復旦大學

本科毕业论文

论文题目：隐马尔可夫路径匹配算法实现及可视化

院　　系：计算机科学技术学院

专　　业：计算机科学与技术

姓　　名：陈炜栋

学　　号：11300240057

指导教师：孙未未

2015 年 6 月 23 日

目 录

摘要

今天，轨迹数据正在大量产生。在人们生活中，智能手机和车辆导航等移动设备的各种软件正在扮演更重要的角色。这些设备产生了大量轨迹数据。我们移动数据管理实验室，对轨迹数据的处理和挖掘有深度积累。其中，对于轨迹数据路径匹配问题，代码库实现了用隐马尔可夫模型来找到代表一组带时间戳经纬轨迹序列的路网轨迹。该算法模型能够把轨迹度量噪声大大降低。在地图匹配问题中，数据的采集相对容易获得，但是地图的真实匹配数据的构造，并没有很好的工具来进行使用。使得实验数据的构造变得难以进行，GPS 数据到路网边的映射的工作复杂而枯燥。在以往工作中减少要使用的数据测试集合来减少人工工作量。

本文工作，以 OpenStreetMap 的开源地图数据为基础，将其加工处理，对其边进行分割，加工提取为研究使用的道路数据。接着在这数据的基础上实现一个网页版地图，能够在地图上标注路网节点，进一步计算并可视化最短路径。同时实现了在可视化地图中标注轨迹到轨迹集方法。在以此基础上实现了基于隐马尔可夫模型的地图匹配的算法提供匹配的参照道路。

关键词： 隐马尔可夫模型；地图匹配；数据生成工具；OpenStreetMap；维特比算法;

Abstract

Nowadays, a marvelous number of trace data being generated. In people's lives, smart phones and car navigation and other mobile devices are playing a more important role. These devices produce a large number of trace data. Our mobile data management laboratory has deep accumulation of processing and mining of trace data. As for map matching problem, code library implements Hidden Markov Model to find the network on behalf of a group of track time stamped latitude and longitude locus sequence. The algorithm model can significantly reduce the track noise. In map matching research, the collection of data is relatively easy to obtain, but the ground truth data is hard to get, and there are no good tools to use. We have to do manual matching, which makes experiment difficult. Manual map matching is complicated and boring. We used to reduce the size of dataset to reduce manual workload.

In this work, I process the original data from OpenStreeMap to extract road data that used in the study, then implementing a web version of the map which can mark road network node on the map, visualize the shortest path. In addition, I design an interactive interface to visualize the process of map matching.

Key Words: Hidden Markov Model; Map Matching; Data Simulator Viterbi algorithm OpenStreetMap;

第一章　绪论

伴随着移动设备的普及，轨迹数据正在大量的产生。移动设备具有的定位和无限通信功能，能够记录并向服务器返回它们的位置。根据 Pyramid Research 做的 2011-2015 的 LBS 市场预测，全球 LBS 市场收益预期将从 2010 年的 28 亿美元，升至 2015 年的 100.3 亿美元。相关的研究伴随着关注大量的进行，其中轨迹匹配是其中的一个关键课题，在这方面已经取得了大量的研究成果。

1.1　相关工作

地图匹配问题可描述为，将一系列的 GPS 坐标匹配到道路地图的边中。为了下文表述方便，做如下标识，输入的 GPS 坐标 $z_0, z_1, \ldots, z_n$. 输出的的序列时把计算出的道路网络段，$x_0, x_1, \ldots, x_n$. 现有的地图匹配算法可大致划分为三大类，增量最大权 (incremental max-weight)[2–11], 全局最大权 (global max-weight)[12–16] 和全局几何最大权。增量和全局最大权方法[17,18] 都包含以下步骤:

1. 对于 GPS 取样点 z_i, 首先确定一系列的候选位置 $x_i^0, x_i^1, \ldots$, 每个点映射到道路段中 $y_i^0, y_i^1, \ldots$, 查找范围为 z_i 为中心的一个圆。
2. 为每个候选计算一个权重。
3. 输出似的权重最大的候选序列。

在最后一步，增量最大权每次计算一个取样点，计算基于先前的的几个或多个取样，或者是先前样本的特征。与之相比，全局最大权则一次性考虑整个候选序列的聚合权重。大多数的全局最大权方法采用隐马尔可夫模型 (Hidden Markov Model)，使用维特比 (Viterbi) 算法。本文实现的地图匹配算法，放射概率和转移概率使用全局属性，因此实现了全局最大权算法。增量最大权算法由于其只考虑先

前的结果，适用于在线地图匹配。对于离线地图匹配，使用全局最大权能得到更高的精度。因为未来的结果是保证样点匹配到正确边上的重要因素。

在计算权值的时候，可考虑的因素有很多，这边归纳为七类，1. 点到直线的距离 2. 方向偏转 3. 地图连通性 4. 两个图边段间的距离 5. 两个连续 GPS 样点的方向 6. GPS 样点的数量，7 速度。因素 1 点到直线的距离为最普遍的度量，所有的论文都以这一属性为基础。讨论增量最大权算法，[Whitte00][2] 考虑了地图连通性。[Yang05][3], [Zheng11][9], [Griffin11][10] 考虑了地图段之间的距离。[Blazquez06][4] 考虑了道路的限速。[Li07][5] ,[Zheng11][9], [Mazhelis10][11] 考虑了连续 GPS 的方向。[Mazhelis10][11] 考虑了 GPS 采样的长度。接下来讨论全局最大权算法，[Pink08][14] 和 [Thiagarajan09][15] 考虑了地图连通性。[Lou09][13] 和 [Newson09][16] 考虑了边之间的最短路。

在这两类最大权方法之外，还有一类方法，全局几何地图匹配。这类方法查找地图匹配的几何相似度上最优的路径。这类地图匹配算法 [2,4] 适用 Fréchet 距离准则[19].

