DS 210 Final Project Write-up – Garrick Zhang

https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html

For this final project, I used a dataset that contains trust scores that each user gives to another user during each transaction. The users represented nodes, and the transactions represented directed edges. The main idea I used for this project is Dijkstra's algorithm, which finds the shortest distance from one node to another. For this project, instead of finding the shortest distance, I wanted to find the path with the highest trust score. Eventually, I will calculate a comprehensive trust score for each user. This trust score is essential because while in the original data, each user may interpret trust scores differently, thus each rating trust score may have different weights. By using Dijkstra's algorithm, every user is going to have the highest possible trust score, and there will be more scores to compare for each user. In such a way, the data will be scaled into a similar weight, and it will be more useful to analyze.

However, I cannot directly use Dijkstra's algorithm. My trust score, which represents the distance, has a range of -10 to 10. To make it work, in my modified_dijkstra function, I will first make sure that all scores are positive (not zero), and invert it, so that the highest trusted score will actually be the smallest number. The shortest path will now actually be the highest trust score. The formula would be: “1/(score + 11)”, and the score will now be in the range of 0.05 to 1. While I don’t care about how many nodes each path has been through, I want to get a score that is from the total ‘distance’ of the path, divided by the amount of nodes that the path has passed through.

In the construct_graph function, it can output 3 generations of graphs, depending on the input command. The first generation will only have 1/3 of the data, and each newer generation will add 1/3 of the data based on the previous generation. In this way, each generation of graph will have a larger network than the previous one, and I can interpret the result based on the comparisons.

After creating the graphs based on HashMap, I looped through all the available nodes within the graph, and each time the node will be treated as a start node and will give the ratings to the rest of the nodes that were based on the modified Dijkstra function. The example output would be something like this:

```
Trust scores from node 1:
To node 585: 0.058333333333333334
To node 478: 0.05375417710944027
To node 590: 0.06666666666666667
.....and etc. (there are around 1500 of them in total for each start node)
```

For the example above, I would consider that the highest possible score that node 1 can give to node 585 would be 0.0583, which in reality would be a score around 6.

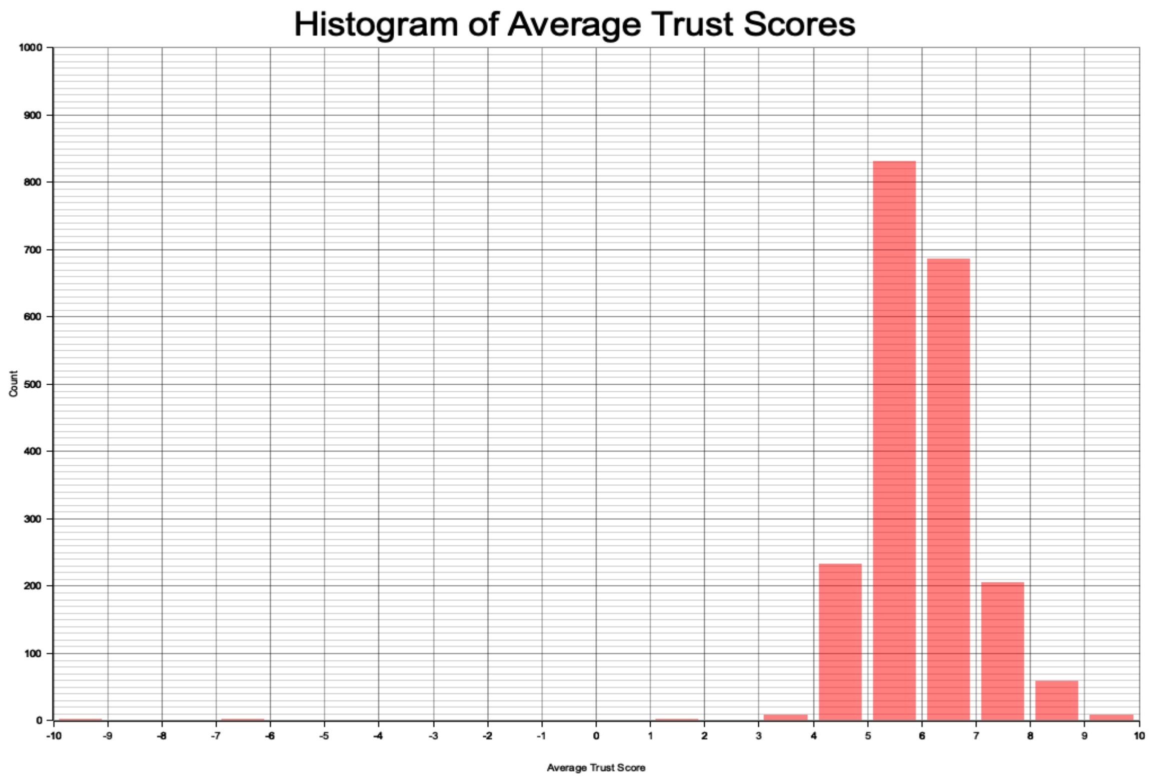
For the main.rs file, it will export all of these outputs into a CSV file. The index for each row of this file will be the ID of the nodes, and it will record all the highest possible scores that the rest of the available nodes can give to this node. For example, using the example I had above. The start node 1 is giving a score to node 590: 0.06667. So for the row of node 590, it will store 0.06667, and for the next loop maybe the start node is node 2, and node 2 may give a trust score of 0.090909 to node 590, and this score will also be stored to the row of node 590, and the rest will be done in the same way.

