Indian Institute of Technology Hyderabad

Fraud Analytics (CS6890)

Assignment : 2 | Implementation of Trust rank using Pregel framework

Name	Roll Number
Manan Darji	CS22MTECH14004
Dhwani Jakhaniya	CS22MTECH14011
Ankit Sharma	CS22MTECH12003
Vishesh Kothari	CS22MTECH12004
Jayanti Mudliar	CS22MTECH14001

Contents

1 | Problem statement

- Implementation of Trust rank using **Pregel Framework** on Iron dealers data set.
- So, we have some Bad dealer data and a List of transactions between all dealers, including the bad dealers.
- Our Goal is to give each dealer a Trust rank(Or Bad Rank) based on their transaction with the bad dealers.

2 | Description of the data set

There are two files provided. Iron_dealers_data.csv and bad.csv

Iron Dealers data file contains 1,30,535 rows and three columns: Seller ID, Buyer ID, and Value. Each of the 1,30,535 rows in Iron Dealers data is a transaction of some Value(column 3) between the given Seller(column 1) and Buyer(column 2).

Bad CSV contains a list of 20 node ids given as malicious/bad dealer nodes. There are 799 dealers, of which 20 we are given as Malicious/Bad dealers.

Out of 1,30,535 transactions, there are **5358 Unique Transaction pairs**(pairs of dealers dealing with each other).

In Figure 2.1, we plotted this graph with 799 nodes and 5358 edges, and we can see that not all nodes are connected. That means a few nodes are not connected to other nodes (no directed outgoing edges). We have to take care of this in our implementation.

By analyzing the data, we can see a maximum transaction of 2,12,40,000 and a minimum transaction of 10,006.

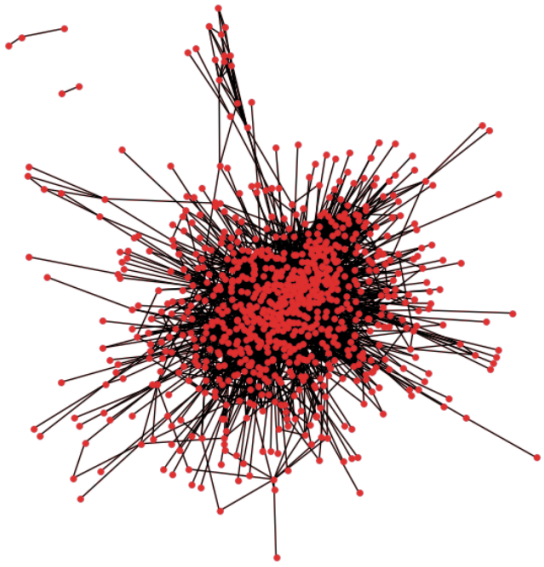


Figure 2.1: Visualization of Data set

3 | Algorithm Used

3.1 | Related Topics

3.1.1 | Pregel

Numerous computing issues of practical significance involve massive graphs. Processing such massive graphs is tedious since they can account for billions of vertices. As a result, Pregel is developed to address the processing of such massive graphs. This framework was developed by Google in 2010 based on the Bulk Synchronous Parallel (BSP) model. Large-scale graphs that cannot fit into the memory of a single machine are distributed across a cluster of machines. The framework will then divide the graph into smaller subgraphs, known as vertex partitions, and allocates each partition to a worker machine. The worker machines will then operate in a parallel and synchronous manner to execute the graph algorithms and exchange messages with the neighboring partitions until the algorithm converges.

3.1.2 | PageRank

PageRank is an algorithm that evaluates the significance of web pages in search engine results. It assigns a numerical value called PageRank score based on the quantity and quality of links it receives from other

web pages. This page rank score decides the order in which web pages appear on search results. The main principle is that the greater the links from other pages, the more important the page is. However, here, we are not employing the concept behind PageRank but something influenced by its principle, TrustRank.

3.1.3 | TrustRank

The main aim of TrustRank is to distinguish between fraudulent and trustworthy web pages. The principle behind this algorithm is based on the transitivity of trust. When trustworthy web pages link, they form a network of dedicated pages. Assuming a limited number of reliable seed pages, one could begin with sources such as government websites or academic institutions. The algorithm starts by exploring the web's link structure, tracking links from the seed pages to other pages, and evaluating them based on their distance from the seed pages to calculate a trust score.

In our current problem statement and dataset, we are not strictly utilizing TrustRank but rather its underlying principles. The scenario involves identifying fraudulent iron suppliers, thus requiring the propagation of negative or "bad" scores instead of positive ones.

