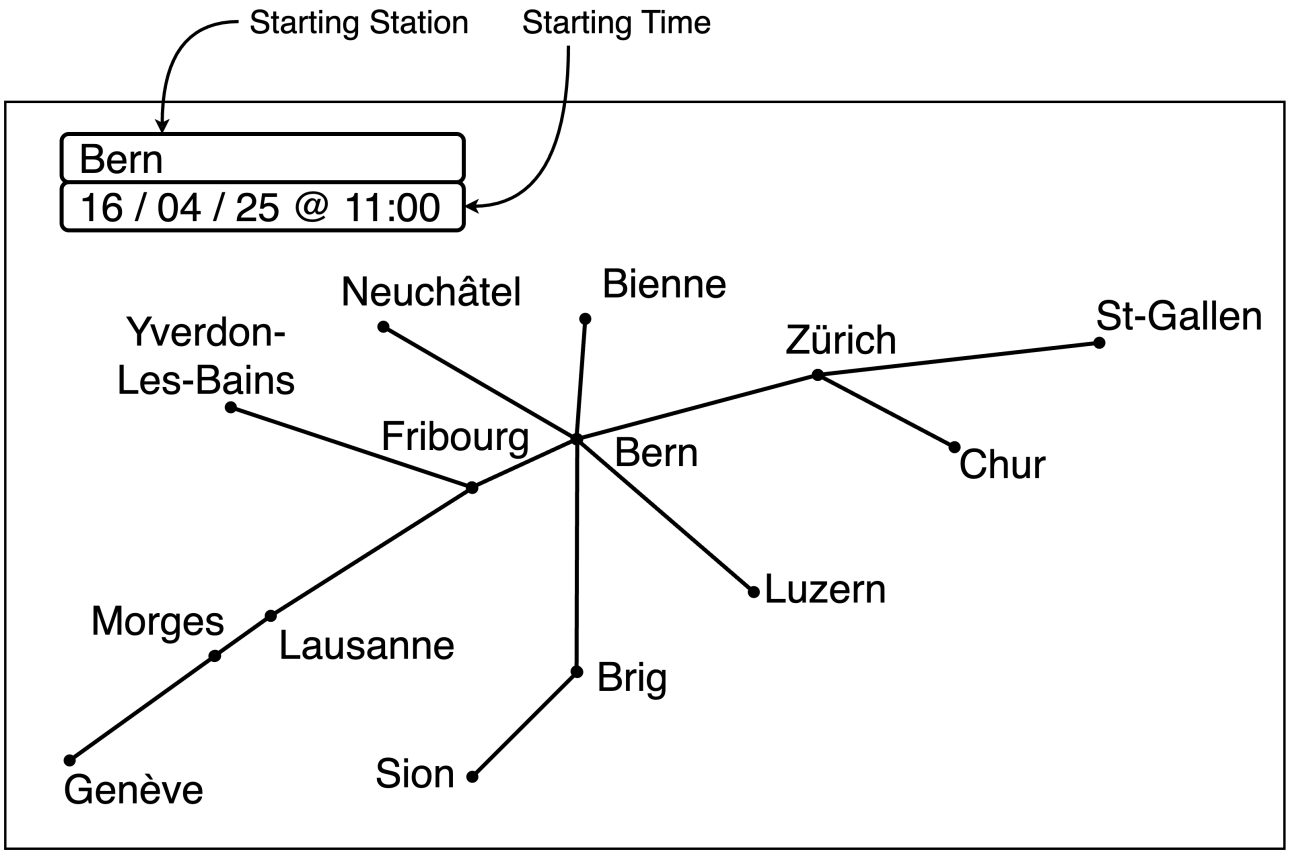
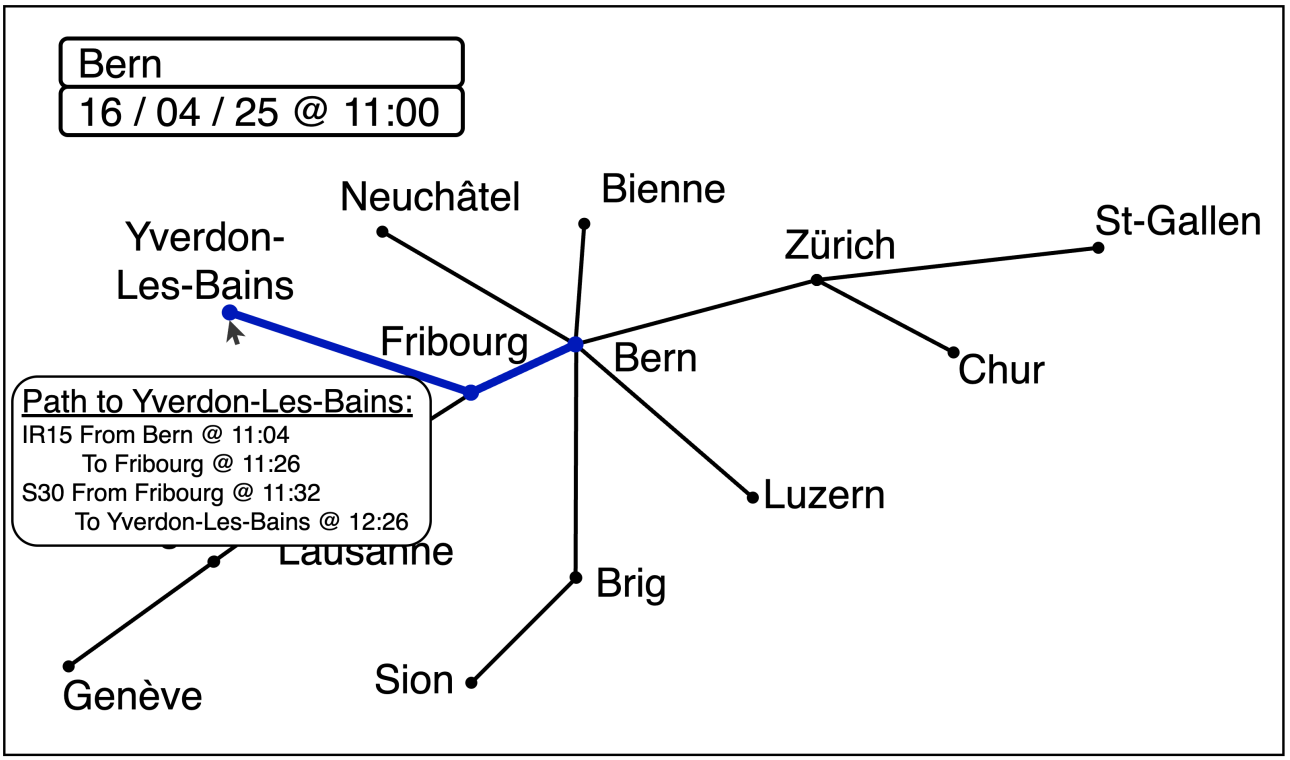
Milestone 2 (18th April, 5pm)

