

The Report

Contributors: *Siran Xianyu, Jialong Li, Kaiyi Zhang*

Our presentation video: https://mediaspace.illinois.edu/media/t/1_wcm6edgz

1. Our Goal:

In this project, we want to find pivot schools in the collaboration networks in a specific academic field and then create a graph representing these collaboration relationships.

Firstly, we are going to use BFS to do the traversal. The input will be a list of nodes and the output will be a matrix representing the connected components while each line refers to one connected component. The time complexity will be $O(V+E)$, and the space complexity will be $O(V)$.

Secondly, we will use Brandes Algorithm to generate centrality for each node in the graph. For each connected component, we will obtain a node with most centrality in each connected component. A list of nodes of most centrality will be our output. The time complexity is $O(nm)$ and $O(n + m)$ is the space complexity, where m is the number of edges and n is the number of nodes. Define $\sigma(s, t)$ as the number of shortest paths between nodes s and t . Define $\sigma(s, t|v)$ as the number of shortest paths between nodes s and t that pass through v . Betweenness centrality is $CB(v) = \sum \sigma(s, t|v) / \sigma(s, t)$. This will be our main algorithm to find pivot schools (most influential) for each academic circle.

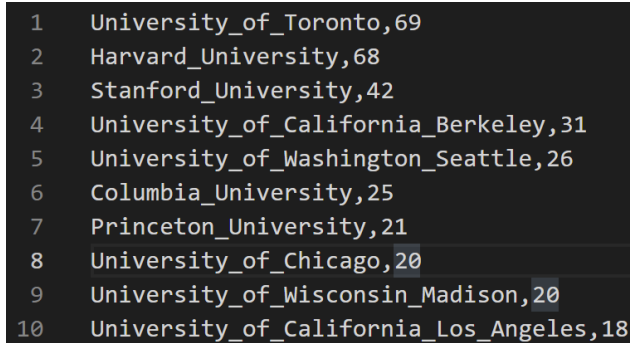
Lastly, we will use forced-directed graph drawing to draw a graph representing the relationships between school and their collaboration. We calculate attractive and repulsive forces between the nodes in the graph and use the net forces to update each node's position. The mass of a node is determined by betweenness centrality. We use the Fruchterman-Reingold algorithm to calculate these forces where the attractive force is inversely related to the square of the nodal distance and the repulsive force is inversely related to the nodal distance. Therefore, our input is a list of centralities and random 2-d coordinates, and our output is a graph with stable nodes, and the edges represent the relative distance according to the net force. The time complexity is $O(n^3)$ and space complexity is $O(n)$, where n is the number of nodes.

2. Pre-process: The Data

From the WOS database

(<https://www-webofscience-com.proxy2.library.illinois.edu/wos/woscc/advanced-search>), we use the search query “(WC=(Statistics & Probability)) AND (TMSO==(“9.92 Statistical Methods”))” and download the top 1000 cited papers. The original format of the data is .xsl. We used R code to do the data cleaning and data correction job and generated 2 new .csv files.

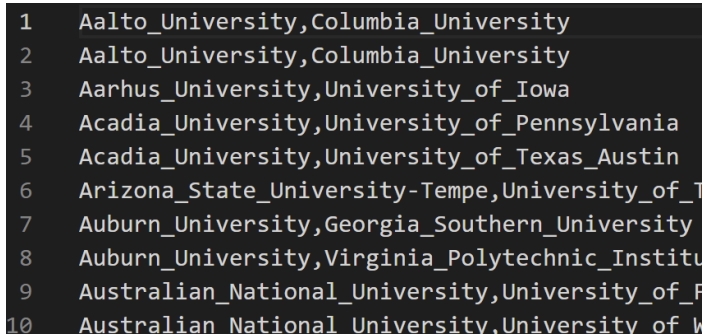
The first csv file is called “data_vertex_262” which stores 266 names of universities in the first column and their occurrence time in the second column.



