Implementation of TrustRank using Pregel Framework

Aman Panwar
CS20BTECH11004
Department of Computer Science

Amulya Tallamraju
AI20BTECH11003
Department of Aritificial Intelligence

Pranav K Nayak
ES20BTECH11035
Department of Engineering Science

Taha Adeel Mohammed
CS20BTECH11052
Department of Computer Science

Vikhyath Sai Kothamasu
CS20BTECH11056
Department of Computer Science

I. Introduction

In this report, we attempt to document our implementation of the TrustRank algorithm by using the Pregel Framework

II. Problem Statement

A. TrustRank

TrustRank is a Search Engine Optimization (SEO) algorithm developed by researchers at Stanford University in 2004. It is designed to combat the manipulation of Search Engine Results Pages by spam websites. These websites will often attempt to trick search engines and boost their positions search results. For example, a commercial website attempting to exploit SEO might populate its homepage with thousands of keywords, allowing it to be pulled up as a result on searches that aren't relevant to the website itself. It can hide this text from users visiting the website by manipulating color schemes and backgrounds. As a result, the web page is displayed as a result for multiple different types of searches.

As another example, search engines will often use the number of incoming links into a web page as a sign that it should be placed higher up in search results. Spammers can then create thousands of bogus web pages, and have links in them that point to the spammer's main web page. This will cause it's rank in the SERP to undeservedly get boosted.

This is where TrustRank comes in. It assigns trust scores to a web page that are used when considering the page's position in a SERP. It does this by assigning a few 'seed' pages, and propagating good scores to those pages that are in proximity to the seed ones.

The algorithm also evaluates the quality of content on a web page, determining if the content on the web page is meets a certain standard. If it does, its score is boosted, while if the content is determined to be substandard, such as spam websites that have multiple links to low score pages, its score is correspondingly lowered.

B. Pregel

Pregel is a large scale graph processing framework, first publicized in a paper by Google in 2010. It is a framework designed for large scale graph processing. Each Pregel computation consists of a series of iterations, called supersteps. Each superstep consists of a computation phase and a communication phase. In the computation phase, each vertex can perform a computation based on its state and the messages it has received from other vertices. In the communication phase, each vertex can send messages to other vertices. The computation and communication phases are repeated until the algorithm terminates.

Pregel is capable of operating on such large graphs by partitioning the graph into smaller subgraphs, and distributing them across multiple machines. Each machine is responsible for a subset of the vertices and edges of the graph. The machines then communicate with each other to perform the computation.

III. Description of the dataset

For this assignment, we were given the same dataset as Assignment 1: Circular Trading detection. It consists of 130,535 rows, each row consisting of three column values. The first two contain IDs for entities within the dataset, with the third column containing a number representing the value of the transaction between them.

We apply TrustRank to this dataset to give scores to all the dealers in the dataset, given a small collection of bad sellers. This score is an indicator of the trust given to each seller by the algorithm, with higher scores indicating less trustworthy sellers.

IV. Algorithm Used

We implement TrustRank here to allow us to quantify how trustworthy dealers are, given a small collection of dealers we know are bad. This is analogous to how TrustRank is implemented for websites, except while the 'seed' pages for the web are trusted pages with a high score, the 'seed' dealers for our dataset are untrustworthy dealers with a high score. The algorithm then propagates this score to other dealers in the dataset, with the score decreasing as the distance from the seed dealers increases. A lower score thus indicates a more trustworthy dealer.

A. Our Implementation in Detail

We begin the discussion of our implementation by going over our class structure:

- `Vertex`: Represents a single node in the graph. It stores the ID, current score,the adjacency list of its outgoing vertices, a list of incoming weights, and a list of outgoing weights
- `Pregel`: Represents a single run of the Pregel system. Contains functionality to run an instance, perform a superstep, partition the vertices, return the ID of the process assigned to a vertex, and check whether vertices are active or not.
- `Worker`: A single thread in the Pregel system, responsible for handling a single superstep for a subset of the graph.
- `TrustRankVertex`: A TrustRankVertex represents a node in the graph. It stores its id, its current score, and the adjacency list of its outgoing vertices, along with the values incoming into it and outgoing from it during the current superstep. Contains functionality to perform updates on its score from incoming edges.

