CZ4052 Cloud Computing Assignment 2

PageRank

Eugene Poh

Introduction

In this study, I aim to explore the PageRank algorithm, we will look at proof of convergence of the two closed forms mentioned in the lecture. Then we will investigate the parameters of PageRank, teleportation probability, web graph matrix M, distribution vector E. Lastly, we will experiment PageRank and Parallel computing by implementing PageRank with MapReduce.

Convergence proof

To prove the two closed forms, we will use the Example from lecture notes 7 page 53 as the graph.

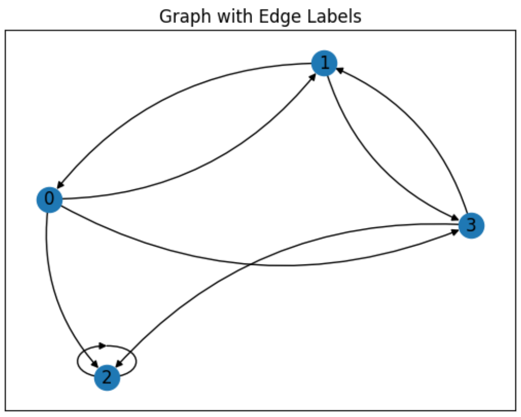

Fig 1. Example from lecture notes 7 page 53 graph.

I will use the simplified PageRank algorithm and the random surfer PageRank algorithm which is the simplified PageRank algorithm but with teleportation probability. Similarly, the modified PageRank algorithm is the same as simplified PageRank algorithm but with teleportation probability and distribution vector, E considered.

In Table 1. We can see that for all the PageRank algorithms, the values converge numerically to the closed form solutions. In addition, graphs are given for reference.

Algorithm	PageRank	Sum of PageRanks	No. of iterations
Simplified PageRank	[0. 0. 1. 0.]	1.0	51

Random Surfer PageRank	[0.08 0.11 0.71 0.11]	1.0	99
Modified PageRank	[0.08 0.11 0.71 0.11]	1.0	23

Table 1. Comparison of different PageRank algorithm on lecture notes example graph

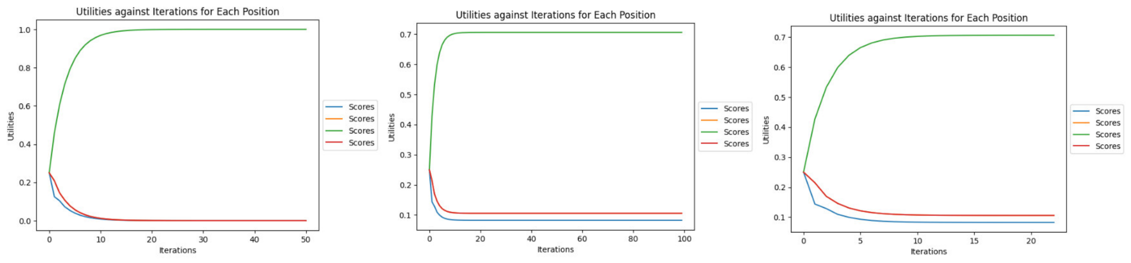

Fig 2. From left to right, simplified PageRank algorithm convergence, Random surfer PageRank algorithm convergence, Modified PageRank algorithm convergence.

Parameter Exploration

As seen above, there is convergence. However, what happens if we use large graphs?

To test this and better see the effects of the parameters, I originally used a large dataset from https://www.cs.cornell.edu/courses/cs685/2002fa/, "grp0_epa", which contains 4772 nodes and 8965 edges. However, despite the code working, its large size obfuscated much of the findings. Instead, I used a function `generate_input_data` to generate a random directed graph of 25 nodes and 40 edges.

