New geometric approaches to the map-matching problem

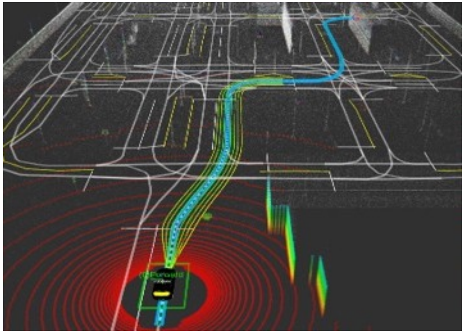
T. Akamatsu, G. Gress, K. Huneycutt, S. Omura
Academic Mentor: Dr. Kano
Industry Mentor: Dr. Yamazaki

August 8, 2022
g-RIPS

Map-matching

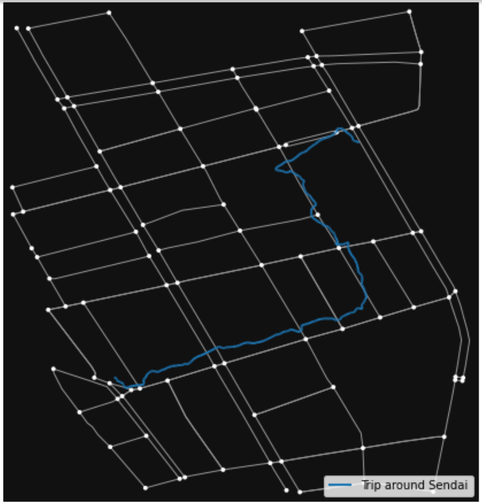
Given GPS trajectory data and a road map, **map-matching** is the process of determining the route on the map that corresponds to the trajectory data.

Web mapping services



Autonomous Vehicles [H]

Problem Statement



Problem Statement

Let us fix $N \in \mathbb{N}$, $N \geq 2$, but almost everywhere we consider the case $N = 2$.

Definition (Trajectory)

A **trajectory** Tr is a sequence $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ of points in $\mathbb{R}^N$ equipped with

- a sequence $t(\mathbf{p}) = (t_1, \ldots, t_n)$ of positive numbers satisfying $t_1 < t_2 < \cdots < t_n$, called the **timestamp** of $\mathbf{p}$,
- a sequence $\mathrm{spd}(\mathbf{p}) = (\mathrm{spd}_1, \ldots, \mathrm{spd}_n)$ of positive numbers called the **speed** of $\mathbf{p}$ (optional),
- a sequence $u(\mathbf{p}) = (u_1, \ldots, u_n)$ of unit vectors in $\mathbb{R}^N$, called the **direction** of $\mathbf{p}$ (optional).

Problem Statement

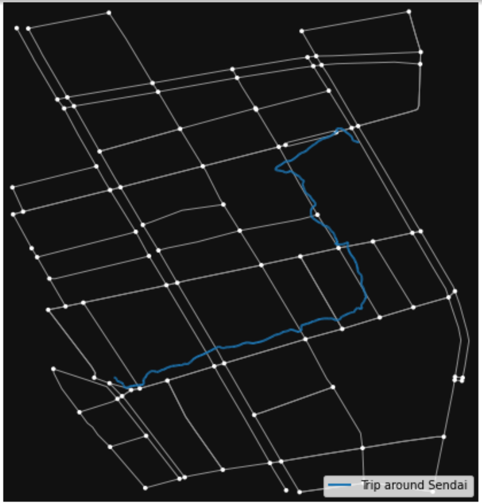
Definition (Road Network)

A **road network** (also known as a map) is a directed graph $G = (V, E)$ consists of the set V (resp. E) of vertices (resp. edges) with an embedding $\phi : |G| \to \mathbb{R}^N$ of the geometric realization $|G|$ of G. We will identify G and the image $\phi(|G|)$ by ϕ as long as there is no confusion.

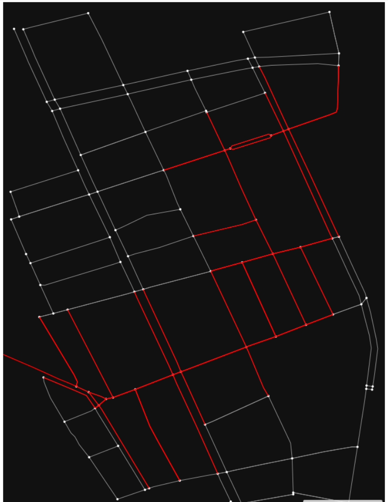
Definition (Local Road Network)

A **local road network** is a directed connected subgraph of $G = (V, E)$.

Road Network



Local Road Network



Problem Statement

Definition (Route)

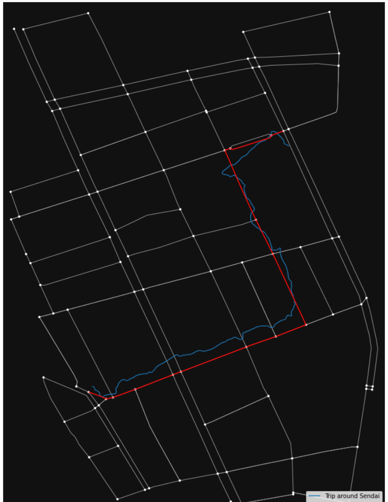
A **route** r on a road network $G = (V, E)$ is a sequence of connected edges $(e_1, e_2, \dots, e_n) \subset E$, i.e. the head of e_i coincides with the tail of e_{i+1} for each $i = 1, 2, \dots, n-1$. Let R denote the set of all routes.

Definition (Candidate Routes)

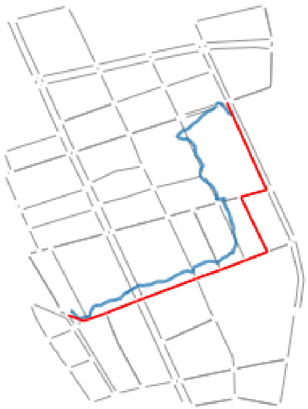
For the local road network graph as H of the road network G, we define

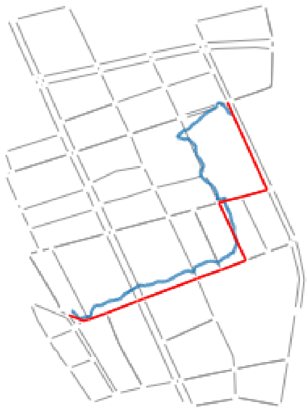
$$\mathcal{CR}_H = \mathcal{CR} := \{\text{routes on a local road network graph } H\},$$