The above are classes that behave as the workhorses of our implementation, performing TrustRank and propagating scores across the graph. We also have a `Graph` class, that represents the graph on which TrustRank acts. We initialize a trust score of $\frac{1}{\text{number of bad nodes}}$ to each of the initial nodes before performing any computations. We also define the following functions to run TrustRank:

1) `pregelTrustRank`: This function runs TrustRank through the Pregel system on our graph. The definition of the function can be found in Figure 1.

```
def pregelTrustRank(vertices, num_workers = 20):
    """ Compute Pagerank using pregel by assisstng set of vertices to thread """
    p = Pregel(vertices, num_workers)
    p.run()
    return mat([vertex.score for vertex in p.vertices.values()]).transpose()
```
Python

Fig. 1. The pregelTrustRank function

2) `trustrank_test`: This function runs a standard matrix-theory based implementation of TrustRank on our graph for comparison with the Pregel run. The definition of the function can be found in Figure 2.

```
def trustrank_test(vertices):
    """Computes the trust rank vector associated to vertices, using a
    standard matrix-theoretic approach to computing trustrank.  This is
    used as a basis for comparison."""
    num_vertices = len(vertices)
    I = mat(eye(num_vertices))
    G = zeros((num_vertices, num_vertices))

    # As the node id's can be larger than the number of nodes, we need to create a mapping from node id to ind
    index = {}
    for i, vertex_id in enumerate(vertices):
        index[vertex_id] = i

    for vertex in vertices.values():
        for (out_vertex_id, weight) in vertex.out_vertices.items():
            G[index[out_vertex_id], index[vertex.id]] = weight / vertex.total_out_weight
    P = (1.0/num_vertices)*mat(ones((num_vertices,1)))

    return 0.15*((I-0.85*G).I)*P
```
Python

Fig. 2. The trustrank_test function

V. Results

We run our `main` function to output the 20 cells with the highest TrustRank score, when using the Pregel based implementation. These 20 are the "worst" nodes in the network. For comparison, we also include the top 20 computed by the matrix-theory based approach, from which one can see that both give identical results.

```
Computing trust rank using pregel

Top 20 nodes the highest trust rank using pregel (Most probability of being bad nodes)
(1088, 0.03657394430976734)
(1144, 0.03568251444266552)
(1007, 0.024310801362824052)
(1094, 0.011916677916890613)
(1201, 0.011274153810394198)
(1173, 0.01099958784091173)
(1122, 0.010519572726655314)
(1041, 0.009535834346174547)
(1138, 0.00727508762119577 75)
(1050, 0.007195702390225238)
(1043, 0.00704659710765 9652)
(1381, 0.006982794426377115)
(1330, 0.006232833503896238)
(1038, 0.005898860691832913)
(1319, 0.005723832233821227)
(1021, 0.005681554371193211)
(1037, 0.005525714137937298)
(1283, 0.00541782432368 1727)
(1114, 0.005326895586388 6485)
(1084, 0.005034540511923777)
```

Fig. 3. The scores computed by the Pregel based implementation

```
Computing trust rank using matrix multiplication

Top 20 nodes the highest trust rank using matrix multiplication
(1088, 0.036581349703505364)
(1144, 0.03569117473003544)
(1007, 0.024311474254747 11)
(1094, 0.011926585822627622)
(1201, 0.011274418292244586)
(1173, 0.011010137477285702)
(1122, 0.01052951821314483)
(1041, 0.009546351524028263)
(1138, 0.007275212076773389)
(1050, 0.007195938908518 1815)
(1043, 0.007050664215818462)
(1381, 0.006987545160100712)
(1330, 0.006237405575273865)
(1038, 0.005899050883764554)
(1319, 0.005723993805024334)
(1021, 0.005681731044805705)
(1037, 0.005525828994341 96)
(1283, 0.005417961002134398)
(1114, 0.005327000521648 48)
(1084, 0.005034675465716346)
```

Fig. 4. The scores computed by the matrix-theory based implementation

While in theory, the Pregel based implementation is supposed to be faster, we observe through noting down the execution times of the notebook's cells that the matrix-theory based implementation is, on average, 8 times faster. We speculate that this is because the overhead of the Pregel system is too large for the small size of our dataset, and the computing power used to manage all the threads is not compensated for. The matrix-theory based approach, while likely inefficient if applied on datasets of the size that Pregel is intended to be used for (i.e. billions of nodes), is much more efficient for our dataset.

In the following figure we have plotted the distribution of the TrustRank scores, on a logarithmic scale.

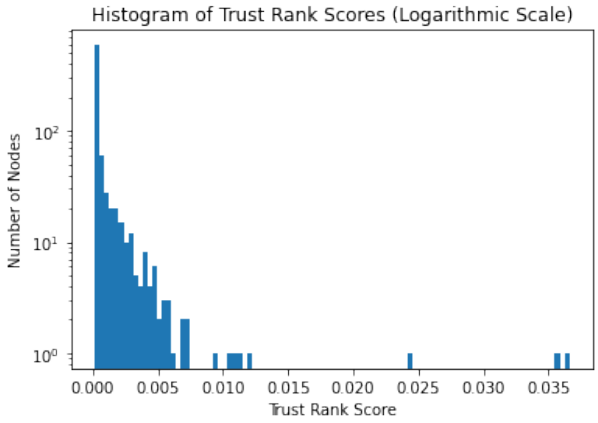

Fig. 5. The distribution of TrustRank scores

VI. References

- Adrian Colyer, ”Pregel: A System for Large-Scale Graph Processing
- Gyöngyi, Garcia-Molina, Pedersen, ”Combating Web Spam with TrustRank”