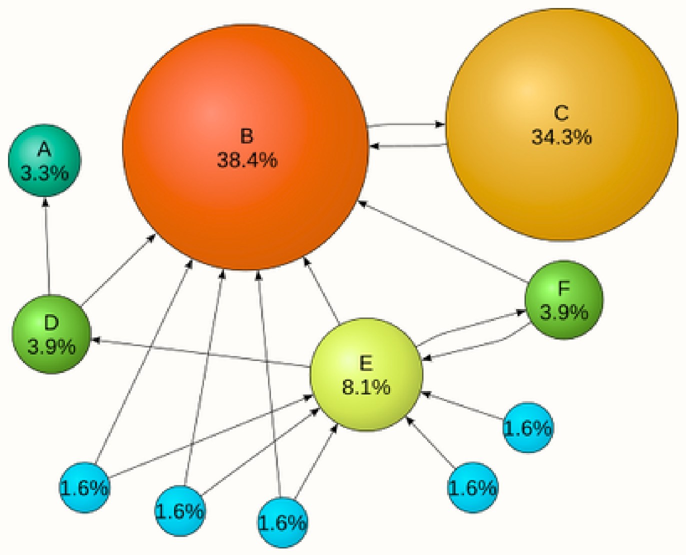


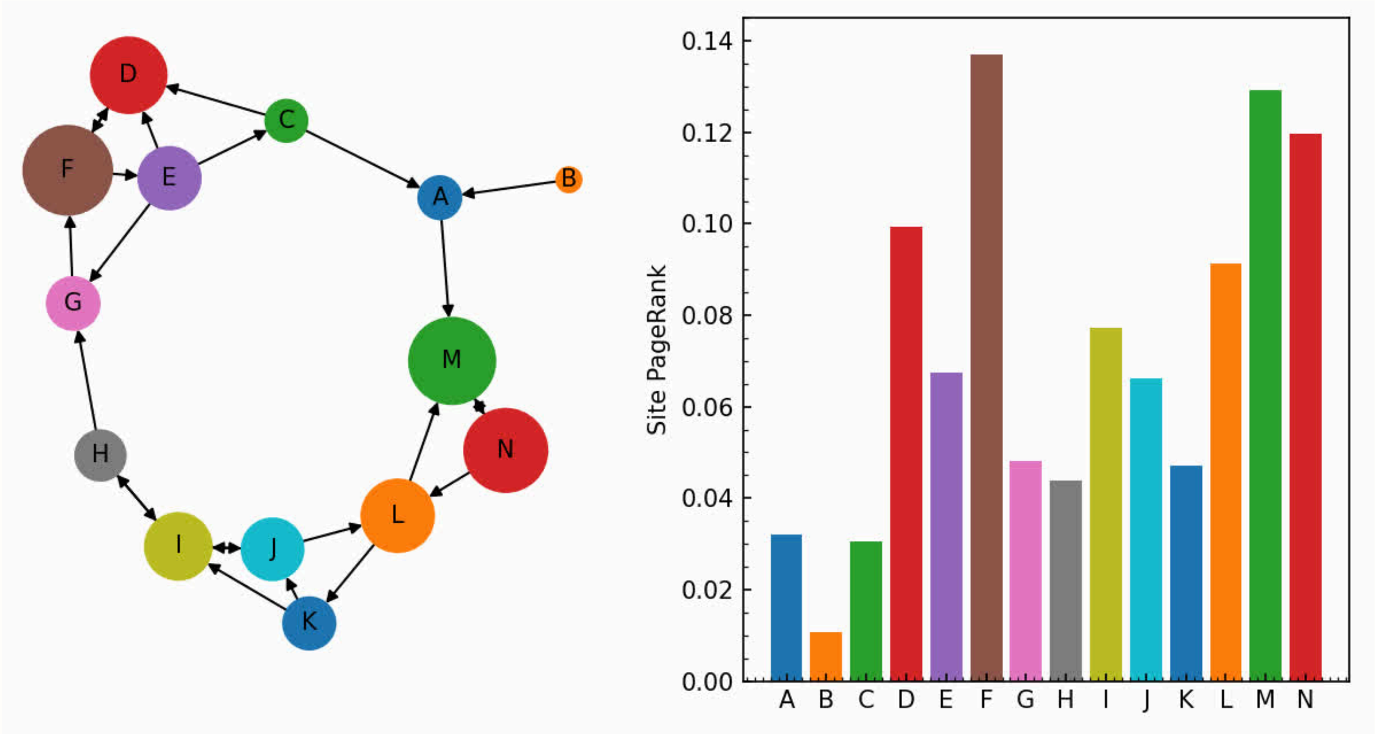
PagerRank:
The Billion Dollar Algorithm

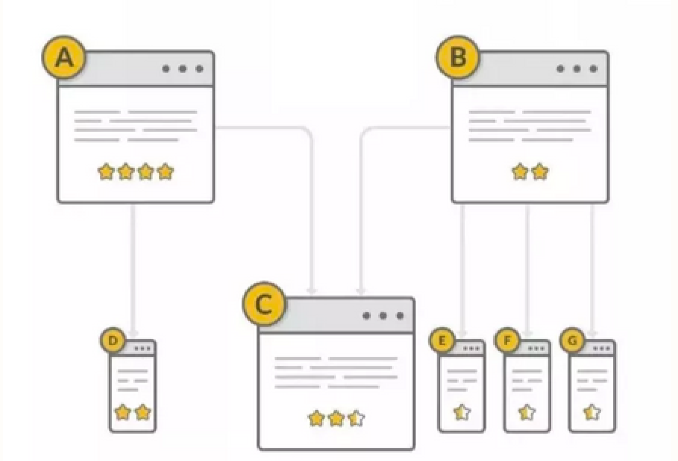
By Anna Sehgal

Table of contents

What is PagerRank?

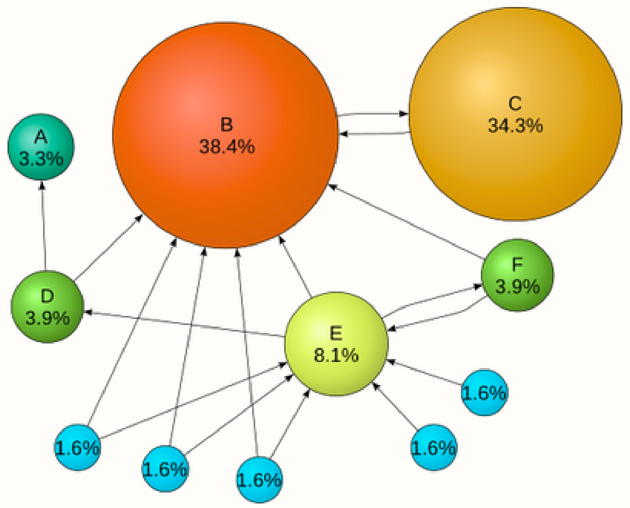
- Created in 1996 at Stanford by Larry Page and Sergey Brin.
- Inspired by how academic papers cite each other.
- Became the core of Google's search engine in 1998.
- Early search engines relied on keyword matching, which was easily exploited for spamming.
- Page & Brin asked: "Can we rank pages by importance using links?"
- PageRank models hyperlinks as weighted recommendations, where recommendations from influential pages contribute more to a page's importance.

