Introduction

PageRank (PR) is an algorithm used by Google Search to rank web pages in their search engine results. It is named after both the term "web page" and co-founder Larry Page. PageRank is a way of measuring the importance of website pages. PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is.

Lecture Example (Slide 53 of Lecture 7)

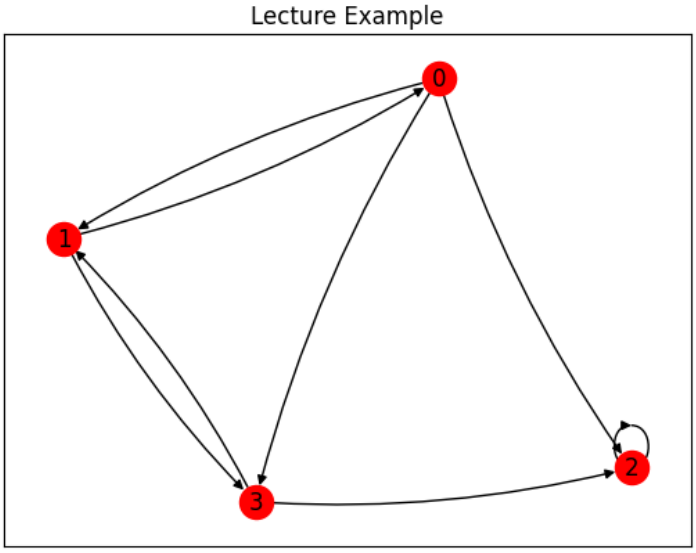

Figure 1: Graph for Slide 53 of Lecture 7

PageRank Algo	Iteration PR	Closed Form PR	Sanity Check	Iteration
Simplified	[0, 0, 1, 0]	Error	NIL	41
Modified	[0.10135159 0.12837872 0.64189097 0.12837872]	[0.10135135 0.12837838 0.64189189 0.12837838]	Correct	24

Figure 2: Results for Slide 53 of Lecture 7

Figure 2 shows the results for illustrative example of four-webpages on Slide 53 of Lecture 7. The simplified PageRank algorithm computes PageRank scores solely based on the link structure of a graph, without considering additional factors. Simplified page rank algorithm does not take into account of dangling nodes, spider traps and dead ends. The lecture example is an example of spider trap. Spider traps are a group of pages that have links only to the pages within the group.

To resolve the issue, modified PageRank algorithm is being used. It considers the teleportation probability as well as the distribution vector to refine the PageRank computation according to specific requirements or user preferences. For modified model, the “random surfer” simply keeps clicking successive links at random, but periodically “gets bored” and jumps to a random page based on the distribution of E which will jump out of the spider traps, dangling nodes or dead ends.

Parameters Tuning

In this section, I have explored tuning several parameters for the page rank algorithm, such as the teleportation probability, web graph matrix M, and the distribution vector.

Teleportation Probability

The teleportation probability refers to the chances of staying on the same node. The higher the teleportation probability, the higher the chances of jumping to a different node.

