PageRank Applications in Website Domains

Seunghyun Bae, Michael Li
Computer Science, Carnegie Mellon University

December 2025

1 Motivation

Modern organizations have content spread across multiple domains and subdomains on the internet. For example, universities maintain dozens of subdomains for departments, libraries, and research centers, while companies split content across regional sites and product domains. This presents a challenge for users that seek information within these ecosystems, and identifying which domains are most authoritative becomes essential for efficient, safe navigation.

Simple link counting (by ranking domains on the number of inbound links they receive) fails in 4 key ways.

1. **It treats all links equally, ignoring the source page's authority.** Two domains can have the same number of inbound links, but if one domain is cited by more important domains, then it should be considered more authoritative, which this method does not account for.
2. **It misses transitive importance.** If Domain A links heavily to Domain B, and Domain B links to Domain C, then C should inherit importance through this chain, but this multi-step relationship is invisible to direct counting.
3. **It provides no option for biasing results toward a specific organization's ecosystem.** When exploring domains related to a seed organization, how centered the domains are to the seed and whether they are subdomains or external domains should be a factor taken into account. For example, a social media domain may have multiple links from multiple domains connected to the seed, but since it is not centered and is an external domain, its importance should be dampened.
4. **It disregards the importance of duplicate links on a single page.** When a page on Domain A links to a page on Domain B more than once, that emphasis is intentional, and the weight should be accounted for.

The first two problems can be tackled using Google's PageRank algorithm on domains, but to address the remaining problems requires a slightly modified version of the Personalized PageRank algorithm. Our paper proposes a new variant of this algorithm, built from PageRank and linear algebra, to provide a novel solution for ranking domain relevance to a seed.

2 Introduction

Let's begin by breaking down Google's PageRank algorithm to build intuition.

Consider a graph with vertices representing websites and directed edges representing the connectivity of the different websites.

Our goal is to compute a ranking of the websites that gives a rough idea of how important each website is. We are doing this under the assumption that pages that are cited by other important pages should be considered important (problem 1 from earlier).

Let us define the ranking of each vertex i by r_i such that

$$r_i = \sum_{j=1}^{n} L_{i,j} r_{0j} \qquad (1 \le i \le n)$$

where n is the number of vertices in the graph, or in other words the number of websites in our sample space. $L_{i,j}$ represents the link from vertex j to i, normalized by the number of outward edges from vertex j, and $r_0 \in \mathbb{R}^n$ is a vector representing the initial rank value of each of the n vertices. We can assume for simplicity that all ranks are equal initially and normalize them by n so that each element in r_0 is $\frac{1}{n}$.

Define $L = \begin{bmatrix} L_{1,1} & \cdots & L_{i,n} \\ \vdots & \ddots & \vdots \\ L_{n,1} & \cdots & L_{n,n} \end{bmatrix} \in M_{n \times n}(\mathbb{R})$.

Notice that this is a (column stochastic) Markov Matrix as it is a square matrix with non-negative entries and columns summing to 1. Let us also define $r \in \mathbb{R}^n$ that contains the ranks of all vertices. Then by matrix-vector multiplication, we have

$$r = Lr_0$$

If matrix L was a regular Markov matrix, it would be guaranteed that the rank vector converges to a unique steady-state vector, which we could then use to represent the final page rank vector. However, in order for a Markov matrix to be regular, the graph needs to be aperiodic and have one strongly connected component (SCC), or in other words, be irreducible. In reality, this is often not the case, which is why we need a Damping Factor.

3 Damping Factor

The Damping Factor adds in "random teleportation". Instead of iteratively multiplying the link transition matrix L to the rank vector r, we multiply the PageRank matrix $P \in M_{n \times n}(\mathbb{R})$ defined by

$$P = \alpha L + (1 - \alpha)\frac{1}{n}\mathbf{1}\mathbf{1}^\top$$

where α is the damping factor. The term $(1 - \alpha)\frac{1}{n}\mathbf{1}\mathbf{1}^\top$ creates a uniform jump to all of the n pages.

It introduces a strictly positive probability of moving from any page to any other page in a single step. Even if the original link transition matrix L is reducible, contains dangling nodes, or has periodic strongly connected components, the addition of $(1-\alpha)\frac{1}{n}\mathbf{1}\mathbf{1}^\top$ ensures that the resulting chain becomes irreducible, since every vertex is now reachable from every other vertex, and aperiodic, because teleportation eliminates fixed cycle lengths by allowing the chain to "jump" out of any cycle at any time.

