Google Summer of Code 2021 Proposal

Implement Brandes' Betweenness Centrality in pgRouting using Boost Graph Library

Proposed by
Saurav Uppoor

Table of content:

1. Contact Details

Name: Saurav Uppoor

Country: India

Email: sauravsuppoor@gmail.com

Phone: +91 8408867834

Location: Mumbai, India, +5:30 GMT

Github: https://github.com/sauravUppoor

Linkedin: https://www.linkedin.com/in/sauravuppoor/

2. Title

Implement Brandes' Betweenness Centrality in pgRouting using Boost Graph Library.

3. Brief Project Description

My project will focus on implementing **Brandes' Betweenness Centrality** algorithm as an addition to the current set of pgRouting algorithms during the GSoC '21 period.

Among various types of centralities in graph theory, betweenness centrality (C_b) is a measure of central positioning of a node based on the shortest paths passing through that particular node. Brande's algorithm is available in the Boost Graph Library (BGL) and can be found here. Centrality can be computed both for weighted and unweighted graphs using this algorithm. The running time of this algorithm is $O(V * E + V * (V + E) * logV)$.

4. State of the Project Before GSoC

Currently, pgRouting does not have an algorithm to find the centralities in a graph.

- Brandes' betweenness centrality is a measure of central positioning of a node based on the shortest paths. Although pgRouting has some shortest path algorithms implemented, there doesn't exist any standard algorithm to compute the centralities, especially the betweenness centrality.

5. Benefits to Community

The algorithm finds its applications in a variety of real scenarios. Below are a few:

- Higher the value of betweenness centrality for a node/edge means higher the amount of traffic (like vehicular, cyclists, public etc) passing through that particular node. This can be used for better resource allocations such as maintenance, infrastructure upgrade.
- Betweenness centrality is one of the most important means of measuring centrality since it helps us derive other centrality measures like closeness, graph and stress centrality from it and help us cluster the node as per these values.
- In network theory and telecommunication, a node with higher betweenness centrality means more control over the graph since more information passes through that node.

6. Deliverables

1. **Implementation of Brandes' Betweenness Centrality algorithm**: I plan to create a function `pgr_brandesBetweennessCentrality()` in pgRouting which will return a vector of values of type float signifying centralities of each node. Based on the number of parameters provided, betweenness centrality can be calculated for directed and undirected graphs.

2. **Documentation and tests for the above function**: I need to prepare the required helper functions and header files along with creating documentations and pgTap tests for the proposed function. Also, I need to prepare a report for each evaluation.

7. Timeline

Community Bonding Period (May 17, 2021 - June 7, 2021)

- Introduce myself to the community, interact and discuss with the mentor(s) the idea to be implemented.
- Study the directory structure of the project and understand the standards for developing and testing.
- Learn about generating documentation and adding examples to them.
- Run the pgTap tests and create some tests for the already implemented pgRouting functions, if possible.
- Create a wiki page for tracking weekly progress.
- Gain an understanding of the integration of PostgreSQL with Boost C++ in pgRouting for data storage and computation.

Official Coding Period

First Coding Phase

Time Period	Estimated Time	Proposed Work
Week 1 (June 7 - June 13)	21	• Design pgr_brandesBetweennessCentrality() function.
Week 2 (June 14 - June 20)	16	• Create a basic skeleton for C,C++,SQL code and for documentation and testing.

Week 3 **(June 21 - June 27)**	21	● Read data using SQL. ● Create C containers for passing SQL data to C++ containers for processing.
Week 4 **(June 28 - July 4)**	21	● Creation of suitable C++ containers for SQL data for Boost processing.
Week 5 **(July 5 - July 11)**	21	● Process the data using Boost. ● Create a report for the first phase.
Total	100	

- **Phase 1 Evaluation**:
  - Submit a report for evaluation.
  - Mentors evaluate me.

Second Coding Phase

Time Period	Estimated Time	Proposed Work
Week 6 **(July 12 - July 18)**	21	● Work on the feedback provided during the first evaluation. ● Transform the results to C containers for passing to SQL.
Week 7 **(July 19 - July 25)**	21	● Write meaningful pgTap tests for the proposed function (no crash test, edge cases, inner query tests etc).
Week 8 **(July 26 - July 1)**	21	● Prepare user documentation. ● Fix any bugs/problems arising in the code base.
Week 9 **(August 2 - August 8)**	21	● Create queries for documentation using the pgRouting sample data.
Week 10 **(August 9 - August 15)**	21	● Integrate the work done to the pgRouting main repository. ● Prepare the final report.
Total	105	

- **Final Evaluation:**
  - Submit the complete project along with the required functions, tests and documentation.
  - Submit final report and evaluation of mentors.