Teleportation Probability	Iteration PR	Closed Form PR	Sanity Check	Iteration
0.1	[0.06024125 0.07831366 0.78313142 0.07831366]	[0.06024096 0.07831325 0.78313253 0.07831325]	Correct	31
0.11	[0.06501468 0.08430242 0.76638049 0.08430242]	[0.06501438 0.08430198 0.76638165 0.08430198]	Correct	30
0.12	[0.06961666 0.0900376 0.75030815 0.0900376]	[0.06961634 0.09003713 0.75030941 0.09003713]	Correct	29
0.13	[0.07405742 0.09553411 0.73487437 0.09553411]	[0.07405719 0.09553378 0.73487524 0.09553378]	Correct	29
0.14	[0.07834676 0.10080621 0.72004082 0.10080621]	[0.07834652 0.10080585 0.72004178 0.10080585]	Correct	28
0.15	[0.0824934 0.10586658 0.70577343 0.10586658]	[0.08249313 0.10586618 0.70577452 0.10586618]	Correct	27
0.16	[0.08650551 0.11072711 0.69204027 0.11072711]	[0.08650519 0.11072664 0.69204152 0.11072664]	Correct	26
0.17	[0.0903905 0.11539858 0.67881234 0.11539858]	[0.09039027 0.11539825 0.67881323 0.11539825]	Correct	26
0.18	[0.09415566 0.11989159 0.66606116 0.11989159]	[0.09415539 0.1198912 0.66606221 0.1198912]	Correct	25
0.19	[0.09780726 0.12421525 0.65376224 0.12421525]	[0.09780706 0.12421497 0.653763 0.12421497]	Correct	25
0.2	[0.10135159 0.12837872 0.64189097 0.12837872]	[0.10135135 0.12837838 0.64189189 0.12837838]	Correct	24
0.21	[0.1047942 0.13239006 0.63042568 0.13239006]	[0.10479391 0.13238963 0.63042683 0.13238963]	Correct	23
0.22	[0.1081402 0.1362567 0.6193464 0.1362567]	[0.10813999 0.13625639 0.61934723 0.13625639]	Correct	23
0.23	[0.1113948 0.13998619 0.60863281 0.13998619]	[0.11139453 0.13998579 0.60863388 0.13998579]	Correct	22
0.24	[0.11456232 0.14358481 0.59826806 0.14358481]	[0.11456212 0.14358452 0.59826884 0.14358452]	Correct	22
0.25	[0.11764732 0.14705921 0.58823426 0.14705921]	[0.11764706 0.14705882 0.58823529 0.14705882]	Correct	21
0.26	[0.12065358 0.15041483 0.57851675 0.15041483]	[0.12065338 0.15041455 0.57851751 0.15041455]	Correct	21
0.27	[0.12358514 0.15365758 0.56909969 0.15365758]	[0.12358488 0.1536572 0.56910073 0.1536572]	Correct	20
0.28	[0.12644528 0.15679219 0.55997033 0.15679219]	[0.12644509 0.15679191 0.5599711 0.15679191]	Correct	20
0.29	[0.12923749 0.15982373 0.55111504 0.15982373]	[0.12923735 0.15982352 0.5511156 0.15982352]	Correct	20
0.3	[0.13196502 0.1627569 0.54252119 0.1627569]	[0.13196481 0.1627566 0.54252199 0.1627566]	Correct	19
0.31	[0.13463057 0.16559564 0.53417816 0.16559564]	[0.13463042 0.16559541 0.53417875 0.16559541]	Correct	19
0.32	[0.13723719 0.16834434 0.52607414 0.16834434]	[0.13723696 0.16834401 0.52607502 0.16834401]	Correct	18
0.33	[0.13978724 0.17100643 0.5181999 0.17100643]	[0.13978707 0.17100618 0.51820056 0.17100618]	Correct	18
0.34	[0.14228335 0.17358572 0.51054521 0.17358572]	[0.14228323 0.17358554 0.5105457 0.17358554]	Correct	18
0.35	[0.14472797 0.17608574 0.50310055 0.17608574]	[0.14472777 0.17608546 0.50310131 0.17608546]	Correct	17
0.36	[0.14712307 0.17850937 0.4958582 0.17850937]	[0.14712293 0.17850915 0.49585876 0.17850915]	Correct	17
0.37	[0.14947103 0.18086 0.48880897 0.18086]	[0.14947079 0.18085966 0.48880989 0.18085966]	Correct	16
0.38	[0.15177353 0.1831401 0.48194627 0.1831401]	[0.15177335 0.18313984 0.48194696 0.18313984]	Correct	16
0.39	[0.15403262 0.18535262 0.47526213 0.18535262]	[0.15403249 0.18535243 0.47526265 0.18535243]	Correct	16
0.4	[0.15625022 0.18750033 0.46874912 0.18750033]	[0.15625 0.1875 0.46875 0.1875]	Correct	15

Figure 3: Tune teleportation probability

Figure 3 explores tuning the teleportation probability from 0.7 to 0.99 with an increment of 0.1.

