湖南省人民政府学位委员会办公室
湖南省教育厅学位管理与研究生教育处

2017 湖南省研究生数学建模竞赛参赛承诺书

我们仔细阅读了湖南省研究生数学建模竞赛的竞赛规则.

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们授权湖南省研究生数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

我们参赛选择的题号是（从组委会提供的试题中选择一项填写）：A

我们的队号为（填写完整的队号）：201718001014

所属学校（请填写完整的全名）：国防科技大学

参赛队员（打印并签名）：

1. 籍然
2. 戴鑫志
3. 侯振宇

指导教师或指导教师组负责人(打印并签名)：

日期：2017 年 4 月 25 日

评阅编号（由组委会评阅前进行编号）：

2017 湖南省研究生数学建模竞赛

编 号 专 用 页

评阅编号（由组委会评阅前进行编号）：

评阅记录（可供评阅时使用）：

评阅人										
评分										
备注										

湖南省第三届研究生数学建模竞赛

题 目 出租车合乘业务系统设计

摘 要：

本文对出租车合乘业务进行研究，设计出高效合理的合乘方案及相应算法。该方案使乘客等待时间尽量短，所需出租车数量尽量少，同时避免因合乘产生大量绕行距离，并兼顾乘客、司机双方的经济利益。

针对问题一，本文首先基于乘客和司机的经济利益诉求、时间利益诉求，推导出多名乘客能够实现合乘的条件。利用该合乘条件设计合乘方案，并借鉴贪心算法的思想，安排出租车依据“就近原则”搭载乘客，降低乘客等待时间；同时吸收层次聚类的思想，将“第一批乘客”进行分类，尽量安排能够合乘的乘客搭乘同一辆出租车，从而减少所需出租车的数量。此外在设计方案时，为进一步优化合乘方案，提出“双向确认策略”，避免“多车‘竞争’问题”；设计基于遍历思想的最短路径搜索策略，优化合乘路径选择。

针对问题二，本文以“合乘时乘客的支出不能高于单独乘车时的支出”以及“合乘时司机的收入不能低于非合乘时司机的收入”为原则设计计费方法。该方法引入利益分配因子，在乘客与司机之间分配因合乘而节约的总费用；利用每名乘客所占路程比例，在不同乘客之间分配因合乘而节约的费用，从而达到乘客与司机、乘客之间的利益平衡，进而调动乘客、司机双方合乘的积极性。

针对问题三，本文将设计的合乘方案与计费算法，应用于某一具体场景。在车速取 60km/h、乘车距离门限为 20m、等待容忍时间门限为 10 分钟、绕路容忍门限为 30%的条件下，合成方案实际使用出租车 541 辆，节省出租车 116 辆，乘客平均等待时间 0.26 分钟，具体分配方案见附件。此外，本文还通过数据分析对方案的合理性高效性进行了验证。

关键词：合乘方案 拼车 等待时间 出租车数量 费用

出租车合乘业务系统设计

1 问题重述和分析

出租车合乘业务是指路线相同或相近的两位或多位乘客共同乘坐同一辆出租车出行，系统根据合乘人数、乘车时间、实际路线等因素，分别计算出每位乘客的车费（通常低于各自独乘时的车费）。司机收入则为所有乘客支付的车费总和。该业务可以在不增加运营车辆总数的情况下提高运力，有助于缓解打车难，而且能够降低乘客出行成本，同时提高司机收入。因此，相当一部分乘客、司机愿意接受该业务，特别是在打车的高峰时段。

某出租车公司拟开展合乘业务。通过调研发现，其他城市或公司的合乘业务主要有以下两种模式。

1) 相同起点模式。合乘乘客必须在同一地点上车，顺路去往相同或不同的目的地。合乘乘客各支付共同路段车费的 a%（两人合乘时，$50 < a < 100$，如北京 60，烟台 70，重庆 80）。不允许一人先上车，中途再招揽其他乘客。该模式只需要改造计价器，不需要其他软件支持，并且计费原则简单明了。但该模式构成合乘的条件较严格，合乘比例偏低。
2) “一口价”模式。利用网上调度系统和手机打车软件，在同意合乘的前提下，乘客通过手机软件提交打车请求（起始位置等信息），系统根据历史数据预估车费，显示为“一口价”，即乘客若接受该报价，则无论实际乘车过程中是否有合乘，均按此一口价结算。该价格一般低于正常的车费。系统针对当前打车需求信息，动态调度合乘路线。该模式对乘客友好，便于控制乘车费用，而且合乘条件低，合乘方案灵活，可以提高合乘比例。但该模式可能出现绕行，引发乘客不满。若全程无法构成合乘，影响司机收入，引发司机不满。

