Map-matching

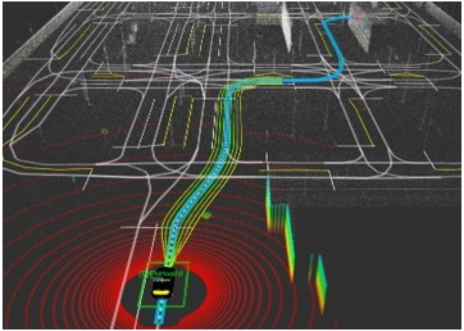
T. Akamatsu, G. Gress, K. Huneycutt, S. Omura
Academic Mentor: Dr. Kano
Industry Mentor: Dr. Yamazaki

July 15, 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 $d \geq 2$ (but almost everywhere we consider the case $d = 2$).

Definition (Trajectory)

A **trajectory** Tr is a sequence of points $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ where $p_i \in \mathbb{R}^d$ for $1 \leq i \leq n$ equipped with

- a sequence $t(\mathbf{p}) = (t_1, \ldots, t_n)$ such that $t_i \in \mathbb{R}^+$ for $1 \leq i \leq n$ and $t_1 < t_2 < \cdots < t_n$, called the **timestamp** of $\mathbf{p}$,
- a sequence $\mathrm{spd}(\mathbf{p}) = (\mathrm{spd}_1, \ldots, \mathrm{spd}_n)$ such that $\mathrm{spd}_i \in \mathbb{R}^+$ for $1 \leq i \leq n$, called the **speed** of $\mathbf{p}$ (optional),
- a sequence $u(\mathbf{p}) = (u_1, \ldots, u_n)$ such that $u_i \in \mathbb{R}^d$ and $\|u\| = 1$ for $1 \leq i \leq n$, 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 $\phi : |G| \to \mathbb{R}^d$ 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.

Problem Statement

Problem Statement

Definition (Route)

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

Problem Statement

Problem Statement

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 $L : R \to \mathbb{R}^+$, called the **loss function.**

