Project 3: Satellite constellation topology design

Future Internet

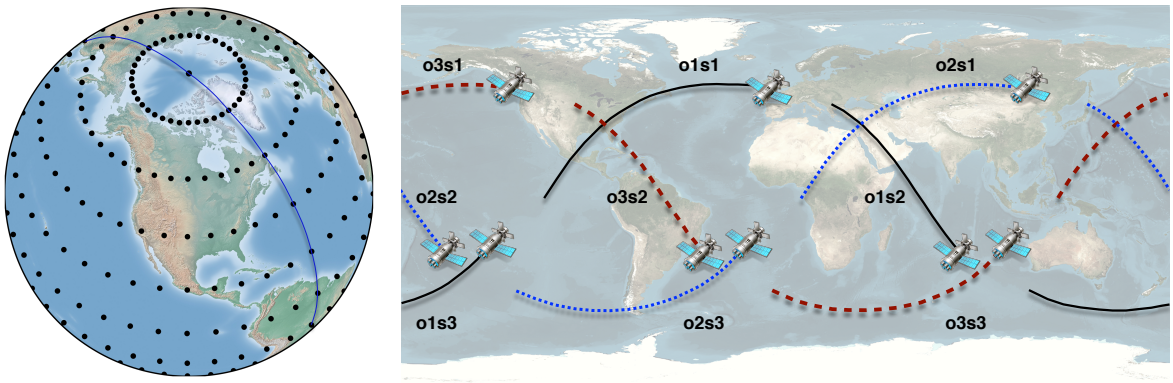
ETH Zürich - Spring Semester 2021

General Future Internet project stipulations:

- This is 1 of 4 projects. Together they account for 50% of your final course grade.
- Always cite and reference appropriately. Do not use other students' code outside of your group.

This particular project stipulations:

- Submission deadline: **10 May 2021 at 15:00**.
- Threshold #1 deadline: **05 May 2021 at 15:00**. If threshold #1 is not reached by this deadline, the total number of points will be reduced by 20%.
- You can receive 12.5 points for this project: achieving threshold #1 (4pt), achieving threshold #2 (8pt), optimize your solution - up to 12.5points.
- You are free to use any programming language and/or tool for this assignment.
- After the deadline, you will have an interview. Each group member must be able to individually explain and demo all parts.
  Interview date: **11 May 2021 at 15:00**.
  Not being able to defend the project solution means zero points from this project for both group members
- You **must not use git push force** and similar commands to revert existing commits or change their metadata.



(a) (b)

Figure 1: *LEO constellations: (a) A LEO satellite constellation consisting of* 20 *polar orbits. Each orbit itself has* 20 *satellites; (b) Satellites with lower inclinations avoid polar regions. This constellation has* 3 *orbits, each with* 3 *satellites.* $o1s2$ *refers to satellite* 2 *of orbit* 1*. Image created using NASA's GMAT [4].*

1 Introduction

In this project, the goal is to construct a low-earth orbit (LEO) satellite network optimally, given all satellite positions, by setting up inter-satellite links (ISLs). In graph theoretic terms, you are given the set of vertices and you need to define the set of edges. The network should ideally minimize the distances and number of hops along the shortest paths between city pairs. In nutshell, it exposes you to the network topology design problem in the context of recently proposed LEO satellite networks.

2 Background

A new space race is imminent, with several large industry players [7, 5, 3] working towards satellite-based Internet connectivity. While satellite networks are not themselves new, these recent proposals are aimed at orders of magnitude higher bandwidth and much lower latency, with constellations planned to comprise thousands of low flying satellites. These are not merely far future plans – SpaceX has already launched few hundreds [1], and substantial planned capacity has already been sold [6]. SpaceX's [7] Starlink constellation is set to comprise more than 40,000 low-earth orbit (LEO) satellites and plans to launch the first phase of 4,425 LEO satellites by March 2027. According to SpaceX's FCC filings, roughly 1,600 LEO satellites deployed early would be flying at roughly 550 km above the earth surface in specific orbits. These fast moving satellites would talk to base stations and user terminals as well as other satellites. We assume these communications to happen along straight lines at the speed of light in vacuum.

Fig. 1(a) shows a constellation of 400 satellites with 20 polar orbits each containing 20 satellites. Polar orbits are perpendiculars to the Equator, but orbits can have lower inclinations, in which case, the orbits look as in Fig. 1(b).

3 Problem

The goal is to optimally connect satellites via ISLs at a fixed point in time (so that they are not moving), in order to serve Internet traffic.