3.2 | Approach we followed

3.2.1 | Generating the graph from dataset

To begin the implementation of this algorithm, we first need to load the dataset, focusing on the first two columns containing the seller ID and buyer ID. Upon examination, we can determine that there are a total of 799 unique nodes, of which 20 are given as malicious. There are a total of 1,30,535 transactions in the dataset. To account for the possibility of multiple directed edges between nodes, we can consolidate them and transform the multi-directed graph into a single-directed graph. This consolidation process results in a graph of 5,358 edges, where the weight of each directed edge represents the total sum of directed transactions between any two given nodes.

Upon analyzing the graph, we may observe certain nodes that lack outgoing edges, indicating that they are not involved in any selling action. However, the current implementation of the Pregel framework provided by sir cannot handle this scenario and may result in leakage problems. We can add new edges from all such nodes to the set of bad nodes to resolve this issue. This addition of edges allows for the propagation of bad scores to all the bad nodes, and we give equal weights to all the bad nodes. So, adding edges to all such nodes will add 1,920 more edges, which will sum up to 7,278 total edges in our graph.

3.2.2 | Trust Rank Vertex Update Algorithm

Based on class discussions and some analysis, we have set the initial value of all the bad nodes to 1 / number of bad nodes and all the good nodes to 0. We have followed this approach as we want the bad scores to be propagated only to the nodes that are somehow in relation to such bad nodes.

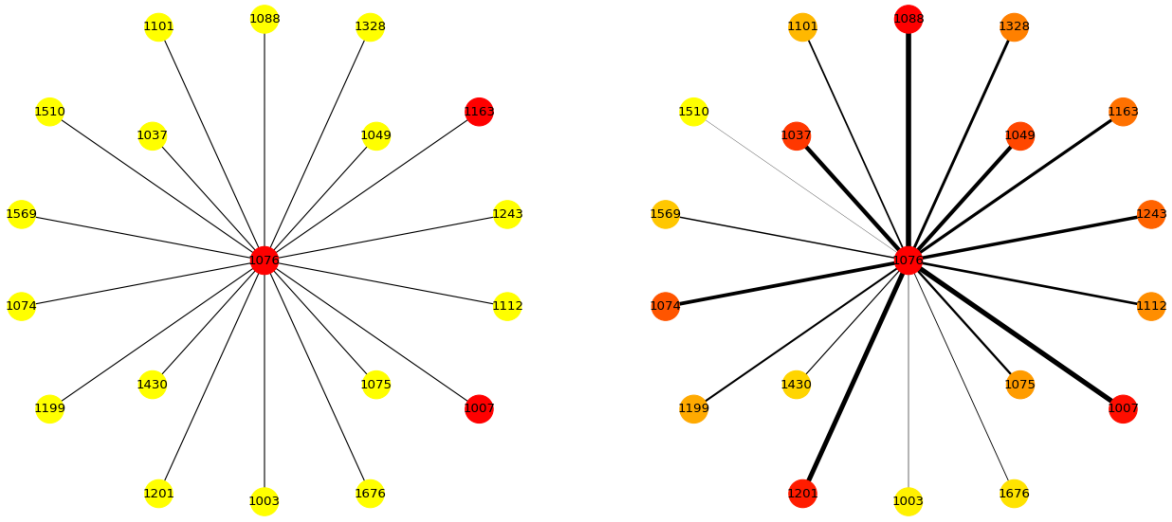
We will create a trust rank vertex for every node in the graph and use the 7,278 edges we have to update the outgoing vertices of each node. Then, we will run a trust rank algorithm for 50 iterations with a damping factor of 0.85 to reproduce bad scores for each node in the graph, and the results can be seen in later sections.

To implement the Bad score propagation variant of TrustRank, we need to create a TrustRank class by extending the Pregel framework and defining a custom update function. **Algorithm 1** provides the pseudo-code for this update function. Unlike the PageRank update function, the major difference is that we do not distribute bad scores equally to all outgoing vertices. Instead, we distribute the bad scores to each outgoing vertex based on a fraction of the transaction they are involved with that current vertex. Figure 3.1 clearly illustrate this concept.

Algorithm 1 Trust Rank Vertex Update Algorithm

Input: incoming_bad_scores list from previous time step, BAD dealer Ids, Damping factor (β), iterations
Output: outgoing_bad_scores from current vertex to all outgoing vertices

if $superstep < iterations$ **then**
  current bad score $= \sum incoming\_bad\_score$
  if current node is bad node **then**
    $value = (1-\beta) * \frac{1}{\text{Number of bad node}} + \beta *$ current bad score
  else
    $value = \beta *$ current bad score
  end if
  Total_Amount$= \sum outgoing_vertices.Transections_Amount$
  for all $V \in outgoing\_vertices$ **do**
    $outgoing\_bad\_score = value * \frac{V.Transections\_Amount}{Total\_Amount}$
  end for
end if



(a) image a **(b)** image b

Figure 3.1: Graph Comparisons

Figure 3.1a we observe a bad node 1076 connected to a few nodes in the graph. And in Figure 3.1b, we can see that the bad score has been distributed based on the transaction value with that node. Node 1510 has a lower bad score than 1007 as it has less fraction of the total transactions with the current node. This amount of transactions is represented as edge weights in this figure.