8. Do you understand this is a serious commitment, equivalent to a full-time paid summer internship or summer job?

Yes, I understand that this is a serious commitment and is equivalent to a full time paid summer internship or job. I assure you that I will work with full diligence and comply with the given deadlines. I will try my best to maintain, test and contribute to this pgRouting even after the SoC period. I really look forward to contributing to pgRouting and implementing my proposed idea.

9. Do you have any known time conflicts during the official coding period?

Apart from my academic studies, I don't have any major known time conflicts during the coding period. I will be able to devote at least 3 hours per day except for the second week of the coding period and hence have reduced the time estimate above.

10. Studies

What is your school and degree?

- **School:** Thadomal Shahani Engineering College, University of Mumbai
- **Degree**: Bachelor of Engineering in Computer Engineering

Would your application contribute to your ongoing studies/degree? If so, how?

Yes, contributing to pgRouting falls in line with my current studies in multiple ways. I will be able to get acquainted with the practical use of various software engineering practices and models, which I am currently studying in semester 6. Also, having studied various courses like Graph Theory, Open Source Technology Lab, Algorithms, Database Management etc, I will be able to apply my learnings directly in this project.

11. Programming and GIS

Computing Experience

- **Programming Languages:** C, C++, Java, Python, PHP
- **Database:** PostgreSQL, MySQL
- **Tools:** Git
- **Operating System:** Windows 10, Ubuntu 20.04

- **Relevant Courses:** Discrete Mathematics, Analysis of Algorithms, Database Management, Object Oriented Programming Methodology, Open Source Technology

GIS experience as a user

- I have tried OpenStreetMap & OsmAnd apps for navigation and its offline support.
- I have also tried various routing functionalities provided by pgRouting and postGIS.

GIS programming and other software programming

- Working on *Encryption Library,* C++ implementations of various encryption algorithms, as course project for Cryptography and System Security - Code

- *CG-Pathfinder,* Dijkstra's algorithm visualiser built using C++ as a course project for Computer Graphics - Code

- *UpGrade*, a web application for students and teachers to conduct tests and upload assignments - Code

- *Caruide,* voice-controlled desktop application for career-related information built using python and APIs - Code

- Solved over 1000 algorithmic problems on CodeForces, CodeChef, Atcoder, HackerRank - StopStalk Profile

Briefly mention and link to former open-source contributions

- Refactored code, created tests and documentation for 'TheAlgorithms/C-Plus-Plus' during Hacktoberfest 2020 - PR#1, PR#2

- Contributed to building the user interface for *Product Social* - Code

- Redesigning college committee webpage as a frontend design team member - Code

- Created a repository containing intermediate C++ notes and codes - Code

12. GSoC Participation

Have you participated in GSoC before?

No, I haven't participated in GSoC before.

Have you applied but were not selected? When?

No. This is my first time applying for GSoC.

Have you submitted/will you submit another proposal for this year's GSoC to a different org? Which one?

No, I will be submitting only one proposal for this year's GSoC, for OSGeo.

13. pgRouting Application Requirements

pgRouting application requirements are mentioned here - https://github.com/pgRouting/pgrouting/wiki/GSoC-Ideas:-2021#pgrouting-application-requirements

Links for the tasks:

- Task 1: Intent of application
- Task 2: Experience with Git and Github
    - Task 2: Pull request
- Task 3: Build pgRouting locally
- Task 4: Get familiar with C++
- Task 5: Get familiar with pgRouting

14. Detailed Proposal

Proposed Documentation: Brandes' Betweenness Centrality

Proposed Description

This algorithm calculates the betweenness centrality value for each vertex in the graph by running the shortest path algorithm between all pairs of vertices.

The main characteristics are:

- The algorithm works for directed and undirected graphs.
- Values are returned when the graph contains at least a single vertex.
- The algorithm returns float values denoting *betweenness centrality* for each vertice.
- Returned values are ordered in ascending order of vertex indices.
- Running time:
    - Unweighted graph: $O(V * E)$
    - Weighted graph: $O(V * E + V * (V + E)logV)$

Proposed Signature

```
pgr_brandesBetweennessCentrality(Edges SQL, [, directed])
RETURNS SET OF (vertex_id, centrality)
OR EMPTY SET
```

Parameters:

Parameter	Type	Default	Description
Edges SQL	TEXT		An SQL query.
directed	BOOLEAN	true	• When true graph is considered *Directed.* • When false graph is considered *Undirected.*

Inner Query:

Edges SQL: an SQL query, which returns a set of rows with the following columns:

Column	Type	Default	Description
id	ANY-INTEGER		Identifier of the edge.
source	ANY-INTEGER		Identifier of first endpoint vertex of the edge.
target	ANY-INTEGER		Identifier of second endpoint vertex of the edge.
cost	ANY-NUMERICAL		Weight of the edge (source, target).

reverse_cost	ANY-NUMERICAL	-1	Weight of the edge (target, source).