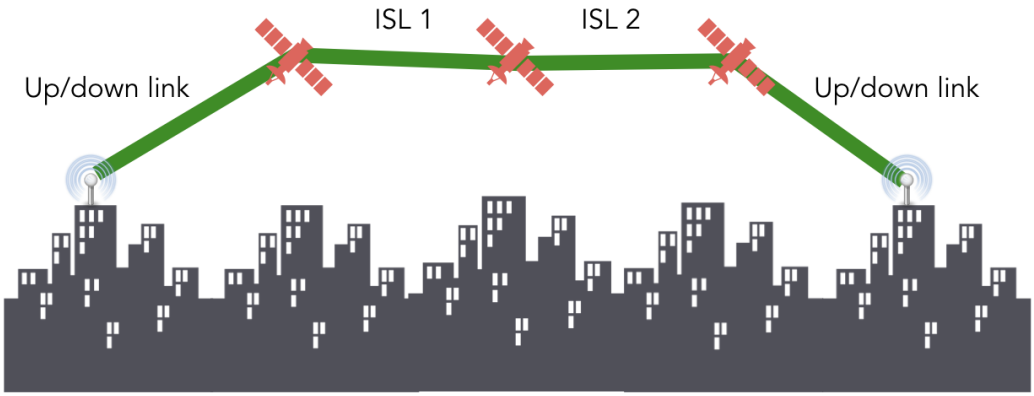


Figure 2: A typical end-to-end path over the satellite constellation.

The satellite positions are already provided so that you do not spend time on understanding the physics behind satellite trajectories. We provide you the positions of all 1600 satellites, the traffic matrix, all valid ISLs, all city-satellite up/down links, and an objective function. We also provide you some constraints to follow while setting up the ISLs. Your job is to come up with an optimal scheme to setup the straight-line ISLs between satellites in order to transfer traffic. Fig. 2 shows a typical path consisting of city-satellite up/down links and ISLs.

4 What do you get from us?

In this section we discuss the input data, scripts, constraints and objective functions that we provide you. A student project skeleton (i.e. containing input data and scripts to check your solution locally) is available under < your_gitlab_repo >/project3

4.1 Inputs

In the project skeleton, you will find a directory `input_data`. This directory contains the following comma-separated files:

- `sat_positions.txt`

  [satellite ID],[orbit ID],[satellite ID within orbit],[latitude],[longitude],[altitude in km above the earth surface]

- `cities.txt`:

[city ID],[city name],[latitude],[longitude],[weight of city]

- `city_pairs.txt`:

  [city ID 1],[city ID 2],[geodesic distance between them in km]

- `city_coverage.txt`: This file contains the links between cities and the satellites (visible from those cities).

  [city ID],[satellite ID],[link length in km]

- `valid_isls.txt` This file contains all valid ISLs from which you should select ISLs to be setup.

  [satellite ID 1],[satellite ID 2],[link length in km]

4.2 Constraints

Below are the constraints to be followed while setting up ISLs between satellites:

- ISLs can only be picked from the ones given in `valid_isls.txt` (discussed in §4.1). This is to make sure that no ISL in the network is longer than a threshold.
- Each satellite can have at most 4 ISLs. A city can have as many up/down links (1 per visible satellite) with different visible satellites as needed (check file `city_coverage.txt` discussed in §4.1).
- All ISLs are full duplex. So, if an ISL is setup between satA and satB, the ISL in the opposite direction is also implicitly setup and counts towards 4-ISLs-per-satellite constraint.

4.3 Objective function

In order to specify the objective functions, let us define two terms:

- `Stretch`: For a city pair, the stretch between them is the ratio of the distance along the shortest path between the pair over the network and the geodesic distance between them. For the geodesic distance between pairs of cities, check the file `city_pairs.txt` discussed in §4.1.
- `Hop count`: The number of hops between city pairs along the shortest path between them over the network.

For the pairs of cities in the file `city_pairs.txt` discussed in §4.1, compute stretch (s) and hop count (h) of the shortest (in length) path, both weighted by the product of city weights (for weights, check `cities.txt` discussed in §4.1). The **objective** is to minimize the sum (M) of the average weighted stretch and the average weighted hop count. **If you are coding in Python, there is already a script provided which computes the metric** (discussed in §4.4).

Note that the metric considers the stretch and hop count of the **shortest paths in terms of path length**. So, in a good solution, the network topology should offer shortest paths with low hop count.