4 | Results

4.1 | Bad scores and Bad ranks plot

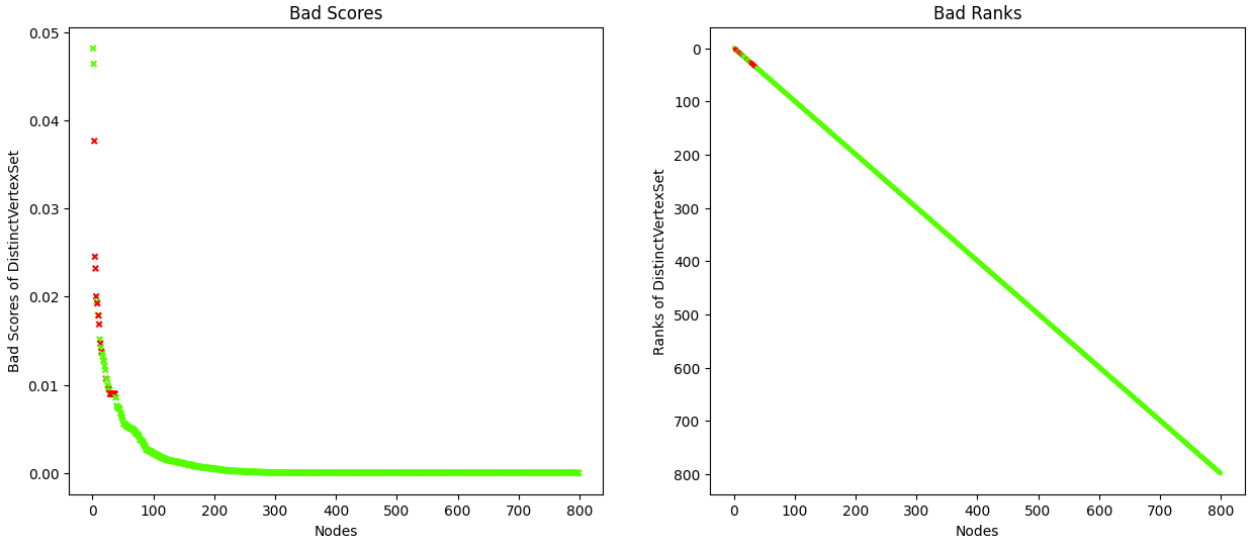


Figure 4.1: Final Bad scores and Bad ranks

In Figure 4.1 on the left-hand side, we have plotted the sorted bad scores of all the nodes in the graph. It is evident from the graph that bad nodes have a higher bad score, but there are other nodes as well with high bad scores. These nodes are likely the ones that are involved in direct or indirect transactions with the bad nodes, and there is a possibility that they may also be malicious. On the right-hand side, we have plotted the bad ranks of all the nodes, and it is apparent that all bad nodes have higher bad ranks, as expected.

