Lab 6 Report

Author: Ryan Hunter (rghunter@calpoly.edu)
Section: CSC 466-03

Intro

In this lab I implemented the PageRank algorithms and variations of it to rank the nodes in provided graphs. I take a look at graphs that represent NCAA Football seasons, dolphin friendships, Les Miserables characters, university karate club friendships, Amazon purchases, Wikipedia administrator elections, and a peer-to-peer file sharing service.

Implementation

My implementation of PageRank utilized the fact that the iterative process of calculating ranks can be optimized with matrix multiplication. So, for each graph I first created an adjacency matrix. This allowed each node to be represented by it's adjacencies as a row in the adjacency matrix. Position [i, j] in the matrix had a non-zero value if there was an edge from node i to node j. In graphs with weighted edges this value was equal to the weight(i, j) / sum(all weight(i)) where weight(i, j) was the weight of the edge. For most datasets this weight was provided in the input, but for the NCAA football dataset weight(i, j) = (score for team j) / (total game score) when team j beat team i. This decision is discussed with the results of the NCAA football dataset later in the report. As some datasets (namely Amazon's dataset) were prohibitively large when storing the full matrix, I had to utilize SciPy's sparse matrix data type which only stores the relevant parts of the dataset in memory.

Once the adjacency matrix had been created for the graph, a vector of ranks was initialized to 1/n where n is the number of nodes in the graph. Then new ranks were iteratively calculated as they are in PageRank. To accomplish this, I took the transpose of the adjacency matrix and multiplied it by the rank vector to sum up, for each node, the edge weights of nodes that have an edge going to it. Then, to adjust for sink nodes and the possibility that our graphs don't truly represent the complex relationship of the datasets we multiply this sum by some $d < 1$ and add $(1 - d)/n$ to all ranks as is done in PageRank. To simplify results, d was fixed at 0.95 for all datasets. This iteration continues until the sum of differences in the weights is less than some epsilon.

Results

Les Miserables

For this dataset I utilized the provided edge weights as described in the implementation section above. Based on my memory of the story and a quick google of the main characters, I believe this an adequate ranking. However, it should be noted that prevalence of characters in chapters may be a poor metric since characters that may appear around main characters but aren't crucial to the story will end up with disproportionate ranks to their importance. Here is the top few ranks:

Valjean with 0.1009
Marius with 0.0589
Enjolras with 0.0463
Courfeyrac with 0.0421
Cosette with 0.0406
Thenardier with 0.0373
Combeferre with 0.0340
Bossuet with 0.0331
Gavroche with 0.0309

Karate Friendships

This dataset was relatively simple and most closely resembled the internet graph ranked by PageRank, with the key difference being that this graph was undirected. A quick visual inspection of the degree of each node suggests the top ranks were all amongst the nodes with the most friends, thus I believe this a good ranking. Here are the top few ranks:

34 with 0.0629
33 with 0.0584
1 with 0.0564
2 with 0.0483
25 with 0.0423
6 with 0.0414
7 with 0.0414
26 with 0.0396
3 with 0.0396
30 with 0.0374

Dolphin Friendships

This dataset closely resembles the Karate dataset, and my analysis of the results are much the same. Based on the degree of the nodes, I believe the most social dolphins are the highest ranked by the algorithm.

Trigger with 0.0342
Jet with 0.0291
Ripplefluke with 0.0286
TR88 with 0.0268
TR120 with 0.0246
Web with 0.0244
SN63 with 0.0234
Patchback with 0.0231
Zipfel with 0.0230
Zig with 0.0222
Scabs with 0.0204
Bumper with 0.0202

NCAA-Football

For this dataset I created the adjacency matrix in such a way to account for the scores as well as the wins of the teams. Ultimately, I believe this was along the right line of thinking, but the results do have some interesting quirks. While most teams towards the top of the rankings had good seasons in 2009, research shows Mississippi was a good team but definitely didn't deserve the number one spot. However, they did have some very large wins over a few "middle of the pack" teams. This ranking alone suggests that our method of accounting for score might not be the best way to incorporate scores into consideration with this dataset. Here are the top few teams:
Mississippi with 0.0381
Florida with 0.0313
Wake Forest with 0.0182
Oregon State with 0.0170
Alabama with 0.0164
Utah with 0.0162
Oklahoma with 0.0146
Vanderbilt with 0.0143
USC with 0.0134
Texas Tech with 0.0132
South Carolina with 0.0128
Virginia Tech with 0.0125

