CS685 HW 4 Report – Josh Wilson

Implementing the PageRank algorithm as outlined by Page and Brin was both fun and a very good learning experience. As usual, I'll outline the design, implementation/optimization, and analysis of data for this project, and I will also discuss the challenges faced along the way.

Design

The design of this project wasn't all that involved of a task, as I roughly knew what steps would need to occur going in from the prior readings we had completed. Essentially, there would be three broad tasks: read in the data and build the google matrix, run the power method on said matrix until a (near) convergence, and then process and display said data. Given that the data for the engr.uky.edu graph was kindly provided in json format, I knew it would just make sense to implement this algorithm in node.js. As I have learned many a time in my career, it is *always* easier to work with json in javascript, full stop. I also planned to make use of both the math.js and lodash libraries for matrix operations and data operations, respectively.

Implementation

Apart from taking a little extra time to fully understand the mathematics at play here, the implementation of this project went off fully in line with the stated design. I solved the first task of reading in the graphs and building the matrix with foreach loops over the raw json data. I took said graph and converted it into matrices H and A, using the same format specified in our prior AMS reading. I then looped over various values of alpha/dampening factor, incremented by +0.05 each iteration. Using alpha, the values of H and A, and the total node count of the graph, I was able to successfully build the "google matrix" G.

For the second task of applying the power method to matrix G, I simply did just that. I took an arbitrary guess as to the eigenvalues of the eigenvector, then looped and multiplied that eigenvector against G until the Frobenius norm of the difference between the previous iteration and the current iteration was greater than 0.00001. With this approach, every test case I tried was able to converge to a reasonable ranking.

Lastly, I tackled displaying the data to the user. For this, I simply added data for the alpha values, iteration counts, and rankings for each node into two separate json objects, which I then saved to .json files on disk (they can be found in the application/RawOutput directory). For the finalRanks_*.json files, rankings for each node are given for each value of alpha used. For the iterationsVersusAlpha_*.json files, you can see how many power method iterations were necessary for a given value of alpha (more on this in the analysis section).

All in all, this implementation went very smoothly. I very much stand by my decision to use javascript here- I was able to implement this entire algorithm in just over 100 lines. It simply reinforces my belief that javascript is the way to go if you're working with json data.

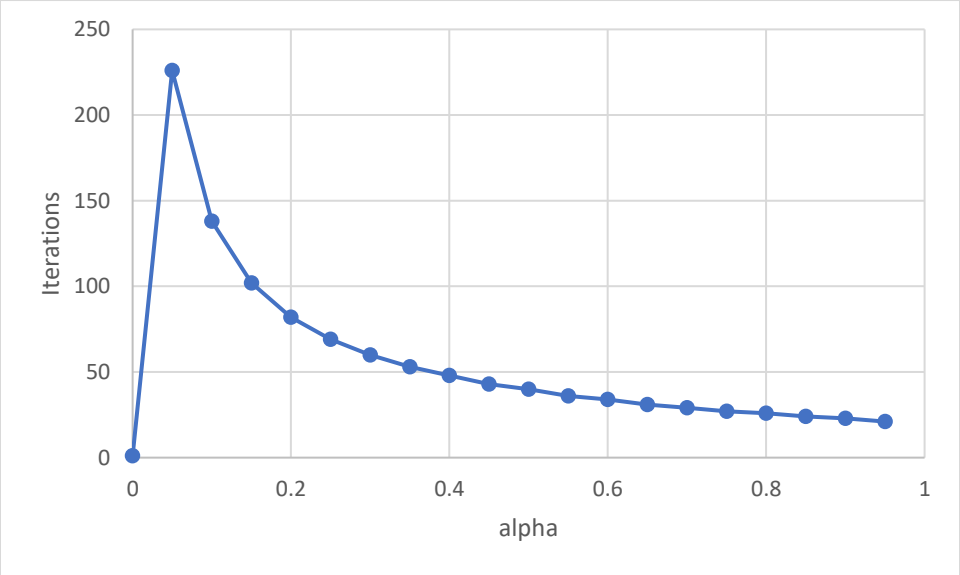
Analysis

The ranks for all of the graphs I ran through this algorithm conformed to what I'd expect, with a few surprises here and there. I ran the algorithm on the three graphs specified in the homework assignment pdf, as well as the eight-node graph from the AMS paper, and of course the engr.uky.edu graph. All of these graphs can be found represented in json in the application/RawInput directory, for your reference. As per Brin and Page, when discussing ranks of graphs, I'll be using the values which were calculated for an alpha of 0.85.

The ranks for the three homework graphs went about how I'd expect, right up until the third one. The first graph had the "Page C" node ranked highest, which makes perfect sense due to its high concentration of inbound links. The second graph had the "Home" node ranked highest and the dangling links ranked lowest, which also checks out. The third graph was a similar story, except that the added "Review" pages shared the same ranking as the dangling nodes, which surprised me. However, upon further investigation, I noticed that the Home/About/Product/Links nodes all gained additional rank from the inclusion of the Review node links, which indicates that everything was working as intended. I also performed some checks to ensure that only the dangling links were being added into matrix A, which proved to be the case. I can only assume that this is just the way the math works out given how the graph was constructed.

For the eight-node graph given in the AMS paper, the rankings were about what I would expect as well. The interesting part was that the values given did not align exactly with what the author had gathered, even though the rank positions of the pages themselves are the same (i.e. the 8 node was ranked highest, the 6 node was ranked second highest, etc.). I can only assume this is due to differences in floating point notation, or for the simple fact that I cut my power method implementation off at a given threshold rather than converging all the way to zero.

Lastly, for the graph of engr.uky.edu, the top ranked and bottom ranked pages all make perfect sense to me. The highest ranked pages were pages that served as directories, such as the search pages and pages for a particular department in the college. This checks out, because a link to the search page is there on almost every page, and a link to each department's home page is at the top of each page within that department. Furthermore, the lowest ranked pages were all pages for individual faculty/staff, which I think occurs because they all have just a few inbound links from directory pages. However, what was absolutely confounding to me was the fact that the top page of this domain ("https://engr.uky.edu") is dead last in the ranking. This initially made no sense whatsoever, as there should be a link to this page from almost every page in the domain (excluding subdomains/department pages)- you simply click on the college of engineering logo to visit it. However, upon examining the raw data, I found that this link appeared exactly once: the single time it appeared as a key. To me, this hints at issues with the implementation of the crawler that generated this graph.



To wrap up this section, I'll give a brief word on the data surrounding the value of alpha versus the number of iterations for engr.uky.edu graph. What struck me as being particularly interesting here is that the number of iterations was highest when alpha was 0.05, and that it decreased somewhat exponentially from there. This stands in stark contrast to every other graph I tried, where the number of iterations increased as alpha increased. I am not sure quite why this is, mathematically speaking, as all of these values were generated from the exact same code, and the rankings seem entirely reasonable to me. Perhaps it's some issue with the data, or that a large data set is just more representative of the way the power method truly works.

Conclusions

All in all, this project was a good experience. I greatly enjoyed implementing the discussed algorithm, and I am very pleased at just how smooth the design and implementation was. Improvements to the application itself would mostly involve adding command line arguments to specify input/output files a bit more cleanly. I also enjoyed analyzing the results that my program kicked out,

and I would love to get more insight into the question of alpha versus iterations, as that's an interesting observation.