Our project aims to calculate and visualize a ***catogram*** of the travel time taken by public transport, starting from a specific location and time, as depicted in the following sketch:



The user must input a starting location and starting time because public transport travel times are not symmetrical. For instance, if you depart at 11:00 AM on April 16th, it takes 51 minutes to travel from Lausanne to Geneva, but only 44 minutes to do the trip in the opposite direction.
In addition, starting at a different time can change the travel time due to irregular transport schedules; and if we computed the travel time between all pairs of points on the map instead of from a single starting location, the result would be harder to interpret and sometimes ill-defined (as in the asymmetric case above).

Once the visualization is computed and displayed, the user can hover over a station, which will create a window displaying the quickest itinerary to that station, as illustrated in the following sketch:



Data Processing

The data processing is split into two parts, the first part is independent of the starting location and time and so can be done once. The second part compute the earliest arrival time from a starting location and time.

Preprocessing

The dataset contains over a billion trips, making over 12 billions stops in 30'000 destinations. These numbers are too big, so we want to preprocess the dataset to only keep trains or transportation in a 10 km radius around EPFL.

Quickest path discovery

Once the preprocessing is complete, Dijkstra's algorithm is employed to compute the quickest arrival time for each destination. This algorithm provides the earliest arrival time to all destinations along with the corresponding path to reach those destinations.

Note that this algorithm give the earliest arrival time and not necessary the shortest travel time to a destination. For example, starting from Lausanne at 11:20, this algorithm will give the following path to Bière:

- IR 15 From Lausanne @ 11:23 to Morges @ 11:33
- R 56 From Morges @ 11:54 to Bière @ 12:25

Giving us a total travel time of 62 minutes, including 21 minutes of waiting at Morges.

If we query the SBB website with the same trip we get the following result:

- R 8 From Lausanne @ 11:34 to Morges @ 11:48
- R 56 From Morges @ 11:54 to Bière @ 12:25

Giving us a total travel time of 51 minutes, including only 6 minutes of waiting at Morges.

