REPORT ALGO PROJECT

Tailored Tourist Tours App

Project realized by Théodore PREVOT and Christophe SAURY

Introduction:

France is a tourist country by excellence, and there are tens of millions of tourists who come to France each year to admire thousands of sites, monuments, festivities, museums ...

The aim of this project is to help tourist agencies proposing solutions to tourists' requests in general, and more especially to those that are like the previous requests taking into consideration place and time constraints.

Tourists that are visiting France have specific constraints:

- Limited time for their visit.
- Possibility of only visiting places that are near each other.
- Limited budget for the dispenses.
- Taste for certain types of sites more than others …

The two most important requirements among these are the two following:

• An estimated period, during which the site is considered to belong to

• A will localized place, in which the site is placed

Our solution to this problem:

Our solution to this issue is to make a web app that could be used by tourists and tourist agencies to generate tailored tourist tours to visit the different sites and museums of France.

The tourists can enter their length of stay, localization and their preferences and our algorithm will generate a tour for them.

Vocabulary Used:

Trip: A trip represents the generated list of sites to visit for a single day.

Tour: A tour represents the list of trips corresponding to the number of days of the tour.

Site: A site is a location to visit irrespective of the type of place it is.

Building: A building is a type of location to visit.

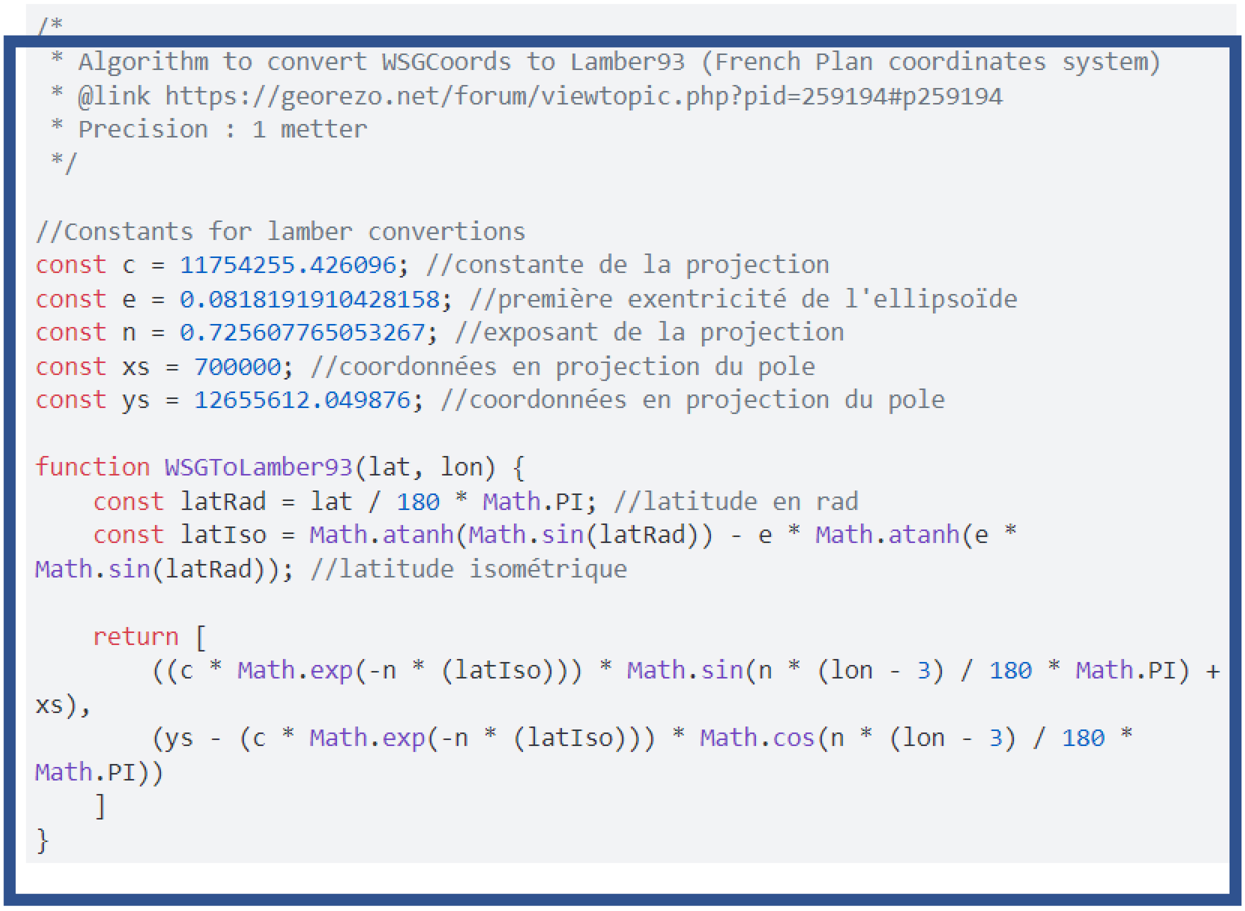
Museum: A building is another type of location to visit.