Approaches to Map-Matching

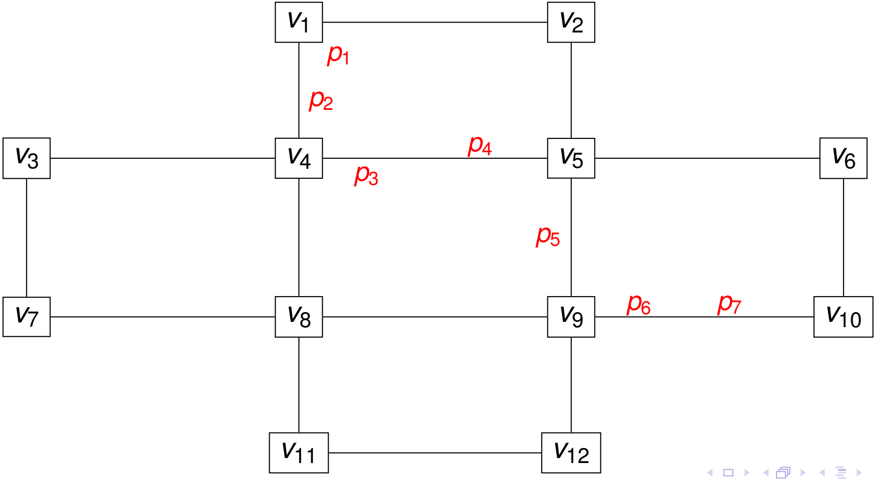
Geometric

- Point-to-point method
- Point-to-curve method

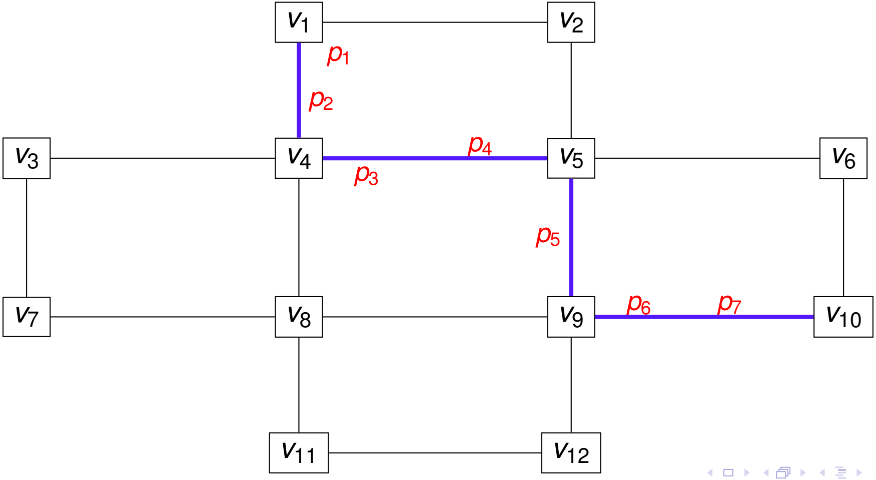
Data-Driven

- Hidden Markov model

Point-to-Curve Method

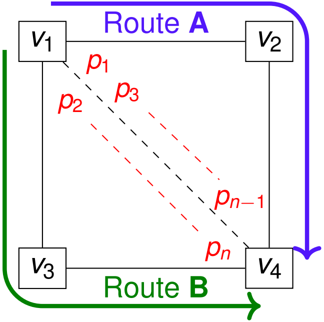


Point-to-Curve Method



Problem & Our strategy

- A square model.
  - $V = \{v_1, v_2, v_3, v_4\}$,
    $E = \{v_1 v_2, v_2 v_4, v_1 v_3, v_3 v_4\}$.
  - $\mathbf{p} = \{p_1, \ldots, p_n\}$: trajectory points
    which are located near the diagonal
    w/ coordinates and timestamps.
  - Route **A** $= \{v_1 v_2, v_2 v_4\}$.
  - Route **B** $= \{v_1 v_3, v_3 v_4\}$.



Strategy: Construction of the "**trajectory-to-route**"-type method.

"Wasserstein" method

Definition ((L^1-)Wasserstein distance ("W_1 *distance*"))

Let (X, d) be a complete and separable metric space.
For $\mu, \nu \in \mathscr{P}(X) :=$ { all (Borel) probability measures on (X, d) w/ finite support }, define

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

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

- W_1 distance is a distance function on $\mathscr{P}(X)$, i.e. quantifies the differences between two probability measures.
- W_1 distance can be calculated by linear programming (under our setting conditions).
- W_1 distance is also called "**Earth-Mover's distance**" or "**Word-Mover's distance**" (in areas such as Natural Language Processing).

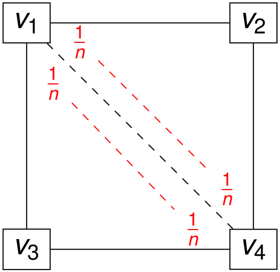


Figure: A prob. meas. $\mu_{\mathbf{p}}$ associated w/ the trajectory **p**. A weight $1/n$ is placed on each trajectory point; $\mu_{\mathbf{p}} := (1/n)\sum_{i=1}^{n}\delta_{p_i}$.

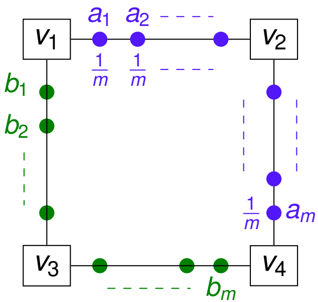


Figure: Prob. meas.s $\nu_{\mathbf{A}}=\nu_{\mathbf{A},m}$ and $\nu_{\mathbf{B}}=\nu_{\mathbf{B},m}$ associated w/ the route **A** and **B**; $\nu_{\mathbf{A}} := (1/m)\sum_{j=1}^{m}\delta_{a_j}$, $\nu_{\mathbf{B}} := (1/m)\sum_{j=1}^{m}\delta_{b_j}$.