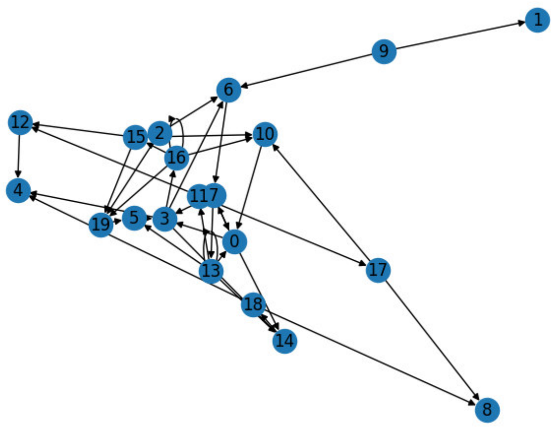

Fig 3. medium_graph generated from generate_input_data

Web matrix

For the initial web matrix, I simply normalized the values of each node by the number of outgoing nodes. E.g If node 1 has 3 nodes, its vote per outgoing connection will be 1/3. To account for dangling links and dead nodes, I initialized the nonexistent outgoing connections as 0.0.

This of course has disastrous consequences,

```
initial page_rank_scores:

[0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05]
closed form 1:

[2.87521244e-08 0.00000000e+00 0.00000000e+00 4.25312584e-08 4.26688800e-08 1.47974562e-08 … 3.67539359e-08 8.61258884e-09]

closed form 2: 3.3328083864285925e-07

no_of_iterations: 67
```

I proceeded to try removing the deadnodes recursively, which worked.

```
initial page_rank_scores:

[0.07142857 … 0.07142857 0.07142857 0.07142857 0.07142857 0.07142857 0.07142857]

closed form 1:

[0.13411761 0 0 0.18352845 0.06352914 0.09176466 0.15882395 0 0.06352976 0.01764699 0.07058827 0.03058846 0.12235312 0.01764701 0.04588258]

closed form 2: 0.9999999999999994

no_of_iterations: 36
```

However, what happens if I replace the dead node with a node that is connected to every other node? This in theory would be redistributing the pageranks. To do so, I used the uniform distribution value i.e 1/number_of_nodes to be the values of all non-existing connections for the dead nodes, and it worked!

```
initial page_rank_scores:

[0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05]
```

```
closed form 1:

[0.0848353 0.01354905 0.00903269 0.11755794 0.10644039 0.05199119 0.04745502 0.08476589 …
0.05122964 0.01821006 0.08505521 0.03363644]

closed form 2: 1.0000000000000002

no_of_iterations: 23
```

This seems to be a better method than removing dead nodes due to the reduced number of necessary iterations.

Teleportation probability

To test for teleportation probability, we use 0.15 and 0.85 values as teleportation probability and use the random surfer PageRank algorithm. As expected, the teleportation probability evens out the ranks of all the nodes. E.g. max value of 0.15 is much higher than that of 0.85. In addition, the standard deviation for 0.85 was much lower, despite the two having almost the same mean.

Teleportation probability	Converged PageRanks	Max value	Mean	Standard deviation
0.15	[0.08173536 0.02119022 0.01487033 0.10799187 0.09581039 0.05533686 0.05045839 0.08091831 0.05641897 0.01487033 0.04258421 0.02261193 0.03158508 0.04553878 0.06871855 0.02507535 0.04802363 0.0244804 0.0732811 0.03849993]	0.107992	0.0499	0.0267
0.85	[0.05549853 0.04693721 0.04366252 0.05706385 0.05703731 0.05274597 0.05235179 0.05429022 0.0510282 0.04366252 0.05124774 0.04509686 0.04931714 0.04781137 0.05001168 0.04544703 0.04758692 0.04704478 0.05116427 0.05099407]	0.057064	0.05	0.0040

Table 2. comparison of teleportation probability

For more comparison, look at "teleporation_probability_experiment.txt" under "results" folder in github. Below is the plot of scores during convergence.

