Mapping Technologies

Petteri Nurmi, Farbod Faghihi
Questions

• How to represent maps and movement constraints?

• How to match location measurements with movement constraints?

• How does point-to-point matching work? What about trajectory-based matching?

• How movement constraint information can be constructed?
Representing Movement Constraints

- Location measurements noisy and do not obey real world obstacles
  - Measurements can be inside walls or other inaccessible areas
  - Sensed trajectories can jump between roads or even pass through buildings or other obstacles
- Information about logical real world constraints can be used to improve quality of location information
- This lecture looks at ways to represent, use, and construct such constraints
Representing Movement Constraints

- Two common techniques:
  - Grid-based models
    - World modeled as grid cells, each cell associated with accessibility information (walkable / not)
    - Alternatively, each grid can be divided into pixels and each pixel associated with binary value indicating accessibility
  - Graph-based models:
    - World modeled as a graph, where vertices denote junction points and edges denote paths between them
    - Common example is a road network
    - But also useful for indoor areas, e.g., each room or corridors considered as a vertex and edges determine possible transitions between areas
Accessibility: Grid / Discrete partitioning

- Simplest solution is to use a discrete partitioning to represent movement constraints
  - Each partition associated with a value that determines whether it is accessible
  - Binary indicator or nominal value
- Hierarchical variant
  - Associate each grid with a pixel map
  - Each pixel in a grid associated with accessibility information (e.g., binary value)
- Different types of partitions can be used:
  - Rectangular, triangular, hexagonal, ...
Example of discrete partitioning: Predestination - Route Guidance

- Probabilistic navigation system for drivers
- Potential destinations are inferred using probabilistic inference
  - Personal model: destinations visited by the user
  - Global model: destinations visited by other users
  - General model: all possible destinations
- Grid-based accessibility data used to assign zero probabilities to inaccessible areas
  - Lakes, mountains, buildings, and other areas where the user cannot drive
Example of discrete partitioning: Viewsheds

- Viewshed corresponds to the geographical area visible from a location
  - Used in terrain analysis, urban planning, archaeology, network planning, military science, …
- Can be represented as *raster* maps
  - "Images" where each pixel associated with binary visibility information
  - Can be calculated automatically from GIS data
Accessibility: Graphs

• Travel constraints (especially road networks) often represented as *graphs*
• Vertices/nodes correspond to intersections or other key points (e.g., landmarks)
• Edges determine the *topological connectivity* between vertices
  – I.e., whether a vertex a can be accessed from vertex b
• Note: while edges are usually represented as straight lines, in geographic context they represent segments
• Each segment can be associated with multiple points and the shape of the segment can be arbitrary
Best known method for determining the shortest path in a graph is Dijkstra’s algorithm. Start from a source node and end when the shortest path to target (or sink) node found. Initialization: set all nodes are unprocessed and mark distance to each node to infinity. Initialize a priority queue with the source node as only element. Operation: pick one unprocessed node from the queue at a time. Set distance to node as the minimum of path cost from any processed node. Add all unprocessed neighboring nodes to the queue. Iterate until queue empty. Another popular alternative is to use the A* algorithm.
Shortest Path: Dijkstra
Shortest Path: Dijkstra
Example: Navigation Systems

- Navigation systems can use graph-based accessibility models to determine optimal path to destination
  - Paths can be pre-calculated to improve runtime performance
  - Trajectory segmentation can be used to determine where to present instructions
Constructing Accessibility Maps

- Two main approaches to constructing maps with accessibility information
  - Digitization: extraction of information from imagery
    - Initially was based on manual effort
    - Nowadays programmable based on remote sensing data (e.g., satellite imagery)
  - Trace-driven: extract information from location traces
    - Administrative/dedicated: dedicated data collection effort (e.g., ordnance surveys)
    - Crowdsourcing: anyone can contribute (OpenStreetMap)
- Hybrid approaches combine both
  - E.g., digitization can be used to post-process / correct trace-driven maps
Constructing Accessibility Maps: Criteria

