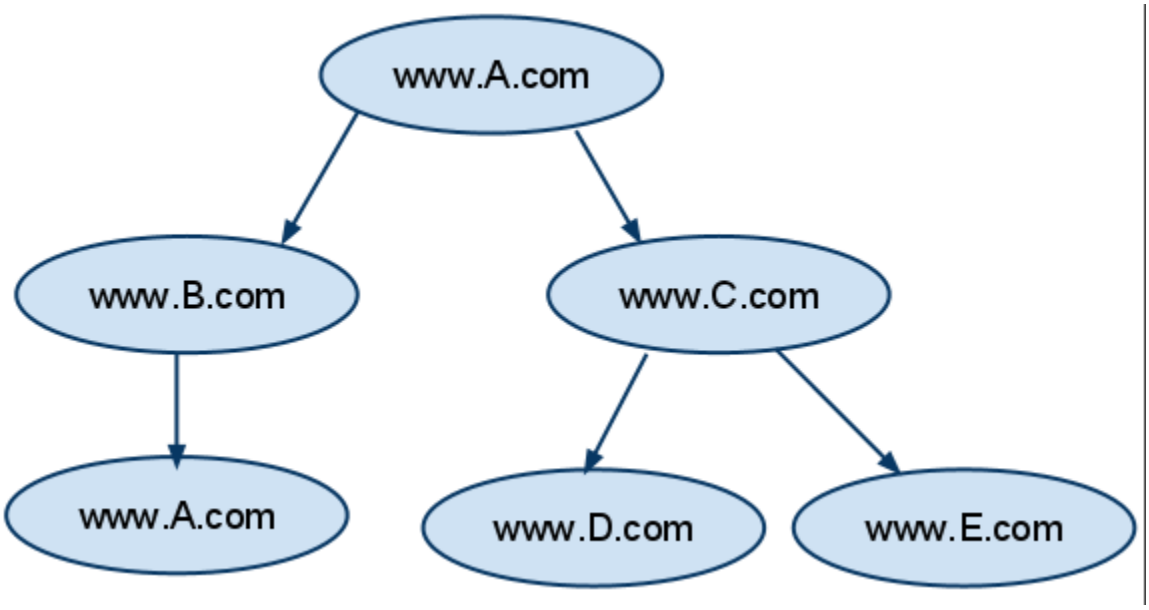
UOIT Blog Link Crawler Manual

Package Contents
"UOIT Blog Link Crawler Manual.doc" – Research description and report.
"v2-blog-link-crawler.py" – The tool for this research.
"Function manual.pdf" – Function guide to help when working with the code of the tool.

Introduction
Thanks for taking interest in this research project. What this crawler does is starting from an initial website it steps threw the hyper-links and records what links there are then steps threw more links and repeats. This allows us to explore and discover blog-based social networks and analyze their topographical nature. "Graph theory has provided the basis for modeling and analysis of networks of different types in the past 300 years. Once a network is modeled by a graph, topological parameters of a given graph can be analyzed in detail to provide important indications of connectivity in the graph. Models for physical networks have been proposed and analyzed in the past twenty years. General web page connectivity has also been analyzed in the past decade. The objective of this project is to extend these studies one layer higher, to the users of specific network services such as social networking sites. A mathematical model of the user connectivity in such cases would allow us to study its implications in business, marketing and social issues; for instance, the reach and effectiveness of advertising on facebook or blogs." (Dr. Heydari)