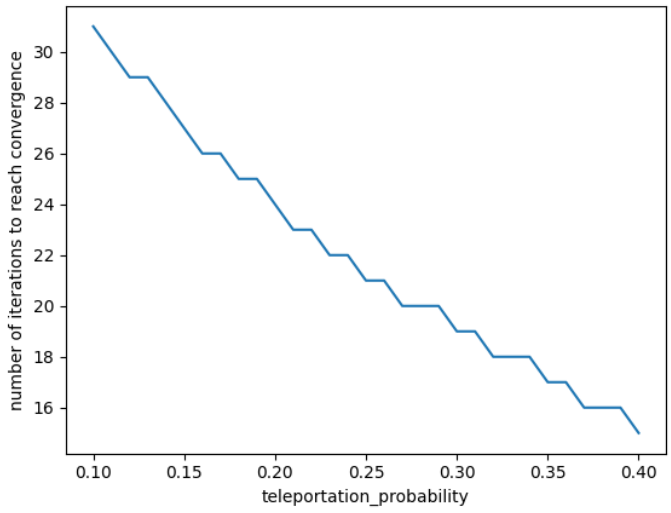

Figure 4: Results for tuning teleportation probability

Figure 4 shows the results for tuning the teleportation probability. As the teleportation probability increases, the number of iterations required for convergence decreases. Two methods are used to retrieve the page rank, the power iteration method as well as the closed form method. A sanity check is done to ensure correctness of the page rank algorithm.

Web Graph Matrix

The web graph matrix, often denoted as M, represents the connectivity between nodes in the graph and directly influences the transition probabilities in the PageRank algorithm. For this parameter, I explored different number of nodes and dense vs sparse graph.

Number of nodes	Sanity Check	Iteration	Time Taken
100	Correct	9	0.043009281158447266
200	Correct	7	0.038008928298950195
300	Correct	7	0.0800178050994873
400	Correct	7	0.1470344066619873
500	Correct	6	0.22005248069763184
600	Correct	6	0.342073917388916
700	Correct	6	0.43809962272644043
800	Correct	6	0.5851335525512695
900	Correct	6	0.726168155670166
1000	Correct	6	0.9232058525085449
1100	Correct	6	1.0912458896636963
1200	Correct	6	1.2482821941375732
1300	Correct	5	1.478334665298462
1400	Correct	5	1.7423954010009766
1500	Correct	5	2.0114548206329346
1600	Correct	5	2.31052303314209
1700	Correct	5	2.645599126815796
1800	Correct	5	2.9256625175476074
1900	Correct	5	3.2247304916381836

Figure 5: Dense Graph

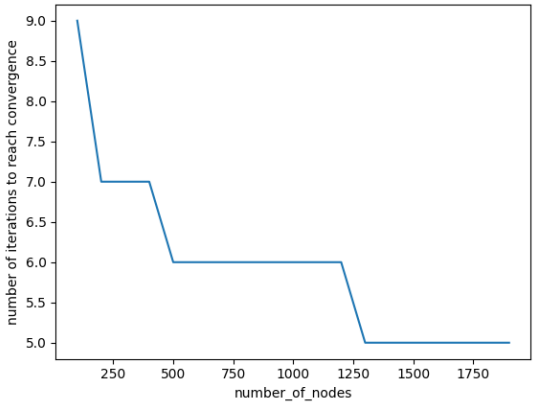

Figure 6: Results for dense graph, number of iterations vs number of nodes

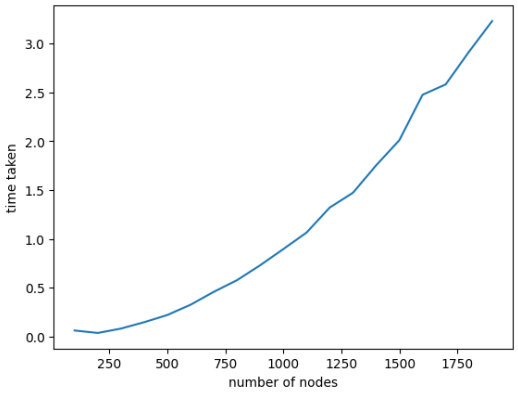

Figure 7: Results for dense graph, number of iterations vs time take

Number of nodes	Sanity Check	Iteration	Time Taken
100	Correct	16	0.16203689575195312
200	Correct	13	0.08201861381530762
300	Correct	11	0.027005910873413086
400	Correct	10	0.04701113700866699
500	Correct	10	0.07401800155639648
600	Correct	9	0.1310288906097412
700	Correct	9	0.15903687477111816
800	Correct	9	0.20804858207702637
900	Correct	9	0.24406075477600098
1000	Correct	8	0.29205799102783203
1100	Correct	8	0.33107566833496094
1200	Correct	8	0.3890876770019531
1300	Correct	8	0.44410133361816406
1400	Correct	8	0.5391209125518799
1500	Correct	8	0.6691515445709229
1600	Correct	8	0.7291650772094727
1700	Correct	8	0.7851769924163818
1800	Correct	7	0.8797533512115479
1900	Correct	7	0.9802224636077881

