**Project Report**

**Title: Stress Centrality Algorithm**

Student 1 Name-ID: Sai Sumana Adurugatla-1002073766

Student 2 Name-ID: Perin Devi Behara-1002080027

**Problem Description:**

Implement the Betweenness Centrality's variation known as the Stress Centrality. Start with a brute-force approach and then improve on it. You have to implement your own algorithm. Compare the accuracy and efficiency with existing algorithms using any graph analysis package of your choice.

**Approach:**

* Stress centrality is calculated by considering the total number of shortest paths that traverse a node, regardless of their direction (incoming or outgoing).
* It sums the number of times a node lies on the shortest paths between all pairs of other nodes in the network.
* The computation of stress centrality often involves algorithms such as breadth-first search (BFS) to find shortest paths efficiently
* Take the data set in .CSV form
* Using BFS algorithm calculating number of shortest paths
* Counting number of shortest paths passing through node
* Starting with Brute-Force Approach and improving it further to implement Stress Centrality
* Comparing new algorithm with existing packages and calculating accuracy and efficiency

**Project setup:**

**1. Environment Setup:**

Using Google colab

Adding input files to folder and executing the code

**Dataset Used**:

We can generate the dataset using following code by giving number of nodes and edges required:

[Generate\_Nodes](https://colab.research.google.com/drive/1CWuVJhv4YYJ9xrDxDyv25mcA7-zpfCkz?usp=sharing)

Sample Dataset:

1000 nodes:[generated\_edges.xlsx](https://mavsuta-my.sharepoint.com/:x:/g/personal/sxa3766_mavs_uta_edu/EZdPvZY_e8FLnvFEMR6V1RwBoZxwiDSe4OIKtZ63mNTlgQ?e=yJleyP)

1500 nodes:[generated\_edges.xlsx](https://1drv.ms/x/s!ApYvaA9aDgOAkSeKMhSvj1eCeF1g?e=J3ZalY)

**Code Link:**

[CodeLink](https://colab.research.google.com/drive/1JQQto1fjBmTPSlhl9tGoFyLU2N700wrb?usp=sharing)

Stress\_centrality\_count\_nodes:

This function also calculates stress centrality, but it keeps track of the number of shortest paths for each node.

**Algorithm:**

**Initialization:**

Create list of nodes from the graph.

**Loop (for each source node):**

Iterate through each node in the graph as a source node.

Initialize data structures for current source node:

visited: Use dictionary to track visited nodes.

distance: Use dictionary to store the distance of each node from the source.

queue: A queue for BFS traversal.

count\_nodes: A dictionary to count the number of shortest paths passing through each node.

**Traversal:**

Enqueue the source node into the queue and mark it as visited.

While the queue is not empty:

Dequeue a node from the queue (current\_node).

For each neighbor of the current\_node:

If the neighbor has not been visited:

Mark the neighbor as visited.

Enqueue the neighbor into the queue.

Update the distance of the neighbor.

Update the count\_nodes for the neighbor.

If the neighbor is at the same distance as the current\_node:

Update the count\_nodes for the neighbor.

**Accumulation of Stress Centrality:**

For each node in the graph (excluding the source node):

Accumulate the count\_nodes values into the stress\_centrality dictionary.

**Return Result:**

Return the stress\_centrality dictionary.

**Time and Space Complexity:**

Outer Loop: *O*(*n*)

**Inner While Loop (BFS Traversal):** *O*(*m*+*n*)

**Accumulate Stress Centrality:** *O*(*n*)

New algorithm: *O*(*nm*+*n*2+*n*)

Time *O*(1)

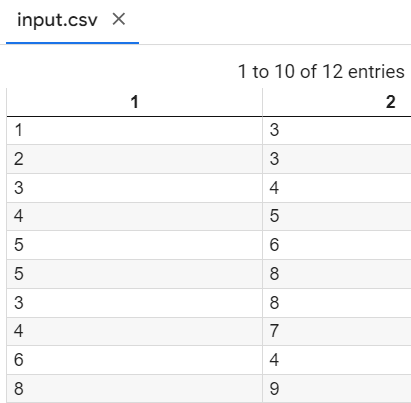
Accuracy verification: *O*(*n*)

space complexity for each function is O(n).

