PI: Shortest Path Trees and Reach in Road Networks

Chane-Sane Elliot, Derchu Joffrey

INF421, 2016

Abstract. We study the road networks of France and analyze the way people tend to cover most of their way using a relatively small subset of the road network. Thus we give a 2-approximation algorithm to calculate the reach of randomly chosen vertices and we draw conclusions concerning the structure of the road network.

Table of Contents

Please find the programs to test our results in the attached directory. Instructions are available in the readme.txt file.

1 First Task: Properties of Shortest-Path Trees

Before we start. We first need to implement a network on Java. We choose to represent a graph using adjacency lists. We represent edges with a class Edge.java, vertices with a class Vertex.java and we put them all in a class Graph.java which contains a map of vertices and a map linking each vertex to its adjacency list. Finally, in a class Main.java, we write methods to read and write graphs as well as the methods solving the questions.

1.1 Problem

Suppose you are a traveler who always follows quickest paths from your starting point (a vertex) to your destination (another vertex). Here, the time needed to travel along a path is measured naturally, as the sum of all travel times of all the directed edges it consists of. We need to find the points you may be located in at the current moment of your trip, when you have been traveling for time exactly t1 from the fixed starting point v towards your destination, which may be any sufficiently distant point in the network.

Remark 1. Those points are the points which are exactly at distance t1 from v (following quickest path). Indeed, if a point p is at distance t1, then you can reach it in t1 when you are traveling to a destination at least t1 away from v (this point). Reciprocally, if you can reach it in t1 when you are traveling to a destination at least t1 away from v following a quickest path, then you can't reach it in less than t1, because if you could, the quickest way from v to the distant point would be different (going from the starting to p and ten from p to the end point would take less than t1). Hence the point is at exactly t1 from v.

Algorithm and implementation We use Dijkstra's algorithm to compute the distance between v and other points. At first we looked for all the vertices at distance t1 plus or minus a minute. Later we changed our program to find all the points (vertices or points on an arc) at exactly t1 to have a more exhaustive result. To do that we had to be able to know from which vertex v' we came when we reached a vertex v, so that we could find a point on the arc if the distance became greater than t1 between v and v'. Thus we used a class Node.java instead of Vertex.java to store where we came from. We save the appropriate nodes v' to be able to compute the points at distance t1.

Complexity Assuming the arcs have a limited length l, we can stop computing distances when we are at a distance t1+2*l from v for example. The Dijkstra's algorithm has a $\mathcal{O}(|E|log|V|)$ and the algorithm runs faster if t1 is small enough.

A useful application We need to find the points you may be located in at the current moment of your trip, when you have been traveling for time exactly t1 from the fixed starting point v towards your destination, when it is known that your destination is at least t2 away from your starting point v. We only need to find the points at t2 and trace back to the points at t1 along the path. The complexity is $\mathcal{O}(|E|log|V|)$ to find the points at t2 and $\mathcal{O}(|V|)$ to find the points at t1, if we avoid visiting a vertex twice while finding the ancestor at t1 from v.

1.2 Pseudo-code

Question 1.1-1.2: Dijkstra's algorithm

```
Input: a directed graph g, a vertex v in g, a distance r
Output: Question1(v,g,r)=the set of points at distance r from v
     d=0;   ## current distance from v
     Q.initialize ## a priority queue containing v
     listaux={}
     while Q is not empty and d<r+90000:
        n=Q.poll
        d=distance(n,v)
        if (r>=distance(n.ancestor,v) and r<=distance(n,v):
            listaux.add(n)
        if n already visited:
            continue
        for (e an edge with origin in n):
            m=e.end
            t=d+longueur(e)
            if (t<distance(v,m)):
                distance(v,m)=t
                Q.add(m)
     list={}
     for (n in listaux):
        list.add(pn) ## pn is the point between n and n.ancestor
                        at distance r from v
     return list
```

Question 1.3

```
Input: a directed graph g, a vertex v in g, a distance r_int and
a distance r_end>r.int
Output: the set of points at distance r_int you pass through if
you go to points at distance at least t_end
     listaux={}
     for (n in Question1(v,g,r_end):
        while (n not already seen and distance(n.ancestor,v)>r_int):
            n=n.ancestor
        if (distance n.ancestor<r_int) listaux.add(n)
     list={}
     for (n in listaux):
        list.add(pn) ## pn is the point between n and n.ancestor at
                       distance r_int from v
     return list
```

Remark 1. We store the ids of the vertices we visit during roll-back in a hashset and we stop the for loop if we meet a vertex which id we stored before reaching distance r_int.