Steps taken to make the solution:

There were many steps to making this app.

First, we had to get data from online datasets from French governement to get the list of sites and museums that exist in France. We had to look for a dataset that had the name, localization, and historical period of a site. It would have been interesting to also have the price of these sites in our data, but we weren't able to find a dataset that gives us this information. We then had to clean the data pulled from the API so that we can have usable data to pull for the different sites. Notably we had to make the historical periods be counted be written in the same way in the database by counting it in centuries everywhere which would allow us to do specific searches. We used a mySQL database with JDBC to store the cleaned data from the APIS. The monument dataset contained 45 0000 rows and the museum dataset contained 1200 rows.

We also decided to convert the localization data of the different sites from WGS83 polar coordinates to a plan projection in meters: lamber93. It allows us to easily compute areas and distances. We use the following mathematical formulas to do those conversions:



```
/*
 * Algorithm to convert WSGCoords to Lamber93 (French Plan coordinates system)
 * @link https://georezo.net/forum/viewtopic.php?pid=259194#p259194
 * Precision : 1 metter
 */

//Constants for lamber convertions
const c = 11754255.426096; //constante de la projection
const e = 0.0818191910428158; //première exentricité de l'ellipsoïde
const n = 0.725607765053267; //exposant de la projection
const xs = 700000; //coordonnées en projection du pole
const ys = 12655612.049876; //coordonnées en projection du pole

function WSGToLamber93(lat, lon) {
    const latRad = lat / 180 * Math.PI; //latitude en rad
    const latIso = Math.atanh(Math.sin(latRad)) - e * Math.atanh(e *
Math.sin(latRad)); //latitude isométrique

    return [
        ((c * Math.exp(-n * (latIso))) * Math.sin(n * (lon - 3) / 180 * Math.PI) +
xs),
        (ys - (c * Math.exp(-n * (latIso))) * Math.cos(n * (lon - 3) / 180 *
Math.PI))
    ]
}
```

Here the return is composed of the value in meters of the position of the site based on the Lamber93 projection.

Secondly, we had to make frontend pages for users to access the website and enter their specific constraints. Those frontend pages can get the constraints and post them to the algorithm for it to then generate a tour which is showcased in another page.

We had to make the mapping of the different pages and realize the algorithm that generates the tours. For this algorithm, we operate under the two following assumptions:

- The tourist lives at a certain place during his stay hence, when generating tours, we need to start and end the tours from the same place to form a loop with the place he is staying at.
- The tourist not only wants to see different sites but also wants to visit different parts of France and see different neighborhoods.

Include a screenshot of the frontend page.

We decided to implement the following algorithm:

