Reflection

Describe the algorithm you used to generate your outputs. Why do you think it is a good approach?

We used a simulated annealing (SA) approach to generate most of our outputs. We used the `argmin` crate in Rust to do the calculations for us with an initial temperature of 150.0 decreasing at a linear rate of $\frac{t_{init}}{i}$. Our initial state was our current best (generated by randomized hillclimbing). We ran each medium input for 10,000 iterations and large for 100,000 iterations. This improved our medium and large inputs significantly. We thought this was a good approach because there was a good library available and it has been proven to work decently well on other problems before.

What other approaches did you try? How did they perform?

Our first approach was to reduce the problem to an LP and call an external ILP solver with a random seed to construct a valid tower solution to the input cities. The variables are all boolean: one for every (x, y) point a tower could possibly be at, corresponding to whether or not a tower gets placed there. A constraint for every city (i, j) is imposed such that the tower values within the service radius of that city sum to >= 1, corresponding to every city being covered by at least one tower. We imposed penalty variables corresponding to the number of overlapping penalized towers for each point. Then, we would minimize this penalty. While this did work somewhat well, it was incredibly slow (only feasible for small inputs).

To solve this problem, we simply removed the penalty minimization and told the LP to minimize the number of towers placed instead. This would generate a random-ish solution with minimal tower coverage. We would then hillclimb from this point, moving towers one-by-one to reduce penalty, until an optimum is reached - when repeated many times, this worked better and faster than the LP approach (we called this approach RLP).

Since hillclimbing puts our solution at a local optimum, we then turned to simulated annealing (detailed in the first question) to further improve our scores.

We noted that the scores attained by this algorithm were quite competitive, especially for noisier inputs. However, for some inputs (e.g. small/141), a solution with 5 towers is worse than one with 6 towers (this can be checked by eye). Additionally, for some of the very simple inputs (on large), the LP/SA struggled to generate a minimum tower solution, and the algorithm failed to find a good solution. For these, because they were very simple inputs, we were able to come close to the top leaderboard score by constructing the output by hand.

We also ran some naive greedy and improved greedy algorithms, considering the least amount of towers covered and the least penalty invoked, but these approaches did not work as well as what was listed above.

What computational resources did you use? (e.g. AWS, instructional machines, etc.)

Most of our inputs were ran on our personal computers. However, we did create a free tier AWS account with a t2.micro EC2 instance to run the simulated annealing overnight. AWS free tier provides 750 hours per month, and we used ~120, keeping us well within the free tier range. We did not really use the instructional machines.