数据结构 PJ——时空数据管理
实验报告

张睿哲
14307130222

December 20, 2015

1

Contents

1 实验概要

此次数据结构 PJ 旨在利用 OpenStreetMap 提供的地理信息数据以及上海部分出租车的轨迹数据，实现对大规模时空数据的高效管理、查询、维护。因此本人开发了名为“MyMap”的软件。此次 PJ 主要由三个部分构成：

1. 时空数据处理。MyMap 实现了地图的可视化，可以在地图上读取道路和建筑的信息，也可以通过道路和建筑的名称对相关信息进行查询。同时 MyMap 实现了在地图上选取起点和终点计算最短路及绘制路径的功能。而且 MyMap 也支持对出租车轨迹数据在地图上进行绘制。在此基础上，MyMap 还可以进行 k-短路计算。

2. 组织和索引。MyMap 实现了通过在地图上选取点进行范围查询和近邻查询，可以返回查询结果中的各项信息。

3. 挖掘和分析。MyMap 将大规模出租车轨迹数据进行了过滤，选取了部分关键点。同时 MyMap 还根据出租车轨迹信息计算出一个地点的打车指数，并会推荐出附近容易打车的地点。

因此本报告的实验过程部分将分三个方面就我在此次 PJ 中所做的工作进行详细说明。

2 实验过程

2.1 时空数据处理

2.1.1 概述

此部分主要由六个方面构成，分别是数据解析和存储、地图绘制、出租车轨迹绘制、最短路及 k-短路计算和地图对象信息查询。在此部分中，k-短路计算是本次实验的亮点，实现了一个理论复杂度非常优秀的算法。本节包含以上六个方面的介绍。

2.1.2 数据解析和存储

由于 OpenStreetMap 的地图数据是 xml 格式，因此本人使用 pugixml 库进行 xml 解析，将地图中的所有点 (xml 中的 Node) 的各项信息、所有道路和建筑（xml 中的 Way）的各项信息进行了存储。在存储过程中，依靠 hash 表对各个地图对象的 id 进行管理。同时，为了方便计算最短路，本人将地图的拓扑信息使用邻接表进行了存储，相当于建立了一张拓扑图，并使用 hash 表对真实地图和拓扑图的节点进行了关联映射。

2.1.3 地图绘制

MyMap 主要依靠 OpenCV 进行地图绘制。在解析 xml 数据的过程中，按照 OpenStreetMap 网站上的说明，对各个地图对象的层级和颜色进行了设置。在绘制过程中，先绘制地形和建筑信息，后绘制道路，各个对象按照设置的层级由低到高进行绘制。

对于绘图的具体细节，主要由以下几个方面：首先，要将经纬度坐标向平面 xy 坐标进行转化，本人经过查找资料，选择了一种比较常用的地图投影方式；其次，在绘制过程中可以发现 xml 中所给的海岸线信息未经处理，需要对其按照各段的起点终点排列拼合；最后，为了能更加方便的查看地图，MyMap 可以实现中心点固定的放大缩小和地图移动等功能，这些功能也是依靠 OpenCV 实现。

经过试验可以发现，MyMap 在运行时最耗时的部分就是地图绘制，由于 OpenCV 没有使用 GPU 渲染，所以绘图的时间较长。为了减小这一部分的时间，一种可行的优化就是对地图对象进行层次显示，就是当显示范围很大时一些细节部分就不予绘制，这样可以减小大约 20% 的时间。另一种优化方法是将地图进行平移时，由于有很大一部分是重合的，所以不需要全部重新绘制，只需要将原图截断，添上新加入的部分即可。但这种方法的需要快速查询一个区域内的地图对象，经试验发现利用 R 树进行查询的耗时比较高，不能明显地优化画图效率。而且 OpenCV 对于图像的拼接表现不是很好，容易出现一条痕迹，虽然可以通过一些计算机视觉的技术手段（如平滑算子）进行处理，但这样不仅会增加代码的复杂程度，而且目前画图的时间已经比较短，特别是在经过一次放大之后，平均一次绘制的时间在 0.2 到 0.3 秒左右，所以没有实现这个优化。

