California State University, Chico

Advanced Algorithms

(CSCI 411)

PageRank Implementation

Author:

Anna Sehgal

Professor:

Professor Tillquist

Abstract

In this report, we will focus on studying the PageRank (PR) algorithm developed by Google, focusing on its intuition, implementation through pseudocode, and runtime characteristics. The main objective is to develop an understanding of the algorithm: how it works, run time efficiency, and interpretation of its results. This report presents detailed pseudocode and examples to exhibit the behavior of PageRank. Furthermore, the algorithm includes a runtime analysis, highlighting factors that influence the algorithm's performance and convergence.

Contents

1 Introduction **3**
1.1 Background and History . . . 3
1.2 Purpose of the Algorithm . . . 3
1.3 Application . . . 3
1.4 Importance . . . 3

2 Intuition **3**
2.1 Example illustration of intution: . . . 4

3 Pseudocode and Detailed Description **5**
3.1 Formal Pseudocode of algorithm . . . 5
3.1.1 Node Class: . . . 5
3.1.2 Graph Class: . . . 6
3.1.3 PageRank algorithm: . . . 7
3.1.4 Util (Build and Print): . . . 7
3.1.5 Main: . . . 8
3.2 PageRank (pseudocode in general) - from my presentation: . . . 8
3.3 Detailed Description . . . 8
3.4 Expectations: Inputs and Outputs . . . 8
3.5 Justification . . . 9

4 Run-Time Analysis **9**
4.1 Run-Time Analysis . . . 9

1 Introduction

1.1 Background and History

The PageRank algorithm was developed in the mid-1990s by two Ph.D students at Stanford University, Larry Page and Sergey Brin, to create better search results to solve a major problem during that time, as the internet grew exponentially and search engines that time heavily relied on keyword matching and simple heuristics to rank web pages online, resulting in information overload leading to irrelevant search results, easily exploitable through spamming and low-quality user experience. [1]

The success of the PageRank algorithm was due to the technique for measuring the rank of web pages, and determining their importance in terms of the importance assigned to the pages hyperlinking to that particular page, linking back to the idea of how academic papers gain credibility through the number of citations. This concept produced better results and became the foundation of Google's revolutionary search engine in September 1998.

1.2 Purpose of the Algorithm

Pagerank assigns a numerical weight (score) to each page, measuring the importance of that web's hyperlink structure, modeling the web as a directed graph, where each node represents a web page, and edges represent hyperlinks between them. A page's importance is determined by two factors: the quantity and quality of the pages linking to it, which extends beyond the idea of the number of incoming links to the page used in academic citations, and focuses on the idea that not all pages hold the same weight. [2]

The algorithm utilizes the "random surfer" model, a hypothetical user navigating the web by randomly following links. The algorithm is computed iteratively by assigning ranks to pages based on the linked-to pages until the rank converges. The pages receiving important incoming pages are ranked higher, identifying the most important pages in the directed web-graph.

1.3 Application

PageRank can be used as a measure of influence in various areas beyond its usage in web page ranking. The idea behind this is effective in recommendation systems for identifying trending products or content that matches users' preferences. In data analysis, it can help find high-impact nodes in data lineage or detect users involved in fraudulent transactions. Additionally, it is beneficial for managing permissions for sensitive data, network optimization by identifying nodes with higher chances of failure, and calculating probabilities of certain malignant events causing severe attacks in cyber security. [3] Furthermore, it can be used in ranking research papers, authors, and journals. Future applications may include ranking nodes in a multi-agent system or machine learning models.

1.4 Importance

PageRank is important as it offers a consistent methodology for measuring the significance of each node in an extensive network structure. The concept introduced an iterative method of gauging information and analysis through a graphical model that can scale for large data sets, and has impacted many current algorithms used in the areas of information retrieval, social network analysis, machine learning, citation network, and spamming detection. Understanding the intuition behind this algorithm helps in identifying important nodes, improving the relevance of searches, and providing a means to evaluate and understand complex systems more efficiently.