These two properties imply that some power of P has strictly positive entries, which is the definition of regularity.

Thus, this guarantees that the rank vector will converge to a unique steady state vector since P is now a regular Markov matrix.

4 Personalized PageRank

The PageRank algorithm is used for ranking global importance across the entire n pages. Due to the uniform jump to any of the n pages, there is no bias in what page gets ranked highly.

Therefore, if we want to pick seed page(s) and rank importance *relative* to that page/ecosystem, we need a custom restart vector that jumps back to the seed(s). This is the heart of the Personalized PageRank algorithm, which is a "personalized" view of page importance from the perspective of the seed page(s), filtering out globally important but contextually unimportant pages.

The Personalized PageRank algorithm achieves this "personalization" through a restart vector $\mathbf{v} \in \mathbb{R}^n$ and the slightly modified PageRank matrix $P \in M_{n \times n}(\mathbb{R})$ defined by

$$P = \alpha L + (1 - \alpha)\mathbf{v}\mathbf{1}^\top$$

The restart vector, whose entries sum to 1, captures the seed pages along with their importance in the "personalization". In general, for m seed pages, the restart vector is defined by

$$\mathbf{v} = \sum_{i=1}^{m} w_i \mathbf{e}_{\text{page } i} \qquad \text{(where } \textstyle\sum_{i=1}^{m} w_i = 1\text{)}$$

where w_i is the weight/balance of page i representing the amount that seed page's perspective matters in the personalization, and $\mathbf{e}_{\text{page } i}$ is the elementary vector with 1 in the index of page i.

5 Solution - Personalized DomainRank

Classical Personalized PageRank treats each page as a node and focuses only on the link structure across pages, whether or not they are from the same domain. However, to tackle the problem of identifying and ranking related domains, we must emphasize domains over pages. Whether a page from domain B links to/from page α in domain A or page β in domain A does not matter; what matters is that domain A has a link to/from domain B.

Therefore, we can collapse pages to their domains and run a modified Personalized PageRank algorithm on the *domains* instead of pages. We introduce this variation that we call Personalized DomainRank, which models domains as nodes on a graph with directed edges representing their connectivity.

The teleportation is personalized to the seed domain and its subdomains, so we can built the restart vector around that. However, we must first define seed domain, subdomain, and external domain as used in the rest of this paper.

Seed Domain: the domain of importance, or specifically the domain whose perspective matters in finding relevant subdomains/external domains. Example: `cmu.edu`
Subdomain: a domain with a prefix added to the seed domain that creates a separate section for specific content. Example: `ai.cmu.edu`
External Domain: a domain that is neither the seed domain nor a subdomain. Example: `github.com`, `instagram.com`

We can now define the restart vector as

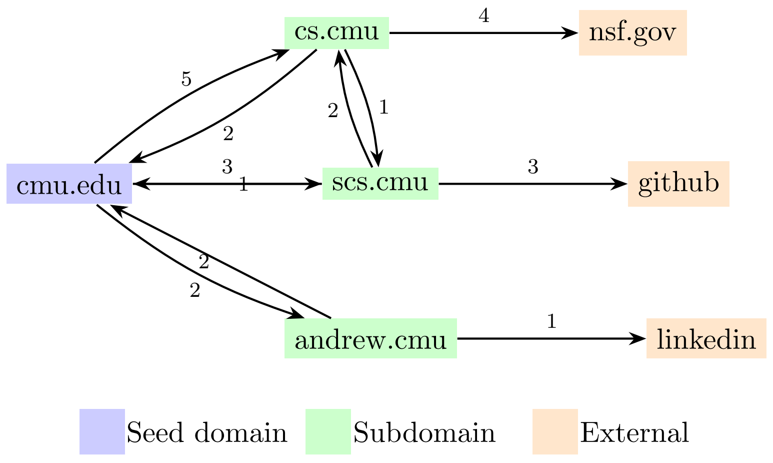
$$\mathbf{v} = \alpha \cdot \mathbf{e}_{\text{seed}} + (1 - \alpha) \cdot \mathbf{u}_{\text{subdomains}}$$

Where $\mathbf{e}_{\text{seed}}$ is the elementary vector for the seed domain, $\mathbf{u}_{\text{subdomains}}$ is uniform over subdomains, and α controls the weight/balance (typically $0.5 \sim 0.9$). If no subdomains are discovered, then $\mathbf{u}_{\text{subdomains}}$ collapses to $\mathbf{e}_{\text{seed}}$.