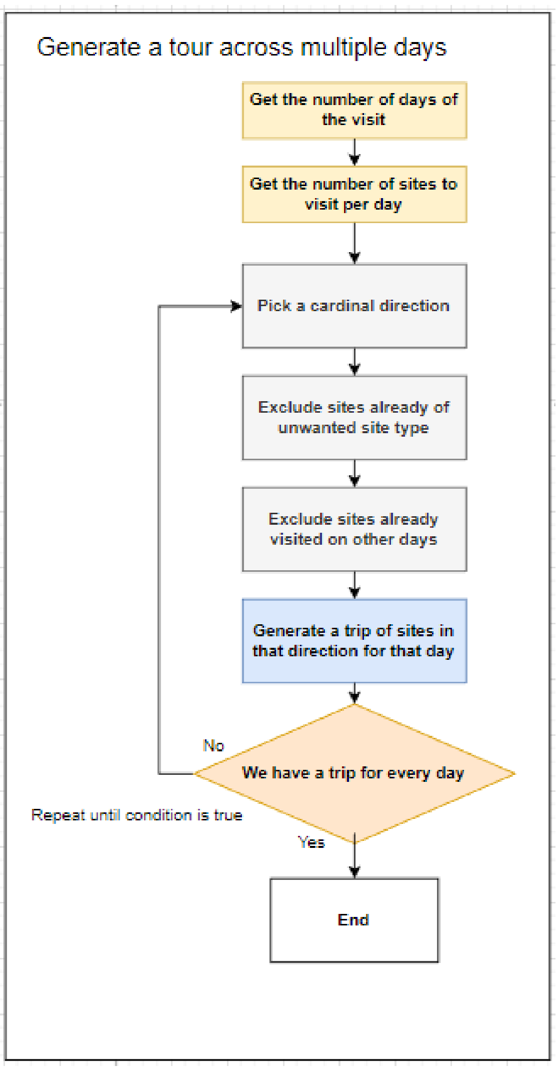
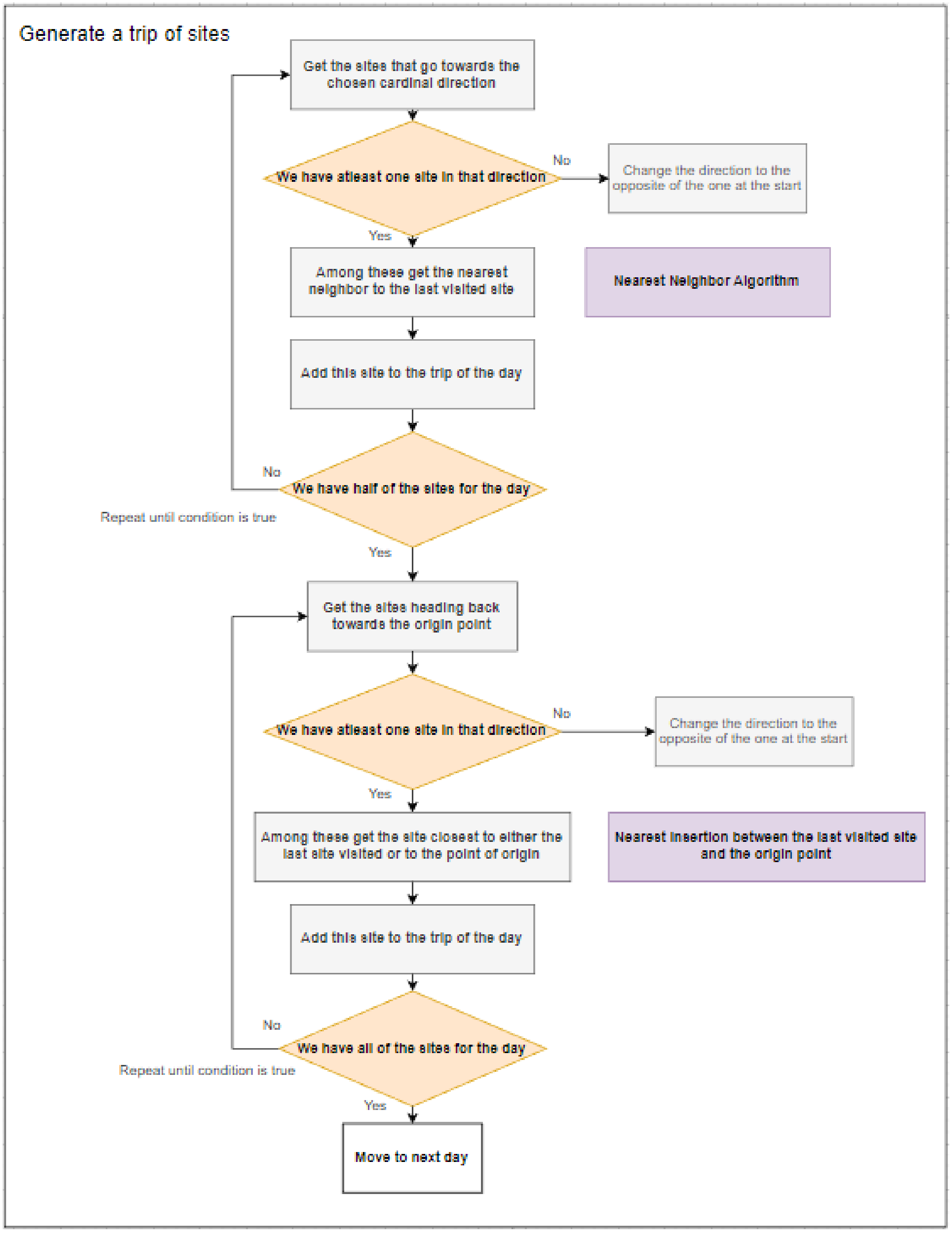

