拼车场景下的路径规划问题

CS7310-Algorithm@SJTU, Group Project

2022 年 11 月 15 日

摘要

近年来，线上订车平台发展迅速，拼车业务及需求快速增长，其符合绿色低碳出行需求，是未来智慧交通的重要构成部分。在拼车场景下，多名乘客（最多三人）同时乘坐一辆车，平台如何在最小化成本的情况下满足所有乘客需求，涉及到订单分配、路径规划等多个问题。**多途经点路径规划问题**就是拼车场景下的基础优化问题，即给定某时刻一辆车的位置和至多三个乘客的上下车位置，要求给出一条串联司机和乘客上下车的最短路径。求解该问题是实时派单的前序条件，平台会根据规划算法拟定的路线确定最终派单方案。请仔细阅读本文档并完成相应任务。

关键词: 拼车, 路径规划，最短路径，多途经点，多候选点

1 拼车业务与路径规划

通过线上订车平台开展的“拼车”业务是近年来新兴的一种绿色低碳出行方式。乘客向平台发出用车请求，包含上车位置、下车位置，平台通过计算对两名或者三名乘客进行拼单，由一位司机沿途分别接载乘客并送至各自的目的地。本项目基于拼车场景，希望能够针对拼车路径规划问题开展算法设计与优化。以下分别介绍相关背景与项目需求。

1.1 基本模型与接乘顺序

当乘客发送乘车需求（订单）给订车平台后，平台需要将合适的订单合在一起（拼单），并帮助司机规划合适的接乘顺序 S 与行车路线。图1是一个拼车的典型示例，出租车 c（car）从位置 p_c

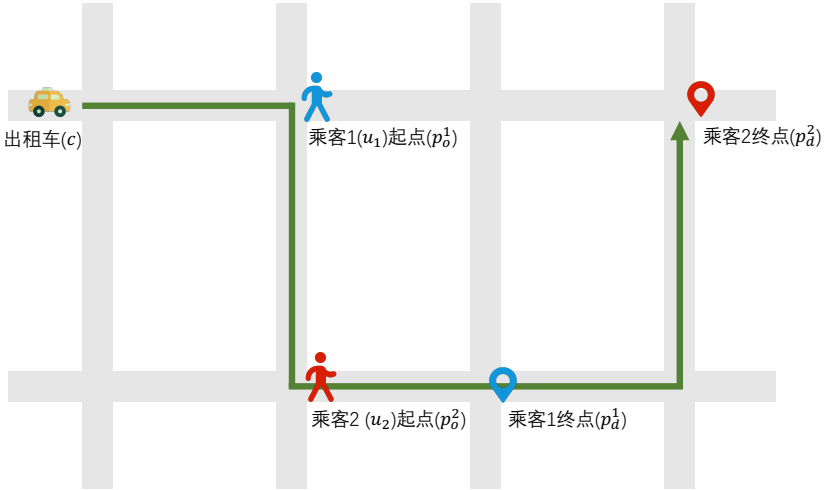

图 1: 双人拼车接乘顺序序列示例 $S_1 = [p_c, p_o^1, p_o^2, p_d^1, p_d^2]$

出发，依次从上车位置 p_o^1 和 p_o^2（起点）接乘两名乘客 u_1 和 u_2，接下来，分别在 p_d^1 和 p_d^2 两个位置（终点）让两名乘客下车，于是形成了一条行车路线序列 $S_1 = [p_c, p_o^1, p_o^2, p_d^1, p_d^2]$。

由于路网拓扑结构以及拼车乘客与车辆的相对位置关系，不同的接送顺序 S 会造成行车路线的巨大差异。如图2所示，如果交换乘客接送的顺序，形成路径序列 $S_2 = [p_c, p_o^2, p_o^1, p_d^2, p_d^1]$, 则其比 $S_1 = [p_c, p_o^1, p_o^2, p_d^1, p_d^2]$（图1）的路径总距离更长。

