Enabling Adoption of High-Performance Centrality Tools

Alexander Palmer, UNC Charlotte
Professor Erik Saule, College of Computing and Informatics

UNIVERSITY OF NORTH CAROLINA CHARLOTTE

Introduction

Importance of Centrality

- Centrality measures the importance or influence of nodes in a network.
- Centrality can be used to identify key players in various fields like **medical research**, **social networks**, **intelligence**, and **anti-money laundering** operations.
- High-performance centrality tools enable faster and more accurate analysis of large networks.

Benefits of High-Performance Centrality Algorithms

- Calculating Centrality for large graphs can take a significant amount of time, like 5 days for a 4.8 million edges graph.
- **BADIOS** is a research prototype framework consisting of high-performance C++ algorithms that significantly reduces the computation time to 16 hours for the same graph.

Low Adoption of High-Performance Centrality Tools

- Tools like **BADIOS** are often research prototypes with limited real-world deployment.

```
Algorithm 1: SeqBC(G = (V, E))
for all v ∈ V do
  bcent[v] ← 0
for each s ∈ V do
  stack ← ∅, queue ← ∅
  queue.push(s), dst[s] ← 0, σ[s] ← 1
  for all v ∈ V \ {s} do
    dst[v] ← ∞, pred[v] ← ∅, σ[v] ← 0
  ▷Forward Phase
  while queue is not empty do
    v ← queue.pop(), stack.push(v)
    for all w ∈ adj(v) do
      if dst[w] < 0 then
        dst[w] ← dst[v] + 1
        queue.push(w)
      if dst[w] = dst[v] + 1 then
        σ[w] ← σ[w] + σ[v]
        pred[w].push(v)
  ▷Backward Phase
  for all v ∈ V do
    δ[v] ← 0
  while stack is not empty do
    w ← stack.pop()
    for v ∈ pred[w] do
      δ[v] ← δ[v] + σ[v]/σ[w] (1 + δ[w])
    if w ≠ s then
      bcent[w] ← bcent[w] + δ[w]
return bcent
```

Fig 1. Sequential Algorithm to compute betweenness centrality

Background

BADIOS

Compresses the graph, splits into multiple disconnected components, and obtains another graph with several graph manipulations.

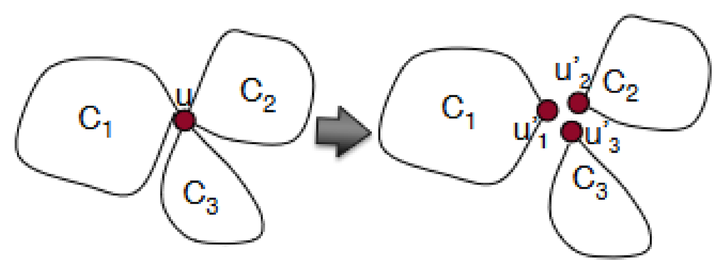

Fig 2. A graph manipulated and cloned into three separate graphs

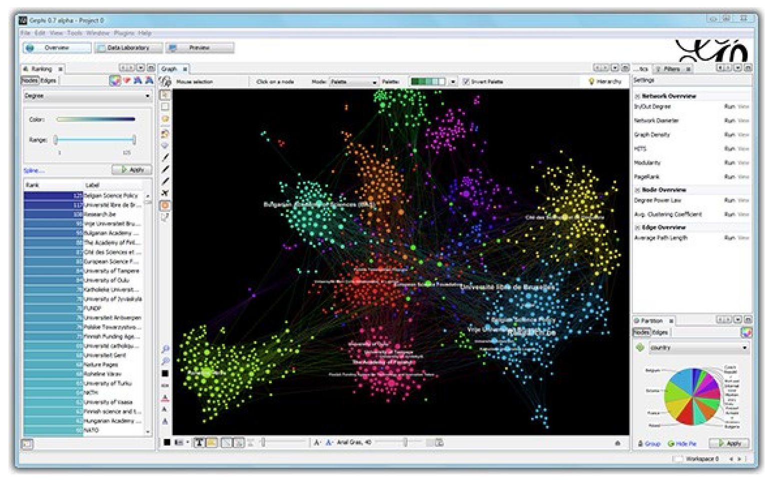

Fig 3. Gephi User Interface

Methods

- Create a new java class that implements Gephi Statistics interface from org.gephi.statocs.spi package.
- In the execute() method of the new class, obtain the GraphModel and Graph object from the input parameters.
- Extract the nodes and edges from the graph
- Package the plugin to be added to Gephi

```java
@Override
public void execute(GraphModel graphModel) {
    Graph graph = graphModel.getGraphVisible();


    Node[] nodes = graph.getNodes().toArray();
    Edge[] edges = graph.getEdges().toArray();

    // Call C++ code
    double[] result = NativeLibrary.calculate(nodes, edges);

    // Create or retrieve result column
    Table nodeTable = graphModel.getNodeTable();
    Column col = nodeTable.getColumn(CALCULATION_RESULT);
    if (col == null) {
        col = nodeTable.addColumn(CALCULATION_RESULT, "Calculation Result"
    }

    // Set result
    for (int i = 0; i < nodes.length; i++) {
        nodes[i].setAttribute(col, result[i]);
    }
}
```

Fig 5. Java method Using JNIA to call C code

Conclusion/Future Work

Integrating BADIOS's high-speed C++ algorithms into Gephi through a plugin will reduce computation time from 5 days to 16 hours for a 4.6 million edges graph.

The plugin will benefit scientists and researchers by enabling them to use tools like BADIOS along with popular tools like Gephi to compute large complex scientific problems.

Future Work

Implement Java methods that call the appropriate C methods from BADIOS and display the results in a Gephi format.

Objectives

Design Goals

- Implement a simple C++ method into Java using JNI and JNA
- Develop a plugin for Gephi that allows users to easily use BADIOS in Gephi

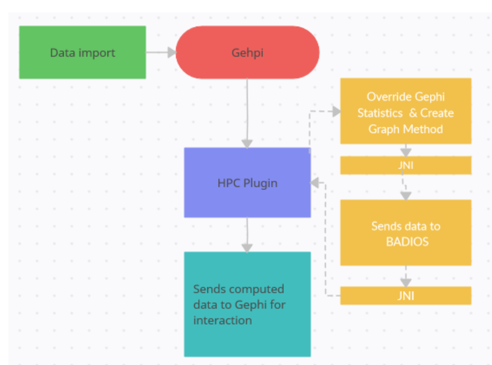

Fig 4. Flowchart displaying the custom plugin process

Challenges

- **Performance:** Calling C code created issues passing objects and data structures
- **Configuration:** The platform-specific libraries and dependencies made it more challenging
- **Debugging:** Debugging both Java and C++ simultaneously is challenging when locating errors

References

unknown. (2022, 04). Gephi. Retrieved from https://gephi.org/developers/

Lu, S. (2023, 04 10). working from Gephi's source. Retrieved from https://seinecle.github.io/gephi-tutorials/generated-pdf/working-from-the-source-en.pdf

Saryuce, A., Kaya, K., Saule, E., & Catalyurek, u. (XXXX, January). Graph Manipulations for Fast Centrality Computation. *ACM TKDD V, N, Article A*, p. 22.

Saryuce, A. E., Saule, E., Kaya, K., & Catalyurek, U. (2014, July). Regularizing Graph Centrality Computations. *JPDC*, p. 41.