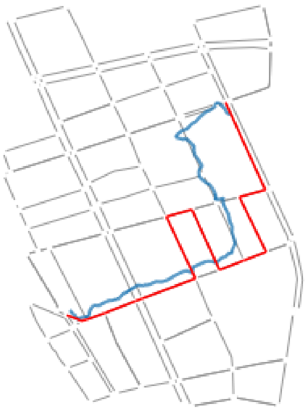
Route



Candidate Routes





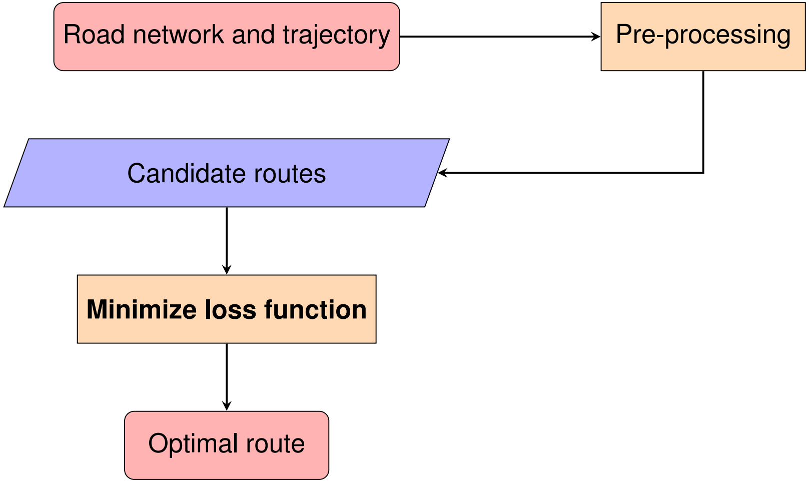


Problem Statement

Definition (Map-Matching)

Given a road network $G = (V, E)$ and a trajectory Tr, the map-matching, $\mathcal{MR}_G(Tr)$, is the route that is the argument of the minimum of some function $L : \mathcal{CR} \to \mathbb{R}^+$, called the **loss function.**

Map-matching Pipeline



Mathematical Formulation

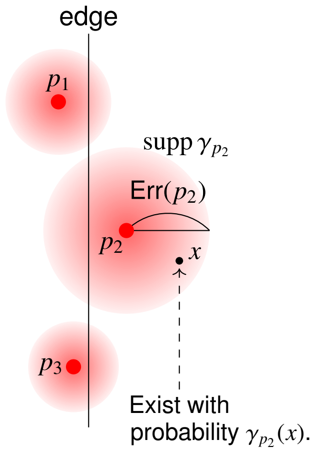
Assumption

- Give the **GPS error** as Err : $\mathbf{p} \to \mathbb{R}_{\geq 0}$ and assume that the *spherically-symmetric probability measure* γ_p (e.g. *Gaussian measure*) is given such that

$$\operatorname{supp}\gamma_p = B(p; \mathrm{Err}(p)) := \left\{ x \in \mathbb{R}^N \;\middle|\; d_{\mathbb{R}^N}(x, p) \leq \mathrm{Err}(p) \right\}.$$

  We assume that $p \in \mathbf{p}$ is truly located at x with probability $\gamma_p(x)$.

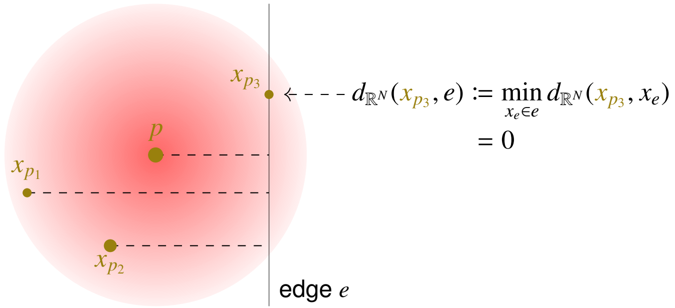
- Suppose that **there is NO error with respect to the speed and direction information.**


edge
p_1
$\operatorname{supp}\gamma_{p_2}$
Err(p_2)
p_2
x
p_3
Exist with probability $\gamma_{p_2}(x)$.

Definition (The "distance" with error between $p \in \mathbf{p}$ and $e \in E$)

- Define the **"distance" with error** $\mathrm{d}_{\mathrm{Err}}$ between $p \in \mathbf{p}$ (*with errors*) and $e \in E$ (*without errors*) as

$$\mathrm{d}_{\mathrm{Err}}(p, e) := \int_{x_p \in B(p;\mathrm{Err}(p))} d_{\mathbb{R}^N}(x_p, e)\, \mathrm{d}\gamma_p(x_p).$$


x_{p_3}
p
x_{p_1}
x_{p_2}
edge e
$d_{\mathbb{R}^N}(x_{p_3}, e) := \min_{x_e \in e} d_{\mathbb{R}^N}(x_{p_3}, x_e)$
$= 0$

Wasserstein method

Definition ((L^1-)Wasserstein distance (*review of mid-term presentation*))

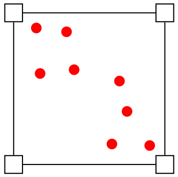
Let (X, d) be a complete and separable metric space. For probability measures μ, ν with finite supports, we define W_1 *distance* between μ and ν as

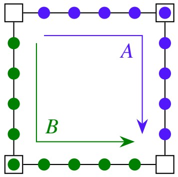
$$W_1(\mu, \nu) := \min_{\pi \in \Pi(\mu,\nu)} \sum_{x \in X} \sum_{y \in X} d(x, y)\pi(x, y),$$

where $\pi \in \Pi(\mu, \nu) \;:\Leftrightarrow\;$ for any $x, y \in X$, $\sum_{y \in X} \pi(x, y) = \mu(x)$, $\sum_{x \in X} \pi(x, y) = \nu(y)$.

Definition (Prob. meas. associated w/ $\mathbf{p}$ and $A \in \mathcal{CR}$)

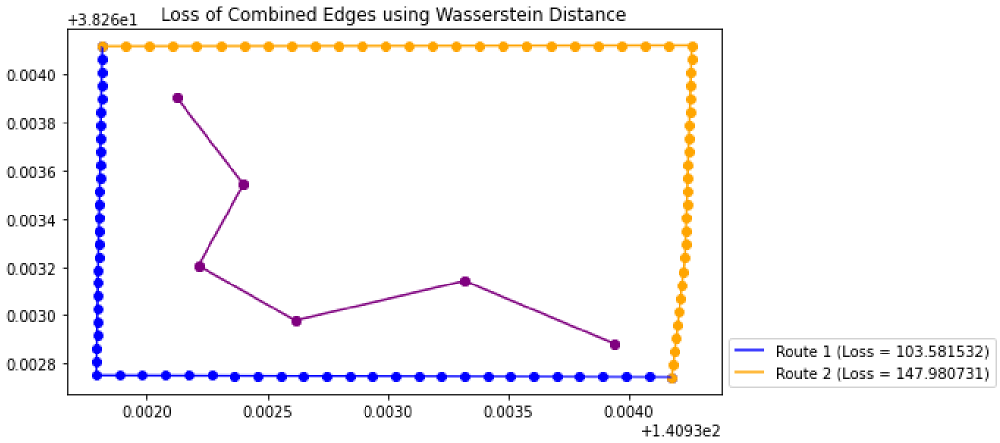
- For the trajectory $\mathbf{p}$, define $\mu_{\mathbf{p}} := (1/n) \sum_{p \in \mathbf{p}} \delta_p$.
- ▹ Devide each $A \in \mathcal{CR}$ into $m + 1$ equal parts and $V(A, m)$ denotes the set of m threshold points.
  ▹ Define $\nu_A := (1/m) \sum_{a \in V(A,m)} \delta_a$.





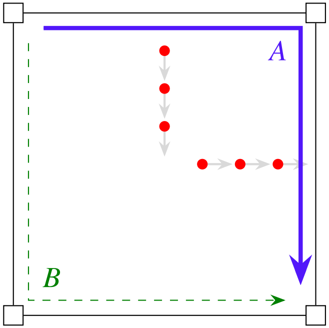
Compare $W_1(\mu_{\mathbf{p}}, \nu_A)$ with $W_1(\mu_{\mathbf{p}}, \nu_B)$.