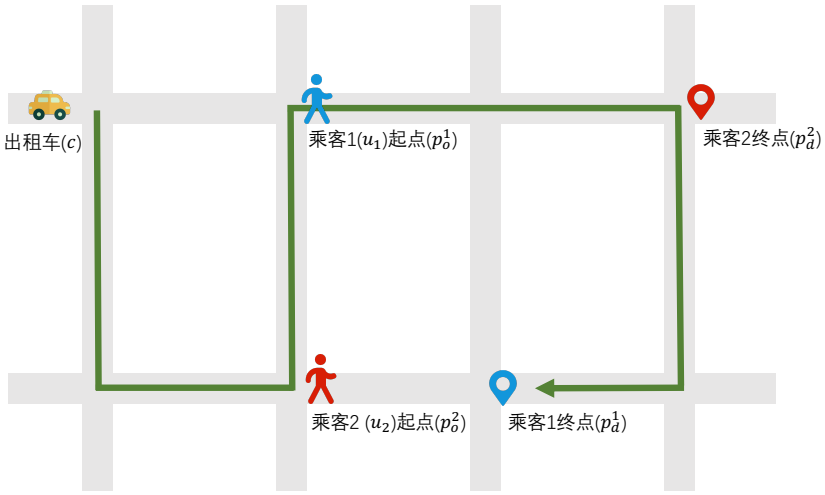

图 2: 双人拼车更改接乘顺序序列 $S_2 = [p_c, p_o^2, p_o^1, p_d^2, p_d^1]$

1.2 问题建模与相关定义

假设城市道路网络可简单抽象为有向图 $G = (V, E)$，其中 $v_i \in V$ 代表道路交叉点，$e_{ij} = (v_i, v_j) \in E$ 代表具体道路，其权值为道路长度 $w(v_i, v_j) \geq 0$。定义路径 $P = v_1 \to \cdots v_k$，其权重为 $w(P) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$。定义最短路径距离函数 $d(v_i, v_j) = \min\{w(P) \mid P\text{是从}v_i\text{到}v_j\text{的路径}\}$。注意，在有向图 G 中，从点 v_1 到点 v_2 的最短路距离 $d(v_1, v_2)$ 一般不等于从点 v_2 到点 v_1 的最短路距离 $d(v_2, v_1)$。

定义 1（多途经点匹配） 定义包括一辆车和多个乘客的一个“拼单”为一个**多途经点匹配**$\{c, U\}$，其中 c 代表车，U 代表乘客集合，可表示为 $U = \{u_i\}$，其中 u_i 是第 i 个乘客。平台可以获取车的位置信息 p_c 以及每个乘客的起点和终点 $\{p_o^i, p_d^i\}$。

定义 2（接乘顺序序列） 对于一个给定匹配 $\{c, U\}$，**接乘顺序序列** S 是一个由汽车位置和乘客的起终点组成的序列，来帮助司机完成拼单接乘需求。

根据业务逻辑，接乘顺序序列 S 需满足如下约束：

- 先序约束：在序列 S 中，p_c 应该是第一个元素，且对于任意乘客 u_i，起点 p_o^i 应在终点 p_d^i 前。
- 容量约束：在一个匹配中，分配给一辆车的乘客数为 2 或 3（乘客数为 2 的情况称为双拼，乘客数为 3 的情况称为三拼）。
- 拼车约束：每个乘客需要与至少一个其他乘客有共乘路段。严格定义为，对于**任意**乘客 u_i，存在 j 使得在 S 中 p_o^j 在 p_o^i 和 p_d^i 之间或 p_d^j 在 p_o^i 和 p_d^i 之间；或者反之，p_o^i 在 p_o^j 和 p_d^j 之间或 p_d^i 在 p_o^j 和 p_d^j 之间。如 $S = [p_c, p_o^1, p_d^1, p_o^2, p_d^2]$ 是不合理的，因为这是两个独立的乘车订单，乘客乘坐之间未有任何共享路段，不在本问题的讨论范围内。（注：本问题中假设给定的匹配符合拼车场景，平台会预先筛选出起点接近、终点接近的订单作为备选匹配。）