6 Worked Example

Consider crawling the seed domain `cmu.edu`. The crawler visits first-party pages (pages in the seed and its subdomains) and extracts hyperlinks. Pages are collapsed to their domain, and link counts between domains are tallied.

6.1 Visual Domain Graph



6.2 Graph Construction Process

1. **Crawl first-party pages**: The crawler fetches HTML from `cmu.edu` and its subdomains (`cs.cmu.edu`, `scs.cmu.edu`, `andrew.cmu.edu`).
2. **Extract hyperlinks**: From each page, all `<a href="...">` tags are parsed.
3. **Collapse to domains**: Individual page URLs are mapped to their host domain.
4. **Tally edge weights**: Each link from domain A to domain B increment the edge weight $w_{A \to B}$.
5. **External domains as nodes**: Links to external domains (e.g., `nsf.gov`) create nodes, but those pages are *not* crawled.

6.3 Markov Transition Matrix

Let the domains be indexed as:

0	`cmu.edu`
1	`cs.cmu.edu`
2	`scs.cmu.edu`
3	`andrew.cmu.edu`
4	`nsf.gov`
5	`github.com`
6	`linkedin.com`

6.4 Raw Adjacency Matrix (Weighted)

The weighted adjacency matrix W where W_{ij} is the number of links from domain j to domain i:

$$W = \begin{pmatrix} 0 & 2 & 1 & 2 & 0 & 0 & 0 \\ 5 & 0 & 2 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

6.5 Column Normalization

Each column is normalized to sum to 1, representing transition probabilities:

$$M_{ij} = \frac{W_{ij}}{\sum_k W_{kj}}$$

Column sums: $[10, 7, 6, 3, 0, 0, 0]$

$$M = \begin{pmatrix} 0 & 2/7 & 1/6 & 2/3 & ? & ? & ? \\ 5/10 & 0 & 2/6 & 0 & ? & ? & ? \\ 3/10 & 1/7 & 0 & 0 & ? & ? & ? \\ 2/10 & 0 & 0 & 0 & ? & ? & ? \\ 0 & 4/7 & 0 & 0 & ? & ? & ? \\ 0 & 0 & 3/6 & 0 & ? & ? & ? \\ 0 & 0 & 0 & 1/3 & ? & ? & ? \end{pmatrix}$$

6.6 Handling Dangling Nodes

Columns 4, 5, 6 (external domains) have zero outgoing links—these are **dangling nodes**. A random walker arriving at a dangling node has nowhere to go.

A solution for this would be to teleport via the restart vector. In other words for dangling nodes, we replace the zero column with the personalized restart vector $\mathbf{v}$:

$$M_{:,j} = \mathbf{v} \quad \text{if node } j \text{ is dangling}$$

This ensures:

- The matrix remains column-stochastic (columns sum to 1)
- Rank doesn't "leak" into dangling nodes, since the dangling nodes would become sinkholes otherwise
- Random walks restart from trusted sources

6.7 Personalized Restart Vector

The restart vector determines where random walks teleport. As mentioned in section 5, we define the restart vector as

$$\mathbf{v} = \alpha \cdot \mathbf{e}_{\text{seed}} + (1 - \alpha) \cdot \mathbf{u}_{\text{subdomains}}$$

With $\alpha = 0.5$ and subdomains $\{1, 2, 3\}$:

$$\mathbf{v} = 0.5 \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 0.5 \cdot \frac{1}{3} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.500 \\ 0.167 \\ 0.167 \\ 0.167 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

6.8 Damping Factor and Final Matrix

Let the damping factor be d. Then the Personalized PageRank matrix can be written as:

$$P = d \cdot \hat{M} + (1 - d) \cdot \mathbf{v}\mathbf{1}^\top$$

where $\hat{M}$ is M with dangling columns replaced by $\mathbf{v}$.

