# Efficient algorithms to match GPS data on a map

**Abstract:** Estimating the distribution of travel times on a transportation network from vehicle GPS data requires finding the closest path on the network to a trajectory of GPS points. In this work we develop 1)An efficient algorithm (MOE) to find such a path and able to detect the presence of cycles, and 2)A faster but less accurate heuristic (MMH) unable to detect cycles. We present computational results that compare these algorithms, for different sampling rates and GPS sensitivities, using GPS trajectories of three networks: a grid graph and street networks of Santiago and Seattle. We show that MOE (MMH) returns in seconds (hundredths of second) paths where on average 93% (91%) of the edges are within a corridor of one meter from the real path.

**Keywords:** Map matching, Travel time estimation

## 1 Introduction

### *1.1 Motivation*

Given the abundance of information that is being collected every day on transportation networks around the world, like never before there exists an opportunity to formulate problems to make operational decisions on real representations of transportation networks. This ability, which can lead to improved performance of a-priori solutions or the possibility of developing dynamic solution strategies, requires accurate estimations of relevant data. For example, determining the shortest path on a transportation network requires accurate estimates of the average travel time on road segments which can be estimated from of the vehicular flow. In this aim, accurate and efficient algorithms capable of translating this raw information into network representations of the existing transportations systems are essential. The objective of this article is the development of such efficient and reliable algorithms for the problem of projecting a GPS trajectory onto a graph representing a transportation network. The abundance of GPS trajectories information not only enables online estimation of travel conditions but makes it possible to build reliable representations of the distributions of travel time on different transportation corridors and times of day. To estimate such distributions of travel times, however, would require repeatedly projecting a massive amount of travel information onto a graph representing the transportation network. It becomes therefore important to be able to reliably and efficiently project real travel information onto graphs.

Finding the correct path that generates a GPS trajectory however is not so straightforward. A first difficulty comes from the size of real sized transportation networks and the volume of GPS data to process. In addition to that scale issue, the problem might not be well posed. Specifically, errors in the GPS data, the sampling rate of this data, and inaccuracies in the graph representation of a transportation network can make it difficult to discern which is the road segment that corresponds to certain GPS trajectory. Therefore,

it is crucial for a solution method to be robust against the uncertainties of the information being processed

Developing a tool that accurately estimates travel time distributions on arcs of a transportation network has implications to many logistic and transportation problems that face uncertainty, such as stochastic vehicle routing problems (Gendreau et al. (1996)), combinatorial or network flow problems on a graph (Averbakh and Lebedev (2004); Bertsimas and Sim (2003)) amongst others. More specific applications include shortest path problems with normally or exponential-Gamma distributed travel times (Olya (2014a,b)), Traveling Salesman Problems on real road networks (Jalali Naini et al. (2013)) or general Vehicle Routing Problems (Mousavipour and Hojjati (2014)).

Our work is motivated by an applied project in collaboration with the Santiago Fire Department, to redesign an emergency vehicle dispatching system. A requirement of the new dispatching system is that it take into account congestion at different times of day. This requires estimating expected travel times at different times of day for every arc on the graph representing the transportation network. In this aim, bus GPS data from Transantiago - the public transportation system of Santiago, Chile - was made available to project onto the city's graph representation and used to build estimates of congestion at different times of day.

### *1.2 Literature review*

Previous work on projecting GPS data on a graph have used a wide range of methodologies: some propose algorithms that locally fit the data to a subgraph of the network closest to every measurement point (White et al. (2000)). Other authors preprocess the input data with Kalman filters in order to eliminate inconsistent data (Yang et al. (2005)). In the same spirtit, in Li et al. (2013), the authors present a method that uses all the information coming from several geolocalization sources, increasing the redundancy of the data, which is then filtered applying Kalman filters before a candidate path is obtained by using a weight based algorithm. On another hand, the online algorithms described in Blazquez (2012); Marchal et al. (2005) and Miwa et al. (2012), allow to project the GPS data dynamically by iteratively constructing a path that minimizes locally the projection error with the GPS data. Further, they periodically reoptimize the current partial path according to geometric considerations or by storing a buffer of past states in order to backtrack if the current path deviates too much from the trajectory. In Cortés et al. (2011) the authors simplify dense GPS trajectories with Douglas-Peucker based algorithms and then identify the road segments that match these simplified trajectories locally. In Kasemsuppakorn and Karimi (2013), the authors develop an algorithm building a pedestrian network by processing massive amounts of GPS data and project them onto an unknown grid. The algorithm developed in Bierlaire et al. (2013) looks for a set of candidate paths and then keeps the ones maximizing a likelihood measure. In a similar way, in Hunter et al. (2013) the GPS data is filtered such that only highly probable vehicle positions in the graph are kept. Second, a path finding procedure is applied on these highly probable states that locally optimizes the likelihood of the path returned. The articles by Lou et al. (2009) and Newson and Krumm (2009) propose algorithms that find the most probable road taken by a trajectory assuming that the measurement error of the velocity data follows a normal distribution. In order to find a path (not necessarily simple) in the graph that matches the GPS trajectory, the two latter works construct an acyclic network from the trajectory information and the network topology. The longest path on this related network is the path that maximizes the likelihood of generating that trajectory data.

On one hand, the subgraphs found by the so-called point-to-point or point-to-curve matching methods are not necessarily connected, which limit their use in the case of complex trajectories. On another hand, the majority of the algorithms present in the literature are heuristics that locally optimize common sense criteria instead of returning an optimal solution for such criteria.

### *1.3 Our contribution*

Our approach consists in two algorithms: the first method we introduce modifies the arc traversal cost in proportion with how close the GPS points are, thus making a shortest path between the first and last GPS points on this modified network a good candidate for the path that generated these points. The second algorithm returns a path that globally minimizes a new error measure (the oriented error) that takes into account both the GPS trajectory information and the topology of the network. To do so, we find a shortest path in a related network similar to the networks presented in Lou et al. (2009) and Newson and Krumm (2009).

Further, we make use of some physical upper bound of the maximum measurement error of modern GPS devices to speed up both map matching processes. A driving feature of our approaches is that they lead to computationally efficient methods as we are thinking of applications where huge amounts of data have to be processed and/or for immediate response. We compared the accuracy and computational efficiency of the proposed methods on real road networks (Santiago, Chile and Seattle, WA) and a square grid.

The rest of the paper is structured as follows: In the next section we present the first optimization based approach, which we refer to as Map Matching Heuristic (MMH). Section 3 introduces the second optimization based approach, referred to as Minimization of Oriented Error (MOE). The computational experiments are described and results presented in section 4. We present our conclusions in section 5.