1.2 本文工作背景及意义

在研究工作中，我们希望更多地是聚焦的算法层面上，至于实现算法的可视化（轨迹，路径等），是繁琐而又难以复用的。很难交流重用。

1. 可视化工作是必要的，恰当的可视化能让阅读者轻松明白所要展现意图。
2. 由于研究方向细化的原因，没有针对性的可视化工具可以使用。往往只能从很低层级的开发层级开始。造成了学习成本和大量的时间花费。
3. 实验室逐渐形成了自己的路网数据规范，轨迹数据及基础算法库的积累，具备了进一步挖掘轨迹数据价值的先决条件

1.3 本文工作

本文要着手解决的问题是比较开放的，从需求的分析到技术的选择。都是未定的。因此，前期我做了充分的调研，以期能够最高效地解决期望的问题。

我的主要工作包括:

- 分析需求
  1. 相比于现在的只有路网边的可视化,需要一个完整渲染的地图,包含各类建筑设施的渲染。
  2. 根据已有的道路网络数据定义特点,集中在支持可视化轨迹和可视化边集.
  3. 可交互,即不同于当前生成图片的方案。将能够支持动态放大缩小。
- 选择地图数据源选择 OpenStreetMap 作为原始地图数据源。因为
  1. OpenStreetMap 是一个世界地图,开源,每个人都可以贡献自己的经验知识给他。可依据开放许可协议自由使用。
  2. 实验室已有的工作也是基于 OpenStreetMap, 熟悉其数据格式,知道提取挖掘数据的方法。
  3. 有良好的各方面的支持,包括
     (a) 数据格式转换工具
     (b) 渲染地图贴片的服务
     (c) 在这个数据集合上衍生出来的成熟技术框架
- 调研使用技术架构选择 B/S(浏览器/服务器)开发架构
  1. 操作系统无关,在一台 Linux 主机上部署,使用者无需安装,只需要一个浏览器即可
  2. 浏览器的普及和发展迅速,其展示渲染技术成熟

在选择技术方案的过程中重点考虑可拓展性易于拓展新的功能,在道路网络研究领域范畴下一般性,可复用性。

1.4 本文结构

绪论对地图匹配的先前的工作进行总结。分析了本文要解决的问题,确定了数据源和技术的大的框架。第二章详细描述应用实现功能的使用流程,从拓展性和复用性的角度对功能进行分类说明。第三章从实现的技术细节出发, 对整体架构进行描画。框架围绕数据,前端可视化模块和基础模块。第四章从行文的完整

性出发，简单描述路网和轨迹匹配相关的概念和定义。第五章详细罗列了实现轨迹匹配等功能中实现的算法。最后对本次毕设工作进行了总结和展望，指出了自己的不足。以及可改进的方向。

第二章　轨迹展示应用的功能

本章节，从应用要关注的问题出发，进行分析，详细描述我期望达到的目标，对地图轨迹展示应用的功能进行阐述。

2.1　应用要解决问题及分析

随着研究的深入，一般化的工具提供的功能不能满足具体的需求。在道路网络中的研究中，我们实验室形成了固定的数据格式。在此基础上，边和点的定义得到了统一。有了在数据基础上进行加工可视化的先决条件。

本应用通过选取易拓展的框架，及宽松的数据格式，对接口进行一般化定义。首先规划了基本必须的基本功能，并对这部分核心的代码进行详尽的单元测试，以期为后续工作建立坚实的基础。在此基础上实现了拓展功能，其中地图匹配的数据生成工作，填补了没有好工具来帮助生成道路匹配过程中真实数据生成的困难。还包括了，上传点和边文件可视化轨迹，及选定起点终点进行路径规划的功能。这些工作建立在基础功能至上，希望能抛砖引玉，能帮助后来者在这个应用框架的基础上快速拓展。

基本功能包括：

1. 可交互的地图中画出点和边。
2. OSM 的 XML 地图数据处理成路网数据。
3. 映射任选的地图坐标，到距离最近的路网节点中。
4. 选取出最近的 k 个，距离较近的路网边。
5. 投射地图点，到某一路网边上

在基本功能包括了 1. 可视化点边。2. 数据的生成处理加工。3. 把连续的地

图点映射到离散的地图中的能力。为了完成这些基本功能，我对地图建立 Kd-Tree 和网格索引。Kd-Tree 用以最近点查询，网格索引用以快速求路网中的 k 近边。

拓展功能包括:

1. 在地图中选取两个点作为起点和终点，进行路径规划
2. 在地图中选取若干节点，形成轨迹，接下来用基于隐马尔可夫模型的道路匹配算法提供一条参考路径。最后对每一个节点，应用提供若干条根据距离由近到远的候选边。依次选择节点的真实匹配边。最终生成点和真实匹配边数据文件。
3. 上传点和边的集合，可视化点和边。

我在路网这个具体的研究方向中选择了一般的数据格式，节点边的映射，投射是一般的，因而针对更加具体的应用可在这基础上进行。在路网算法的研究过程中，可能有各式各样的研究目的，但不变的是点和边。比如，在热门路径的研究中，最终描述热门路径必然可以转化为，对路网点和边的描述，不同的是对道路热度的描述，这时，基本功能得到很好的复用。

2.2 地图匹配

本功能综合运用了各个基本功能，实现了交互良好的可视化地图匹配的人工标注应用。其中，基于隐马尔可夫模型的道路匹配算法中复用了基本功能路网 k 近边。

操作流程:

1. 打开 mapmatching.html
2. 在地图中点击若干节点，得到一个轨迹序列 $(lat_0, lon_0), (lat_1, lon_1), \ldots, (lat_n, lon_n)$
3. 点击左上角第一个按钮，进行基于隐马尔可夫模型的道路匹配算法，地图上会画出一条参考的地图匹配路径(可选)
4. 点击左上角第二个按钮，开始匹配过程。匹配过程按照节点顺序依次进行，每个节点有对应的若干条按照距离从近到远的边供选择，鼠标移动到具体的边上面，该边会被变色加粗高亮出来。点击高亮边就完成了对节点候选边的选择。依次选择所有节点的候选边。

5. 点击右上角第三个按钮，将会生成轨迹点文件及人工标注的边文件。

至此，我们得到了一条模拟的轨迹及对它进行人工标注的方法。

2.3 轨迹展示

本小节描述了我在开发应用的过程中，开发的拓展功能。其中最短路径规划基本功能的映射地图坐标到地图中，在此基础上，在道路网络中求给定起点和终点的最短路径。上传可视化点边集合简单应用了所选框架提供的点边的可视化功能，本功能具有较强的实用性。点 ID 集合和边 ID 集合极其一般，换句话说，在各式各样的研究目的中，生成点 ID 集合和边 ID 集合都极为方便。对于 GeoLife 数据集，作为被重度使用的数据集合，对其进行可视化，方便对其数据属性的观察。

2.3.1 最短路径规划

1. 打开 trajectory.html
2. 依次在地图上点击两个点，每次选取点，会进行一次地图坐标到道路节点的映射。得到道路中 $sourceID$ 和 $targetID$.
3. 进行最短路径的规划，在地图中以粗线描画
4. 可重复再地图点击点，两个点为一对起始点终止点对。(可选)

2.3.2 上传可视化点边集合

1. 选择要上传的点 (边) 文件
2. 点选上传按钮
3. 点选展示轨迹按钮
4. 点击清除图层按钮可清除已经画在地图画布上的轨迹 (可选)

2.3.3 查看 MSRA-GeoLife 收集的地图数据集合

1. 打开 trajectory.html
2. 再级连下拉框中选择对应数据收集人和对应手机时间的轨迹

3. 点击原轨迹按钮，在地图上可视化出初始的道路轨迹
4. 点击匹配边按钮，在地图上可视化出进行基于隐马尔可夫模型（离线）地图匹配的边集合的轨迹。
5. 点击匹配点按钮，在地图上可视化出将点映射到边上的轨迹。