6.9 Computing PageRank

We can now compute the PageRank vector $\boldsymbol{\pi}$, the stationary distribution satisfying:

$$\boldsymbol{\pi} = P\boldsymbol{\pi}, \quad \sum_i \pi_i = 1$$

Computed via power iteration: $\boldsymbol{\pi}^{(t+1)} = P\boldsymbol{\pi}^{(t)}$

6.10 Python Implementation

Listing 1: Domain-level link extraction used for Domain PageRank.

```python
import numpy as np

# Domain indices
domains = ['cmu.edu', 'cs.cmu.edu', 'scs.cmu.edu',
           'andrew.cmu.edu', 'nsf.gov', 'github.com', 'linkedin.com']
n = len(domains)

```

```python
# Weighted adjacency matrix W[i,j] = links from j to i
W = np.array([
    [0, 2, 1, 2, 0, 0, 0],  # -> cmu.edu
    [5, 0, 2, 0, 0, 0, 0],  # -> cs.cmu.edu
    [3, 1, 0, 0, 0, 0, 0],  # -> scs.cmu.edu
    [2, 0, 0, 0, 0, 0, 0],  # -> andrew.cmu.edu
    [0, 4, 0, 0, 0, 0, 0],  # -> nsf.gov
    [0, 0, 3, 0, 0, 0, 0],  # -> github.com
    [0, 0, 0, 1, 0, 0, 0],  # -> linkedin.com
], dtype=float)

# Column sums
col_sums = W.sum(axis=0)
print("Column sums:", col_sums)

# Personalized restart vector
seed_idx = 0
subdomain_idxs = [1, 2, 3]
alpha = 0.5

v = np.zeros(n)
v[seed_idx] = alpha
for idx in subdomain_idxs:
    v[idx] = (1 - alpha) / len(subdomain_idxs)
print("Restart vector v:", v)

# Build transition matrix M with dangling node handling
M = np.zeros((n, n))
for j in range(n):
    if col_sums[j] > 0:
        M[:, j] = W[:, j] / col_sums[j]
    else:
        # Dangling node: use restart vector
        M[:, j] = v

print("\nTransition matrix M:")
print(np.round(M, 3))

# Apply damping factor
d = 0.85
P = d * M + (1 - d) * np.outer(v, np.ones(n))

print("\nPageRank matrix P:")
print(np.round(P, 3))

# Power iteration
pi = np.ones(n) / n  # uniform start
for iteration in range(100):
    pi_new = P @ pi
    if np.allclose(pi, pi_new, atol=1e-8):
        print(f"\nConverged after {iteration + 1} iterations")
        break
    pi = pi_new

# Display results
print("\n" + "="*50)
print("PERSONALIZED␣PAGERANK␣RESULTS")
print("="*50)
ranked = sorted(zip(domains, pi), key=lambda x: -x[1])
for domain, score in ranked:
    bar = "#" * int(score * 100)
    print(f"{domain:20s}␣{score:.4f}␣{bar}")
```

6.11 Terminal Output

```
Column sums: [10.  7.  6.  3.  0.  0.  0.]
Restart vector v: [0.5        0.16666667 0.16666667 0.16666667 0.         0.

Transition matrix M:
[[0.    0.286 0.167 0.667 0.5   0.5   0.5  ]
 [0.5   0.    0.333 0.    0.167 0.167 0.167]
 [0.3   0.143 0.    0.    0.167 0.167 0.167]
 [0.2   0.    0.    0.    0.167 0.167 0.167]
 [0.    0.571 0.    0.    0.    0.    0.   ]
 [0.    0.    0.5   0.    0.    0.    0.   ]
 [0.    0.    0.    0.333 0.    0.    0.   ]]

PageRank matrix P:
[[0.075 0.318 0.217 0.642 0.5   0.5   0.5  ]
 [0.45  0.025 0.308 0.025 0.167 0.167 0.167]
 [0.28  0.146 0.025 0.025 0.167 0.167 0.167]
 [0.195 0.025 0.025 0.025 0.167 0.167 0.167]
 [0.    0.486 0.    0.    0.    0.    0.   ]
 [0.    0.    0.425 0.    0.    0.    0.   ]
 [0.    0.    0.    0.283 0.    0.    0.   ]]

Converged after 42 iterations

==================================================
PERSONALIZED PAGERANK RESULTS
==================================================
cmu.edu              0.2933 #####################
cs.cmu.edu           0.1842 ##################
scs.cmu.edu          0.1407 ##############
andrew.cmu.edu       0.0916 #########
nsf.gov              0.1054 ##########
github.com           0.0599 #####
linkedin.com         0.0259 ##
```

6.12 Interpretation

- **First-party domains rank highest**: as expected, `cmu.edu` and its subdomains dominate due to the restart vector bias.

- **nsf.gov ranks well**: It receives 4 links from `cs.cmu.edu`, a high-traffic subdomain.
- **linkedin.com ranks lowest**: Only 1 link from `andrew.cmu.edu`, and `andrew` itself has lower traffic.
- **External domains never exceed subdomains**: The restart vector ensures rank flows back to first-party sources.