## 2 Map matching heuristic

The central idea of this first approach is to modify the graph weights by lowering the length of edges that are close to the trajectory of points. On this modified graph we then find a shortest path - with respect to these modified lengths - between the first point of the trajectory and the last one. This method is guaranteed to generate a path in the given graph, which is not always true with many of the existing projection methods. In the following we consider that the transportation network is represented by a graph $G = (V, E)$, where the set of nodes has $|V| = n$ elements, and there are $|E| = m$ elements in the set of edges. Let $d_e$ be the non-negative length of edge $e \in E$, with $d \in \mathbb{R}_+^m$ its vector notation, and by $\{p_k\}_{k\in\{1,\dots,q\}}$ the chronologically sorted sequence of the $q$ trajectory points, i.e. the vehicle was recorded at position $p_k$ at time $k$. We denote the non-negative distance of a given trajectory point $p_k$ to node $v \in V$ by $d(p_k, v)$ and to edge $e \in E$ by $d(p_k, e)$.

### *2.1 MMH Outline*

For every point $p_k$ of the trajectory, we find the edges of the graph that are within a radius $R > 0$ of $p_k$ and we lower their lengths by the distance $d(p_k, p_{k+1})$ between $p_k$ and $p_{k+1}$. Once we have applied this procedure on the entire sequence of points, we identify a reasonable

set of ending and starting nodes, according to the trajectory. To do so, we define two sets of candidate nodes $S := \{v \in V : d(p_1, v) \leqslant R\}$ and $T := \{v \in V : d(p_q, v) \leqslant R\}$, that respectively correspond to nodes that are close to the first (last) trajectory point $p_1$ ($p_q$).

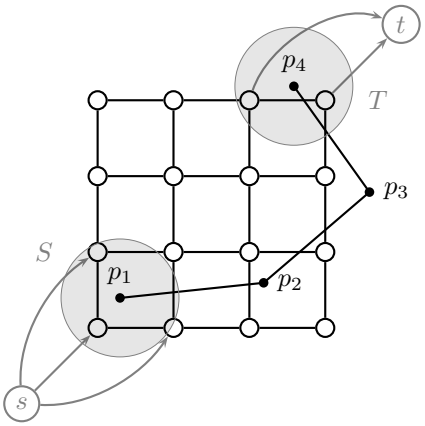


**Figure 1**: Example of S/T zones.

As depicted in figure 1 we define an artificial starting node $s$ connected to every node $v \in S$ with a directed arc of length $d(p_1, v)$ and connect every node $v \in T$ with an artificial end node $t$ with a directed arc of length $d(p_q, v)$. Since the nodes in $S$ and $T$ are within a radius of $R$ of a trajectory point, the edges incident to these nodes $s$ and $t$ have been modified by the first part of this heuristic. This procedure can potentially decrease the length of an arc multiple times so we make sure to keep the positive part of the modified edge lengths in order to keep the arcs' costs nonnegative. We then select a path corresponding to the trajectory of points from this modified network by finding a shortest path between $s$ and $t$. Since all arcs are non-negative we can find the shortest path in a very efficient way. We remark that if a trajectory point is close to an intersection, this procedure reduces the length of all edges on the intersection by an amount equal to the distance to the next trajectory point. This can reduce by a significant amount the length of edges that are not oriented in the direction of the trajectory, thus increasing the probability of selecting a wrong edge in the final path. For this reason it is preferable to reduce the length of the edges with small quantities several times - corresponding to several trajectory points - rather than just one large reduction. This is achieved by adding between every two consecutive trajectory points a number of artificial points in a straight line. We implement this *densification* of the trajectory imposing that the distance between consecutive points has to be at most $d_{\max}$. This ensures that the weight of the arcs not following the trajectory is only reduced by small quantities.

### *2.2 Pseudocode of the MMH heuristic*

In every algorithm developed in this paper, we extensively use shortest paths algorithms as subroutines. The reference method we use here is Dijkstra's algorithm (Dijkstra (1959)). This iterative method gradually improves the shortest distance from some origin node $s$ to any other node of the graph by finding increasingly better $(s, i)$-paths until it reaches some target node $t$ and the current $(s, t)$-path is provably the shortest one. Embedded in the

algorithm appears a subroutine returning at each iteration the node with the current smallest distance from $s$. Depending on the data structure used, the complexity of Dijkstra can vary: in this paper we choose to use a binary heap to store the smallest distance's node at each iteration.

The pseudocode of the MMH heuristic is presented in Algorithm 1; Its output $SP$ is the subset of nodes corresponding to the shortest path found in $\bar{G}$ that we identify as the path generating the trajectory of points $(p_k)_{k\in\{1,\dots,q\}}$. We denote by `dijkstra`$(G, w, u, v)$ the implementation of a Dijkstra algorithm that computes a shortest path between nodes $u$ and $v$ in graph $G = (V, E)$ with respect to weights $w \in \mathbb{R}_+^m$.

**Algorithm 1:** MMH Heuristic Outline

**Data**: $G = (V, E)$, $P = (p_k)_{k\in\{1,\dots,q\}}$, $R$, $d \in \mathbb{R}^m$, $d_{\max}$
. **Result**: A map-matching path $SP$.
$w = d$, $p_t = p_1$;
**for** $k = 1, ..., q-1$ **do**
  $n_k = \lfloor d(p_k, p_{k+1}) / d_{\max} \rfloor$;
  **for** $l = 0, ..., n_k$ **do**
    $p_h = p_k + (p_{k+1} - p_k) \cdot l \cdot d_{\max} / d(p_k, p_{k+1})$;
    **for** $e : d(p_t, e) \leqslant R$ **do**
      $w_e = [w_e - d(p_t, p_q)]_+$;
    $S = \{v \in V : d(p_0, v) \leqslant R\}$;
    $T = \{v \in V : d(p_q, v) \leqslant R\}$;
    $\bar{G} = \{V \cup \{s, t\}, E \cup \{(s, v)_{v\in S}, (v, t)_{v\in T}\}\}$;
    **for** $v \in S$ **do**
      $w_{(s,v)} = d(p_0, v)$;
    **for** $v \in T$ **do**
      $w_{(v,t)} = d(p_q, v)$;
    $SP =$ `dijkstra` $(\bar{G}, w, s, t)$;

### *2.3 Complexity*

There are at most

$$Q = \frac{1}{d_{\max}} \sum_{k=1}^{q-1} d(p_k, p_{k+1})$$

points belonging to the dense data so the complexity of the entire projection framework can be detailed as follows: for each of the $O(Q)$ points of the dense trajectory, find its closest edges has $O(m)$ complexity, giving a total complexity for all the dense points of $O(Qm)$. To find a shortest $(s, t)$-path in the graph with modified weights, we use a binary heap implementation of Dijkstra's algorithm. This implementation terminates in $O(m + n \log n)$ operations. Putting everything together, the total complexity of MMH is $O(Qm + n \log n)$, which is fast in an algorithmic sense, but can be potentially slow in practice because it requires to compute the distance between every point-edge pair.

