



# **Data Structures: Fraud Detection via Personalized PageRank**

Instructor: Dr. Katanforoush

Shahid Beheshti University — Fall 2025

# 1 Project Overview and Context

This project needs you to build a robust, scalable fraud detection system from the ground up using Graph analysis. The approach is based on the proven success of the **TrustRank** and **BadRank** algorithms which spot potentially dodgy entities by taking a close look at how far away from known fraudulent entities they are in a transaction network.

The key part of this assignment is to create your own version of the **Personalized PageRank (PPR)** algorithm that's been optimized for this task. You not only need to implement the math behind it but also figure out how to get graph algorithms to run smoothly on massive, sparse data sets.

## Core Idea: "Guilt by Association"

In financial and social networks, fraud tends to cluster in a social way. If someone's interacting all the time with people they know are fraudsters then it's likely they are high risk as well. By representing entities (users / wallets) as nodes and interactions (transactions) as connections, we can spread a "suspicion score" from a small group of known fraudsters (the seed set) out to the rest of the network.

## Algorithmic Foundation

You will implement **Personalized PageRank**, a variation of the standard PageRank algorithm.

- **Standard PageRank:** Measures global importance based on the random surfer model.
- **Personalized PageRank (PPR):** Biases the random surfer to "teleport" back to a specific subset of nodes (the seed set) rather than a uniform distribution. This measures proximity to the seeds.

This project also touches upon **TrustRank**, which is essentially PPR used to propagate "trust" from good seeds (or "distrust" from bad seeds).

# 2 Tasks and Technical Requirements

You must design, implement, and document the system. The focus is on **algorithmic correctness**, **efficiency**, and **handling edge cases**.

## Implementation Requirements

1. **Custom PageRank Engine:** Implement the Power Iteration method for Personalized PageRank.

$$\mathbf{r}^{(t+1)} = (1 - \alpha) \cdot \mathbf{r}^{(t)} M + \alpha \cdot \mathbf{p}$$

Where  $\mathbf{r}$  is the rank vector,  $M$  is the transition matrix,  $\alpha$  is the teleportation probability (usually 0.15), and  $\mathbf{p}$  is the personalized teleportation vector (non-zero only for known fraudsters).

2. **Sparse Matrix Optimization:** Real-world graphs are sparse. Storing a full  $N \times N$  matrix is impossible for large  $N$ . You must implement or utilize a **Sparse Matrix** representation (e.g., Compressed Sparse Row/Column logic or Adjacency Lists) to handle the transition matrix  $M$  efficiently in memory.
3. **Handling Edge Cases:** Your algorithm must explicitly handle:
  - **Dead Ends (Dangling Nodes):** Nodes with no outgoing edges. These act as "rank sinks" and leak probability mass. You must implement a strategy to redistribute their mass (e.g., to the seed set or uniformly).
  - **Disconnected Components:** The graph may not be fully connected.
  - **Graph Data Support:** Your code must be able to handle both weighted and unweighted directed graphs. Your documentation should clearly specify the input format you expect.
4. **Convergence Criteria:** Implement a rigorous stopping criterion using the  $L_1$  norm of the difference between iterations:  $\|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}\|_1 < \epsilon$ . Do not just use a fixed number of iterations.

5. **Visualization:** Provide a clear visualization of your algorithm's output. You may use a graphical interface such as [Pygame](#) or [Tkinter](#), or present the results in a well-formatted manner in the command line or by writing them to a file. The more effort you put into producing an informative and polished visualization, the higher your final score is likely to be.



**Warning:**

Using libraries such as [NetworkX](#) to represent structures and Base algorithms like PageRank are allowed, provided you also understand the underlying mechanics.

## Analysis & Evaluation Requirements

You should test your system on at least two datasets - a small synthetic one for thorough debugging and a larger real-world dataset for thorough evaluation. The recommended dataset to start with is the **Facebook dataset** mentioned in Section 6 (the Caltech Facebook dataset). That said, if you've got time to try out some other datasets from the **Stanford SNAP library** - these will also count towards your **bonus** points.

- **Scalability Test:** Measure and plot the runtime of your algorithm as the graph size increases (e.g., 1k, 10k edges).
- **Parameter Sensitivity:** Analyze how the **damping factor** ( $\alpha$ ) affects the "spread" of suspicion. Does a higher  $\alpha$  keep suspicion closer to the fraudsters?
- **Precision@K:** If you have ground-truth labels, calculate the Precision@K for the top  $K$  flagged nodes. How many of your top 50 suspects are actually fraudulent?