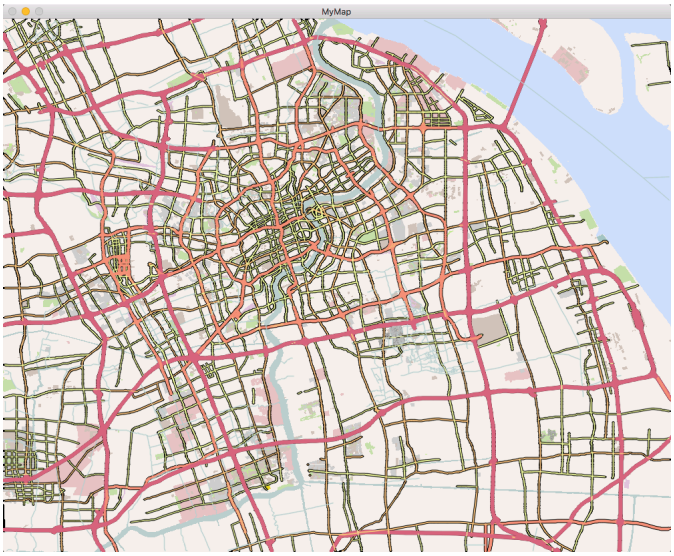

Figure 1: MyMap 绘制的上海市地图

2.1.4 出租车轨迹绘制

有了地图绘制的基础后，出租车轨迹绘制的实现就能很方便地实现。为了能更加清晰地显示从而进行可视化分析，MyMap 用红绿两种颜色来绘制轨迹，分别代表载客状态和空车状态。

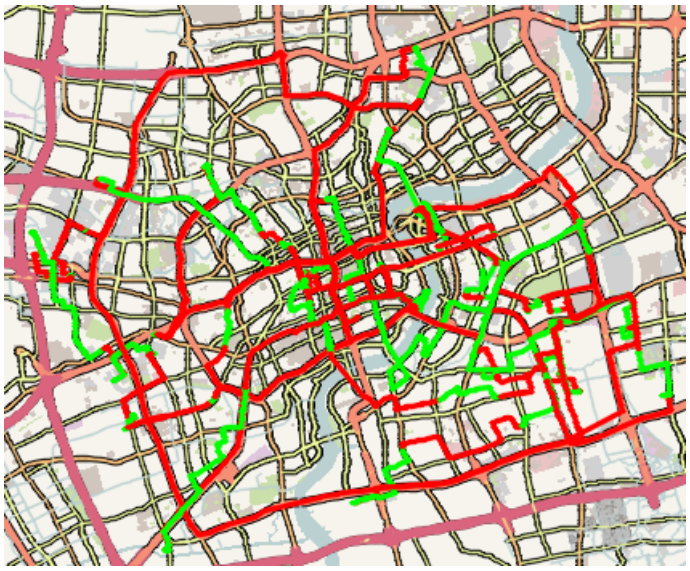

Figure 2: 出租车轨迹

2.1.5 最短路计算

通常我们在使用地图的过程中，我们主要进行两点间的最短路查询。MyMap 使用三种算法进行最短路计算，分别是 Dijsktra、SPFA、A* 算法。在具体实现细节上，针对两点间最短路查询的特点进行了剪枝优化。首先 Dijsktra 算法在执行到 T 点从堆中弹出后即可停止，同样 A* 在 T 点弹出堆后也可以停止，这样一定可以得到最优解，对于 SPFA 算法，可以通过一个点到终点的直线距离进行最优性剪枝，显然一个可能在最短路上的点必须要满足：$dis[u] + dist(u, v) \leq dis[T]$，其中 dis 是 SPFA 算出的从起点出发的最短路距离，dist 表示两点间的直线距离。[1]该优化可以减小 50% 左右的运行时间。

在具体实现的过程中，由于是在地图上选择起点和终点，所以很容易选到不在路上的点，因此我使用 k-d 树进行临近查询，分别在起点和终点附近找到两个点，两两之间做最短路，最后选择路程最短的输出答案。

[1]此优化由龚绩阳同学提出。

这三种算法的运行时间如下表所示：

100 组随机数据	测试 1	测试 2	测试 3	测试 4	测试 5	平均值
A*	1672.99ms	1720.43ms	1725.75ms	2113.97ms	1447.14ms	1736.06ms
Dijkstra	2392.35ms	2370.94ms	2317.81ms	2269.33ms	1973.78ms	2275.64ms
SPFA	33597.16ms	34628.33ms	34515.27ms	37637.92ms	24978.22ms	33071.38ms

