Crawler for CSExpert Search Engine: "OpenCrawler"

Ashish Aggarwal (aaggrwl7), Jenny (Man) Tran(mantran2),
Tanmay Shah (tshah29)
University of Illinois at Urbana-Champaign
CS 410 - Spring 2017

Abstract

Search engines have proven to be one of the key growth areas for our wireless and immensely connected world. So far, the search engine that most people are familiar with, Google, does an excellent job at retrieving relevant documents bound by a search query. However, Google's search realm is vast and sometimes misses out on the smaller documents that are not searched as often. Therefore, our vision is to create CSExpert- a specialized expert search engine for finding the homepages of computer science experts. We hope this tool will benefit individuals on both spectrum: those who are new and know little about research as well as those who are looking to find reliable, accurate, and detailed information about a particular subject matter.

Introduction

Our project is very research-based. The inspiration of our project originates from the need of a CS expert crawler. Traditional crawler does a generic crawl of the web given a specific link to crawl from. However, ours performs a similar role but more refined in that we sort through the retrieved links to filter our irrelevant links. In return, our crawler retrieves more relevant feedback, specifically towards research on a campus.

We genuinely hope to make research more conveniently accessible for any individuals. Entering college can be a daunting phase of any individual's life. On top of figuring out *what* he could be doing for the rest of his life, he needs to figure out *how* to move towards that specific fields. Therefore, many people are interested in research because it is one of the ways to figure out if the passion exists for the field or not. Many could be intimidated by the breadth of research a campus could provide. Our search engine, although might not be comprehensive like a horizontal-search engine, serves as an entry point for this purpose.

Related Work

a)description of any existing similar systems or similar work as described in research papers:

Of course, there are existing resources for gathering the data our crawler needs: the vast number of faculty homepages from Computer Science departments and research labs at universities across the nation.

As far as existing systems that perform similar tasks to the CS Expert tool, there is a search feature on the main Illinois homepage. The way this system works is by returning results from university-wide sources that match the textual keywords in the query. This general search tool is often not helpful for learning about research or a particular subject matter, and this is because it does not return in-depth results about experts or research areas.

Also, after conducting thorough research, it was found that some CS departments-- but not all-- have their own search pages. Hence, Open Crawler fills this gap.

Next, Google Scholar is an alternative which easily comes to mind. After researching this tool, however, it is clearly evident that it does not focus specifically on computer science research, and it doesn't guarantee information to come from university-based faculty or research labs.

Next, after reading some research papers, it was found that the most common workaround for finding 'expert information' is to use a hyperspecific method of altering the Google search query. Namely, this includes imperfect approaches such as "use lower case letters" or "leave out punctuation" or "enter search terms in singular forms".

Using a step above this method, researchers at University of Toronto and Google have prototyped "Hilltop: A Search Engine based on Expert Documents". The key difference between Open Crawler and Hilltop is that Open Crawler focuses on homepage identification, whereas Hilltop simply returns a "ranked list of documents", which could be homepages or children links buried deep.

Overall, Open Crawler is different from existing expert finders because it specifically crawls faculty homepages (and generates a subset of faculty homepages). We have chosen to crawl the homepages because homepages of faculty ensure accuracy of expert information.

b) a brief discussion of how your project is related to previous work

Out of everything described above, Hilltop and Google Scholar is the existing work which is most similar to Open Crawler. The main way in which Open Crawler is similar to a) Google Scholar b) Hilltop is in its overall information retrieval technique. All of these systems crawl a large set of websites in order to find the appropriate information which the user has queried. What is different is the way in which the subset of appropriate information is outputted.

c) highlight the novelty of your project as compared with existing work

There are several novel aspects of our Crawler when compared with existing work.

First, we wrote our crawler by hand and in Python. This is different from existing tools like wget, cURL, etc. because these tools are written in C. Plus, these tools simply download files from the web, which is a different function than our Open Crawler.

After doing research, we found that Open Crawler is unique, as there aren't well known crawling softwares in the market that can generate a filtered list of URLs and perform link prediction.

