基于信令轨迹的路网匹配项目方案

一、项目问题介绍

本项目研究面向信令轨迹的路网匹配技术，设计并实现相关算法。算法以交通路网和信令轨迹（带时间戳的基站序列）为输入，输出轨迹所对应的行驶路径（路段序列）。

算法评判标准：基于 moving object 产生的信令轨迹与 GPS 轨迹，分别进行路网匹配得到 path1 和 path2，以 GPS 轨迹匹配结果 path2 为 ground truth，通过 RMF 和 CMF 度量计算误差。

问题的核心挑战：信令的噪声、稀疏等数据质量问题。

二、方案设计

2.1 算法框架

如下图所示，信令轨迹的路网匹配算法包括以下三个主要步骤，分别解决“数据质量”、“路网映射”、“结果评价”问题，如图 1 所示。

（1）步骤一，**信令轨迹数据与路网数据预处理：**

- 信令轨迹预处理：旨在进行必要的轨迹切分、去噪、平滑操作，得到与真实运动轨迹相符的信令序列，作为路网匹配的输入；
- 路网数据预处理：对原始路网数据进行重组并构造空间索引，为路网映射做准备；

（2）步骤二，**信令轨迹的路网映射**：信令空间向路网空间的转换，将信令序列转换为路段序列，计为 path1。算法基于 HMM，包含发射概率计算、候选路段选取、转移概率计算和路径匹配四个步骤；

（3）步骤三，**指标计算**，将结果与 GPS 轨迹的匹配结果 path2 比对，计算误差，评估算法精度。

方案流程图如下：

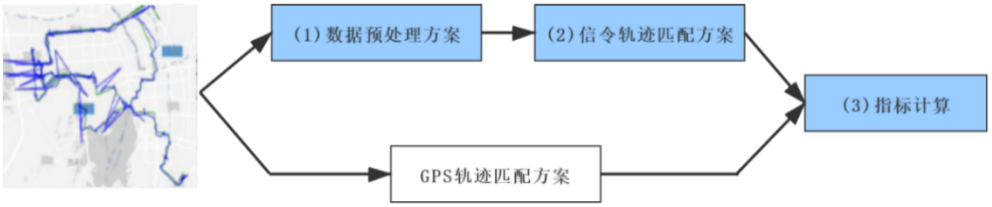


图 1. 总体框架

* 图 1 中，GPS 轨迹匹配采用‘Hidden Markov Map matching through noise and sparseness’方法，其结果作为信令轨迹匹配的 Ground truth。

2.2 数据预处理方案

2.2.1 噪声过滤

由于传感器噪声和其他因素（例如在城市峡谷中接收到的定位信号很差），空间轨迹可能存在较大噪声，如图 2（a）所示，不能直接进行路网匹配。因此需要对轨迹进行噪声过滤，解决数据质量问题，得到与真实 GPS 轨迹相符合的轨迹，如图 2(b)。这对保证路网匹配精度十分重要。

图 2 (a) 原始信令轨迹

图 2 (b) 信令轨迹噪声过滤

本方案对轨迹数据进行三种过滤处理，分别是速度过滤、角度过滤、加权均值过滤。速度与角度过滤旨在过滤明显的违背常识的位置点,如形成不规则的瞬时速度或“尖锐”角度的位置点等。加权均值过滤旨在平滑轨迹，并检测出前两种过滤中没被捕捉到的异常点。

- **速度过滤**。核心思想：相邻轨迹点之间的移动速度应该小于一定阈值(例如 150 千米/小时)，即速度约束下是否可达。我们首先利用相邻轨迹点之间的速度可达性进行去噪。详见算法 1。

  具体来说，计算轨迹序列中各位置点的速度值（由欧式距离和时间计算），若一个位置点的当前速度超过了设定的阈值 v_{max}（Line 3），则将该位置点认为是由采样误差造成的噪声将其过滤。例如,图 2-2-2 中位置点 p_t 速度为 185km/h,超过设定阈值 v_{max}=150km/h，因此过滤后轨迹序列为…→p_{t-2}→p_{t-1}→p_{t+1}→p_{t+2}→…。

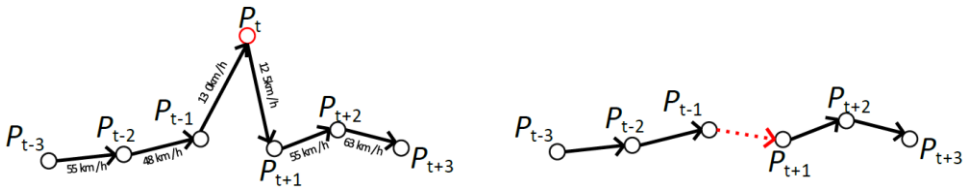


图 2-2-2

算法 1：速度过滤

Input： 一条轨迹$P = \{p_1, p_2, \cdots, \cdots p_N\}$，速度阈值 v_{max}

1. L ←|P| //轨迹长度
2. **while** i ← 0 **to** $L - 1$ **do**
3. **if** 点间速度(p_i, p_{i+1}) > v_{max}
4. **remove** p_{i+1} from P
5. **else**
6. i ← i+1
7. **end while**

Output： 速度过滤后轨迹P

- **角度过滤**。核心思想：轨迹通常向重点方向持续移动，较少出现连续的大幅度变向，即“乒乓”效应，我们利用角度特征进行去噪。

  具体来说，我们计算对于当前位置点 p_t，我们计算连续两个夹角∠$p_{t-1}p_tp_{t+1}$与∠$p_tp_{t+1}p_{t+2}$，如果这两个角度同时小于设定角度阈值 θ_{min}，，即产生“乒乓”效应，则将位置点 p_t 视为角度异常位置点并将其过滤（Line 4），得到修正轨迹…→p_{t-1}→p_{t+1}→p_{t+2}→…。考虑到一条轨迹上连续出现多个“乒乓”效应的可能，需要对一条轨迹持续执行上述操作直至此轨迹不再变化。角度过滤详见算法 2。

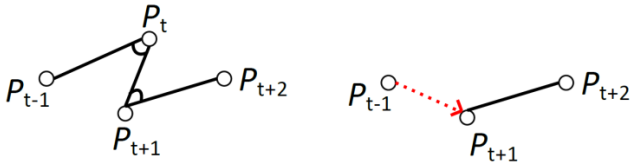


图 2-2-3

算法 2：角度过滤

Input： 一条轨迹$P = \{p_1, p_2, \cdots, \cdots p_N\}$，角度阈值 θ_{min}

1. L ←|P| + 1//记录轨迹长度用于比较
2. **while** |P| < L **do**

3. L ←|P|
4. **for** i ← 1 **to** |P| − 2 **do**
5. **if** 点间角度(p_{i-1}, p_i, p_{i+1}) > v_{max} **and** 点间角度(p_i, p_{i+1}, p_{i+2}) > v_{max}
6. **remove** p_i from P
7. **else**
8. i← i+1
9. **end for**
10. **end while**

Output：角度过滤后轨迹P

- 加权均值过滤。核心思想：对轨迹进行平滑处理，相互校正（考虑周围密度）。我们发现轨迹中存在一些异常轨迹点不能被上述速度过滤和角度过滤检测到，且轨迹存在采样密度不均匀的现象，需要通过平滑操作来进一步修正。详见算法 3.