The presence of 'inf' (infinity) in my CSV file means that some nodes are not reachable from others, and my modified Dijkstra algorithm will record the distance to these nodes as infinity (f64::INFINITY).

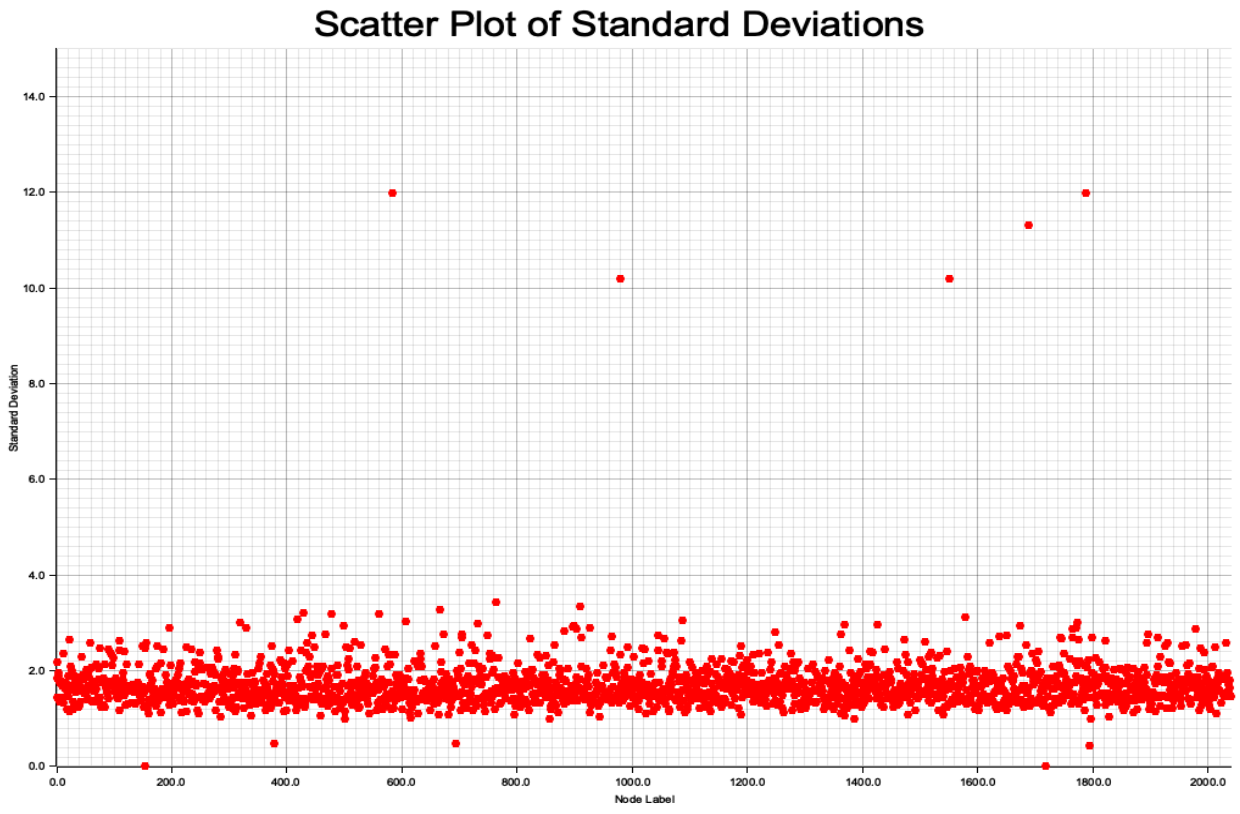
Based on the CSV file exported from the main.rs, I was able to calculate the mean trust scores for each of the nodes, and the standard deviations for each of the nodes. I put all the analysis into another file named "analysis.rs", and I can run it in the command line by typing: "cargo run analyze".

Before analysis, the read_and_process_csv function helps to read and preprocess the data, such as converting the scores back from range of -10 to 10, and filter data that are not available.