Figure 8: Sparse Graph

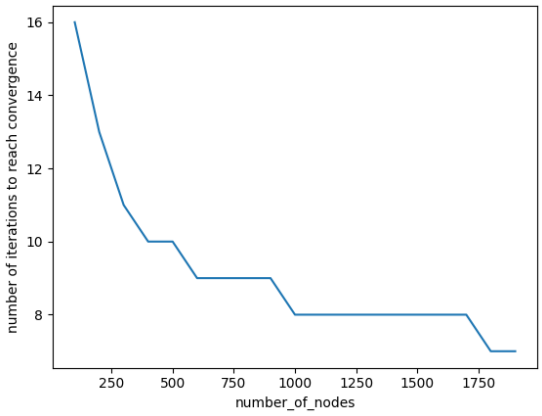

Figure 9: Results for dense graph, number of iterations vs number of nodes

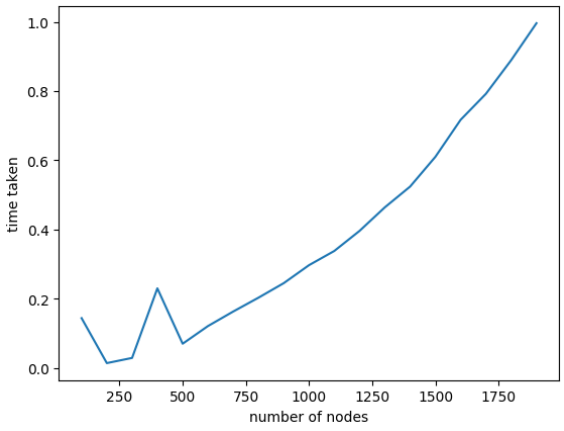

Figure 10: Results for sparse graph, number of iterations vs time taken

As seen from the Figure 6 and 9, as the number of nodes increases, the number of iterations to convergence decreases. When number of nodes is 100, it took 9 iterations for the dense graph to converge and it took 16 iterations for sparse graph.

From Figure 7 and 10, as the number of nodes increases, the time taken for convergence increases. When number of nodes is 1900, it took 3.22 seconds for the page rank to converge for dense graph and it took 0.98 seconds for sparse graph to converge.

Very Large Graph

```
G = nx.read_edgelist('web-Google.txt', create_using = nx.DiGraph)
print('Number of nodes', len(G.nodes))
print('Number of edges', len(G.edges))
print('Average degree', sum(dict(G.degree).values()) / len(G.nodes))
```

✓ 16.9s

```
Number of nodes 875713
Number of edges 5105039
Average degree 11.659160021605253
```

```
MemoryError: Unable to allocate 5.58 TiB for an array with shape (875713, 875713) and data type float64
```

For a very large graph used, I faced memory error and was unable to calculate the page rank.

Distribution Vector

Distribution Vector	Sanity Check	Iteration
[0.01713829 0.05624961 0.14358557 0.14799177 0.00865597 0.03874657 0.17195557 0.18384356 0.09351719 0.13831591]	Correct	13
[0.15731005 0.02753179 0.05010627 0.20503824 0.09587277 0.03711136 0.01621054 0.19389566 0.10304761 0.11387571]	Correct	13
[0.1104326 0.03765219 0.17562758 0.00231416 0.12060631 0.07832691 0.22956244 0.03272258 0.13059728 0.08215794]	Correct	14
[0.04617585 0.08696102 0.15620661 0.01563001 0.11278305 0.13087788 0.15980686 0.05842202 0.10286212 0.1302746]	Correct	13
[0.17532187 0.03786968 0.15862419 0.03189502 0.13298586 0.05707289 0.11718028 0.03245135 0.10625518 0.15034368]	Correct	14
[0.16837273 0.10987462 0.14572746 0.10441668 0.04929462 0.01307927 0.04511052 0.16382961 0.11874563 0.08154887]	Correct	13
[0.08537639 0.07467892 0.03050189 0.01547566 0.1969351 0.20604082 0.16151913 0.05020488 0.09171709 0.08755011]	Correct	13
[0.18505899 0.02582256 0.1174784 0.11869362 0.01441506 0.14427101 0.13090158 0.05943224 0.09439528 0.10953125]	Correct	14
[0.16905643 0.07379228 0.17298734 0.13687469 0.02346663 0.07986002 0.19185759 0.03286483 0.05746638 0.06177381]	Correct	14
[0.04474534 0.10953785 0.15173463 0.1581375 0.13850673 0.04764261 0.1195722 0.08366767 0.07740643 0.06904903]	Correct	13

Figure 11: Tuning Distribution Vector

As the distribution vector should be decided based on specific goals of the PageRank analysis, and any domain-specific considerations. I randomized the distribution vector and the number of iterations it took for the graph to converge stays similar.