  传统的均值滤波器是利用一个滑动窗口对坐标进行平滑。如果窗口过小，则当轨迹点采样间隔较小时，平滑效果不明显。如果窗口过大，则当轨迹点采样间隔较大时，会发生过度过滤的现象。

  因此，我们考虑信令轨迹采样密度的不均匀性，采用加权均值过滤的方式。基本思想是在平滑位置点 p_t 时，设置时间上远离 p_t 的位置点的平滑影响小于时间上靠近 p_t 的位置点的影响，即利用高斯函数建立平滑窗口内一个位置点对另一个位置点影响的权重分布，以实现对于不同采样频率的轨迹数据的适应性平滑处理。具体地，对于位置点 p_i，窗口中时间戳为 t_j 的位置点 p_j 对位置点 p_i 的影响权重$w(p_i, p_j)$定义为：

$$w(p_i, p_j) = \frac{1}{\sqrt{2\pi}\sigma_m} \exp\left[-\frac{(t_j - t_i)^2}{2\sigma_m^2}\right] \quad \text{（式 2-2-1）}$$

因此，对于位置点 p_i 的长度为 η 的窗口，我们根据 x 坐标轴和 y 坐标轴对窗口内所有的位置点进行排序并生成两个有序列表$\mathcal{L}_x = \left\{p_x^{(x_1)}, p_x^{(x_2)}, \cdots, p_x^{(x_\eta)}\right\}$和$\mathcal{L}_y = \left\{p_y^{(y_1)}, p_y^{(y_2)}, \cdots, p_y^{(y_\eta)}\right\}$（$p_x^{(x_1)} \le p_x^{(x_2)} \le \cdots \le p_x^{(x_\eta)}$，$p_y^{(y_1)} \le p_y^{(y_2)} \le \cdots \le p_y^{(y_\eta)}$）(Line 7-8)，加权均值过滤器根据以下公式调整位置点 p_i 的坐标以达到平滑效果(Line 14-15)：

$$\hat{p}_x^{(i)} = \frac{1}{\sum_{k=2}^{\eta-1} w\left(p_x^{(x_k)}, p_i\right)} \sum_{j=2}^{\eta-1} w\left(p_i, p_x^{(x_j)}\right) \cdot p_x^{(x_j)} \quad \text{（式 2-2-2）}$$

$$\hat{p}_y^{(i)} = \frac{1}{\sum_{k=2}^{\eta-1} w\left(p_y^{(y_k)}, p_i\right)} \sum_{j=2}^{\eta-1} w\left(p_i, p_y^{(y_j)}\right) \cdot p_y^{(y_j)} \qquad \text{(式 2-2-3)}$$

算法 3：加权均值过滤

Input： 一条轨迹$P = \{p_1, p_2, \cdots, \cdots p_N\}$，窗口大小 α，权重参数σ

1. radius ← (α-1)/2 //窗口半径
2. **for** i ← radius **to** $N - radius$ **do** //以大小为 α 的窗口滑动遍历
3. coef_lng ← 0 //纬度权重之和
4. sum_lng ← 0 //纬度加权和
5. coef_lat ← 0 //经度权重之和
6. sum_lat ← 0 //经度加权和
7. lnglist ←P[i: i+$\alpha - 1$]内所有采样点的经度值排序并去除最大最小值后的集合
8. latlist ←P[i: i+$\alpha - 1$]内所有采样点的纬度值排序并去除最大最小值后的集合
9. **for** j ← 0 **to** |lgnlist| **do** //利用窗口内点的加权和来修正坐标
10. coef_lng ← coef_lng + w(lnglist[j], P[i+radius],σ) //权重函数 w(.)见式 2-2-1
11. sum_lng ←sum_lng + w(lnglist[j], P[i+radius],σ)*P[i+radius].lng//计算加权和
12. coef_lat ← coef_lat + w(latlist[j], P[i+radius],σ)
13. sum_lat ←sum_lat + w(latlist[j], P[i+radius],σ) *P[i+radius].lat
14. P[i+radius].lng ← sum_lng/coef_lng //权重归一化
15. P[i+radius].lat ← sum_lat/coef_lat
16. **end for**
17. **end for**

Output： 加权均值过滤后轨迹P

综上，对于原始信令轨迹，本方案依次通过速度过滤，角度过滤和加权均值过滤算法进行噪声过滤，从而得到与“真实轨迹”（图 2 a）相符的信令轨迹（图 2 b）。

2.2.2 轨迹分段

原始的 GPS 和信令轨迹是行驶车辆定位的连续采样序列，体现车辆的长期移动历史，包含了多个 trips，如图 3 所示。我们因此需要通过轨迹分段（trajectory segmentation）对长时段的 GPS 与信令轨迹（例如以天、月为单位）的合理切分与标注，切分后的每个

子段代表一次出行记录（trip），具有真实的数据语义。由于本项目同时涉及 GPS 和信令数据，下文分别介绍 GPS 轨迹和信令轨迹的分段策略。

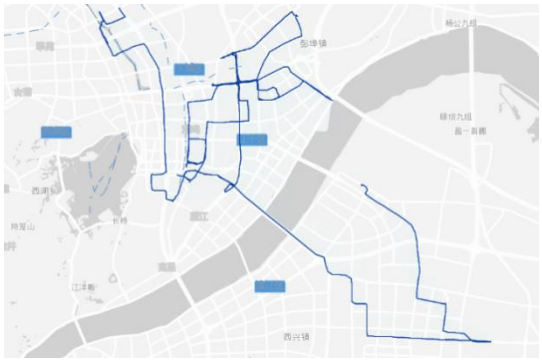


图 3（a）原始 GPS 轨迹

图 3（b）原始信令轨迹

GPS 轨迹分段。对于 GPS 轨迹，我们采用车辆载客状态标签与停留时间阈值相结合的分段策略。其核心思想是利用车辆载客状态标签识别出租车的每次载客（视为一段），并对未载客的轨迹段利用停留时间阈值进一步分段。

具体来说，我们首先获取各轨迹点的载客状态，状态标签为 1 表示车辆载客，标签为 0 表示车辆未载客。算法提取连续载客轨迹段（状态标签为 1 的连续子段，Line 9）和连续未载客轨迹段（状态标签为 0 的连续子段，Line 11-14）。详见算法 4。

算法 4：GPS 轨迹切分

Input： 车辆的 GPS 轨迹$P = \{p_1, p_2, \cdots, \cdots p_N\}$, 时间间隔阈值ε

1. label ← 0　　//初始化载客状态标签为 0，表示未载客
2. beginID ← 1　　//初始化开始位置
3. list ← ∅　　//存储切分后 GPS 轨迹
4. **for** i ← 1 **to** N **do**
5. 　　tempLabel ← 获取当前采样点p_i的载客状态(0 或 1)
6. 　　**if** tempLable ≠ label **then**　　//当前采样点载客状态发生改变
7. 　　P_{new}←$P[beginid : i]$　　//提取划分轨迹段
8. 　　**if** label = 1 **then**　　//如果为P_{new}载客轨迹段
9. 　　将P_{new}加入 list
10. 　　**else**
11. 　　**for** j ← 1 **to** $|P_{new}|$-1 **do**
12. 　　timediff ← p_{new}[j+1].time − p_{new}[j].time　　//计算当前采样点与后面采样点的时间间隔
13. 　　**if** timedff > ε **then**
14. 　　在当前采样点切分轨迹段并将切分轨迹加入 list
15. 　　label ← tempLable
16. 　　beginID ← i　　//重置轨迹开始位置
17. **end for**

Output：切分后 GPS 轨迹集 list

由于出租车的载客连续子段是一个打车的 trip，算法将其记为一个轨迹段（Line 9）。而对于连续未载客的连续子段（Line 11），其中可能包含了多个 trips，我们将通过算法 2 根据停留的时长阈值进一步切分。具体来说，我们校验两个连续采样点之间的时间间隔，若其间隔大于给定阈值 ε （例如 5 分钟）（Line 13），则认为此处是 trip 的中断，在此处对轨迹进行切分。例如图 2-2-1 中，若轨迹点 p_2、p_3的时间间隔为 10 分钟，则将该轨迹切分为两段（$p_1 \rightarrow p_2$和 $p_3 \rightarrow \cdots \rightarrow p_9$）。