1.3 路径长度计算

为了计算一个拼单最终的路径长度，在实际场景下，车及乘客的位置会被垂直映射到最邻近的道路上得到映射点位置，令 rt 表示该点的路长占比，表示该点到道路起点的距离占道路长度的比例。如图3所示，c 被垂直映射到 (v_0, v_1) 上的点 p_c，若 $w(v_0, v_1) = 100$，$rt_{p_c} = 0.4$，则 p_c 到 v_0 的距离为 $w(v_0, v_1) \cdot rt_{p_c} = 40$。路径总长度基于车及乘客位置绑定的道路和路长占比计算。

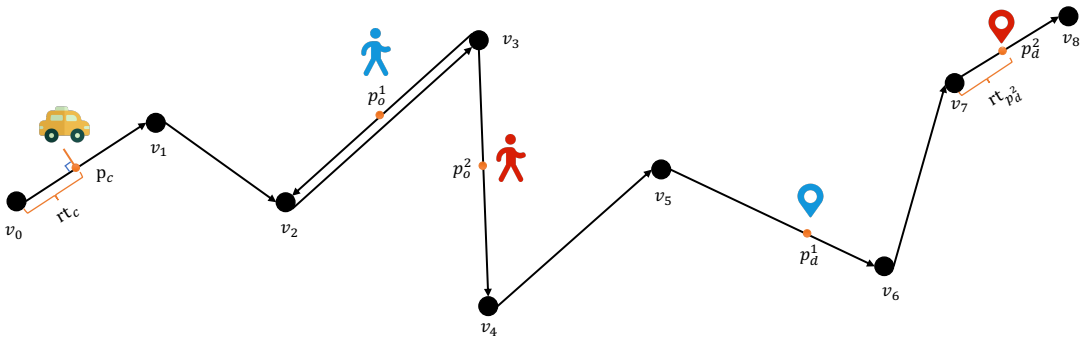

图 3: 绑路运算及路长占比示例

由于有向图 $G = (V, E)$ 上的最短路径是定义在点到点之间的，因此需要继续讨论如何把在道路（图 G 中的边）上的乘客绑定到道路交叉点上的问题。继续观察图3可以发现，如果车 c 想要接到乘客 u_1，实际需要走的路径为 $v_0 \to v_1 \to v_2 \to v_3 \to v_2 \to v_3 \to v_4 \cdots$，但是若直接把 u_1 绑定在该道路端点 v_2 或 v_3，都会导致规划路线为 $v_0 \to v_1 \to v_2 \to v_3 \to v_4 \cdots$，这与实际情况不符。于是需要进一步详细定义绑路计算的规则。

定义 3 (绑路计算规则) *当乘客 u_k 的位置映射到道路 (v_i, v_j) 之后，若以 u_k 为起点计算最短路，则将该乘客绑定至道路终点 v_j 上，并在最短路计算中加上 $w(v_i, v_j) \cdot (1 - rt_{p_o^k})$。反之，若以 u_k 为终点计算最短路，则将该乘客绑定至道路起点 v_i 上，并在最短路计算中加上 $w(v_i, v_j) \cdot rt_{p_o^k}$。*

最终，在确定接乘顺序序列 S 后，根据上述规则即可求出相应的最短路径，其路径长度是序列中两个相邻位置映射到道路后的最短路径长度之和。如图3的接乘顺序 $S_1 = [p_c, p_o^1, p_o^2, p_d^1, p_d^2]$，其最短路径应为公式(1)所示：

$$
\begin{aligned}
d(S_1) &= d(p_c, p_o^1) + d(p_o^1, p_o^2) + d(p_o^2, p_d^1) + d(p_d^1, p_d^2) \qquad (1)\\
&= w(v_0, v_1) \cdot (1 - rt_{p_c}) + d(v_1, v_3) + w(v_3, v_2) \cdot rt_{p_o^1} && \Rightarrow \text{计算} d(p_c, p_o^1)\\
&\quad + w(v_3, v_2) \cdot (1 - rt_{p_o^1}) + d(v_2, v_3) + w(v_3, v_4) \cdot rt_{p_o^2} && \Rightarrow \text{计算} d(p_o^1, p_o^2)\\
&\quad + w(v_3, v_4) \cdot (1 - rt_{p_o^2}) + d(v_4, v_5) + w(v_5, v_6) \cdot rt_{p_d^1} && \Rightarrow \text{计算} d(p_o^2, p_d^1)\\
&\quad + w(v_5, v_6) \cdot (1 - rt_{p_d^1}) + d(v_6, v_7) + w(v_7, v_8) \cdot rt_{p_d^2} && \Rightarrow \text{计算} d(p_d^1, p_d^2)
\end{aligned}
$$

