互联网出行 实验报告

2015011308 计53 唐适之

#### 设计框架

本实现分为前端UI和后端算法两部分，前端使用Node.js实现，后端使用C++实现，后端在编译时直接与Node.js解释器链接（Node.js native module）而实现通信。

用户在前端显示的地图上点选出发地和目的地后，由后端求解。后端找出沿路网行驶绕路距离不超过10km的，离出发地最近的5辆出租车，求出其送达各乘客的最优路径，及绕路统计信息，返回给前端显示。

#### 算法

求解本问题的核心，在于求解路网中任意两点间的最短路。由于本问题中的路网是真实存在的，任意两点间的路网距离，一定不会大于其间沿地球表面的球面距离，故可以利用此下界对朴素的最短路算法进行剪枝。我对Dijkstra算法做出了如下修改：

若求解图中起点到终点的距离，经典的Dijkstra算法会在候选结点中，不断选择最小的结点进行增广，并将被增广到的结点纳入候选节点，直到所有结点均完成增广，其中等于已求得的到的最短路。可以看出，经典Dijkstra在选取增广点时，优先选择离起点近的结点，但忽视了其与终点的距离。因此，本实现中将Dijkstra改为选择 最小的进行增广，其中为到的球面距离。这样，选取增广点时，即考虑了其与起点间的准确距离，又考虑了其与终点间的估计距离，避免了向偏离终点的其他方向盲目增广。

如上所述，表示球面距离的函数实为到的路网距离的下界，因而此算法实际上是一个A\*算法。可以证明，其时间复杂度一定不会高于原Dijkstra算法。

有了求任意两点间路网距离的算法后，给定一辆出租车及其各个目的地，直接枚举目的地的排列，即可求出总路程最短的路径。计算不同排列的目的地的总路程时，可能需要重复计算某结点到的最短路。为避免重复计算，本实现先计算各目的地间两两的最短路，然后再据此计算目的地各种排列时的总路程。

当每个查询到来时，本实现使用朴素的Dijkstra求由查询中出发地出发的单源最短路径，并对增广到的每一辆出租车运行上述算法，根据上述算法求出的最优总路程计算绕路距离，保留绕路距离不超过10km的作为候选。此Dijkstra不必执行完，当求得5个候选或增广点离出发点超过10km时，即可停止算法。

实测表明，本实现的时间效率非常高。程序运行的主要性能瓶颈在于前端UI，而非后端算法。

#### 使用说明

见README.md。

#### 依赖的第三方库

本实现依赖以下第三方库，均用作前端UI。

* Electron：Node.js前端框架；
* AngularJs：JavaScript前端框架；
* JQuery：JavaScript前端框架；
* Bootstrap：前端样式库；
* 百度地图。