### *2.4 Improved version*

We can avoid computing each point-edge distance noticing that the only weights needed during the execution of MMH are the ones that are being considered by Dijkstra's algorithm. In consequence, each time that we are looking an outgoing edge $e$ of the current node with the lowest label, we compute its modified weight $w_e$. Although it does not change the theoretical worst-case complexity, doing so allows to considerably speed-up MMH in practice. In order to differentiate paths having zero modified weight, we use a double priority heap during the execution of Dijkstra's algorithm, that -when in presence of 'zero-zero' ties between edges - gives the priority to the one adding the least physical distance.

**Algorithm 2:** MMH on the fly

**Data**: $G = (V, E)$, $P = (\bar{p}_k)_{k\in\{1,...,q\}}$, $R$, $d \in \mathbb{R}^m$, $d_{\max}$.
**Result**: A map-matching path $SP$.

$w = d$;
$T = \{v \in V : d(p_q, v) \leqslant R\}$;
$H = \emptyset$;
father$_v = -1, \forall v \in V$;
$\pi_v = +\infty, \forall v \in V$;
**for** $v \in V$ **do**
  **if** $d(p_0, v) \leqslant R$ **then**
    `insert_heap`$(H, v, d(p_0, v))$;
**while** $H \neq \emptyset$ **do**
  $i =$`get_root`$(H)$;
  `delete_root`$(H)$;
  **if** $i \in T$ **then**
    **return** *i*;
  **for** $j \in \delta^+(i)$ **do**
    **for** $k = 1, ..., Q-1$ **do**
      **if** $d(\bar{p}_k, (i, j)) \leqslant R$ **then**
        $w_{ij} = [w_{ij} - d(p_k, p_{k+1})]_+$;
    **if** $\pi_j > \pi_i + w_{ij}$ **then**
      $\pi_j = \pi_i + w_{ij}$;
      father$_j = i$;
      **if** $j \notin H$ **then**
        `insert_heap`$(H, j, \pi_j)$;
      **else**
        `change_val`$(H, j, \pi_j)$;
**return** $T$ *not reached.*;

We present a pseudocode of this improved version in Algorithm 2, where $(\bar{p}_k)_{k\in\{1,...,Q\}}$ represents the densified trajectory, $H$ is a binary heap, `insert_heap(`$H$`,`$v$`,`$\pi$`)` is a routine that inserts the node index $v$ with value $\pi$ into the heap $H$, `change_val(`$H$`,`$v$`,`$\pi$`)`

is a routine that changes the value of the node index $v$ inside the heap $H$ to the value $\pi$, `get_root(`$H$`)` is a routine that returns the node index of the root node of the heap $H$ and `delete_root(`$H$`)` is a routine that deletes the root node of heap $H$.

### *2.5 Known problems*

In some situations MMH can identify a path that does not correspond to the original itinerary. For example, MMH can partially fail to identify the correct edges when the trajectory contains a tight U-turn part. Due to the proximity between the forward and the backward pieces of the trajectory, the edges connecting them in the entire graph will have their lengths at very low values. As it can be observed in figure 2, those connecting edges have lower values than the real path edges MMH can cut off an entire U-turn part (dotted) by passing directly from the outgoing segment to the incoming segment (dashed).

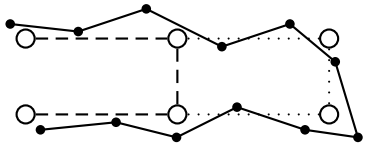

**Figure 2**: Example of tight U-turn type:
GPS trajectory (solid), real path (dotted), output (dashed).

## 3 An exact algorithm

We now present a way to avoid this kind of error by finding a path that minimizes a new type of error measure.

### *3.1 Oriented error measure*

Given a trajectory, one of the simplest methods to project this information on a map is to associate each point to its closest edge in the network. A drawback of this method is that it freely chooses an edge for each GPS point. For example, in Figure 3 we can see that point $p_5$ is matched with the lower edge when it should be associated with the upper one.

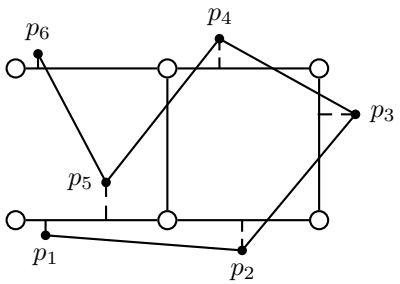


**Figure 3**: Example of classic closest point-edge matching (dashed).

This kind of wrong association is due to the fact that we allow each point to be projected on any edge without any consideration for the network's topology. To avoid making these errors, we define the *oriented error* $OE(C)$ of some path $C$ as follows: The oriented error is a measure that sequentially computes the closest edge of each trajectory point as before, with the difference that we cannot project a point on an edge if it is not the edge associated to the previous point or one of its outgoing edges. Formally, given some path $C = \left(c_1, ..., c_{|C|}\right) \subseteq E$, its ordered error is defined as

$$OE(C) := \frac{1}{q} \sum_{k=1}^{q} d\left(p_k, c_{e(k)}\right),$$

with $e(k) := \arg\min \left\{d\left(p_k, c_i\right) : i \geqslant e(k-1)\right\}$ and $e(0) := 1$. In other words, it represents the least distance between a point and the edges of the path $C$ following the edge associated with the previous point. Notice that this error measure significantly penalizes the paths that cut U-turns. In the next section we present an algorithmically fast way to find a path that minimizes the oriented error measure.

### *3.2 Shortest path reduction*

In order to find a path that minimizes the oriented error, we have to find a point-edge association minimizing the classic error such that every point is projected on an outgoing edge of the previous point's associated edge. We show that it can be done in polynomial time using a shortest path algorithm in a related network. To define this network, we use the notation $d_p^e = d(e, p)$ to represent the distance between some edge $e \in E$ and some trajectory point $p$. Let $\delta^-(j)$ represent the set of outgoing arcs from node $j \in V$. Let define the directed graph $\bar{G} = (\bar{V}, \bar{E})$ such that $\bar{V} = \{s\} \cup V_1 \cup V_2 \cup ... \cup V_q \cup \{t\}$ where for each point $p_r$ of the trajectory the node set $V_r$ is defined as $V_r := \{v_r^e, \forall e \in E\}$. The edge set $\bar{E}$ is defined as the union $\bar{E} = E_1 \cup E_2 \cup ... \cup E_q \cup E_{q+1}$ where the subsets $E_q$ are defined as follows:

$$\begin{cases} E_1 = \{(s, v_1^e), \forall e \in E\} \\ E_r = \bigcup\limits_{(i,j) \in E} \left\{ (v_{r-1}^{(i,j)}, v_r^{(i,j)}), \left( (v_{r-1}^{(i,j)}, v_r^{(j,k)}) \right)_{k \in \delta^-(j)} \right\}, \forall r \in \{2, ..., q\} \\ E_{q+1} = \left\{(v_q^e, t), \forall e \in E\right\} \end{cases}$$

with the following edge weights:

$$\begin{cases} w_{(s, v_1^e)} = d_{p_1}^e, \forall e \in E \\ w_{(v_{r-1}^{(i,j)}, v_r^{(i,j)})} = d_{p_r}^{(i,j)}, \forall r \in \{2, ..., q\}, \forall (i,j) \in E \\ w_{(v_{r-1}^{(i,j)}, v_r^{(j,k)})} = d_{p_r}^{(j,k)}, \forall r \in \{2, ..., q\}, \forall (i,j) \in E, \forall k \in \delta^-(j) \\ w_{(v_q^e, t)} = 0, \forall e \in E \end{cases}$$

An example of such a graph $\bar{G}$ coming from the instance of Figure 4 is depicted in Figure 5.

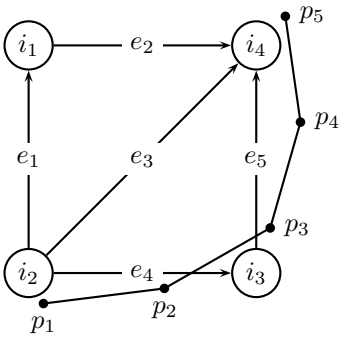


**Figure 4**: Example of graph $G = (V, E)$ with some GPS trajectory.

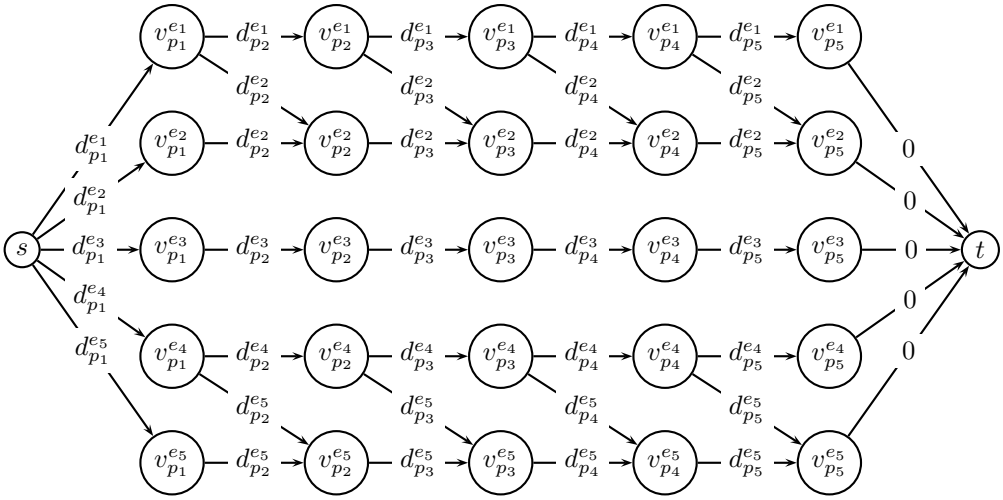


**Figure 5**: Derivated acyclic graph $\bar{G}$ associated to $G$ and the GPS trajectory.

We can easily prove that a path that minimizes the total ordered distance can be obtained from any shortest $(s,t)$-path $C = (c_k)_{k\in\{1,\dots,q\}}$ in the graph $\bar{G}$. In effect, 1)any shortest $(s,t)$-path $C$ in $\bar{G}$ defines a unique path in $G$ and by construction, it has the smallest classic error and projects and 2)it projects only on the previously selected edge or one of its outgoing edges. Observe that MOE allows to backtrack and form cycles, which is a good feature considering that the trajectory can loop at some point. On another hand, it can lead to situations where several consecutive GPS points are successively projected on an edge $(i,j)$ and its counterpart $(j,i)$. This problem was tackled using a double priority shortest path algorithm optimizing first in terms of the oriented error, and in case of tie between edges, take the one adding the least real distance to the actual path. In practical terms, when running the shortest path algorithm, we use a double priority criterion instead of the classic one: we first optimize with respect to the lengths of the network $\bar{G}$, and then we force it to choose the path with minimum real path length.

### *3.3 Pseudocode of MOE*

**Algorithm 3:** MOE algorithm Outline

**Data**: $G = (V, E)$, $P = (p_k)_{k \in \{1,...,q\}}$, $d \in \mathbb{R}^m$, $d_{\max}$.
**Result**: A map-matching path $SP$.

Build the graph $\bar{G} = \left(\bar{V}, \bar{E}\right)$ and calculate the edge weights $w$ as described in subsection 3.2;
$\overline{SP}$ =`double_priority_shortest_path`$\left(\bar{G}, w, d, s, t\right)$;
Let $\overline{SP} := \left\{\bar{e}_1, ..., \bar{e}_{\left|\overline{SP}\right|}\right\}$;
$SP = \emptyset$;
**for** $a = 1, ..., \left|\overline{SP}\right|$ **do**
    Catch the edge $e \in E$ associated to the edge-point edge $\bar{e}_a \in \bar{E}$;
    **if** $SP = \emptyset$ **then**
        $e_1 = e$;
        $SP = \{e_1\}$;
    **if** $e \neq e_{|SP|}$ **then**
        $e_{|SP|+1} = e$;
        $SP = SP \cup \left\{e_{|SP|+1}\right\}$;
**return** $SP$;

where `double_priority_shortest_path`$\left(\bar{G}, w, d, s, t\right)$ is a routine that returns a path between nodes $s$ and $t$ in graph $\bar{G}$ that is shortest with respect to the weights $w$, and amongst the shortest paths with respect to $w$, is also the shortest with respect to the distances $d$.

### *3.4 Theoretical complexity*

Noticing that $\bar{G}$ is acyclic, we can find a $(s, t)$-shortest path in $O\left(\left|\bar{V}\right| + \left|\bar{E}\right|\right)$ time with a search algorithm. Let compute the total complexity in function of the original parameters. First, for any $r \in \{1, ..., q\}$ the number of nodes of the subset $V_r$ is $|V_r| = m$ implying $\left|\bar{V}\right| = 2 + qm$. Second, the number of edges of each subset $E_q$ is $|E_1| = |E_{q+1}| = m$ for the first and the $(q + 1)$-th one, and for any $r \in \{2, ..., q\}$ we have

$$|E_r| = \sum_{(i,j) \in E} \left[1 + \left|\delta^-(j)\right|\right] = m + \sum_{i \in V} \left|\delta^+(i)\right| \cdot \left|\delta^-(i)\right|.$$

Putting everything together the total number of edges in $\bar{E}$ is $O\left(qm^2\right)$ implying a worst case complexity for finding a shortest $(s, t)$-path over graph $\bar{G}$ in $O\left(qm^2\right)$.