比如，当计算 $d(p_c, p_o^1)$ 时，根据定义3，p_o^1 将绑定至道路 (v_3, v_2) 的起点 v_3，整个路径分成了三部分：(1). p_c 到所在边 (v_0, v_1) 终点 v_1 的距离；(2). v_1 到 v_3 的最短路径 $d(v_1, v_3)$；和 (3). p_o^1 所在边 (v_3, v_2) 起点 v_3 到 p_o^1 的距离。

1.4 假设、约束与目标

定义 4 (最优顺序序列) *对于给定匹配 $\{c, U\}$，存在多个满足先序、容量、拼车约束的接乘顺序序列 S，定义使得总路径长度最短的序列为*__最优顺序序列__$S^* = \arg\min_S d(S)$。

在实际应用场景下，行车路线的规划需要综合考虑路况、交规等因素选择总行车时间较短的路线。为简化问题，本项目以最小化行车路线总长度替代最小化行车时间作为优化目标，提出**多途经点路径规划**问题。在求解该问题时，常需调用最短路算法计算两点间距离。但因为真实路网拓扑存在百万级的点和道路，计算任意两点间的距离存在较大的计算开销。当存在海量拼车订单时，大量调用最短路算法计算两点间距离会对计算资源造成巨大压力。本着节约计算资源和加快响应速度的目的，本项目希望设计优化算法在求解最优顺序序列时减少对最短路算法的调用次数（下述复杂度分析均指分析对最短路算法的调用次数）。

定义 5（多途经点路径规划问题） 给定一个多途经点匹配 $\{c, U\}$，设计合理路径规划算法，在求得最优顺序序列 S^* 的条件下使调用最短路算法的次数最少。

注意 1： 真实业务中找到两点间最短路径及其长度的算法非常复杂，本项目中将作为黑箱给出，可以查表使用。请勿自行实现最短路算法来代替此黑箱。

注意 2： 真实业务中每次最短路算法仅能找到两点间最短路，比如分别寻找从 A 点到 B, C, D 三点的最短路需要 3 次调用。这一点也是黑箱特性，在本项目中不可更改。

注意 3： 求解多个接乘顺序序列所调用的最短路算法次数不等于逐个求解每个接乘顺序序列的调用次数之和。比如，为求解 $d(S_1)$，需要调用 4 次最短路算法分别计算 $d(p_c, p_o^1)$, $d(p_o^1, p_o^2)$, $d(p_o^2, p_d^1)$ 和 $d(p_d^1, p_d^2)$。但若同时求解 $S_1 = [p_c, p_o^1, p_o^2, p_d^1, p_d^2]$ 和 $S_2 = [p_c, p_o^1, p_o^2, p_d^2, p_d^1]$ 对应的 $d(S_1)$ 和 $d(S_3)$，则仅需调用 6 次最短路算法，而不是调用 8 次，因为 $d(p_c, p_o^1)$ 和 $d(p_o^1, p_o^2)$ 的结果可复用。

2 多候选点场景

事实上，乘客可以通过步行微调上下车位置来方便司机接乘，但这使得路径规划问题变得更加复杂。司机和乘客上车候选点的相对位置不同时会影响路径规划的结果。如图4展示了选择不同候选点作为上车点时司机的行车路线。当乘客在道路左侧上车时（图4上），汽车需要调头接乘客，然后再次调头驶向目的地，而当乘客在道路右侧上车时（图4下），汽车可以顺畅行使至目的地。可见，不合适的上车点会导致司机承担额外的调头成本，使行驶长度增加，并对后续的路径规划造成不良影响（如反向、绕路等）。