- Coverage: How large geographic area does the information cover?
- Resolution: How detailed information does the map contain?
- Reliability/accuracy: How correct is the information?
- Actuality: Is the information up-to-date or have paths/routes changed significantly?
- Geographical distribution: How equally different areas are covered?
- Logical consistency: How well topological relationships are preserved?
Crowdsourcing-Based Construction: Issues

- Data quality and reliability
  - Anyone can provide data, hence quality of traces varies and data can be erroneous
    - Subgraphs are not necessarily closed when they should be
    - Possibility of duplicates
  - Varying geographical coverage, some areas high quality and dense coverage, others more sparse and inaccurate
- Lack of common ontology
  - Geographic entities can have different tags associated with them
- Vulnerable to malice / fraudulent data
OpenStreetMap (OSM)

- OpenStreetMap was created as a free geographical database of the world with the help of crowdsourcing.
- Inspired by peer production based services such as Wikipedia, Open Street Map was created based on the User Generated Content (UGC), which is a known feature of Web 2.0. The abundance of GPS-enabled devices enabled the gathering of geo-referenced data by users.
- Specifically, OSM is based on large number of users collaboratively edit the world map which is known as Volunteered Geographic Information (VGI).
- Registered users can edit and contribute by uploading GPX traces obtained by tracking devices, resolving the errors, and digitizing geographical features. Anonymous users are not supported because of legal reasons.
OSM GeoStack

- OSM was founded in 2004 at University of College London as a worldwide geographical database.
- OSM infrastructure is known as GeoStack with the following components: Editing Tools, Technical Infrastructure, Mapping Outputs.
OpenStreetMap – Editing

- Users can collect data and contribute to the project by gathering GPS traces, taking photos, and written or spoken notes.
- After the data collection, Users can add the information to the OpenStreetMap database by using one of the freely available editing applications: Potlatch, JOSM, Merkaartor
- From 2010, users been able to collect data by using Bing aerial imagery which enables them not be physically available at the region to collect data.
Each of the provided editing applications have their own characteristics that the user can choose from based on their needs:

<table>
<thead>
<tr>
<th></th>
<th>JOSM</th>
<th>Potlatch</th>
<th>Merkaartor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Platform</strong></td>
<td>Operating systems with</td>
<td>Web browser with flash</td>
<td>Operating systems with QT</td>
</tr>
<tr>
<td></td>
<td>Java SE 1.5 or higher</td>
<td></td>
<td>installed</td>
</tr>
<tr>
<td><strong>Runs without</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>installation</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Offline editing</strong></td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>mode</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Photo mapping</strong></td>
<td>Yes</td>
<td>Partially</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>support</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Audio mapping</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>support</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Geographical features in OSM

- OSM's philosophy towards doing things is "the simplest thing that will work", and that's why they use a unique different data model compared to other Geographic Information Systems.
- The geographical features have two categories: Primitive types and free-form tagging.
Data primitives are composed of nodes, ways, and relations. They are tagged with key-value pairs to represent the geographical features they refer to.

XML format is used for representing the data model. Each primitive type has a numerical ID, and the id is only unique within the data type category. Therefore, a node and a way may have the same numerical ID.

Each data object in the database contains a version number and a changeset ID that refers to the set of changes that version was created. In addition, the display name and user ID of the user who has made that change are also recorded. Display name shows the current chosen display name by the user, but the user ID is always unique. The other attributes are visible and timestamp. If visible is true means that the current object is the last version, otherwise the object is outdated. Timestamp refers to the time when the object was last updated.
Data Primitives - Nodes