1.3 Figures

We choose a starting point v located at the Arc de Triomphe.

Question 1.1 Identify the number of different possible points you may be located in at the current moment of your trip, when 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.

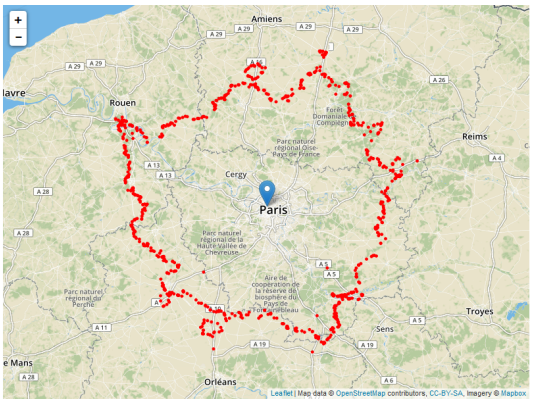

Fig. 1: With t1 = 1 hour, we find 984 different points.

Question 1.2 Identify the number of different possible points you may be located in at the current moment of your trip, when you have been traveling for time exactly t1 = 2 hours from the fixed starting point v towards your destination, which may be any sufficiently distant point in the network.

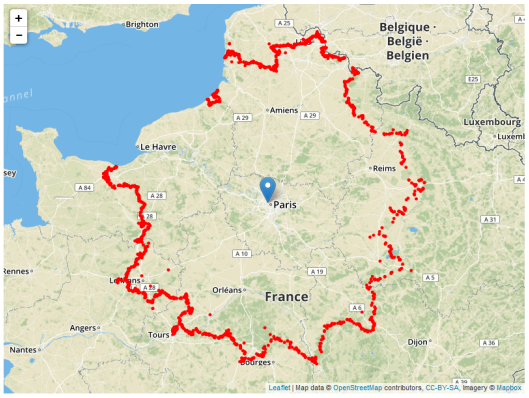

Fig. 2: With t1 = 2 hours, we find 2441 different points.

Question 1.3 Identify the number of different possible points you may be located in at the current moment of your trip, when 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.

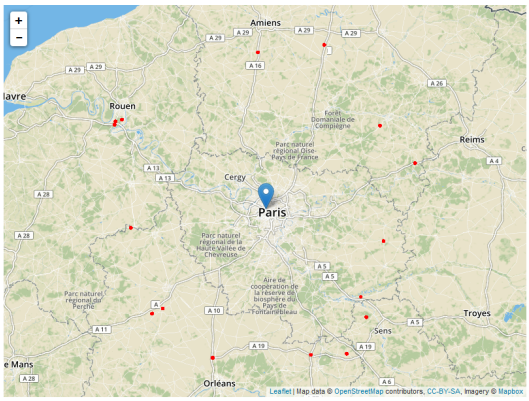

Fig. 3: With t1 = 1 hour and t2 = 2 hours, we find 16<984 different points.

Remark 3. We find fewer points than in question 1: people pass through a limited number of vertices when they go to a farther destination.

2 Option B: More theory and algorithms

2.1 More theory

Definition 1 For a vertex v in a directed graph, its reach r(v) is the largest non-negative real value r such that there exists a pair of vertices s and t, such that the following conditions hold:
-Some fastest path from s to t passes through vertex v.
-The travel time from s to v following the fastest path is at least r.
-The travel time from v to t following the fastest path is at least r.

Question B.1 Let D be the diameter of the directed graph (the longest possible travel time between a pair of vertices, following quickest paths). Then the reach of a vertex in this graph is at most $D/2$.
Proof: If a vertex v has a reach $r > D/2$ then the definition of reach gives s and t such as some fastest path from s to t is at least $2 * r$. So the diameter is (strictly) greater than D: absurd. We conclude tat the reach of a vertex in this graph is at most $D/2$.

Definition 2 Let v be the vertex in the center of Paris chosen in part 1. Let Tout be the set of points to which the travel time from v is exactly 1 hour when traveling to a destination at least 2 hours' way away. Likewise, let Sin be the set of points from which the travel time to v is precisely 1 hour, when following a quickest path from a starting point located at least 2 hours' way from v.