Where:
ANY-INTEGER : SMALLINT, INTEGER, BIGINT
ANY-NUMERICAL : SMALLINT, INTEGER, BIGINT, REAL, FLOAT

Result Columns:

Returns SET of (vertex_id, centrality)

Column	Type	Description
vertex_id	BIGINT	Identifier of the vertex.
centrality	FLOAT	Centrality value of the vertex. • Minimum value of centrality is 0.

Proposed Query

```
SELECT * FROM pgr_brandesBetweennessCentrality (
     'SELECT id, source, target, cost, reverse_cost FROM my_edge
ORDER BY id,
     TRUE
);
```

Sample Output

```
 vertex_id | centrality
-----------+------------
         1 |          0
         2 |          8
         3 |          0
         4 |          0
         5 |         28
         6 |    13.3333
         7 |          0
         8 |          8
         9 |     3.6667
        10 |    14.1667
```

```
        11 |     5.3333
        12 |          0
        13 |          0
        14 |          0
        15 |          7
        16 |          0
        17 |          0
(17 rows)
```

Theory

Centrality metrics measures the importance of each node with varying definitions of importance. Out of all the centrality metrics - closeness, graph, betweenness and stress, betweenness centrality is the most popular since it can be modified to find other centrality metrics. Betweenness centrality is a measure of the central positioning of a node in the graph. It is a measure of the extent to which a node plays a bridging role in the graph.

The betweenness centrality of a node v is defined in terms of the shortest paths that pass through v. Formally:

1. Assume a directed/undirected, weighted/unweighted graph $G =< V, E >$.
2. Define $\sigma(s, t)$ as the count of shortest paths from s to t.
3. Define $\sigma(s, t|v)$ as the count of shortest paths from node s to t passing through node v.
4. $C_b(v)$ be the betweenness centrality of node v.

One way to compute $C_b(v)$ efficiently is to use an algorithm proposed by Brandes, which gives the following result

$$C_b(v) = \sum_{s,t \in V} \delta(s, t|v)$$

where $\delta(s, t|v) = \frac{\sigma(s, t|v)}{\sigma(s, t)}$ is referred to as *pair-wise dependency* in Brandes' algorithm[1] and $s, t \neq v$.

We can normalise the above betweenness centrality values by scaling it down by $\frac{2}{n^2 - 3n + 2}$ where $n = |V|$. This normalised value is also referred to as *relative betweenness centrality* denoted by $C'_b(v)$. This value isn't a part of our query results

since Boost C++ Library has a different function to compute it and manually computing it might reduce the precision.

For finding the shortest paths, it is optimal to use BFS and Dijkstra/Bellman-Ford algorithm for unweighted and weighted graphs respectively. Since the shortest path computation needs to be done for all pairs of vertices in the graph, Brandes' algorithm is efficient for Sparse graphs. A graph is said to be sparse if the following holds true[3]

$$0 \leq \frac{E}{(\frac{V*(V-1)}{2})} \leq 0.5$$

where symbols have their usual meaning. However, it is not a necessary condition for Brandes' algorithm. For undirected graphs, each shortest path is chosen twice, hence the value of $C_b(v)$ is scaled down by 2. The running time of this algorithm is $O(V*E)$ for an unweighted graph and $O(V*E+V*(V+E)*logV)$ for a weighted graph.

Approach

1. For every node $v \in V$, set $C_b(v) = 0$.
2. For each node $s \in V$:
    a. Use BFS or Dijkstra, to find all shortest paths from s to all other nodes. Store all the paths for each target t.
    b. For each t, for each vertex v that occur in the stored path, count the number of times it occurs in total to give $\sigma(s,t|v)$ and divide with the total number of paths from s to t (i.e., $\sigma(s,t)$). Add the result to $C_b(v)$.
3. $C_b(v)$ gives the final result.

This is a naive approach to the algorithm. There are a couple of optimisations with storage possible.[1]

Code

- Boost Graph Library - Brandes' Betweenness Centrality: https://www.boost.org/doc/libs/1_75_0/libs/graph/doc/betweenness_centrality.html
- Examples used in the sample testing below have been implemented here: https://github.com/sauravUppoor/SampleCode
- Animations for the examples can be found here: https://github.com/sauravUppoor/SampleCode#visualisation

Sample Testing

Example 1

The graph for testing is from the pgRouting sample data: Network for queries marked as directed and only cost column is used. Fig. 1 shows the graph from sample data.