- Nodes are used to represent the points. It’s the only primitive data that contains positional information and other data types depend on nodes for locating their placement.
- Nodes can be used as the junction point of the multiple ways, showing the change in direction, or representing a Point of Interest (POI).
- Here follows the XML code for a node which represents the Unicafe located in Physicum. The longitude and latitude are stated up to 7 decimal places. This gives a latitudinal and a longitudinal resolution of around 1cm.
- The only childs of the nodes are tags which are key-value pairs giving information regarding the point.

```
<node id="1412998560" visible="true" version="2" changeset="21305629" timestamp="2014-03-25T14:09:40Z" user="ij_" uid="139957" lat="60.2048979" lon="24.9628913">
    <tag k="amenity" v="cafe"/>
    <tag k="name" v="Unicafe Physicum"/>
    <tag k="operator" v="Unicafe"/>
</node>
```
Data Primitives - Ways

- Ways are made of an ordered list of nodes and used for representing linear features such as paths, waterways, and roads.
- Ways are semi-independent of the nodes it contains. If the nodes in the way or the taggings are left the same, the way's version will remain the same even if the nodes were moved.
- The direction of a way is stated by the order of the nodes that it's made of. A way should contain a minimum of 2 nodes and a maximum of 2000 nodes (limit caused for performance optimisation).
- Areas are also represented by ways. Areas are defined by ways where the first and last node of the way are the same.
- Two ways can cross each other but they only have a common node if they are physically connected.
- Ways do not have any unique attribute. It only has a new child `<nd>` which has the `ref` attribute to reference a node that makes the way.
Data Primitives - Relations

- Relations are used to provide a list of primitive data types. It can contain nodes, ways, and even relations.
- Used for geographical features that cannot be captured with a node, or when the two overlap such as branching streets, long distance routes.
- The attributes used by the relation tag is the same as the attributes used by other primitive data types. It has member child elements that lists the objects it is made of. Member tags have type which shows the datatype, ref referencing the object, and role which can be empty but in general explains the role of the element in the relation.
Changesets are another data type used in OSM data model that group the changes made to the primitive data types. Changesets are created to better track and manage the changes made to the features on the map. Every modification should be mapped to a changeset.

Here's an example of the changeset tag which correspond to the change group of the Unicafe node example. You can see that the changes themselves are not part of the changeset tag and they are only saved on the OSM database. It is possible to download another XML file that contains the changes using the reference.

```xml
<changeset id="21305629" created_at="2014-03-25T14:09:39Z" closed_at="2014-03-25T14:09:40Z" open="false" user="ij_" uid="139957" min_lat="60.2039291" min_lon="24.9618375" max_lat="60.2053129" max_lon="24.9640491" comments_count="0">
  <tag k="comment" v="tweak"/>
  <tag k="created_by" v="JOSM/1.5 (5623 SVN en)"/>
</changeset>
```
Tagging

• Tags are the key-value pairs that give information about what the feature mapped by the primitive data types are. The key-value pair can hold any valid unicode characters up to 255 characters. They are the semantic aspect of the data model.

• There's no other restriction on what the key-pair value can contain. It is possible to use custom tags. In general the tags follow the same key=value structure. The format that is mostly used for the values are words separated by underline character.

• In certain cases where key has multiple values, a prefix or a suffix is used after a colon.

<tag k="name" v="Helsinki"/>
<tag k="name:ca" v="Hèlsinki"/>  
<tag k="name:de" v="Helsinki"/>  
<tag k="name:en" v="Helsinki"/>  
<tag k="name:fi" v="Helsinki"/>  
<tag k="name:ru" v="Хельсинки"/>  
<tag k="name:sv" v="Helsingfors"/>
Exporting customized maps

- OpenStreetMap enables creating of custom maps in line with your specifications and export them.
- One of the simplest ways for this aim is to use OpenStreetMap exporter. You can select your area of interest and download the data corresponding to it either by downloading the XML data or by downloading an image or creating a HTML code to embed the map into your Location based Application.
Exporting Raw Data

- There are different ways that you can export and make use of the data provided by the OpenStreetMap for your own purposes. The raw data can be extracted by getting the planet files, using the OSM API, and using the Extended API (XAPI).