### *3.5 An improved optimal algorithm*

The computationally heavy part of MOE is the construction of the network $\bar{G}$. If there exists an upper bound $\delta$ over the maximum error between the real path and its trajectory, we can *a priori* omit the point-edge associations having an error greater than $\delta$. This implies

that many edges are not included in the network $\bar{G}$, considering only the edges of $E$ in a corridor of width $\delta$ around the trajectory. Further, we are able to speed up MOE using the same trick that improved MMH by only computing the weights of the edges we are encountering during the algorithm execution. In consequence, we can modify the shortest path algorithm in the same way and compute dynamically the distances between trajectory points and edges.

## 4 Experimental results

In this section, we test the sensitivity of both algorithms - in terms of accuracy and execution time - with respect to 1)the sampling rate of the GPS sequence of points and 2)the magnitude of the physical GPS induced error. We present results on generated data and also on real GPS records.

### *4.1 Data sets*

We test MOE and MMH on two real networks: Santiago de Chile and Seattle, WA and a ficticial grid network of $100 \times 100$ nodes with edges of length $100$ meters. An overview of their properties is presented in Table 1 and snapshots of the two real networks are presented in Figures 11 and 12 of the Annex.

| Network | $n$ | $m$ | avg node degree | avg edge length $[m]$ |
|---|---|---|---|---|
| Santiago | 325261 | 662579 | 2.04 | 42.7 |
| Seattle | 423240 | 857406 | 2.03 | 50.9 |
| Grid | 100 | 454 | 4.54 | 129.3 |

**Table 1** Networks overview.

For each network we generated 100 different paths between nodes that were between 5 and 10 kilometers apart thus creating paths of at least 5-10 kilometers. Second, we sampled a trajectory of points following that path, such that the sampling rate between points was in $s \in \{10, 30, 50, 70, 90, 110, 130, 150, 170, 190, 210, 250, 300, 400, 500\}[m]$. These generated points were perturbed using a Gaussian noise $e \rightsquigarrow \mathcal{N}(\mu = 0, \sigma)$ with $\sigma \in \{1, 2, 5, 7, 10, 20, 50\}$[m] in both coordinates, without correlation with each other. In order to analyze the influence of the sampling rate $s$ and the standard deviation $\sigma$ on the execution time and the accuracy of the output, we define a base case $(s, \sigma) = (150[m], 20[m])$ and let vary each parameter independently. This base case is similar to real GPS trajectories. Finally, two consecutive points of the densified trajectory must be closer than $d_{\max} = 20[m]$ and we considered $\delta = R = 500[m]$.

Around 900 real GPS trajectories were available for the particular case of Santiago de Chile: we tested both MOE and MMH on these instances without knowing what their real paths were. In consequence, we were not able to to compute the real difference between the original path and the output of the algorithms, only the oriented and classic errors.

### *4.2 Results*

Let define $R = \left(r_1, ..., r_{|R|}\right) \subseteq E$ the real path of the trajectory. In order to check if the algorithm is efficiently identifying the correct subgraph of each trajectory, we defined an

error measure comparing the real path $R$ and the output path $C$ of some algorithm. Let define the distance $d(e,a)$ between two edges $e=(i,j)\in E$ and $a=(u,v)\in E$ as the minimum distance between two points of the segments defined by the two edges. Now we can define the notion of average real error of some output path $C$:

$$RE(C) := \frac{1}{|C|}\sum_{i=1}^{|C|} \min_{j\in\{1,\ldots,|R|\}} d\left(c_i, r_j\right),$$

that represents the average distance between an edge of the path found and the real path the trajectory comes from. The classic error measure, defined as the average of the point-to-closest-edge distances can be written:

$$CE(C) := \frac{1}{q}\sum_{k=1}^{q} \min_{i\in\{1,\ldots,|C|\}} d\left(p_k, c_i\right).$$

The algorithms presented in this paper were coded in C programming language and run over a cluster node of 2.4GHz with 6Gb Ram.

### *General results*

| | | $T[s]$ | | | $RE(C)[m]$ | | | $OE(C)[m]$ | | | $CE(C)[m]$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Net. | Alg. | avg | stdev | max | avg | stdev | max | avg | stdev | max | avg | stdev | max |
| Santiago | MOE | 5.056 | 3.180 | 17.202 | 6.2 | 3.5 | 19.0 | 14.1 | 10.1 | 113.1 | 12.9 | 1.1 | 17.6 |
| | MMH | 0.049 | 0.055 | 0.575 | 9.8 | 13.6 | 137.3 | 16.6 | 16.6 | 179.4 | 16.3 | 16.6 | 179.4 |
| Seattle | MOE | 3.187 | 1.537 | 7.079 | 5.8 | 3.5 | 17.7 | 11.8 | 1.5 | 23.5 | 11.7 | 0.9 | 14.6 |
| | MMH | 0.044 | 0.029 | 0.241 | 9.8 | 13.0 | 123.2 | 15.0 | 14.5 | 153.4 | 14.7 | 14.1 | 150.3 |
| Grid | MOE | 6.901 | 2.767 | 13.866 | 3.7 | 1.3 | 6.9 | 12.9 | 0.9 | 15.1 | 12.9 | 0.8 | 15.1 |
| | MMH | 0.015 | 0.005 | 0.031 | 10.8 | 3.2 | 18.0 | 17.3 | 1.9 | 23.3 | 17.2 | 1.9 | 23.3 |
| Overall | MOE | 5.073 | 3.002 | 17.202 | 5.2 | 3.1 | 19.0 | 13.0 | 6.0 | 113.1 | 12.5 | 1.1 | 17.6 |
| | MMH | 0.036 | 0.039 | 0.575 | 10.1 | 11.0 | 137.3 | 16.3 | 12.7 | 179.4 | 16.1 | 12.6 | 179.4 |

**Table 2** Average performance of algorithms on different network types for the base case $(s,\sigma) = (150[m], 20[m])$.

