Gannon Barnett
12/20/20

Summary

I implemented a greedy algorithm that would assign beams to least-covered users first. There are three main data structures to support my algorithm: *scenario_t*, *vector<SatBeamEntry>*, and *vector<UserVisibilityEntry>*. I wrote my solution in C++14 using g++10.

Scenario_t is similar to the scenario dict in evaluator.py - stores ids and positions of all users, satellites, and interferers.

vector<SatBeamEntry> contains a list of all color beams of each satellites (size = colors / sat * # sats). This list allows for easy checking of the self-interference and max-beams constraints.

vector<UserVisibilityEntry> contains a list of all users and their respective visible satellites, where a satellite is visible if it is in visible range of the user and respects the non-starlink interference constraint.

Once these structures are populated, I iterate through the list of user visibility in order of number visible satellites (fewest visible -> most visible), and greedily assign beams while respecting self-interference and max-beams constraints.

Using this method I was able to achieve fairly good coverage in a small amount of time. I think improving coverage further would take significant changes to the base algorithm and data structures.

Results

It's hard to quantitatively interpret coverage results without writing further projects to determine absolute best-case coverage. That being said from rough analysis of test cases and my solution's coverage statistics, I think my solution falls close to best-case coverage (80-90%), but further verification is required.

I was initially disappointed with my performance on the final test case - 30% coverage - but then noted that the best coverage while taking only the max beams constraint into account is 47%. Given the density of users and interferer satellites, the true absolute best case is certainly lower, and thus the low coverage of 30% is not that disappointing.

I was happy with the time performance of my solution - around 11s on final test case. Further work could be done to parallelize the data structure population and bring down this number, but considering that the specs specified a standard machine, I didn't think diving into performance further was necessary.