1	University_of_Toronto,69
2	Harvard_University,68
3	Stanford_University,42
4	University_of_California_Berkeley,31
5	University_of_Washington_Seattle,26
6	Columbia_University,25
7	Princeton_University,21
8	University_of_Chicago,20
9	University_of_Wisconsin_Madison,20
10	University_of_California_Los_Angeles,18

[Figure 1. the screenshot of data_vertex_262.csv]

The second file is called “data_edge_617” which stores 617 pairs of universities. They are matched together because they once collaborated in a paper.



1	Aalto_University,Columbia_University
2	Aalto_University,Columbia_University
3	Aarhus_University,University_of_Iowa
4	Acadia_University,University_of_Pennsylvania
5	Acadia_University,University_of_Texas_Austin
6	Arizona_State_University-Tempe,University_of_T
7	Auburn_University,Georgia_Southern_University
8	Auburn_University,Virginia_Polytechnic_Institu
9	Australian_National_University,University_of_P
10	Australian_National_University,University_of_W

[Figure 2. the screenshot of data_edge_617.csv]

The data processing steps are pretty tricky. We mainly use “string split”, “Regex” to do that. You can check our “data_cleaning.rmd” for more information. In addition, we also use R to write test cases.

After that, we will create an unweighted, undirected graph using an adjacency list implementation.

3. the I/O stream

3.1 How to run our code

Run the code

...

make

./main

...

To get the "Betweenness_Centrality_Table.csv" and "" Graph.

Note: It usually takes 30-40 seconds.

3.2 How to run the test cases

Test the algorithm

...

make tests

./tests/tests

...

We have 3 test cases for the ConnectedComponents() function, which is based on the BFS().

We also have 3 test cases for Brandes() function, which is based on the BFS4ST().

There are a total of 13 assertions in 6 test cases.

Note: It usually takes 15 seconds.

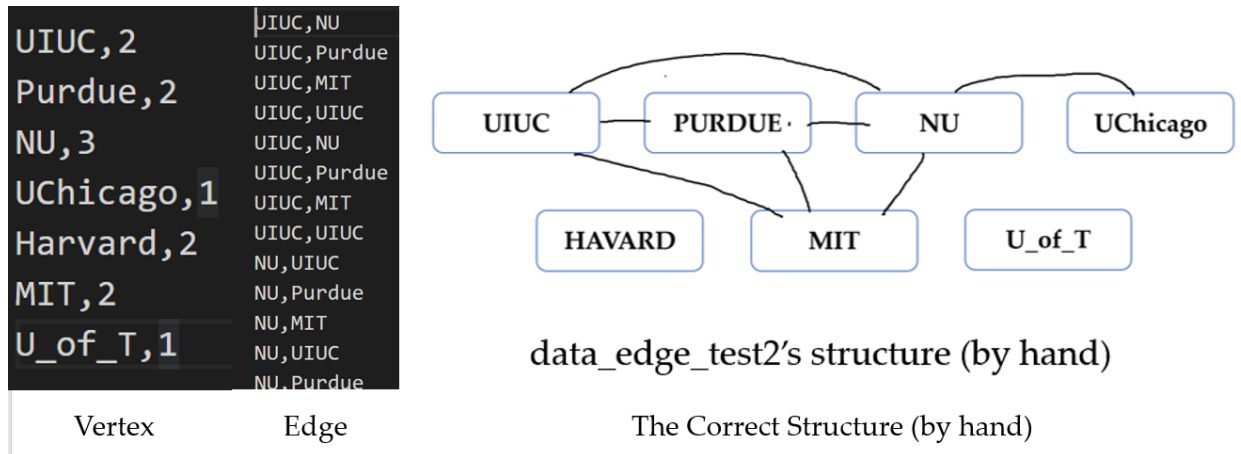
4. The Algorithms

4.1 BFS for connected components:

We use the BFS algorithm to find the connected components. The reason why we need to do this is, we have to get a connected component aka. subgraph first to calculate the betweenness centrality.