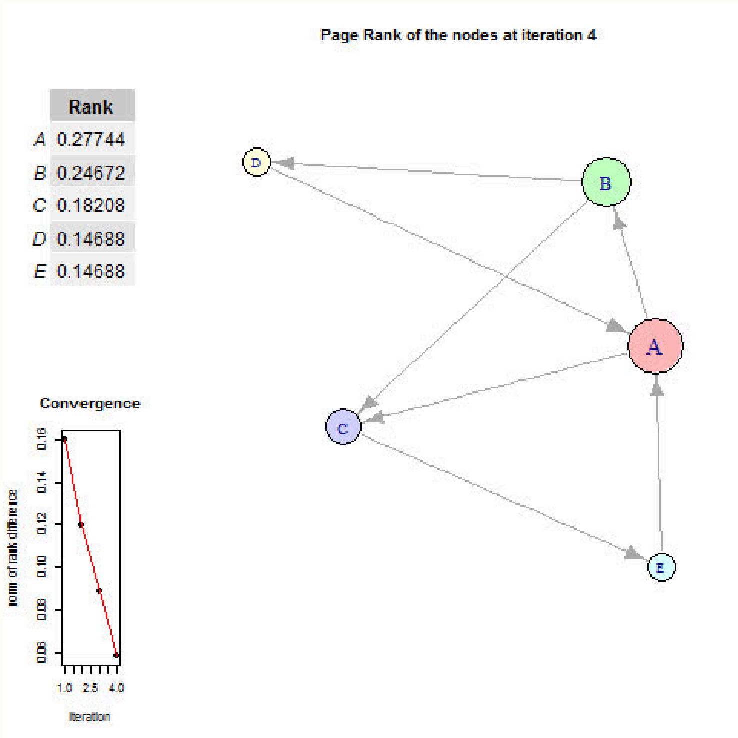


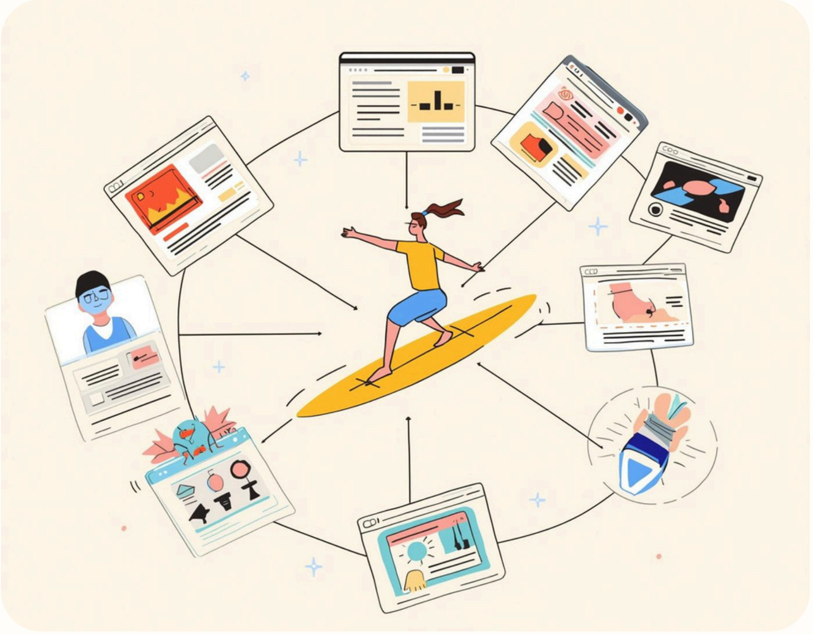


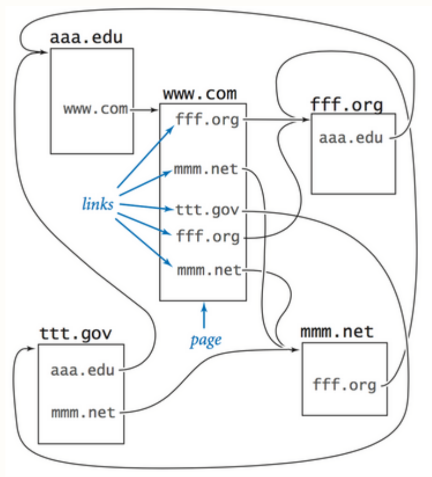
Intuition: Random Surfer Model

- PageRank is based on the **random surfer model:** imagine a user who moves from **page to page** across the web.
- At each step, the surfer either:
  - Follows a **random outgoing link** from the current page, or
  - **Teleports to a random page** with small probability (the damping factor).
- A page is considered **"important"** if the surfer is likely to land on it frequently during infinite random navigation.
- Importance flows through links: a link from a highly important page passes on more influence than a link from a low-quality page.
- Instead of treating all links equally, PageRank interprets links as weighted recommendations, capturing both the **quantity and quality of incoming links.**
- Over repeated iterations, the process stabilizes, producing a score that reflects global influence in the entire network, not just local link counts.