- Each of these methods have their own advantages and disadvantages that should be taken into account based on your application needs. All of the specified to extract data use XML and it requires familiarity with the language to be able to use the data.
Planet Files

• Planet files are a dump of the whole OSM database in an XML file. Therefore, in situations where there's a need for the whole world map data Planet files can be used. The Planet files in compressed format are around 60GBs in size.

• There's a weekly update for the new planet files containing all the new features and changes. If the location based service needs a real-time need for the new changes, between the release of each planet file OSM releases Diff files containing the changes made after the last full planet file release which can be used for updating the data.

• The planet files can be accessed from the following URL: http://planet.openstreetmap.org/

• Because of the size of the planet files that makes maintaining the database difficult and the use case of the location based application, there only might be a need for the data for a continent or a country. These can be extracted from the following URL: http://download.geofabrik.de/osm/
OpenStreetMap REST API

- OpenStreetMap API is a web service that enables the direct access to the database.
- OSM API always return the last version of a data object. Therefore, if there's multiple sources to access a data, we can use the API with ease of mind since we know it returns the last update of the data.
- The API can be used easily to extract data but for any other operations such as deleting and updating the user should be registered and use authentication before being able to perform the specified tasks.
- The needed data can be easily retrieved by using its URI using an HTTP request:  http://api.openstreetmap.org/api/<apiversion>/<type>/<id>
- Where **apiversion** is the current version 0.6, type is either node, way, and relation, and the id is the reference id of the object.
OpenStreetMap REST API

- `/full` can be added to the given URI to also extract all the nodes contained in a way or a relation in addition to the object itself.
- `/ways` can be used to extract all the ways that use a specific node.
- `/relations` can be used to extract all the relations that use a specific feature is a member of.
- `/history` can be used to extract the update history of a specific feature.
- It’s also possible to retrieve all the geographical features in an area using a bounding box:
  - `http://api.openstreetmap.org/api/0.6/map?bbox=<left>,<bottom>,<right>,<top>` where `left`, `bottom`, `right`, and `top` respectively are the edges of the box bounding the area of interest.
In addition to the OSM REST API, a more flexible method to query the data of interest is using the Extended API (XAPI) which enables the users to make queries based on tags and specified areas.

The syntax used by XAPI is similar the syntax covered for the OSM API. The only difference is that the introduces options such as /full are unusable here and are only accessible through the standard API. XAPI by default returns all the nodes that are a member of the queried data object. Syntax example: http://xapi.openstreetmap.org/api/0.6/<type>/

XAPI queries can retrieve data without specifying the feature ID: http://xapi.openstreetmap.org/api/0.6/*[...] where with combination with other filters all the matching primitive data types will be returned.

Similar to the standard API it is also possible to retrieve data based on a bounding box.

Example using filtering for retrieving the data: http://xapi.openstreetmap.org/api/0.6/way[bbox=-25.09,62.94,-12.55,67.42][natural=coastline] retrieves the only the ways within the specified bounding box that are tagged with natural=coastline key-value pair.
OpenStreetMap Data Manipulation

- After retrieving the data, it's usually required to do some processing on the data. In addition to what can be done specifically with programming languages, there packages such as OSMnx for Python that enables specific actions to be done on the data. One of the easiest tools for processing OSM data is Osmosis which is a command line Java application.

- The pluggable nature of the Osmosis makes it easy to chain multiple components to perform a large operation. Osmosis has components for reading/writing databases and files, applying changes to the data, and sorting the data.

