Page Rank Christopher Donlan

1 Report

1.1 Implementation

Considering the ergodic version of the Page Rank transition matrix, P, an $n \times n$ matrix, $n =$ number of websites, the values in P correspond to the equation:

$$\alpha = 0.85 \tag{1}$$

$$P(y, x) = \frac{1-\alpha}{n}(\text{no link}) \tag{2}$$

$$P(y, x) = \frac{1-\alpha}{n} + \frac{\alpha}{|\text{Out}(y)|}(\text{link exists}) \tag{3}$$

$$\tag{4}$$

Realizing that a vector multiplication between a prior page rank, π and P evaluates to a sum over i of $\pi_i * \frac{1-\alpha}{n} + \ldots$, and combining this with the knowledge that $\sum_{i=1\to n} \pi_i = 1$, each vector multiplication reduces to:

$$\frac{1-\alpha}{n} + \sum_{i=1\to n} \frac{\pi_i * alpha}{|\text{Out}(j)|}(\text{all summed over j})$$

Since an internet graph is highly sparse, implementing this calculation with an adjacency list (and substituting the fraction $\frac{1-\alpha}{n}$ for the sum) allows each iteration to be done in $O(n)$ time.
This is the implementation I used, in C++.

1.2 Results

In order to test the change to the page rank vector with respect to alpha, I decided to check the behavior of the top ten sites and calculate both the L1 and L2 norms.

Varying alpha caused the page rank with respect to the lower ranked sites to swap around 0.85 exactly. As alpha increases from then on, the page rank continues to decline for those sites.

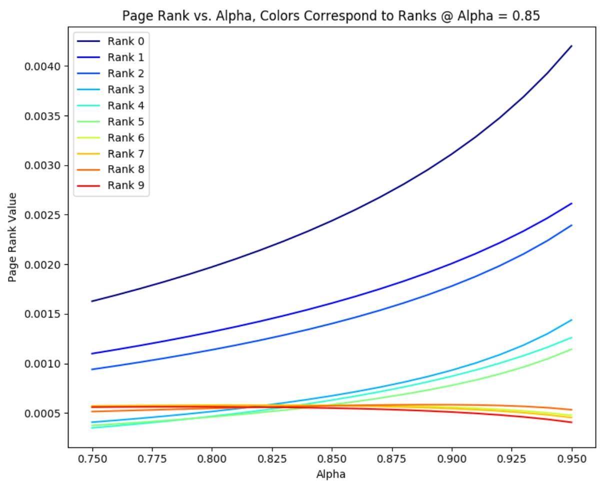
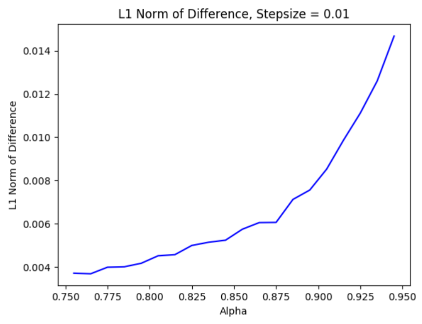
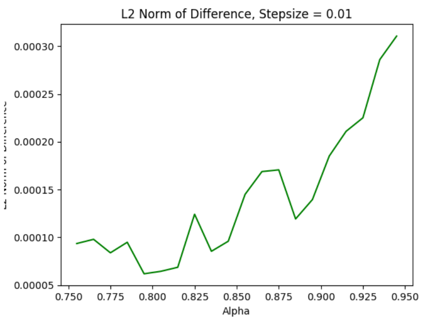


Figure 1: The results of varying alpha. The colors correspond to rank at $\alpha = 0.85$, from blue (0) to red (9).

Additionally, I checked the L1, $\sum_{u\in V} |PageRank1(u) - PageRank2(u)|$, and the L2 norm (euclidean distance).