Furthermore, our project is novel & useful to its target audience. None of the other tools described above specifically cater to the needs of the following users, and this is why Open Crawler shines:

- UIUC Undergraduate students
- UIUC Graduate students
- UIUC Faculty/professors/researchers
- Prospective undergraduate students who would like to get involved in research or find out what is happening in a particular discipline
- Prospective graduate students who would like to better understand the specialities and expertise the campus community offers
- Outside people who may be interested in finding experts for a conference, partnership, or other venture

Problem Definition:

The challenge that Open Crawler aims to address is that broad search engines (i.e. Google, Bing, etc.) provide a lot of information but often in a scattered format. These methods are often called "horizontal search". This format is not ideal for a user who is looking to search for information in a specific domain. Rather, the user needs a way to find this domain-specific information quickly and reliably. Domain specific (i.e. "vertical") search engines have not become popular yet, specifically for Computer Science research topics.

According to a variety of sources, vertical search engines have many advantages:

- "Greater precision due to limited scope"
- Able to "leverage domain knowledge"
- They provide "support of specific unique user tasks"

In addition to vertical search, a research question Open Crawler aims to address is that of link prediction. Although we have not implemented actual link prediction algorithms, Open Crawler is poised to be able to handle the algorithms. One of the output files (described later on) creates a graph showing dependencies between parent and child links. In the future, we can use a hierarchical probabilistic model and "given links in a network" to "predict the links that will be added to the network during the later time interval". This is a powerful functionality that can allow users of Open Crawler to easily find information.

This is Open Crawler's formally-defined computation problem: the input is the vast collection of computer science webpages from universities across the nation. Its output is a crawled file which contains context-appropriate & useful URLs of faculty homepages. The Open Crawler system passes this output to the Information Extraction Team which extracts the textual content from the links.

The specifications of implementation will be discussed later on.

Open-Source Collaboration:

A major distinguishing factor for CSExpert is that it is not a stand alone project. Instead, it was a joint effort of 5 teams of students. Our team is the Open Crawler team.

- Faculty Homepage Crawler Team: Tanmay Shah, Jenny Tran, Ashish Aggarwal
- Recommendation/Crawler Team: Zijian Yao, Jiayou Sun, Ziyao Li
- Faculty Information Extractor Team: Rohit Ramkumar, Deepak Mani
- Search Engine Team: Hang Yuan, Eddie Wu, Xiao Tang
- Profiling/Summarization Team: Yuqi Wang

Hence, not only is each part of the pipeline constructed in an innovative manner, but the overall result is a significantly new system. The end goal of CSExpert is to create an engine that can be expanded to serve regions all across the world. After the semester, each of the teams may integrate their product into a singular pipeline

Throughout the project, an important and noteworthy skill which the teams in CSExpert had a chance to exercise was the art of collaboration. Two main tools were used to distribute tasks, share files, and discuss progress- Wiki space and Slack.

Along with collaborating as a large team, our Open Crawler team in particular made it a point to collaborate frequently with the Professor to gain his insights and shape our project better. We visited office hours every week for a few weeks, and we also attended the virtual meetings. We took his thoughts and did further research, read more research papers, and really understood how this project could be taken to the next step.

Methods:

A Web Crawler also known as a Spider, is a bot that browses the web to obtain a some sort of information that can be passed to the extractor to get useful information. The information could be in the form of URLs or some parsed information online. Our crawler uses the first approach. It will take the URL of a University Web Page and generate a list of all the useful URLs present on that page and the pages within those. In our opinion this is the best approach since it provides us homepages which should have useful information and the extractor can take whatever information that is needed from those pages, rather than working with whatever parsed information was provided to it. The purpose of a Search Engine is to be able to provide useful