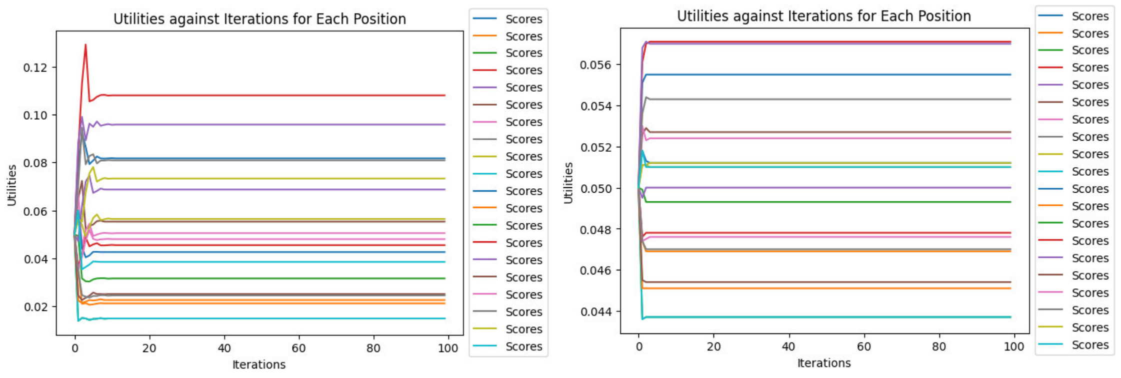

Fig 4. Teleportation convergence comparison

In addition, looking at the lecture notes example scenario's Table 1, where node C is a spider trap. We can see that without the teleportation probability, node C hogs all the rank, leaving nothing for the rest. But in the random surfer PageRank algorithm, it is somewhat mitigated.

Therefore, having the teleportation probability, helps us mitigate the spider traps and dead nodes.

Distribution vector

To test for the distribution vector E, we used E' which contains the average initial pagerank scores and E'' which contains the normalized uniform incremental pagerank scores based on order of the nodes.

From the experiments, it looks like we can influence the rankings of certain websites via E. For example, if we look at the last node. When E is average, the value 0.03849992, is lower than its linear counterpart, 0.04563386.

Distribution Vector, E	Converged PageRanks	Max value
average	[0.08173544 0.02119031 0.01487042 0.10799243 0.09580976 0.05533703 0.05045835 0.08091833 0.05641838 0.01487042 0.04258416 0.02261205 0.03158516 0.04553874 0.06871872 0.02507538 0.04802353 0.0244805 0.07328095 0.03849992]	0.107
linear	[0.07564731 0.01411946 0.00881417 0.10011836 0.09559825 0.05766523 0.04308816 0.07081974 0.0605239 0.01434048 0.04782665 0.02361327 0.0352954 0.04525783 0.06869039 0.0301791 0.05224382 0.03069199 0.07983262 0.04563386]	0.100

Table 3. Distribution Vector E comparison.

PageRank with parallel programming

For the map reduce, I adapted from https://www.youtube.com/watch?v=9e3geIYFOF4&ab_channel=VictorLavrenko‘s map reduce algorithm to compute PageRank.

The mapper calculates the contribution of each node's PageRank to its neighboring nodes. The reducer updates the PageRank of each node based on the contributions received from its neighbors. It also adds a teleportation factor for nodes that don't have incoming links, ensuring a degree of randomness in the ranking. In the main map reduce function, it initializes the PageRank values for each node, sets up the damping factor, and then iteratively applies the mapper and reducer functions for a specified number of iterations to update the PageRank values until convergence. It returns the final PageRank values.

Converged PageRank	Sum of PageRanks
[0.0795 0.0205 0.0150 0.1101 0.0930 0.0584 0.0511 0.0810 0.0543 0.0150 0.0397 0.0228 0.0325 0.0458 0.0687 0.0277 0.0447 0.0247 0.0734 0.0419]	1

Table 4. results of Map reduce PageRank.

Appendix

https://github.com/Eugene7997/CZ4052_assignment_2