The results presented in Table 2 are the average, standard deviation and maximum from the 100 instances generated for each network in the base case (i.e.: 300 for the overall line). As expected, all the types of error measures are lower for the paths found by MOE than those computed by MMH. In particular, this result holds for the real error measure, where the map matching returned by MOE is almost two times more accurate than the map matching returned by MMH. On another hand MMH is two orders of magnitude faster than MOE, running in hundredths of second. In figure 6 we see that in the base case, any algorithm on any network (Santiago, Seattle, Grid) returns (on average) map matching paths where at least 91% of their edges are within a corridor of one meter from the real path. Once again, we can see that MOE behaves better than MMH with at least 93% on average of their path edges in a corridor of one meter around the real path. Further, on the artificial grid instances, we can see that there is a peak of the cumulative distributions at 100 meters, which corresponds to the length of all the edges. Another peak is observed at 150 meters, which corresponds to point-edge associations wrongly associated with a neighboring edge or an edge which is at an opposite corner of the square the edge belongs to. The same phenomenon occurs as well at 200 meters.

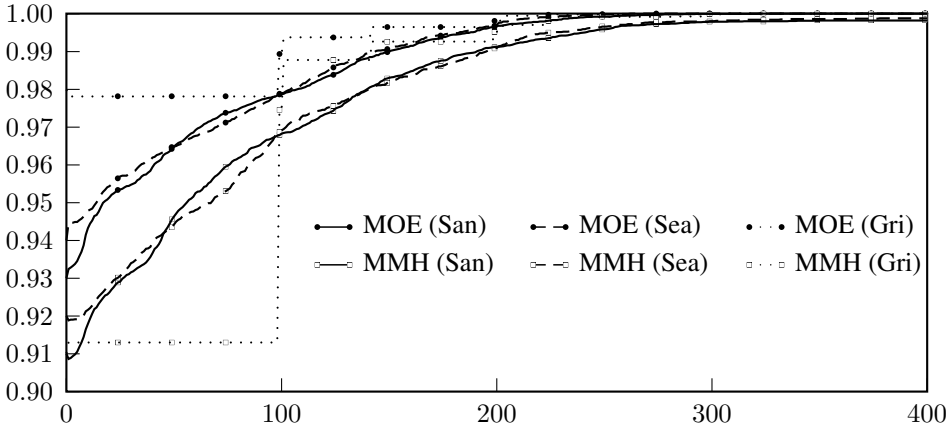


**Figure 6**: Proportion of the edges of the output path contained in a corridor of $x$ meters of the real path for the base case $(s, \sigma) = (150[m], 20[m])$.

*Accuracy sensitivity*

Figure 7 shows that the real error of both algorithms depends of the sampling rate in an almost linear way ($RE(C) \approx 0.07 \cdot s$), MOE behaving better than MMH once again. Figure

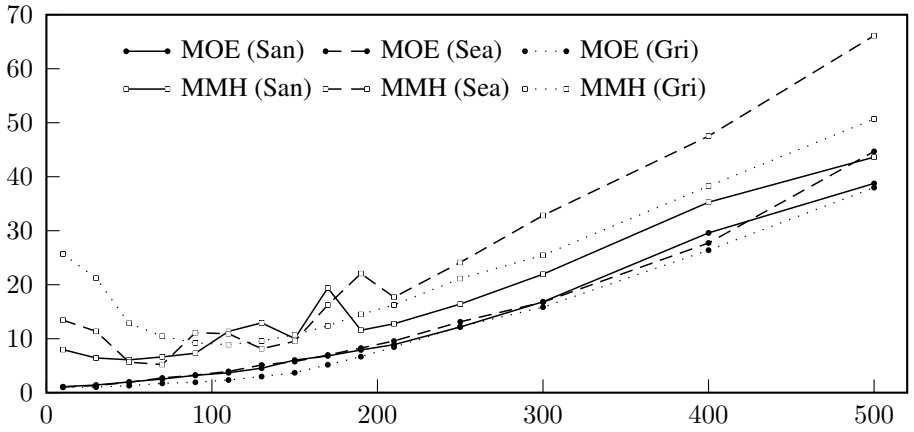


**Figure 7**: Average real error $RE(C)[m]$ as function of the GPS sampling step $[m]$.

8 shows that the standard deviation of the Gaussian noise we applied to the trajectory has close to no influence up to $\sigma = 20[m]$, which is in the range of modern GPS sensitivity. For errors larger than $20[m]$, we observe that MMH is considerably less robust than MOE.

*Algorithmic sensitivity*

In figure 9 we can see that the sampling step has a weak influence on the execution time of both algorithms except for $s = 10[m]$ where the solution time for both algorithms is significantly higher. This is due to the densification procedure that always generates

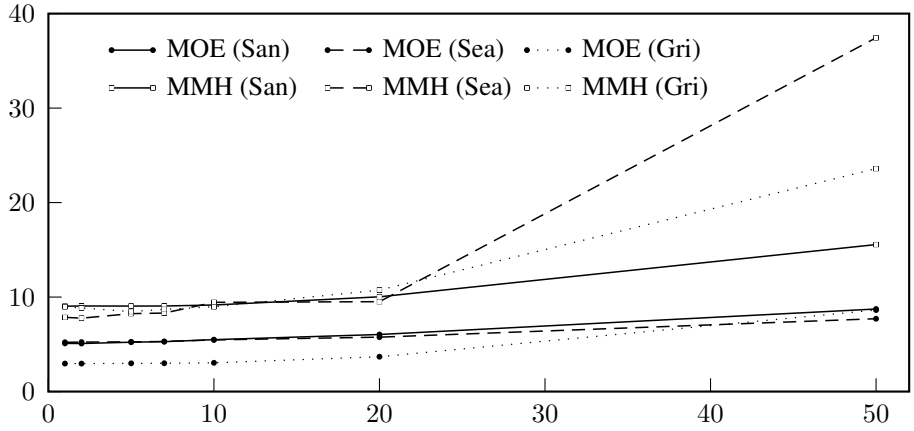


**Figure 8**: Average real error $RE(C)[m]$
in function of the GPS noise standard deviation $\sigma[m]$.

instances where the artificial sampling step is at most $d_{\max}$. Nevertheless, its weak influence can be explained by a deeper - then slower - search of the shortest path algorithm, due to the presence of similarly good candidates for the edges to project the trajectory on. In