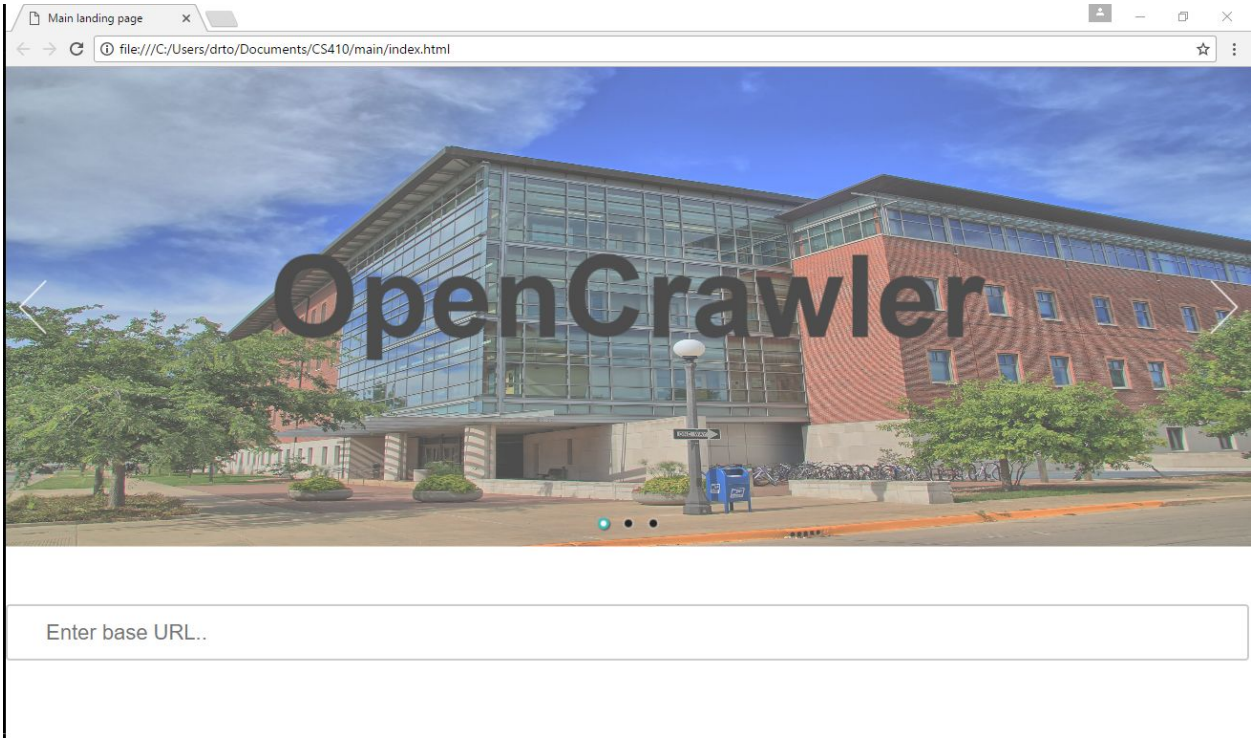
information and it should be programmed in a way such that it could be easily modified to obtain the required information.

Now suppose we had chosen the technique where we crawled the web and got specific data and not the links. This data could be anything, like the name of the professor and the field they're doing research in. Now what if we needed the URL to their website? Then we'd have to change the crawler to get that link and update the extractor accordingly. Or if we had the URL and the name of the professor we would still go into the URL to get more information. That is why we chose the path which would be useful in any circumstance. We would pass on a list of important URLs to the extractor which could then take the useful information. If we needed to change the information we wanted, we would just have to make a small change in the extractor and it would give us just the information that we need. This doesn't mean that this technique doesn't have it's drawbacks. For a website with several links it might take more time, however the performance and the utility of the result is much better in this case than in any other.

While coming up with the solution we analyzed the user class and the main purpose of the project. Keeping that in mind we knew that the end result held greater importance over speed. Therefore, going with this technique sounded the best. As with any code there were some issues that we faced, however the most important one was that while cycling through a web page our code used to enlist links like "/home/about-us/info/" and when we tried to go into this link to find more pages, the code would stop with an error. The solution we came up with was to check the link before going into it. This sounds like a simple error with a straightforward solution, however it was one of those minute details which a coder always overlooks. Apart from those we had some timeout issues which we were able to solve by changing the code a bit, but nothing major.

Evaluation/Sample Results:

Before diving into the specifics of the Open Crawler itself, here is a screenshot of what a potential user interface might look like. Although the crawler's functionality is not stand alone (since it is part of a much bigger CSExpert pipeline), it is still important to show the usefulness through a UI.



After completing the code we initially tested with some websites to see how the data was being traversed. From that we made one observation, that our code would crash due to time out on websites which had a large number of websites. However, we decided to test our code further and therefore created a list of faculty home pages that we would use to test our code. While this list was not exhaustive, we made sure we covered all sorts of websites that our product might encounter. Below is a part of the list that we used to test our code. This list was generated by looking at the website's of professors at UIUC, which gave us a decent sized test dataset. The websites were divided on the basis of the research area of the professors.