Table 1: 最短路效率测试

可以看出，实测性能最好的是 A* 算法，其次是 Dijsktra 算法，SPFA 的速度最慢。

2.1.6 k-短路计算

在进行 k-短路计算时，我实现了两种算法。一种是 A* 算法，通过控制终点出队的次数得出第 k 优解；另一种算法是清华大学的俞鼎力同学在其论文中介绍的算法[2]，是对偏离算法（Yen 算法）进行了大幅度的复杂度优化。由于 A* 算法十分经典，所以下面主要介绍第二种算法。

偏离算法的思想就是第 k 优解一定可以在前 k-1 优解构成的路径树中进行边的替换完成（即偏离过程）。具体分为以下几个过程：

1. 将原图 G 中的边反向，构建从 t 点出发的最短路树 T。对原图中的每一条边 $e \in G$，维护

$$\delta(e) = w(e) + dis(tail(e)) - dis(head(e)).$$

其中 $tail(e)$ 表示 e 这条边的终点，$head(e)$ 表示 e 这条边的起点，$dis(v)$ 表示 v 到 t 的最短路径长度。对于以下 $\delta(e)$ 数组，有以下几个性质：

a) $\forall e \in T, \delta(e) = 0, \forall e \notin T, \delta(e) \geq 0.$

b) 对于一条 s 到 t 的路径 p，定义其中不再 T 中的所有边的集合为 $sidetracks(p)$。则对于 $sidetracks(p)$ 中的相邻两边 e、f，要么 tail(e) 和 head(f) 相同，要么 tail(e) 是 T 中 head(f) 的孩子。也即任一 s 到 t 的路径一定是由最短路树上的边和非最短路树上的边间隔而成。

c) $(p) = dis(s) + \sum_{e \in sidetracks(p)} \delta(e)$. 这个性质是本算法的核心，式子中的求和号代表了这条路径相对于最短路径的偏离量。

d) 确定 T 的情况下任意一个满足第一个性质的边序列 sidetracks 一定对应了唯一一条 s 到 t 的路径。显然如果给定了一个 sidetracks，将如果其中两条边不相邻，则通过添加 T 中的边使其相邻（即从 head(f) 到 tail(e) 的树上路径），因此可以唯一地将从 s 到 t 的路径还原出来。这个性质可以得出只要我们找出了第 k 小的偏移量（最短路偏移量为 0），也就找出了第 k 短路。

2. 依据偏移量从小到大构造一系列 sidetracks。最原始的方式是通过 bfs，利用以序列偏移量为关键字的优先队列，每次从优先队列中取出一个序列，令其最后一条边的起点为 v，将起点在 T 中 v 到 t 路径上的边每次加入一条到该序列中，构成一系列新的序列，将其加入到优先队列中。将上述过程重复 k 次即可。

以上就是偏离算法的主要过程，显然最后一步有很大的优化余地。首先满足“起点在 T 中 v 到 t 路径上的边”这个性质的边有很多条，但我们只要求第 k 短路，因此队列中有很多无用结点。实际上我们可以对每个点维护一个堆 $g(v)$ 保存所有满足该性质的边，每次更新时考虑最后一条边的终点 v 和倒数第二条边的终点 u，要么用 $g(u)$ 中下一条边替换倒数第二条边，要么将 $g(v)$ 中的第一条边加入到该序列中。由于空间有限，而且每次操作只有查询没有修改，因此我们只需要实现部分可持久化堆即可。具体实现是要维护两个可持久化堆，第一个堆 $h(v)$ 保存所有以 v 为起点不再 T 中的边，这一步可以 $O(m)$ 完成。第二个堆 $H(u)$ 以 $h(v)$ 为元素，以 $h(v)$ 中堆顶元素为关键字，建立 $H(v)$ 可以通过对 T 进行 dfs，每个节点的 H 堆将父亲结点的 H 堆和自己的 h 堆可持久化合并（相当于在父亲结点的 H 堆中插入一个元素），总复杂度 $O(n \log n)$，假设我们的可持久化堆可以在 $O(\log n)$ 时间完成合并。之后在计算第 k 短路时可以就是利用 $H(u)$ 的两个孩子结点更新序列，或者用 $H(v)$ 的堆顶元素更新序列。堆中元素访问可以 $O(1)$ 完成，所以计算第 k 短路的时间复杂度是 $O(k \log k)$。因此该算法的总复杂度是 $O(m + n \log n + k \log k)$。