2 Intuition

The main intuition behind PageRank is through the estimation of the importance of web pages based on the random surfer model, for example, a user is moving through page to page across the web, starting from a random page. At each step, the surfer either follows a random outgoing link from that current page to move to another page or teleports to a completely random page. A page is considered "important" if the surfer has a higher probability of landing on it frequently during infinite random surfing, which is calculated through this algorithm by calculating the probability of the surfer landing on each page, determining importance based on the probability score. When a page has many outlinks, the probability of the surfer following any specific link decreases. Pages with no outgoing or no incoming links allow the surfer jump to a random page with a certain probability. A link from a highly important page passes on more influence than a link from a low-quality page. Links are treated as weighted recommendations, capturing both the quantity and quality of incoming links. This repeats iteratively until convergence and reflects the global importance of that node in the network of the graph. In other words, the intuition works as follows, step-by-step:

1. Initially, every page is assigned the same rank $1/N$ ($N =$ total number of pages), meaning that all pages are important first. [4]

2. The PageRank of each page is updated based on the PageRank of the pages linking to it. Each page passes a portion of its rank to the pages it links to. Pages with more outgoing links divide their ranks among more pages, so each linked page gets an equal portion of that particular page rank.

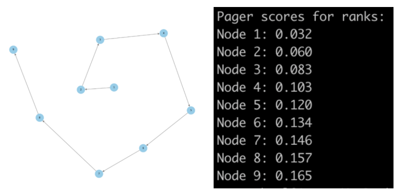
3. The algorithm deals with the teleportation factor for pages with no links, or to deal with random jumps through an idea called the damping factor, which is usually set to 0.85, a value estimated from the frequency with which an average web user uses their browser's bookmark feature. [5] In simple words, it means that 0.85 is the ratio of users being on the same link, and the other 0.15 is of teleportation to a random page. This damping factor allows every page to receive some rank, even if it has no links coming towards it, especially useful in disconnected graphs.

4. Over each iteration, the rank is recalculated for all pages using incoming links and teleportation, and this repeats until the rank stabilizes (i.e., converges), meaning that it becomes very small, indicating stabilization.

5. After that, pages with higher probability/scores are more important. This means ranks reflect the overall structure of the network and the influence of each node.

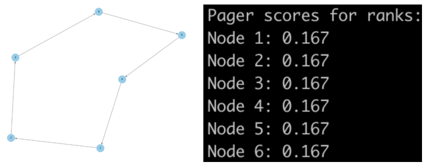
Therefore, PageRank ranks webpages by simulating a user surfing the web by assessing the page's importance from the perspective of linking pages. Through this iterative simulation, the algorithm determines the most influential nodes within the web-network.

2.1 Example illustration of intution:



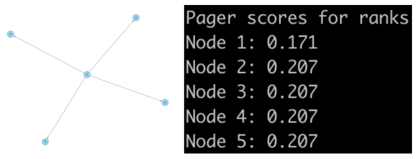
```
Pager scores for ranks:
Node 1: 0.032
Node 2: 0.060
Node 3: 0.083
Node 4: 0.103
Node 5: 0.120
Node 6: 0.134
Node 7: 0.146
Node 8: 0.157
Node 9: 0.165
```

Figure 1: Linear Chain: When using a linear chain of nodes, every node points to its successor. Though there is no incoming link pointing to Node 1, there are incoming links pointing to Node 9 from all of its predecessor nodes. The PageRank of the nodes in a linear chain will increase from the 1st Node toward Node 9; thus, Node 9 will be ranked as the highest PageRank node because it has accumulated PageRank from all other nodes before it. A node such as Node 1, that does not have any incoming links will have a small PageRank due to the presence of the Damping Factor, which allows the possibility of random jumps. This type of example demonstrates how PageRank spreads importance through a linear link structure and demonstrates how the Damping Factor affects both the link structure and the final PageRank values of the nodes in a linear structure.