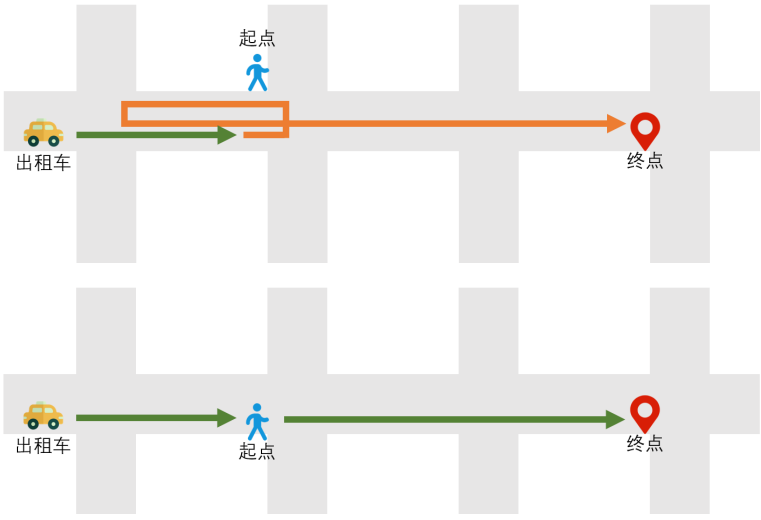

图 4: 不同上车候选点位置对规划路径的影响

为解决这一问题，对于订单中的乘客起点，平台通过位置计算和数据挖掘技术，可以给出一个该乘客的候选上车点集合 $P_o^i = \{p_o^{i(1)}, p_o^{i(2)}, \cdots, p_o^{i(k)}\}$。类似地，也可给出候选下车点集合 $P_d^i = \{p_d^{i(1)}, p_d^{i(2)}, \cdots, p_d^{i(j)}\}$。如图 5，对于序列 $S_1 = [p_c, p_o^1, p_o^2, p_d^1, p_d^2]$，如果选择 $p_o^{1(2)}$ 上车，会导致司机需要先沿 (v_2, v_3) 行驶，抵达点 v_3 后调头，再沿 (v_3, v_2) 行驶接乘客 u_1，抵达点 v_2 后调头，再向前行驶。而选择在 $p_o^{1(1)}$ 点上车，可以在总路径上节省 $w(v_2, v_3) + w(v_3, v_2)$ 的长度。这有效优化了路径规划的结果。

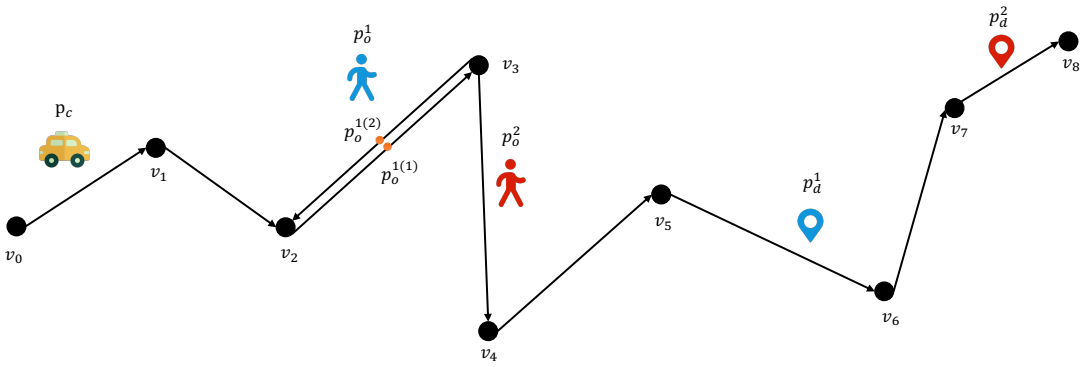

图 5: 多候选点对路径长度的影响

但是，多候选点的引入会导致求解 S^* 时对最短路算法调用次数的增加。如图 6，对于序列 $S_1 = [p_c, p_o^1, p_o^2, p_d^1, p_d^2]$，在不考虑多候选点时，为求得 $d(S_1)$，需要调用 4 次最短路算法。然而，在考虑多候选点集 $P_o^1 = \{p_o^{1(1)}, p_o^{1(2)}, p_o^{1(3)}\}$ 后，因为需要分别计算 p_c 到各个候选点的距离，以及各个候选点到 p_o^2 的距离，为求得 $d(S_1)$，总共需要调用 8 次最短路算法。