- Generating planet dumps from a database and loading them into the database, creating change sets with the help of database history tables or applying change sets to a local dataset, comparing two planet dumps and creating a change set are some examples of what can be done with the help of Osmosis.
There are multiple packages developed to retrieve and process the OSM data. One example of such packages is osmar that is a package developed for R. In the following example the OSM API is selected as the data source and first the node with id 1412998560 (Unicafe example) is extracted and saved into node variable. Next, a bounding box is defined by specifying a center point and the height and width in meters to extract data for the kumpulan kampus. Then the features corresponding to the bus stops and highways are extracted and visualized.

```r
library(osmar)
src <- osmsource_api()
node = get_osm(node(1412998560), source = src)
bounding.box <- center_bbox(24.79476, 60.18271, 500, 500)
data <- get_osm(bounding.box, source = src)
bus.stop.ids <- find(data, node(tags(v %agrep% "busstop")))
highway.ids <- find(data, way(tags(k == "highway")))
highway.ids <- find_down(data, way(hw_ids))
bus.stop <- subset(data, node_ids = bs_ids)
highway <- subset(data, ids = hw_ids)
plot(data)
plot_ways(highway, add = TRUE, col = "chartreuse2", lwd = 2)
plot_nodes(bus.stop, add = TRUE, col = "deepskyblue", lwd = 4)
```
Since the launch of OpenStreetMap in 2004, the service managed to gain more than 3 million registered users and provide a free comprehensive mapping database of the earth. The data can be used for different purposes, and many third party started using the OpenStreetMap as part of their services. These can be broken into the following applications.

- **Maps**
  - Many applications make use of the map provided by OSM because its customizability and free nature. It's possible to get the raw map and change it according to the application needs.

- **Routing**
  - OSM can also be used for routing. One example is the Look and Listen Map which provides routing for blind people.

- **Research**
  - Because of the usage limitations enforced by other map providers, OSM is used in different research studies

- **GIS software**
  - Compared to the proprietary GIS softwares that are closed source, very expensive, and difficult to work with, OpenStreetMap can be used as a free, open, and easy to use GIS software
Applications – Context Inference

- Using the semantic tags used by OSM to give information regarding the points of interests it’s possible to classify the area into different categories based on the POIs.
- In the following example the POIs are extracted and then clustered, then each cluster classified into a category based on the tags, and finally using Voronoi algorithm an adaptive visualization of different neighborhood type is created.
Legal Issues

- OpenStreetMap records all the changes and the user who has the changes mainly for detecting the use of copyrighted data and blocking users that enter false data into the database.
- OSM should be able to show the source of a data when asked specifically about the data that it uses. Therefore, changes only possible by registered users.
- It's not possible to guarantee that the database if free of the copyrighted data.
- Adding data to the database by users automatically makes the data open source as well.
OSM Summary

- OpenStreetMap is a successful case of a Volunteered Geographic Information (VGI) system based on the power of crowdsourcing.
- The data provided by the OSM is completely free to use, and it's possible to redistribute the data to anyone for any purpose.
- You can contribute to the project or edit the data to correct the errors with the provided means.
- It's possible to create custom maps using the rendering applications.
- The data retrieved from the OpenStreetMap can be used in a traditional Geographic Information System (GIS)
**Map Matching**

- Refers to the process of aligning measurements to a logical model of the world
  - Outdoors: snapping measurements onto roads (or other walkable areas)
  - Indoors: aligning measurements with areas that are walkable
    - Remove measurements inside walls or inaccessible areas
- Algorithms for map matching divided into offline and online techniques
  - Offline: process all data after collection (e.g., aligning a trajectory with roads)
  - Online: process data as measurements arrive

Source: https://en.wikipedia.org/wiki/File:Map_Matching_Example_with_GraphHopper.png
Map Matching: Naïve Solution

- Naïve solution: match each point to the closest road segment (or other walkable area)
  - E.g., using Euclidean distance between point and segment
  - Known as *point-to-point matching* (or *point-to-curve* when points compared to segments)
- How to implement efficiently using spatial indexing?
  - Given position error $E$, find all road segments that within $E$ from current position $P$
    - If no solutions, increase $E$ until segment found (so-called radial search)
    - Otherwise return closest match
Map Matching: Naïve Solution

• Problem I: Several points can be equally close
  • Error estimate of position fix can be used to find most likely solution
    – E.g., 2D-Gaussian model (ellipsoid) for GPS
    – Probabilistic point matching forms the basis of more complex techniques