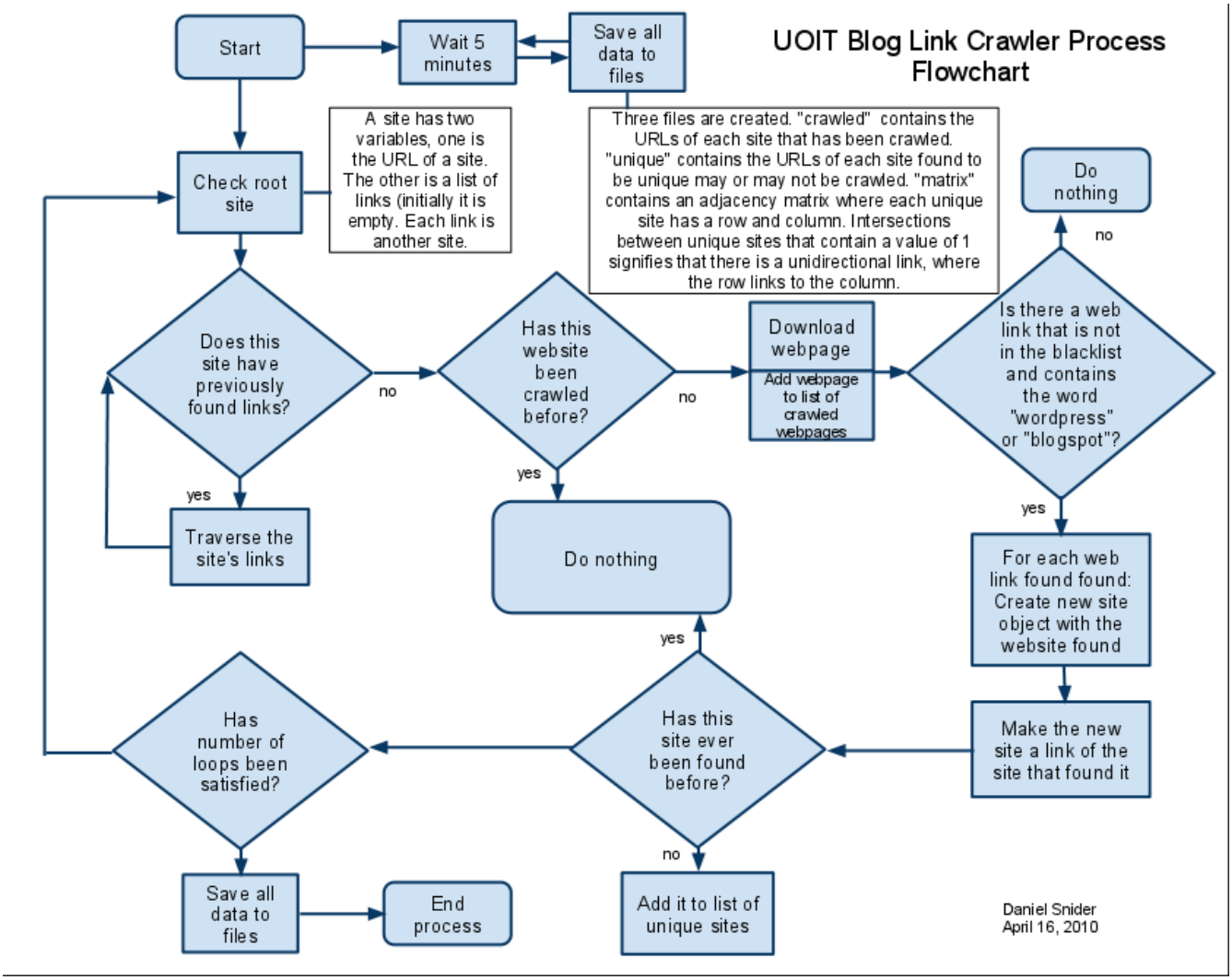
The Tool
The tool is written in python. The in-code comments are javadoc style comments. A manual describing each function, the function's parameters and the functions return values is available and should be included with this package. The crawling is done by downloading web pages using *urlib*. The web pages are inspected by *HTMLParser* and each new hyper-link that is found is



stored in a tree like structure. Below is an example of that data structure. The tree expands in a breadth first fashion. Also the crawler should not crash when it encounters a site that that is no long there, that redirects or is in a different language. Sites with these errors are ignored.

Tool Operation
The flow chart below describes the operation well. I feel it covers all the major operations of the tool. If you have any questions or need help feel free to contact me (contact section below).