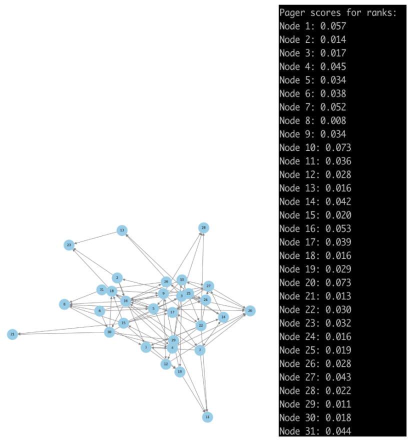
```
Pager scores for ranks:
Node 1: 0.167
Node 2: 0.167
Node 3: 0.167
Node 4: 0.167
Node 5: 0.167
Node 6: 0.167
```

Figure 2: Small Directed Cycle: A directed cycle is created when a node has only one other node to point towards, returning to the node from which it began, in this example. Because each node has both an incoming and an outgoing link, all nodes in the cycle receive equal amounts from their neighbours. This makes it easy to see that the PageRank of each node will eventually reach approximately the same number because of this. Since the cycle is symmetric and balanced, it is a clear example of how the algorithm uniformly distributes the importance across all nodes in the cycle. The damping factor is used to ensure that when the PageRank for all nodes in a cycle are summed together, they always equal 1, while the random jump portion ensures that all nodes retain a non-zero PageRank value.



```
Pager scores for ranks
Node 1: 0.171
Node 2: 0.207
Node 3: 0.207
Node 4: 0.207
Node 5: 0.207
```

Figure 3: Star Hub Structure: When you look at the structure of this hub-and-spoke model, you will see that the central hub (in this case, node 1) has outgoing edges to nodes 2, 3, 4, and 5. Node 1 does not receive any incoming links from any of these other nodes, resulting in a comparatively low PageRank score for node 1 because it does not receive any rank contributions from other nodes. Conversely, nodes 2, 3, 4, and 5 receive all or most of their PageRank score from node 1, thereby providing a similar PageRank score for each of them. Although node 1 does not have any incoming links, it will continue to have a small (but greater than 0) PageRank score due to the damping factor of 0.85 (on average), and nodes 2-5 receive a similar PageRank score from node 1. Thus, while there is a significant contribution of rank from the central hub (node 1), there is a random teleportation component also accounted for in the PageRank algorithm.



```
Pager scores for ranks:
Node 1: 0.057
Node 2: 0.014
Node 3: 0.017
Node 4: 0.045
Node 5: 0.034
Node 6: 0.038
Node 7: 0.052
Node 8: 0.008
Node 9: 0.034
Node 10: 0.073
Node 11: 0.036
Node 12: 0.028
Node 13: 0.016
Node 14: 0.042
Node 15: 0.020
Node 16: 0.053
Node 17: 0.039
Node 18: 0.016
Node 19: 0.029
Node 20: 0.073
Node 21: 0.013
Node 22: 0.030
Node 23: 0.032
Node 24: 0.016
Node 25: 0.019
Node 26: 0.028
Node 27: 0.043
Node 28: 0.022
Node 29: 0.011
Node 30: 0.018
Node 31: 0.044
```

Figure 4: Random Sparse Graph: The PageRank scores for nodes that have many incoming links to them is greater than other nodes (those with many incoming links will also have a higher PageRank). Nodes that have no or fewer than two incoming links tend to have low PageRank scores. All nodes still get some PageRank, however, even those that are not connected to any other nodes through an edge, because they are receiving a small, but non-zero PageRank value from the random teleportation aspect of the algorithm. This shows how the PageRank algorithm allows for fairness in assigning PageRank scores to every node, while balancing the structure of the graph and also including a component that allows for the random teleportation of nodes.

