Computation of Trust Rank using Pregel Framework

Anirudh Raghav CH19BTECH11033 | Anushka Khare CH19BTECH11029 | Tanmay Goyal AI20BTECH11021 | Tanay Yadav AI20BTECH11026 | Vedika Verma CS19BTECH11057

Abstract

Web spam pages use various techniques to achieve higher-than-deserved rankings in a search engine's results. While human experts can identify spam, it is too expensive to manually evaluate a large number of pages. Thus, we use an iterative approach to ranking pages on the World Wide Web called the PageRank *algorithm. This approach can further be extended to compute the reliability of a page, using an algorithm called as the* TrustRank *algorithm. For computing this, we use* Pregel*, a framework for distributed graph parallel computation developed by Google.*

1. Problem Statement

Web spam pages use various techniques to achieve higher-than-deserved rankings in a search engine's results. While human experts can identify spam, it is too expensive to manually evaluate a large number of pages. Thus, we use an iterative approach to ranking pages on the World Wide Web called the *PageRank* algorithm. This approach can further be extended to compute the reliability of a page, using an algorithm called the *TrustRank* algorithm. For computing this, we use *Pregel*, a framework for distributed graph parallel computation developed by Google. This TrustRank methodology will make use of the biased PageRank Algorithm, which assigns a different static score to each trader depending upon how reliable they are.

2. Dataset

The given dataset consists of Iron dealers, where each row consists of one invoice. Each row consists of the Seller ID, Buyer ID, and the value involved in the transaction. Together, the dataset consists of 130535 rows of invoices and involves a set of 703 unique Sellers and 371 unique Buyers. The number of Buyers being almost half the number of Sellers raises suspicion, which is further aggravated by the number of invoices. We also have a list of 20 traders, known to be bad traders. Thus, based on this information, we wish to compute the Bad Scores for each of the traders and find other suspicious traders as well.

	Seller ID	Buyer ID	Value
0	1309	1011	1225513
1	1309	1011	1179061
2	1309	1011	1119561
3	1309	1011	1200934
4	1309	1011	1658957
...	...	...	...
130530	1344	1390	212390
130531	1914	1390	28739
130532	1914	1390	46861
130533	1914	1390	10585
130534	1914	1390	14972

Figure 1. A snapshot of the dataset

3. Algorithm and Methodology

To implement the Node2Vec Algorithm, we need to model all the transactions as a graph. In light of this, we take the following steps:

1. For convenience, we first map all the traders to unique integer keys and replace our original data frame with this new mapping. In light of this, we make two dictionaries, *integer_to_trader*, to hold the mappings from each integer to each trader, and *trader_to_integer* to hold the reverse mappings from each trader to each integer. Each of these can then be used as per convenience.

2. We make two dictionaries: *sell_buy* to hold the unique seller-buyer pairs for each transaction and *transactions* for the transaction value for each invoice. Thus, we have two graphs here, one to model the seller-buyer relationship regardless of the number of transactions or the amount of the transactions. And one to model all the information about the transactions.

3. The Pregel framework has an inherent disadvantage; it cannot handle nodes that have no outgoing edges. Thus, to overcome this, we find the list of nodes with no outgoing edges, i.e. have an outgoing degree of 0. These nodes can be easily found using the fact that these nodes are not buyers. Thus, we can take a set

difference between all traders and buyers. These nodes with an outdegree of 0 are then mapped to each bad node with an equal weight of 1.

4. We also read in the bad traders and converted them using our newly converted mapping.

5. We now convert the graph *transactions* from a multigraph to a weighted directed graph. For this, we sum up all the edges between two nodes, i.e, the final edge $e(i \to j)$ can be written as:

$$e(i \to j) = \sum_i e_n(i \to j)$$

where e_n is the n^{th} edge between i and j.

6. We now make a new class *TrustRankVertices*, inherited from the class *Vertex*, defined in the file *Pregel.py*. This class is used for parallel computation over distributed graphs by modeling a score propagation algorithm as messages being sent by one node to all its connected nodes through its outgoing edges. We use this to propagate the Bad Score across all nodes. The *TrustRankVertices* class takes the following as arguments for its constructor:

   - Id of the node
   - Value or Score for the node
   - All the instances of the class *TrustRankVertices* for the vertices connected to the node by an outgoing edge
   - the weights of the outgoing edges
   - If the node is a bad node
   - The total number of bad nodes in the graph
   - The damping factor which we choose to be 0.85.
   - The number of iterations for the Algorithm

   The detailed working of this algorithm will be explained later.

7. We initialize the initial trust rank for each of these traders given by: Initial Trust Rank = $\begin{cases} 0 & \text{if the trader is a good trader} \\ \frac{1}{N} & \text{otherwise} \end{cases}$
   where N is the number of bad traders.

8. We now create instances of the class *TrustRankVertices*. While creating these instances, care has to be taken to store the instances of the class *TrustRankVertices* for the vertices connected to the node by an outgoing edge, and not the vertices themselves. We also store the weights of these outgoing edges(note that now there is only one edge between a pair of nodes).

9. Finally, we pass this all to the Pregel framework and allow it to update the scores.

10. These scores are then written to the output file and are plotted to check the distribution of the Bad Scores.

We now explain the PageRank and TrustRank Algorithm in greater depth: We first define the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of all edges. We define an edge $p \to q$ as an edge from p to q. We now define the Transition Matrix T to be

$$T(p,q) = \begin{cases} 0 & p \to q \notin \mathcal{E} \\ \frac{1}{\omega(q)} & p \to q \in \mathcal{E} \end{cases}$$

where $\omega(.)$ represents the outdegree of a node.

