Project Reflection

Team `AnnealBeforeTheKing`
Gilbert Feng, Jason Xiong, Jonathan Pan, Kyle Lui

CS 170: Efficient Algorithms and Intractable Problems
Fall 2020

§1 Input Generation

We used 3 different approaches to generate our inputs. Each aimed at hindering a certain type of algorithm:

- Small: This input was created to stump teams who thought minimizing the number of rooms would yield optimal output, as the 4-room optimum was 156.536 while the true optimum of 192.007 occurred at 5 rooms. 0-stress and 0-happiness edges were also added to cause divide-by-zero errors. This input was manually generated.
- Medium: This input was designed to counter approaches that greedily looked at edges based on $\frac{\text{happiness}}{\text{stress}}$ ratio. We inserted edges that had better $\frac{\text{happiness}}{\text{stress}}$ but would cause certain rooms to become too stressful. This input was generated by a script we wrote.
- Large: This input was intended to foil approaches that greedily looked at edges based on only happiness or only stress. We inserted high-happiness edges ("rocks") and low-stress edges ("pebbles"). Algorithms that greedily added rocks or pebbles initially would get cornered as these edges later displaced certain rooms in the optimum configuration. This input was also generated by our script.

Looking back, many teams were able to achieve our solutions. Perhaps we thought too narrowly about which types of solvers we were trying to defeat. Had we been given more time, we would have tuned the parameters of our input generator to make our intended outputs harder to find.

§2 Algorithm

We utilized a 2-stage 4-pronged strategy, looping through every possible number of rooms k (our approaches required fixed k).

Algorithm 2.1 — For $k = 1$ to n:

Stage 1: Heuristic

Randomly select and assign k students to the k rooms; this initial randomness ensures any combination of students could possibly be together. Then, the 4 prongs were run for $I(k) = i_1(k) + i_2(k) + i_3(k) + i_4(k)$ iterations (each prong run for i_1, i_2, i_3, i_4 iterations respectively).

- Approach 1: Greedily add each remaining student into the room that maximizes $\frac{\text{happiness}}{\text{stress}}$ ratio.
  - We felt this was the most "intuitive" greedy strategy.
- Approach 2: Greedily add each remaining student into the room that minimizes stress.
  - This aims at finding a valid configuration for k on which we could anneal towards an optimum, rather than going for maximum happiness.
- Approach 3: For each room, while possible continuously assign to that room the remaining unassigned student that would add the least stress.
  - We found this approach better at diversifying "groups" of students, leading to a wider range of configurations.
- Approach 4: Randomly assign all remaining students.
  - We hoped this approach could cover certain configurations the other approaches neglected.

Stage 1.5

For invalid configurations, we applied a stress reduction procedure that attempted to get all rooms under the stress threshold (a necessary invariant for Stage 2).

Stage 2: Annealing

Independently for each prong in Stage 1, modified simulated annealing was used to improve results towards local maxima.

- "Neighboring" configurations were valid assignments resulting from moving any one student OR swapping any pair or triplet of students.
- Rather than randomly picking one neighbor at each step, we considered multiple possible neighbors.

Over $I(k)$ iterations (i_1 of approach 1, i_2 of approach 2, i_3 of approach 3, i_4 of approach 4) at k rooms, we saved the best configuration after Stage 2.

Our final output submission for each input was the best configuration over all k.

> **Remark** (Runtime Optimizations)**.** Approaches 1 and 4 typically took more iterations to find a valid configuration to anneal on, so we tuned the ratios $i_1 : i_2 : i_3 : i_4$ accordingly. Furthermore, we set $I(k)$ using an "adaptive optimization" strategy: if the best configuration found at k rooms was the best overall so far, our strategy re-ran at k rooms for many more iterations. Otherwise, we monotonically decreased $I(k)$ due to the general negative correlation we observed between total happiness and number of rooms, as well as runtime considerations (with more rooms our approaches took longer, affording less iterations).

§3 Other Approaches

Most approaches we tried were integrated into our final strategy to widen our coverage. One approach we tried but eventually discarded (due to consistently subpar performance in trials) was a "Kruskal-like" algorithm greedily adding edges (pairs of students) into rooms based on happiness or $\frac{\text{happiness}}{\text{stress}}$.

§4 Computational Resources

- We ran small inputs on a local machine, which finished in ~ 30 minutes.
- We ran medium inputs on local machines; running took ~ 64 hours in total over 4 machines.
- We ran large inputs on AWS EC2 instances; each of our 4 team members used an AWS account, and running took ~ 1900 free hours in total over 4 accounts.
- We did not use non-standard libraries.

§5 Potential Improvements

We should have debugged much more; many days were wasted on bugs such as floating-point roundoff, division-by-zero, infinite loops, typoes, etc. We also could have run more trials to cut out redundant approaches and commit more time to the best $1 - 2$ prongs of our strategy.

With more time, we would have explored guided random approaches (e.g. genetic algorithm).

§6 Closing Remarks

Our team really enjoyed this project, and our competitive spirit drove us to constantly refine our strategies. We thought this project was a great avenue to apply and implement ideas from the course (such as greedy algorithms and $\mathcal{NP}$-completeness).