3 Pseudocode and Detailed Description

3.1 Formal Pseudocode of algorithm

3.1.1 Node Class:

```
class Node
    # each webpage has:
    # id: unique identifier
    # scorenow: current pagerank
    # scorenext: next iteration pagerank
    # outarrow: list of outgoing links
    # inarrow: list of incoming links

    function constructor(id)
        self.id = id
        self.scorenow = 1.0
        self.scorenext = 0.0
        self.outarrow = [ ]
        self.inarrow = [ ]

    function nodeID()
```

```
        return self.id

    function ranknow()
        return self.scorenow

    function ranknext()
        return self.scorenext

    function nodeout()
        return self.outarrow

    function nodein()
        return self.inarrow

    function arrowoutside(v)
        # add v to outgoing list
        self.outarrow.push_back(v)

    function arrowin(u)
        # add u to incoming list
        self.inarrow.push_back(u)

    function setterrank(score)
        self.scorenow = score

    function setternextrank(score)
        self.scorenext = score

    function addnextiterationrank()
        # move next rank to current rank
        self.scorenow = self.scorenext

    function resetaddnextscore()
        # reset temporary accumulator
        self.scorenext = 0.0
```

3.1.2 Graph Class:

```
class Graph
    # webpages: list of Node objects

    function constructor()
        webpages = [ ]

    function insertnode(id)
        # add a new Node to graph
        webpages.push_back(new Node(id))

    function accessnode(id)
        # return node whose ID matches
        for each pg in webpages
            if pg.nodeID() == id
                return pg
        return null

    function noofnodes()
        return size of webpages

    function clear()
        webpages = [ ]

    function insertedge(start, end)
        # add directed edge start      end
        s = accessnode(start)
        e = accessnode(end)

        if s != null and e != null
            outlist = s.nodeout()
            exists = false
            for each x in outlist
                if x == end
                    exists = true
            if not exists
                s.arrowoutside(end)
                e.arrowin(start)

    function findedge()
        return webpages

    function noofedges()
        total = 0
        for each pg in webpages
            total = total + size of pg.
               nodeout()
        return total

    function out(pageid)
        # return unique outgoing neighbors
        node = accessnode(pageid)
        if node == null
            return [ ]

        outu = [ ]
        for each val in node.nodeout()
            if val not in outu
                outu.push_back(val)
        return outu

    function in(pageid)
        # return unique incoming neighbors
        node = accessnode(pageid)
        if node == null
            return [ ]

        inu = [ ]
        for each n in node.nodein()
            if n not in inu
            inu.push_back(n)
        return inu
```

3.1.3 PageRank algorithm:

```
class Pagerank
    # dampingfactor: d
    # e: convergence threshold

    function constructor(damp, eps,
       maxIter)
        dampingfactor = damp
        e = eps
        maxit = maxIter

    function simulate(gr)
        n = gr.noofnodes()
        if n == 0
            return

        pages = gr.findedge()

        # initialize all ranks to 1/n
        for each pg in pages
            pg.setterrank(1 / n)
            pg.setternextrank(0)

        iteration = 0

        while iteration < maxit
            converged = true

            # compute sum of dangling
               nodes
            dsum = 0
            for each pg in pages
                if pg.nodeout() is empty
                    dsum = dsum + pg.
                        ranknow()

            # compute next rank for each
               node
            for each node in pages
                nextr = (1 - dampingfactor
                   ) / n
                nextr = nextr + (
                   dampingfactor * dsum /
                   n)

                inNodes = node.nodein()
                for each u in inNodes
                    inNode = gr.accessnode
                        (u)
                    outDegree = size of
                       inNode.nodeout()

                    if outDegree > 0
                        nextr = nextr + (
                           dampingfactor *
                            inNode.ranknow
                           () / outDegree)

                node.setternextrank(nextr)

            # update ranks and check
               convergence
            for each node in pages
                oldRank = node.ranknow()
                newRank = node.ranknext()

                if abs(newRank - oldRank)
                   > e
                    converged = false

                node.addnextiterationrank
                   ()
                node.resetaddnextscore()

            if converged
                break

            iteration = iteration + 1

        # normalize ranks to sum to 1
        total = 0
        for each node in pages
            total = total + node.ranknow()

        if total > 0
            for each node in pages
                node.setterrank(node.
                   ranknow() / total)
```