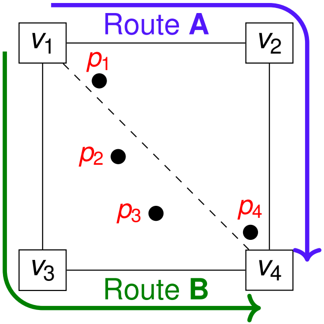
We define $\varphi(A) = \varphi(A, m) := W_1(\mu_{\mathbf{p}}, \nu_{\mathbf{A}}),\ \varphi(B) = \varphi(B, m) := W_1(\mu_{\mathbf{p}}, \nu_{\mathbf{B}})$.

$\rightsquigarrow$ If we obtain $\varphi(A) < \varphi(B)$, then we conclude that the route **A** is the true route.

Further problem: Construction of W_1 method taking speed and direction information into account.

- Modification of transport way (, i.e. the objective function of W_1 distance).
  - Loss function of W_1 method: $\sum_{x \in X} \sum_{y \in X} d(x, y)\pi(x, y)$.
  - Modified W_1 distance is likely to be difficult to handle.
- Modification of probability measures $\mu_{\mathbf{p}}$ (or $\nu_{\mathbf{A}}$ and $\nu_{\mathbf{B}}$).
  - We are trying to modify $\mu_{\mathbf{p}}$ using information from speed and direction information. (Under consideration...)

"Electrical charge" method



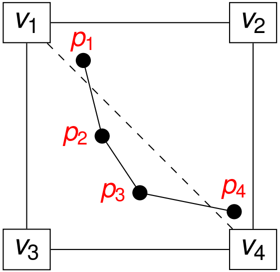


Figure: Connecting trajectory points.

- Considering not only trajectory points, but also the entire polyline.
- Comparing it with the entirety of each route.

"Electrical charge" method

1. Giving the candidate routes and polyline opposing electrical charges.
2. Choosing the route which exerts the most force on the polyline as the true route.

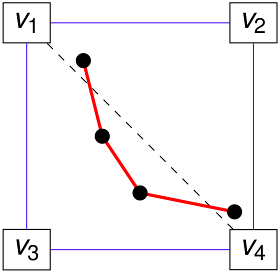


Figure: Giving electrical charges.

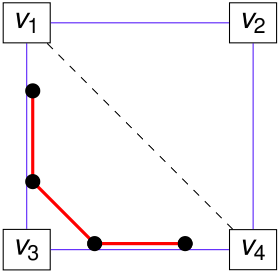


Figure: Moving to "closer" route.

“Electrical charge” method

Further problem: Taking into account information such as

- speed,
- direction,
- error.