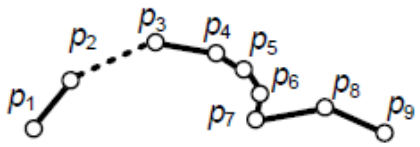


图 2-2-1

信令轨迹分段。考虑到由基站分布导致的信令轨迹采样频率不均衡问题，针对长时段的信令轨迹不能采用固定时间间隔阈值的切分方式。对此，我们设计一个时间间隔阈值自适应的切分方式。

算法的核心思想是：对于信令轨迹点 p_i 和 p_{i+1}，我们将分别考虑其前后两个窗口之内的平均时间间隔(具有相似基站分布)（Line 5-6），作为判断是否在该轨迹点处进行切分的自适应时间间隔阈值。若相邻采样点的时间间隔同时大于前后两个窗口内时间间隔的均值(Line 7)，则将信令轨迹从该采样点处进行切分。考虑到实际应用中采样频率存在正常波动，本方案引入时间间隔缩放因子 λ 对窗口内时间间隔均值进行调整。详见算法 5。

算法 5：信令轨迹切分算法

Input： 一个车辆的信令轨迹$P = \{p_1, p_2, \cdots, \cdots p_N\}$, 窗口大小 α, 时间间隔缩放因子 λ

1. beginid ← 1 //初始化开始位置
2. tflist ← 存储相邻采样点间的时间间隔
3. list ← ∅ //存储切分后信令轨迹
4. **for** i ← 1 **to** N **do**
5. tf_{pre} ← avg(tflist[i-α : i]); //计算前窗口内相邻采样点时间间隔的均值
6. tf_{aft} ← avg(tflist [i : i+α]); //计算后窗口内相邻采样点时间间隔的均值
7. **if** tflist[i] > λ* tf_{pre} **and** tflist[i] > λ* tf_{aft} **then**
8. $P_{new} \leftarrow P[beginid : i]$

9. 将P_{new}加入 list

10. beginid ← i

11. **end for**

Output：切分后信令轨迹集 list

2.2.3 路网数据重组及构造空间索引

原始路网数据存在仅为路段中间点即只联通了前后两个路段的路口节点。路网数据重组是将所有的此类路口节点降级为路段中间折线点,以便于后期进行候选路段合并操作。

算法 6：路网数据重组及空间索引算法

Input：路网数据 G=<V,E>

1. **for e in E do**

2. **if** e.inDegree==e.outDegree==1 **then**

3. 将 e 的前后路段合并

4. **end for**

5. **for e in E do**

6. 将 e 拆分为最小粒度的路段后存入 Rtree 中

7. **end for**

Output：重组后的路网 G,空间索引 Rtree

2.3 信令轨迹匹配方案

轨迹地图匹配是将原始经纬度坐标序列转换为路段序列的过程。了解车辆所处的道路，对评估交通流量，引导车辆导航，预测车辆行驶方向，寻找车辆起始点间最频繁路径等问题都非常重要。考虑到复杂的道路类型，如盘桥，高架，平行道路等、稠密的路网路段、低密度的采样率、高偏差的信令坐标、高噪声的干扰，地图匹配问题存在巨大挑战。

地图匹配问题的关键是在轨迹采样点匹配出的路段的邻近程度与路段连接后道路的可行性之间的权衡。为避免匹配出的路径的不合理性,本方案考虑路网中道路的连通性，采用隐马尔可夫模型（Hidden Markov Model, HMM）以及改进的候选篱笆图进行信令轨迹的路网匹配，以有效地集成有噪声的轨迹数据和路径约束。

针对稠密路网，以往的 HMM 方法往往依据固定距离选取候选路段，如在市区选取周边 200 米的道路，可能存在三百余条。这样会导致匹配时间过长、整体效果不佳的问题，因此我们考虑根据轨迹采样的密度进行动态调控候选路段选择的数量。

此外，我们考虑到普通的 HMM 维特比算法在高噪声的情况下会出现绕路匹配的情形，所以针对性地对维特比候选篱笆图进行改进，建立必要的跳跃路径，通过维特比算法的路径选择来全局地判断是否认为其是噪声点从而进行跳跃，避免噪声点对整体的影响。

路网算法将包含五个阶段：候选路段选择，构建候选篱笆图（candidate lattice graph），计算发射概率与转移概率，最优路径匹配。我们的算法框架如图 2-3-1 所示。

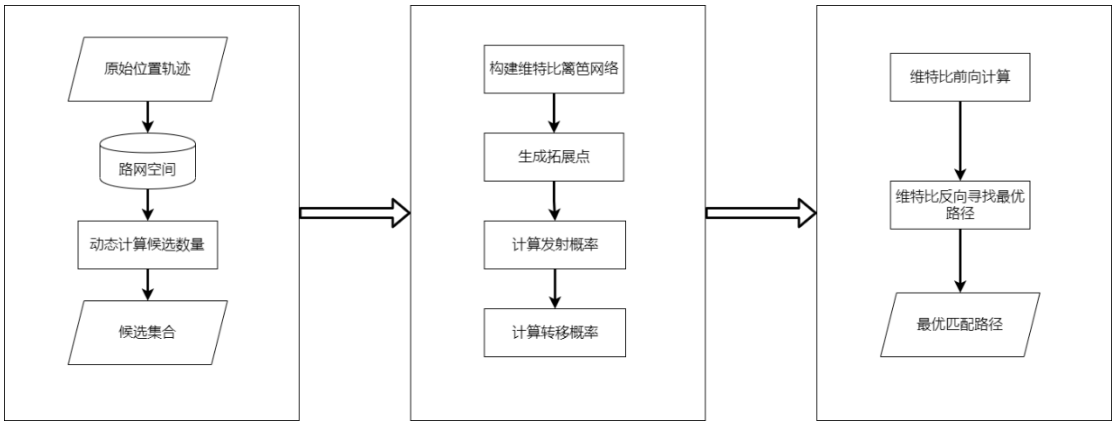


图 2-3-1 算法框架

1) **候选路段选择**：候选路段选择是在路网$G<V,E>$中将每一个轨迹点z_t投影到一个候选路段集合（$c_{t,1}$，$c_{t,2}$，$c_{t,3}$，…）。这些候选路段表示轨迹点可能是由这些候选路段映射来的，可能是这个轨迹点所对应的真实位置。这个组件输入为路网$G<V,E>$、轨迹点序列 Z，利用路网的几何信息与拓扑信息、轨迹的位置信息与密度信息，输出一组候选集合 C。

2) **构建候选篱笆图（candidate lattice graph）**：候选篱笆图中的每一列代表观测状态所对应的可能的隐藏状态。但是由于轨迹点坐标存在较大噪声，其候选路段可能无法命中真实路径路段，即 Candidates miss，导致局部结果的匹配错误，出现绕路。为了增强算法的鲁棒性，我们增加必要的 shortcuts，使得篱笆图包含必要的层间跳跃。这样将使得路径有机会跳过高噪轨迹点，屏蔽其 candidates miss 带来的影响，避免错误的绕路匹配。

3) **计算发射概率与转移概率**：在路网匹配方案中，发射概率用于量化一个轨迹点匹配到某个潜在可行路段的可能性，我们假设越接近轨迹点的路段越有可能是

其真实位置；而转移概率用于量化相邻两个候选路段之间转移的可能性，我们假设其路网距离越接近对应轨迹点的欧氏距离转移概率越大。

4) **最优路径匹配**：综合带拓展路段的候选篱笆图与发射概率、转移概率，通过维特比动态规划的正向推算，得到各个的路径概率，然后通过维特比算法思想搜索概率最大的路径，得到全局最优路径。

