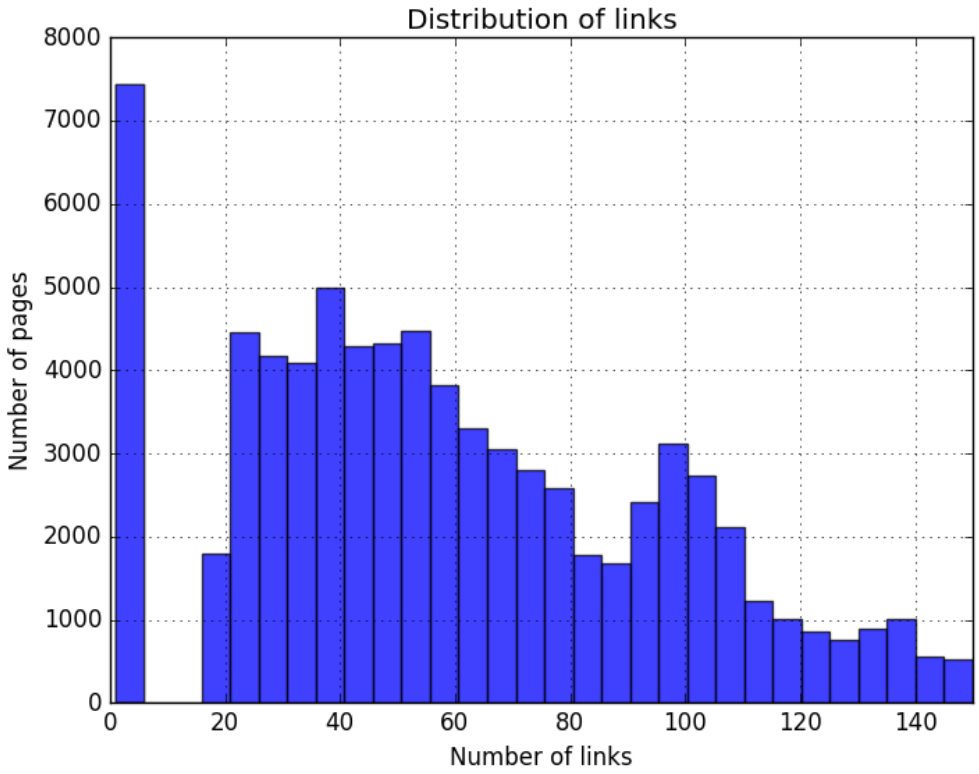
1. For every page that you have downloaded, count the number of internal links and external links.
2. Provide a *scatter plot* with number of internal links on the X axis and number of external links on the Y axis.

Answer:
Phase 1:

1. Pages found: 80918
2. Links found: 5752080
3. Average: 71.085, Median: 56.0

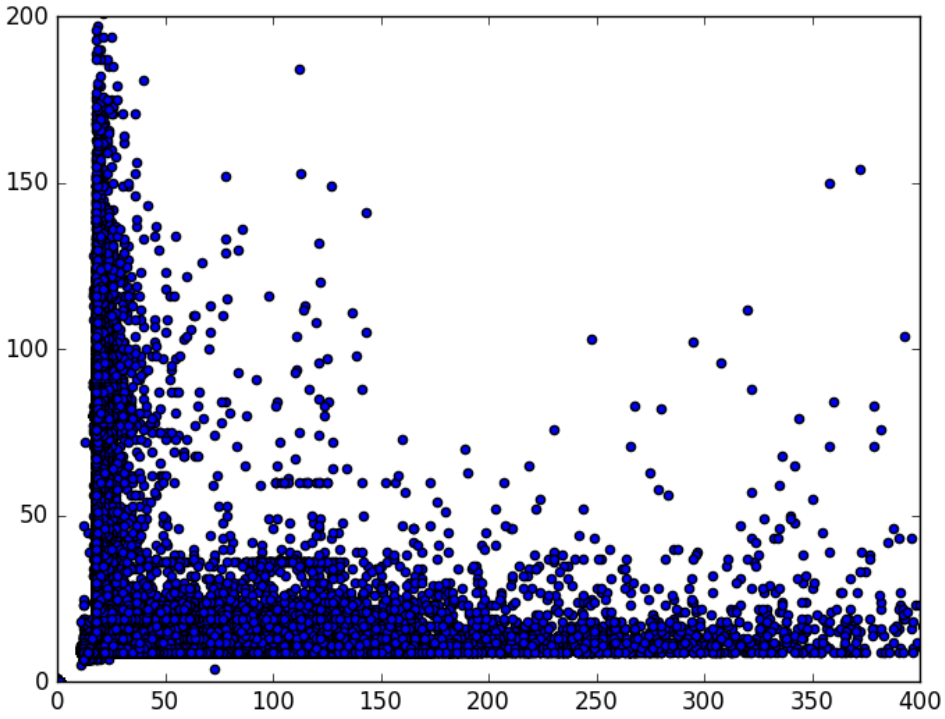
```
Total web pages: 80918
Total link number: 5752080
Average links per page: 71.08529622580885
Median links per page: 56.0
```

