SHORTEST PATH TREES AND REACH IN ROAD NETWORKS

09/01/2017

Zuli HUANG
Mingkun LIU

1 Properties of Shortest-Path Trees

To identify the number of different possible points we may be located in at the current moment of our trip, we choose the example vertex:

```
v 298251056 2277576 48783477
```

1.1 Question

You have been traveling for time exactly t1 = 1 hour from the fixed starting point v towards your destination, which may be any sufficiently distant point in the network.

We can obtain Figure 1 generated from the example vertex. Totally, there are 926 sufficiently distant points.

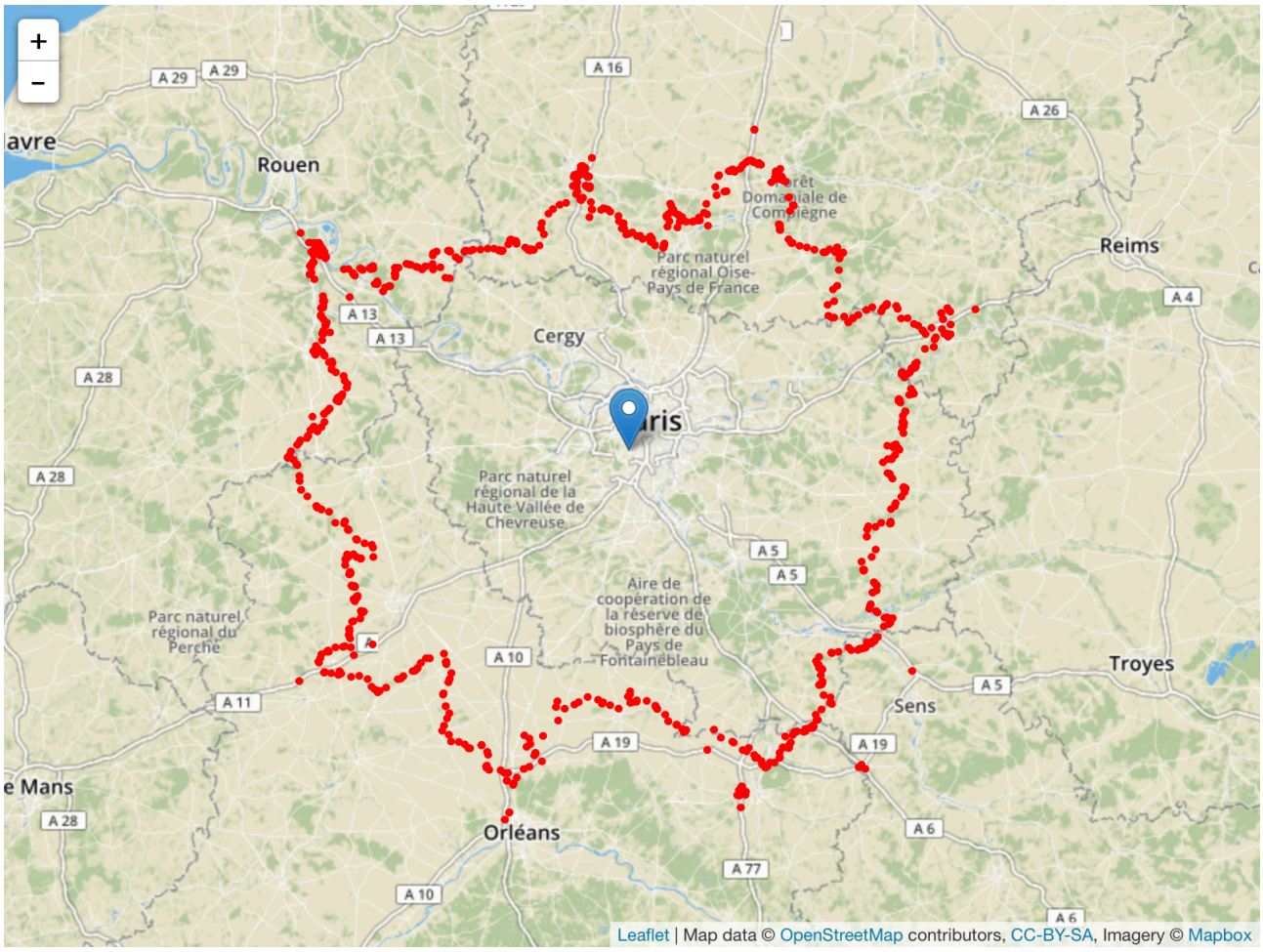

Figure 1: Points with 1h distance to vertex 298251056

1.2 Question

You have been traveling for time exactly t2 = 2 hours from the fixed starting point v towards your

destination, which may be any sufficiently distant point in the network.

It's similar to Q1.1. We choose the same center vertex v in the precedent question. And we obtain a map of sufficiently distant points. Totally, there are 2436 sufficiently distant ones.

Figure 2: Points with 2h distance to vertex 298251056

1.3 Question

You have been traveling for time exactly t1 = 1 hour from the fixed starting point v towards your destination, when it is known that your destination is at least t2 = 2 hours' way away from your starting point v.

We just need to back propagate points, obtained in the question 1.2, with 1h distance. And we get several appropriate points.