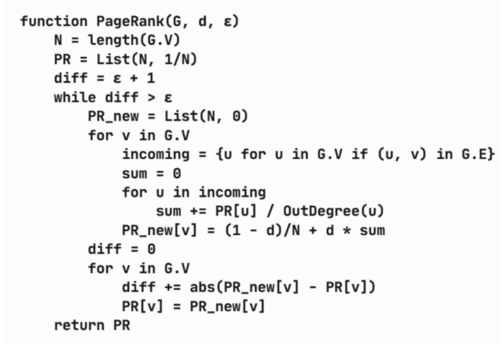
3.1.4 Util (Build and Print):

```
class Util

    function builder(gr)
        # read number of nodes
        read nodes

        # add nodes labeled 1..nodes
        for i from 1 to nodes
            gr.insertnode(i)

        # read number of edges
        read edges

        # read and insert edges
        for i from 1 to edges
            read u, v
            gr.insertedge(u, v)

    function ranker(gr)
        pages = gr.findedge()
        for each pg in pages
            # formatted printing (rounded)
            output "Node", pg.nodeID(), ":
               ␣", round(pg.ranknow(), 3)
```

3.1.5 Main:

```
function main()
    gr = new Graph()

    Util.builder(gr)

    pr = new Pagerank(0.85, 1e-6, 100)

    pr.simulate(gr)

    Util.ranker(gr)

    return 0
```

3.2 PageRank (pseudocode in general) - from my presentation:



```
function PageRank(G, d, ε)
    N = length(G.V)
    PR = List(N, 1/N)
    diff = ε + 1
    while diff > ε
        PR_new = List(N, 0)
        for v in G.V
            incoming = {u for u in G.V if (u, v) in G.E}
            sum = 0
            for u in incoming
                sum += PR[u] / OutDegree(u)
            PR_new[v] = (1 - d)/N + d * sum
        diff = 0
        for v in G.V
            diff += abs(PR_new[v] - PR[v])
            PR[v] = PR_new[v]
    return PR
```

Figure 5: PagerRank pseudocode

3.3 Detailed Description

The `Node` class is a representation of a single webpage on the internet; it holds the current and the next iteration of the PageRank score, incoming and outgoing links. The `Graph` class manages all the nodes and edges, and provides access to unique incoming or outgoing neighbors. The `Pagerank` class updates PageRank for each `Node` iteratively, considering contributions from incoming links, dangling nodes, and teleportation through a damping factor set to 0.85. The `Util` class handles input and output, building the graph, and printing final ranks. The algorithm keeps iterating until ranks converge (i.e., changes are below the threshold ε) and then normalizes the scores to sum to 1.

In particular, the `Pagerank` class does the calculations for the PageRank score by:

1. Initializing all pages with rank $1/N$, where N is the total number of pages.

2. On an iterative basis, computing the next rank of each page by calculating contributions from incoming pages and the damping factor (0.85), including a correction for dangling nodes (pages with no outgoing links).

3. Utilizing an epsilon threshold ε to determine convergence and repeating until pages become stable or until the maximum number of iterations has been reached.

4. Finally, normalizing the final rank for each page so that the total rank sum is equal to 1.

3.4 Expectations: Inputs and Outputs

Inputs:

- Number of nodes (webpages) = N: the total number of webpages in the graph.

- List of directed edges $u \to v$: representing hyperlinks from node u to node v.

- Damping factor d (usually set to 0.85 in code): the probability that the random surfer follows a link.

- Convergence threshold ε: a small value to determine when PageRank has stabilized.

- Maximum iterations (`maxIter`): the limit to prevent infinite loops if convergence is slow.

Outputs:

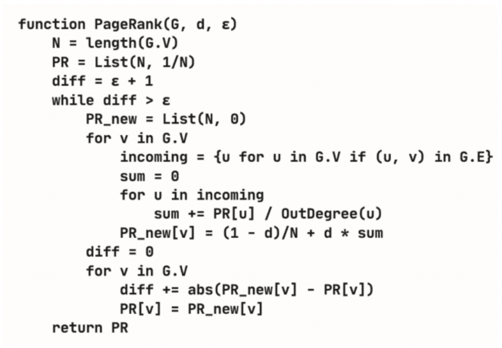
- PageRank of each node: a probability between 0 and 1 indicating the page's importance based on the score.
- Normalized PageRank: the sum of all PageRank scores equals 1.