We have 3 test cases for this process, take test2 as an example:

The input data are:



[Figure 3. The input vertices, Edges (partial), and the structure by hand]

The output are:

```

12 // print all connected components
13 for (unsigned i = 0; i < connected_component.size(); i++) {
14     for (unsigned j = 0; j < connected_component.at(i).size(); j++) {
15         std::cout << connected_component.at(i).at(j).name << ", ";
16     }
17     std::cout << std::endl;
18 }
19
20 //graph.brandes();
21 //graph.printBrandes();

```

PROBLEMS OUTPUT TERMINAL

```

root@LAPTOP-SOAIK877:~/cs225_final_graph# ./main
UIUC, NU, Purdue, MIT, UChicago,
Harvard,
U_of_T,

```

[Figure 4. The cout result]

To help you get an intuitive understanding, we first put a cout result here.

```

unsigned answer = 3;
unsigned num_of_subgraphs = connected_component_3.size();
REQUIRE(answer == num_of_subgraphs);

std::vector<std::string> answer_1{"UIUC", "NU", "Purdue", "MIT", "UChicago"};
std::vector<std::string> answer_2{"Harvard"};
std::vector<std::string> answer_3{"U_of_T"};

REQUIRE(cc_result_3[0] == answer_1);
REQUIRE(cc_result_3[1] == answer_2);
REQUIRE(cc_result_3[2] == answer_3);

```

[Figure 5. The most import part of a test case]

In the tests.cpp, we write test cases to check if the number of subgraphs is correct, and whether their result is one-to-one respondents. Note the reason why we can check the data in this way is that the order of the connected components is fixed. it's controlled by the order of the

data_edge_test2_file. That is, as you can see in Figure 3, UIUC is the first value in column 1, then NU, Purdue, MIT..., and the last one is U_of_T.

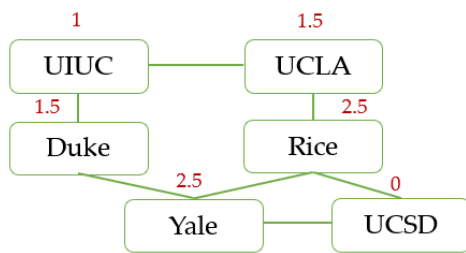
We hope this example can help you to know how the algorithm works, you can also try out 2 more test cases for more information.