~~The~~ Formula

$$PR(v) = \frac{1-d}{N} + d \sum_{u \in M(v)} \frac{PR(u)}{L(u)}$$

- **PR(v)**: PageRank score of page v, representing its importance.
- **d**: Damping factor (usually 0.85), probability the random surfer follows a link rather than jumping randomly.
- **N**: Total number of pages; (1-d)/N distributes rank evenly across all pages.
- **M(v)**: Set of pages linking to v; only these pages contribute to v's rank.
- **PR(u)**: PageRank of a page u linking to v; higher-ranked pages contribute more.
- **L(u)**: Number of outgoing links from page u; rank is divided among its links.
- **Σ (sum)** over **u ∈ M(v)**: Sum of contributions from all pages linking to v.
- This Repeat until PageRank values converge (when scores stop changing significantly).

Pseudocode

```
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
```

Run Time Analysis

Pseudocode

```
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
```

Let N = number of pages (nodes) and M = number of links (edges) in the graph.
Let k = number of iterations until PageRank values converge.

Initialization

- Assign initial PageRank values to all pages.
- Time Complexity: O(N)
- Space Complexity: O(N)

Main Iteration

- For each iteration, compute new PageRank values based on incoming links.
- Cost per iteration:
    - Loop over all pages: O(N)
    - Sum over incoming links: O(M), where M = number of edges
- Update PageRank values and calculate difference: O(N)

Total per iteration: O(N + M) ≈ O(M) for sparse graphs

Number of Iterations

- Let k = number of iterations until convergence (depends on damping factor d and threshold ε)
- Overall Time Complexity: **O(k · (N + M)) ≈ O(k · M)**
-

Space Complexity

- Storing PageRank vectors: O(N)
- Graph data: O(N + M)

Total Space Complexity: O(N + M)
Overall runtime: O(k · (N + M))

Importance & Uses

Determines importance of web pages based on link structure, not just content.

Forms the core of Google Search ranking, influencing billions of searches daily.

Useful in other domains:

- Social networks: ranking influential users or nodes.
- Citation networks: ranking influential papers or authors.
- Recommendation systems: weighting connections or relevance.

Captures global influence: a page is valuable if important pages link to it.

Helps filter spam or low-quality pages by assigning low PageRank.

Limitations

Ignores page content

only considers links, not relevance of text.

Vulnerable to link manipulation

spam farms or artificial link schemes can distort rankings.

Slow for very large graphs

convergence can require many iterations (k can be large).

Assumes static web

doesn't handle rapidly changing web content in real-time.

Equal damping factor

same teleport probability for all pages; may not reflect realistic surfer behavior.

Difficulty with dangling nodes

pages with no out-links require careful handling.

Implementation

Theme: PageRank Algorithm for Node Importance
Goal: Compute the importance of each node in a graph (webpages, entities, etc.) using iterative scoring.
Type: Iterative graph algorithm

Inputs / Features:

- **Graph:** nodes and directed edges
- **Damping factor:** probability of following links vs teleporting randomly
- **Convergence threshold:** stopping criterion for iteration
- **Maximum iterations:** upper limit to ensure termination

Implementation Approach:

- Represent graph using adjacency lists and out-degree counts
- Initialize PageRank values equally across all nodes
- Iteratively update scores based on incoming links and damping factor
- Handle dangling nodes (nodes with no outgoing edges)
- Repeat until change in scores is below the threshold
- Normalize final scores so they sum to 1
- Support CSV output and simple testing.

Thankyou!