We define the page rank R for a page p as:

$$R(p) = \alpha \sum_{q:q\to p \in \mathcal{E}} \frac{R(q)}{\omega(q)} + (1-\alpha)\frac{1}{N}$$

where α is the damping factor and N is the total number of pages. This can be explained as follows: A page should get a higher rank if it is being referred by a page with a higher rank. Also, the fewer pages this page refers to, the more important these references become. However, some pages might never be referred. This is where we assign a static score to each page, which is explained by assuming some user might randomly surf the internet. We assume this to be a uniform distribution; hence, each page gets an equal static component.

Thus, in matrix form, we can write this as:

$$\mathbf{R} = \alpha.\mathbf{T}.\mathbf{R} + (1-\alpha).\frac{1}{N}.\mathbf{1}_N$$

TrustRank uses a biased PageRank algorithm where the static components of the score might not be equal for all pages. That is, it uses a vector $\mathbf{d}$ such that $\sum_i d_i = 1$ and

$$\mathbf{R} = \alpha.\mathbf{T}.\mathbf{R} + (1-\alpha).\mathbf{d}$$

Note that for Pregel, we do not have to worry about the transition matrix T. This is taken care of by the framework itself, which propagates the score to all the vertices connected by an outgoing edge.

We shall explain the TrustRank Algorithm using this problem, where we compute the Bad Score for each trader instead. We assume the Bad Score vector $\mathbf{t}$ to be given by:

$$t(i) = \begin{cases} 0 & i \notin \text{ Bad Traders} \\ \frac{1}{N} & \text{otherwise} \end{cases}$$

Where N is the total number of Bad Traders. Note that this initial Bad Score vector also acts as our static distribution.

Thus, the static component of this Bad Score for trader i is given by $(1-\alpha) \times \mathbf{t}(i)$. The rest of the score is contributed by the incoming edges to the node, damped by the damping factor.

There are two approaches for the propagation of this score to the next node. Each of these approaches is explained below:

1. Dampening: The scores are reduced by a factor of β as we move away from the node.
2. Splitting: Here we split the scores. The logic is that fewer the pages a node refers, the higher should be the trust in them.

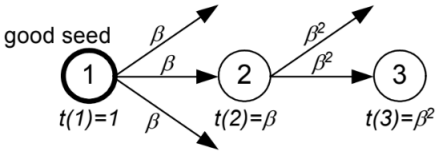

Figure 3: Trust dampening.

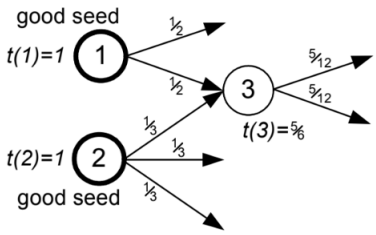

Figure 4: Trust splitting.

Figure 2. Methods of Trust Propagation

We use an alternative approach, which is similar to splitting. We split the scores proportional to the weights in the edge. Here, the weights refer to the sum of all transaction values between two traders. What this means is that the higher is the edge weight, i.e the amount of money moved between the two traders is very high, which could imply a large number of transactions between the traders, the more suspicious the trader is. Thus, it should get a larger portion of the Bad Score that is being propagated. This approach propagates the Bad Scores from one trader to the next.

4. Results

The Bad Scores are given in the output file *Results_bad_score.csv*. The results are shown below as well:

The top 20 bad traders with their bad scores are:

	Trader ID	Bad Score
87	1088	0.048159
141	1144	0.046401
6	1007	0.037647
203	1210	0.024521
33	1034	0.023195
38	1039	0.020017
10	1011	0.019433
41	1042	0.019227
85	1086	0.017886
75	1076	0.017849
290	1309	0.016856
93	1094	0.015157
144	1147	0.014666
167	1173	0.014256
97	1099	0.013732
194	1201	0.013540
119	1122	0.013195
78	1079	0.012787
135	1138	0.012648
40	1041	0.012113

Figure 3. Top 20 traders sorted by their Bad Scores.

We also obtain a histogram showing the distribution of the Bad Scores and the number of traders associated with it. This shows that only few of the traders involved in all the sales are suspicious.

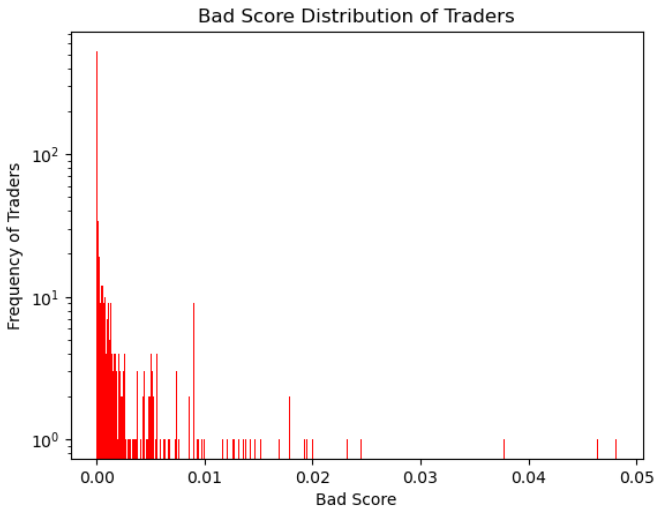

Figure 4. Distribution of Bad Scores output by our algorithm