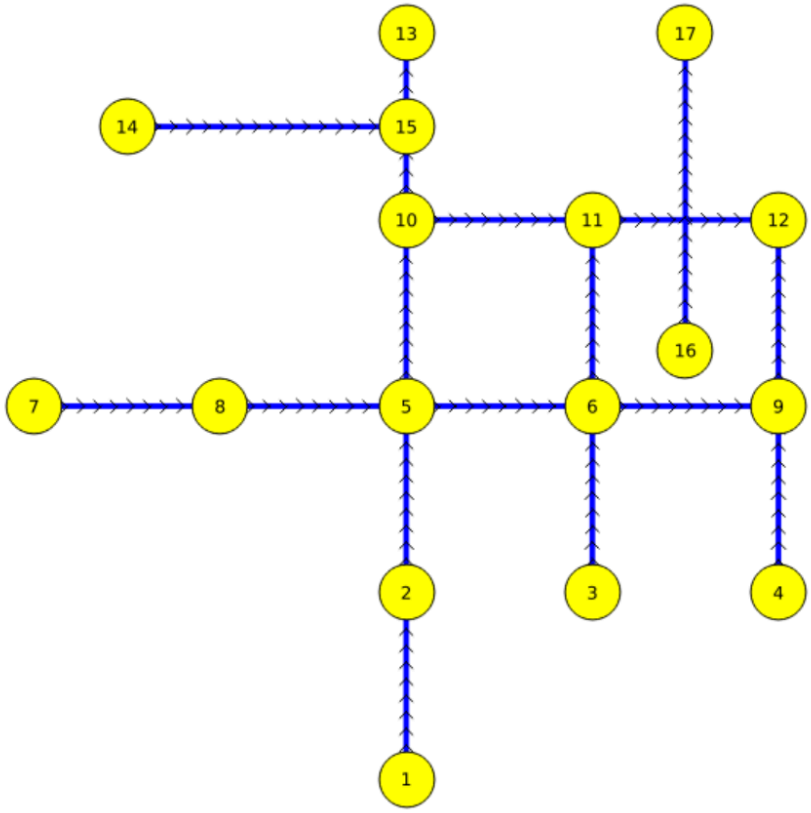


Fig. 1 pgRouting sample data directed graph

CREATE

```
CREATE TABLE my_edge (
     id BIGINT,
     source BIGINT,
     target BIGINT,
     cost BIGINT,
     reverse_cost BIGINT
);
```

INSERT

```
INSERT INTO my_edge (id, source, target, cost, reverse_cost)
VALUES
(1, 1, 2, 1, -1),
(2, 2, 5, 1, -1),
(3, 3, 6, 1, -1),
(4, 4, 9, 1, -1),
(5, 7, 8, 1, -1),
(6, 8, 5, 1, -1),
(7, 5, 6, 1, -1),
(8, 6, 9, 1, -1),
(9, 5, 10, 1, -1),
(10, 6, 11, 1, -1),
(11, 9, 12, 1, -1),
(12, 10, 11, 1, -1),
(13, 11, 12, 1, -1),
(14, 16, 17, 1, -1),
(15, 10, 15, 1, -1),
(16, 14, 15, 1, -1),
(17, 15, 13, 1, -1);
```

SELECT

```
SELECT * FROM my_edge;
 id | source | target | cost | reverse_cost
----+--------+--------+------+--------------
  1 |      1 |      2 |    1 |           -1
  2 |      2 |      5 |    1 |           -1
  3 |      3 |      6 |    1 |           -1
  4 |      4 |      9 |    1 |           -1
  5 |      7 |      8 |    1 |           -1
  6 |      8 |      5 |    1 |           -1
  7 |      5 |      6 |    1 |           -1
  8 |      6 |      9 |    1 |           -1
  9 |      5 |     10 |    1 |           -1
 10 |      6 |     11 |    1 |           -1
```

```
 11 |      9 |     12 |    1 |          -1
 12 |     10 |     11 |    1 |          -1
 13 |     11 |     12 |    1 |          -1
 14 |     16 |     17 |    1 |          -1
 15 |     10 |     15 |    1 |          -1
 16 |     14 |     15 |    1 |          -1
 17 |     15 |     13 |    1 |          -1
(17 rows)
```

Query

```
SELECT * FROM pgr_brandesBetweennessCentrality (
    'SELECT id, source, target, cost, reverse_cost FROM my_edge
ORDER BY id,
    TRUE
);
```