可以看出该算法具有优秀的复杂度，特别是当 k 很大时，将远远优于 A* 的 $O(kn \log n)$ 的复杂度。但在 k 很小的时候要比 A* 慢。具体数据如下：

[2]俞鼎力，《堆的可持久化和 k 短路》，NOI2014 冬令营讲稿

100 组随机数据 $k \leq 100$	测试 1	测试 2	测试 3	测试 4	测试 5	平均值
A*	3895.58ms	3968.40ms	3829.00ms	3888.70ms	3956.67ms	3907.67ms
偏离算法	12868.78ms	13006.43ms	12696.57ms	12937.94ms	13166.77ms	12935.30ms

Table 2: k-短路效率测试

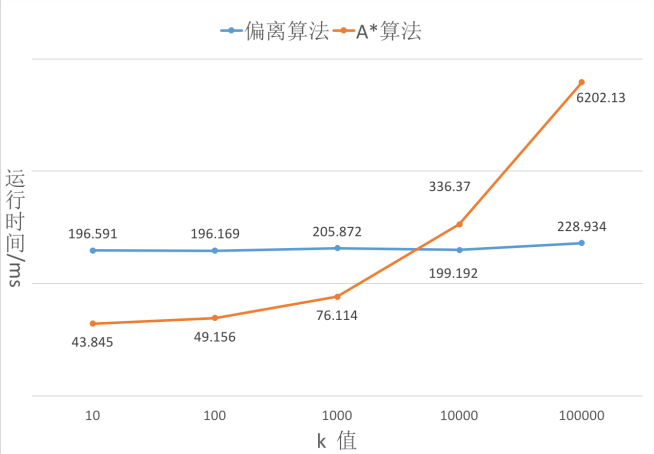

Figure 3: 随 k 的规模增长运行时间的变化

通过实验数据可以看出，偏离算法的运行时间几乎只和图的规模有关，即使询问的 k 的规模很大，也可以保持稳定的运行时间。但 A* 算法在 k 比较小的时候有巨大优势，但其运行时间的增长也几乎和 k 的增长成正比。

但是很遗憾，在地图上进行 k-短路计算有很多问题，比如 k 很大的查询将失去实际意义，同时在地图上存在一些长为 2 或 3 的小圈，会发现 k-短路算出的路径会在圈上不断转圈，这也是没有意义的路径。因此 MyMap 没有实现给定点 k 短路查询的 API，只是提供了一个随机数据测试的功能。但总而言之第二种算法确实是一个理论上非常优秀的算法，但其实际意义还有待开发。

2.1.7 地图对象信息查询

MyMap 可以支持对道路和建筑的汉语名称查询，包括全称或部分名字，但没有实现模糊查询。对于每条道路，可以显示出其 id 和长度，对于每个建筑可以显示出其 id。而且对于每个查询到的对象都可以在地图上进行高亮显示。同时 MyMap 还支持在地图上点击地图对象进行查询，每次点击一个位置可以高亮出距离这个位置最近的地图对象，并在主窗口中显示其信息。这一功能依靠 k-d 树维护近邻点查询，从而在查询结果中筛选出距离最近的地图对象。

2.2 组织和索引

2.2.1 概述

此部分由两个方面构成，分别是区域查询和近邻查询，共实现了两个数据结构（k-d 树和线段树），在实现上均对原始算法进行了一些改进。在实际实现上还有一点小问题，就是可能一个点距离一段路的距离很近，但距离这段路的端点很远，从而返回的结果将会忽略这条路，但显然这是不符合实际情况的。为了避免这种情况，我会扩大查询的范围，将查到的点对应的道路与查询点计算点到线段的距离，从而选出距离最近的 k 条道路。本节将分别就 k-d 树和线段树这两种数据结构在本部分中的实现和应用进行介绍。

2.2.2 k-d 树

k-d 树是一种经典的空间数据索引数据结构，可以很方便地进行区域查询和邻近查询，但一个问题就是如果经过大规模的插入和删除操作后将导致 k-d 树不平衡，从而使复杂度退化。对于这个问题，可以使用类

似替罪羊树的思想对 k-d 树进行维护，当检测到一个点的不平衡性（比较单个孩子的 size 和整个子树 size 的比值）超过一个阈值时，对整棵子树进行重新构建，从而可以在均摊意义下复杂度不退化。