2.3.1 候选路段选择【平行路段可以显著优化，不在本方案】

确定候选路段个数。对每个轨迹点，我们需要选择适量的路段作为匹配候选，从而保证整体的匹配性能。简单的方法是选择固定数量的候选路段或固定距离的候选路段，例如每个轨迹点的候选匹配路段为发射概率最高的 60 个路段或者 200 米内的所有路段。但是固定的候选数量与固定距离的候选无法适应基站密度过高或过低的场景。对此，本算法采用可变数量的候选路段选择策略。

令 $Z=(z_1, z_2, \cdots, z_n)$为信令轨迹的观测位置序列。给定第 i 个轨迹点的观测位置 z_i，我们为其选取 NumOfCandidates 条转移概率最大的路段作为候选路段，确保对真实路段的覆盖。为了兼顾覆盖与效率，我们采用可变数量的候选策略，对不同轨迹点根据实际情况确定 NumOfCandidates 数量。当 z_i 与 z_{i-1} 或 z_{i+1} 较远时，需要选取较多的候选路段以确保命中；反之，当与 z_{i-1} 或 z_{i+1} 较近时，可以降低候选路段数量 NumOfCandidates，以便提高匹配性能并避免噪声候选的干扰。

具体来说，NumOfCandidates 被量化为：

$$\text{NumOfCandidates} = \text{DefaultNumber} + [\text{Min}(|z_{i-1}, z_i|, |z_i, z_{i+1}|) - \text{ReferDistance}] * \text{ResizeNumber} \quad (式\ 2\text{-}3\text{-}2)$$

其中 DefaultNumber 为默认候选数量（设为 60），$|z_{i-1}, z_i|$与$|z_i, z_{i+1}|$为当前观测点与前后观测点间的距离，ReferDistance 为参考距离，ResizeNumber 为候选路段调整的基准数量。经验表明合适的候选数量与$(\text{Min}(|z_{i-1}, z_i|, |z_i, z_{i+1}|) - \text{ReferDist})$基本符合线性关系。我们还使用了上下边界候选数量以提高健壮性。

确定候选路段。对给定的观测点 z_t 的条候选路段搜索(line 3)利用 Rtree 空间索引，查找在 z_t 一定范围内且路段限速大于 SpeedLimmit 的最近的 NumOfCandidates 个路段

考虑到轨迹所在区域的复杂性，在城区的观测点附近充满了大量的低限速路段(如小区内部路段)，但是总体路段距离较近；在郊区的观测点附近低限速路段较少，但是总体路段距离较远；上述查找候选路段操作的意义在于以 z_t 为中心逐渐扩大搜索范围，直至

找到数量为 NumOfCandidates 个候选路段，且这些路段的路段限速均大于 SpeedLimit，能够较好的应对不同区域内候选路段查找的问题。

算法如下：

算法 7：路网匹配轨迹点候选路段算法

Input： 轨迹点测量序列 $Z=(z_1, z_2, \cdots, z_n)$，路网 R-Tree 索引 root，发射概率参数σ_z，候选路段限速阈值 SpeedLimit,候选路段数量NumOfCandidate。

1. 初始化候选路段集合 $C=(c_1, c_2, \cdots, c_n)$，发射概率矩阵 B。
2. **for** $t \leftarrow 1$ **to** n **do**
3. $c_t \leftarrow$ 基于 R-Tree 的最临近查询得到距离位置测量 z_t 最近且路段限速大于 SpeedLimit 的前NumOfCandidate个候选路段;
4. $c_m \leftarrow$ 对c_t按照路网连通性进行候选合并，每条路上保留一个候选路段
5. **for** $j \leftarrow 1$ ***to*** $|c_t|$ **do**
6. $B(z_t, r_j) \leftarrow$ 根据式 2-3-1 计算发射概率 $p(z_t|r_j)$;
7. **end for**
8. **end for**

Output： 候选路段集合 C，发射概率矩阵 B

2.3.2 构建候选篱笆图（candidate lattice graph）

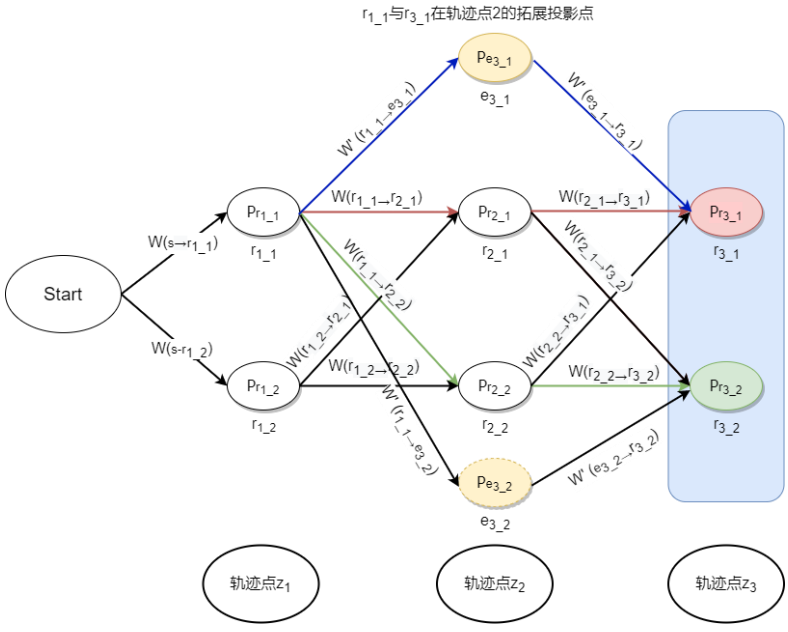


图 2-3-2 构建候选篱笆图示意图

如图 2-3-2 所示，候选篱笆图是维特比算法解决最优路径时所提出的。候选篱笆图中的每一列代表观测状态所对应的可能的隐藏状态，在路网匹配问题中便是观测到的轨迹点所对应的可能的候选真实路段，如“轨迹点z_1”可能对应路段 r_{1_1}、r_{1_2}。每个候选路段上都有一个累积概率，如$P_{r_{1_1}}$代表路段 r_{1_1}的累积概率。连接两个候选路段的是带有概率权重W的有向边，由节点累积概率、后面 2.2.3 的发射概率和 2.2.4 的转移概率计算得到。当维特比向前计算时，会通过权重不断更新候选路段的累积概率值。普通维特比算法的候选篱笆图只包含图 2-3-2 中间的中间部分，不包含黄色节点以及相连边。这样也就是说每一个轨迹点z_t都会变成必经点，在它所对应的候选路段中选择一个代表来参与到最终的路段生成中。但如果某个轨迹点z_t是噪声点，距离正常路径很远，这样会导致不正确的匹配，产生异常绕路。普通的维特比算法很难去识别以及对抗噪声的影响。例如在图 2-3-2 中，轨迹点z_3

生成拓展候选路段。考虑到轨迹点序列存在的噪声问题，我们通过在构建候选篱笆图时，尝试建立必要的跳跃路径。这样使得对Z_t的每个候选路段r_{t_i}都能通过一个 shortcut 路径与Z_{t-2}相连，来对抗噪声。如图 2-3-2 所示，假设轨迹点z_2为噪声点，距离正常路径偏差很大，候选篱笆图中边权W已经计算完成。我们想要在普通维特比的候选篱笆图中建立一条$z_1 \to z_3$的 shortcut 路径来避免噪声点z_2的影响。我们对轨迹点z_3的每个候选路段r_{3_k}寻找对应的r_{1_i}能够使得路径$r_{1_i} \to r_{2_j} \to r_{3_k}$的边权最大。即

$$r_{1\_i} = \underset{i,j}{\operatorname{argmax}} P(r_{1\_i}) + W(r_{1\_i} \to r_{2\_j}) + W(r_{2\_j} \to r_{3\_k}), i \in c_1, j \in c_2 \qquad \text{（式 2-3-1）}$$

