CSCI-4974/6971: Assignment 3 (100 pts)

Social Graph Analysis

Due Date: Monday 17 Oct. 2016, 16:00

For this assignment, we're going to combine a few concepts we learned during the social analysis and partitioning lectures to do the following:

1. Implement and run a clustering/community detection algorithm
2. Perform graph coarsening using output from our clustering algorithm
3. Calculate betweenness centrality on the coarsened graph

For background material and reference, use the lecture slides and our hands-on code along with:

- `https://en.wikipedia.org/wiki/Label_Propagation_Algorithm`
- `https://en.wikipedia.org/wiki/Betweenness_centrality`
- `https://en.wikipedia.org/wiki/Floyd-Warshall_algorithm`

Submit the code file to `slotag@rpi.edu` by the due day and time listed above. Put your responses to the short answer questions in comments in the code file.

1 Label Propagation Graph Coarsening (50 pts)

To begin, first download the Assignment 3 code outline and datasets that will be used for this part:

- `hw03-social.cpp`
- `test.graph`
- `LiveJournal.graph`

You can use the smaller test graph for testing your code, but your code should also be able to run on the larger LiveJournal graph from SNAP as well (note: I did some preprocessing on it to make things a bit cleaner and also added randomly generated edge weights). You'll be filling in some empty functions in the cpp file.

We wish to perform a betweenness centrality analysis on our graph. However, if you recall, the most straightforward approach uses the Floyd-Warshall algorithm, which requires $O(n^3)$ time to compute. This is prohibitively expensive for a graph the scale of LiveJournal. So first, we're going to use label propagation to to coarsen the graph by exploiting the graph's inherent community structure. Do the following in `label_prop()`:

1. Implement any version of parallel label propagation to create community assignments for each vertex with `comm_assignments`
2. Determine and run a sufficient number of iterations to end up with $O(1000)$ communities

We're next going to create a coarse graph where each community in the original graph as output by LP is a vertex in our coarse graph, and edges between communities become coarsened into a single edge that is the sum of the weights of the prior edges. E.g., community A has several vertices with edges going to vertices in community B - these edges will be represented as a single edge between two single vertices in the coarse graph with a weight equal to the sum of the weights of the prior edges. To create our coarse graph, we're going to call `create_csr()` with the following that we calculated in `coarsen_graph()`:

1. `num_comms`: the total count of communities, or the number of vertices in our new coarse graph
2. `num_intercomm_edges`: the number of connections in between the communities in the original graph, or the number of edges in our new coarse graph
3. `coarse_srcs`, `coarse_dsts`, `coarse_wgts`: Arrays of length num_intercomm_edges. These will hold the edge data, where a given index in all three arrays would represent a source vertex, a destination vertex, and the weight of the edge.

Note: `create_csr()` and our graph struct requires that vertex identifiers be less than the number of vertices in the graph. So you'll need to relabel the coarse vertices from [0, num_verts) to [0, num_comms). Also, we'll need for every (`coarse_srcs`[i] = A, `coarse_dsts`[i] = B) pair, a corresponding pair (`coarse_srcs`[j] = A, `coarse_dsts`[j] = B), where i != j and i,j = [0, `num_intercomm_edges`) and `coarse_wgts`[i] = `coarse_wgts`[j].

This will be evaluated on:

- Parallel label propagation implementation - 10 pts
- Properly determining `num_comms` and `num_intercomm_edges` - 10 pts
- Properly filling in `coarse_srcs`, `coarse_dsts`, and `coarse_wgts` - 30 pts

2 Calculating Betweenness Centrality (50 pts)

The second part of the assignment is to calculate the betweenness centrality for all vertices in the coarsened graph using the Floyd-Warshall algorithm. This part of the assignment can all be done in serial. Remember that betweenness centrality is the number of shortest paths that pass through a given vertex for all (i,j) vertex pairs in a graph. Although the basic F-W algorithm will only calculate shortest distances between the (i,j) vertex pairs, the psuedocode given on the Wikipedia page for the F-W algorithm will give details on how to use F-W to also determine the associated paths. So, in the `calc_betweenness()` function:

1. Use Floyd-Warshall to calculate shortest paths distances (*dist*) and *next* values for all vertex pairs in the coarse graph
2. Use the Floyd-Warshall outputs to find the shortest paths between all vertex pairs, and use that to determine each vertex's betweenness centrality

Finally, we're going to perform some basic analysis using our coarsened graph and output betweenness centrality values. We'd like to find the following:

1. The top 5 vertices in terms of betweenness centrality. How (if at all) are these vertices connected to one another?
2. The top 5 most important edges (*bridges*), where importance here is defined as the sum of betweenness centralities of the vertices connected by an edge

This will be evaluated on:

- Implement the Floyd-Warshall algorithm - 20 pts
- Use the output of Floyd-Warshall to calculate betweenness centrality - 20 pts
- Output the top 5 most central vertices and edges - 10 pts