And using chat-gpt, I was able to create some visualization of these data.
Here is a histogram of the average trust score of each node (1st generation):

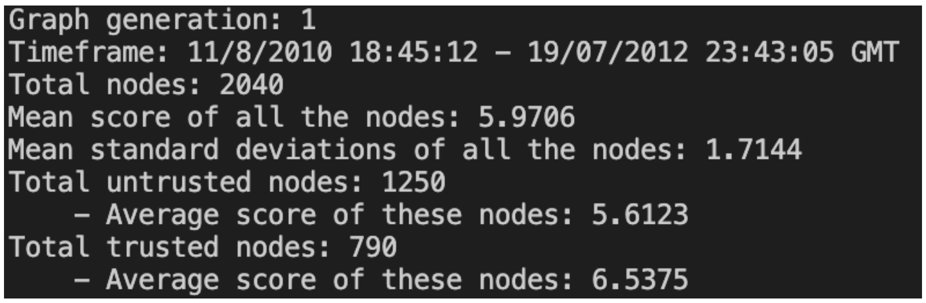


And also a scatterplot for the standard deviation for each of the nodes (1st generation):

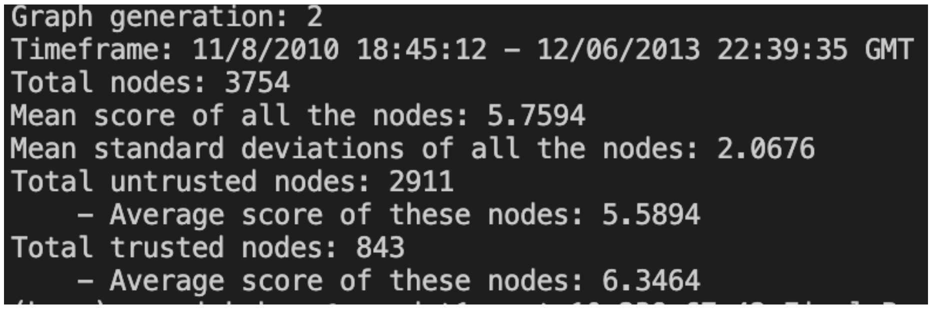


By observing the plot and the histogram from 1st generation, I manually set the threshold of std dev as 2.2 and the mean score of 6. For the trusted nodes, it has to be both above the mean score threshold and below the std dev threshold, meaning that it has an average of high trusted score while the variance is low to prevent the score anomaly and potential fraud. For the untrusted nodes, it would be either below the mean score threshold or above the std dev threshold. The reason I set thresholds only based on the data from the first generation is because this way, I would see a clearer difference in how many nodes are passing the thresholds in later generations.

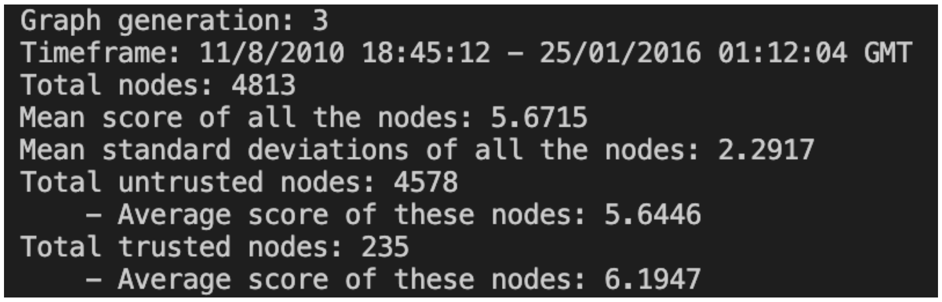
Here are my final outputs:



```
Graph generation: 1
Timeframe: 11/8/2010 18:45:12 - 19/07/2012 23:43:05 GMT
Total nodes: 2040
Mean score of all the nodes: 5.9706
Mean standard deviations of all the nodes: 1.7144
Total untrusted nodes: 1250
    - Average score of these nodes: 5.6123
Total trusted nodes: 790
    - Average score of these nodes: 6.5375
```



```
Graph generation: 2
Timeframe: 11/8/2010 18:45:12 - 12/06/2013 22:39:35 GMT
Total nodes: 3754
Mean score of all the nodes: 5.7594
Mean standard deviations of all the nodes: 2.0676
Total untrusted nodes: 2911
    - Average score of these nodes: 5.5894
Total trusted nodes: 843
    - Average score of these nodes: 6.3464
```



```
Graph generation: 3
Timeframe: 11/8/2010 18:45:12 - 25/01/2016 01:12:04 GMT
Total nodes: 4813
Mean score of all the nodes: 5.6715
Mean standard deviations of all the nodes: 2.2917
Total untrusted nodes: 4578
    - Average score of these nodes: 5.6446
Total trusted nodes: 235
    - Average score of these nodes: 6.1947
```

By observing these outputs from 3 generations, I realized that there were increasing amounts of untrusted scores and decreasing amounts of trusted scores. One way to interpret why this is happening is because the mean of the standard deviations is increasing in later generations, while I set the threshold of standard deviation based on the first generation. The result can help us to interpret that while some of the users may be trusted in a shorter period of time, they may be given more lower trust scores in the future. It also shows us that most users in Bitcoin OTC transactions are untrustworthy in the long run.

Outside resources:

- https://docs.rs/serde/latest/serde/trait.Deserialize.html
- https://docs.rs/csv/latest/csv/struct.Writer.html
- https://doc.rust-lang.org/std/cmp/enum.Ordering.html
- https://doc.rust-lang.org/std/env/index.html
- Chat-GPT for:
    - data visualizations
    - Initialize the dijkstra algorithm function, such as storing additional information like node counts and customizing the struct (state). I later modified it a lot to fit into my project's indications.