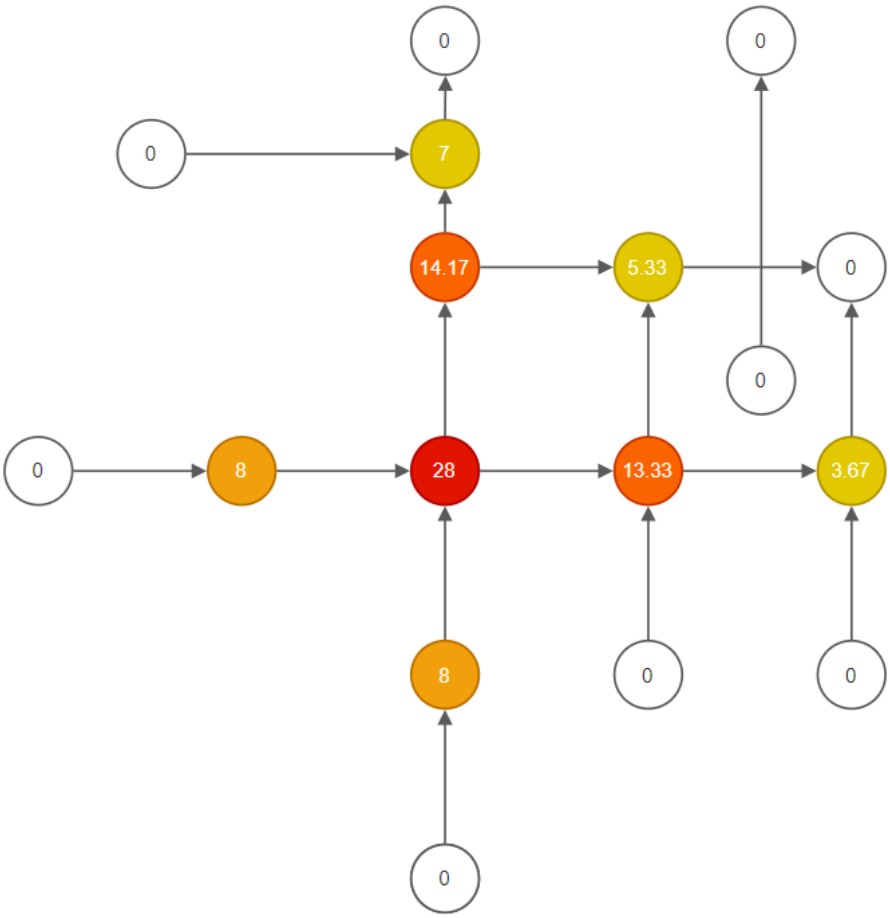
Output

```
 vertex_id | centrality
-----------+------------
         1 |          0
         2 |          8
         3 |          0
         4 |          0
         5 |         28
         6 |    13.3333
         7 |          0
         8 |          8
         9 |     3.6667
        10 |    14.1667
        11 |     5.3333
        12 |          0
        13 |          0
        14 |          0
        15 |          7
        16 |          0
```

```
        17 |           0
(17 rows)
```

Visualization

Animated version of the diagrams can be found here.



Colour	○	●	●	●	●
BC Range	0-3	3.1-7	7.1-11	11.1-15	> 15

Fig. 2 BC values for a sample graph

Fig. 2 shows the sample graph with the betweenness centralities values. Node 11 (marked with blue square) has the highest C_b value since, observably, it is more centrally placed. Let's look at how the $C_b(11)$ was computed.

s	t	Shortest paths from s to t	$\delta(s,t\|11)$
7	12	**7 → 8 → 5 → 10 → 11 → 12** **7 → 8 → 5 → 6 → 11 → 12** 7 → 8 → 5 → 6 → 9 → 12	⅔ = 0.6667
8	12	**8 → 5 → 10 → 11 → 12** **8 → 5 → 6 → 11 → 12** 8 → 5 → 6 → 9 → 12	⅔ = 0.6667
5	12	**5 → 10 → 11 → 12** **5 → 6 → 11 → 12** 5 → 6 → 9 → 12	⅔ = 0.6667
2	12	**2 → 5 → 10 → 11 → 12** **2 → 5 → 6 → 11 → 12** 2 → 5 → 6 → 9 → 12	⅔ = 0.6667
1	12	**1 → 2 → 5 → 10 → 11 → 12** **1 → 2 → 5 → 6 → 11 → 12** 1 → 2 → 5 → 6 → 9 → 12	⅔ = 0.6667
6	12	**6 → 11 → 12** 6 → 9 → 12	½ = 0.5
3	12	**3 → 6 → 11 → 12** 3 → 6 → 9 → 12	½ = 0.5
10	12	**10 → 11 → 12**	1
$\sum$			5.333

For the rest of the combinations of (s,t), the value of $\delta(s,t|11)$ is 0 since node 11 will not be a part of their shortest paths. Hence, it is not shown in the above table. Likewise, we can compute betweenness centrality using brandes' algorithm for the rest of the nodes.