2.4 本章小结

考虑需求的各式多样性，我从整体出发，搭建了一个从可视化，到基本核心功能单元，到内部的数据松耦合的系统。在基础功能之上，通过实现了几个具体的应用功能，展示框架的可拓展性。下一小节，从技术层面出发，具体阐述应用的结构。

第三章　轨迹展示应用实现

本节从实现的技术细节出发，对整体架构进行描画。主要包括五个模块，包括: 前端可视化模块，Nodejs 后端模块，Java Web 服务后端模块，基础功能模块，以及数据模块。

3.1　总体结构

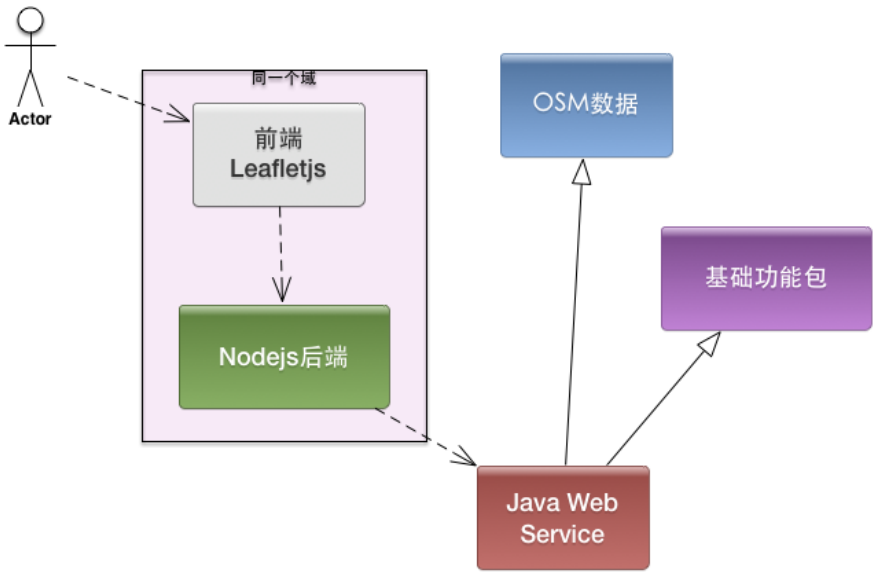

图 3-1: 应用整体架构

围绕基础功能模块，利用 Leafletjs 框架丰富的特性集合，Nodejs 转发请求来解决跨域请求问题，Java Web 服务端来承担数据处理和计算，实现了一个松耦合

的系统。通过这个系统可快速添加新的算法，完成新的可视化需求。这样的架构，兼顾了工程开发效率和算法运行效率。

3.2 前端可视化模块

由于决定选择 B/S 架构，前端技术的使用就确定下来了。在经过充分的查资料之后，决定使用 Leaflet.js 框架。它是一个开源的可交互地图 JavaScript 库。只需要经过简单的配置，就可以获得产品级的地图展现。

- 添加地图的贴片服务，下面代码片段展示了将以 OSM 的贴片服务做为贴片服务器。

```
1 // add an OpenStreetMap tile layer
2 L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
3 attribution: '&copy; <a href="http://osm.org/copyright">
     OpenStreetMap</a> contributors'
4 }).addTo(map);
```

- 设置地图的视点，下面代码片段展示了将设置背景为地图中心的视点。

```
1 // create a map in the "map" div, set the view to a given
     place and zoom
2 var map = L.map('map').setView([39.9067,116.3978], 11);
```

- 给出坐标集合下面代码片段展示了将折线线段画出到地图上

```
1 // create a red polyline from an array of LatLng points
2 var polyline = L.polyline(latlngs, {color: 'red'}).addTo(map)
     ;
```

除了上面的基本配置，在实现地图匹配的过程中，为了做到动画效果。鼠标停留高亮选中边的效果，还用到了 GeoJSON。对事件 mouseover 进行监听，对颜色，线条宽度进行相应的修改。

```
1 edgesLayer = L.geoJson(geoJson, {
2     style : style,
3     onEachFeature, onEachFeature,
4
```

```
});

map.addLayer(edgesLayer);

...
function onEachFeature(feature, layer) {
    layer.on({
        mouseover : highlightFeature,
        mouseout : mouseoutHandler,
        click : clickHandler
    });
};
...
}
```

leaflet 提供了多功能

1. 各种图层: 包括可定制的标记点，可替换的贴片图层，对多边形，折线，圆形，矩形等有丰富的支持。
2. 跨平台的交互特性，特别的在桌面浏览器中，支持滚轮滑动和双击的缩放，放大缩小到特定区域，还支持通过键盘的控制 (+/-). 在移动端，支持多点触碰和的轻触缩放。
3. 定义了各类事件，方便开发具有交互式的应用，地图匹配功能中的鼠标停留高亮就是在这基础上开发的。

这是一个产品级的前端地图交互框架，选择它的重要原因是因为其完善的文档示例。在此基础上，前端的展示工作变得非常简单，通过和后端交互点集合 $(lat_0, lon_0), (lat_1, lon_1), \ldots, (lat_n, lon_n)$ 就能够满足简单的展示需求。对于更加进一步的需求，比如地图匹配的动态交互功能，通过查阅文档及示例也能够比较快速找到解决方案。至于更加复杂的特性，社区提供各式各样的第三方插件。前端可视化模块建立在一个拥有丰富特性，易拓展的框架之上。经过对文档阅读和开发地图匹配功能过程中的实践，将地图坐标和后端的请求的交互，可满足我们可视化道路网络的需求。

3.3 后端模块

Nodejs 和 Java 各有优势

- 使用 Java，因为为地图构建 Kd-Tree 索引和最短路径规划，网格索引，都需要大量的运算。使用 Nodejs 会有性能瓶颈问题。
- 使用 Nodejs 因为 Tomcat 服务不支持前端代码的热插拔（在对前端代码进行小幅度修改后），同时其启动周期慢，不能快速响应修改，前后端都部署到 Tomcat 上会造成开发的不便。而 Nodejs 没有这样的问题。

在本应用采用前后端分离。后端 Java 用来处理有大量运算的请求。前端自带 Nodejs 服务器，用来处理静态文件的直接返回。

3.3.1 Nodejs 模块

现在 Web 开发团队主流方法也是会在前端代码上自己加一层 Nodejs 后端后来进行请求转发，用来 Mock 前端 js 的返回来快速开发。

本应用的 Nodejs 模块包含下列功能:

- 转发请求，用来避免跨域请求。Nodejs 启动的服务器的前端，发送请求道 Nodejs 后端，这时，浏览器发送的请求道 Nodejs 后端，两者处于同一域名下。Nodejs 收到浏览器发来的请求，重新发送给 Java 后端，因为不是浏览器发送，也就没有了浏览器防止安全问题而限制的跨域请求的问题。Java 后端收到请求后，计算出结果返回 Nodejs 后端。Nodejs 后端收到返回结果，再异步地将数据送回浏览器，从而完成一次请求。
- 接收上传的文件，Nodejs 后端讲浏览器上传的文件进行保存，并开发了读取文件进行显示轨迹的接口。基于这样的考虑，Nodejs 不进行透明转发。在实现上传点和边集合功能时，并没有计算量，因而透明转发给 Java 后端，只带了额外的传输开销，也并没有得到计算效率上的提升。

3.3.2 Java Web 服务模块

以 Java 作为 Web 服务端，是对算法效率和工程效率的综合考虑。在本模块中，采用 Jersey + Spring 的框架结构。用 Spring 的做依赖注入，用 Jersey 实现 RESTful 接口。在附录可找到 RESTful 接口文档。

本应用的 Java Web 服务模块包含下列功能：

- 依赖基础功能模块，读取数据，在后端建立地图的相关索引。
- 接收 Nodejs 端发来的请求，Jersey 对路径参数和 JSON 参数等进行解析以获取到请求的数据。
- 进行计算和处理得到结果，Jersey 对结果进行处理包装，返回 JSON 数据给 Nodejs 段。

抽象出基础服务模块后，Java Web 后端成为很简单的一层，在后续的功能拓展时，只需依模仿便可轻松添加接口。

3.4 基础功能模块

核心的算法代码。这部分代码对复用性的要求较高，在设计接口是进行了慎重的思考，接口开发后，对其进行单元测试。使用单元测试框架 TestNG, 在附录中可找到测试用例文档。所实现代码的核心算法的描述在接下来两节中进行细致的阐述。

本应用的基础功能模块现阶段包含下列功能:

- 数据处理，把从 OpenStreetMap 中下载的 XML 文件，加工处理为研究中使用道路网络数据格式。中间包含处理细节在下小节进行阐述。
- 求两线交点及点到边的投射点。
- 对道路网络建立 Kd-Tree 索引，满足最近邻的查询。
- A-Star 最短路算法，在道路网络中求最短路。
- 对道路网络建立网格索引，满足球 k 近边的查询。
- 给定轨迹，用隐马尔可夫模型地图匹配方法，匹配到边集合上。

基础模块是本文工作的重点，把这一模块独立出来时因为不同于 web 服务代码，这部分功能具有对正确性复用性的高度要求。为此需要对其进行详细的开发

测试和代码注释。另一方面，从降低整体系统耦合性的角度考虑，设计成独立的模块，有助在脱离应用框架来使用它。按照把这部分代码设计成代码库的方法在实现它。

3.5 数据模块

下载目标城市的的 OSM 数据[20]，在基础功能包中，提供了 OSMProcess.java 用以数据格式的转化

3.5.1 OSM 数据格式

```
}
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
 <bounds minlat="54.0889580" minlon="12.2487570" maxlat="
    54.0913900" maxlon="12.2524800"/>
 <node id="298884269" lat="54.0901746" lon="12.2482632" user="
    SvenHRO" uid="46882"
 visible="true" version="1" changeset="676636" timestamp="
    2008-09-21T21:37:45Z"/>
 <node id="261728686" lat="54.0906309" lon="12.2441924" user="
    PikoWinter" uid="36744"
 visible="true" version="1" changeset="323878" timestamp="
    2008-05-03T13:39:23Z"/>
 <node id="1831881213" version="1" changeset="12370172" lat="
    54.0900666" lon="12.2539381"
 user="lafkor" uid="75625" visible="true" timestamp="2012-07-20
    T09:43:19Z">
  <tag k="name" v="Neu Broderstorf"/>
  <tag k="traffic_sign" v="city_limit"/>
 </node>
 ...
 <node id="298884272" lat="54.0901447" lon="12.2516513" user="
    SvenHRO" uid="46882"
```

```
16  visible="true" version="1" changeset="676636" timestamp="
       2008-09-21T21:37:45Z"/>
17  <way id="26659127" user="Masch" uid="55988" visible="true"
       version="5" changeset="4142606"
18   timestamp="2010-03-16T11:47:08Z">
19   <nd ref="292403538"/>
20   <nd ref="298884289"/>
21   ...
22   <nd ref="261728686"/>
23   <tag k="highway" v="unclassified"/>
24   <tag k="name" v="Pastower Straße"/>
25  </way>
26  <relation id="56688" user="kmvar" uid="56190" visible="true"
       version="28"
27  changeset="6947637" timestamp="2011-01-12T14:23:49Z">
28   <member type="node" ref="294942404" role=""/>
29   ...
30   <member type="node" ref="364933006" role=""/>
31   <member type="way" ref="4579143" role=""/>
32   ...
33   <member type="node" ref="249673494" role=""/>
34   <tag k="name" v="Küstenbus Linie 123"/>
35   <tag k="network" v="VVW"/>
36   <tag k="operator" v="Regionalverkehr Küste"/>
37   <tag k="ref" v="123"/>
38   <tag k="route" v="bus"/>
39   <tag k="type" v="route"/>
40  </relation>
41  ...
42 </osm>
```

- 在这里 *bounds* 标签包含了 $minlat, minlon, maxlat, maxlon$, 分别表示最小纬度,最小经度,最大纬度,最大经度。刻画了提取的城市数据范围.
- *node* 标签包含了节点 id, lat, lon, 分别表示节点 id, 纬度,经度等,这刻画了节点信息。

- way 标签中包含了,自标签多个 nd, ref 为节点 ID,这刻画了整条轨迹.
- tag 中的 $k =$ "$highway$" 的道路是我们关心的道路
- tag 中 $k =$ "$oneway$" 的道路表示这是一条单向路
- 其他的标签在我们的应用中不需要考虑

3.5.2 目标数据即转化数据

- 道路节点格式

  $i\ latitude_i\ lontitude_i$

  0 39.9061898 116.3894982

  1 39.8987502 116.3898158

  2 39.8988392 116.3934744

  ...

  - 第一个字段表示顶点 ID
  - 第二个字段表示顶点的纬度坐标
  - 第三个字段表示顶点的经度坐标
- 道路边格式

  $i\ sId\ eId\ latitude_1\ lontitude_1\ latitude_2\ lontitude_2\ \ldots\ latitude_n\ lontitude_n$

  0 0 25365 2 39.9061898 116.3894982 39.9059947 116.389502

  1 25365 51418 2 39.9059947 116.389502 39.9017383 116.389744

  2 51418 130 3 39.9017383 116.389744 39.9015146 ...

  ...