Summary of parameters tuning

In summary, the teleportation probability in the PageRank algorithm balances the trade-off between exploration and exploitation. Higher teleportation probabilities promote exploration, leading to faster convergence but potentially less accurate rankings, while lower teleportation probabilities prioritize exploitation, resulting in more accurate rankings but potentially slower convergence. The choice of teleportation probability depends on factors such as the size and structure of the graph, the desired level of randomness, and the trade-off between convergence speed and ranking accuracy.

In a sparse graph where each node has relatively few outgoing links, the web graph matrix will have many zero entries. This sparsity can slow down the convergence of the PageRank algorithm because random walks may encounter dead ends or take longer paths to explore the entire graph. Conversely, in a dense graph where each node has many outgoing links, the web graph matrix will have fewer zero

entries. This denseness can speed up the convergence of the PageRank algorithm because random walks have more opportunities to explore different paths and distribute PageRank scores more evenly.

The distribution vector determines the initial probability distribution of the random surfer when starting the random walk. Nodes with higher probabilities in the distribution vector are more likely to be visited early in the random walk process. The distribution vector can influence the convergence speed of the PageRank algorithm. A distribution vector that accurately reflects the underlying importance or relevance of nodes in the graph may lead to faster convergence. However, as the graphs used are not domain-specific, the importance of the nodes are not determined for this experiment.

MapReduce

Spark can be used for the implementation of the Map Reduce framework.

Implementing PageRank in a parallel programming framework like MapReduce/Hadoop involves dividing the PageRank algorithm into Map and Reduce tasks that can be executed in parallel across multiple compute nodes. The page rank algorithm can be split into 5 phases.

1. Map phase: Each Map task processes graph data, emitting key-value pairs for each node and its outgoing links.
2. Shuffle and sort phase: Intermediate key-value pairs from Map tasks are shuffled and sorted based on node IDs.
3. Reduce Phase: For each node i, sum the upcoming votes and update Rank value (***R***i).
4. Iteration phase: The MapReduce framework iterates Map and Reduce tasks for a fixed number of iterations or until convergence criteria are met.
5. Termination phase: The MapReduce job terminates when the specified number of iterations is reached or convergence criteria are satisfied. Final PageRank scores are output as the result of the MapReduce job.

Pseudocode:

```
Map Phase:

function Map(node_id, adjacency_list):

    // Emit contributions to PageRank scores for each node

    emit(node_id, 1 / num_nodes)  // Initialize PageRank score for current node


    // Emit contributions from outgoing links

    for each link in adjacency_list:

        emit(link, PageRank_score / num_outgoing_links)


Reduce Phase:

function Reduce(node_id, contributions):

    // Aggregate contributions from all incoming links

    PageRank_score = 0

    for each contribution in contributions:

        PageRank_score += contribution


    // Apply PageRank formula to update node's score
```

```
    new_PageRank_score = damping_factor * (PageRank_score) + (1 - damping_factor) /
num_nodes


    emit(node_id, new_PageRank_score)


Main:
// Initialize PageRank scores for all nodes
for each node in graph:
    emit(node_id, 1 / num_nodes)


// Perform multiple iterations or until convergence
for iteration in range(max_iterations):
    // Map phase: compute contributions to PageRank scores
    map_output = Map(graph_node, adjacency_list)


    // Shuffle and sort intermediate key-value pairs


    // Reduce phase: aggregate contributions and update PageRank scores
    reduce_output = Reduce(node_id, map_output)


// Output final PageRank scores
output_final_PageRank_scores()
```

By distributing the computation of PageRank across multiple Map and Reduce tasks running in parallel, the MapReduce framework enables efficient processing of large-scale graphs. The framework handles data distribution, task scheduling, fault tolerance, and scalability, making it well-suited for implementing PageRank and other graph algorithms in distributed environments.