• Problem II: Independent of context
  • Consider the example on the left
  • Which lane / road is the user driving?
Map Matching: Probabilistic Matching

- Basic idea: probability of a point matching a segment is inversely proportional to its distance
- One dimensional case (distance-based):
  - For a given point P, derive a projection x as the point on a road segment that is closest to P
  - Let d = |x – P| denote the distance between x and P
  - Assume d follows a (zero mean) Normal distribution and determine probability of road segment based on this probability
- Two-dimensional case (point-based):
  - Like before, but consider P as the mean µ of a Gaussian distribution that is characterized by covariance matrix Σ
  - Probability of matching depends on shape of covariance matrix and distance between points
  - Probability given by:

\[ \sqrt{|2\pi \Sigma|}^{-1} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right) \]
Consider the following example:

- $x_1 = [8,9]$ and $x_2 = [10, 11.5]$ are projections of measurement $z$ onto two road-segments.
- $z = [9,10]$ is a (noisy) GPS measurement.
- Let $\sum = [2, 1; 1, 2]$ denote the covariance matrix of the GPS measurement (derived from DOP).

Probabilities of measurements given by:

- $P(x_1) = N(x_1|z, \sum) = 0.0658$
- $P(x_2) = N(x_2|z, \sum) = 0.0513$

Segment $x_1$ better point match.
Map Matching: Trajectory-Based Matching

- A fundamental problem with point-to-point matching is that it ignores "context"
  - Consider the example on the left, what is the most likely segment of the middle point?
  - Visually relatively easy to identify, but point-to-point matching would fail
- Trajectory-based map matching fits the entire sequence of points along a movement graph
  - Combine point matching with spatial and temporal constraints
Trajectory-Based Matching: Transition Probability

- Trajectories typically follow one road instead of jumping between multiple roads
  - E.g., in the example on the left, c is likely a better candidate than c' despite having larger distance

- How to enforce?
  - Intuition: shortest path between successive alignments should closely match the distance between successive location measurements
  - Techniques differ in how they compare match between the two distances
Trajectory-Based Matching: Transition Probability

- Notation:
  - Let $x$ denote the previous alignment
  - Let $w(x, c)$ denote the length of the shortest path between $x$ and candidate $c$
  - Let $d(P(t), P(t-1))$ denotes distance between successive measurements

- Different transition costs:
  - $V(x, c) = \frac{d(P(t), P(t-1))}{w(x, c)}$
    - I.e., ratio between (Euclidean) distance of location measurements and the length of the shortest path
  - $V(x, c) = \frac{1}{\beta} \exp\left(-\left|d(P(t), P(t-1)) - w(x, c)\right|\right)$
    - Probability from an exponential distribution using the difference in distance between the two measures
Consider the example on the left

What is the most likely matching for y, 14 or 15?

- Let \( d(x,y) = 1.5 \) (grid cells)

Assume grid distance (Manhattan distance between) grid cells

- Length of 20 \( \rightarrow \) 15 = 1
- Length of 20 \( \rightarrow \) 14 = 2

1. \( V(x,c) = \frac{d(P(t), P(t-1))}{w(x,c)} \)
   - \( V(x,14) = \frac{1.5}{2} \)
   - \( V(c,15) = \frac{1.5}{1} \)

2. \( 1/\beta \exp(-d(P(t),P(t-1)) - w(x,c)) \)
   - \( V(x,14) = 1/\beta \exp(-0.5) = 1/\beta \times 0.61 \)
   - \( V(c,15) = 1/\beta \times \exp(0.5) = 1/\beta \times 1.65 \)