  - 第一个字段表示边的 ID
  - 第二个字段表示边的起点的 ID
  - 第三个字段表示边的终点的 ID
  - 第四个字段表示边中间经过的坐标数
  - 第五个字段表示第一个边经过的道路节点的纬度
  - 第六个字段表示第一个边经过的道路节点的经度
  - 第七个字段表示第二个边经过的道路节点的纬度

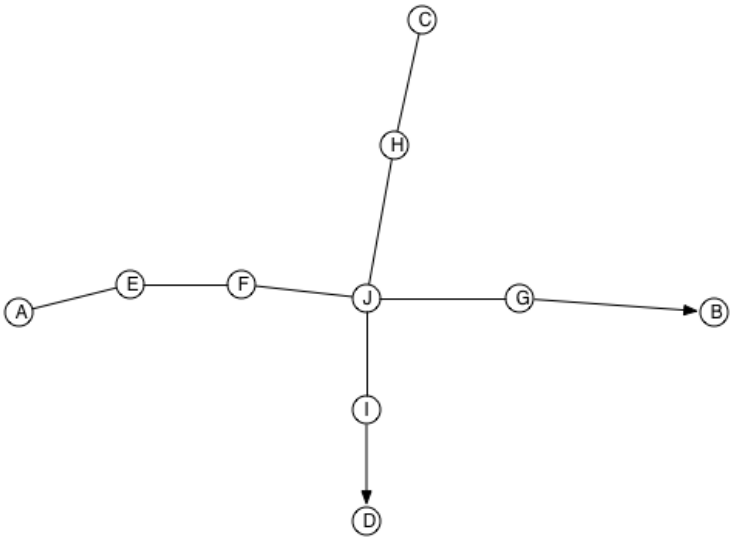

图 3-2: 道路相交分割

– 第八个字段表示第二个边经过的道路节点的经度
– ... 以此类推

3.5.3 数据的初步提取

$map\_new\_in\ 2014-06-26.xml$ 北京地图，经初步提取，raw 数据节点数有 397992，边数有 40977。所谓初步提取就是以 XML 文件中的一个 $<node/>$ 标签作为一个节点，一个 $<way/>$ 标签作为一条边。

这时候有几个问题：

1. 节点数太多，而且存在非常多的度数为 2 的点，这些点，起刻画作用，在道路网络中实施算法时，并不需要考虑这些度数为 2 的节点.
2. 边不能很好地连接节点，因为对于一条 way 来说，如果只考虑起起点和终点，中间的刻画轨迹的点忽略掉，那么本来连接到的点没有连接到。详细见图例子

3.5.4 数据的进一步分析提取

这个时候边集合有 $(A,B),(C,D)$，点集合有 A,B,C,D,E,F,G,H,I，在这个图里，查询 $A \to C$ 的距离，会查询不到。很显然，D 作为交叉路口，应当把 $(A,B),(C,D)$ 两条边分割成，$(A,J),(J,B),(C,J),(J,D)$ 四条边。观察到点 E,F,G,H,I 为道路内点，在地图中用来刻画轨迹，在道路网络算法中，这些节点意义不大。故而应当在数据提取时，筛除这些节点。

总结：

1. 在处理数据的时候应当去处度数为 2 的点
2. 为了提取出的路网图的的连通性，在 T 字路口和十字路口处应该把边进行恰当的分割。

3.5.5 实现细节

在实现时，由于 XML 文件比较大，应当使用流式的 XML 处理方法。[21]

在 JAXP(Java API for XML Processing) 中有三种处理方式，DOM, SAX, StAX, 其中 DOM 接口最简单，最易操作。但是效率也最低，因为它需要把整个 XML 文件读入内存中进行处理。SAX 效率最高，它从文件头读到文件尾部，通过调用回调函数来对读入的数据进行处理。它把数据 Push 给回调函数。StAX 在两者之间，首先，它也是从文件头读到文件尾部一次。但是它提供了 PULL 数据的机制，像文件流一样，需要时去流中取数据。在处理复杂的文件时，PULL 的机制能简化代码量。

我在选择的时候考虑，易编写性和效率，选择 StAX 接口来进行编程。

3.6 本章小结

本小节，阐述了所预期要完成的功能，并对要采用的工程技术进行分析选择。决定采用前端 Html5+CSS3+JavaScript, 后端混合 Nodejs Server 和 Java 的组合。进一步，明确并罗列了应用中接触到的数据及格式。最后从选择的技术出发，定义了简单明了的 Restful 的接口。

第四章　应用的核心算法

4.1　算法的背景知识

为了完整性，本节简单描述路网和轨迹匹配相关的概念和定义。

4.1.1　道路网络

我们把道路网络 G 定义为一个节点结合 N 和一个边集合 E 组成的带权图，即 $G = (N, E)$。一个节点 $n \in N$ 代表一个道路路口，一条边 $(u, v) \in E$ 代表连接节点 u 和 v 的路段。

4.1.2　最短路径

$w(u, v)$ 表示相应边的权值，该权值使用路段长度或者是通过时间，为简单起见，本文使用路段长度为权值并且假设所有的长度都是正数。一条路径 $P(s, t)$ 代表连接节点的一系列边的集合，其长度 $|P(s, t) = \sum_{(u,v)\in P(s,t)} w(u, v)|$。在所有连接节点 s 和 t 的最短路径 $SP(u, v)$。最短路径规划返回了最短路径。

4.1.3　轨迹数据

时空轨迹是记录移动对象的位置和时间的序列。理论上讲，时空轨迹是连续的，但出于硬件限制和存储开销的现实考虑，人们通常对移动对象的运动过程进行采样，采样的信息主要包括地理位置信息、移动对象的属性信息以及时间信息，属性信息主要包括采样点速度、方向及车辆相关属性。从而以离散的方式，用一组

时空轨迹点序列表示轨迹[22]

$$T = \{(x_1, y_1, t_1), (x_2, y_2, t_2), (x_3, y_3, t_3), \ldots, (x_n, y_n, t_n)\}$$

4.1.4 轨迹匹配问题

轨迹匹配是一个确定采集的 GPS 数据点属于的道路网络的过程。用一个例子阐述，下图中有三个标注的黑点，轨迹匹配问题时去找到车辆在哪条车道上行驶。最显然的方法是直接把轨迹点匹配到距离最近的道路。因为传感器测量误差，这样简单的策略会得到错误的结果。如图示，实际的道路很显然是浅灰色标柱出的道路。但时如果按照匹配距离轨迹点最近路径的办法，第二个和第三个点会匹配错误。考虑到轨迹数据有时间性，节点一二三按照顺序被采集。可以挖掘这一性质，使用隐马尔可夫模型，避免简单匹配最近道路边带来的错误。

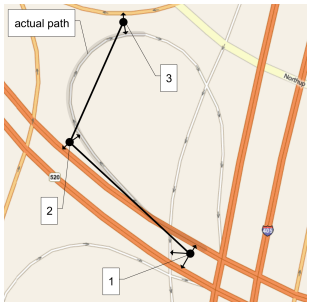

图 4-1: 轨迹匹配

本 Web 应用支持了如下特性

1. 在地图中选择任意点
2. 在上面特性的基础，任意选择两个点，提供最短路径规划
3. 处理轨迹数据，展示地图匹配后的可视化图像

为了支持上述特性，需要 Kd-Tree, A-Star 算法, 和基于 Viterbi 算法的隐马尔模型的道路匹配算法。同时为了能支持高效的查询，在道路网络中建立网格索引用以快速解决 k 近边查询.