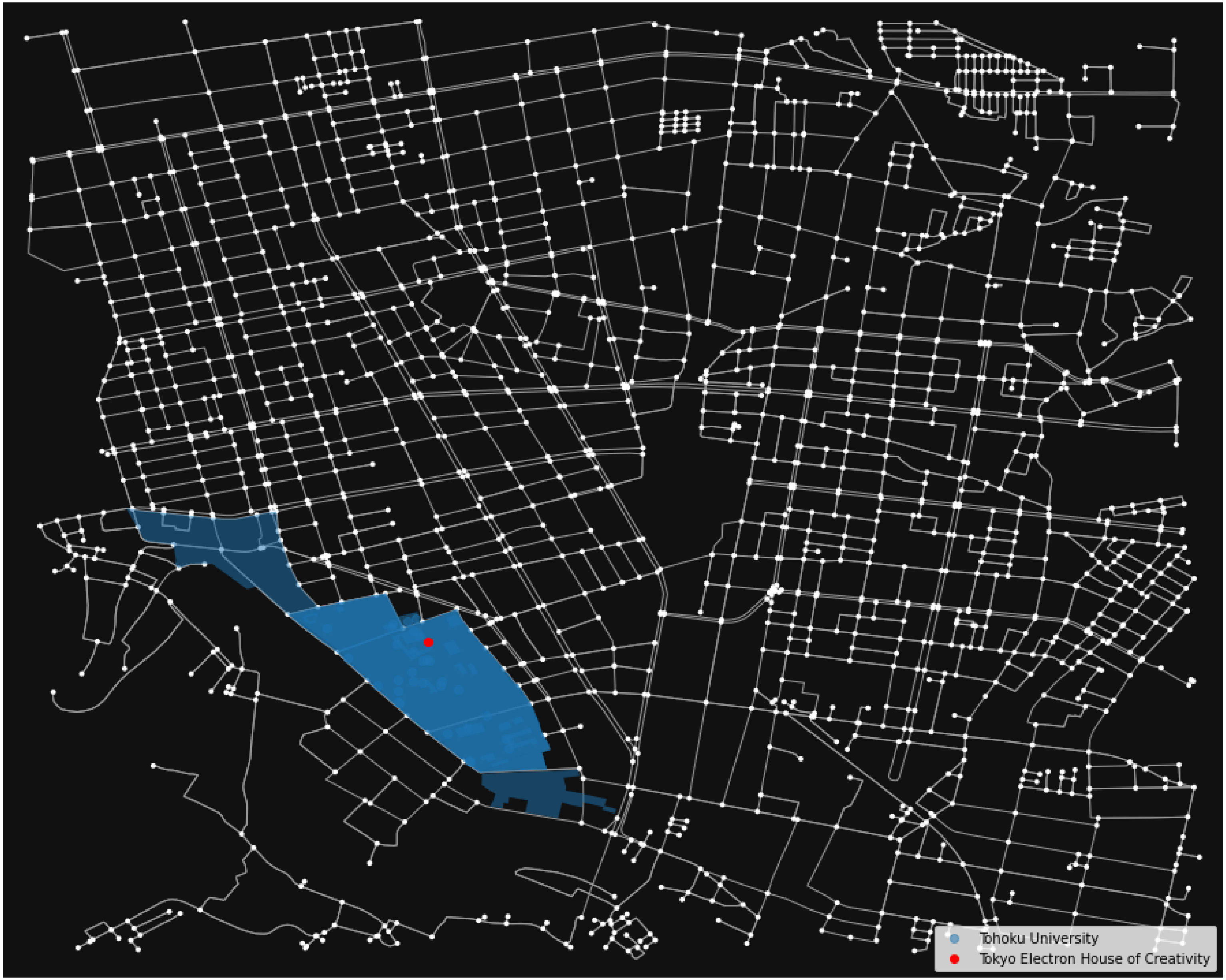
- Varying the electric density instead of assuming uniformity.

Jupyter Demonstration

How do we represent road networks and GPS sequences computationally?

```
In [2]: bignetwork = ox.graph_from_address(
        "Sendai, Minamimachi-dori, Chuo 3-chome, Aoba Ward, Sendai, Miyagi Prefecture
        dist=1750, network_type='drive')

        fig, ax = ox.plot_graph(bignetwork, figsize = (16,16),show=False,close=False)

        campus = ox.geometries.geometries_from_place('Katahira Campus ',tags = {'name
        campus.plot(ax=ax, alpha=0.5)

        gpd.GeoSeries([Point((140.87387,38.25448))]).plot(ax=ax, color='red')

        ax.legend(handles=[ax.collections[4],ax.collections[5]],
                  labels=['Tohoku University', 'Tokyo Electron House of Creativity'],
                  loc = 'lower right')
```

```
Out[2]: <AxesSubplot:>

Out[2]: <AxesSubplot:>

Out[2]: <matplotlib.legend.Legend at 0x7f678f643070>
```


Tohoku University
Tokyo Electron House of Creativity

The default format that OSMnx models networks is as `MultiDiGraph` objects. In order to manipulate these objects geometrically, we will need to convert this to `GeoDataFrame`.