The difference is caused by our algorithm choosing the earliest available trip to Morges even though the next one would also work to catch the R 56 connection.

Vizualization

Once we have the earliest arrival time for every destination and the route to reach them, we visualize it using a spring layout.
There are three types of springs:

- Between the source and each destination, with a preferred length proportional to the total travel time.
- Between each destination and the previous stop on the way from the source, if it exists, with a preferred length proportional to the time of the last leg. This is to encourage the layout to reflect the paths taken.
- Between each stop and its geographical location, with a preferred distance of 0. This is to encourage the layout to not diverge too much from actual geography.

We set the strengths of these spring forces to be in appropriate balance.

Tools and classes

For our project, the classes of most interest are the classes on maps, graphs, storytelling with data and creative coding, and D3.js.
For the data processing, we are using jupyter notebooks, pandas and Python's csv libraries. For the vizualization layout, we are using the force-directed layout module of D3.

Extra ideas of improvements

We have three main ideas to improve our vizualization:

- Warping other map features like lakes, mountains, and borders to enhance the visualization of the warping. However, this approach faces a challenge in preserving the order between cities. For instance, traveling from Bussigny to Yverdon-Les-Bains is sometimes quicker via Renens. Consequently, our visualization will attempt to align these cities in the order Bussigny -> Renens -> Yverdon-Les-Bains. We thus cannot rely on the relative position between those cities and map features to find a new location for them.
- We are currently only considering travel using public transport but sometime walking is faster or not reachable by public transport. Taking these walkable distances could make the travel time more realistic.

- Another way to make the travel time more realistic is to take into account transfer time at station. We currently assume an instantaneous transfer between the platform of a train stations and the surrounding bus stops. The dataset contains a `transfer.txt` file which gives us this minimum transfer time.

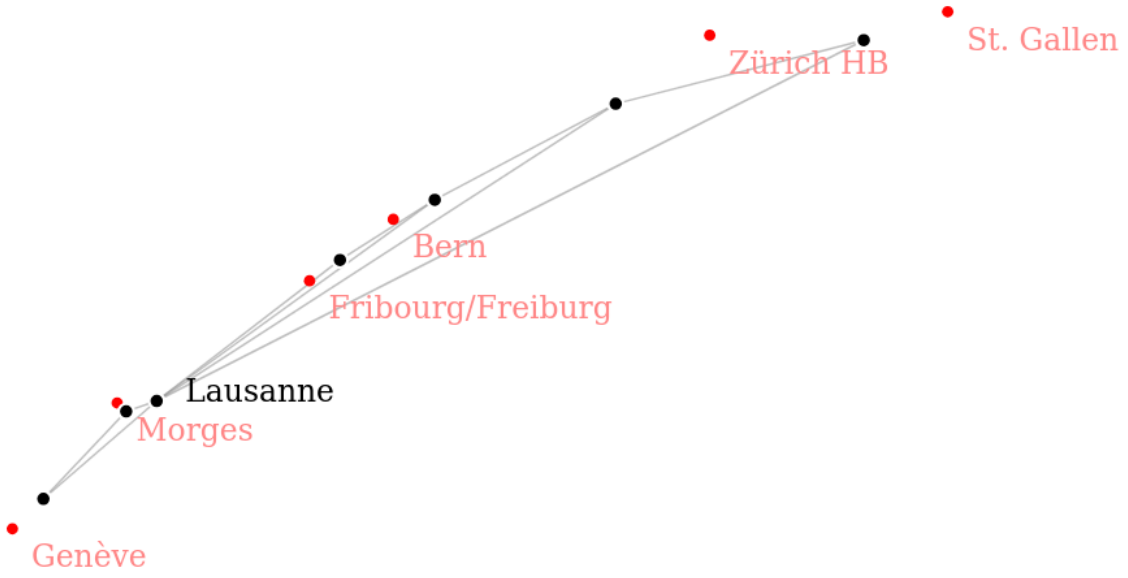
Prototype

To have a local version of the webpage which display the travel time from Lausanne to a few cities of Switzerland, you need to have `npm` installed on your machine and then run the following command

inside the `www` directory:

```
$ npm install && npm run dev
```

It will display a link to open in your web-browser to see the vizualization which looks like the following:



The red dots are the original location of the cities and the black one are the locations adjusted for the travel time from Lausanne at 16:20 on Apr 17. The gray lines are the springs used in the layout.