Largest connected components in our dataset:

```
rootedf405175b25/workspaces/c9225/release-122/c9225_finala_./main
University_of_Toronto, Duke_University, Michigan_State_University, Sunnybrook_Research_Institute, University_Toronto_Affiliates, University_of_Manitoba, University_of_Warwick, Carnegie_Mellon_University, Columbia_University, Lancaster_University, Stanford_University, Universita_Ca_Foscari_Venezia, University_of_Michigan, University_of_Minnesota_Twin_Cities, Harvard_University, University_of_Miami, Roma_Tre_University, Universidade_de_Lisboa, University_of_California_Berkeley, University_of_Illinois_Urbana-Champaign, Massachusetts_Institute_of_Technology_(MIT), University_of_Bristol, University_of_Sheffield, Northwestern_University, Open_University_-_UK, York_University_-_Canada, University_of_Chicago, Iowa_State_University, Sapienza_University_Rome, Aalto_University, Indiana_University_Bloomington, National_Institutes_of_Health_(NIH)_-_USA, University_of_Cambridge, NIH_National_Heart_Lung_Blood_Institute_(NHLBI), SUNY_Community_College, Dortmund_University_of_Technology, New_York_State_Psychiatry_Institute, Rutgers_State_University_New_Brunswick, Picardie_Universites, Princeton_University, Universite_de_Picardie_Jules_Verne_(UPJV), UDIIC-French_Research_Universities, Johns_Hopkins_University, Universite_Paris_Cite, University_of_Basel, University_of_California_San_Francisco, Vienna_University_of_Economics_Business, Vrije_University_Amsterdam, Yale_University, Ohio_State_University, University_of_North_Carolina_Chapel_Hill, University_of_Nebraska_Medical_Center, North_Carolina_State_University, Pennsylvania_State_University_-_University_Park, Australian_National_University, Imperial_College_London, Universite_Catholique_Louvain, Boston_University, Dana-Farber_Cancer_Institute, University_of_Milan, Texas_ASM_University_College_Station, Temple_University, University_of_Trieste, Worcester_Polytechnic_Institute, Schering-Plough_Research_Institute, Cornell_University, Indian_Institute_of_Management_Ahmedabad, Saith_College, University_of_California_Los_Angeles, University_of_Florida, University_of_Washington_Seattle, University_of_Wisconsin_Madison, Hebrew_University_of_Jerusalem, University_of_Copenhagen, University_of_California_San_Diego, University_of_Auckland, Swiss_Federal_Institutes_of_Technology_Domain, University_of_Arizona, University_of_Erlangen_Nuremberg, University_of_Pittsburgh, University_of_Texas_Austin, Universidade_Nova_de_Lisboa, Institut_National_de_la_Sante_et_de_la_Recherche_Medicale_(Inserm), Nazarbayev_University, KIMEP_University, University_of_Goettingen, Queensland_University_of_Technology_(QUT), Indian_Statistical_Institute_Kolkata, Northern_Arizona_University, University_of_Oslo, Chinese_University_of_Hong_Kong, Western_University_(University_of_Western_Ontario), University_of_Pennsylvania, University_of_Waterloo, NIH_National_Cancer_Institute_(NCI), SUNY_Maritime_College, University_of_Jinan, University_of_Hong_Kong, University_of_London, University_of_St_Andrews, Murdoch_Children's_Research_Institute, University_of_Melbourne, University_College_London, University_of_Iowa, Universite_de_Technologie_de_Loapleigne, University_of_Southern_California, Colorado_State_University, University_of_California_Irvine, Sorbonne_University, PSL_Research_University_Paris, Universite_Paris-Dauphine, Universite_de_Bordeaux, Hopital_Universitaire_Pitie-Salpetriere_-_APHP, Drexel_University, University_of_Munich, University_of_Valencia, University_of_New_South_Wales_Sydney, SAS_Institute_Inc, University_of_Maryland_College_Park, University_of_Connecticut, Universite_de_Orleans, Peking_University, Xiamen_University, University_of_Western_Australia, University_of_Oxford, University_of_Oregon, University_of_California_Los_Angeles_Medical_Center, University_of_South_Carolina, University_of_South_Carolina_Columbia, Pompeu_Fabra_University, Tel_Aviv_University, University_of_North_Carolina_Charlotte, Washington_State_University, Simon_Fraser_University, University_College_Dublin, University_of_Perugia, National_Institute_of_Standards_Technology_(NIST)_-_USA, Georgia_Institute_of_Technology, Kangwon_National_University, University_of_Rochester, Aarhus_University, Utrecht_University, University_of_Bath, Webster_University, Acadia_University, University_of_California_Davis, Norwegian_Institute_of_Public_Health_(NIPH), University_of_Bergen, Shanghai_Jiao_Tong_University, Purdue_University, Purdue_University_West_Lafayette_Campus, University_of_Kentucky, Flinders_University_South_Australia, European_University_Institute, Johannes_Gutenberg_University_of_Mainz, Universidade_de_Sao_Paulo, University_of_Southampton, Telethon_Kids_Institute, McGill_University, Vanderbilt_University, University_of_Freiburg, University_of_Birmingham, Oxford_University_Hospitals_NHS_Foundation_Trust, Louisiana_State_University, New_York_University, Universidade_Federal_de_Pernambuco, Universidade_Federal_de_Pernambuco_(UFPE), Kansas_State_University, University_of_Eimbrebury, Washington_University_(WUSTL), Leiden_University, Leiden_University_Medical_Center_(LUMC), Leiden_University_-_Excl_LUMC, University_of_Amsterdam, University_of_Massachusetts_Amherst, Erasmus_University_Rotterdam, University_of_Virginia, Virginia_Polytechnic_Institute_State_University, University_of_West_Florida, Auburn_University, Georgia_Southern_University, Arizona_State_University-Tempe, University_of_Texas_Dallas, California_Institute_of_Technology, University_of_Alabama_Tuscaloosa, Southern_Illinois_University_Edwardsville, Hungarian_Alfred_Renyi_Institute_of_Mathematics, Bowling_Green_State_University, Institut_Polytechnique_de_Paris, Norwegian_University_of_Science_Technology_(NTNU), University_of_British_Columbia, National_University_of_Singapore, University_of_Edinburgh, University_of_Illinois_Chicago, University_of_Illinois_Chicago_Hospital, University_of_New_Brunswick, Indian_Institute_of_Technology_(IIT)_-_Kanpur, University_of_Maine_Orono, University_of_Padua, University_of_Bologna, Cardiff_University, Institut_Agro, London_Metropolitan_University, Rice_University, University_of_Antwerp, University_of_Cologne,
```

4.2 BFS4ST for Brandes

First, I'd like to introduce the math principle of betweenness centrality.