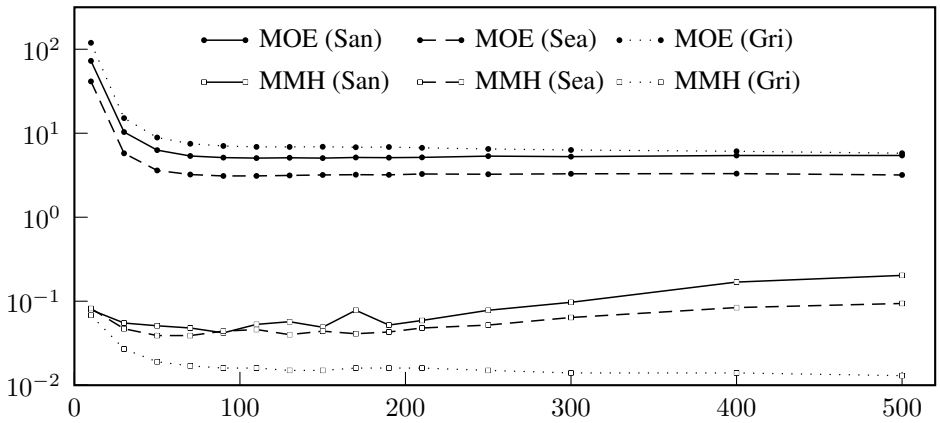


**Figure 9**: Average execution time $[s]$
in function of the GPS sampling step $[m]$ (Log scale).

Figure 10 we observe that the standard deviation $\sigma$ of the Gaussian noise has a significant influence on the execution time of both algorithms. In effect, for MOE, a high dispersion of the trajectory increases the size of the network $\bar{G}$ and then, also the possibilities of path to be explored. For MMH, a high dispersion implies that a higher number of edges will be decent candidates, hence increasing the number of ties during the shortest path search.

*Real GPS data*

The general statistics over the real trajectories coming from Santiago's public transportation system (Transantiago) are summarized in Table 3. Again, we can see that MOE behaves

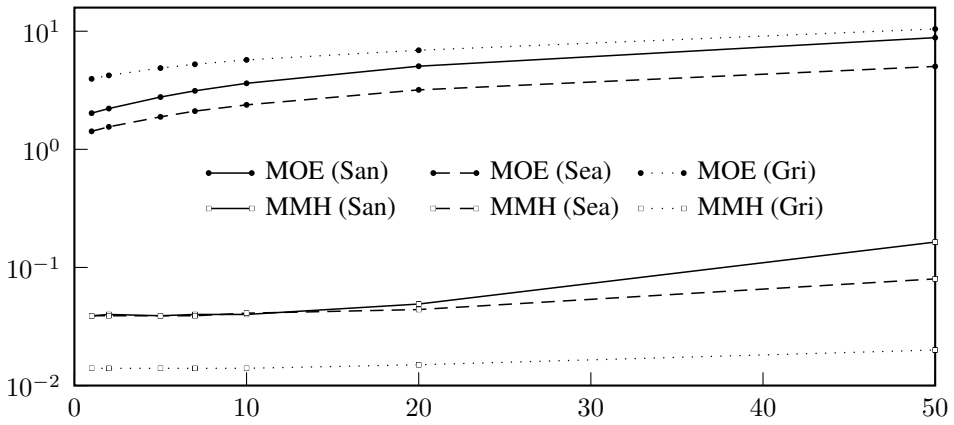


**Figure 10**: Average execution time $[s]$
in function of the GPS noise standard deviation $\sigma[m]$ (Log scale).

| | $T[s]$ | | | $OE(C)[m]$ | | | $CE(C)[m]$ | | |
|---|---|---|---|---|---|---|---|---|---|
| Algorithm | avg | stdev | max | avg | stdev | max | avg | stdev | max |
| MOE | 4.459 | 5.712 | 35.567 | 14.3 | 44.0 | 370.7 | 3.0 | 2.7 | 28.9 |
| MMH | 0.122 | 0.157 | 1.415 | 33.9 | 71.9 | 537.1 | 29.4 | 59.3 | 366.7 |

**Table 3** Algorithms' performances on Transantiago data.

better than MMH in terms of distances between the output path and the trajectory: on average, a trajectory point is ten times closer to the output path returned by MOE than the path returned by MMH. On another hand, we can confirm that on average MMH is again two orders of magnitude faster than MOE. Although MMH is less accurate than MOE, the speed up it provides makes it the right candidate to project in real time massive amounts of GPS data on large scale networks. Indeed, when projecting thousands of GPS trajectories at once it can be prohibitive to spend several seconds per vehicle to guess the path the user is taking from its GPS coordinates.

## 5 Conclusions

In this paper, we define an oriented measure of error that takes into account the ordering of the path when computing its discrepancy level with the trajectory. We develop an algorithm that finds a path minimizing the oriented error measure and add some algorithmic steps reducing its execution time when information about the measurement error of the GPS signal is known. In the case of dynamic map matching, projecting large amounts of GPS trajectories, or very large instances, faster algorithms may be necessary. We explore the tradeoff between computational efficiency and accuracy for this problem by introducing a heuristic algorithm. The proposed heuristic runs two orders of magnitude faster on average, at a reasonable cost of the fidelity of the path it returns, but lacks the ability of detecting cycles or backtracking. In particular, when the sampling step is around 100 meters with realistic GPS errors (less than 20 meters), the accuracy of the MMH heuristic is comparable to (less than twice) the accuracy of the MOE exact algorithm.

The work in this paper constitutes a confident way to develop accurate estimates of travel time distributions on a large portion of the different arcs representing a road network. Consequently, it provides data that can be used to generate a risk averse dispatch of emergency vehicles under different congestion conditions. Being able to construct a distribution of travel times on each arc becomes a first step to accurately evaluate different risk measures. In a future work, it could be interesting to also make use of other kinds of information to project GPS data more accurately: for example, if we know that several trajectories actually correspond to a same path of the network, working on the *average trajectory* could help to give a more accurate and faster map-matching algorithm.

## References

Averbakh, I. and V. Lebedev (2004). Interval data minmax regret network optimization problems. *Discrete applied mathematics 138*, 289–301.

Bertsimas, D. and M. Sim (2003). Robust discrete optimization and network flows. *Mathematical Programming 98*(1), 49–71.

Bierlaire, M., J. Chen, and J. Newman (2013). A probabilistic map matching method for smartphone GPS data. *Transportation Research Part C: Emerging Technologies 26*, 78–98.

Blazquez, C. (2012). *A Decision-Rule Topological Map-Matching Algorithm with Multiple Spatial Data*. INTECH Open Access Publisher.

Cortés, C., J. Gibson, A. Gschwender, M. Munizaga, and M. Zúñiga (2011). Commercial bus speed diagnosis based on GPS-monitored data. *Transportation Research Part C: Emerging Technologies 19*(4), 695–707.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische mathematik 1*(1), 269–271.

Gendreau, M., G. Laporte, and R. Seguin (1996). Stochastic vehicle routing. *European Journal of Operational Research 88*(1), 3–12.

Hunter, T., P. Abbeel, and A. M. Bayen (2013). The path inference filter: model-based low-latency map matching of probe vehicle data. In *Algorithmic Foundations of Robotics X*, pp. 591–607. Springer.