WikiVote

This dataset, like the rest of the snap datasets, was most similar to the Dolphins and Karate datasets in style: an undirected, unweighted graph. Given that the relationships in this graph represent "votes" of sorts, I think this is a decent ranking of the most deserving of the position being voted for only if voters could choose how many people to vote for. Then, votes would be more highly impactful on the rank if it came from a voter who only voted for one

person. I see voting for a single individual vs. many as representative of higher confidence in the vote. Otherwise, I'm not sure this is a great representation of anything. Here is the top few nodes with rankings:
4037 with 0.00069298
6634 with 0.00063893
15 with 0.00057531
2625 with 0.00053102
2398 with 0.00043092
2237 with 0.00036587
4191 with 0.00036112
2470 with 0.00035884
7553 with 0.00035113
5254 with 0.00033795
5412 with 0.00032696

GNutella

Gnutella is a peer-to-peer file sharing site. This implies that the network of peers represented by the graph somewhat represents networks of content. That is to say, we would expect a closely connected group of nodes to have or want somewhat related content. This informs an idea similar to the original of PageRank. A heavily connected node likely has a lot of good content that peers want, and the amount of other influential nodes that get data from it would imply it has a higher importance. It is a very similar network to the web's network of webpages. Because of this, I believe the ranking represents a very reasonable ranking of peers on the network. The top few nodes with rankings is as follows:
1676 with 0.00010585
1020 with 0.00010325
386 with 0.00009839
222 with 0.00009796
388 with 0.00009557
227 with 0.00009518
389 with 0.00009329
688 with 0.00008987
842 with 0.00008635
226 with 0.00008585

Amazon

This dataset was the largest and presented the most significant challenge to my implementation's efficiency. However, my use of sparse matrices and reliance on matrix operations proved enough to calculate results for this ranking relatively quickly. My belief is that this algorithm does an alright job of finding which items are most likely to be purchased with another item since an item has a higher rank if it's purchased more frequently with other items. I

also think the part of PageRank that considers a random change of web page works well here, a user may purchase an entirely unrelated item so there is always a chance of relationship to some items. Here's an output of the top few items and their ranks:

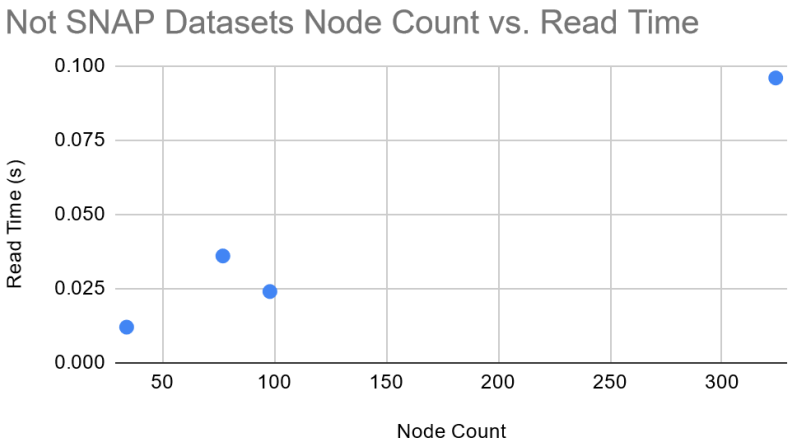
593 with 0.00393174287876
595 with 0.00369819691867
591 with 0.00367715638841
972 with 0.00265710277153
590 with 0.00260498985392
976 with 0.00225385929251
974 with 0.00219906944920
89 with 0.00214763541465
978 with 0.00214755439655
975 with 0.00191419765902