Question B.2 If the reach of v is less than 1 hour, then there cannot exist a point s in Sin and a point t in Tout, such that some quickest path from s to t passes through v.
Proof: Suppose that the reach of v is less than 1 hour, and that there exist a point s in Sin and a point t in Tout, such that some quickest path from s to t passes through v. Then s and t fulfill the conditions of definition 1 with $r = 1$: some fastest path from s to t passes through v and the travel time from s to v or from v to t following the fastest path is at least 1. As the reach of v is defined as the maximum of such values r, we have $1 \leq range(v)$. Thus we have a contradiction with $range(v) < 1$.

Question B.3 If the reach of v is at least 2 hours, then there must exist a point s in Sin and a point t in Tout, such that some quickest path from s to t passes through v.
Proof: If the reach of v is at least 2 hours, then there exist s2 and t2 which fulfill the conditions of definition 1 with r=2. Some fastest path p goes from s2 to t2 and passes through v. Let s and t be the points on the path between s2 and t2 such that the distance between s2 and s is 1 and the distance between v and t is 1. Then s is in Sin and t in Tout. A quickest path from s to t passes through v, else the quickest path from s2 to t2 would not pass through v: if we replace the path from s to t by a quickest path from s to t which does not pass through v in p, we get a path from s2 to t2 which does not pass through v and which is shorter p (which was a quickest one): contradiction.

Question B.4: Algorithm We derive a 2-approximation algorithm to compute the range of a vertex v.

Remark 4. The reach of v is either zero or greater than the minimum length of all edges coming from or going to v. Indeed, with this value of r, the points s in Sin and t Tout are located on the edges coming from or going to v. If Sin or Tout is empty, then $reach(v) = 0$ (we can either only leave v or only go to v). Else the points s and t are closer to v than to the over extremity of their edges. Thus a quickest path from s to t passes through v, so $r \leq range(v)$.

If $range(v)! = 0$ we start with $r < range(v)$ and we repeat the following loop until it returns a result. We compute Sin and Sout with starting point v, intermediate distance $r/2$ and end distance r. If there is a point s in Sin and a point t in Sout such that a quickest path from s to t passes through v then, according to Question B.2, the reach of v is at least $r/2$ and we return r. Else, according to Question B.3, the reach of v is greater than r: we do $r \leftarrow 2r$.
At the end of this algorithm, we return a number r such that
$r/2 \leq reach(v) \leq r$.

2-approximation algorithm for range

```
Input: a graph g, a vertex v in g
Output: a number r such that r(v)/2 <= r <= r(v)
    if v has only incoming or outgoing edges return 0
    else:
     start with r as the minimum of the lengths of the arcs starting in v;
     ## r<=reach(v)

     repeat:
       Tout=Tout(r/2,r) ## the set returned by Question 1.3 with
                           t1=r/2 and t2=r
       Sin=Sin(r/2,r)   ## the set returned by Question 1.3 with
                           t1=r/2 and t2=r and reversing the edges
       for s in Sin:
         if distance from s to a point in Tout is r:
           return r
        r=2*r
```

Optimization To check if there is a point s in Sin and a point t in Tout such that a quickest path from s to t passes through v, we use a Dijkstra algorithm with starting point s in Sin and we stop if we find a point in Tout at distance at least r. To do that, we add to the algorithms finding Sin and Tout a few lines to add the newly found points and edges to the graph, which we remove each time a while loop is finished.

2.2 Question B.6: results

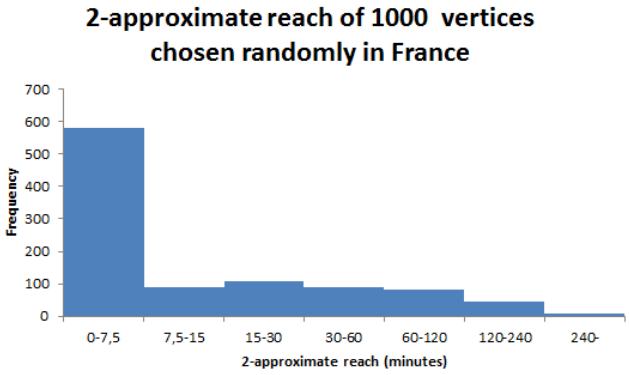

Fig. 4: We find the expected result: if travelers were to adhere to some reasonable navigation scheme (e.g., always choosing fastest routes), there are many roads on the map they should never use, unless they are very close to the starting point of their trip or to their destination. The running time we needed to compute the reach of 1000 vertices in France was 3:10 hours (using a standard PC).