Example 2

Note: Since Boost Graph Library didn't have any example for this algorithm, the following example has been created by me and implemented here.

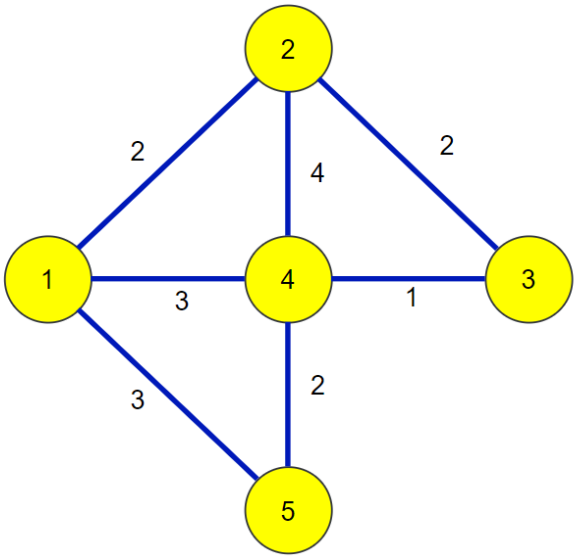


Fig. 3.1 Undirected, weighted graph

CREATE

```
CREATE TABLE my_edge (
     id BIGINT,
     source BIGINT,
     target BIGINT,
     cost BIGINT,
     reverse_cost BIGINT
);
```

INSERT

```
INSERT INTO my_edge (id, source, target, cost, reverse_cost)
VALUES
(1, 1, 2, 2, 2),
(2, 1, 4, 3, 3),
(3, 1, 5, 3, 3),
(4, 2, 3, 2, 2),
(5, 2, 4, 4, 4),
(6, 3, 4, 1, 1),
(7, 4, 5, 2, 2);
```

SELECT

```
SELECT * FROM my_edge;
 id | source | target | cost | reverse_cost
----+--------+--------+------+--------------
  1 |      1 |      2 |    2 |            2
  2 |      1 |      4 |    3 |            3
  3 |      1 |      5 |    3 |            3
  4 |      2 |      3 |    2 |            2
  5 |      2 |      4 |    4 |            4
  6 |      3 |      4 |    1 |            1
  7 |      4 |      5 |    2 |            2
(7 rows)
```

Query

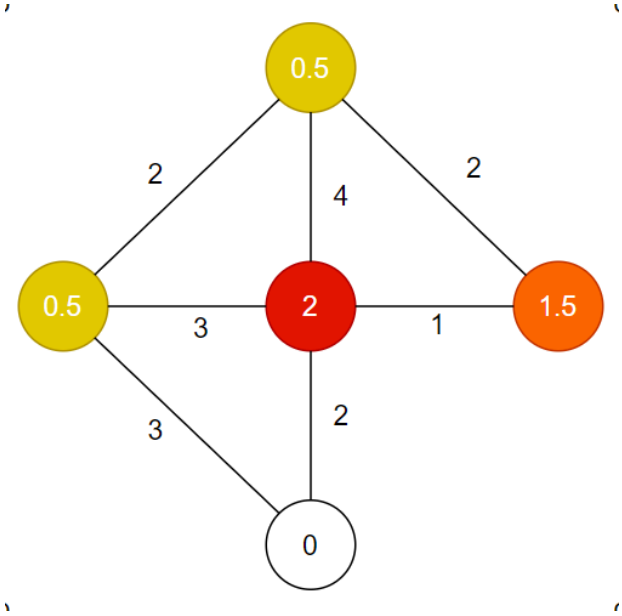
```
SELECT * FROM pgr_brandesBetweennessCentrality (
     'SELECT id, source, target, cost, reverse_cost FROM my_edge
ORDER BY id,
     FALSE
);
```

Output

```
 vertex_id | centrality
-----------+------------
         1 |        0.5
         2 |        0.5
         3 |        1.5
         4 |          2
         5 |          0
(5 rows)
```

Explanation

Animated version of the below visualisation can be found here.