## 3 Documentation Structure

Your report must be a self-contained PDF following this exact structure:

1. **Problem Definition:** 4-6 sentences on the "Guilty by Association" model.
2. **Algorithmic Approach:** Explain the math of PPR and the "Teleportation" vector.
3. **Data Representation:** Detail your sparse storage choice (CSR, Adjacency List) and why it matters for  $O(V + E)$  complexity.
4. **Detailed Algorithm Design:** Pseudocode handling the "Dangling Node" fix and Power Iteration.
5. **Implementation Details:** Architecture and key optimizations.
6. **Experiments & Methodology:** Datasets used, machine specs, and convergence thresholds ( $\epsilon$ ).
7. **Results & Analysis:**
  - Convergence plots (Error vs. Iterations).
  - Runtime plots (Time vs. Graph Size).
  - Interpretation of the top-ranked suspicious nodes.
8. **Limitations:** Discuss "[Cold Start](#)" problems and dynamic graph updates.
9. **References:** Cite papers, LLMs and resources used.

## 4 Advanced Challenges (Bonus)

For students aiming for the highest grade, implement one of the following:

- **Monte Carlo Approximation:** Implement a random-walk-based approximation for PPR and compare its accuracy/speed against your Power Iteration method.
- **Incremental PageRank:** Implement a method to update scores dynamically when a new edge is added, without recomputing from scratch.

## 5 Recommended Approach

It is recommended completing the project in three main phases, obviously you can approach this project how you want as long as the final submission follows what is written in this handout.

### Phase 1: Understand the PageRank Algorithm

Begin by studying how the PageRank algorithm works. Focus on the intuition behind the algorithm, the mathematical formulation, and the role of link structures. Once you understand the underlying concepts, implement a basic version to solidify your grasp of the method.

### Phase 2: Extend PageRank for Fraud Detection

After implementing the base algorithm, modify your code to incorporate fraud-detection mechanisms. This may involve adjusting the ranking logic, adding heuristic rules, or introducing new features that help identify suspicious or manipulative behavior in the graph. Your goal in this phase is to transform the standard PageRank algorithm into a version tailored for detecting fraud.

### Phase 3: Apply, Evaluate, and Optimize

Finally, run your modified algorithm on a **real-world dataset**. Analyze its behavior, tune your parameters, and optimize for both accuracy and efficiency. During this phase, document your experiments, decisions, and observations. This material will form the foundation of your final report and submission.

## 6 Resources and Further Reading

Use these resources to deepen your understanding:

### Papers & Articles:

- **Fraud Detection with Personalized PageRank (Blog Article):** Theodo Data Science Blog. [\[Link\]](#)
- **PageRank Implementation Guide (GeeksforGeeks):** Step-by-step PageRank explanation and code. [\[Link\]](#)
- **PageRank Course Project Notes (EECS 390):** Detailed overview of PageRank concepts and implementation details. [\[Link\]](#)
- **Original Paper:** Page, L., et al. (1999). *The PageRank Citation Ranking: Bringing Order to the Web*. [\[Link\]](#)
- **TrustRank Paper:** Gyöngyi, Z., et al. (2004). *Combating Web Spam with TrustRank*. [\[Link\]](#)

### Datasets:

- **Stanford SNAP Datasets:** Large-scale real-world network datasets (Epinions, Slashdot, Bitcoin, etc.). [\[Link\]](#)
- **Social Graph Dataset:** Caltech Facebook Network Dataset from Network Repository. [\[Link\]](#)

### Videos:

- **Visual Explanation:** “PageRank: A Trillion Dollar Algorithm.” [\[Link\]](#)
- **PageRank Intuition & History:** Animated introduction to PageRank and its motivation. [\[Link\]](#)

## Code Repositories:

- **TrustRank Python Implementation:** Clean example implementation of TrustRank and variations for graph-based ranking. [\[Link\]](#)
- **PageRank Implementation:** Implementation of PageRank algorithm along with TopicSpecific and TrustRank. [\[Link\]](#)



### Info:

While it's not a must to digest every last detail in this section to get PageRank up and running, I'd really recommend taking a look at those videos and giving those articles & repositories a read. That way you'll get a much clearer picture of what's going on and it'll make the whole implementation process a whole lot smoother.

## 7 Submission

Submit a single ZIP file containing:

- `src/`: Source code (modular, commented).
- `docs/`: The PDF report, it should contain all the testing and parameters that been tried leading up to the final algorithm.
- `data/`: The datasets used (or links/scripts to download them).
- `results/`: Plots and top-k ranking files, similar to the report you submit after each lab session.
- `README.md`: Instructions to install dependencies and run the code.



### Warning:

The README file and your Report should cover different ground. Reports are about laying out the nitty gritty of what you did, how you did it, what worked and what didn't. We also want to see the journey that led you to your final product, so that we can tell if you really did put the work into it. On the other hand, the README is just some basic information for people who want to run your project - they don't need to know about all the ins and outs.

*Stay hungry, stay foolish*