In [3]:

```
gdf_nodes, gdf_edges = ox.graph_to_gdfs(bignetwork)
#gdf_edges.dtypes
gdf_edges = gdf_edges.reset_index([0,1,2])
gdf_edges = gdf_edges.set_index('osmid')
gdf_nodes[['geometry']].head()
```

Out[3]:

	geometry
osmid	
244879417	POINT (140.87168 38.26613)
244879418	POINT (140.87520 38.26010)
301789611	POINT (140.87349 38.25607)
301789618	POINT (140.87595 38.25510)
301789634	POINT (140.87949 38.25282)

In [4]:
```
gdf_edges[[ 'oneway', 'name', 'maxspeed', 'length', 'geometry']].head()
```

Out[4]:

	oneway	name	maxspeed	length	geometry
osmid					
218028552	True	定禅寺通	60	17.586	LINESTRING (140.87168 38.26613, 140.87148 38.2...
461330966	True	東二番丁通	60	66.732	LINESTRING (140.87168 38.26613, 140.87178 38.2...
30999231	True	青葉通	NaN	15.304	LINESTRING (140.87520 38.26010, 140.87503 38.2...
899682371	True	東二番丁通	60	94.102	LINESTRING (140.87520 38.26010, 140.87564 38.2...
837910375	False	NaN	NaN	11.727	LINESTRING (140.87349 38.25607, 140.87347 38.2...

Now let's look at the GPS data from a day walking around Sendai.

In [5]:

```
# Enable KML driver
fiona.drvsupport.supported_drivers["KML"] = "rw"

# Read file from KML
#fp = "history-2022-06-21.kml"
fp = "7-13-22.geojson"
with open(fp) as f:
    data = json.load(f)
    tripdata_nodes = gpd.GeoDataFrame.from_features(data)
tripdata_nodes = tripdata_nodes.sort_values(by='timestamp').reset_index(drop=
tripdata_edges = mm_utils.point_to_traj(tripdata_nodes)
```

In [6]:

```
fig, ax = ox.plot_graph(bignetwork, figsize = (16,16),show=False, close=False
tripdata_nodes.plot(ax=ax)
tripdata_edges.plot(ax=ax)
ax.legend(handles=[ax.collections[3]],labels=['Trip around Sendai'], loc = 'l
```

Out[6]:

```
<AxesSubplot:>
```

Out[6]:

```
<AxesSubplot:>
```

Out[6]:

```
<matplotlib.legend.Legend at 0x7f6732183f40>
```


Trip around Sendai

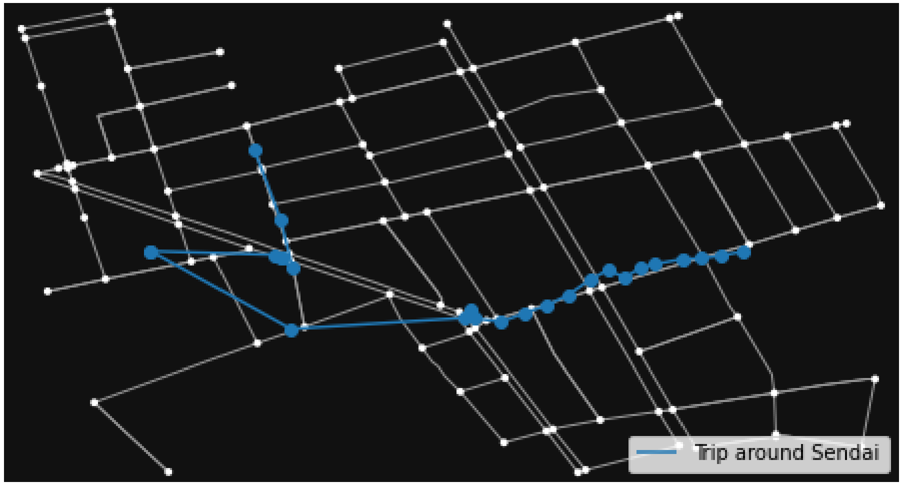
Let's zoom in a bit.