Colour					
BC Range	0-0.4	0.5-0.9	1-1.4	1.5-1.9	>1.9

Fig. 3.2 BC values of each node

Fig. 3.2 shows betweenness centrality values of each node of the graph in fig. 3.1. The computation for node 3 (central node in fig 3.1) is shown below:

s	t	Shortest paths from s to t	$\delta(s,t\|3)$
1	4	**1 → 2 → 3 → 4** 1 → 0 → 4	½ = 0.5
2	0	**2 → 3 → 0** 2 → 1 → 0	½ = 0.5
2	4	**2 → 3 → 4**	1
$\sum$			2

For rest of the combinations of (s, t) node 3 isn't included in their shortest paths, hence not displayed in the above table. Hence, $C_b(3)$ = 2. Likewise, we can find the centrality values for all the nodes.

Proposed Directory Structure and Files

Directory	Files
pgrouting/doc/betweennessCentrality/	CMakeLists.txt, pgr_betweennessCentrality.rst
pgrouting/docqueries/betweennessCentrality/	CMakeLists.txt, doc-betweennessCentrality.result, doc-betweennessCentrality.test.sql, test.conf
pgrouting/include/betweennessCentrality/	pgr_betweennessCentrality.hpp
pgrouting/include/drivers/betweennessCentrality/	pgr_betweennessCentrality.h
pgrouting/pgtap/betweennessCentrality/	innerQuery.sql, no_crash_test.sql, types_check.sql, edge_cases.sql
pgrouting/sql/betweennessCentrality	CMakeLists.txt, _betweennessCentrality.sql, betweennessCentrality.sql
pgrouting/src/betweennessCentrality	CMakeLists.txt, betweennessCentrality.c, betweennessCentrality_driver.cpp

Applications

The algorithm finds its applications in a variety of real scenarios. Below are a few:

- **Urban space analysis and resource allocation**

For urban space analysis by GIS, streets segments and intersections are considered. PostGIS also uses a similar approach for mapping spatially embedded networks like streets, nodes, settlements etc. *Space Syntax* is a methodology for urban analysis. It has shown growing evidence about the correlation between the centrality

and the urban phenomenons like vehicular and pedestrian flow, retail commerce vitality and even crime rates[3].

Higher the value of betweenness centrality for a node/edge means higher the amount of traffic (like vehicular, cyclists, public etc) passing through that particular node. This can be used for better resource allocations such as maintenance, infrastructure upgrade. This can also be used in other situations like classifying areas as low vehicle density zones and can improve the time prediction for navigation systems.

- **Compute Central Point Dominance**:

Another metric for finding the centrality of a graph, in a general sense, is the central point dominance. Using relative betweenness centrality, we can compute the central point dominance. It is an overall "betweenness" value for a graph. Let's denote central point dominance as $C_b'(G)$. Formally we can define central point dominance as

$$C_b'(G) = \frac{\sum_{v \in V} C_b(v*) - C_b'(v)}{n-1}$$

where $v*$ denotes a node with highest betweenness centrality value[4].

Its values lie between 0 and 1. Fig. 4.1 shows a complete graph (K_4) and fig. 4.2 shows a Wheel graph. Consider a graph G_1 as a complete graph and G_2 as a wheel graph then we have $C_b'(G_1) = 0$ and $C_b'(G_2) = 1$.

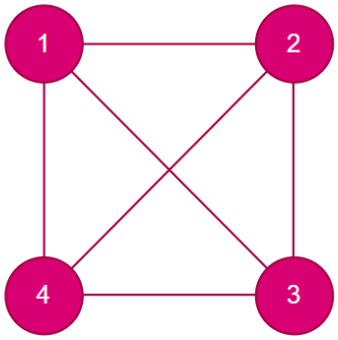


Fig. 4.1 K_4 graph

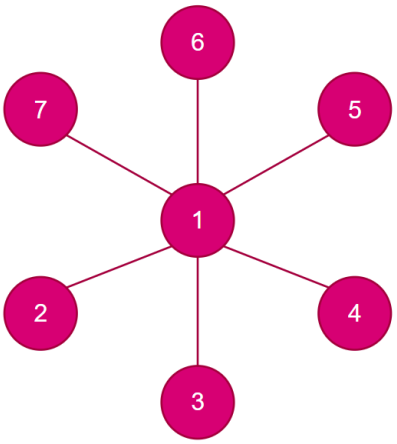


Fig. 4.2 Wheel graph

- **Identify critical nodes in a network**:

Sometimes, a node/edge could go down due to construction/roadblock/accidents etc. If a node goes down, we can compute the effects (traffic flow) due to diversions on

the whole graph. This can assist in finding the critical nodes and avoid bottlenecks and help in congestion control. It is not encouraged to have nodes in networks which have comparatively very high centrality values since the chances of the disruption of the whole network increases. Hence, emphasis should be more on distributing the centrality values throughout the network.

