Maximum Edge Retention with Curtailed Simulated Annealing Adjustments (MERCSAA)

Emily Zhang, Vincent Tantra, Alex Chang

For the solution to this NP-hard problem, our group decided to find a solution by following processes similar to linear programming and simulated annealing. MERCSAA proceeds to find an initial solution and then improves the solution by making small adjustments. We adjust the solution until improvements are deemed unnecessary or when our solution is at a local maximum in terms of optimality.

First, we create an initial solution by adding connected groups to the buses. We start with a random assignment of one student to each bus, fulfilling the constraint that each bus must contain at least one student. Then, we perform the following algorithm, similar to dynamic programming: for each bus, we save students to a dictionary where their value is the amount of connections they have to that bus. We then assign the most connected student to that bus, and update the values for the other students with that new assignment. We perform this until all students have been assigned.

However, this algorithm returns semi-satisfactory solutions as we still ignore rowdy groups. We try to account for this by using a process like simulated annealing. We make small adjustments to an initial solution with a score, and save the adjustments made if they result in better scores. This allows us to alter our solution in the direction of a local optimum, taking steps toward solutions where no small adjustment will result in a better score. Our score function finds the percent of friendships saved by our solution; after finding a new solution, we recalculate the score and compare them.

One of the issues we encountered with this approach is that it is computationally intensive, as it technically checks all possible adjustments before choosing the optimal one and proceeding in that direction. This can result in an unfeasible amount of calculations for even small graphs. Therefore, we make the following adjustments to our algorithm to curtail our processes.

Firstly, instead of computing all possibilities before selecting the largest optimal adjustment, we proceed in the algorithm once we encounter the first adjustment that results in a more optimal solution. However, the worst case persists: what if the one step that improves our solution occurs at the end of all our tests? We decided to further compromise on the optimality of our solution by curtailing the algorithm after the first several adjustments made. Finally, we realized that for inputs with many rowdy groups of small sizes, our algorithm would act for long periods of time making micro-adjustments to the solution. Since the payoff of these small improvements was small when accounting for the time required to run the program, our final compromise was to move on and save a solution after it has been altered a certain amount of times where the payoff of those alterations was less than a certain amount.

These compromises, combined with our algorithm, resulted in a program that can be run in a satisfactory amount of time while still finding a solution that approaches local optimums.