Project Reflection

Fifteem: Ernest Lu, Anita Liu and Dylan Huynh

CS170 Spring 2022

1 Algorithm

Our algorithm consists of 2 steps:

Step 1. Greedily choose a set of towers and save this solution.

Step 2. Let our current optimal solution consist of x towers. Randomly choose a subset of size $[x/2, x-1]$ towers of this optimal solution and cover all cities covered by this subset. Then, run a greedy algorithm to choose a set of remaining towers. Update the optimal solution if this new solution is better. Repeat this step indefinitely.

In total, Step 2 added about 40 lines on top of a greedy implementation. Our greedy algorithm was modified a little bit throughout the project, but generally followed this structure:

Repeat until all cities are covered:

- Consider all possible tower locations, get a nonempty set of towers that satisfy some condition
- Randomly choose a tower from this set and cover all uncovered cities that this tower services.

Conditions used:

- Towers that cover the most cities.
- Towers that cover the most cities and intersects with least amount of towers.
- Of the towers that can cover the perimeter (can cover cities with max x or max y), towers that cover the most cities.

We found that more complex greedy parameters (distances, weighting the possible tower choices) actually did not perform as well as these "dumber" conditions. This might've been because more parameters constrain the possible set of outcomes.

The algorithm was developed from the following observations:

- We can make greedy be guaranteed to return an optimal solution when the optimal solution needs just one tower to be placed. In general, the greedy algorithm can be very strong on sparse and small inputs.
- Fixing a set of towers can create a small and sparse set of remaining uncovered cities.
- Random choice in greedy can create a lot of variation in solutions
- A lot of the inputs that were not sparse had rectangular, line-like, or simple-ish structures, so a greedy from the perimeter would get good results a lot of the time.
- A lot of optimal solutions had a subset of towers that can be achieved from greedy solutions. This meant that repeatedly choosing a random subset had the potential to continuously improve solutions.
- C++ is fast

2 Other Approaches

We tried the following other solutions to be recursed:

- Weighting the chosen locations, making all locations possible to be chosen at every iteration. This was based on a combination of:
  - number of uncovered cities covered (higher more likely),
  - number of towers intersected with (lower more likely)
  - sum of distances from other towers (higher more likely)
  - sum of distances from other cities (lower more likely)
- "Wiggling" (Changing a tower location by a small amount) subsets of placed towers and running greedy again.
- Greedy including all combinations of given parameters (distance + towers + cities). Greedy with ordered processing orders (reverse, etc).
- Naive solution / solving cases by hand - we did this on simple tricky cases.

All in all, we found keeping our greedy parameters simple to be more effective.

3 Computational Resources

We used the C++ standard library templates and ran a batch file overnight and during the day from 4/28/22 - 5/2/22. repeatedly running our algorithm on the inputs and outputs. Programs were all run on Ernest's personal laptop because we couldn't figure out how to use AWS.