4.2 | Histograms of Bad scores

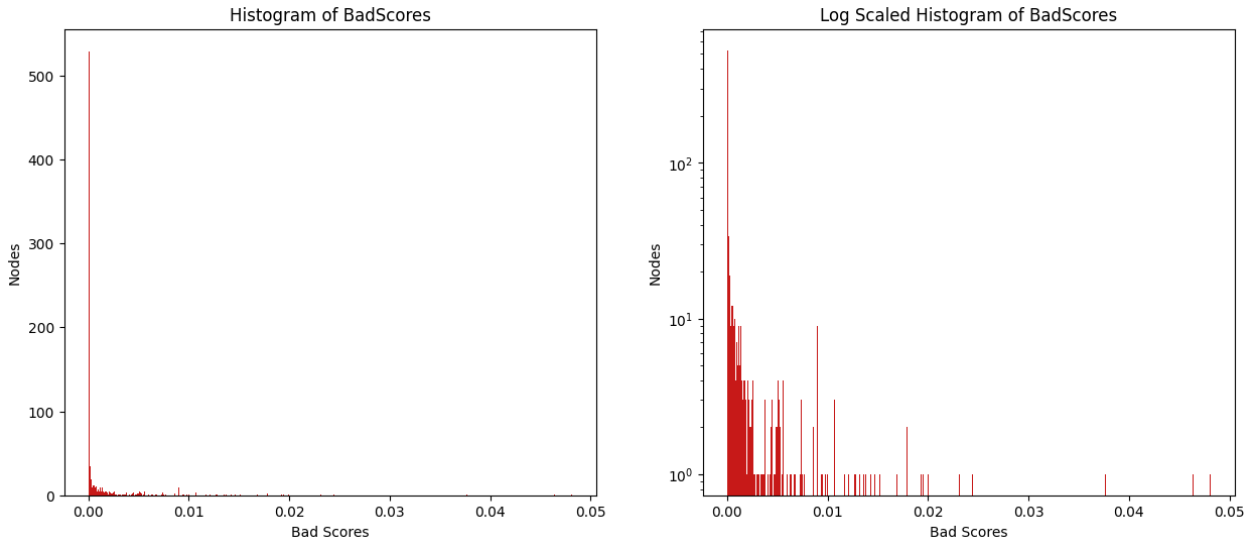


Figure 4.2: Final Bad scores and Bad ranks

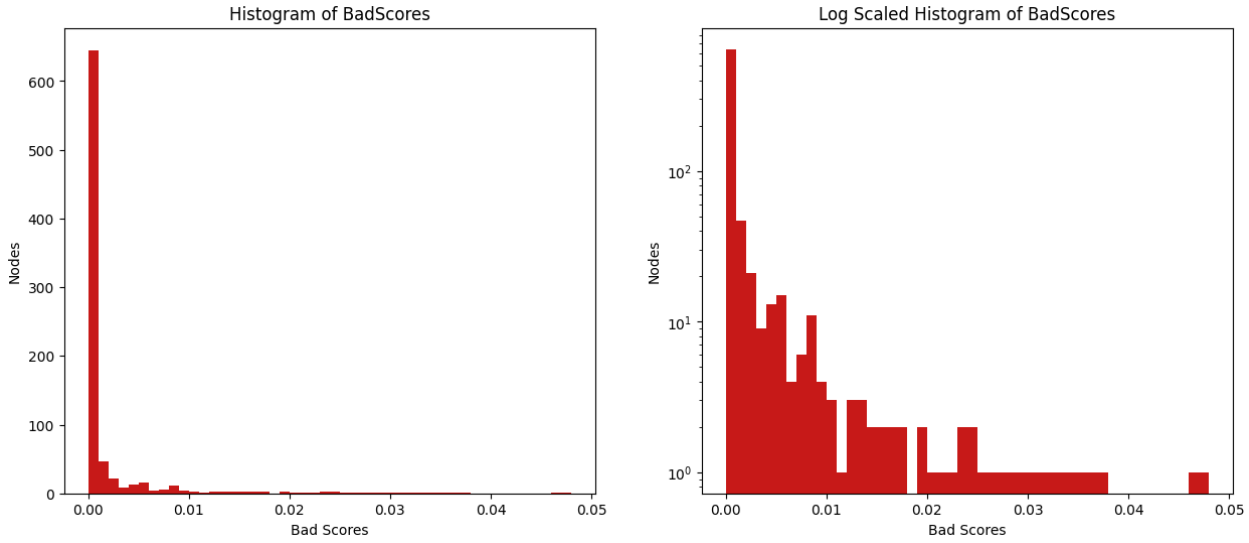


Figure 4.3: Final Bad scores and Bad ranks

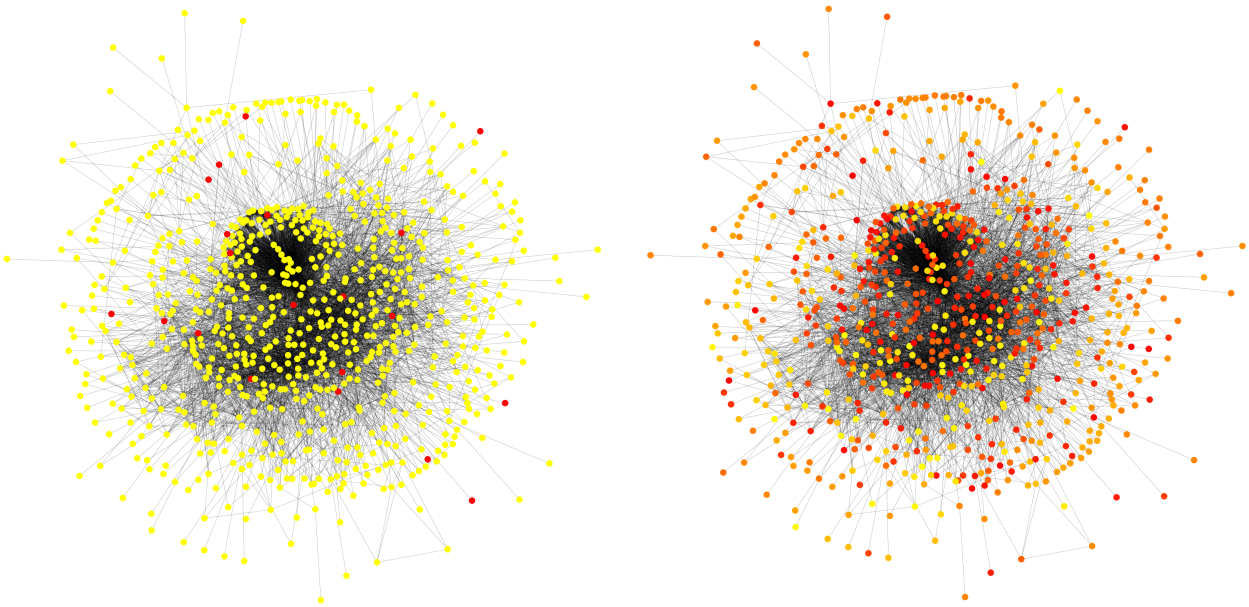
So as discussed in class, we have also provided the histograms for the bad scores. We have plotted the Normal scaled Histogram in Figure 4.2 on the left-hand side, but as it is hard to observe the results, we have plotted the log-scaled Histogram, which clearly shows that there is a higher frequency of nodes with less bad scores and very few nodes with bad high scores, as expected.

In Figure 4.3, we have rounded the bad score to four fractional values and combined the bins to visualize the histogram more clearly. This will help observe the decreasing frequency trend with higher bad scores.

4.3 | Graph base result Comparisons