Architecutre-
http://iacoma.cs.uiuc.edu/josep/torrellas.html

AI-
http://jianpeng.web.engr.illinois.edu/

Bioinformatics-
http://www.sinhalab.net/

Databse-
http://sundaram.cs.illinois.edu/

Graphics-
http://msl.cs.uiuc.edu/~lavalle/

Programming methods-
http://mir.cs.illinois.edu/marinov/

Scietnfic computing-
https://andreask.cs.illinois.edu/aboutme/

Systems-
http://caesar.web.engr.illinois.edu/

Theory-
http://jeffe.cs.illinois.edu/

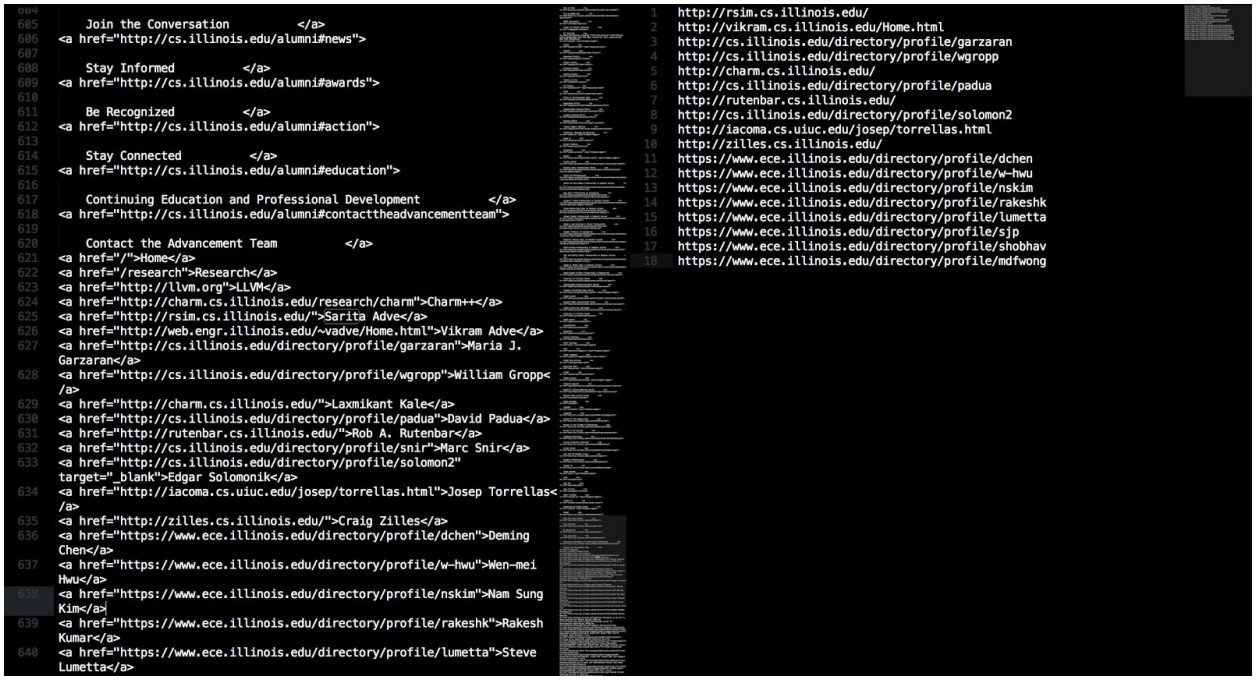
After testing our code extensively, we could conclude that our code worked perfectly and give us a well filtered list of URLs to faculty homepages, allowing the extractor to take whatever information necessary from those links and provide it to the end user. The tests were decided after talking to some of our peers in the Computer Science department. We chose them because they would be one of the end users for this product. Getting their input on the testing allowed us to check if our code would actually work and solve the problem that we intend it to. Generating a list randomly would not solve the purpose as we might end up testing a bunch of sites irrelevant to our project. However, we would like to mention that our code is still in the prototype stages and needs some work done to it before actually combining it with the other parts of the bigger project that it belongs to.