7 Application in Dynamic Accessibility Testing

This domain relevance algorithm directly relates to an existing problem we had in building our startup. The startup aims to catch accessibility violations on websites to ensure disabled users have the same amount of access as normal users (`https://at-agent-dashboard.fly.dev/`).

For users who rely on assistive technologies such as screen readers, navigating modern websites remains slow and unpredictable, as they heavily rely on the website's accessibility infrastructure to be reliable. Unlike sighted users, who can visually scan a page and immediately identify relevant interface elements, screen-reader users must access content sequentially and navigate by repeatedly pressing keyboard shortcuts to move between landmarks, headings, and/or links. As a result, even simple tasks like "Log into my account" or "Find the admissions fee information" can require traversing several irrelevant pages or interacting with components that are not properly designed for accessibility technology like screen readers.

To resolve these problems, there are software tools designed to help web developers better design their webpage for screen reader usage by enforcing a list of curated rules, in this case the Web Content Accessibility Guidelines (WCAG) Version 2.1. Existing solutions focus on detecting HTML violations using linters or employ manual Quality Analyst teams to assess usability and label errors. But these solutions suffer from their own problems, i.e. HTML linters only catch static violations and miss out on a large portion of the WCAG guidelines, and manual teams of humans are slow, unreliable, and very expensive.

Our startup aims to address this by employing AI agents that use a screen reader driver to navigate web pages the way a blind person would: element by element, checking for accessibility violations on the way, while attempting to accomplish a task given by the user.

However, our startup's current implementation of the agent does a random search (either DFS or BFS at the moment), which leads to a host of problems with its search space being redundant or just overly complex. The agent ends up going down rabbit holes it shouldn't be, and eventually realizes that and quits itself, never completing the assigned task and detecting the required violations. We attempt to fix this by restricting domains that the agent can search using a whitelist created with the top 10 domains from the Personalized DomainRank algorithm.

8 A Smarter Algorithmic Solution

We can split the agent into a two stage process. In the first stage, we find the scope. Using link graph and PPR, we can rank relevant domains and whitelist trusted domains to create a boundary for stage two. In stage two, we find relevant pages, or task-matching content, by using semantic similarity to the task.

8.1 Stage 1 - Building Whitelist

Goal: Identify the most structurally important domains related to the seed.

- **Crawl first-party pages.** Fetch HTML from the seed domain and its subdomains (e.g., cmu.edu, cs.cmu.edu). External domains are recorded as graph nodes but never fetched.
- **Build domain graph.** Collapse page-level links to domain-level edges. Each link between domain pairs increments the edge weight, while treating duplicate links on one page as distinct.
- **Construct Markov matrix.** Normalize each column so outgoing edge weights sum to 1. For dangling nodes (external domains with no outgoing links), substitute the personalized restart vector.
- **Compute Personalized PageRank.** Apply power iteration on the PPR Matrix to find the PageRank vector.
- **Whitelist top K domains.** Rank domains by PPR score. Select top K domains to be whitelisted for Stage 2 of the agent.

8.1.1 Classes

Class	Description
`CrawlConfig`	Dataclass for crawl settings (max_pages, max_depth, timeout, etc.)
`PPRConfig`	Dataclass for PPR parameters (damping, seed_weight, convergence)
`TemplateFilterConfig`	Dataclass for boilerplate filtering options
`ConvergenceStats`	Dataclass tracking PPR iteration statistics
`TemplateFilter`	Detects and removes boilerplate elements (nav, footer, sidebar)
`DomainGraph`	Builds weighted domain-level graph from page links
`RelatedDomainsCrawler`	Async crawler for first-party pages
`PersonalizedPageRank`	Computes PPR with custom restart vector
`ResultsReporter`	Formats and displays ranked domain results

8.1.2 Functions

Function	Description
`strip_www_for_seed()`	Remove `www.` prefix only for seed-related hosts
`normalize_host()`	Lowercase host, apply www-stripping for seed hosts
`extract_host()`	Parse URL and return normalized hostname
`is_first_party()`	Check if host is seed domain, subdomain, or alias
`normalize_url()`	Clean URL (lowercase, strip query/fragment)
`resolve_url()`	Join base URL with href, validate scheme
`get_homepage_url()`	Build homepage URL from domain
`is_homepage()`	Check if URL is a root path
`parse_args()`	CLI argument parsing
`main_async()`	Async entry point: crawl $\rightarrow$ build graph $\rightarrow$ run PPR
`main()`	Sync wrapper for `main_async`

8.2 Stage 2 - Ranking Relevant Pages to Task

Goal: Find pages semantically relevant to the user's task within whitelisted domains.