Wasserstein Distance: Proof of Concept



The ratio of the route losses is 1.428

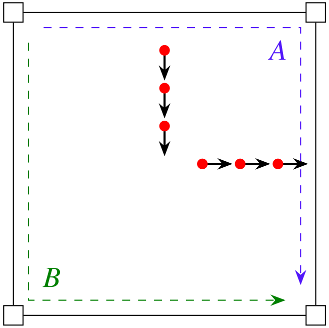
Motivation



- Each $p \in \mathbf{p}$ is located on the vertical or parallel bisectors.
- spd(p), Err(p) are the same at each p, respectively.

- Location only.
  - $W_1(\mu_{\mathbf{p}}, \nu_A) < W_1(\mu_{\mathbf{p}}, \nu_B)$.
- **We should select the route B.**
- Introduce probability measures $\mu^{\varepsilon}_{\mathbf{p},A}$, ν^{ε}_A and ν^{ε}_B that include speed and direction information.
  - Compare $W_1(\mu^{\varepsilon}_{\mathbf{p},A}, \nu^{\varepsilon}_A)$ with $W_1(\mu^{\varepsilon}_{\mathbf{p},B}, \nu^{\varepsilon}_B)$.
  - The effect of location is still strong.
- Normalization.

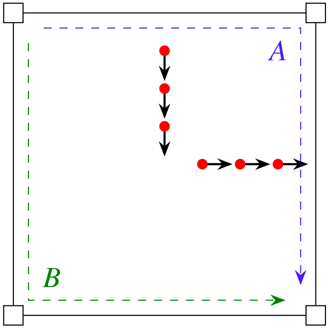
Motivation



- Each $p \in \mathbf{p}$ is located on the vertical or parallel bisectors.
- $\mathrm{spd}(p)$, $\mathrm{Err}(p)$ are the same at each p, respectively.

- Location only.
  - $W_1(\mu_{\mathbf{p}}, \nu_A) < W_1(\mu_{\mathbf{p}}, \nu_B)$.
- **We should select the route B.**
- Introduce probability measures $\mu^{\varepsilon}_{\mathbf{p},A}$, ν^{ε}_{A} and ν^{ε}_{B} that include speed and direction information.
  - Compare $W_1(\mu^{\varepsilon}_{\mathbf{p},A}, \nu^{\varepsilon}_{A})$ with $W_1(\mu^{\varepsilon}_{\mathbf{p},B}, \nu^{\varepsilon}_{B})$.
  - The effect of location is still strong.
- Normalization.

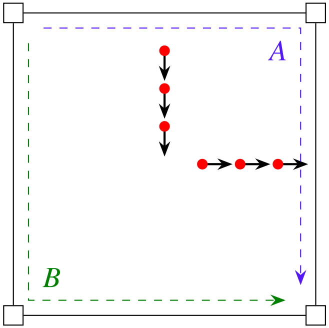
Motivation



- Each $p \in \mathbf{p}$ is located on the vertical or parallel bisectors.
- spd(p), Err(p) are the same at each p, respectively.

- Location only.
  - $W_1(\mu_{\mathbf{p}}, \nu_A) < W_1(\mu_{\mathbf{p}}, \nu_B)$.
- **We should select the route B.**
- Introduce probability measures $\mu^{\varepsilon}_{\mathbf{p},A}$, ν^{ε}_A and ν^{ε}_B that include speed and direction information.
  - Compare $W_1(\mu^{\varepsilon}_{\mathbf{p},A}, \nu^{\varepsilon}_A)$ with $W_1(\mu^{\varepsilon}_{\mathbf{p},B}, \nu^{\varepsilon}_B)$.
  - The effect of location is still strong.
- Normalization.

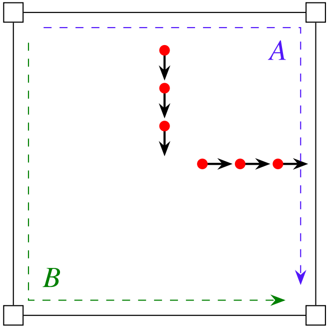
Motivation



- Each $p \in \mathbf{p}$ is located on the vertical or parallel bisectors.
- spd(p), Err(p) are the same at each p, respectively.

- Location only.
  - $W_1(\mu_{\mathbf{p}}, \nu_A) < W_1(\mu_{\mathbf{p}}, \nu_B)$.
- **We should select the route B.**
- Introduce probability measures $\mu^{\varepsilon}_{\mathbf{p},A}$, ν^{ε}_A and ν^{ε}_B that include speed and direction information.
  - Compare $W_1(\mu^{\varepsilon}_{\mathbf{p},A}, \nu^{\varepsilon}_A)$ with $W_1(\mu^{\varepsilon}_{\mathbf{p},B}, \nu^{\varepsilon}_B)$.
  - The effect of location is still strong.
- Normalization.

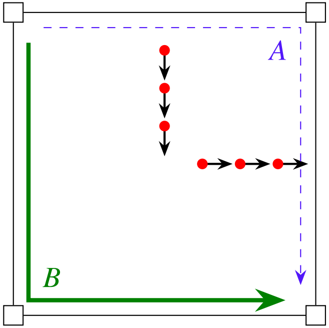
Motivation



- Each $p \in \mathbf{p}$ is located on the vertical or parallel bisectors.
- spd(p), Err(p) are the same at each p, respectively.

- Location only.
  - ▹ $W_1(\mu_{\mathbf{p}}, \nu_A) < W_1(\mu_{\mathbf{p}}, \nu_B)$.
- **We should select the route B.**
- Introduce probability measures $\mu^{\varepsilon}_{\mathbf{p},A}$, ν^{ε}_A and ν^{ε}_B that include speed and direction information.
  - ▹ Compare $W_1(\mu^{\varepsilon}_{\mathbf{p},A}, \nu^{\varepsilon}_A)$ with $W_1(\mu^{\varepsilon}_{\mathbf{p},B}, \nu^{\varepsilon}_B)$.
  - ▹ The effect of location is still strong.
- Normalization.
  - ▹ Compare $\dfrac{W_1(\mu^{\varepsilon}_{\mathbf{p},A}, \nu^{\varepsilon}_A)}{W_1(\mu_{\mathbf{p}}, \nu_A)}$ with $\dfrac{W_1(\mu^{\varepsilon}_{\mathbf{p},B}, \nu^{\varepsilon}_B)}{W_1(\mu_{\mathbf{p}}, \nu_B)}$.

Motivation



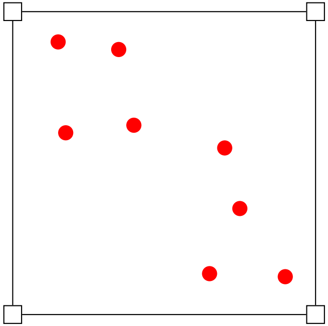
- Each $p \in \mathbf{p}$ is located on the vertical or parallel bisectors.
- spd(p), Err(p) are the same at each p, respectively.