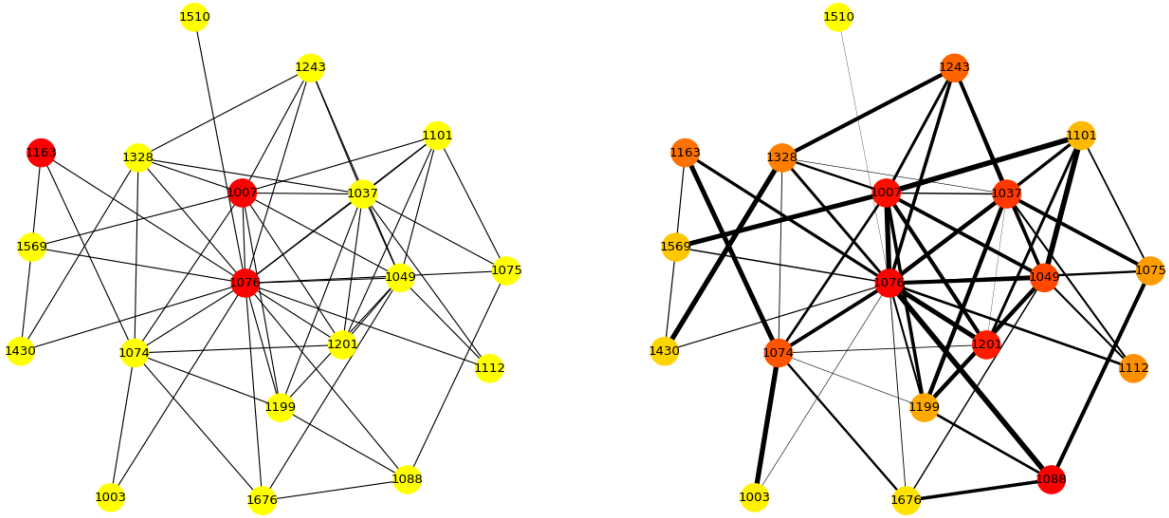
We can see that 20 nodes with the color red are the bad nodes, whereas the yellows are normal nodes. We can also see the Figure 4.4b as a scenario after the bad score propagation, but in a graph this large, it is hard to discuss any results, so we have taken a subset of the graph in Figure 4.5.

Figure 4.5 depicts the subset of the original graph with few bad nodes and few of the normal nodes, and as expected, the nodes with higher transaction values with the bad nodes have higher bad scores. This amount of transactions is represented as edge weights. For example, node 1510 has a lower bad score compared to 1007 as it has less fraction of the total transactions with the current node.



(a) image a **(b)** image b

Figure 4.4: Graph Comparisons



(a) image a **(b)** image b

Figure 4.5: Graph Comparisons