Generate a tour across multiple days
Get the number of days of the visit
Get the number of sites to visit per day
Pick a cardinal direction
Exclude sites already of unwanted site type
Exclude sites already visited on other days
Generate a trip of sites in that direction for that day
We have a trip for every day
No
Repeat until condition is true
Yes
End

Figure 1 - Representation of our algorithm to generate a tour over multiple days.

The reason why we pick a cardinal direction at the beginning of the generation of a day's trip is because of the second assumption which is that tourists want to see different areas of a city not just different sites. The tourists don't necessarily want the most optimal path in terms of distance, they want a tour that is close to optimal in terms of distance while respecting their constraints and seeing different areas.

The way we chose for the user to see different parts of a city is to have the tourist visit sites according to a different cardinal direction every day.



Generate a trip of sites
Get the sites that go towards the chosen cardinal direction
We have atleast one site in that direction
No
Change the direction to the opposite of the one at the start
Yes
Among these get the nearest neighbor to the last visited site
Nearest Neighbor Algorithm
Add this site to the trip of the day
No
We have half of the sites for the day
Repeat until condition is true
Yes
Get the sites heading back towards the origin point
We have atleast one site in that direction
No
Change the direction to the opposite of the one at the start
Yes
Among these get the site closest to either the last site visited or to the point of origin
Nearest insertion between the last visited site and the origin point
Add this site to the trip of the day
No
We have all of the sites for the day
Repeat until condition is true
Yes
Move to next day

Figure 2 - Representation of the algorithm used to generate tours for a given day.

To generate a trip, the algorithm goes towards a certain cardinal direction until it added half of the desired number of sites to the trip. It then looks for more sites between the last visited node and the original point so that the trip can loop.

Schemas of an example of how the algorithm works for a trip that includes 4 sites. The grey rectangle represents the search area for the next sites.

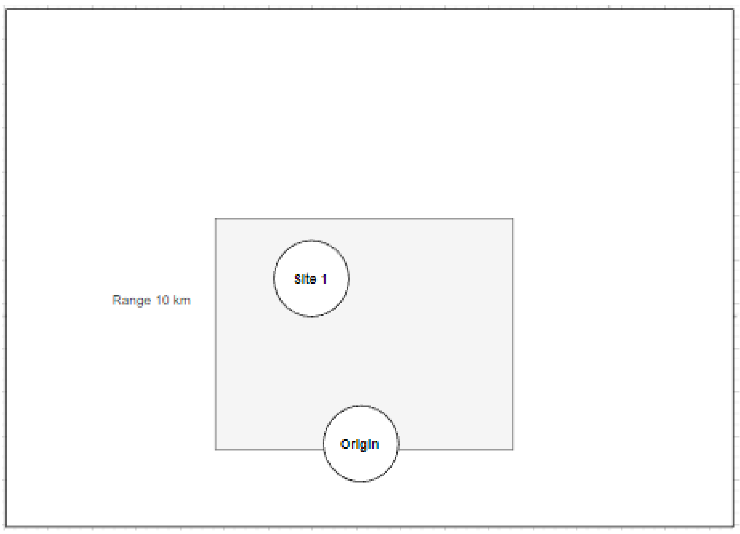


Figure 3 - First step of the algo

The algorithm starts by looking for sites in a specific direction, here the direction is north. We use the nearest neighbor algorithm to select the specific site we want to add.