4.2 Kd-Tree 查找最近点

Kd-Tree 是一种分割 k 纬数据空间的数据结构。主要用于多维空间关键数据的搜索，如范围搜索和最近邻搜索。这本文中，我使用 $k = 2$ 的情形。在地图应用中，用户选择地图中的任意点，我们需要把用户选择的点映射到地图数据中的点中，从而能够根据已有的数据提供查询服务 (比如最短路径规划)。

4.2.1 Kd-Tree 的构建

首先我们需要对地图的定点集合构造树。本文应用了 $k = 2$ 的情形（地图在二维平面上）。首先选出中位数，度量依次按照纬度评判，将顶点集合平分为两份，如此构造出一棵平衡的二叉树。算法 1 描述了具体流程。

Algorithm 1 Tree Construction

```
1: function KDTREE(pointList, depth)
2:     axis ← depth mod k                    ▷ choose dimension, we use k = 2
3:     medium ← select median by axis from pointList
4:     node.location ← medium
5:     node.leftChild ← kdtree(points in pointList before medium, depth + 1)
6:     node.rightChild ← kdtree(points in pointList after medium, depth + 1)
7:     return node
8: end function
```

以一个简单直观的实例来介绍 Kd-Tree 算法。假设有 6 个二维数据点 (2,3),(5,4),(9,6),(4,7),(8,1),(7,2)，数据点位于二维空间内。插入第一个结点 (2,3)，树为空，(2,3) 成为根节点。插入第二个节点 (5,4) 时，根节点为以 Y 坐标为判据，节点插入右子树，又子树为空树 (5,4) 成为其根，成为 (2,3) 的右孩子。插入第三个节点 (9,6), Y 坐标比根节点大，插入右子树，和极点（5，4）比较，此时以 X 坐标为判据，节点插入右子树，(5,4) 的右子树为空，(9,6) 成为 (5,4) 的右子树。以此类推，可构造出如右下图的树，左下图为平面中可视化的树，其中红线表示该节点以 Y 坐标作为判据，绿线表示以 X 坐标作为判据。

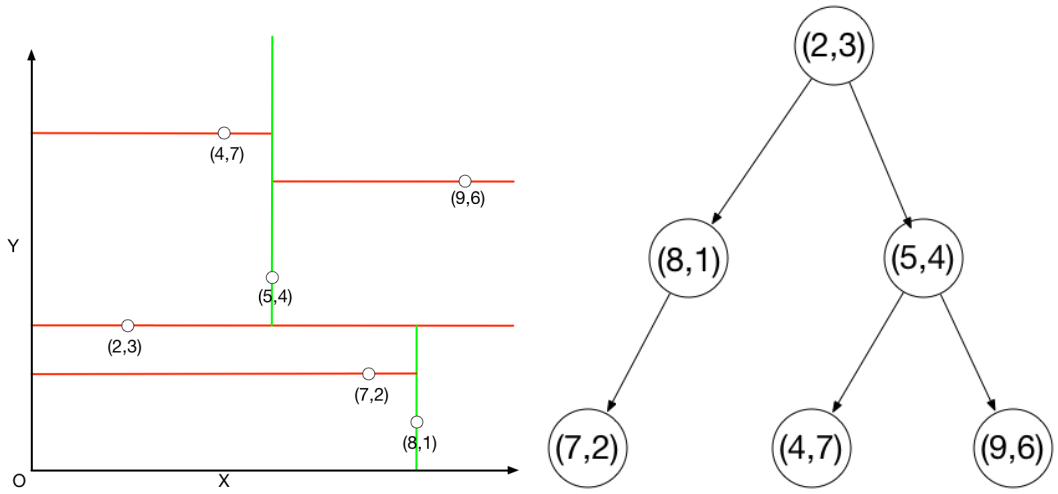

图 4-2: KdTree 构造示意图

4.2.2 查询最近点

在前小节得到的树的基础上，我们可进行高效的最近邻查询。首先注意到每一个子树时包含在一个矩形框内。我们定义一个点到矩形框的距离为矩形框内任意点到查询点的最短距离。如此我们可以得到一个最优性枝 $minmumDistance < distanceToMBR$ 则跳过该子树。算法 2 描述了具体流程。

4.3 A* 路网最短路

A-Star 算法是一种静态路网中求解最短路最有效的直接搜索方法。本文实现该算法为地图应用提供了距离最短最优的路径规划方案。像所有的启发式搜索一样，路网 A-Star 算法寻找最有可能得到结果的路径。A-Star 算法和纯粹的贪心算法（Dijstra) 不同的是，它不仅考虑了已经遍历的路径长度，它还把估计值 $g(x)$ 考虑进来。而不只是基于已经计算出来的值进行决策。

从起始点开始，首先维护一个待松弛节点的优先队列，成为待拓展集合，对于节点 x,$f(x)$ 越小，优先级越高。在算法的每一步，取出优先级最高的节点，松弛并拓展其邻居节点。算法不断迭代直到 $goal$ 成为优先级最高的节点。

Algorithm 2 Nearest Point Search

```
1: function NEAREST(tree, queryPoint)
2:     distanceToMBR ← minumum distance to tree's MBR
3:                                          ▷ MBR Minimum Bound Rectangle
4:     if distanceToMBR < minumDistance then
5:         return
6:     end if
7:     δ ← ||queryPoint, tree.location||
8:     minumDistance ← min(minumDistance, δ )
9:     nearest(tree.leftNode, queryPoint)
10:    nearest(tree.rightNode, queryPoint)
11:    return node
12: end function
```

4.4 网格划分路网求近边集合

在下节地图匹配算法 Viterbi 算法中，我们需要在路网中支持快速高效的 k 近边查询。

我们在对地图进行网格划分，为路网点和边进行网格区域的划分。在查询某个坐标的 k 近边时，首先查询根据坐标定位到其所属的网格，然后向外迭代查找 (9 宫格，25 宫格，以此类推) 直到找到满足条件的近邻。

路网中 k 近边定义为，距离采用上点到边的距离定义，该函数返回，路网中距离查询点前 k 近的路网边。

如上图所示，我们将地图划分成了 5×5 的网格。

图 4-3 中查询点距离各边的距离为图中虚线所示，其中，最近的边是边 3 而不是边 1

注意到本图的查询点落在了网格 8 中，如果我们查询网格一开始用的大小是 1 的话，即只遍历落在网格 8 中的边，那么会得出最近边时 1 的结论，实际上的最近边是边 3. 网格划分求 k 近边不是精确的. 为了处理大多数的情况，在查询网格