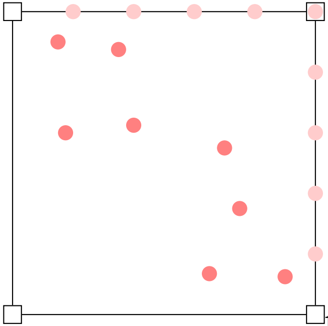
- Location only.
  - $W_1(\mu_{\mathbf{p}}, \nu_A) < W_1(\mu_{\mathbf{p}}, \nu_B)$.
- **We should select the route B.**
- Introduce probability measures $\mu^{\varepsilon}_{\mathbf{p},A}$, ν^{ε}_A and ν^{ε}_B that include speed and direction information.
  - Compare $W_1(\mu^{\varepsilon}_{\mathbf{p},A}, \nu^{\varepsilon}_A)$ with $W_1(\mu^{\varepsilon}_{\mathbf{p},B}, \nu^{\varepsilon}_B)$.
  - The effect of location is still strong.
- Normalization.
  - $\dfrac{W_1(\mu^{\varepsilon}_{\mathbf{p},A}, \nu^{\varepsilon}_A)}{W_1(\mu_{\mathbf{p}}, \nu_A)} > \dfrac{W_1(\mu^{\varepsilon}_{\mathbf{p},B}, \nu^{\varepsilon}_B)}{W_1(\mu_{\mathbf{p}}, \nu_B)}$.
  - **We can select the route B.**

Our strategy: Perturb $\mu_{\mathbf{p}}$ and each ν only by ε according to speed and direction.

- Let $0 < \varepsilon \ll 1$. For each $p \in \mathbf{p}$, $A \in \mathcal{CR}$, $a \in V(A,m)$ and $x \in V(A,m)$, define

$$\mu^{\varepsilon}_{\mathbf{p},A}(x) := \begin{cases} (1-\varepsilon)/n & (x = p), \\ \varepsilon \cdot (\textbf{our weight including } S\&D) & (x \in V(A,m)), \end{cases}$$

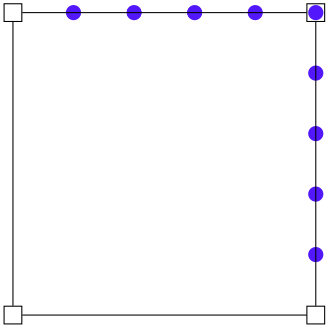


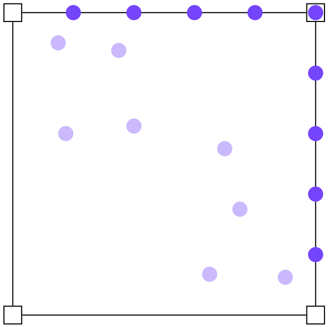


Our strategy: Perturb $\mu_{\mathbf{p}}$ and each ν only by ε according to speed and direction.

- Let $0 < \varepsilon \ll 1$. For each $p \in \mathbf{p}$, $A \in \mathcal{CR}$, $a \in V(A,m)$ and $x \in V(A,m)$, define

$$\nu_A^{\varepsilon}(x) := \begin{cases} (1-\varepsilon)/m & (x \in V(A,m)), \\ \varepsilon/n & (x \in \mathbf{p}). \end{cases}$$





Summary & Future problem

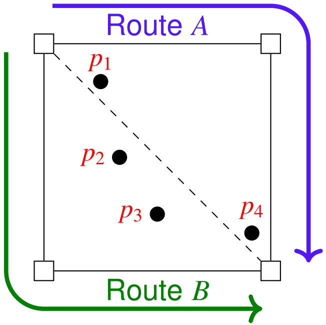
Summary

- Quantify the distance between $p \in \mathbf{p}$ and each $A \in \mathcal{CR}$ by making $\mu_{\mathbf{p}}$ and ν_A ε-pertubation according to speed and direction information.
- Conclude that **the route** A **with the smallest** $W_1(\mu^{\varepsilon}_{\mathbf{p},A}, \nu^{\varepsilon}_A)/W_1(\mu_{\mathbf{p}}, \nu_A)$ **is the true route.**

Future problem (from a theoretical point of view)

- Is this method also effective when $\mathcal{CR}$ is dense?
- Formulation of $\mu_{\mathbf{p}}$ with $(\gamma_p)_{p\in\mathbf{p}}$.

“Electric” Method : Review of Mid-presentation



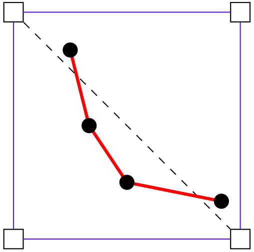


Figure: Connecting traj. pts. and giving charges.

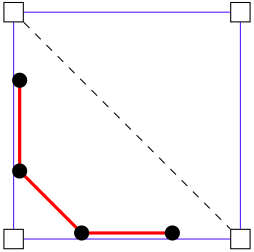


Figure: Moving to “closer” route.

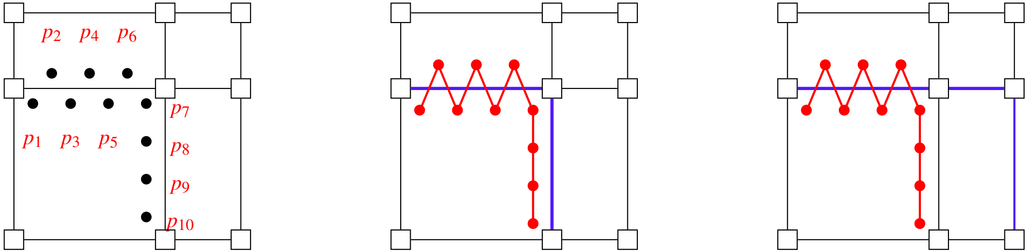
- Considering not only trajectory points, but also the entire polyline.

“Electric” Method : Problems

Problems:

- × Not taking into account speed and direction information.
- × Divergence problem : $\int_{\text{polyline}} \int_{\text{route}} r^{-2}$

- Even if $\int_{\text{polyline}} \int_{\text{route}} (r^2 + \varepsilon)^{-1}$, affects of intersection point is too large.



"Electric" Method : Strategy to Solve the Problems

Strategy:

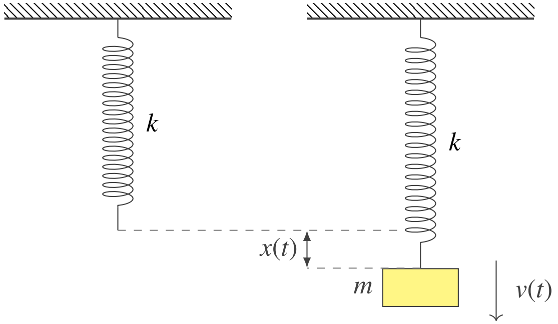
- Replace "maximizing inverse square" to "minimizing square";

$$\max_{A \in \mathcal{CR}} \frac{1}{r^2} \text{ : "electric" method}$$

$$\downarrow$$

$$\min_{A \in \mathcal{CR}} r^2 \text{ : "harmonic oscillator" method}$$

"Harmonic Oscillator" Method : General Setting



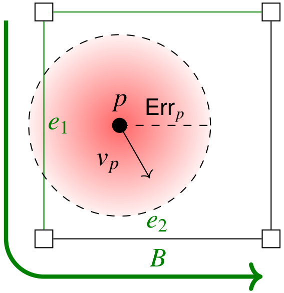
- m :mass,
- k :spring constant,
- $v(t)$:verocity of mass point,
- $x(t)$:displacement from natural length of spring
- *Lagrangian*