2.2.3 线段树

线段树维护二维数据的经典算法是二维线段树。外层线段树以 x 为关键字，每个节点维护一颗线段树，以 y 为关键字。这样每次查询是可以先在外层线段树中查询给定区域的 x 范围，最多找到 $\log n$ 个节点，再在每个节点中查找给定区域的 y 范围。如果只是要把区域中的点找到的话不需要在内层维护一个线段树，只要一个有序的数组即可，每次在数组中二分即可找到对应元素。这样一次查询的复杂度是 $O(\log^2 n)$。

有一种叫“Fractional Cascading”的技术以空间换时间，将单次查询的复杂度减小到了 $O(\log n)$。[3]其主要思想是考虑每一次对内层数组的二分，所要二分的范围都是相同的，就是给定区域的 y 坐标范围，同时考虑线段树的结构，当前结点中所保存的点一部分被划归到了左子树，一部分被划归到了右子树，可以让当前结点的内层数组中的元素多保存两个指针，指向左右孩子的内层数组的位置，其意义就是如果当前结点二分范围的边界是这个元素，那么对左右子树进行二分时的边界就是其指针指向的元素。这个结构可以在线段树建树过程中自底向上构建出来。

2.2.4 效率分析

首先测试了 k-d 树的平衡性问题，测试方法是在原地图点的基础上选取了左上角小邻域每次插入 1000 个随机点，之后查询 10^5 次包含该邻域的矩形。同时还对平衡系数 α 进行了调整测试。运行结果如下：

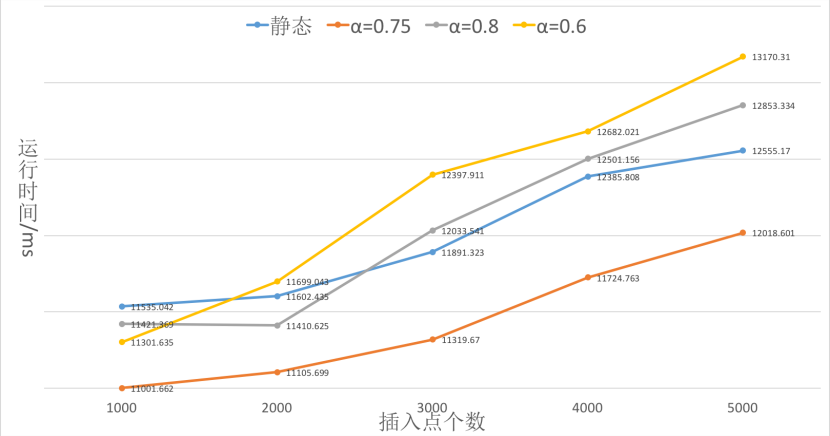

Figure 4: 插入易退化点后的查询时间

可以看出对 k-d 树进行部分重构后确实有一定的防退化效果，但这也和平衡系数的选取有关。通过实验数据可以看出在替罪羊树中常用的 $\alpha = 0.75$ 确实是一个表现优秀的参数。但是由于大规模的数据修改、添加或删除在实际生活中几乎很少发生，所以就实际情况来看很难让 k-d 树发生退化。而且由于 MyMap 每次都会从 xml 中读取数据，所以对于修改的情况可以直接通过修改 xml 文件实现，因此 MyMap 没有支持动态修改的功能。

同时我还对线段树和 k-d 树的查询效率进行了测试，对于线段树的实现，一种是直接内层二分，另一种使用了“Fractional Cascading”。共使用了三种测试数据，一是纯随机矩形，二是 500*500 的大矩形（占全图面积的 32% 左右），三是 10*10 的小矩形。运行结果如下：

随机查询	10^4 次随机区域（5 次平均）	10^4 次大区域（5 次平均）	10^5 次小区域（5 次平均）
k-d 树	8803.04ms	9419.87ms	470.71ms
线段树 $O(\log n)$	8819.20ms	9311.83ms	691.34ms
线段树 $O(\log^2 n)$	8748.13ms	9373.30ms	776.15ms

Table 3: 区域查询不同测试集运行时间

[3]Erik Demaine, Advanced Data Structures, http://courses.csail.mit.edu/6.851/spring12/lectures/L03.html

可以看出在纯随机和大范围查询的情况下，线段树 $O(\log^2 n)$ 查询的算法比较有优势，略快于另外两种算法。但对于小范围查询的情况，k-d 树具有明显的优势。这说明 k-d 树的常数要远小于线段树的常数。同时由于“Fractional Cascading”有一定常数，所以在实际表现上没有比直接的二分查询要优。但在小范围查询上可以看出 $O(\log^2 n)$ 算法的劣势。