4. Histogram

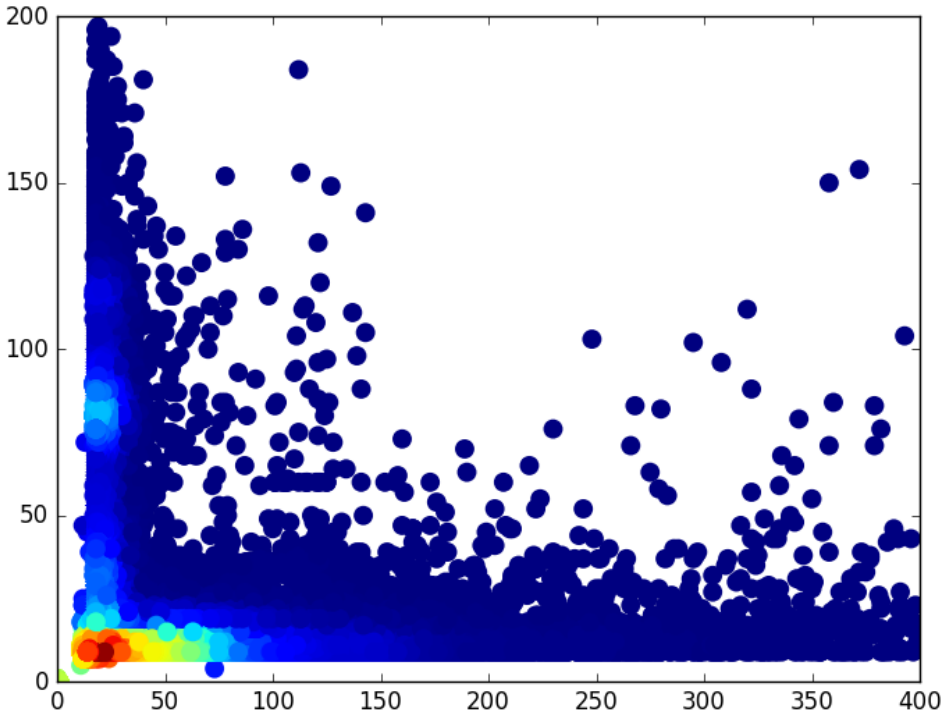


Phase 2:

1. Counting of links done while parsing the file
2. Scatter plot



3. Heat map, addressing the problem of many dots in one place:


200
150
100
50
0
0
50
100
150
200
250
300
350
400

Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment5/` in your group's repository.
- The name of the group and the names of all participating students must be listed on each submission.
- Solution format: all solutions as *one* PDF document. Programming code has to be submitted as Python code to the github repository. Upload *all* `.py` files of your program! Use `UTF-8` as the file encoding. *Other encodings will not be taken into account!*
- Check that your code compiles without errors.
- Make sure your code is formatted to be easy to read.
  - Make sure you code has consistent indentation.
  - Make sure you comment and document your code adequately in English.
  - Choose consistent and intuitive names for your identifiers.
- Do *not* use any accents, spaces or special characters in your filenames.

Acknowledgment

This latex template was created by Lukas Schmelzeisen for the tutorials of "Web Information Retrieval".

LaTeX

Currently the code can only be build using LuaLaTeX, so make sure you have that installed. If on Overleaf, there's an error, go to settings and change the LaTeXengine to `LuaLaTeX`.