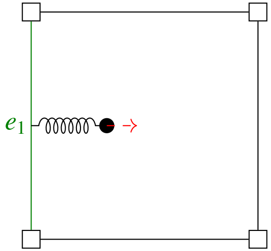
$$L(t) = \frac{1}{2}mv(t)^2 + \frac{1}{2}kx(t)^2$$

- *Action*

$$\text{Act} = \int L(t)\,dt = \int \left\{ \frac{1}{2}mv(t)^2 + \frac{1}{2}kx(t)^2 \right\} dt$$

"Harmonic Oscillator" Method : Settings in Map-matching Problem



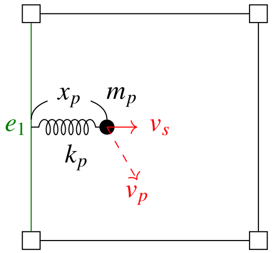


- $p \in \mathbb{R}^N$: trajectory point,
- $v_p = \mathrm{spd}_p u_p \in \mathbb{R}^N$: speed at p,
- $\mathrm{Err}(p) \in \mathbb{R}$: error of p,
- $B \in \mathcal{CR}, \quad e \in B$
- $S(p, e)$: *score* of edge e

$$:= \langle v, \vec{e} \rangle_{\mathbb{R}^N} \exp\left(-\mathrm{d}_{\mathrm{Err}}(p, e)\right)$$

- Connect trajectory point to the highest score edge by "spring".
- Define "Lagrangian" of this system.

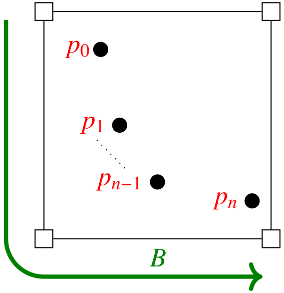
“Harmonic Oscillator” Method : Settings in Map-matching Problem

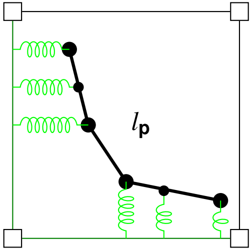


- v_s : spring direction component of v_p
- $x_p = d_{\mathrm{Err}_p}(p, e)$: “displacement” of p
- $m_p = \frac{1}{1+\mathrm{Err}(p)}$: “mass” of p,
- $k_p = \exp(-\mathrm{Err}(p))$: “spring constant” w.r.t. p,
- $M(p) = m_p \left\| \frac{v_s}{\log(1+|v_p|)} \right\|^2$: “momentum” of p,
- $P(p) = k_p x_p^2$: “potential” of p,

$L := M(p) + P(p)$: “Lagrangian”.

"Harmonic Oscillator" Method : Settings in Map-matching Problem





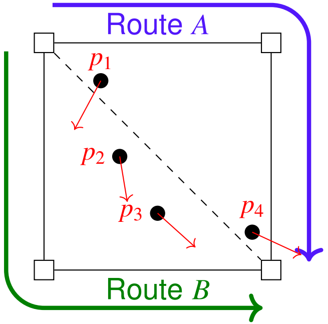
- $\mathbf{p} = (p_i)_{0\le i\le n}$: traj. points,
- $v_{\mathbf{p}} = (v_i = \mathrm{spd}_i u_i)_{0\le i\le n}$: velocity,
- Err : $\mathbf{p} \to \mathbb{R}$: error,
- $B \in \mathcal{CR}$: route.

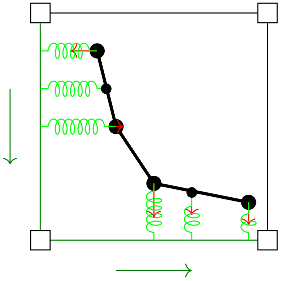
$$\downarrow$$

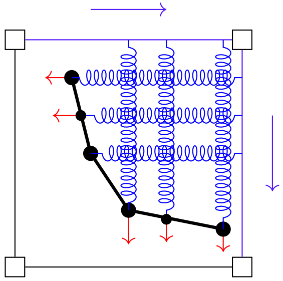
- $l_{\mathbf{p}} : [0,1] \to \mathbb{R}^N$: polyline,
- $v_{l_{\mathbf{p}}} : [0,1] \to \mathbb{R}^N$: velocity,
- $\mathrm{Err}_{l_{\mathbf{p}}} : [0,1] \to \mathbb{R}$: error,

$$L(t) := M(l_{\mathbf{p}}(t)) + P(l_{\mathbf{p}}(t)), \quad Act_{\mathbf{p}}(B) := \int_{[0,1]} L(t)\,dt$$: "action" of $\mathbf{p}$ on B.

"Harmonic Oscillator" Method : Settings in Map-matching Problem







- Calculate $Act_{\mathbf{p}}(A)$ and $Act_{\mathbf{p}}(B)$.
- Choose the route that minimize action.

"Harmonic Oscillator" Method : Strength/Weakness and Future Problem

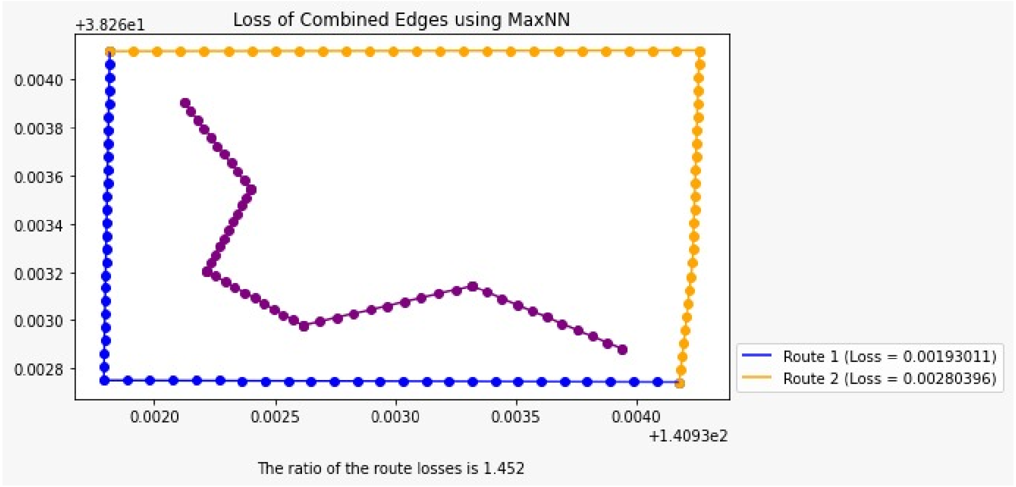
Strength/weakness:

- ✓ No divergence problem,
- ✓ Taking into account speed and direction naturally,
- × Insufficient consideration of the entire route.