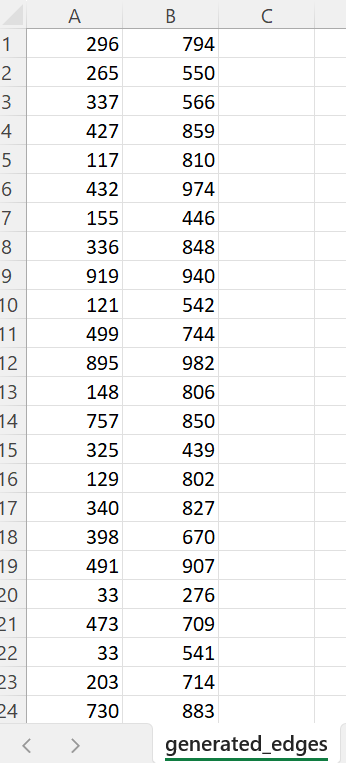
**Experimental results:**

**Input:**

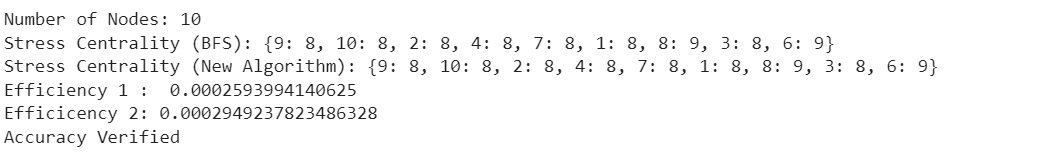
Csv file with node edges (a,b)-(source,target)

 A green circles with black lines and numbers

Description automatically generated



**Output for 10 nodes to check accuracy:**

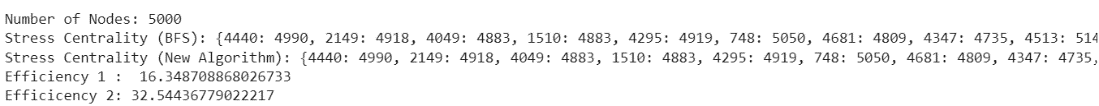


**Output for 1000 nodes:**

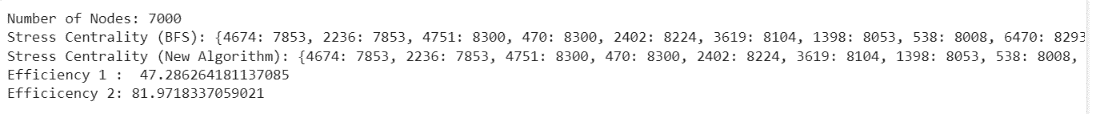
A close-up of a computer screen

Description automatically generated

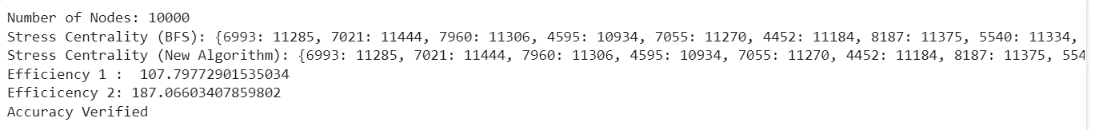
**Output for 5000 Nodes:**



**Output for 7000 Nodes:**



**Output for 10000 Nodes:**



**Challenges Faced:**

* Large Data sets results did not match the existing networkx package function results.
* What nodes to be considered (Is it only middle or first and last) for the Nodes with high stress centrality are essential for maintaining connectivity, and their removal could significantly impact the network's ability to transmit information efficiently.
* Selecting appropriate datasets and methods for accurate comparisons with existing algorithms.
* Interpreting the results of Stress Centrality is challenging when dealing with complex networks.

**Outcome of Project:**

A graph of efficiency and number of nodes

Description automatically generated

**Conclusion:**

In conclusion, the implemented Stress Centrality algorithm initially utilizing a brute-force approach, provides a foundational understanding of the algorithm's functionality. The algorithm reads input from a CSV file, representing an adjacency matrix, and calculates stress centrality values for each node in the graph. During basic implementation while faced some challenges with efficiency especially when applied to larger datasets due to its cubic time complexity.

For further improvement and optimization, considerations such as parallelizing shortest path calculations, optimizing memory management, and implementing sparse matrix handling techniques could enhance the algorithm's performance. The comparison with existing graph analysis packages, such as NetworkX, is essential for evaluating efficiency and accuracy. NetworkX library offers optimized algorithms implementation on larger datasets.

To ensure practical utility, extensive testing on datasets of varying sizes and densities is crucial. The implemented algorithm serves as a starting point, and refinement is necessary to address performance bottlenecks and enhance its applicability to real-world network analysis.

**References:**

<https://www.centiserver.org/centrality/Stress_Centrality/>

<https://symbio6.nl/en/blog/analysis/stress-centrality>

[https://networkx.org/documentation/stable/reference/algorithms/centrality.html#shortest-path-betweenness](https://networkx.org/documentation/stable/reference/algorithms/centrality.html)

[https://med.bioinf.mpi-inf.mpg.de/netanalyzer/help/2.7/index.html#stressDist](https://med.bioinf.mpi-inf.mpg.de/netanalyzer/help/2.7/index.html)

<https://www.youtube.com/watch?v=XEjFUY1hfN0>

https://stackoverflow.com/questions/44078279/networkx-stre...

<https://www.centiserver.org/centrality/Stress_Centrality/>