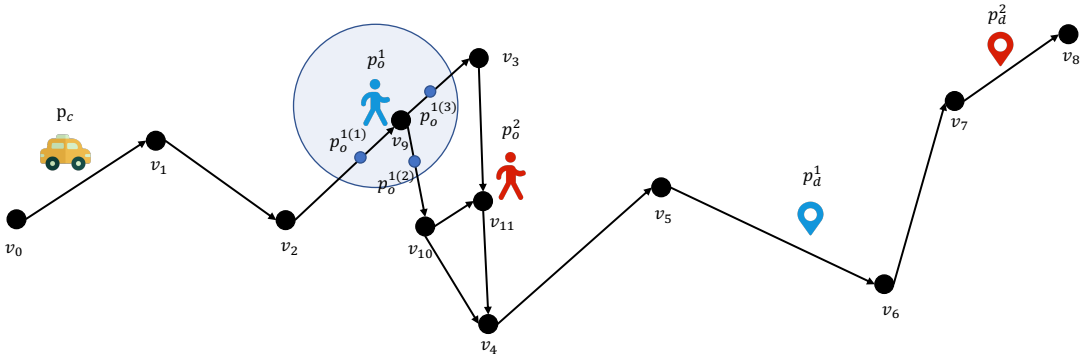

图 6: 多候选点对最短路算法调用次数的影响

由此，当给定每位乘客的候选点集合后，对于每一个多途经点匹配，项目进一步提出多途经点多候选点路径规划问题，如定义6所述。

定义 6 (多途经点多候选点路径规划问题) 给定匹配 $\{c, U\}$ 及每个 $u_i \in U$ 的 P_o^i 和 P_d^i 候选点集合，在求得 S^* 的条件下使调用最短路算法的次数最少。

3 符号表与附件说明

为方便运算，本章进一步整理了项目所需的符号、数据集和相关代码，其具体介绍如下。

3.1 符号表

表 1整理了项目所用到的符号及含义。

表 1: 符号与定义

符号	定义
$G=(V,E)$	路网有向图 G, 包括道路交叉点集 $V=\{v_i\}$ 和道路边集 $E=\{(v_i,v_j)\}$
$w(\cdot)$	权重函数，$w(v_i,v_j)$ 为边长度，$w(P)$ 为路径长度，$w(S)$ 为序列长度
$d(\cdot)$	最短路长度函数，$d(v_i,v_j)$ 为 v_i 到 v_j 的最短路长度，$d(S)$ 为序列最短路长度
$U=\{u_i\}$	乘客集合，u_i 是具体乘客 ($i\in\{1,2,3\}$)，有双拼和三拼两个情况
$\{p_o^i,p_d^i\}$	乘客 u_i 的起、终点
P_o^i, P_d^i	乘客 u_i 的上、下车候选点集合，$P_o^i=\{p_o^{i(1)},\cdots,p_o^{i(k)}\}$，$P_d^i=\{p_d^{i(1)},\cdots,p_d^{i(j)}\}$
c, p_c	车 c 与车所在的位置 p_c
S	接送驾顺序序列，如 $[p_c,p_o^1,p_o^2,p_d^1,p_d^2]$

3.2 附件数据说明

附件 1：“R1-link.csv” 本文档提供了成都市的简化路网信息，其中每一行数据描述了一条有向“原子道路”，从起点到终点是可行驶方向，“原子道路”指除端点外与其他道路无交叉点的道路，数据包括原子道路两端点的经纬度及实际路径长度。请注意，该数据包括单行道和双行道，如 link1(0,48) 和 link3052(48,0) 表示该路是双行道，link173(115,61) 表示该路是单行道。(link 的编号从 1 到 5943，和由 link 起点指向 link 终点的边等价，如 link1=$e_{0,48}=(v_0,v_{48})$。) 汽车调头仅可在满足行驶方向的合适路口（道路的端点），在非路口或非满足行驶方向的路口车辆均不可调头。

