CS 170: Efficient Algorithms and Intractable Problems

Project Reflection

Team: `discordggTYsy64VcWT`
Nishant Bhakar, Xavier Plourde, Rohan Bohra
Fall 2022

1 Algorithm

Our main approach to solve the test cases was simulated annealing. At each step, one node is randomly chosen and assigned to a random team that differs from its current one. If the change improved the score, it was always taken. Changes that didn't improve the cost were still taken with a probability that was inversely related to the delta cost and a decaying temperature heuristic. Adjacent states in our annealing algorithm were those that could be reached by moving a node from one team to another, swapping the team of two nodes, or "dissolving" a team and distributing its members to the remaining teams (we realized the first two methods yielded the best results, and we could just bruteforce our way to the optimal team count by bounding via the best solution available on the leaderboard and noting the minimum possible cost as a function of team size is most likely convex). We frame this in a larger genetic algorithm heuristic that uses simulated annealing as mutations. The collective population has a shared temperature.

One of the biggest challenges was when our algorithm got stuck at a local minima that we knew to be costlier than the best known minima on the leaderboard. To deal with this, the algorithm was initialized with 1000 random starting assignments that were run simultaneously. In addition, when a score stagnated at a value that was too high for too long, the temperature was increased and the nodes were shuffled around.

As the contest continued, we noticed that it was possible to collect information on better performing solutions from the cost reported on the leaderboard. For example, for some of the test cases on which we weren't first, we were beaten by solution(s) that were better by an exact integer amount. We realized that this meant that those solutions had the same number of teams and nodes per team as our suboptimal solution, but that we simply weren't using the cheapest edges possible. We implemented a version of annealing which consisted only of swapping pairs of nodes.

For the small test case we generated a complete graph where the edge weight between node u and node v was the product of their indices (ie edge (u, v) had weight $(u + 1)(v + 1) \bmod 1000$). The aim of this test case was to ensure that competing approaches were able to soundly distinguish team assignments where it is impossible to satisfy every dispute.

For the medium test case we created a complete graph of the first thirty-ish nodes and randomly connected them each to an equal amount of the remaining nodes. The weight of the edge between u and v was chosen randomly from the range $[\max(\min(u, v), 1), \max(u, v)]$, which ensured that the number of successful solutions is smaller (larger labeled nodes more likely have larger edge weights and should be kept on different teams).

For the large test case we generated a crystal-like structure for which it is possible to get a solution where each dispute is included. Our belief was that such a solution would be difficult for annealing approaches to approximate because there is one optimal answer and many local optima. At the start, our annealing algorithm struggled to solve the example bipartite graph given in the homework problem for the project. However, these graphs are solvable using an n-coloring approach.

2 Other Approaches

There were many smaller optimizations we attempted on the annealing algorithm that didn't work out. We tried choosing the change made at each step of the algorithm from a distribution based on a heuristic, (e.g. if one team is too large and another too small, at each step of the algorithm "slightly" bias towards removing from the larger team and/or adding to the smaller team), but this didn't work for us as well as choosing from the even distribution did. We also tried using different methods for initializing the teams, like n-coloring, but using a large number of random initializations seemed to work the best.

We tried a randomized genetic algorithm, but it did not work as well as annealing. We also tried branch and bound, but couldn't get an efficient and bugfree implementation. Given more time, we would revisit these algorithms and explore other optimizations to our main annealing algorithm, like tuning hyperparameters (temperature and the number of starting states).

3 Computational Resources

We mainly ran the annealing on our personal laptops and desktops, which was enough to compute most of the test cases because we started pretty early.