2.3 挖掘和分析

2.3.1 概述

出租车的轨迹数据具有规模大、冗余度高、存在错误数据等特点，因此在处理这些数据时，我首先进行的工作是对数据的整理和筛选，从众多数据点中找出信息量高的数据点进行维护。同时，我还利用数据中给出的出租车载客状态信息，构建了一个打车指数的评估方法，表征一个地点的打车难易程度。最后我采用了均值漂移算法来为用户推荐附近容易打车的地点。因此本节将从关键点筛选和打车指数评估两个方面进行介绍。

2.3.2 关键点筛选

对于所给的出租车轨迹数据，我主要选择了编号、坐标、时间、载客状态这几个维度。由于文件体积过大，所以采用按出租车编号进行切割的方法，将 1G 左右的大文件分割成许多小文件，每次要查询一辆车的数据时从硬盘中进行读取。对于出租车的轨迹数据，含有许多冗余信息，比如司机将汽车长时间停在一个地方后其数据依然会被采样。同时一些路段由于无法上下车，所以在这些路段的行驶数据也不是很有意义。所以我目前只选取了上下车时的数据点，大约有十万个点，将其在地图上进行标记后结果如下图所示：

Figure 5: 关键点标记

可以看出在靠近上海市中心的地区打车的人非常多，越向边缘越稀少。同时打车次数的多少也反映了居民进行出行活动的频繁程度以及人口密度，可以看出浦西的人口密度要大于浦东。

但只处理上下车关键点会造成数据中的信息量大幅减少，同时关键点过少也不利于进行进一步的数据分析。因此结合下一步所要进行的工作，我对空车状态时采样的数据点进行处理，主要根据相邻点的距离进行筛选，这样可以有效地将一些无用信息（比如长时间停车时的采样点）剔除掉，而且还可以保留部分轨迹的信息。经过这一步处理，共保留了接近 10^6 个关键点，已经可以很方便地用数据结构进行维护。

2.4 打车指数评估

进行打车指数评估的目的主要是从实际生活出发，帮助用户判断一个地点拦到空车的难易程度。为了实现这项功能，我将筛选过的空车数据用 k-d 树进行维护。对于一个地点的打车难易程度，可以使用以这个地点为中心的小邻域内包含的空车状态的采样点数目，数目越多说明有空车经过的概率越高。我采用的是查询一个点为中心 10*10 的正方形内空车点的个数，经过测试可以发现这种评估方法可以在一定程度上正确表征打车难易程度，比如可以清晰地看到在火车站的进出站口打车指数较高，在复旦大学本部附近打车的话校门口的打车指数较高。另外一种评估方式是对这个点周围的第 k 个空出租车点距离此点的远近作为评估标准，

距离越近说明这个点附近的空车采样点越密集。我使用 k=50 进行计算，同样可以较好的表征打车的难易程度。

对于用户来说，仅仅查询一个点的打车指数数值并没有太大意义，因此 MyMap 实现的一项重要功能就是为用户推荐附近比较容易打车的地点。对于这个问题我采用了均值漂移（mean shift）算法，每次找出以当前点为中心 10*10 的正方形内的所有点，计算其重心，从而计算出漂移向量对点进行移动，这样重复迭代一定次数后回收敛到一个局部最优点。但最原始的均值漂移算法的收敛速度较慢，我使用了高斯核函数的方法，为每个点根据其到中心点的距离设置一个权重，从而较近的点权比较大，较远的点权重小，这样处理后再速度和准确性上都有所改善。同时由于高斯核函数的计算涉及到多次浮点运算，在迭代时比较耗时，因此可以采用对邻域进行网格划分后将第一次的计算结果保存下来，之后在同一格子内的点直接查询即可，从而在减少多次迭代时产生的重复计算。经过这些优化之后目前已经有不错的效果，可以给出一些比较符合常识的推荐（路边或路口）。但该算法还有一些问题，比如鲁棒性不是很好，会有一定概率收敛到比较差的点，同时并不适用于第二种评估方式，因为以第 k 辆车的远近进行评估将忽略小域内点的信息，从而产生错误的结果。另外一个问题是该算法没有考虑到人的移动能力有限，可能会在多次迭代后移动到一个较远的最优点。目前我采用的解决方法是用参数控制漂移的步长和迭代的次数，从而在最优性和可行性之间达到一个平衡点。

