Link Analysis of DLPB Dataset

CSCI 485 – Data Mining
Vancouver Island University
Spring 2012

Casey Yardley

Contents

Goal

The goal of this report is to create a search engine to find the most influential papers in a dataset[1] from DPLB Computer Science Bibliography[2]. The dataset has 1,632,442 papers and 2,327,450 citation relationships[3], and its size is 845.4MB. The dataset is processed to create a matrix of references, which then has link analysis preformed on it by using a Page Rank[4] algorithm. The ranked results are then saved and searched in order for keywords, and a listing of the most influential papers are displayed.
A cache of papers from the top five most influential publishers is searched first. Since the references between papers are very disjoint, I combined references for each publisher and preformed link analysis on the publishers.

Concepts

The PageRank algorithm is a Link Analysis method that is similar to how the original Google search engine worked[5]. It works by creating a web of links between pages, and ignoring pages that don't have outgoing links. The algorithm depends on the idea of hubs and authorities. A page's hub value is influenced by the values of the authorities that it links to. A page's authority value is influenced by the values of the hubs that link to it[6]. Since the definition of hubs and authorities depend on each other, we must apply the calculations many times until the values converge[7]. Once this process is done, the values with the higher ranks are the most influential.
The main challenge was that references between papers are very disjoint. Many papers that are referenced frequently by others do not reference many papers themselves , and papers that references others frequently are not referenced by many other papers . There are many papers that are not referenced by any other papers at all. To solve this issue, I had to find a way to group papers together and then preform the link analysis between those groups. I decided to group them together by common publishers, since 'most influential publisher' seems like a good metric for searches.

Methods and Algorithms

1: SQL Database: *manage_db.py*

The first issue was how to efficiently look up papers. Scanning through the entire file once for every query is very impractical. I choose to populate a SQLite database with the dataset. This significantly reduced look up times, and allowed for efficient slicing and ordering of data.

2i: Create Reference Tables: *web.py*

The reference tables are created with data from SQL queries, and then used to construct the matrix of references. The *ref_dict* reference table maps paper IDs to a list of papers that reference that paper. The *pub_dict* reference table maps paper IDs to their publishers. The *ipub_dict* reference table is the inverse of *pub_dict* and maps publishers to the reference id of some paper that it published. The reference tables are then serialized and saved to disk for future access. A *hub_dict* and *auth_dict* are also created, with record initial hub and authority values of each paper. Only papers with initial hub and authority values larger than *hubmin* and *authmin* are included when generating the *pub_dict*.

2ii: Create Web Table: *web.py*

The *web_dict* reference table maps publishers to a set of the publishers who publish a paper that reference a paper that it published. It starts out with an initial set of publishers, and then removes all publishers which don't have papers referencing them.

3: Create Matrix of References: *web.py*

The *web_matrix* is a zeros matrix of size equal to the number of publishers in the *web_dict*. After it is created, each column is marked according to the corresponding reference from *web_dict*. This creates the matrix of references between publishers. The *order* of the publishers in the matrix and *web_matrix* is then serialized and saved.

4: Link Analysis: *PageRank.py*

Now the PageRank algorithm takes the *web_matrix* as input and simulates on step of traffic following the reference links. After several iterations of this, the list of publishers are sorted by how influential they are, serialized, and saved to disk.

5. Cache: *manage_db.py*

Now that the list of top publishers is saved, a cache of their papers can be saved in order to speed up searches. Since the cache contains significantly less papers than the full database, the search time on the cache is much faster (at least for small queries or frequent search terms).

6: Search: *search.py, index.py*

Users can enter search terms, and the papers from the most influential publishers are displayed to the user. First the cache is searched, and if the query needs more results then the full database of papers is searched. There is a command line and a web interface. The web interface allows the user to specify how many papers they want, a maximum time for the query to run, and a minimum number for how often a result has been referenced by other papers in the dataset.

Results

The following table compares a naive ranking of how often publishers reference and are referenced by others to the results of the PageRank algorithm. In the table:

- **Naive Hubs** refers to the publishers who reference the most other publishers
- **Naive Authorities** refers to the publishers who are referenced by the most other publishers
- **The Most Influential Publishers** refers to the most influential publishers, generated by the PageRank algorithm.

	Naive Hubs	**Naive Authorities**	**The Most Influential Publishers:**
1	Commun. ACM	CoRR	Commun. ACM
2	Theor. Comput. Sci.	ICRA	IEEE Computer
3	IEEE Trans. Computers	Discrete Mathematics	J. ACM
4	DAC	Commun. ACM	ACM Comput. Surv.
5	Winter Simulation Conference	Theor. Comput. Sci.	SIGMOD Conference
6	IEEE Trans. Pattern Anal. Mach. Intell.	IEICE Transactions	STOC
7	Inf. Process. Lett.	Microwave Theory and Techniques, IEEE Transactions on	IEEE Trans. Software Eng.
8	IEEE Trans. Software Eng.	IEEE Transactions on Information Theory	ACM Trans. Program. Lang. Syst.
9	Pattern Recognition Letters	Electronics Letters	POPL
10	VLDB	HICSS	VLBD

The search implementation can be demonstrated by running *search.py* in a terminal, or by launching *index.py* with a web browser.

Future Work

This implementation of a DLPD papers search engine could be be improved by including the papers in the reference matrix. To do this the disjoint papers would need to be connected. One way this could be done is by joining similar papers in sets smaller than entire publishers. For example, clustering based on keyword frequency. Another issue is SQL query speeds. They are much faster than other data access methods, but the current search implementation could probably be optimized to execute less of them, which would significantly speed up searches. The cache does help to fix this, but large queries can still be slow. Optimizations to the cache would help.

References

1. Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. ArnetMiner: Extraction and Mining of Academic Social Networks. In *Proceedings of the Fourteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (SIGKDD'2008). pp.990-998.

2. http://www.informatik.uni-trier.de/~ley/db/about/faqdblp.html

3. http://arnetminer.org/DBLP_Citation

4. http://www.peterbe.com/plog/blogitem-040321-1/PageRank.py

5. http://www.peterbe.com/plog/blogitem-040321-1

6. http://nlp.stanford.edu/IR-book/html/htmledition/hubs-and-authorities-1.html

7. http://webhome.cs.uvic.ca/~mgbarsky/DM_LABS/LAB_9/Lab9_rankingWEB.pdf