随后通过求r_{1_i}与r_{3_k}的最短路径来建立$z_1 \to z_3$的 shortcut 路径，但由于原来r_{3_k}的累积概率是由$P(r_{1_i})$、$W(r_{1\_i} \to r_{2\_j})$、$W(r_{2_j} \to r_{3_k})$三部分组成的，shortcut 路径的新累积概率相应地是由$P(r_{1_i})$、$W'(r_{1_i} \to r_{3_k})$两部分组成的，其中$W(r_{1_i} \to r_{2_j})$、$W(r_{2_j} \to r_{3_k})$与$W'(r_{1_i} \to r_{3_k})$不能够直接拿来相比。

考虑z_2到 shortcut 路径的投影位于r_{1_i}与r_{3_k}的中间部分；在无噪声情况下投影点与原先的候选路段r_{2_j}位置接近；在有噪声情况下投影点往往会比原先的候选路段r_{2_j}权重更大。使用z_2到 shortcut 路径的投影点e_{3_k}作为轨迹点z_2与路径$r_{1_i} \to r_{2_j} \to r_{3_k}$的候选拓展点既保证了可比性，又添加了 shortcut 路径来减小噪声的影响。

这样原先$P(r_{3_k})$的最大累积概率计算为

$$P(r_{3\_k}) = P(r_{1\_i}) + W(r_{1\_i} \to r_{2\_j}) + W(r_{2\_j} \to r_{3\_k}) \qquad \text{（式 2-3-2）}$$

通过 shortcut 路径求得$P'(r_{3_k})$的新累积概率为

$$P'(r_{3\_k}) = P(r_{1\_i}) + W'(r_{1_i} \rightarrow e_{3_k}) + W'(e_{3\_k} \rightarrow r_{3\_k}) \quad \text{（式 2-3-3）}$$

对于每个$r_{3_k} \in c_3$都有其对应的 shortcut 路径与拓展候选点，通过 2.3.5 中计算最优匹配路径部分中的寻找全局最优路径时会对这些 shortcut 路径作出评估，从全局的角度考虑是否采用这些 shortcut。在轨迹点z_t是噪声点或者偏差很大的情况下，只需要很少的计算代价，也能做出正确匹配，对绕路匹配情况进行修复。

2.3.3 构建发射概率

在路网匹配方案中，发射概率用于量化一个轨迹点匹配到某个潜在可行路段的可能性。具体来说，对于路网匹配，给定一个位置测量 z_t（即采样的轨迹点），对于每一个路段 r_i 都存在对应的发射概率 $p(z_t|r_i)$，表示如果车辆实际在路段 r_i 上，产生测量 z_t 的概率。直观上，采样轨迹点与路段的距离越近则该路段与采样轨迹点的匹配程度越高（即发射概率越高），而如果距离某采样轨迹点较远的一些路段，则不太可能产生对应的采样轨迹点。

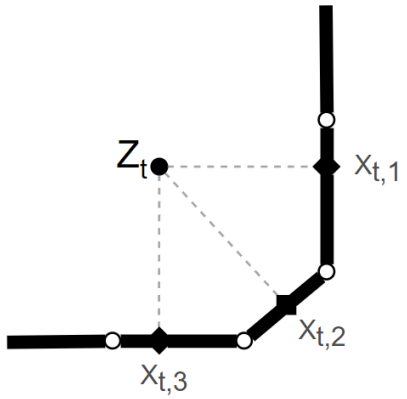


图 2-3-3 轨迹路网匹配距离度量示意图

根据上述假设，我们形式化轨迹路网匹配的发射概率。如图 2-3-1 所示，对于给定的位置测量 z_t 和候选路段 r_i，我们考虑候选路段 r_i 中所有折线段并求出每一段折线段与位置测量 z_t 最近的位置点（$x_{t,1}$，$x_{t,2}$，$x_{t,3}$，…）。假设路段 r_i 上距离 z_t 最近的位置点（即投影位置点）为 $x_{t,i}$，地球表面上位置测量点和候选点之间球面距离为$||z_t - x_{t,i}||_{\text{great circle}}$，则发射概率被定义为：

$$p(z_t|r_i) = \frac{1}{\sqrt{2\pi}\sigma_z} e^{-0.5\left(\frac{\|z_t - x_{t,i}\|_{great\ circle}}{\sigma_z}\right)^2} \quad \text{(式 2-3-1)}$$

初始状态概率 π_i, i=1…N_r 表示车辆在开始行使时所有路段分别是起始路段的概率，本方案中，我们使用第一个位置的发射概率测量 z_1, 并设置 $\pi_i = p(z_1|r_i)$

2.3.4 构建状态转移概率

对于每一个当前位置测量 z_t 和下一位置测量 z_{t+1}，都有对应的可能匹配的候选路段列表。转移概率表示车辆在这两组候选路段之间移动的概率。显然，一些需要复杂操作的转移是不可能的，且我们倾向于车辆行驶距离和位置测量之间的球面距离相等的转移。

具体地，对于位置测量 z_t 和候选路段 r_i，路段 r_i 上距离 z_t 最近的位置点为 $x_{t,i}$。对于下一位置测量 z_{t+1} 和候选路段 r_j，路段 r_j 上距离 z_{t+1} 最近的位置点为 $x_{t+1,j}$。我们使用传统的A^*算法计算这两个位置点间的最短路网距离 $||x_{t,i} - x_{t+1,j}||_{route}$（算法 7 中 Line 5），并将路网距离 $||x_{t,i} - x_{t+1,j}||_{route}$ 和两位置测量值之间的球面距离 $||z_t - x_{t,i}||_{great\ circle}$ 进行比较（算法 7 中 Line 6）。由于在道路中，一对被正确匹配的位置点之间相对较短的距离与位置测量（即轨迹采样点）之间的距离是相同的，因此状态转移概率定义为：

$$p(d_t) = \frac{1}{\beta} e^{-\frac{d_t}{\beta}} \quad \text{(式 2-3-2)}$$

具体地，

$$d_t = \left| \|z_t - z_{t+1}\|_{great\ circle} - \|x_{t,i^*} - x_{t+1,j^*}\|_{route} \right| \quad \text{(式 2-3-3)}$$

其中 i^*和 j^*为位置测量 z_t 和 z_{t+1} 真实所在路段（ground truth 路段）(算法 7 中 Line 7)。

算法 8：路网匹配转移概率算法

Input： 轨迹点测量序列 $Z=(z_1, z_2, \cdots, z_n)$，候选路段集合 $C=(c_1, c_2, \cdots, c_n)$。

1. 初始化状态转移概率矩阵 A
2. **for** $t \leftarrow 1$ ***to*** $n-1$ **do**
3. **for** $r_i \in c_t$ ***do***
4. **for** $r_j \in c_{t+1}$ ***do***
5. A^*算法计算路段r_i与路段r_j之间的最短路径距离
6. 根据式 2-3-3 计算距离度量d_t;

7. $A(r_i, r_j) \leftarrow$ 根据式 2-3-2 计算转移概率 $p(d_t)$；

8. **end for**

9. **end for**

10. **end for**

Output：状态转移概率矩阵 A

2.3.5 计算最优匹配路径

结合上述组件，构造候选篱笆图，计算所得发射概率、初始状态概率以及状态转移概率。通过 Viterbi 算法正向推算以及反向寻找全局最优路径两部分，计算给定信令轨迹的最优匹配路段序列并输出。详见算法 9。

