Pregel-based TrustRank Fraud Detection

Fraud Analytics Project Report

Sanskriti Agarwal
CS24MTECH14002

Kocherla Sai Kiran
AI24MTECH02003

Kota Dhana Lakshmi
AI22BTECH11012

Instructor: Prof. Sobhan Babu
Institution: IIT Hyderabad

March 5, 2025

Contents

Abstract

Financial transaction networks have grown explosively in the digital era, enabling rapid payments but also facilitating anonymous fraud. Detecting fraudulent entities in such networks is challenging because of the scale and the subtle link patterns among illicit actors. TrustRank is an algorithm originally proposed for web spam detection that propagates trust from a seed set to rank nodes. In this project, we adapt TrustRank for fraud detection in a payment network using a **Pregel-based vertex-centric framework**. In this model, each account propagates a fraud (or distrust) score to its neighbors through a series of synchronized supersteps. Our implementation includes object-oriented components for the `Vertex`, `Worker`, and `PregelEngine` classes, enabling parallel message passing and iterative computation.

1 Problem Statement

We model a payment network as a directed multi-graph, where:

- **Nodes** represent user accounts.
- **Edges** represent payments, weighted by transaction amounts.

A subset of senders is known to be fraudulent (bad senders). The goal is to propagate a *fraud score* from these seed nodes through the network to identify additional fraudulent senders. The assumption is that fraudulent senders tend to transact mostly with other fraudulent senders, creating a cluster that can be detected by the algorithm.

2 Dataset Description

Two datasets are used in this analysis:

- **Payments.csv:** Each record consists of a sender ID, receiver ID, and amount transferred.This file has a total of 130,535 entries.
- **bad_sender.csv:** Contains 20 known fraudulent sender IDs.

Sample Data from Payments.csv

The payment data represents transactions between different entities (such as individuals, businesses, or bank accounts). This data is utilized to construct a directed graph where:

- **Nodes** represent entities (e.g., individuals or organizations involved in transactions).
- **Edges** represent payment transactions between two entities, with the *direction* of the edge indicating the flow of money (from the payer to the payee).

3 The Pregel Framework in TrustRank Implementation

The algorithm is implemented using the **Pregel** model, which is a vertex-centric *Bulk Synchronous Parallel* (BSP) framework. In this model:

- Each vertex (user account) independently performs a compute operation.

Sender	**Receiver**	**Amount**
1309	1011	123051
1309	1011	118406
1309	1011	112456
1309	1011	120593
1309	1011	166396
1309	1011	177817
1309	1011	169351
1309	1011	181038
1309	1011	133216
1309	1011	133254

Table 1: First 10 entries of `Payments.csv`

- Computation proceeds in **supersteps**; in each superstep, every vertex:
  (a) Receives messages from neighbors (sent during the previous superstep).
  (b) Updates its *trust score* based on the received messages.
  (c) Sends out messages to its outgoing neighbors, proportionally to the weight of the edges.
- Global synchronization occurs at the end of each superstep.

The architecture comprises three main components:

- **Vertex:** Represents a single node. It stores its ID, current score, outgoing edges (neighbors with weights), and a flag indicating if it is a known bad sender.
- **Worker:** Handles a partition of the graph. It executes the compute function for all vertices in its partition and manages message passing for them.
- **PregelEngine:** Orchestrates the entire process by initializing the graph, partitioning vertices among Workers, coordinating supersteps, aggregating messages, and checking for convergence.

Tools and Libraries

- **Python** for implementation
- **Pandas** for data handling
- **Matplotlib** visualization.
- **Threading** for parallel processing.

Message Passing and Convergence

During each superstep, a vertex sums all incoming messages (which are fractions of trust scores from its neighbors) and updates its own score.The update rule is:

$$\text{new\_score} = (1-\beta)\,\text{bias} + \beta \times \left(\sum \text{incoming\_messages}\right),$$

where:

- β is the damping factor (set to 0.85).
- bias $= \frac{1}{|L|}$ if the vertex is a known bad sender (seed), and 0 otherwise.
- $\sum$ incoming_messages is the total of the trust fractions received from in-neighbors.

This process continues until either no vertex changes its score by more than a small tolerance (indicating convergence) or a maximum number of supersteps (e.g., 100) is reached. After the iterations, a post-processing normalization adjusts the total trust score to sum to 1.0.(*Additionally, any remaining trust mass* $(1 - \text{total\_trust})$ *is redistributed equally among the known bad senders to ensure the total trust sums to 1.0.*)

4 High-Quality Pseudocode

Below is the pseudocode that captures the object-oriented design and flow of the Pregel-based TrustRank algorithm.