In [7]:
```
smallnetwork = mm_utils.df_to_network(tripdata_nodes, as_gdf = False)
fig, ax = ox.plot_graph(smallnetwork, figsize=(8,8), show=False, close=False)
tripdata_nodes.plot(ax=ax)
tripdata_edges.plot(ax=ax)
ax.legend(handles=[ax.collections[3]],labels=['Trip around Sendai'], loc = 'l
```

Out[7]:
```
<AxesSubplot:>
```

Out[7]:
```
<AxesSubplot:>
```

Out[7]:
```
<matplotlib.legend.Legend at 0x7f678f62f0a0>
```



In [8]:
```
from algorithms import fmm_bin
from fmm import FastMapMatchConfig
### Define map matching configurations

k = 8
radius = 0.003
gps_error = 0.0005

# create a text trap and redirect stdout
#text_trap = io.StringIO()
#sys.stdout = text_trap

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



fmm_sim = fmm_bin.FMM(cfg = fmm_config)

fmm_sim.run(tripdata_edges)

# now restore stdout function
#sys.stdout = sys.__stdout__
```

```
/home/gjgress/G-RIPS-2022-Mitsubishi-A/Code/algorithms/fmm_bin.py:50:
UserWarning: Column names longer than 10 characters will be truncated
when saved to ESRI Shapefile.
  gdf_nodes.to_file(filepath_nodes, encoding=encoding)
```

```
[2022-07-15 01:12:42.821] [info] [network.cpp:72] Read network from f
```

```
ile temp/network_edges.shp
[2022-07-15 01:12:42.830] [info] [network.cpp:170] Number of edges 21
8 nodes 104
[2022-07-15 01:12:42.830] [info] [network.cpp:171] Field index: id 12
source 0 target 1
[2022-07-15 01:12:42.830] [info] [network.cpp:174] Read network done.
[2022-07-15 01:12:42.875] [info] [network_graph.cpp:17] Construct gra
ph from network edges start
[2022-07-15 01:12:42.875] [info] [network_graph.cpp:30] Graph nodes 1
04 edges 218
[2022-07-15 01:12:42.875] [info] [network_graph.cpp:31] Construct gra
ph from network edges end
[2022-07-15 01:12:42.875] [info] [ubodt_gen_algorithm.cpp:76] Start t
o generate UBODT with delta 0.02
[2022-07-15 01:12:42.875] [info] [ubodt_gen_algorithm.cpp:77] Output
format csv
[2022-07-15 01:12:42.881] [info] [ubodt_gen_algorithm.cpp:105] Progre
ss 10 / 104
[2022-07-15 01:12:42.882] [info] [ubodt_gen_algorithm.cpp:105] Progre
ss 20 / 104
[2022-07-15 01:12:42.883] [info] [ubodt_gen_algorithm.cpp:105] Progre
ss 30 / 104
[2022-07-15 01:12:42.883] [info] [ubodt_gen_algorithm.cpp:105] Progre
ss 40 / 104
[2022-07-15 01:12:42.884] [info] [ubodt_gen_algorithm.cpp:105] Progre
ss 50 / 104
[2022-07-15 01:12:42.887] [info] [ubodt_gen_algorithm.cpp:105] Progre
ss 60 / 104
[2022-07-15 01:12:42.888] [info] [ubodt_gen_algorithm.cpp:105] Progre
ss 70 / 104
[2022-07-15 01:12:42.891] [info] [ubodt_gen_algorithm.cpp:105] Progre
```

```
ss 80 / 104
[2022-07-15 01:12:42.893] [info] [ubodt_gen_algorithm.cpp:105] Progre
ss 90 / 104
[2022-07-15 01:12:42.894] [info] [ubodt_gen_algorithm.cpp:105] Progre
ss 100 / 104
[2022-07-15 01:12:42.900] [info] [ubodt.cpp:208] Reading UBODT file
(CSV format) from /tmp/tmp2htuo9dz
[2022-07-15 01:12:42.906] [info] [ubodt.cpp:243] Finish reading UBODT
with rows 9805
[2022-07-15 01:12:42.907] [info] [gps_reader.cpp:337] GPS data in tra
jectory shapefile format
[2022-07-15 01:12:42.907] [info] [gps_reader.cpp:45] Read trajectory
from file /tmp/tmphjew8268.shp
[2022-07-15 01:12:42.907] [warning] [gps_reader.cpp:69] Timestamp col
umn timestamp not found
[2022-07-15 01:12:42.907] [info] [gps_reader.cpp:81] Total number of
trajectories 24
[2022-07-15 01:12:42.907] [info] [gps_reader.cpp:82] Finish reading m
eta data
```