在初始化第一层的维特比候选路段的概率后，逐层进行维特比正向运算，得到每个候选的最优概率与最优前驱。在每层维特比正向运算结束后，取得每个候选r_j前两层的局部最优路径的概率P_{r_j}（算法 9 中 Line 4），并与其前驱r_i的前驱r_k查找最短路径sp_j（算法 9 中 Line 5）。此时将前驱节点r_i所对应的轨迹点向最短路径sp_j做投影得到拓展候选点e_j，并按照$r_k \rightarrow e_j \rightarrow r_j$的顺序构造路径并求得新的最优概率$P'_{r_j}$（算法 9 中 Line 7），若$P'_{r_j}$比$P_{r_j}$更优则更新$r_j$的前驱为此候选（算法 9 中 Line 9）。

算法 9：最优匹配路径算法

Input：轨迹点测量序列 $Z=(z_1, z_2, \cdots, z_n)$，候选路段集合 $C=(c_1, c_2, \cdots, c_n)$，发射概率矩阵 B，状态转移概率矩阵 A，初始状态概率矩阵π。

1. 使用发射概率初始化P_0
2. **for** $t \leftarrow 1$ to T ***do***
3. **for** $r_j \in c_{t+1}$ ***do***
4. $P_{r_j} = \max_{r_i \in c_t}(A[r_i, r_j] * B[z_{t+1}, r_j] + P_{r_i})$, 更新$r_j$的前驱
5. 求最短路径$sp_j = \text{Dijkstra}(r_k, r_j)$
6. 求观测点z_t到最短路径sp_j的投影点e_j
7. $P'_{r_j} = A[r_{e_j}, r_j] * B[z_{t+1}, r_j] + P_{e_j}$
8. **if** $P'_{r_j} > P_{r_j}$ **do**
9. 更新r_j的前驱为e_j, $P_{r_j} = P'_{r_j}$
10. viterbi.backward(C)

Output：最优路径I^*

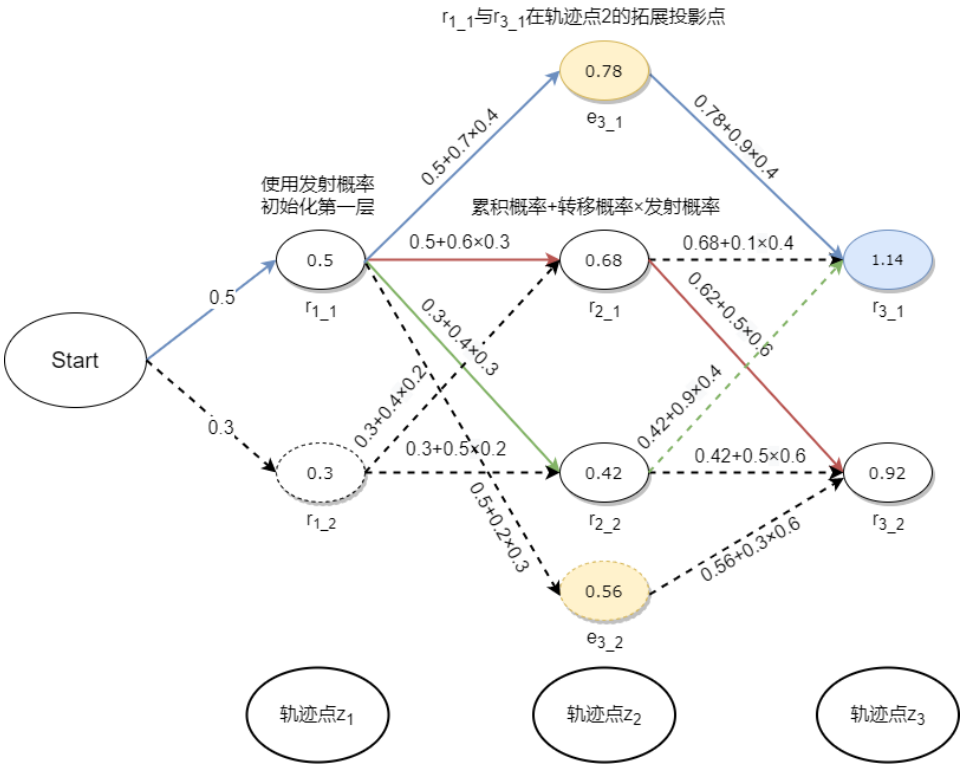


图 2-3-4 改进维特比算法示意图

举例说明算法的工作过程：在使用发射概率初始化第一层维特比后，r_{1_1}与r_{2_1}分别为 0.5 与 0.3，第二层进行维特比正向运算，r_{2_1}由第一层的累积概率+*转移概率*×*发射概率*生成，选择能连接向r_{2_1}的最大值作为新的累积概率，并记录前驱。这样，r_{2_1}的累积概率为 0.68，前驱为r_{1_1}；r_{2_2}的累积概率为 0.42，前驱为r_{1_1}。前两层由于不足三层，所以不需要进行绕路检测与拓展点的生成。第三层进行维特比正向运算，得到r_{3_1}的累积概率为 0.78，前驱为r_{2_2}；r_{3_2}的累积概率为 0.92，前驱为r_{2_1}。随后进行生成拓展点绕路检测部分，对于候选路段r_{3_2}，依据前驱找到当前的最优路径$r_{1_1} \rightarrow r_{2_2} \rightarrow r_{3_1}$，此路径的累积概率为$P = 0.78$，同时在$r_{1_1}$与$r_{3_1}$之间计算最短路径，并通过中间轨迹点 2 向最短路径投影，得到r_{3_1}所对应的拓展候选点e_{3_1}，生成$r_{1_1} \rightarrow e_{3_1} \rightarrow r_{3_1}$依照维特比正向运算的方法计算该拓展路径的$r_{3_1}$累积概率$P' = 1.14 > P$。所以更新$r_{3_1}$的概率与前驱为拓展路径。同理对$r_{3_2}$也是得到拓展路径，并与原最优路径比较，但新概率$P' < P$，所以不更新概率与前驱。这样通过维特比寻找全局最优路径的过程决定了拓展候选路段是否被采用，以及是否跳跃某些轨迹点，避免绕路，从而以很小的计算代价可以实现关键候选的拓展，极大地改善了匹配效果。命中率由原先的 87%通过拓展候选提高至 93%，CMF 由 0.156 提升至 0.149。

相比标准维特比算法，本算法在Line 4与Line 7中将前后概率累积的过程由相乘改为了相加，这是为了避免少数候选由于过大或过小的概率直接决定了最终路径选择的情况。使用概率相加目的是允许一些对全局效果有较大增益但自身概率较差的候选存在。

不过需要注意的是噪声点与转弯的关键点十分相似，我们需要通过调整对拓展候选路段的重视程度来在噪声与转弯处做出一定程度的平衡。

为了调控候选点对匹配的影响力度，增设了拓展发射距离折扣因子α,修改拓展点的发射概率计算函数为：

$$p(z_t|r_i)=\frac{1}{\sqrt{2\pi}\sigma_z}e^{-0.5\left(\frac{\alpha\|z_t-x_{t,i}\|_{great\ circle}}{\sigma_z}\right)^2}$$

α的取值在 0 到 1 之间，通过实验确定，取值越低则拓展候选点的影响力越大。

具体效果如图 2-3-5 与图 2-3-6 对比所示，左面为经过拓展点改进的维特比匹配结果，右面为普通的维特比匹配结果。可以看到，右图中存在某些采样点的候选路段数量不足，探测不到 groundtruth 所在的路段，而普通的维特比算法必须经过这个采样点，所以产生了局部或者大片的绕路情况。而通过生成拓展点，使得维特比可以尝试性地进行层间跳跃，让前后两个采样点直接相连，不必经过这个偏差较大的噪声点，而决定是否层间跳跃的，是全局维特比对这个新路径的评估，所以避免了绕路的产生。