- **Compute Closeness, Stress and Graph Centralities using Betweenness Centrality algorithm**:

Each of these centralities denotes significance of an edge or a node in a graph. Betweenness centrality helps us compute these other centralities using minimal changes. Here $d_t(v)$ denotes the shortest path from node v to t.

a. *Closeness centrality* is a measure of how close a node is to the rest of the nodes. It is defined as

$$C_c(v) = [\sum_{t \in V} d_t(v)]^{-1}$$

b. *Graph centrality* is the distance of a node to its most remote counterpart. It is given by

$$C_g(v) = [\max_{t \in V}(d_t(v))]^{-1}$$

c. *Stress centrality* is the absolute count of shortest paths passing from a node.

$$C_s(v) = \sum_{s,t \in V} \sigma(s, t|v)$$

- **Social network analysis**:

Centrality is a fundamental concept in network analysis. Bavelas was the first to realize that central individuals in a social network very often play a prominent role in the group, or in other words a good location in the network structure corresponds to power in terms of independence, influence and control on the others. He applied the idea of centrality to human communication and their social circles; he was interested in the characterization of the communication in small groups of people and assumed a relation between structural centrality and influence and/or power in group processes[3].

In social networks, network theory and telecommunication, a node with higher betweenness centrality means more control over the graph since more information passes through that node. Such a node can then decide whether to pass the information or not, thus has a higher stature.

15. Future Scope

There are many graph metrics and sparse matrix algorithms that are still not implemented in pgRouting. The following algorithms could be implemented in future as an extension to this project/pgRouting:

- **Relative Betweenness Centrality**:

As mentioned in the theory section, relative betweenness centrality is a scaled down version of absolute betweenness centrality. It is defined as follows:

$$C'_b(v) = \frac{2 * C_b(v)}{(V-1) * (V-2)}$$

Calculating the value of $C'_b(v)$ manually using the Boost's **`brandes_betweenness_centrality()`** might result is deterioration of precision of floating values, hence I think instead of having parameters in the current proposed function, it is better to have a separate function in pgRouting collection and use the inbuilt **`relative_betweenness_centrality()`** function provided by the Boost Library for calculative $C'_b(v)$.

- **Betweenness Centrality Clustering:**

This algorithm implements clustering in a graph based on the values of edge betweenness centrality. It is an iterative algorithm that removes an edge with the highest value of edge betweenness centrality. As a result, the graph gets divided into various connected components called clusters. Boost Graph Library has an inbuilt function, **`betweenness_centrality_clustering()`**, for this algorithm. Complete Boost documentation can be viewed here.

- **Bandwidth and ith bandwidth**:

BGL has an inbuilt function to calculate Bandwidth and ith bandwidth of a given graph. For a graph, lets assign each node in the graph with a unique $index$ with the value in the range $[0, |V|)$. Bandwidth of a graph is formally defined as:

$$B(G) = \max_{(u,v) \in E} |index[u] - index[v]|$$

Similarly, ith bandwidth of a graph is bandwidth for a node i. It is given by:

$$B_i(G) = \max_{(i,j) \in E} |index[i] - index[j]|$$

Complete documentation of Boost Graph Library for the above function can be found here.

These algorithms and properties mentioned above are some of the most widely used graph metrics algorithms from the Boost Graph Library and I think having them in the pgRouting library will greatly benefit the users.

16. References

[1] Dr Simone Teufel and Prof Ann Copestake, *Brandes' Betweenness Centrality algorithm, Implementation and Optimisations,* University of Cambridge.

[2] Yuntao Jia, Victor Lu, Jared Hoberock, Michael Garland, John C. Hart, *Edge v. Node Parallelism for Graph Centrality Metrics*, University of Illinois Urbana-Champaign.

[3] Aisan Kazerani and Stephan Winter, *Can Betweenness Centrality Explain Traffic Flow?*, Department of Geomatics, The University of Melbourne.

[4] Linton C. Freeman, *A Set of Measures of Centrality Based on Betweenness*, University of California, Irvine.

[5] Baeldung, Sparse vs Dense Graphs.

[6] Brandes' Betweenness Centrality, Boost C++ official documentation.

[7] Wikipedia, Betweenness Centrality.

[8] Boris Schaling: Boost.Graph for Beginners, Boost Graph Library lecture.

[9] Boost Library Concepts, BGL function return types, Boost C++ documentation.

[10] Exploring Boost Graph Library, IBM Boost C++ graph documentation.

[11] Sample Data, pgRouting Sample Graph Data for pgTap testing.

[12] Algorithms Index, pgRouting Manual (3.2-dev).

17. Resume