UCLA – Rice	0/1	0
UCLA – Yale	0/1	0
UCLA – UCSD	0/1	0
UCLA – UIUC	0/1	0
Rice – Yale	0/1	0
Rice – UCSD	0/1	0
Rice – UIUC	0/1	0
Yale – UCSD	0/1	0
Yale – UIUC	1/1	1
UCSD – UIUC	1/2	0.5

An example ->
betweenness centrality for Duke is 1.5

```
UIUC, 1
UCLA, 1
Duke, 1
Rice, 1
Yale, 1
UCSD, 1
```

Input order

```
Betweenness
1
1.5
1.5
2.5
2.5
0
```

print result

```
// The answer :
std::vector<double> answer{1,1.5,1.5,2.5,2.5,0};

// Result
std::vector<double> VofBetweenness;
VofBetweenness = graph.getBetweenness();

REQUIRE(answer == VofBetweenness);
```

Testcase, correct!

[Figure 6. An example of the calculation of betweenness centrality]

If we want to know the betweenness centrality for Duke, we need to find the amount of shortest paths from one node to another node (except Duke) as the denominator. And, among those shortest paths, how many paths go through the Duke as the numerator. Take UCLA - Rice as an example, there is only 1 shortest path. That is, UCLA - Rice, so the denominator is 1. However, this path does not go through Duke so the numerator is 0. We got $0/1 = 0$.

So, that's why we need to use BFS4ST() to find "shortest paths".

4.3 Brandes algorithm

Betweenness Centrality (Brandes Algorithm)

Math Definition of Betweenness Centrality:

1. Denote Node v in Graph $G = \langle V, E \rangle$ (unweighted, undirected, connected), betweenness centrality for Node v is the **proportion of shortest paths that go through v** .
2. Define $\sigma(s, t)$ as the number of shortest path between source nodes s and target nodes t .
3. Define $\sigma(s, t | v)$ as the number of shortest path between source nodes s and target nodes t that **pass through Node v** .
4. Betweenness Centrality of $v = \text{sigma}(\sigma(s, t | v) / \sigma(s, t))$
5. If $v \in s, t$, then $\sigma(s, t | v) = 0$. (When Node v is source or target in a path, we do not consider this path going through v)

Algorithm Implementation:

1. BFS integration: different than normal simple BFS, the BFS4ST() function we used traverse all nodes in all connected components, and return the shortest path when given source node and target node.
2. Create 2D array to store values for $\sigma(s, t)$, the size is `all_nodes_size * all_nodes_size`, initialize the 2D array with value 0.
3. For node pair (i, j) , use BFS to find all shortest paths. Calculate the number of shortest paths from i to j passed through v (when checking each path of the shortest paths, increment $\sigma(v, t)$, or more precisely $\sigma(v, j)$, as appropriate when each node v is reached).

4. Divide each $\sigma(v, j)$ by total number of shortest paths ($\sigma(i, j)$), update the centrality of each node.
5. Time complexity: $O(V^3)$ Space complexity: $O(V^2)$

Difficulties:

1. Normal simple BFS would not traverse all nodes in all connected components. We traverse all connected components first and check whether passed arguments (pair of node i and node j are in the same component).
2. It is difficult at first to think about how to store all shortest paths and how to pick up the path that passes through a particular node.
3. We use triple for loop: We use double loop first to locate a node pair and then find all shortest paths and also shortest paths passing through v . There might still be space to optimize time complexity.

4.4 Fruchterman-Reingold algorithm

In this algorithm, we treat each vertex as a mass point in a 2 dimensional space, and the mass of each point is seen as the centrality of a vertex in one connected component. We move all the vertices in a single connected component several times to find the stable status of the graph. We will traverse all the connected components one by one.