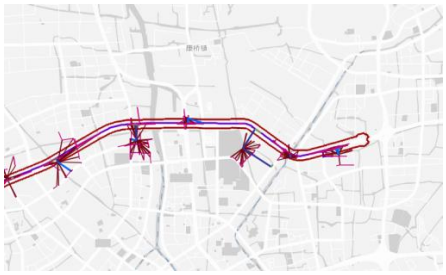


图 2-3-5 改进维特比算法

图 2-3-6 普通维特比算法

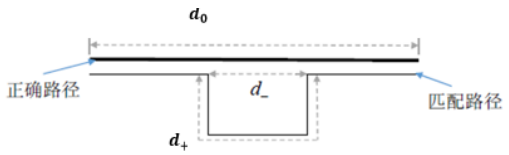
2.4 道路错误匹配指数计算

本方案采用道路错误匹配指数(Route Mismatch Fraction，RMF)评估路径匹配度，计算公式为：

$$RMF=\sum_{i=1}^{n}\left(RMP(t_i,r_i)\right)/_n \qquad (式\ 2\text{-}4\text{-}1)$$

其中，n 为轨迹的条数，t 为“GPS 轨迹匹配方案”输出的路口序列，r 为“信令轨迹匹配方案”输出的路口序列。

RMP 函数，计算的是不正确路径所占比例，对从正确路径错误添加和错误减去的不正确道路的长度求和，然后除以正确路径的长度，如下图所示。



$d_-(r_i, t_i)$ = 错误减去的长度
$d_+(r_i, t_i)$ = 错误增加的长度
$d_0\ (t_i)$ = 正确路径总长度
RMP(r_i, t_i) = $(d_-(r_i, t_i) + d_+(r_i, t_i)) / d_0\ (t_i)$

对于某个轨迹段，输出了 Top K 条匹配路网轨迹，统计 K 条路网轨迹中的 RMP 最小值作为该轨迹段的 RMP。默认 K=3。

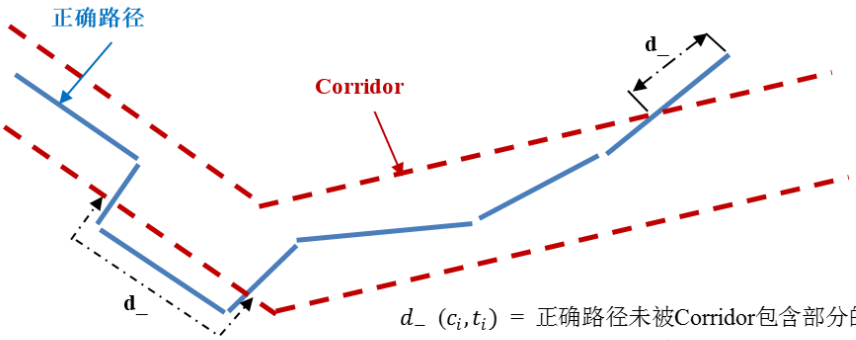
2.5 走廊错误匹配指数计算

本方案采用走廊错误匹配指数（Corridor Mismatch Fraction， CMF）评估 Corridor 匹配度，计算公式：

$$CMF = \sum_{i=1}^{n} (CMP(c_i, t_i))/n$$

其中，n 为轨迹的条数，t 为“GPS 路网匹配算法”输出的路口序列，c 为“信令路网匹配算法”输出的 Corridor。

CMP函数，计算的是Corridor未包含正确路径的长度所占的比例，用Corridor未包含的路径的长度除以正确路径的长度，如图2-5-1所示：



$d_-\ (c_i, t_i)$ = 正确路径未被Corridor包含部分的长度
$d_0\ (t_i)$ = 正确路径总长度
CMP(c_i, t_i) = $d_-\ (c_i, t_i) / d_0\ (t_i)$

图 1-5-1 corridor匹配度计算示意图

CMP 函数的计算主要分为以下几个步骤：（1）首先对于“信令轨迹匹配方案”输出的路口序列 r，方案将以路段为单位构建走廊。（2）随后对于“GPS 路网匹配算法”输出的路口序列 t，方案将计算每一条路段t_i被走廊覆盖的长度，最终汇总所有路段$t_i \in t$的计算结果从而得到 Corridor 未包含正确路径的长度所占的比例。

2.5.1 走廊构建

首先在走廊构建阶段，对于每一条“信令轨迹匹配方案”输出的路段r_i（其起点和终点分别记作s_i和e_i），方案将沿着路段的行驶方向将顶点往两侧垂直拓展 w/2 米（走廊宽度为 w 米）从而得到走廊与该路段平行的两条边$(s_i^1, e_i^1), (s_i^2, e_i^2)$。考虑到地球是一个椭球面，平面几何算法在垂直扩展的时候可能存在较大的误差，这里将引入球面几何的相关原理进行计算，详见算法 9

算法 10：走廊垂直扩展算法

Input：路段$r_i(s_i, e_i)$，走廊宽度w，地球赤道半径Rc

1. x←$\sin(s_i.lng - e_i.lng) * \cos(e_i.lat)$; //路段跨纬度长度
2. y←$\cos(s_i.lat) * \sin(e_i.lat) - \sin(s_i.lat) * \cos(e_i.lat) * \cos(s_i.lng - e_i.lng)$; //路段跨经度长度
3. brng←$\arctan\left(\frac{y}{x}\right)$; //路段方向角
4. brng←$(brng\ +\ 360)\ \%\ 360$; //路段与正北方向的顺时针夹角
5. angle1←brng+90; //顺时针旋转 90 度得到路段的垂直方向角
6. angle2←brng-90; //逆时针旋转 90 度得到路段的垂直方向角
7. angle1, angle2 ← radians(angle1), radians(angle1); //角度转弧度
8. $s_i,\ e_i$ ← radians(s_i), radians(e_i);
9. $s_i^1.lat$←$\arcsin\left(\sin(s_i.lat) * \cos\left(\frac{w}{2Rc}\right) + \cos(s_i.lat) * \sin\left(\frac{w}{2Rc}\right) * \cos(angle1)\right)$
10. $s_i^1.lng$←$s_i.lng + \arctan\left(\frac{\cos\left(\frac{w}{2Rc}\right) - \sin(s_i.lat) * \sin(s_i^1.lat)}{\sin(\text{angle1}) * \sin\left(\frac{w}{2Rc}\right) * \cos(s_i.lat)}\right)$
11. $s_i^2.lat$←$\arcsin\left(\sin(s_i.lat) * \cos\left(\frac{w}{2Rc}\right) + \cos(s_i.lat) * \sin\left(\frac{w}{2Rc}\right) * \cos(angle2)\right)$
12. $s_i^2.lng$←$s_i.lng + \arctan\left(\frac{\cos\left(\frac{w}{2Rc}\right) - \sin(s_i.lat) * \sin(s_i^2.lat)}{\sin(\text{angle2}) * \sin\left(\frac{w}{2Rc}\right) * \cos(s_i.lat)}\right)$
13. $e_i^1.lat$←$\arcsin\left(\sin(e_i.lat) * \cos\left(\frac{w}{2Rc}\right) + \cos(e_i.lat) * \sin\left(\frac{w}{2Rc}\right) * \cos(angle1)\right)$
14. $e_i^1.lng$←$e_i.lng + \arctan\left(\frac{\cos\left(\frac{w}{2Rc}\right) - \sin(e_i.lat) * \sin(e_i^1.lat)}{\sin(\text{angle1}) * \sin\left(\frac{w}{2Rc}\right) * \cos(e_i.lat)}\right)$
15. $e_i^2.lat$←$\arcsin\left(\sin(e_i.lat) * \cos\left(\frac{w}{2Rc}\right) + \cos(e_i.lat) * \sin\left(\frac{w}{2Rc}\right) * \cos(angle2)\right)$
16. $e_i^2.lng$←$e_i.lng + \arctan\left(\frac{\cos\left(\frac{w}{2Rc}\right) - \sin(e_i.lat) * \sin(e_i^2.lat)}{\sin(\text{angle2}) * \sin\left(\frac{w}{2Rc}\right) * \cos(e_i.lat)}\right)$
17. $s_i^1, s_i^2, e_i^1,\ e_i^2$ ← degree(s_i^1), degree(s_i^2), degree(e_i^1), degree(e_i^2); //弧度转角度