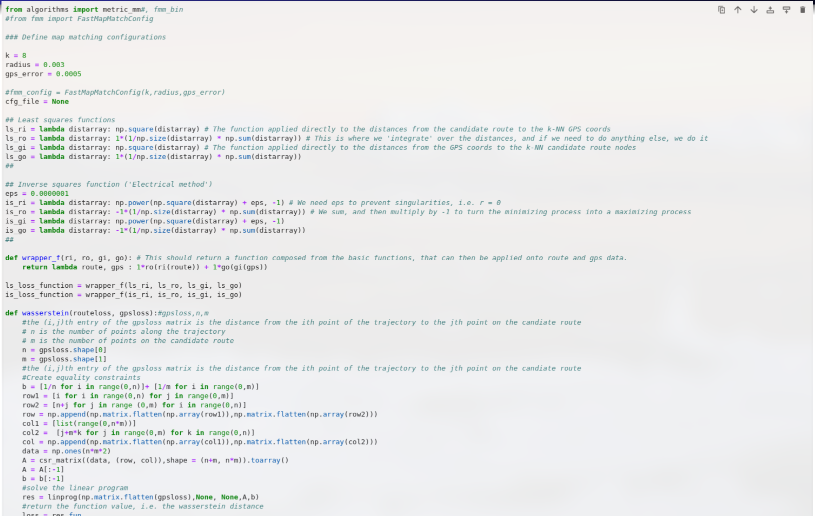
Future problems:

- Consider more appropriate way to define mass and spring constant.
- Connect to all edge of a route with appropriate weight to consider the whole route.

"Electric Method" and "Harmonic Oscillator": Proof of Concept



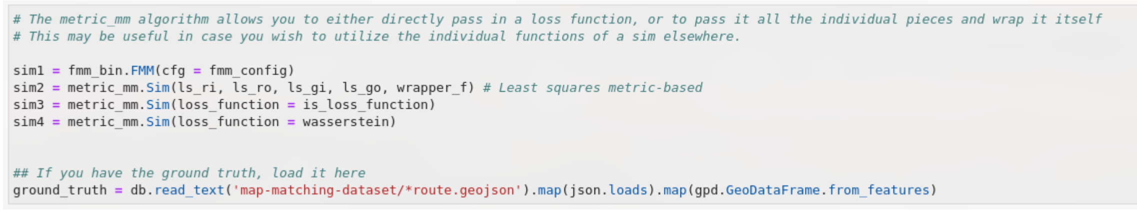
Implementing metric-based methods



```
from algorithms import metric_mm#, fmm_bin
#from fmm import FastMapMatchConfig

### Define map matching configurations

k = 8
radius = 0.003
gps_error = 0.0005

#fmm_config = FastMapMatchConfig(k,radius,gps_error)
cfg_file = None

## Least squares functions
ls_ri = lambda distarray: np.square(distarray) # The function applied directly to the distances from the candidate route to the k-NN GPS coords
ls_ro = lambda distarray: 1*(1/np.size(distarray) * np.sum(distarray)) # This is where we 'integrate' over the distances, and if we need to do anything else, we do it
ls_gi = lambda distarray: np.square(distarray) # The function applied directly to the distances from the GPS coords to the k-NN candidate route nodes
ls_go = lambda distarray: 1*(1/np.size(distarray) * np.sum(distarray))
##

## Inverse squares function ('Electrical method')
eps = 0.0000001
is_ri = lambda distarray: np.power(np.square(distarray) + eps, -1) # We need eps to prevent singularities, i.e. r = 0
is_ro = lambda distarray: -1*(1/np.size(distarray) * np.sum(distarray)) # We sum, and then multiply by -1 to turn the minimizing process into a maximizing process
is_gi = lambda distarray: np.power(np.square(distarray) + eps, -1)
is_go = lambda distarray: -1*(1/np.size(distarray) * np.sum(distarray))
##

def wrapper_f(ri, ro, gi, go): # This should return a function composed from the basic functions, that can then be applied onto route and gps data.
    return lambda route, gps : 1*ro(ri(route)) + 1*go(gi(gps))

ls_loss_function = wrapper_f(ls_ri, ls_ro, ls_gi, ls_go)
is_loss_function = wrapper_f(is_ri, is_ro, is_gi, is_go)

def wasserstein(routeloss, gpsloss):#gpsloss,n,m
    #the (i,j)th entry of the gpsloss matrix is the distance from the ith point of the trajectory to the jth point on the candiate route
    # n is the number of points along the trajectory
    # m is the number of points on the candidate route
    n = gpsloss.shape[0]
    m = gpsloss.shape[1]
    #the (i,j)th entry of the gpsloss matrix is the distance from the ith point of the trajectory to the jth point on the candiate route
    #Create equality constraints
    b = [1/n for i in range(0,n)]+ [1/m for i in range(0,m)]
    row1 = [i for i in range(0,n) for j in range(0,m)]
    row2 = [n+j for j in range (0,m) for i in range(0,n)]
    row = np.append(np.matrix.flatten(np.array(row1)),np.matrix.flatten(np.array(row2)))
    col1 = [list(range(0,n*m))]
    col2 =  [j+m*k for j in range(0,m) for k in range(0,n)]
    col = np.append(np.matrix.flatten(np.array(col1)),np.matrix.flatten(np.array(col2)))
    data = np.ones(n*m*2)
    A = csr_matrix((data, (row, col)),shape = (n+m, n*m)).toarray()
    A = A[:-1]
    b = b[:-1]
    #solve the linear program
    res = linprog(np.matrix.flatten(gpsloss),None, None,A,b)
    #return the function value, i.e. the wasserstein distance
    loss = res.fun
```

Implementing metric-based methods

Why are we defining each of the metric-based functions here, instead of in a separate Class? Because metric_mm is designed to accommodate any distance-based loss function, allowing for simple and customizable simulator creations.

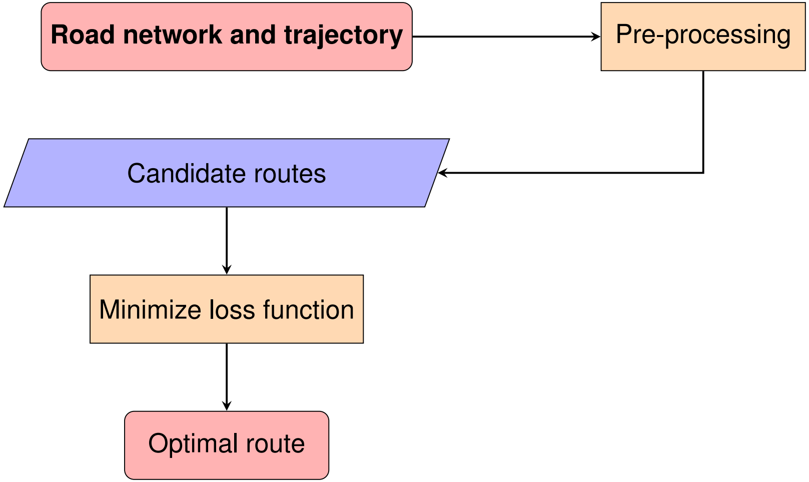


```
# The metric_mm algorithm allows you to either directly pass in a loss function, or to pass it all the individual pieces and wrap it itself
# This may be useful in case you wish to utilize the individual functions of a sim elsewhere.

sim1 = fmm_bin.FMM(cfg = fmm_config)
sim2 = metric_mm.Sim(ls_ri, ls_ro, ls_gi, ls_go, wrapper_f) # Least squares metric-based
sim3 = metric_mm.Sim(loss_function = is_loss_function)
sim4 = metric_mm.Sim(loss_function = wasserstein)


## If you have the ground truth, load it here
ground_truth = db.read_text('map-matching-dataset/*route.geojson').map(json.loads).map(gpd.GeoDataFrame.from_features)
```

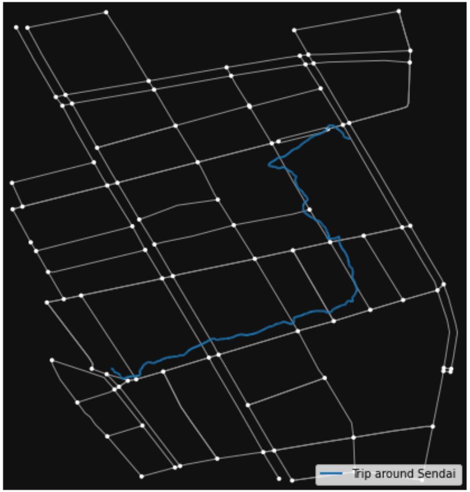
Sendai Map Revisited

Now we revisit the Sendai map case.