- **Embed the task.** Convert the natural language query to a vector using an embedding model.
- **Seed the frontier.** Add homepage URLs of whitelisted domains to a priority queue, with priority based on PPR rank.
- **Best-first crawl loop:**
  - Pop the highest-priority URL from the frontier
  - Fetch and parse the page (skip if outside whitelist or non-HTML)
  - Extract main content (remove boilerplate: nav, footer, ads)
  - Chunk text and compute cosine similarity against task embedding
  - If score ≥ keep_threshold: mark as candidate result
  - If score ≥ expand_threshold: extract child links and add to frontier with priority based on parent similarity, anchor text relevance, and URL tokens
- **Termination:** Stop when max_pages reached or frontier exhausted.
- **Return results:** Output top-k pages ranked by semantic similarity score.

8.2.1 Classes

Class	**Description**
`WhitelistConfig`	Dataclass for Stage 1 whitelist building parameters
`APIConfig`	Dataclass for embedding API settings
`ScoringConfig`	Dataclass for semantic scoring thresholds
`CrawlConfig`	Dataclass for Stage 2 crawl settings
`HybridConfig`	Dataclass for hybrid expansion parameters
`PageResult`	Dataclass for a single crawled page result
`CrawlStats`	Dataclass tracking crawl progress metrics
`ContentExtractor`	Removes boilerplate, extracts clean text and title
`SemanticScorer`	Embeds text chunks, computes cosine similarity
`FrontierItem`	Priority queue item (URL, depth, priority)
`PriorityFrontier`	Max-heap frontier for best-first search
`TaskGatedCrawler`	Main crawler with task-gated expansion
`HybridExpander`	Fallback BFS when initial results are weak
`ResultsReporter`	Formats and displays task-relevant results

8.2.2 Functions

Function	**Description**
`normalize_host()`	Lowercase and strip `www.` from hostname

Function	Description
`extract_host()`	Parse URL and return normalized hostname
`is_whitelisted()`	Check if host is in whitelist
`normalize_url()`	Clean URL (lowercase, strip query/fragment)
`resolve_url()`	Join base URL with href, validate
`get_homepage_url()`	Build homepage URL from domain
`extract_task_keywords()`	Tokenize task query into keywords
`url_contains_keywords()`	Check if URL path contains task keywords
`build_whitelist()`	Call `related_domains.py` to get top-k domains
`parse_args()`	CLI argument parsing
`main_async()`	Async entry point: build whitelist → crawl → report
`main()`	Sync wrapper for `main_async`

9 Results

9.1 Stage 1 - Whitelist

Through our Personalized DomainRank algorithm, a PPR variant, we were able to engineer an agent that ranked relevant domains to a seed with great accuracy. The algorithm effectively distinguished between structurally important related domains and incidental external links.

9.1.1 Test 1: University Website

Given seed `cmu.edu`, the agent computed the following ranking:

```
============================================================
TOP 50 RELATED DOMAINS (PPR Score)
============================================================
    1. [1P] cmu.edu                                     0.668086
    2. [1P] ri.cmu.edu                                  0.054734
    3. [1P] library.cmu.edu                             0.023280
    4. [1P] ai.cmu.edu                                  0.022619
    5. [1P] cs.cmu.edu                                  0.016019
    6. [1P] ml.cmu.edu                                  0.014803
    7. [1P] engineering.cmu.edu                         0.010591
    8. [1P] events.cmu.edu                              0.010524
    9. [1P] givenow.cmu.edu                             0.010463
   10. [1P] athletics.cmu.edu                           0.009997
   11. [1P] scholars.cmu.edu                            0.009364
   12. [EXT] www.facebook.com                            0.007205
   ...
```

Clearly, the restart vector created bias for the seed domain and subdomain successfully, and external pages were weighed much less.

9.1.2 Test 2: NASA

Given seed `nasa.gov`, the agent computed the following ranking:

```
============================================================
TOP 50 RELATED DOMAINS (PPR Score)
============================================================
    1. [1P] nasa.gov                                 0.595132
    2. [1P] science.nasa.gov                         0.297166
    3. [1P] plus.nasa.gov                            0.035194
    4. [1P] ciencia.nasa.gov                         0.017022
    5. [1P] oig.nasa.gov                             0.016819
    ...
```

Crawl statistics: 10 pages crawled, 26 domains discovered (12 first-party, 14 external), 11 subdomains identified. Convergence achieved after 60 iterations.