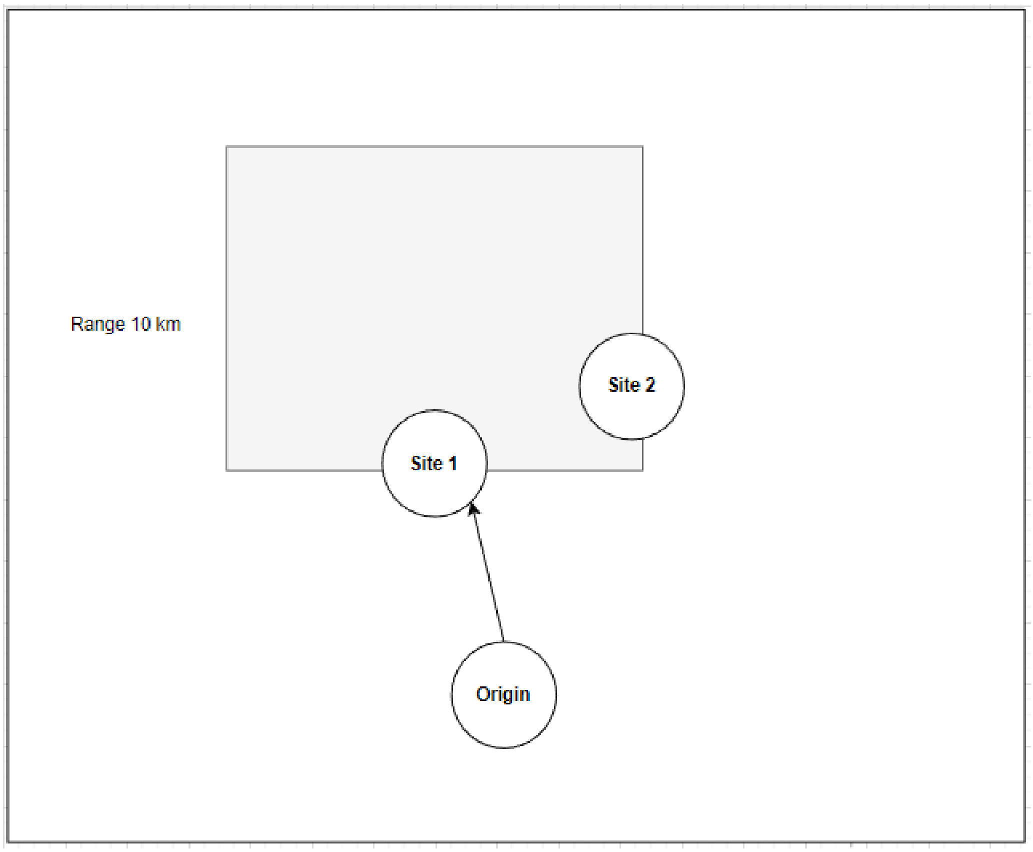


Figure 4 - Second step of the algorithm

We continue looking for sites in that direction while still using the nearest neighbor algorithm. We add the site found.