3 实验结论

在此次数据结构 PJ 中，本人开发的“MyMap”软件使用各类数据结构对上海地图信息和出租车轨迹信息进行维护，从而实现了较为有效的时空数据管理和查询。当然，这个项目还有许多可以改进的地方，比如对交通网络的精细处理（包括方向和限速等）、在地图上显示文字、在出租车轨迹数据中挖掘更多有效信息等。但由于时间有限，同时也受机器硬件条件的限制，目前这些方面没有去实现，可能将会作为本项目进一步开发的目标。

4 参考文献

1. 俞鼎力，《堆的可持久化和 k 短路》，NOI2014 冬令营讲稿
2. Erik Demaine, ”Advanced Data Structures”, http://courses.csail.mit.edu/6.851/spring12/lectures/L03.html
3. Wikipedia, ”Mean shift”, https://en.wikipedia.org/wiki/Mean_shift

5 附录：MyMap 说明文档

5.1 配置说明

本软件使用 Qt5 在 Mac OSX 10.11.1 下开发，使用了“Desktop Qt 5.5.0 clang 64bit”作为构建套件。本软件使用的第三方库包括 OpenCV2 和 pugixml（无需单独配置）。本软件在运行是需要“shanghai_map”地图数据文件和“taxi”文件夹下的出租车轨迹数据文件。本软件在地图绘制时的分辨率为 1000*800，在低于此分辨率的屏幕上运行会造成窗口显示不全等问题。另外在用 Qt 编译源码时请手动更改运行目录到源代码目录下，否则会找不到相关文件。

5.2 文件结构说明

本软件的全部代码均在主目录下，共包含 15 个头文件、13 个 C++ 源文件、1 个界面文件和 1 个 Qt 项目配置文件。除了第三方库 pugixml 外其余每个“.h”头文件对应同名“.cpp”源文件。主要文件的说明如下：

1. dataload ——读取并解析 xml 文件，提取地图数据中的有效信息。
2. draw ——涉及地图绘制，提供 API 实现不同类型的绘图功能。
3. findbyname ——对地图对象构建中文名称索引。
4. graph ——由地图信息构建拓扑图，方便进行最短路计算。
5. kdtree ——k-d 树的实现。
6. mainwindow ——涉及主窗体开发，包含了窗体内的各种 Qt 槽函数。
7. segtree ——线段树的实现。
8. spath ——最短路算法的实现。
9. taxidata ——涉及出租车轨迹相关的函数。
10. top-k ——k-短路算法的实现。

5.3 软件功能说明

本软件的所有操作均可在主界面和地图界面（界面会在需要时弹出）完成。由于本软件作为数据结构的 PJ，因此保留了一些测试用的按钮。相关按钮说明如下：

1. 地图绘制——会弹出地图界面，用键盘方向键移动地图，用’+’ 和’-’ 键放大和缩小。当用鼠标点击涂上一个点的时候，在主界面中会显示距离此点最近的地图对象的简要信息，并在地图上将此对象高亮显示，’q’ 键退出。
2. 地图上两点间最短路——在地图界面上点击选择两个点（右键可以取消选择），选择好后按’q’ 键会在地图上绘制这两点间的最短路，同时在主界面上也会显示其路径信息。
3. 区域兴趣点查询——操作方法同上，将会在主界面中显示该区域的信息。
4. k-近邻查询——即可以在地图上选择点，也可以输入查询点的 id，同时还要输入 k 的值。之后会在地图上显示查找到的相关点，具体信息会在主界面显示。
5. 查找——输入一个对象的中文名称后会显示地图中包含该名称的对象，单击列表中的元素会显示其基本信息，双击后会在地图界面将其高亮显示。
6. 查找出租车——输入出租车编号，或直接点击出租车列表中的元素即可查看该车的轨迹信息（包括行驶长度、采样时间、载客时间比例等），双击该元素会将轨迹信息绘制在地图上。
7. 我要打车——在地图上点击要打车的地点，主界面中会显示该点的打车指数，右键单击后将会在地图中显示 MyMap 推荐的打车地点，同时主界面也会显示该点的打车指数。

8. 其余按钮均是测试性能用按钮。

由于开发时间有限，难免会有一些 bug，还望谅解！欢迎大家发现问题后及时与我沟通交流[4]。

[4]作者邮箱：14307130222@fudan.edu.cn