Files
Three files are created. "crawled" contains the URLs of each site that has been crawled. "unique" contains the URLs of each site found to be unique may or may not be crawled. "matrix" contains an adjacency matrix where each unique site has a row and column. Intersections between unique sites that contain a value of 1 signifies that there is a unidirectional link, where the row links to the column. In the matrix that is generated **spaces separate columns and new lines separate rows.**



Installation Guide

On Linux:

1. download file “v2-blog-link-crawler.py”
2. open terminal
3. navigate to download directory using “cd [directory]” command
4. make the file executable using “chmod 755 v2-blog-link-crawler.py”
5. execute program with “./v2-blog-link-crawler.py”

When executing the program as described in step 5 you can also add on option. Like so “./v2-blog-link-crawler.py -d 7” this option sets the depth of crawling. Data will still be saved every 5 minutes. I wanted this too be changeable by using an option too but I didn't have time. Instructions are below on how to do that.

On windows:
I think (and I just Googled it) if you have python installed, you can type in cmd “C:\> python \Steve\Projects\Python\v2-blog-link-crawler.py” where the path is where the tool is located.

Output
Each “.” that is printed to terminal signifies that a site is about to be crawled. This helps to show that the program is still running. The numbers that show up indicate how many crawls have occurred, where each crawl searches all sites that have not been discovered but not downloaded or searched for links. The format is “[depth] = [number of links]”.

Code alteration tips
How to change the initial node:

Go to the line of code that says *root = Node("..... ")* and put a site where the …..s are. The URL must contain blogspot or wordpress in the URL.

How to ignore a site or domain so that it not crawled or collected:
Add a line of code near the other lines like these *blacklist.append("http://2chan.us/").* Add another site with the same style.

How to change how often data is saved:
Change the global variable "save_data_frequency". Unit is seconds.

Possible Future Improvements
-Multi threading the tool to crawl many sites at once would yield much larger data sets.
-Time to run feature would be nice. Could be implemented with a thread that runs till a certain amount of time then saves and exits. I just didn't have time to implement and test this. If this is still wanted I will volunteer my time to do this!!

Thoughts on This Tool
Professer Heydari said that he was interested in seeing chances in the topology over time. And this should be possible as all the files that are generated have time stamps. Also it is possible to determine what row or column is what website. Each each row or column number maps to the line number of the "unique" file.

I believe that with some thought and looking at some test results that interesting conclusions could be made about topographical characteristics and blog reach ability. In seems to me that because hyper links are unidirectional there is very poor reachability from one blog to a random other blog. But this could be due to not having a full view of the blogosphere or because we generally do not have very large depth (hop count from top to bottom) in our searches. It can take hours to crawl with depth greater than 6.

Also interesting is the speed at which the the number of links increase depending on the initial site. I found that art blogs and anime blogs expand rapidly, but a real-estate blog and a music blog expanded more slowly.

Thoughts on Facebook
Facebook has the technical capacity to be crawled in the same manner. However interpreting the Facebook pages is more difficult because the information important to us is written in JavaScript. But still against us is Facebook's policy states that "You will not collect users' content or information, or otherwise access Facebook, using automated means (such as harvesting bots, robots, spiders, or scrapers) without our permission."

Thoughts on Infoscape Research Group
Of Infoscape's research tools, no code was available for blog crawling. I do think we are doing something different than what Infoscape is doing because they crawl within a predefined list of political sites.

The one tool who's code was available was a facebook wall crawler that searched for links. It does not collect links to friends but instead hyper links that friends have posted on a persons wall. It could be manipulated to collect the friends who have posted on peoples walls, this would be a good start. But we could never collect the full list of friends without doing a lower level crawl.

Final Thoughts
I'm excited to see what results that might found when using matlab to analyze this tool's data. There are probably many mathematical qualities that threw graph theory would better help us understand the data. I truly hope this research is continued.

Contact
The author of this program and report is Daniel Snider, UOIT BIT Student. You can reach me at danielsnider12@gmail.com

Happy researching!