Jalali Naini, S. G., M. M. Paydar, J. Jouzdani, and M. Fathian (2013). Fuzzy stochastic linear programming-based approach for multiple departures single destination multiple travelling salesman problem. *International Journal of Operational Research 17*(4), 417–435.

Kasemsuppakorn, P. and H. A. Karimi (2013). A pedestrian network construction algorithm based on multiple GPS traces. *Transportation research part C: emerging technologies 26*, 285–300.

Li, L., M. Quddus, and L. Zhao (2013). High accuracy tightly-coupled integrity monitoring algorithm for map-matching. *Transportation Research Part C: Emerging Technologies 36*, 13–26.

Lou, Y., C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang (2009). *Map-matching for low-sampling-rate GPS trajectories*. ACM.

Marchal, F., J. Hackney, and K. W. Axhausen (2005). Efficient map matching of large global positioning system data sets: Tests on speed-monitoring experiment in zürich. *Transportation Research Record: Journal of the Transportation Research Board 1935*, 93–100.

Miwa, T., D. Kiuchi, T. Yamamoto, and T. Morikawa (2012). Development of map matching algorithm for low frequency probe data. *Transportation Research Part C: Emerging Technologies 22*, 132–145.

Mousavipour, S. and S. M. H. Hojjati (2014). A particle swarm optimisation for time-dependent vehicle routing problem with an efficient travel time function. *International Journal of Operational Research 20*(1), 109–120.

Newson, P. and J. Krumm (2009). *Hidden Markov map matching through noise and sparseness*. ACM.

Olya, M. H. (2014a). Applying Dijkstra's algorithm for general shortest path problem with normal probability distribution arc length. *International Journal of Operational Research 21*(2), 143–154.

Olya, M. H. (2014b). Finding shortest path in a combined exponential–gamma probability distribution arc length. *International Journal of Operational Research 21*(1), 25–37.

White, C., D. Bernstein, and A. Kornhauser (2000). Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies 8*(1), 91–108.

Yang, J., S. Kang, and K. Chon (2005). The map matching algorithm of GPS data with relatively long polling time intervals. *Journal of the Eastern Asia Society for Transportation Studies 6*, 2561–2573.

## 6 Annex

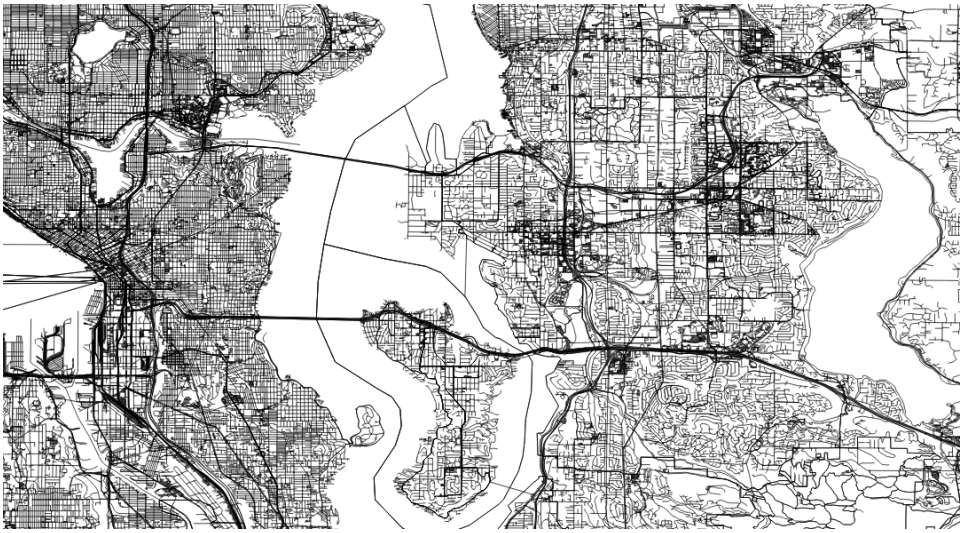

**Figure 11**: GIS snapshot of Seattle's network.

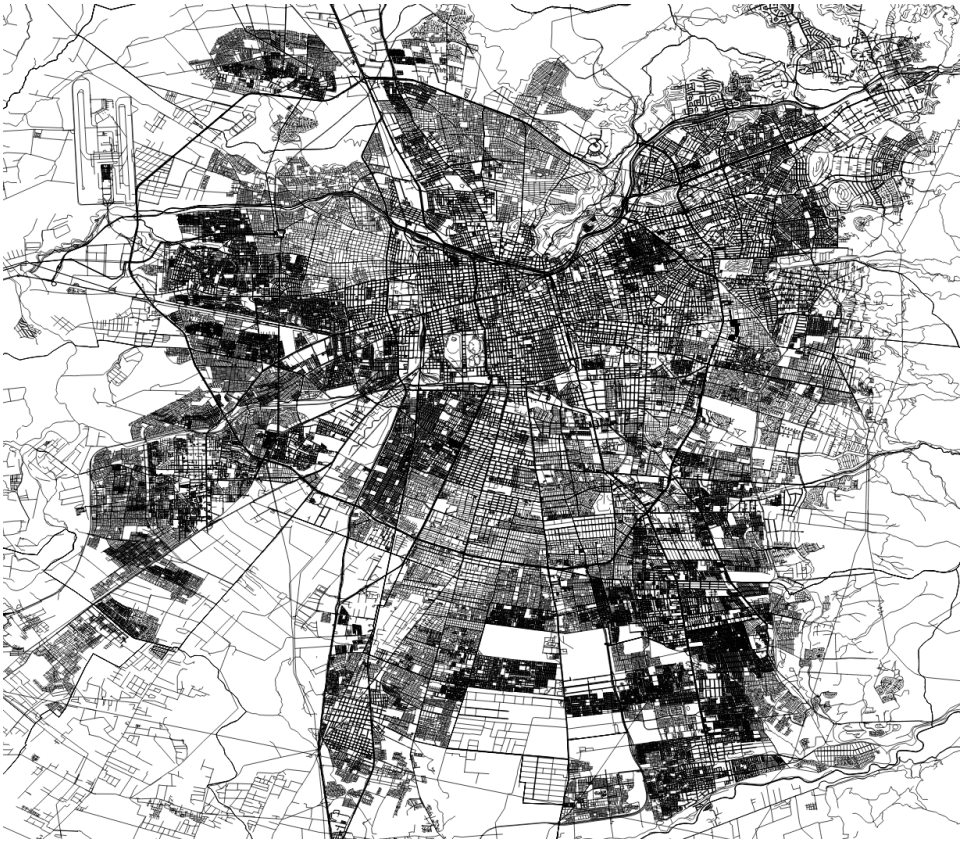

**Figure 12**: GIS snapshot of Santiago's network.