Both select (correctly) cell 15
Map Matching: Additional Constraints

- Velocity or travel time another widely used constraint for map matching
- Given a segment $x$, the average velocity $v(x)$ along the segment is given by $\frac{\text{length}(x)}{\text{diff}(t_1, t_2)}$
  - $\text{length}(x)$ is the physical distance of the road segment
  - $\text{diff}(t_1,t_2)$ denotes the difference of first and last timestamps falling to the segment
- Given a path $p$, the average velocities of the segments along $p$ form a vector $v(p) = (v(1), \ldots, v(n))$
  1. Let $v(\text{traffic})$ denote average velocities derived from speed limits or traffic flow information
     - Similarity of $v(\text{traffic})$ and $v(p)$ can be measured, e.g. using Pearson correlation or cosine similarity
  2. Assume we are given a speed distribution $P(v)$ for different types of roads
     - We can calculate the probability of speeds using this distribution
     - E.g., freeways should have higher average speed than roads in the city center
Modern map matching techniques predominantly assume road network is represented as graph.

Two main matching techniques for trajectory-based map matching:

1. Graph-based techniques
   - Each transition between points associated with cost depending on its match with road network
   - Path/route with smallest cost returned as optimal path (e.g., Dijkstra or A*)

2. Probabilistic techniques
   - Transitions associated with probabilities, depending on likelihood of measurements and overall smoothness of the path
   - Return path/route with highest overall probability (e.g., Viterbi algorithm for HMMs)

Basic principles similar in both, main differences related to how far ahead / behind matching looks at.

Additional techniques, such as voting, can be used to refine matching.
Optimal Path: ST-matching

- ST-matching (spatio-temporal):
  - Candidate road segments correspond to vertices
  - Edges between segments correspond to transitions between vertices
  - Each edge associated with a score, best match given by the path that maximizes the score
  - Point-match probability
  - Transition probability
  - Temporal constraints
  - $F(\text{edge}) = F(\text{temporal}) \times F(\text{transition}) \times F(\text{point})$
  - Optimal path thus corresponds to weighted shortest path
Probabilistic variant of graph matching, basic idea similar to ST-matching

- States of the HMM correspond to true segment of the user
- Observable states of the HMM correspond to noisy location measurements
- Optimal match can be calculated using the Viterbi algorithm
- Scoring based on a combination of measurement probabilities and transition probabilities
Recursive algorithm for finding the most probable path in an HMM $H$ that ends in state $i$ at time $t$ given a sequence of observations $O = O(1), \ldots, O(t)$

- Define $v(i,t) = \max P(q(1)\ldots q(t-1), q(t) = i, O |H)$
  - Probability of most probable path ending in state $i$ at time $t$

- We can calculate $v(j,t)$ recursively using:
  - $v(j,t) = \max [v(i,t-1) P(i,j) e(O(t),j)$

- Probability up to state $t-1$
- Transition from $i$ to $j$
- Observation probability of $O(t)$ in state $j$
HMM Viterbi

- Can be implemented using dynamic programming:

**Viterbi**

**Initialization**

\[
\begin{align*}
T_1(i,1) &= P(1,i)e(O(1);1) \\
T_2(i,1) &= 0
\end{align*}
\]

**Recursion**

\[
\begin{align*}
\text{for } j = 2, \ldots, t \\
T_1(i,j) &= \max(T_1(k,j-1) P(k,i)) e(O(j); i) \\
T_2(i,j) &= \operatorname{argmax}(T_1(k,j-1) P(k,i))
\end{align*}
\]

- Optimal path can be found by backtracking matrix $T_2$
Summary of Map Matching

- Map matching is the process of aligning location measurements with real-world movement constraints
  - Point-based alignment: match one point to a road segment
  - Trajectory-based alignment: match a sequence of points to a road network
- Overall alignment calculated by combining multiple alignment constraints / scores
  - Spatial point match: e.g., Normal distribution
  - Transition match / cost: relationship between point distance and distance of shortest paths
  - Temporal match: match between velocity / travel time and road constraints (if available)
Literature


• Bennet, Jonathan “OpenStreetMap”, Packt Publishing, 2010


• Rabiner, L. R., A tutorial on hidden Markov models and selected applications in speech recognition, Proceedings of the IEEE, 1989, 77(2), 257-286