In [13]:
```
fmm_sim.results[['index', 'osmid', 'geometry']].head()
```

Out[13]:

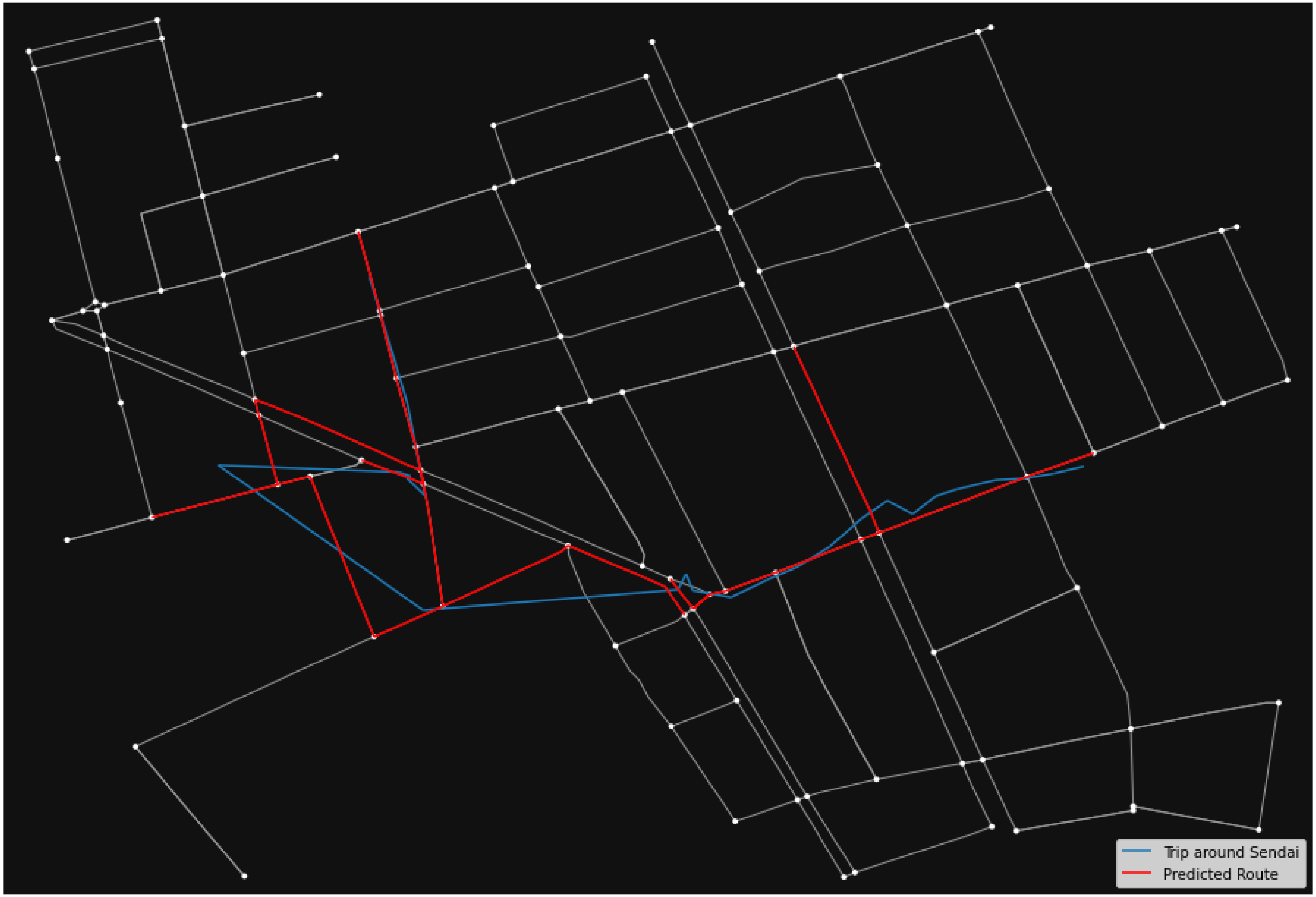
	index	osmid	geometry
0	0	837910375	LINESTRING (140.87349 38.25607, 140.87347 38.2...
1	1	[837910369, 837910371]	LINESTRING (140.87349 38.25607, 140.87335 38.2...
2	4	32896012	LINESTRING (140.87595 38.25510, 140.87576 38.2...
3	11	153276508	LINESTRING (140.87697 38.25709, 140.87702 38.2...
4	13	837910348	LINESTRING (140.87916 38.25612, 140.87909 38.2...