> M is your score in the leaderboard. The lower the value, the better. M should be a positive value.

4.4 Scripts

In the project skeleton, you will find a directory `scripts`. This directory contains the following scripts:

- `check_score.py`: This is the script you should run to validate your solution and compute the objective function value locally. If executing this throws any validation error, the leader board would also definitely fail to compute your score correctly.
- `util.py`: This script provides various utility functions written in Python with explanations therein.
- `visualize.py and static_html/*`: This script (and the static files) can be used to generate visualizations. It works with Python 3.5+. It generates a visualization file `viz.html` within the same directory. You can visualize end-to-end paths for various city pairs (use city IDs) by calling `generate_html(ID1, ID2)`

> You are free to use any of the scripts above in any way to generate your solution.

5 Part-1

Your baseline solution should follow the specified constraints in §4.2 and should be able to generate end-to-end paths for each city pair.

Here is a suggestion on how to do this (you can come up with your own scheme). You can connect the satellites in a grid. Satellite k of orbit p sets up ISLs with satellites $k+1$ and $k-1$ of the same orbit p, and satellites k of adjacent orbits $p-1$ and $p+1$. For orbit IDs and satellite IDs within each orbit, check file `sat_positions.txt` discussed in §4.1. This scheme generates a regular grid-like network of satellites.

Threshold #1: achieving a positive score of 12 or less will give you at least 4 points. If threshold #1 is not reached by **05 May 2021 at 15:00**, the total number of points will be reduced by 20%.

6 Part-2

Here you are free to explore the solution space and come up with sets of ISLs that reduce the objective function value further. You are free to use any algorithm/heuristic that you think makes sense. Some suggestions include:

- Generating other regular grid-like networks with different ISL orientations.
- Generating random regular graphs.
- Writing a linear program to solve the problem. Note that we do not guarantee that the program would converge within reasonable time given the scale of the problem.

We do not vouch for any suggestion. You are free to come up with your own unique solution.

Threshold #2: achieving a positive score of 8.5 or less will give you at least 8 points. There is no deadline for this threshold. Note: The lower the score, the better.

7 What do we get from you?

In your Git repository, this project comes under `project3`.
You should use following directory structure to submit your solution:

```
project3
|-- input_data
|-- scripts
|-- output_data
    |-- sat_links.txt
|-- explain.md
|-- team_name.txt
```

Do not modify `input_data` - it contains all the input files. All code should go under directory `scripts`. The directory `output_data` should contain the generated file **sat_links.txt** which contains the list of ISLs. File `sat_links.txt` should be comma-separated with each row specifying

[sat ID 1],[sat ID 2]

* Do not include links of the form [sat ID 2],[sat ID 1] as ISLs are full duplex.

* Before the deadline, write a short description (up to 5 sentences) about how your algorithm works and store it in `explain.md`. Do not forget to add your team name in `team_name.txt`.

You can find the leader board at
http://bach20.ethz.ch/leaderboard_satnet.html

8 FAQ

Below are some frequently asked questions and corresponding answers:

- `How can we compute shortest paths between vertices?`
  For generating networks and finding out shortest paths, you can use Python `networkx` library. The python script provided uses Dijkstra's algorithm to compute shortest paths.

- `Which linear program solver should be used?`
  First, we do not guarantee that a linear program would be suitable to solve this assignment, given the scale of this problem. But if you want to explore linear programming, you can use Gurobi [8], OR-Tools [2], or any other solver that you are acquainted with.

- `Can ISLs be cherry-picked manually instead of writing code?`
  No. We expect you to setup ISLs programmatically. So, there has to be some logic behind selecting ISLs. Generating a random graph, for example, is fine, but handpicking ISLs is not fine.

References

[1] D. Etherington. Watch SpaceX launch 60 more Starlink satellites and attempt a Falcon 9 re-use record. `https://tinyurl.com/v2rru63`.

[2] Google. Google OR-Tools. `https://developers.google.com/optimization`.

[3] LeoSat. `http://leosat.com/`, 2018.

[4] NASA. GMAT tool. `https://software.nasa.gov/software/GSC-17177-1`.

[5] OneWeb. `http://www.oneweb.world/`, 2018.

[6] T. Pultarova. OneWeb weighing 2,000 more satellites. `https://tinyurl.com/ycqam3vb`, 2017.

[7] SpaceX. `http://www.spacex.com/`, 2018.

[8] G. Team. Gurobi Optimization. `http://www.gurobi.com/`.