(a) L1 Norm



(b) L2 Norm

This revealed that, while the absolute distance between the page rank vectors remains fairly smooth, there are spikes in euclidean distances, especially around 0.85. Combining the three graphs, it seems that the value of alpha is "smoothing" the page rank differences. As alpha approaches 1, the page rank difference between the top site and the rest of the sites becomes very large. If alpha is reduced past 0.85, then the page rank becomes less accurate. So alpha is likely, from this perspective, called the "dampening factor" because it dampens the differences between the top sites and the intermediate and lower ranked sites. However, if it is lowered too far, then the random likelihood of arriving at any given site is so high that underlying graph structure (which Page Rank is designed to identify) disappears underneath all the random reset edges.

2 Extra Credit

2.1 Part 1:

In order to solve the extra credit problems, I figured I would try a binary search from the bottom (lowest ranked sites) to see how many low ranked sites it would take to reach the top 1%. I found that it took very few for both the small and large datasets. The algorithm proceeds by adding incoming links from all websites ranked 875712 up to β, the current binary search value. I then run Page Rank, and find out whether I am in the top 1%. Once I am in the top 1%, I search backwards to the optimal value (just a binary search).

So, I started with a binary search. Then, I realized that what I am looking for is the **optimal region** of the graph to start adding (since, with a binary search, I add a continuous block of sites to my set of incoming links).

In this light, I created a dynamic programming solution wherein I iterated from the lowest rank to the highest rank, and used the binary search starting from the current rank as the sub problem.

$$p(i) : \text{binary search sub-problem starting at } i \text{ and returning a list of neighbors to add} \quad (5)$$

$$c(l) = \text{cost function of a list } l \text{ of neighbors to add.} \quad (6)$$

$$T[i] = min\{c(p(i)), T[i-1]\} \quad (7)$$

$$L[i] = \text{ stores the minimum cost neighbor list} \quad (8)$$

Results This method only ran to completion on the small dataset. In order to simulate the large dataset, I configured the cost equation such that the cost of links from the lowest ranked sites had the same value as they would have had in the large dataset.

In the running of this algorithm on small, there was an initial solution (from the lowest ranked sites) that was replaced only once over the course of the DP iteration.

So, I decided to keep this algorithm and run it on the large dataset overnight. In doing so, I arrived at a final solution with incoming edges from the first 416 lowest ranked websites, with a final cost of: $1.07231 * 10^8$.

As I solved this problem, I noticed a key issue that could disqualify a valid solution. In machine learning, there can be "future leak" when testing a learning method. In that context, I realized there could be page rank "leak," where I used the page ranks of the sites *after* adding the links in order to decide which links to add in the next iteration.. So I removed this in order to produce my solution.

2.2 Part 2:

Using the small dataset, I experimented. First I tried adding complete graphs of fake sites (everyone connected to each other), and connected them all to my site. Then I tried forming tree structures of connectivity, a tree structure that was connected to my site, connecting my site to all of the above...and I found that the most effective way to boost my site's rank was to simply connect the websites to my site with outgoing links only. No other links.

I also thought about connecting my site to other (real) sites, and connecting those sites to the fake sites to form a loop with the real sites. However, I did not have a chance to experiment with this. After exploring the ideas above, I suspect that this would not work either.

So I implemented a Binary Search for the number of fake sites needed to get within the top 1%. If $\tilde{8}0$ fake sites improved my ranking more than the lowest ranked real site, and the total number of fake sites needed was relatively low, then I would not need to account for swapping real sites with fake sites.

What I found was that the number of fake sites needed was very small and the cost was much, much lower than the comparable cost of the real site links.

I found that only about 4 fake sites (cost: $\$4,000$) were needed for the small dataset, and that 10 were needed for the large dataset (cost: $\$10,000$). So, I add sites $\#$'s 875714 through 875724, and add incoming links from those sites to my site.

I think this may indicate that the Page Rank algorithm is a little vulnerable...and that there may be room for improvement. Maybe this would be an opportunity...