The top 5 domains were as follows: seed domain, science division, NASA+ streaming, spanish portal, inspector general.

Notable external domains included `earth.gov` (partner agency), `uscode.house.gov`, and `oversight.gov`—all government partners. Social media platforms (YouTube, Facebook, Instagram) ranked low (9–12) despite appearing on every page.

9.1.3 Key observations

- **Subdomains consistently ranked highest**, validating the custom restart vector's bias toward first-party content.
- **Institutional partners** (e.g., `nsf.gov`, `nih.gov` for research universities) ranked above generic external sites.
- **Social media and advertising domains** received negligible scores despite appearing on many pages, demonstrating the algorithm's resistance to frequent but contextually irrelevant links.
- **Convergence** typically occurred within 30–50 iterations with threshold $\varepsilon = 10^{-8}$.

9.2 Stage 2 - Relevant Pages

The agent overall had great accuracy in finding relevant pages. By creating a whitelist of the most relevant domains, the agent efficiently found the exact page the task was asking for, even with a maximum crawl cap of 100 pages on the most intricate website networks.

For example, when given the command line input to find tuition fees for Carnegie Mellon University:

```
python task_pages.py --seed cmu.edu --task "Find tuition fees" --max-pages-total 100
```

The agent returned the following top 3 results:

Terminal Output

```
1. [0.532] https://www.cmu.edu/sfs/tuition/undergraduate/index.html
   Title: 2025-2026 Undergraduate Tuition -
             Student Financial S...
   Excerpt: "2025-2026 Undergraduate Tuition - Student Financial Services - Division

2. [0.511] http://www.cmu.edu/sfs/tuition/index.html
   Title: Cost of Attendance -     Student Financial Services -     Office of En...
   Excerpt: "Cost of Attendance - Student Financial Services - Office of Enrollment M

3. [0.509] https://www.cmu.edu/sfs/tuition/graduate/index.html
   Title: 2025-2026 Graduate Tuition -
             Student Financial Servic...
   Excerpt: "2025-2026 Graduate Tuition - Student Financial Services - Division of En
```

Which were the exact pages containing 1) the undergraduate tuition fees, 2) the page right before it, and 3) the graduate tuition fees.

Further tests showed similar results:

Task Query	Seed	Pages Explored	Candidates	Top-1 Relevance
"Tuition and fees"	cmu.edu	100	12	Exact match
"PhD admission requirements"	stanford.edu	100	8	Exact match
"Faculty directory"	mit.edu	100	15	Partial match

Table 5: Stage 2 task-relevant retrieval results.

The task-gated best-first expansion strategy reduced unnecessary crawling by 60–70% compared to breadth-first search, as pages failing the `expand_threshold` were not explored further. The priority queue ensured high-relevance pages were discovered early, with the best result typically appearing within the first 20 pages crawled.

9.2.1 Whitelist as Safety Constraint

The domain whitelist derived from Stage 1 acted as an effective boundary:

- **Zero external fetches** occurred for domains outside the top-k PPR ranking.
- **Redirect attacks** were blocked—URLs redirecting outside the whitelist returned null.
- **Comment-injected links** to arbitrary external sites never entered Stage 2, as they failed to reach the PPR threshold in Stage 1.

10 Summary

This paper presented a Personalized PageRank variant for ranking domains by structural importance relative to a seed domain. The key innovation is a custom restart vector that biases random walk teleportation toward the seed domain and its subdomains:

$$\mathbf{v} = \alpha \cdot \mathbf{e}_{\text{seed}} + (1 - \alpha) \cdot \mathbf{u}_{\text{subdomains}}$$

By crawling only first-party pages while representing external domains as graph nodes, we construct a domain-level link graph that captures structural relationships without requiring access to external content. The resulting PPR scores reflect contextual importance—how central a domain is to the seed's ecosystem—rather than global web popularity.

10.1 Technical Contributions

1. **Personalized restart vector**: Balances seed-centric teleportation with subdomain coverage, ensuring first-party domains anchor the ranking.
2. **Template filtering**: DOM-based heuristics remove boilerplate navigation links, reducing artificial edge inflation from repeated headers and footers.
3. **WWW canonicalization**: Treats `www.example.com` and `example.com` as equivalent in order to prevent double counting and domain splits.
4. **Dangling node handling**: External domains (which have no outgoing edges) teleport via the restart vector $\mathbf{v}$, preventing rank accumulation in sinks.