Listing 1: Pregel-based TrustRank Pseudocode

```
Class Vertex:
    Attributes:
        id // Unique account identifier
        score // Current trust score
        out_neighbors // List of (neighbor_id, normalized_weight)
        is_bad_seed // Boolean flag: True if vertex is a known bad sender

    Method compute(messages, beta, alpha):
        incoming_sum = sum(messages)
        seed_bias = (is_bad_seed ? 1/|L| : 0)
        new_score = seed_bias + beta * incoming_sum + (1 - beta) * alpha

        // Distribute new score to outgoing neighbors
        For each (neighbor_id, weight) in out_neighbors:
            sendMessage(neighbor_id, new_score * weight)

        // Update and check for convergence (optional)
        delta = |new_score - score|
        score = new_score
        If delta < TOLERANCE:
            vote_to_halt()
        Else:
            remain_active()

Class Worker:
    Attributes:
        vertices // List of Vertex objects assigned to this worker

    Method compute_superstep(incoming_messages, beta, alpha):
        For each Vertex v in vertices:
            msgs = incoming_messages[v.id] if exists, else []
            v.compute(msgs, beta, alpha)
        Return all messages generated by vertices

Class PregelEngine:
```

```
Attributes:
    workers // List of Worker objects
    beta // Damping factor (e.g., 0.85)
    max_steps // Maximum number of supersteps (e.g., 100)
    tol // Convergence tolerance
    alpha = 1/|V| // Uniform teleportation factor

Method run(graph, bad_seed_list):
    Partition graph among workers
    For each Worker w:
        w.initialize_vertices(graph_partition, bad_seed_list)

    // Superstep 0: Seed initialization message passing
    incoming_messages = {}
    For each Worker w:
        For each Vertex v in w.vertices:
            If v.is_bad_seed:
                For each (neighbor_id, weight) in v.out_neighbors:
                    incoming_messages[neighbor_id].append(v.score * weight)

    For step from 1 to max_steps:
        all_outgoing = {}
        For each Worker w in parallel:
            outgoing = w.compute_superstep(
                          filter(incoming_messages for w), beta, alpha)
            Merge outgoing into all_outgoing

        If no messages in all_outgoing:
            Break // Convergence reached

        incoming_messages = all_outgoing

    // Post-processing: Normalize total trust score to 1.0
    total_score = sum(score of all vertices)
    For each Worker w:
        For each Vertex v in w.vertices:
            v.score = v.score / total_score

    Return final scores from all vertices
```

5 Implementation Analysis

5.1 Initialization and Score Assignment

- **Seed Initialization:** Known bad senders (from `bad_sender.csv`) are initialized with a score of $\frac{1}{|L|}$, while all other vertices start with 0.
- **Edge Normalization:** Outgoing edge weights for each vertex are normalized so that they sum to 1. This ensures that a vertex distributes its entire score among its neighbors.

5.2 Trust Propagation Dynamics

- In each superstep, vertices aggregate incoming messages (score contributions) and update their scores using:

$$\text{new\_value} = (1 - \beta)\,\text{bias} \;+\; \beta \times \text{incoming\_sum}.$$

5.3 Convergence Behavior

- The algorithm runs for a maximum of 100 supersteps. In our implementation, even after many iterations, the score of at least one vertex changed by more than a small tolerance, so all 100 iterations were executed.
- Convergence could be improved by adjusting the tolerance or better handling of dangling nodes (vertices with no outgoing edges).

5.4 Trust Mass Redistribution

- At termination, the total trust score summed to approximately 0.71485 rather than 1.0. This is likely due to sinks (dangling nodes) absorbing trust without redistribution.
- A post-processing normalization was applied to scale the scores so that they sum to 1.0, preserving the relative ranking.(which means redistributing the leftover mass uniformly to the known bad senders.)

5.5 Effect of Hyperparameters

- **Damping Factor (β):** Balances propagation from neighbors and uniform score injection. A value too high may overemphasize network structure, while a value too low may wash out differences.
- **Seed Score:** Uniformly initializing known bad senders ensures that the fraud propagation is biased toward the known fraud cluster.

6 Results and Visualization Interpretation

The algorithm produced several outputs:

- **Convergence Information:** The system printed "Superstep 1 completed" through "Superstep 100 completed." The algorithm ran for all 100 iterations before termination.
- **Total Trust Scores:** Before normalization, the total trust score was approximately 0.71485. After redistribution, it was normalized to 1.0.
- **Top Fraudulent Senders:** Table 2 shows the top 10 fraudulent sender accounts based on the final trust scores.

Top 10 Fraudulent Senders

Sender ID	**Trust Score**
1007	0.042789
1210	0.035751
1042	0.031054
1034	0.027101
1076	0.026253
1048	0.025059
1088	0.024917
1099	0.024852
1144	0.024496
1147	0.024412

Table 2: Top 10 Fraudulent Sender Accounts

Visualization Explanation

Three key plots were generated:

(a) **Log-Scale Histogram:** Displays the distribution of trust scores on a logarithmic scale. Most nodes have extremely low scores, with a long tail for nodes in the fraudulent cluster. This visualization highlights the separation between the vast majority of genuine nodes and the few high-score nodes.

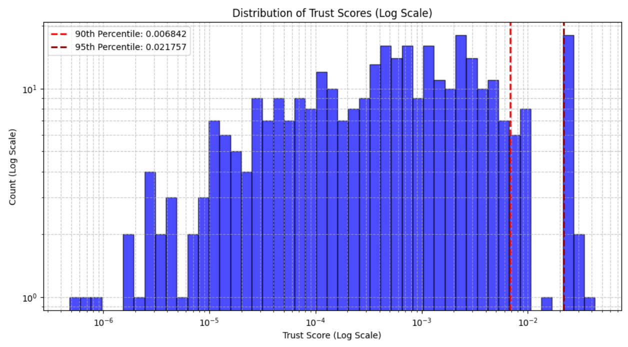

Figure 1: Histogram depicting the distribution of Trust Scores

7 Conclusion

In this report, we detailed the implementation of a Pregel-based TrustRank algorithm for fraud detection in a payment network. By employing an object-oriented, vertex-centric approach, we distributed the computation of fraud scores across the network in iterative supersteps. The algorithm leverages the idea that fraudulent nodes tend to link to one another, and by seeding known bad senders, we propagated this suspicion throughout the graph. Our analysis showed that even after 100 iterations, the trust scores required post-processing normalization due to score leakage to sinks.

References

[1] Gyöngyi, Z., Garcia-Molina, H., & Pedersen, J. (2004). *Combating Web Spam with TrustRank.* VLDB Endowment. `https://www.vldb.org/conf/2004/RS15P3.PDF`

[2] Malewicz, G., et al. (2010). *Pregel: A System for Large-Scale Graph Processing.* Proceedings of ACM SIGMOD. .