In [11]:

```
mm_utils.plot(network = smallnetwork, input_data = tripdata_edges, results = 
plt.gca().legend(handles=[plt.gca().collections[2],plt.gca().collections[3]],
```

Out[11]:

```
<matplotlib.legend.Legend at 0x7f673022d480>
```



Datasets

After formulating the proposed mathematical methods into robust map-matching algorithms, we will implement them in python to evaluate their performance numerically using these datasets:

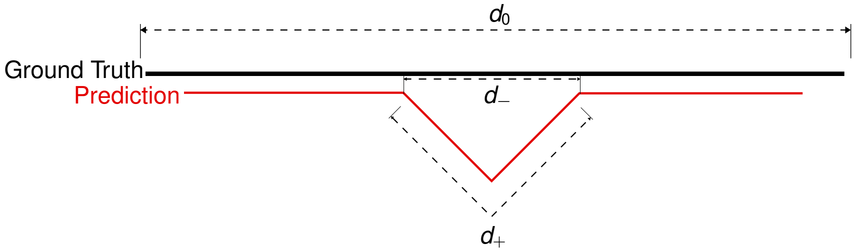
- *Dataset for testing and training of map-matching algorithms* [KCMMN] (GPS only, has ground truths),
- The BDD100K open data set provided by Berkeley [YCWXCLMD] (for GPS and IMU data, no ground truths).[1]

We will also compare the performance of our methods to a geometric method, such as point-to-curve, and HMM method, such as an extended Kalman filter (EKF) or Fast Map-Matching [YG].

[1] Because there are no public annotated ground truths, we compare our predictions with the standard EKF approach. This evaluation method is flawed but unavoidable.

Evaluation

How do we measure the accuracy of our prediction?



$$\mathrm{Err} = \frac{d_- + d_+}{d_0}$$

d_0 = length of ground truth

d_- = length of prediction route erroneously subtracted

d_+ = length of prediction route erroneously added

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).