Research Project Summary:

What are SBM's?

SBM's (Stochastic Block Models) are a technique used for graph sharding. Graph sharding is a well studied problem with many known problems such as the Minimum Bisection problem (finding a partition of a graph into two shards of equal size while minimizing the number of crossing edges between shards) and the Balanced Graph Partitioning problem (extends the Minimum Bisection problem to k shards). The latter problem in particular is what we are focusing on using block inference via SBM's.

The main idea behind SBM's is to infer blocks which contain locally clustered vertices in the same block, i.e. the vertices within each block have more internal edges towards other vertices in the same block rather than outgoing edges towards vertices external to the block. This is accomplished by an iterative process discussed in Duong et al's paper, which is based on the convergence of variational free energy discussed in Hofman and Wiggin's paper. On a high level, the process goes as follows:

1.) Do a random assignment of the nodes to blocks
2.) Do a discounted vote (discounted since we only look at the blocks the neighbor of each vertex of the target vertex is connected to) and decide whether and where to move the node
3.) Keep doing 2.) until the free energy converges (locality improvement less than 1%)
4.) Do a greedy assignment of the blocks to shards (cut off shards if necessary)

Also critically, the formula for calculating each node's "attraction" to another block is as follows:

$$z_i \leftarrow \operatorname{argmax} \; Jw_k - J'(n_k - \delta_{z_i,k}) - h_k$$

Each attraction is weighted in the sense of J and J', which are determined via the Beta distribution, and h_k which is a logarithm determined by the Dirichlet distribution. In the context of an undirected graph, w_k is the number of neighbor's the node has that is located in the block. (n_k - delta_zi,k) is the number of nodes minus the number of nodes in the block. For undirected graphs, w_k and delta_zi,k are equivalent, but this may change in the directed case. Intuitively, the nodes you are connected to are 'voting' for you since your inclusion would benefit their clustering in the block, while the ones you aren't connected to are 'voting against' you since your inclusion would detriment their clustering in the block. Votes may not all have equal weight however, which is why we have J and J' determined by the Beta distribution. Furthermore, the block might be far from or reaching its capacity. Hence, depending on how 'full' the block already is, h_k will be positive/negative to increase/decrease a node's attraction to the block.

This is done for each node per iteration, and at the end of the iteration the node's are reassigned to their new blocks or stay where they are. Something to note is that since the assignment process happens at the end of each iteration, a prior assignment in the same iteration should not affect another assignment.

Fixes/Modifications:

- The main bottleneck in the existing SBM implementation was in finding a node's new assignment. Specifically, a function (countEdgesBetweenNodeAndBlock) that calculates the number of edges from a block to the target node was called repeatedly per node which went through all nodes in a particular block. This caused the entire program to have a hidden factor of the block size rather than just running in linear time. Instead of calling this function repeatedly, everything was instead done in one pass via lines 371-395 clus.cpp by storing the block assignments beforehand. This allows the program to run in linear time, while space complexity is not affected significantly (still linear), and has a huge impact for larger graphs as shown in the runs below
- The existing implementation in the node assignments modified the assignments in the same iteration, updating the node's assignment immediately rather than all at once at the end of the iteration. This was corrected at lines 599-605 clus.cpp
- The existing implementation included all nodes in the block, not just the nodes that aren't connected to the target node, when calculating (n_k - delta_zi,k) in the node assignment. This was corrected at line 389 clus.cpp
- Several additional functions were included (in both clus.cpp and greedyassignment.cpp) to make analysis easier, such as outputting the block sizes in sorted order and the largest block size for each iteration. After the greedy assignment is completed, the edges in blocks, edges in blocks but different shards, edges in clusters, edges not in blocks but within clusters are now outputted as well.
- A program to 'normalize' graphs input to their adjacency matrix forms with node ID's starting from 0 was created (translate.cpp in dataset folder)

Note: All real time measurements were obtained via the linux time command

File	Nodes	Edges	Old runtime	New runtime
facebook_combined.txt	4039	88234	0m9.247s	0m6.501s
Slashdot.txt	82168	948464	9m49.506s	0m24.066s
as-skitter.txt	1696415	11095298	92m21.396s	2m20.419s

Optimizations:

C++ via the GNU compiler has a series of built-in optimization flags. These flags, O, O2 and O3, increasingly improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program. The below dataset shows what happens when each of these flags are turned on, comparing the old versus new runtime. Even enabling the most basic optimization level, -O, had significant advantages to improving the runtime. However, any optimizations after that had little effect. The optimizations were run on the largest available data set, as-skitter.txt, that has 1696415 nodes and 11095298 edges.

Optimization level	Nodes	Edges	Runtime
No optimization	1696415	11095298	2m16.638s
-O	1696415	11095298	0m46.681s
-O2	1696415	11095298	0m48.931s
-O3	1696415	11095298	0m45.931s

Github Link:

- https://github.com/Sparks0219/SBM.git