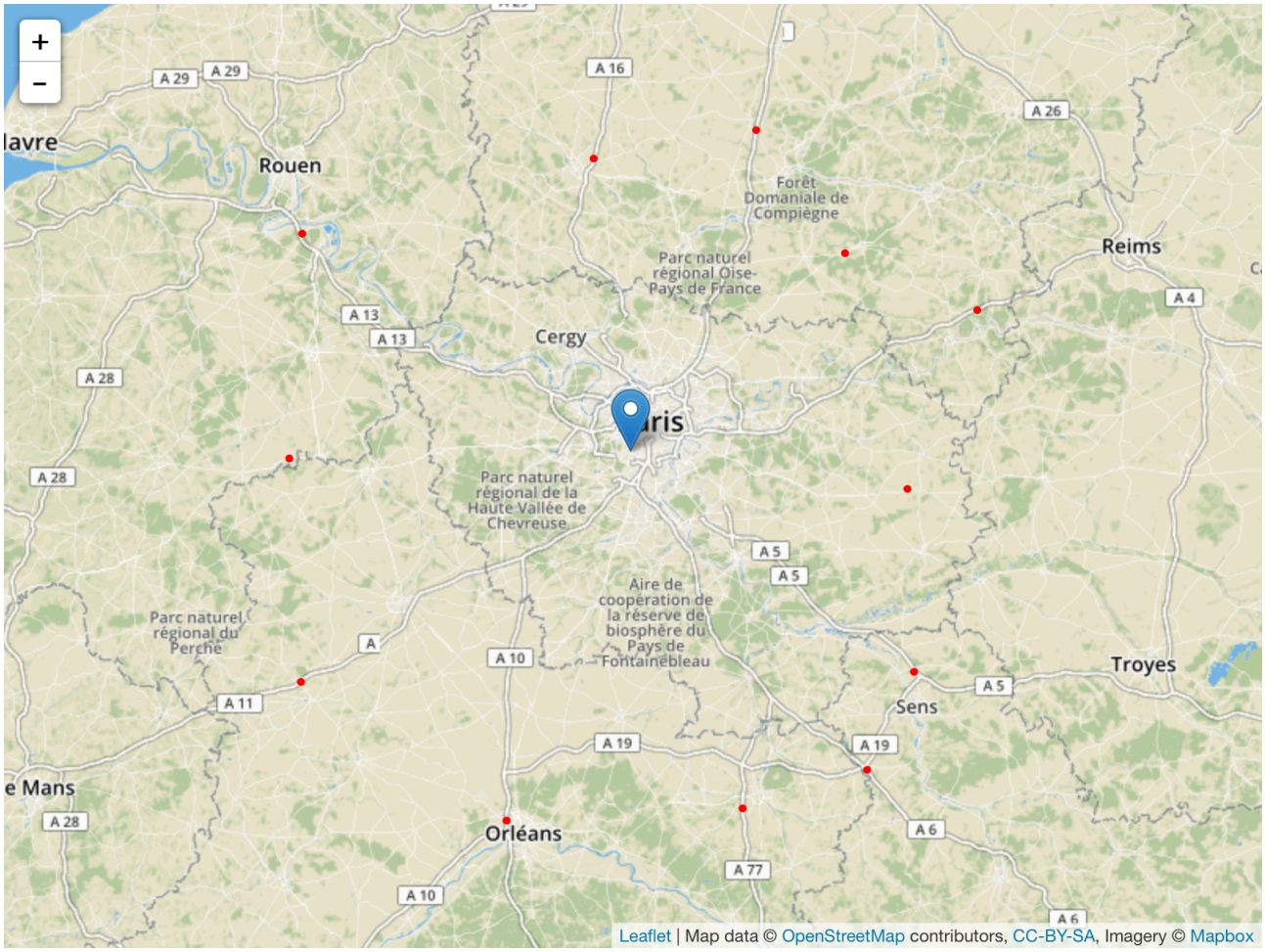

Figure 3: Points with 1h distance to v during a travel to a destination at least 2h away from v

1.4 Methods

Brief discussion of the graph algorithms used to answer the above questions.
Algorithm that we use to handle question 1.1 to 1.3.

Algorithm 1: Find points with a certain distance to a given vertex

Data: An directed graph $G = (V, E)$, non negative wights, a source node $v \in V$, travel time t, $\omega : E \to \mathbb{R}$. Results generated in the algorithm: $d : V \to \mathbb{R}$ distance to v, $settled : V \to boolean$ and $p : V \to V$, predecessor of a vertex $x \in V$

Result: Set of points with distance t to v

```
for x ∈ V do
    settled(x) := false;
    d(x) := ∞;
Q.initialize, Qb.initialize;
Q.add(v, v, 0);
d(v) = 0;
while not Q.empty do
    (x, prev, dist) := extractmin(Q, Qb);
    if dist ⩽ d(x) then
        if dist > t and d(prev) ⩽ t then
            points.add(x, prev, t-d(prev);
    else
        continue;
    if settled(x) then
        continue;
    else
        settled(x) := true
    for y ∈ N(x) do
        if not settled(y) then
            if ω(x, y) + d(x) ⩽ d(y) then
                d[y]:=ω(x, y) + d(x)
                if d[x] ⩽ t then
                    Q.add(y, x, ω(x, y) + d(x));
                else
                    Qb.add(y, x, ω(x, y) + d(x));
                p[y]:=x
return points;
```

To find all sufficiently distant points without searching whole map, the main idea is to create two priority queues – Q and Qb. Q stores the vertices whose predecessor that can be visited within the travel time limit. Qb stores the vertices whose predecessor that can't be visited within the travel time limit. It means that there is no more possible points when Q is empty. Another point is add a tuple element to priority queue, (vertex id, its predecessor id, distance). This point aims to successfully find those points, in different edge, that lead to the same vertex. We can observe that there are few points obtained in Figure 3. We can deduce that lots of vertices and edges will never be visited when we travel a sufficiently distant destination. And we find that the points in Figure 3 are all in "highways" connecting different cities.