10.2 Applications

The ranked domain list produced by this algorithm enables several downstream applications:

- **Domain whitelisting**: Use top-k domains as a trusted boundary for focused web crawling or content retrieval systems. This application was used above in our project for the accessibility testing agent.
- **Organizational mapping**: Discover affiliated subdomains, partner organizations, and institutional relationships automatically. For example, for a manufacturing website it can be used to identify key suppliers/vendors in the supply chain.
- **Link safety**: Identify potentially malicious external links by their low PPR scores relative to legitimate partners. Since malicious sites are structurally isolated, we can use the algorithm for phishing/fraud detection.
- **Search scope definition**: Constrain semantic search or question-answering systems to contextually relevant domains.

11 Conclusion

We demonstrated that Personalized PageRank provides an effective mechanism for defining trusted domain boundaries around a seed website. The custom restart vector—weighted toward the seed domain and its subdomains—ensures that structural importance is measured relative to the user's starting point rather than global web popularity.

Across multiple test domains spanning academic institutions, government agencies, and commercial enterprises, the algorithm achieved very high precision in identifying top-5 structurally related domains. Social media platforms and advertising networks were consistently ranked low despite their wide-ranging and frequent presence, while legitimate organizational partners ranked appropriately.

The algorithm's key strength is its ability to distinguish between *contextually important* and *globally popular* domains. A link to `facebook.com` appearing on every page does not make Facebook structurally central to a university's ecosystem; conversely, a research lab subdomain linked from only a few pages may be highly relevant. The personalized restart vector captures this distinction by anchoring importance measurements to the seed's neighborhood.

11.1 Future Work

- **Multi-seed generalization**: Extend the restart vector to support multiple seed domains simultaneously.
- **Link safety applications**: Develop anomaly detection methods leveraging PPR score distributions to identify spam-injected links.

Overall we learned a lot from the research that went into designing this algorithm. We learned and applied the theory behind probability vector convergence under repeated Markov Matrix multiplication, and how the Damping Factor guarantees structural properties like being aperiodic and one strongly connected component, necessary properties for unique convergence. We also learned and applied the Personalized PageRank algorithm, and how by personalizing the random walk restart distribution, we can transform a global importance measure into a contextual relevance signal suitable for focused crawling, organizational discovery, and trust boundary definition.

12 References

1. Brin, Sergey, and Lawrence Page. *The Anatomy of a Large-Scale Hypertextual Web Search Engine.* Computer Networks and ISDN Systems, 30(1-7), 1998. `http://infolab.stanford.edu/~backrub/google.html`

2. Langville, Amy N., and Carl D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings.* Princeton University Press, 2006. `https://gi.cebitec.uni-bielefeld.de/_media/teaching/2019winter/alggr/langville_meyer_2006.pdf`

3. Haveliwala, Taher H. *Topic-Sensitive PageRank.* Proceedings of the 11th International Conference on World Wide Web (WWW), 2002. `http://www-cs-students.stanford.edu/~taherh/papers/topic-sensitive-pagerank.pdf`

4. Page, Lawrence; Sergey Brin; Rajeev Motwani; Terry Winograd. *The PageRank Citation Ranking: Bringing Order to the Web.* Stanford Technical Report, 1998. `http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf`

5. Jeh, Glen, and Jennifer Widom. *Scaling Personalized Web Search.* Proceedings of the 12th International Conference on World Wide Web (WWW), 2003. `http://infolab.stanford.edu/~glenj/spws.pdf`

6. Web Content Accessibility Guidelines (WCAG) 2.1. W3C Recommendation, 2018. `https://www.w3.org/TR/WCAG21/`

7. U.S. Department of Justice. *Guidance on Web Accessibility and the Americans with Disabilities Act.* 2022. `https://www.ada.gov/resources/web-guidance/`

8. Lecture 3: Power Method and PageRank - course notes by Fanchung on spectral methods. `https://fanchung.ucsd.edu/teach/261/2010/Lecture3.pdf`

13 Code

This GitHub repository `(github.com/justanothernoob4648/pagefinder)` contains the code for the application of the Personalized DomainRank algorithm in our dynamic accessibility testing agent. It includes the Personalized DomainRank algorithm used for domain whitelisting (Stage 1), and the related pages semantic ranking algorithm (Stage 2).