附件 2：“R2-distance.csv” 在实际优化问题中，需要调用最短路算法计算两点间距离。为简化问题，本文档给出了“R1-link.csv”所包含道路构成的路网中每两点间的最短路径距离，即可通过查表替代调用最短路算法来求得数值，故问题优化目标转换为最小化查表次数。为符合真实业务场景，**请勿通过自行实现最短路算法的方式来避免对此表的查询**。

附件 3：“R3-case-num.csv” 本文档提供了成都的 10 个拼车数据实例，其中每一行代表一个拼车实例，起/终点 ‘X’ 代表第 ‘X’ 个乘客的起终点信息，邻近 link 表示该乘客邻近的道路编号。若经纬度信息为 0 代表该乘客不存在。出租车经纬度代表出租车的当前位置，所在 link 表示车辆所在道路编号且行驶方向与对应道路的方向一致。其中 num 表示队号，每个队伍的数据不同，各有 5 个双拼实例和 5 个三拼实例。

3.3 附件代码说明

为方便计算，本项目附件同时也提供了部分 python 代码帮助，包括：

1. 数据读取子程序，将.csv 格式的数据读取为常见 python 数据结构。
2. 序列距离计算子程序，给定序列 S 时，该子程序可以计算出相应最短距离 $d(S)$。
3. 可视化子程序，将路线展现在地图上，生成.html 文件。

注意，以上都是可选附件，每个队伍可以采用任意自己喜欢的编程语言和相关工具完成本项目。

4 项目任务要求

任务一：最优顺序序列 S^* 求解

1. 在不考虑多候选点的情况下（即每个乘客都给定唯一的起点和终点），分别列举出在双拼和三拼场景时多途经点匹配 $\{c, U\}$ 所有可能的接乘顺序 S 以及基本情况下求解 $d(S)$ 所需的最短路查表次数。注意，根据查表规则，此处需列出累计查表次数，其输出格式如表2所示。

表 2: 双拼可能顺序序列及查表次数

编号	序列顺序	累计次数
1	$p_c p_o^1 p_o^2 p_d^1 p_d^2$	4
2	$p_c p_o^1 p_o^2 p_d^2 p_d^1$	6
...	...	...

若表格内容过长，可采用双栏表撰写结果，如表3所示。也可采用其他形式展现这些内容。

表 3: 三拼可能顺序序列及查表次数

编号	序列顺序	累计次数	编号	序列顺序	累计次数
1	$p_c p_o^1 p_o^2 p_o^3 p_d^1 p_d^2 p_d^3$	6	21	$p_c p_o^2 p_o^3 p_o^1 p_d^2 p_d^1 p_d^3$	...
2	$p_c p_o^1 p_o^2 p_o^3 p_d^1 p_d^3 p_d^2$	8	22	$p_c p_o^2 p_o^3 p_o^1 p_d^2 p_d^3 p_d^1$	...
...	...	...	...	...	...

2. 根据附件 “R1-link.csv” 提供的成都路网信息及附件 “R3-case-num.csv” 提供的成都拼车实例，计算每个实例的最优顺序序列 S^* 及其相应的长度。请描述算法设计思路并给出结果表格，示例如表4所示。

表 4: 成都拼车实例计算表

编号	场景	序列顺序	S^* 长度
Case01	双拼	$p_c p_o^1 p_o^2 p_d^1 p_d^2$	...
...	...	...	...
Case06	三拼	$p_c p_o^2 p_o^3 p_d^2 p_d^3 p_o^1 p_d^1 p_d^3$	...
...	...	...	...

任务二：多途经点路径规划算法设计

针对附件 “R3-case-num.csv” 中的实例，在不考虑多候选点情况下优化任务一中类似枚举的算法，在求出 S^* 的情况下最小化调用最短路算法的次数。如果你设计的算法无法总是求出最优解，则尽可能缩小算法结果与最优解之间的差距（以百分比计算，如 105%）。给出算法设计思路与对应数值结果，示例如表5。【提示：可以通过球面距离、方向角等指标去除不合理的顺序从而减少对最