Algorithm 3 Road Network A Star search shortest path

```
1: function A*(start, goal)
2:     closedset ← the empty set                ▷ The set of nodes already evaluated
3:     openset ← start                          ▷ The set of tentative nodes to be evaliated
4:     came_from ← the empty map                ▷ Cost from start along best konwn path
5:     g[start] ← 0
6:     f[start] ← g[start] + heuristic_cost_estimate(start, goal)
7:     while openset ≠ ∅ do
8:         current ← the node in openset having the lowest f[] value
9:         if current = goal then
10:            reconstruct_path(came_from, goal)
11:        end if
12:        remove current from openset
13:        add current to closedset
14:        for each neighbor in neighbor_nodescurrent do
15:            if neighbour ∈ openset then
16:                continue
17:            end if
18:            tentative_g ← g[current] + ||current, neightbor||
19:            if neighbor not in openset or tentative_g < g[neighbor] then
20:                came_from[neighbor] ← current
21:                g[neighbor] ← tentative_g
22:                f[neighbor] ← g[neighbor +‖neighbor, gaol‖
23:                if neighbor not in openset then
24:                    add neighbor to openset
25:                end if
26:            end if
27:        end for
28:    end while
29:    return failure
30: end function
```

Algorithm 4 Construct Road Network Path

```
1: function RECONSTRUCT_PATH(came_from, current)
2:     total_path ← [current]
3:     while current in came_from do
4:         current ← came_from[current]
5:         total_path.append(current)
6:     end while
7:     return total_path
8: end function
```

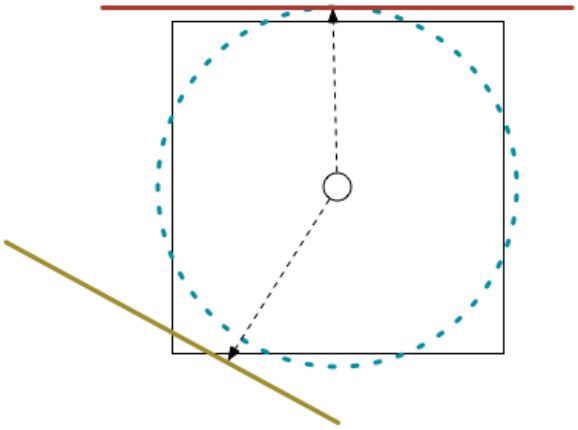

图 4-3: 近边网格失效特例图

时，应查找网格的 9 宫格。这样，我门对网格划分得足够细致，尽量不出现网格为空的情况。那么就可以处理大多数的情况了。

上图，正确的最近边是红边，但是利用网格划分的方法会求出黄边，本测旨在说明这一情况。

通过对网格划分，在查询运算时只考虑了在离查询点接近的网格内的边，可以认定对地图的网格划分可以损失极小的正确率的情况下，极大的提高效率。此外如果要保证绝对的正确性，可用当前的找到的最近距离为上界，考虑在所有界内的网格。

4.5 隐马尔可夫模型轨迹匹配

本小节,描述了地图应用所实现的匹配轨迹的算法[16]。先简单介绍了地图匹配的定义和隐马尔可夫模型,再分析轨迹匹配问题。对模型进行转化,细微处修改定义使其适用于道路网络数据量大的特点。最终用 Viterbi 算法实现该算法。

4.5.1 简要地图匹配定义

地图匹配的定义为将一系列的 GPS 坐标映射到道路网络中。将 GPS 坐标表示为 $z_0, z_1, z_2...z_n$, 将地图匹配的对应坐标的边表示为 $e_0, e_1, e_2, ...e_n$, 将地图匹配对应的边上的点定义为 $x_0, x_1, x_2...x_n$

4.5.2 隐马尔可夫模型

隐马尔可夫模型(Hidden Markov Model,HMM)是统计模型,它用来描述一个含有隐含未知参数的马尔可夫过程。其难点是从可观察的参数中确定该过程的隐含参数。

在正常的马尔可夫模型中,状态对于观察者来说是直接可见的。这样状态的转换概率便是全部的参数。而在隐马尔可夫模型中状态并不是直接可见的,但受状态影响的某些变量则是可见的。每一个状态在可能输出的符号上都有一概率分布。因此输出符号的序列能够透露出状态序列的一些信息。

有四个要素:

- x 隐含状态, 在地图匹配问题下,对应 e 即将要匹配的边集合
- y 可观察输出, 在地图匹配问题下,对应 x 即得到的 GPS 轨迹坐标
- a 转换概率,在地图匹配问题下,对应从 $e_i \rightarrow e_j$ 的概率,在这里表现为边与边之间的连通性和距离.
- b 输出概率,在地图匹配问题下,对应从路网边到达所获取的 GPS 坐标之间的距离.

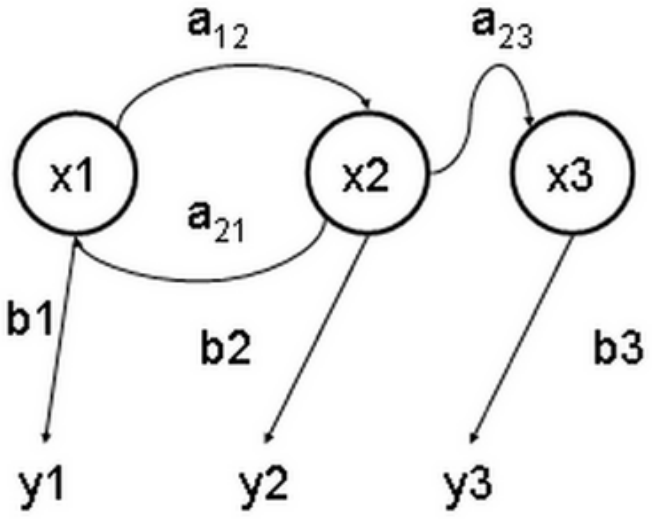

图 4-4: 隐马尔可夫模型示例

4.5.3 维特比算法[1]

应用维特比算法求解隐马尔可夫模型。维特比算法说白了就是动态规划实现最短路径，假设上图每一列分别有 $n_1, n_2, \ldots, n_n$ 个节点，如果不使用动态规划的话，那么计算复杂度就是 $O(n_1 * n_2 * \ldots * n_n)$。而维特比算法的精髓就是，既然知道到第 i 列所有节点 $X_{ij} j = 1, 2, 3$ 的最短路径，那么到第 $i+1$ 列节点的最短路径就等于到第 i 列 j 个节点的最短路径为第 i 列 j 个节点到第 $i+1$ 列各个节点的距离的最小值。