Output：走廊平行边$(s_i^1, e_i^1), (s_i^2, e_i^2)$

为了保证走廊的衔接性以及避免相邻走廊之间的重叠，在相邻路段的走廊交界处，方案将根据相邻路段的走廊平行边之间的相交情况得到两对交点。方案将从中根据角度关系判别并选择正确的交点对作为走廊边界构建走廊（如图 2-5-2 所示）。这里，对于相邻走廊垂直扩展所得到的两组平行边，方案对其进行两两配对并求得其交点作为走廊边界的候选顶点

（这些交点将构成一个平行四边形）。这里方案将根据路段的转向情况选择该平行四边形的某一条对象线作为走廊的边界。显然，从图 2-5-2 中我们发现，正确边界与两条相邻路段所构成的较小角应该都为锐角（或者直角），由此通过判断角度我们得到正确的走廊边界。算法详见算法 10。

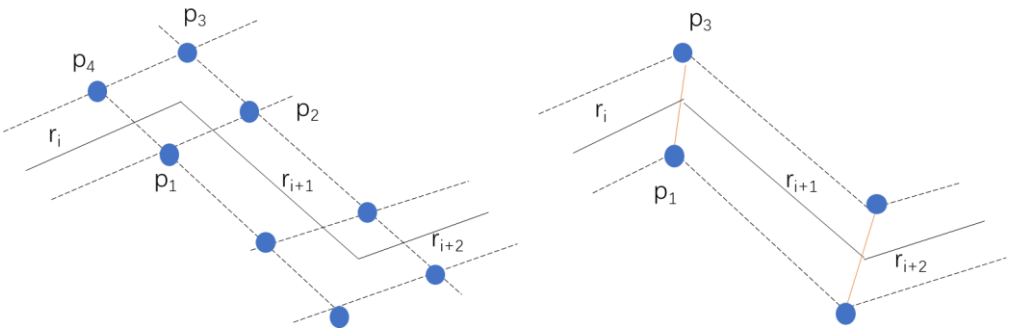


图 2-5-2 走廊示意图

算法 11：走廊边界选择算法

Input：相邻路段$r_i(s_i, e_i)$和$r_{i+1}(s_{i+1}, e_{i+1})$，走廊宽度w，地球赤道半径$Rc$

1. $(s_{i+1}^1, e_{i+1}^1), (s_{i+1}^2, e_{i+1}^2)$ ←**走廊垂直扩展算法**(r_{i+1}, w, Rc)
2. **if** r_i不存在 **then**
3. line←(s_{i+1}^1, s_{i+1}^2)
4. **Return** line
5. $(s_i^1, e_i^1), (s_i^2, e_i^2)$ ←**走廊垂直扩展算法**(r_i, w, Rc)

   //求相邻走廊边界的交点
6. p_1，p_3←Intersection((s_i^1, e_i^1)，(s_{i+1}^1, e_{i+1}^1)), Intersection((s_i^2, e_i^2)，(s_{i+1}^2, e_{i+1}^2))
7. p_2，p_4←Intersection((s_i^1, e_i^1)，(s_{i+1}^2, e_{i+1}^2)), Intersection((s_i^2, e_i^2)，(s_{i+1}^1, e_{i+1}^1))

   //根据角度关系选择走廊边界
8. **if** $\angle p_1 e_i s_i \le 90°$ **and** $\angle p_1 s_{i+1} e_{i+1} \le 90°$ **or**

   $\angle p_3 e_i s_i \le 90°$ **and** $\angle p_3 s_{i+1} e_{i+1} \le 90°$ **then** //顶点与路段的连线夹角是否为锐角
9. line←(p_1，p_3)
10. **else**
11. line←(p_2，p_4)

Output：走廊边界 line

对于任意路段r_i，通过算法 10 得到的走廊边界，我们可以最终构建得到该路段的走廊c_i。接着，通过遍历路段序列r中的所有路段，方案将最终得到“信令路网匹配算法”输出的走廊c。详见算法 11。

算法 12：走廊构建算法

Input：路段序列$r = (r_1, r_2, \dots, r_n)$，走廊宽度w，地球赤道半径$Rc$

1. $line_0$ ←**走廊边界选择算法**(ϕ，r_{i+1}, w, Rc)；//初始走廊边界
2. c←ϕ
3. **for** i← 1 ***to*** n − 1 **do**
4. $line_i$←**走廊边界选择算法**(r_i, r_{i+1}, w, Rc);
5. c_i←($line_{i-1}, line_i$)
6. **end for**
7. $line_n$ ←**走廊边界选择算法**(ϕ，$r_n'(e_i, s_i)$, w, Rc)；//终点走廊边界（输入终点路段的反向路段）

Output：走廊集合c

2.5.1 CMP 函数计算

对于 CMF 的计算，一个直接的做法是对于任一“GPS 路网匹配算法”输出的路段t_i,方案遍历所有走廊并计算t_i被每一段走廊所覆盖的长度，显然这种做法的效率比较低。因此对于 CMP 函数的计算中，方案将引入 R-tree 空间索引结构来进行加速。具体地，在走廊构建阶段，对于每一段走廊c_i，方案将构建其最小外接矩形$MBR(c_i)$并利用 R-tree 进行索引。在 CMP 函数计算阶段，对于任一“GPS 路网匹配算法”输出的路段t_i，我们构建其最小外接矩形$MBR(t_i)$并利用 R-tree 检索其中所有相交的最小外接矩形。这样我们可以得到可能与t_i相交的候选走廊集合candidate(t_i)。对于其中的候选走廊，方案将通过线段与四边形的相交关系判断并计算路段被走廊覆盖的长度。通过遍历所有“GPS 路网匹配算法”输出的路段，我们可以计算得到最终的 CMP 函数值。详见算法 12。

算法 13：CMP 函数计算算法

Input：“GPS 路网匹配算法”路段序列$t = (t_1, t_2, \dots, t_m)$，走廊集合 c，R-tree 索引 root

1. 初始化路段序列t总长度$len(t) \leftarrow 0$，路段序列t走廊覆盖长度$cover(t) \leftarrow 0$;
2. **for** i← 1 ***to*** $|c|$ **do**
3. $MBR(c_i)$←构建c_i的最小外接矩形
4. 将$MBR(c_i)$存入 root 进行索引
5. **end for**
6. **for** i← 1 ***to*** m **do**
7. $len(t) = len(t) + len(t_i)$；//计算路段$t_i$长度并更新总长度
8. $cover(t_i) \leftarrow 0$；//初始化路段t_i被走廊覆盖长度

9. $MBR(t_i)$←构建t_i的最小外接矩形
10. Candidate(t_i)←SearchIntersection(root, $MBR(t_i)$); //利用 R-tree 检索与$MBR(t_i)$相交的最小外接矩形并提取所对应的候选走廊
11. **for** c′ **in** candidate(t_i) **do**
12. $cover(t_i) = cover(t_i) + cover(t_i, \mathrm{c}')$; //计算路段$t_i$被候选走廊c′覆盖的长度并更新其被覆盖的总长度
13. **end for**
14. $cover(t_i) = \min\big(cover(t_i),\ len(t_i)\big)$; //修正因走廊之间交叉导致的路段被覆盖长度大于其总长度的情况
15. $cover(t) = cover(t) + cover(t_i)$
16. **end for**
17. $\mathrm{CMP}(\mathrm{c},\mathrm{t}) = \frac{len(t)-cover(t)}{len(t)}$

Output：$\mathrm{CMP}(\mathrm{c},\mathrm{t})$