3.5 Justification

The PageRank algorithm operates as intended due to the way it updates each individual node's page rankings through incremental steps, taking into account both the outgoing and incoming links from each node and where they will direct visitors (i.e, incoming nodes). It provides a means of effectively calculating the ranking of pages with no incoming links through the use of the damping factor, allowing for an accurate representation of a random surfer model. The convergence of the ranking from our calculations is guaranteed since the iterative updates can be modelled as random processes on a finite-sized graph with a normalization factor used each iteration. The point at which we stop iterating occurs when the amount of difference in rankings falls below a defined threshold ε or a maximum number of iterations has been performed. Thus, this produces a stable ranking where total PageRank sums to 1, and represents how important each page is, based on its relationship to all other pages in the network, following the textbook approach for PageRank. [5]

4 Run-Time Analysis

4.1 Run-Time Analysis



```
function PageRank(G, d, ε)
    N = length(G.V)
    PR = List(N, 1/N)
    diff = ε + 1
    while diff > ε
        PR_new = List(N, 0)
        for v in G.V
            incoming = {u for u in G.V if (u, v) in G.E}
            sum = 0
            for u in incoming
                sum += PR[u] / OutDegree(u)
            PR_new[v] = (1 - d)/N + d * sum
        diff = 0
        for v in G.V
            diff += abs(PR_new[v] - PR[v])
            PR[v] = PR_new[v]
    return PR
```

Figure 6: PagerRank pseudocode

Let

$$N = \text{number of pages (nodes)}$$

$$E = \text{number of links in the graph}$$

1) Initialization step:

- Allocate arrays for current and next PageRank: $O(N)$
- Initialize all PageRank values to $1/N$: $O(N)$
- Allocate an empty adjacency list for each node: $O(N)$

Total step:

- Time Complexity = $O(N)$
- Space Complexity = $O(N)$

2) Graph Construction step:

- Reading all input lines: $O(N + E)$
- Adding N nodes to the graph: $O(N)$
- Adding E edges to the adjacency list (O(1) insertion): $O(E)$

Total step: Time complexity = $O(N + E)$

3) PageRank iterations (until convergence or maxIter reached). Each iteration:

1. Dangling Nodes (nodes with no outgoing edges) feed their ranks back into the system: $O(N)$
2. Rank update from incoming edges: visiting every edge once over the graph: $O(E)$
3. Apply damping factor and calculate new rank: $O(N)$
4. Check convergence by comparing old and new rank values: $O(N)$

Total cost per iteration: $O(N + E)$
Total PageRank runtime: $O(\text{maxIter} \cdot (N + E))$

4) Normalization step:

- Normalize the rank vector so that the ranks sum to 1: $O(N)$

5) Output step:

- Printing ranks: $O(N)$

Overall Runtime:

- Initialization: $O(N)$
- Graph construction: $O(N + E)$
- PageRank iterations: $O(\text{maxIter} \cdot (N + E))$
- Normalization: $O(N)$
- Output: $O(N)$

Final Overall Complexity:

$$\text{Time Complexity} = O(\text{maxIter}\cdot(N+E)), \quad \text{Space Complex}$$

References

[1] Damilola, “A brief history of the PageRank algorithm – Computing for All,” *Computing for All*, Aug. 6, 2025. https://computing4all.com/courses/introductory-data-science/lessons/a-brief-history-of-the-pagerank-algorithm/

[2] “PageRank Algorithm - an overview,” *ScienceDirect Topics*. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/pagerank-algorithm

[3] “PageRank Algorithm for graph databases,” *Memgraph Blog*. [Online]. Available: https://memgraph.com/blog/pagerank-algorithm-for-graph-databases

[4] A. Yadav, “PageRank algorithm explained,” *PageRank Algorithm Explained*, Medium. [Online]. Available: https://medium.com/biased-algorithms/pagerank-algorithm-explained-5f5c6a8c6696. Accessed: Dec. 2025.

[5] Wikipedia contributors, “PageRank,” *Wikipedia*, Nov. 24, 2025. [Online]. Available: https://en.wikipedia.org/wiki/PageRank