As mentioned before the code still timed out on bigger websites. That is one of the most important features that we feel we need to work on before taking this project forward. The screenshot below shows the error message received on doing this.

```
Traceback (most recent call last):
  File "CS410_Project.py", line 13, in <module>
    for ee in re.findall('''href=["'](.[^"']+)["']''', urllib.urlopen(i).read(), re.I):
  File "/usr/local/Cellar/python/2.7.10_2/Frameworks/Python.framework/Versions/2.7/lib/python2.7/urllib.py", line 87, in urlopen
    return opener.open(url)
  File "/usr/local/Cellar/python/2.7.10_2/Frameworks/Python.framework/Versions/2.7/lib/python2.7/urllib.py", line 213, in open
    return getattr(self, name)(url)
  File "/usr/local/Cellar/python/2.7.10_2/Frameworks/Python.framework/Versions/2.7/lib/python2.7/urllib.py", line 467, in open_file
    return self.open_ftp(url)
  File "/usr/local/Cellar/python/2.7.10_2/Frameworks/Python.framework/Versions/2.7/lib/python2.7/urllib.py", line 550, in open_ftp
    ftpwrapper(user, passwd, host, port, dirs)
  File "/usr/local/Cellar/python/2.7.10_2/Frameworks/Python.framework/Versions/2.7/lib/python2.7/urllib.py", line 875, in __init__
    self.init()
  File "/usr/local/Cellar/python/2.7.10_2/Frameworks/Python.framework/Versions/2.7/lib/python2.7/urllib.py", line 884, in init
    self.ftp.connect(self.host, self.port, self.timeout)
  File "/usr/local/Cellar/python/2.7.10_2/Frameworks/Python.framework/Versions/2.7/lib/python2.7/ftplib.py", line 135, in connect
    self.sock = socket.create_connection((self.host, self.port), self.timeout)
  File "/usr/local/Cellar/python/2.7.10_2/Frameworks/Python.framework/Versions/2.7/lib/python2.7/socket.py", line 575, in create_connection
    raise err
IOError: [Errno ftp error] [Errno 60] Operation timed out
Ashishs-MacBook-Pro:Desktop Ashish$
```

Even though we were assigned a small part of the bigger project, we would like to complete this in future and collaborate with members from other teams. The reason being that while talking to other students in the Computer Science department many of them showed interest in this project. Some of them were looking for tools that would allow them to just enter their field of interest and get a list of professors working in that field, rather than scrolling through each page on the UIUC CS webpage until they find the right match. Another student mentioned that once this project is taken to a larger scale it would allow people deciding to go for Masters, look for professors and Universities working on research projects that they would be interested in. An interesting review that we got was from a professor. According to them this tool would be beneficial for professors as well, since they are always looking to collaborate with others on larger scale projects. This clearly goes to prove that this project has a lot of scope and the system as a whole would turn out to be a useful one.

To summarize, our prototype right now works well with specific research area websites, such as the output we got below when looking at the UIUC CS research page for Architecture, Compilers & Parallel Computing. As we described earlier in the report, notice the structuring of the output to show the dependencies among parent & child links:



This screen grab cannot show everything, but we also have a timestamp at the very bottom of the output which shows the amount of time that crawling iteration took. As we describe in our Conclusions & Future Work section, we aim to analyze this data to optimize the crawler.

However, when crawled through the actual CS Page we encountered a timeout. So, in the long run this code is highly useful in the field of academia, however, there is still a lot of work that needs to be done. We hope to continue working on it after the semester ends and possibly submit a better version of it in future.

Challenges that you need to solve in developing the tool and why you have chosen to address the challenges in a particular way:

The first challenge was choosing which language we would use to implement our crawler. After researching scraping/downloading tools like wget and cURL, we found that they are all written in C. Hence, we conducted thorough research around frameworks like scrapy and we found that it is best to implement our crawler in Python.

We came to the choice of Python for several reasons:
Python serves as an entry point to coding for many individuals because of its easy-to-read syntax. In addition, Python is a high level language which means that this program isn't

particular to a certain CPU. Therefore, the code for this could be easily understood and maintain by anyone who has acquired interests in coding recently. Also, we chose Python because all of our team mates have had experiences with Python.