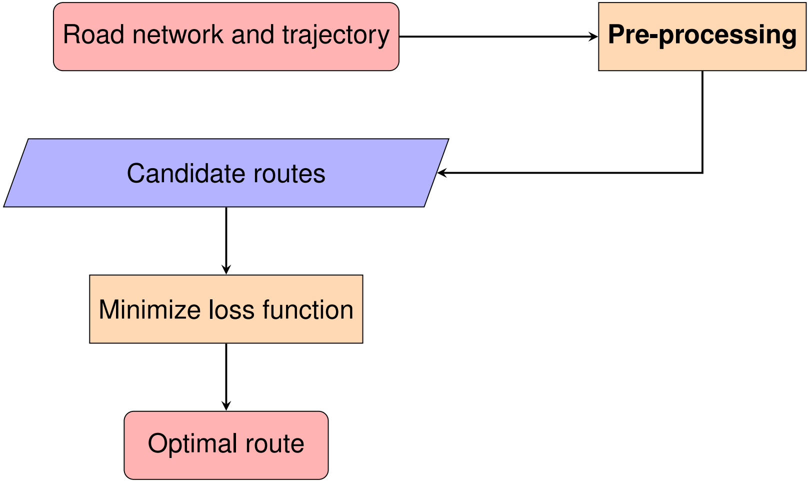
Map-matching Pipeline



Sendai Map Revisited



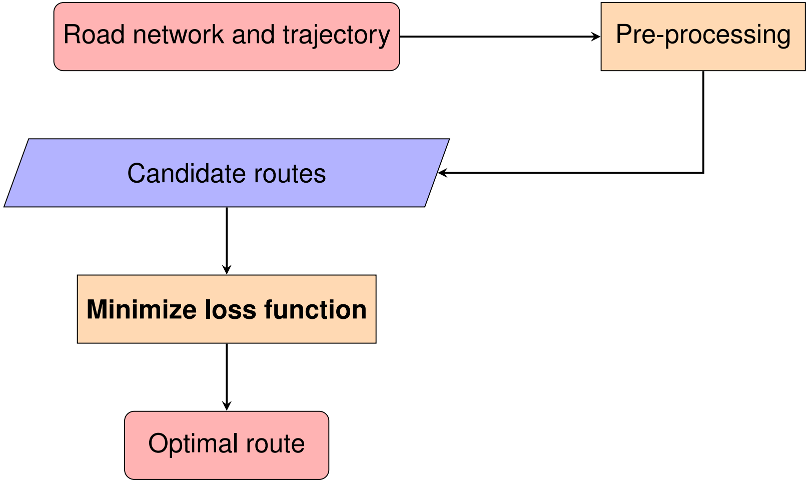
Map-matching Pipeline



Sendai Map Revisited

```
Obtaining Candidate Routes (Dijsktra’s Algorithm)
CPU times:  user 45.8 s, sys:  49.7 ms, total:  45.9 s
Wall time:  45.9 s

Preprocessing Candidate Routes
CPU times:  user 12.7 s, sys:  103 ms, total:  12.8 s
Wall time:  12.8 s
```

Map-matching Pipeline

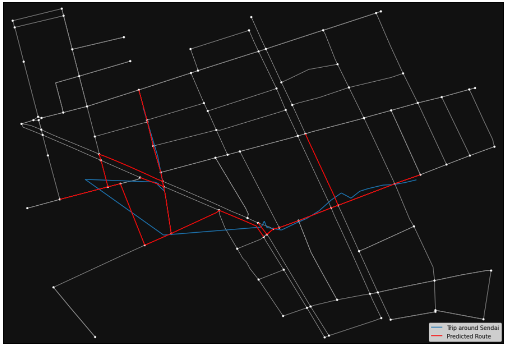


Sendai Map Revisited

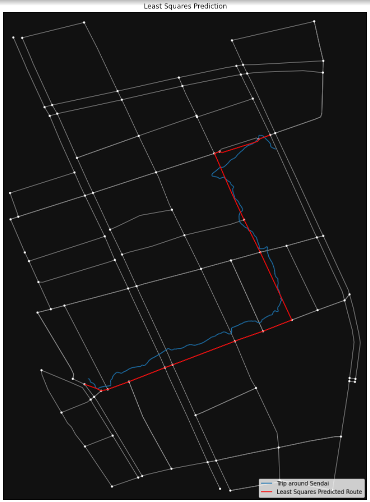
```
Least Squares Runtime
CPU times:  user 1.53 s, sys:  29.9 ms, total:  1.56 s
Wall time:  1.55 s

Inverse Squares Runtime
CPU times:  user 14.2 s, sys:  392 ms, total:  14.6 s
Wall time:  10.2 s

Wasserstein Runtime
(Computation times vary wildly based on parameters- anywhere
from 20 seconds, to 20 minutes, to 2.5 hours)
```

Sendai Map Revisited

Unfortunately, due to FMM relying on outdated libraries, we could not produce an image demonstrating FMM's performance on the new trip data. Our preliminary findings demonstrated that FMM still performed poorly on the dataset. For the sake of comparison, recall FMM's results on the older GPS dataset:

