ECE 454 Assignment 2 Part B

cpoenaru, am3clark, tszwiega

Get Adjacency List

Knowing that in order to limit the triangles we record, we only list the triangle in increasing order or vertex ID such as (1, 5, 7) and do not allow for a triangle to be listed as (5, 1, 7) etc. Knowing this, when we read in the file listing the neighbors for each vertex, we only include neighbors with a higher value than the vertex we are building the adjacency list for. This optimization means that during our counting algorithm the number of loop iterations will be decreased. The adjacency list for each vertex is stored as a HashSet of integers to allow O(1) inserts and lookup, and all of these HashSets are stored in an ArrayList.

Triangle Counting

To count triangles, a loop iterates through each and looks for triangles starting with each vertex i, trying to find pairs of neighbors j and k such that i < j < k. We know that because of the optimization when building the adjacency lists that all neighbors in the adjacency list for i are greater than i, we do not need to check the condition for i < j and i < k. Instead, for each combination of two neighbors of i, the following process is followed:

```
if j < k:
    if adjacency list of j contains k:
        result += new Triangle(i, j, k)
```

Multi-threading

If more than one core is being used (n cores), n>1, a threadpool with n threads is created each with id 0 to n. The triangle counting algorithm we use is read only, and because we have the restriction that triangles are only recorded in the order i < j < k, we can divide the work between the threads for checking the triangles starting with each vertex. Each thread will run on each vertex starting at the thread's id and incrementing through by n to avoid collisions between threads.

The process of reading the file into the adjacency list has not been multi threaded.