Another challenge/obstacle was deciding how to format the output of the crawled files. With the two outputted files, we decided to make the original one structured in such a way to show the dependencies (parent & child links). We decided to not structure the second file in this way because we realized that this file anyway contains the subset of relevant links--hence, there is no need to show the dependencies.

Another hurdle was deciding how we would initiate the crawling process. We came to the conclusion that for the most stable approach, it would be best to have the user input a base URL which is a known faculty homepage. This ensures that the crawler does not go haywire. Of course, as described earlier in the paper, we will reach a point (in our further improvement) where the user doesn't have to input a base URL.

Many of the challenges and obstacles for this project are yet to come, as described in the Conclusions & Future Work section.

Conclusions & Future Work:

There are several key points that we would like to work on expanding for the Open Crawler system.

First and foremost, we will expand the crawler so that it retrieves an even bigger number of [relevant] web pages. The system is capable of being expanded across the entire nation's campuses, not only limited to UIUC or midwest campuses.

Also, we will conduct run-time analysis on our crawler across multiple iterations in order to test its performance. This is in order to optimize the crawler. And we are in a good position for optimization because we don't actually download the files (unlike wget) hence our solution is efficient. There is space efficiency and potential for optimization. It is easier to do post processing and parallelization since we have only links, and it is easier to store future content. If someone wants to fetch a subset of info, they can do so using minimal space, rather than having to transfer large downloaded files

We also plan to take the 'filtering' capability we have and scale it up to become a filter for any topic (i.e. not just for recognition of faculty homepages). In other words, our tool is very customizable and can be thought of as a "Crawler for X". It can become a plug-in for any other

developer interested in the tool or interested in expanding it-- this is a huge potential impact to the research community.

We will also expand on the "recognition of homepages" functionality. We envision doing this by utilizing link prediction algorithms. Furthermore, even though the current output is useful, we plan to implement a framework which allows the generation of a separate file that is showing the dependency graph of the relationships between the links in the original output file. This is because currently, the output shows a "tree-like" structure of a parent link with an indent to show its child links.

As a side point, as was seen in the above screenshot, we have a simple UI created to serve as a homepage for the Open Crawler. Although the Open Crawler is part of a larger pipeline, having this UI helps envision a user actually using the tool and finding it useful.

Lastly, as a far-out goal, we are interested in introducing machine learning and having a recommendation system for CS research or literature news.

All in all, the above points illustrate that the potential impact with the Open Crawler as well as CSExpert is significant. We have learned a lot:
a) crawler implementation using Python b) optimization techniques c) sophisticated and research-heavy techniques such as link prediction d) each web page is formatted differently and it is a hassle to crawl it with 100% accuracy and no timeout.

Appendix: Individual Contributions:

Tanmay Shah (Project Coordinator)- read research papers to understand crawling algorithms as well as vertical-search engines & link prediction, assisted with project report, assisted with implementation and troubleshooting of crawler, created Powerpoint and recorded video presentation, created project proposal, scheduled several office hours appointments to better understand project scope and receive clarifications
Ashish Aggarwal- implementation of crawler, read resources to understand how to implement crawler and create the right output files, assisted with project report, assisted in creation of Powerpoint presentation, created project proposal, visited office hours
Jenny Tran- assisted with project report and proposal, assisted in creation of Powerpoint presentation, visited office hours

References: list references of work related to your project or websites with related systems here.

[1] https://www.youtube.com/watch?v=XjNm9bazxn8

[2] http://be.amazd.com/link-prediction/

[3] http://ieeexplore.ieee.org/document/5231915/?reload=true

[4] www.acsij.org/documents/v4i3/ACSIJ-2015-4-3-730.pdf

[5] https://medium.com/@tonywangcn/how-to-build-a-scaleable-crawler-to-crawl-million-pages-with-a-single-machine-in-just-2-hours-ab3e238d1c22

[6] http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1264819

[7] http://waset.org/publications/10075/information-retrieval-in-domain-specific-search-engine-with-machine-learning-approaches

[8] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.3818&rep=rep1&type=pdf

[9] http://statweb.stanford.edu/~owen/courses/319/achouldechova.pdf

[10] http://www.leonidzhukov.net/hse/2016/networks/papers/SNDA11.pdf

[11] https://www.minitex.umn.edu/Training/DisplaySessionHandout.aspx?SessionID=213