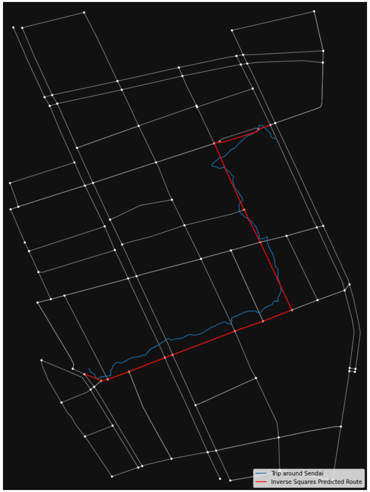


Sendai Map: Least Squares (Harmonic Oscillator) Result

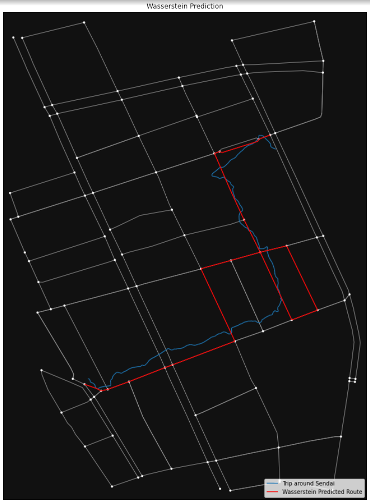


Sendai Map: Inverse Squares (Electrical Method) Result

Electric Method Prediction

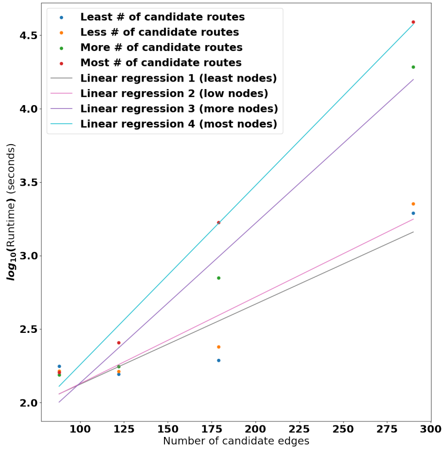


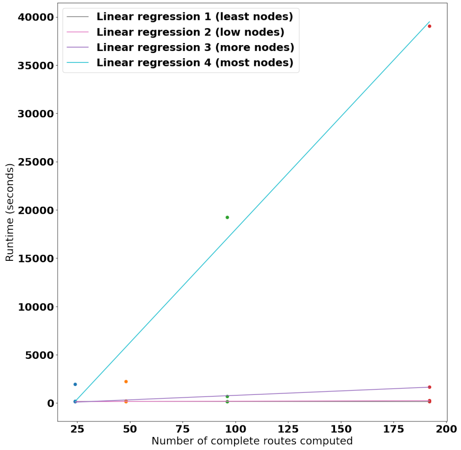
Sendai Map: Wasserstein Method Result



Computational Complexity (Preprocessing)

Preprocessing (Dijkstra) Runtime



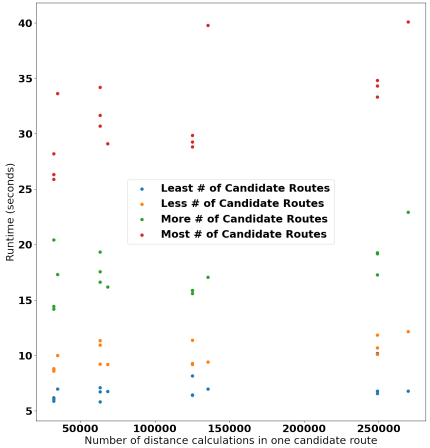


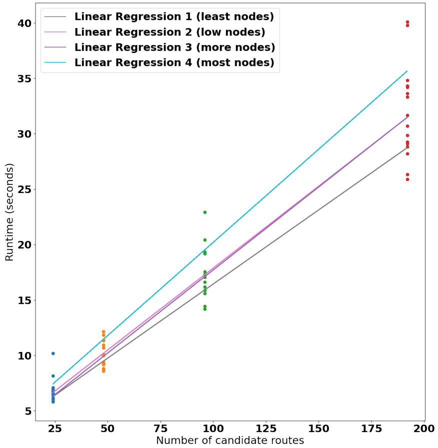
Computational Complexity (Metric Calculations)

The following scatter plots demonstrates how computation grows as a function of input nodes.

Computational Complexity (Metric Calculations)

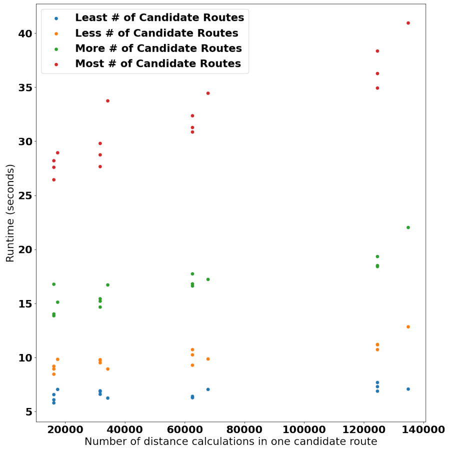
Least Squares Runtime

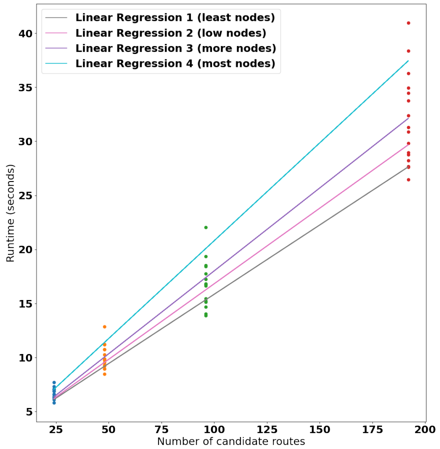




Computational Complexity (Metric Calculations)

Inverse Squares Runtime





Computational Complexity

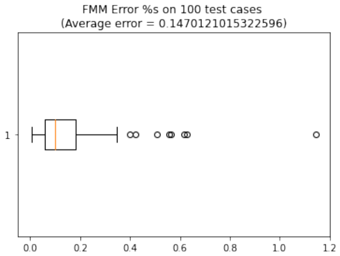
This suggests that simple distance methods (like inverse squares and least squares) have a computational complexity of $\mathcal{O}(n)$. Due to processing power, we could not obtain enough data to infer the computational complexity of the Wasserstein method. Because it relies on linear programming, we hypothesize that the growth rate is a polynomial of degree > 1.

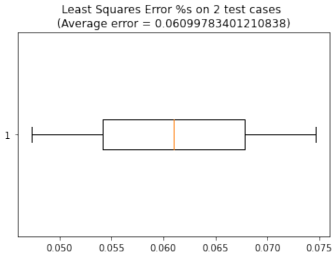
Some aspects of these algorithms can be improved upon– simple parallelization techniques offer a noticeable increase. However, other aspects are inherent to the method and are difficult to improve.

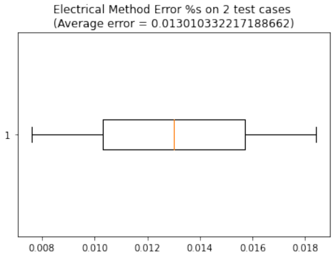
Preliminary Numerical Results

Due to limits in processing power, we were unable to test our algorithms against all of the data in the dataset. However, we demonstrate a few cases here.

Preliminary Numerical Results







Future Work (Implementation)

- While a naive Dijkstra method generates a good set of candidate routes, it has many shortcomings. It is still relatively slow, it is conditional on the given parameters, and it cannot handle stranger routes (traversing an edge more than once). Each of these individually can be addressed; or it may be more suitable to find an alternative candidate route generation method.
- Investigate Wasserstein method inconsistencies more thoroughly
- Incomplete road networks
- Implement IMU-based map matching approaches

Thank You! And References

- High-assurance Mobility Control Lab.
https://hmc.unist.ac.kr/research/autonomous-driving/
- M. Kubička, A. Cela, P. Moulin, H. Mountier and S. I. Niculescu, *Dataset for testing and training of map-matching algorithms*, In 2015 IEEE Intelligent Vehicles Symposium (IV), 1088–1093 (2015).
- F. Santambrogio, *Optimal transport for applied mathematicians. Calculus of variations, PDEs, and modeling*, Progress in Nonlinear Differential Equations and their Applications, Birkhäuser/Springer, Cham. (2015).
- F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan and T. Darrell, *BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning*, In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2636–2645 (2020).
- C. Yang and G. Gidófalvi, *Fast map matching, an algorithm for integrating a hidden Markov model with precomputation*, International Journal of Geographical Information Science. Taylor & Francis, **32**(3), 547–570 (2018).