Group Members

Group Name: Staff_Solution
Karthik Krishnan: 3034843072
Vinayaka Srinivas: 3034803890
Tarang Srivastava: 3034634981

Generating Inputs

We started with a desired output that we wanted to create an input for. That is, how many breakout rooms and how many people in each breakout room. We generated a script that used a Gaussian distributed points in a certain happiness and stress range. We had two Gaussian distributions for happiness and stress based on if the edge was in our desired output. For edges in our output, we generated happiness using a higher mean Distribution with higher values and stress in a range with lower values. We did the opposite for stress, with lower happiness and higher stress. This would allow our input to match our desired output. We then picked a stress budget that was as close as possible with our desired output.

$$\text{Undesired} = \mathcal{N}(3, 0.65) \qquad \text{Desired} = \mathcal{N}(0.25, 0.125)$$

If we had more time, we would likely try to take a more random approach. Instead of picking a safe difference in the happiness to stress ratio for our unused edges in our output, we would try to bring it closer to our desired edges. This would test algorithms more and make it more difficult to determine a solution. We would also try to pick tricky rooms with edge cases such as 0 stress to trip up algorithms that are entirely happiness-based. This would be a more rigorous input that could have been created if we had more time.

Approach 1: Greedy Local Search

We initially implemented a local search algorithm that greedily chose vertices to add to a breakout room based on highest happiness while monitoring the stress budget.

$$\text{happiness} =$$

The only moves to change neighbors that were valid were additions of one vertex to the room. Upon testing this algorithm on our data, it performed really well on approximately one-third of the data while performing poorly on the remaining two-thirds. In order to incorporate more flexibility with the moves being made to access different neighbors, instead of freeing one student and adding them to another room, we would free two vertices and then re-add them optimally. We also incorporated swaps to get closer to optimality when a vertex addition was too drastic of a move to make. Although this partially improved the results on some of our data, it ended up performing worse on other inputs. From this, we realized that a degree of freedom of two students was not sufficient to consider the broad possibility of solutions and would get us stuck in a particular configuration that is not optimal.