CS 170 Project Deliverable 2 Final Report

Lily Bhattacharjee, Thien Phan, Tina Zhao

December 2, 2018

1 Introduction

We coded five different approaches, calculated the score of each, and returned the output from the approach that had the highest score. Runtime uses N (number of students).

2 Evenly Distributed Greedy Approach

This approach does not error out on any of the graphs. It selects the students in the most rowdy groups and places them in different buses. It then adds a vertex and its first-degree friends to a bus. We keep moving onto each bus until all students are in a bus. It works best when there is a large bus capacity compared to the number of buses. Runtime: $O(N^3)$

3 Full Bus Greedy Approach

The same as the previous approach, except it populates the entire bus first before moving onto the next one. It works well if the bus size to bus number ratio is small since it's less likely we get empty buses. Runtime: $O(N^3)$

4 Straightforward Clique Approach

It looks through all rowdy groups and the graph to generate all possible cliques. It sorts these cliques by size in descending order and puts the largest cliques in each bus. When a clique is placed in a bus, corresponding vertices from the other cliques are removed and we re-sort. After the initial iteration, we place inside each bus a clique closest to that size of the space left. We keep doing this until all students are inside. When filling up buses after, it may add students who belong to the same rowdy groups. It performs badly on large, sparse graphs because it could run out of cliques. Runtime: $O(2^N)$

5 Broken Clique Approach

The same as the previous, but does not sort the cliques. It adds cliques to a bus until it is full and moves onto the next one. It works well when the bus number and capacity are similar, but it might leave some buses empty. Runtime: $O(2^N)$

6 Kernighan-Lin Algorithm Approach

It partitions the vertices evenly among the buses (number of partitions = number of buses). We perform Kernighan-Lin on all possible pairs of partitions to reduce lost friendships. It works best for dense graphs with many edges. Breaking an edge costs more than having a rowdy group. Runtime: $O(2^N * N^2 \log N)$

7 Conclusion

Average Runtime: $O(2^N * N)$ (n swaps/partition)

Overall Runtime: $O(2^N * N^2 \log N)$

The straightforward clique approach has the most accurate results because it looked at friend and rowdy groups before partitioning into buses.

To improve, we could consider rowdy groups in our Kernighan-Lin approach by adding weighted edges between each rowdy member. If all members of a rowdy group are inside a partition, the weights on the edges from a vertex to its rowdy teammate would be equal to the negative of the number of members in its corresponding rowdy groups. The rest would be weight 1.

8 Libraries and Citations

Libraries: NetworkX, numpy, ast, and community from NetworkX.algorithms (Kernighan-Lin function)

Yan, Bowen, and Steve Gregory . *Detecting Communities in Networks by Merging Cliques.* pp. 1–5. https://arxiv.org/pdf/1202.0480.pdf