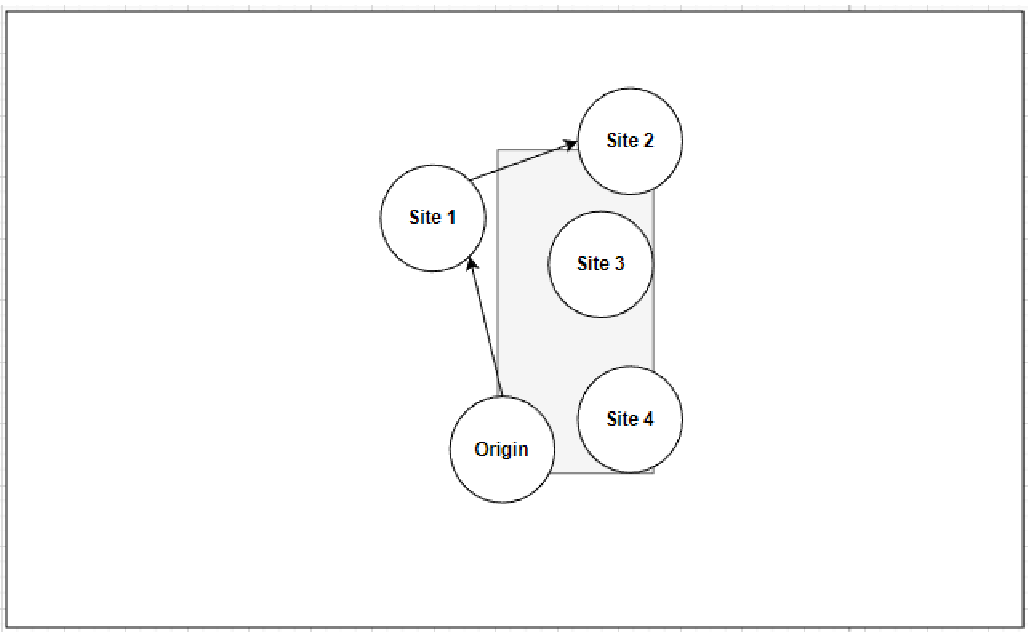


Figure 5 - Third step of the algo

We already have 2 sites out of 4 in our trip hence we start looking for sites in the area between the last visited city and the point of origin. We use the nearest insertion between the last visited site and the origin to select the remaining sites all at once. The nearest insertion algorithm will add the sites closest to either the last visited site or to the point of origin. Hence, we then end up with this.

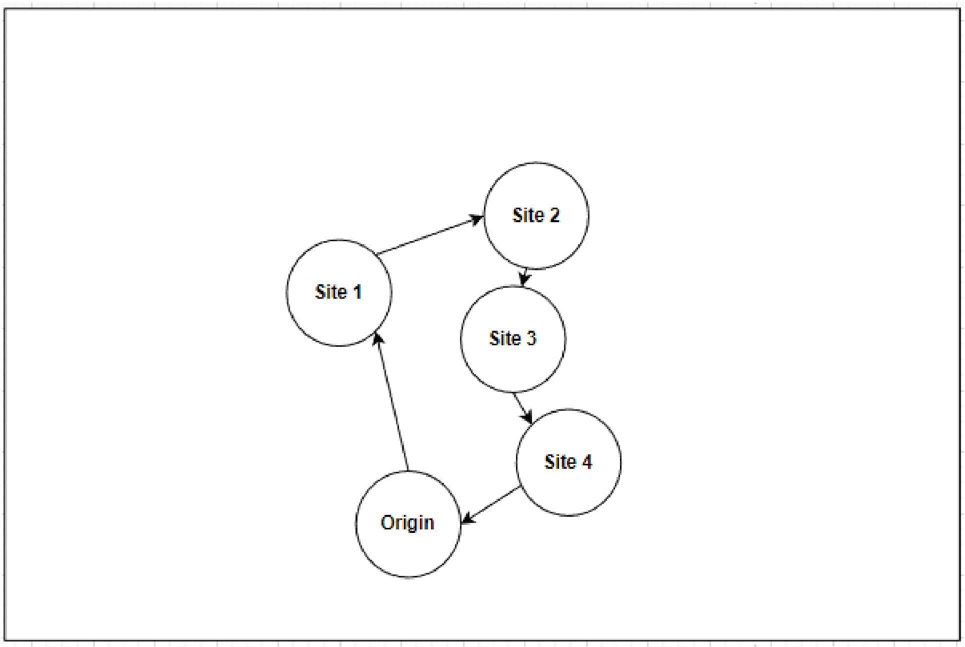


Figure 6 - Final result of the algorithm

We chose to use the nearest neighbor for the first part of the trip because we decided to make the trip keep going from neighbor to another neighbor that is located at a specific cardinal direction of the previous neighbor. The nearest neighbor heuristic algorithm is a greedy algorithm with suboptimal quality of result, but with high computational speed. This algorithm won't always find the shortest path, but it has a time complexity of O(n^2).

In our situation, we are not looking for the shortest path because we assume that the tourist doesn't just want to visit sites but also wants to see the neighborhoods. The tourists also have other constraints that they might find more important than having the shortest path possible. Optimizing computational speed also makes the app more enjoyable than finding a shorter path that takes forever to generate.

We then choose a nearest insertion between the last visited site and the point of origin for the way back once we have visited half of the sites for the day. The nearest insertion heuristic algorithm also has a time complexity of O(n^2) while demonstrating better code quality in this scenario then the nearest neighbor algorithm.

The execution time is somewhat long for our complete algorithm, but the algorithm is for the most part done directly in the SQL queries hence even though execution time is long, we can have a lot of users starting queries at the same time.