4.6 本章小结

本章详细描述了在应用中所使用的核心算法，为道路地图建立 Kd-Tree 和网格索引，来支持转化离散化道路坐标和快速查找 k 近边的需求。还阐述了路径规划用到的 A-Star 算法和维特比算法实现的基于隐马尔可夫模型轨迹匹配算法。

Algorithm 5 Construct Road Network Path

function VITERBI(O, S, π, Y, A, B)

Output:

The most likely hidden state sequence $X = \{x_1, x_2, \ldots, x_T\}$

  for each state s_i **do**

    $T1[i, 1] \leftarrow \pi_i \cdot B_{iy_1}$

    $T2[i, 1] \leftarrow 0$

  end for

  for i $\leftarrow$ 2,3, ..., T **do**

    for each state s_j **do**

      $T1[j, i] \leftarrow \max_k (T_1[k, i-1] \cdot A_{kj} \cdot B_{jy_i})$

      $T2[j, i] \leftarrow \arg\max_k (T_1[k, i-1] \cdot A_{kj} \cdot B_{jy_i})$

    end for

  end for

  $z_T \leftarrow \arg\max_k (T_1[k, T])$

  $x_T \leftarrow s_{z_T}$

  for i $\leftarrow T, T-1, \ldots, 2$ **do**

    $z_{i-1} \leftarrow T2[zi, i]$

    $x_{i-1} \leftarrow s_{z_{i-1}}$

  end for

  return X

end function

第五章　结论和展望

经过一个多月的努力，得到了初步满足需要的地图展示需求的应用。基本达到一开始定下的目标。做成了一个操作简单的网页应用，在其内核算法方面做了以下工作，利用 Kd-Tree 匹配最近点，使用 A-Star 算法最短路径规划，对道路进行网络网格划分来提高近边集合的查询。实现了 Viterbi 算法的隐马尔可夫模型。从分析需求到，完成需求，实现算法代码的过程中，我收获良多。

受制于时间等客观因素，应用的功能还显得简单单薄，界面操作略显单调。尽管如此我做了相当的调研，从可拓展性的角度，把应用划分成五个模块。围绕数据处理，基础功能和前端 Leafetjs 可视化模块三个核心模块，能够较快捷地开发新的功能。在基本功能可视化点和边和把连续的地图点映射到离散的地图中的能力的基础上应用实现出来的三个功能，路径规划功能，可视化点和边的功能以及地图匹配功能，证明了应用的可拓展性。其中地图匹配功能可视化建立轨迹匹配真实路径的标注过程，显示了可视化的威力。

在本次毕设中，对于地图渲染用得是产品级的贴片服务。注意到实验室有工作在做地图补全，这个需要从 OpenStreetMap 的原始数据开始，对地图数据进行补全后，架设贴片服务器读取地图数据，渲染出贴片服务。这部分工作具有相当的挑战[23]，未来完成了这部分工作后，那么应用的灵活性更好了。

在研究工作中，我们希望更多聚焦到算法层面上，对于可视化工作的关注较少，对于一次性的工作来讲，不考虑美观性，简单暴力的画布上直接画会比较快。但是从积累的角度上来讲，找到一套可视化的方法流程，并把大家的才智积累下来是更经济高效，正所谓磨刀不误砍柴工。

参考文献

[1] Yin Wang Hong Wei and Yanmin Zhu. Fast viterbi map matching with tunable weight functions. *GIS*, 2012.

[2] C. E. White, D. Bernstein, and A. L. Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C:Emerging Technologies*, 8(1-6):91–108, 2000.

[3] J. Yang, S. Kang, and K. Chon. 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, 2005.

[4] C. A. Blazquez and A. P. Vonderohe. Simple map-matching algorithm applied to intelligent winter maintenance vehicle data. *Transportation Research Record*, 1935(1):68–76, 2006.

[5] X. Li, M. Li, W. Shu, and M. Wu. A practical map-matching algorithm for gps-based vehicular networks in shanghai urban area. *IET Conference on Wireless, Mobile and Sensor Networks*, 2007.

[6] J. S. Greenfeld. Matching gps observations to locations on a digital map. *Transportation Research Board*, 2002.

[7] M. A. Quddus, L. Zhao W. Ochieng, and R. Noland. A general map matching algorithm for transport telematics applications. *GPS Solutions*, 7(3):157–167, 2003.

[8] N. R. Velaga, M. A. Quddus, and A. L. Bristow. Developing an enhanced weight-based topological map-matching algorithm for intelligent transport systems. *Transportation Research Part C: Emerging Technologies*, 17(6):672–683, 2009.

[9] Y. Zheng and M. A. Quddus. Weight-based shortest path aided map-matching algorithm for low frequency gps data. *Transportation Research Board*, 2011.

[10] T.Griffin, Y.Huang, and S.Seals. Routing-based map matching for extracting routes from gps trajectories. *International Conference on Computing for Geospatial Research Applications*, 2011.

[11] O. Mazhelis. Using recursive bayesian estimation for matching gps measurements to imperfect road network data. *Intelligent Transportation Systems*, 2010.

[12] F. Marchal, J. Hackney, and K. Axhausen. Efficient map matching of large global positioning system data sets: Tests on speed-monitoring experiment in zu¨rich. *Transportation Research Record*, 1935(1):93–100, 2005.

[13] Y. Lou, C. Zhang, X. Xie Y. Zheng, W. Wang, and Y. Huang. Map-matching for low-sampling-rate gps trajectories. *ACM SIGSPATIAL GIS*, 2009.

[14] O. Pink and B. Hummel. A statistical approach to map matching using road network geometry, topology and vehicular motion constraints. *ntelligent Transportation Systems*, 2008.

[15] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson. Vtrack. accurate, energy-aware road traffic delay estimation using mobile phones. *SenSys*, 2009.

[16] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. *GIS*, 2009.

[17] Alt H, Efrat A, Rote G, and Wenk. C. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003.

[18] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. *VLDB*, 2005.

[19] Wikipedia. Fréchet distance — wikipedia, the free encyclopedia, 2015. [Online; accessed 8-June-2015].

[20] OpenStreetMap Wiki. Planet.osm — openstreetmap wiki,, 2015. [Online; accessed 26-May-2015].

[21] Wikipedia. Java api for xml processing — wikipedia, the free encyclopedia, 2014. [Online; accessed 26-May-2015].

[22] Yijiao Chen, Kaixi Yang, Hao Hu, Zhangqing Shan, Renchu Song, and Weiwei Sun. Finding time-depent hot path from gps trajectories. *WAIM*, 2014.

[23] OpenStreetMap Wiki. Creating your own tiles — openstreetmap wiki,, 2013. [Online; accessed 10-June-2015].

致谢

时光飞逝，毕业在即，有很多要感谢的人。在遇到困难时，实验室的师兄师姐总能给出及时的帮助。在困惑时，有孙未未老师的真挚的建议。