In the function [Distribute](#), [CalculateAttractive](#) and [CalculateRepulsive](#), we distribute vertex randomly and calculate the scalar force between two vertices.. The scalar force between two vertices is calculated only if there is an edge between two vertices. Before we calculate the force, we assign each vertex with a coordinate in 2 dimensional space. Then, we calculate the attractive force using the formula $\text{Force} = \text{distance} * k_1$, where k_1 is a constant parameter. Additionally, the repulsive force is calculated using the formula $\text{Force} = \text{mass}_1 * \text{mass}_2 * k_2 / \text{distance}^2$, where k_2 is another parameter. These two steps are finished in functions [CalculateAttractive](#) and [CalculateRepulsive](#).

In the function [CalculateNetforce](#), we calculate the netforce of a vertex in a connected component. After we get the scalar attractive force and repulsive force between two vertices, we find the direction of each force, which is represented by positive values and negative values. Therefore, the vector force is found between two vertices and we will add this force to the net force of vertex in a connected component. The total net force of a vertex is obtained by adding all the forces between a single vertex and all of its neighbors.

The function [Move](#) is for moving all the vertices in a connected component together. After finding all the netforce of a vertex in a connected component, we will move the vertex using the formula: $\text{distance} = \text{Force} / \text{mass} * \text{time}^2 / 2$. After we move vertices for n times, we will draw the graph and output the graph in ppm image, which is implemented in functions [Draw](#) and [toPPM](#).

The time complexity for this algorithm is $O(n^3)$ and space complexity is $O(n^2)$, where n is the number of nodes.

The output:

```
1 force on x coordinate = 664.072, force on y coordinate = 836.754
2 force on x coordinate = 195.325, force on y coordinate = -150.874
3 force on x coordinate = -12.3, force on y coordinate = -82.3
4 force on x coordinate = -115.8, force on y coordinate = -159.8
5 force on x coordinate = 10.5, force on y coordinate = 70.9
6 force on x coordinate = -83.3, force on y coordinate = -91.6
7 force on x coordinate = -48.3687, force on y coordinate = 144.48
8 force on x coordinate = 133.719, force on y coordinate = -60.8446
9 force on x coordinate = -336.134, force on y coordinate = -212.623
0 force on x coordinate = -3.3, force on y coordinate = -96.3
1 force on x coordinate = 441.489, force on y coordinate = -247.528
2 force on x coordinate = -131.9, force on y coordinate = -202.3
3 force on x coordinate = -92.6163, force on y coordinate = -144.7
4 force on x coordinate = -157.981, force on y coordinate = 314.229
5 force on x coordinate = -662.745, force on y coordinate = 609.827
6 force on x coordinate = 37.7, force on y coordinate = -53.6
7 force on x coordinate = -63.9, force on y coordinate = 39.1
8 force on x coordinate = -96.8, force on y coordinate = 41.3
9 force on x coordinate = -814.481, force on y coordinate = 115.557
0 force on x coordinate = 118.368, force on y coordinate = 169.063
1 force on x coordinate = 251.229, force on y coordinate = 260.628
2 force on x coordinate = 349.089, force on y coordinate = -264.358
3 force on x coordinate = -118.9, force on y coordinate = 17
4 force on x coordinate = -4.1, force on y coordinate = 41.2
```

[Figure 7. The result of Fruchterman-Reingold algorithm]

5. Our Achievement:

This graph represents the vector net force of a single vertex of connected components. As we have mentioned, the result of betweenness centrality can show how popular (importance) the node is. The result of the Fruchterman-Reingold algorithm shows the influence of all neighbors on a vertex.

For the visualization part. We found that as the centrality of a vertex varies a lot in a connected component, distance based on the force also varies from 10000 to -2000, making it impossible to build an image containing stable connected components that could run on our computer.

6. Individual thoughts:

1. Data quality matters
2. Part that we like, the mathematical and the physics part are interesting
3. Part that we did not like, the visualization is impossible to draw, and we can not predict if it can work before we run our code.