Ways to improve the algorithm:

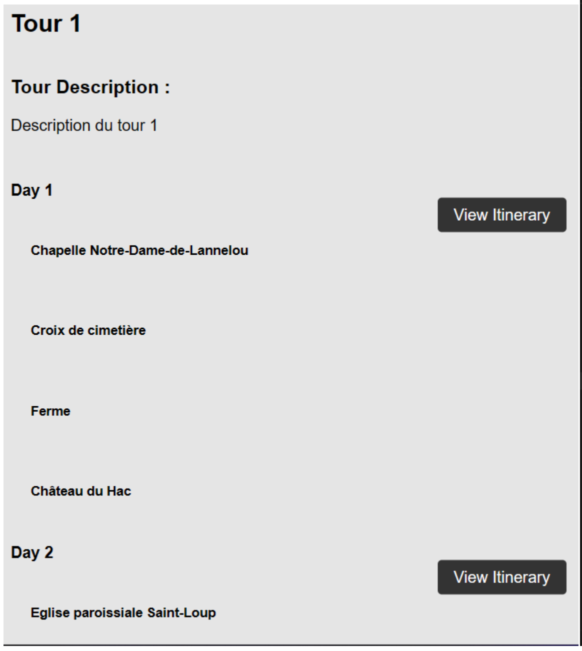
One way we can improve the algorithm speed of computations would be by parallelizing the trip generations. Hence when we would generate a tour, it would generate the trips for each day simultaneously.

It would have been interesting to add a visualization of the itinerary of a tour to the app, we could use for example the google maps API.

We could also use hashmaps for the adjacency list representation of our graph instead of using arraylists of arraylists.

Final Result:

Hence, we get this result once we ask the app to generate a tour for us after feeding it parameters through the search page:



Tour 1

Tour Description :

Description du tour 1

Day 1

View Itinerary

Chapelle Notre-Dame-de-Lannelou

Croix de cimetière

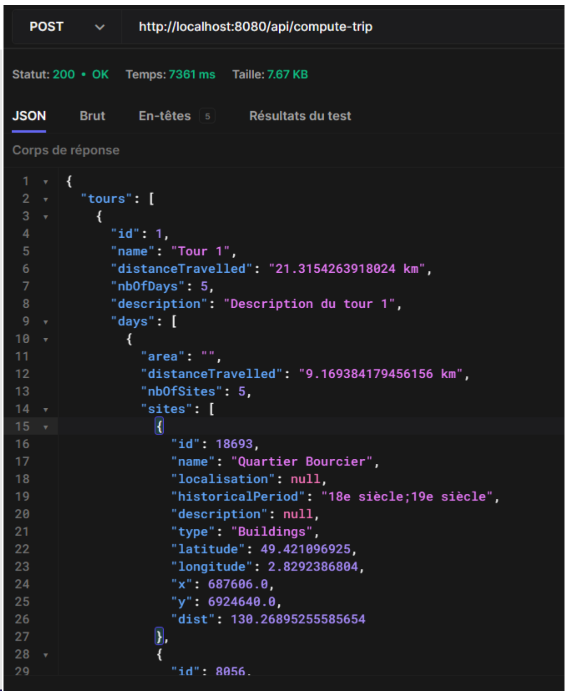
Ferme

Château du Hac

Day 2

View Itinerary

Eglise paroissiale Saint-Loup



POST http://localhost:8080/api/compute-trip

Statut: 200 • OK Temps: 7361 ms Taille: 7.67 KB

Corps de réponse

```
{
  "tours": [
    {
      "id": 1,
      "name": "Tour 1",
      "distanceTravelled": "21.3154263918024 km",
      "nbOfDays": 5,
      "description": "Description du tour 1",
      "days": [
        {
          "area": "",
          "distanceTravelled": "9.169384179456156 km",
          "nbOfSites": 5,
          "sites": [
            {
              "id": 18693,
              "name": "Quartier Bourcier",
              "localisation": null,
              "historicalPeriod": "18e siècle;19e siècle",
              "description": null,
              "type": "Buildings",
              "latitude": 49.421096925,
              "longitude": 2.8292386804,
              "x": 687606.0,
              "y": 6924640.0,
              "dist": 130.26895255585654
            },
            {
              "id": 8056,
```

Conclusion

This project allowed us to use the notions related to graphs and heuristic algorithms which we saw in class to solve a concrete problem. We applied our interpretation of the problem to make a personalized algorithm that uses both the nearest neighbor heuristic algorithm and the nearest insertion algorithm.