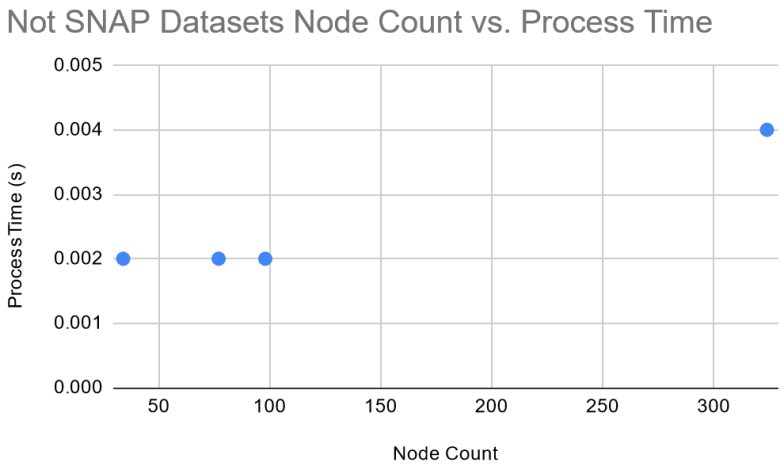
Summary

Ultimately, PageRank did an okay job at ranking the datasets. I think it worked best in situations where the dataset had relationships that were roughly translatable to the same problem that PageRank solved. It also did well on simple graphs mapping friendships and basic relationships. I'm a little suspicious of its college football rankings, but that could be due to my own failure to create the graph well.

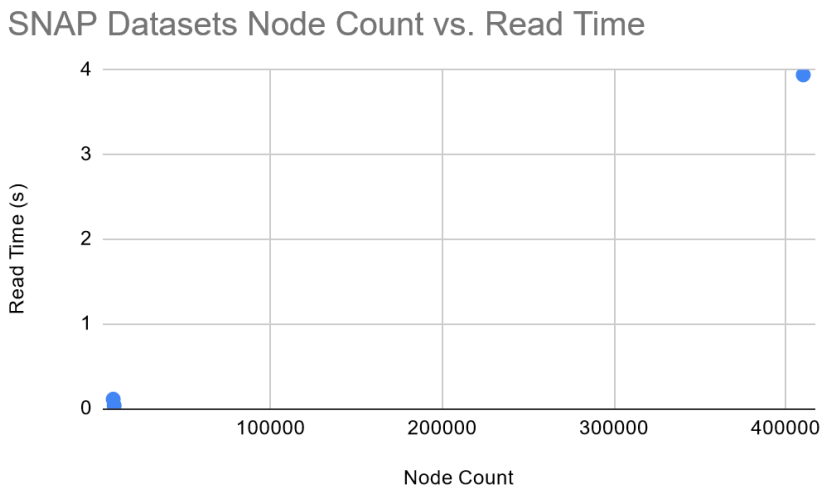
Performance Evaluation

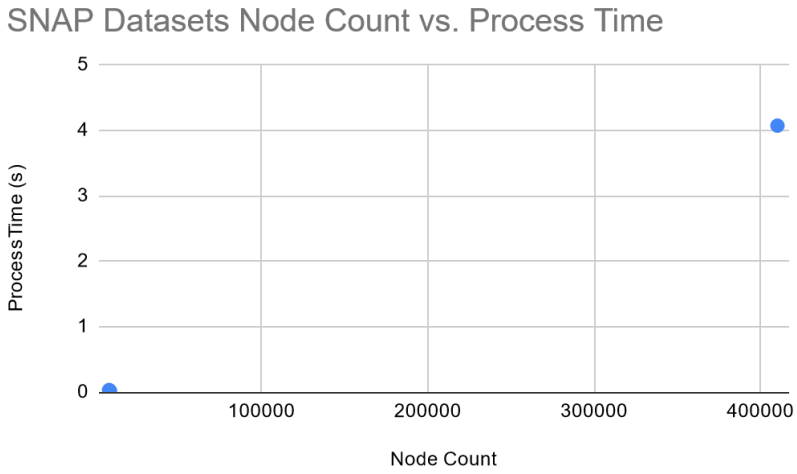
Not SNAP Datasets





SNAP Datasets





The timing results suggest that both the reading and the processing run in close to linear time. The reading portion of the results is self-explanatory. For processing, this makes sense considering my implementation relies heavily on sparse matrix multiplication which is best case linear time.

Appendix

usage: pageRank.py [-h] [--data DATA]

optional arguments:
  -h, --help show this help message and exit
  --data DATA, -d DATA csv or txt file to run version of pageRank on

If the file ends in .txt pageRank assumes it's a SNAP dataset and processes it as such.
No modification to epsilon or d can be done from the command line. My implementation automatically
figures out if a provided csv dataset is directed or undirected, and weighted or unweighted.