短路算法的调用次数。如图 7，假设 $S_1 = [p_c, p_o^1, p_o^2, p_d^2, p_d^1]$，则 $angle(p_o^2, p_d^2, p_d^1)$（图中 $\angle\alpha$）小于 45°，表明可能存在到达后反向行驶造成绕远。当存在有更好方向角的顺序时，可将 S_1 剪枝掉，从而不用计算 $d(p_o^2, p_d^2)$。本任务也鼓励其他剪枝或优化方法。】

表 5: 成都拼车实例计算表

编号	场景	S 序列顺序	S 长度	与 S^* 比值（%）	查表次数
Case01	双拼	$p_c p_o^1 p_o^2 p_d^1 p_d^2$	...	...	...
...	...	...	...	...	...
Case06	三拼	$p_c p_o^2 p_o^3 p_d^2 p_d^1 p_d^1 p_d^3$	...	...	...
...	...	...	...	...	...

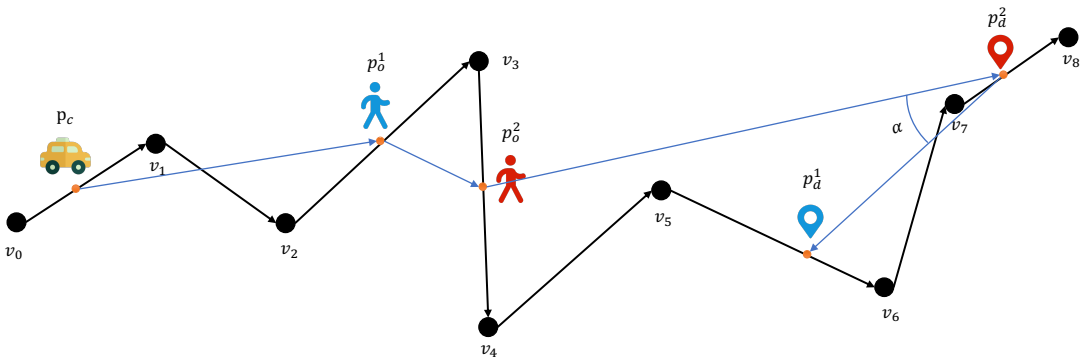

图 7: 剪枝优化示例

任务三：多途经点多候选点路径规划算法设计

试针对附件“R3-case-num.csv”中双拼和三拼的第一个实例（Case01、Case06）分析选择不同候选点作为上下车位置对路径规划的影响。并针对多候选点场景设计算法在求出 S^* 的情况下最小化调用最短路算法的次数。需给出候选点集合中合适的具体上下车位置，每个实例的最短行驶路径总长度，接乘顺序和调用最短路算法的次数。【提示：可以通过对 case 的分析删除不合适的候选点。本任务也鼓励其他方法。】

任务四：可视化分析

以图示方式针对附件“R3-case-num.csv”中双拼和三拼的第一个实例（Case01、Case06）呈现求解结果（可基于提供的 Python 地图可视化代码）。每个实例需要呈现以下四种场景的结果：

1. 不考虑候选点情况下，最优顺序序列 S^* 的路径。
2. 不考虑候选点情况下，任务二算法求得序列 S 的路径。
3. 考虑候选点情况下，最优顺序序列 S^* 的路径。
4. 考虑候选点情况下，任务三算法求得序列 S 的路径。

根据可视化呈现，进一步分析算法与对应的结果。

5 报告要求

每个队伍请提交一份项目报告，满足如下要求：

1. 报告应包括标题、作者姓名、学号、电子邮件地址、页眉、页码、任务的结果和讨论、仿真的图片、讨论和比较的表格，以及相应的图表标题。
2. 报告应结构清晰，按章节划分。
3. 报告应包括参考部分和致谢部分，也可以在致谢部分表述你的感受、建议和评论。
4. 报告应明确定义变量，并将其添加到符号表中。

Acknowledgements

This problem is motivated from a real-world corporation for MaaS. It is formulated by Prof. Xiaofeng Gao (gao-xf@cs.sjtu.edu.cn) from Department of Computer Science and Engineering at Shanghai Jiao Tong University. Yucen Gao drafted the project requirements and provided the data. Yulong Song tested the map and case data. Jiale Zhang and Yang Luo provided solutions and codes. Hengyu Ye helped on proofreading.