本文将尝试设计新的合乘模式，充分调动乘客、司机等各方参与合乘的积极性（假设不需要考虑对公司现有调度平台和手机打车软件的改造难度），完成以下任务：

问题一：设计高效的合乘方案及其相应算法，使乘客等待时间尽量短，所需出租车数量尽量少。

分析：乘客的等待时间主要取决于出租车与该乘客之间的距离，乘客距离可搭乘的出租车越近则乘客等待的时间就越短。因此在安排车辆时，需要尽量安排距离乘客最近的出租车载客。要使得所需的出租车数量尽量少，就需要安排尽量多的乘客合乘，从而避免本可以合乘的乘客乘坐不同的出租车，导致车辆浪费。上述两个要求实际存在一定矛盾，一方面要使得乘客等待时间尽量短，就应该安

排更多的出租车去接附近的乘客，这导致了出租车的使用量增加。另一方面，要减小出租车使用量，就需要安排尽量多的乘客合乘，这有可能会导致部分乘客的等待时间延长。因此，设计合乘方案时需要综合考察这两个需求，找到两种需求的平衡点。

问题二：设计与合乘方案相应的合理的车费计算方法。

分析：车费计算方法是合乘方案需要解决的一个关键问题。车费计算时必须遵循两个基本的原则：1）不能损害乘客的合乘积极性，即合乘时乘客的支出不能高于单独乘车时的支出。2）不能损害司机的合乘积极性，即合乘时司机的收入不能低于非合乘时司机的收入。上述两个条件也是达成合乘的先决条件。因此在设计计费方法时，要照顾到乘客和司机双方的经济利益。

问题三：假设某城市的路网为正方形网格，网格边长 500 米，道路均可双向行驶。附件 1 是该城市某日某时刻之前 3 分钟内的打车需求数据，附件 2 是当前空驰出租车的位置信息，请按照你们设计的合乘方案给出具体的计算结果，并按附件 3 中指定的格式输出结果。假设，附件 2 中未涉及的出租车在计算合乘方案时不可用（非空驰，不考虑），并且合乘方案不要求用到附件 2 中的全部出租车，即，允许部分出租车仍保持空驰状态。乘客等待时间均以当前时刻为 0 时刻开始计时，即，不考虑之前 3 分钟内已经等待的时间。

分析：这是一个静态合乘问题，需要针对当前需求信息为乘客分配车辆。由于所有乘客的打车需求信息已经收集，只需要利用之前设计的出租车合乘系统进行车辆和人员的安排即可。此外，还需要结合“正方形网格”道路特点，采用“街区距离”而不是直线距离表示两点之间的距离。

2 假设和说明

1) 调度平台能够实时获取每辆出租车的状态，包括出租车当前位置，当前载客人数，当前行驶目的地。
2) 在满足合乘的条件下，用户总是愿意拼车。
3) 所有的出租车都是以一定的速度匀速行驶的，不考虑堵车以及等待红绿灯的情况。经查阅相关资料，出租车行驶速度定为 60Km/h。
4) 考虑到合理性以及节约出租车资源，如果某一乘客的出发地和目的地之间距离小于最小的乘车距离门限，则不安排车辆接该乘客。本文设置最小乘车距离为 20 米。
5) 为了减少出租车空载时间，如果与出租车距离最近的乘客距离该出租车超过了最远距离门限，则不安排该出租车接客。本文设该门限为 10Km。
6) 所有乘客都有一个相同的乘车等待容忍时间门限。本文设该门限为 10 分钟。

7） 所有乘客都存在一个绕路容忍时间门限，即当合乘的绕路时间在乘客的绕路容忍时间门限内，乘客会选择合乘。反之，乘客不会选择合乘。本文设该门限为乘客直达时间的 30%。

8） 进行合乘安排时，只考虑当前发起乘车需求的乘客，不考虑之后其他时刻的乘车请求。

3 符号说明

符号	含义
Fee_{share}	乘客合乘时的支出
Fee_{alone}	乘客单独乘车时的支出
$Income_{share}$	司机合乘时的收入
$Income_{alone}$	司机单独载客时的收入
L	多名乘客合乘时的路径
$Distance_{share}$	多名乘客合乘时路径的距离
$T_{waitting}$	乘客等待时间
$T_{waitting\_per}$	乘客等待时间容忍门限
$T_{redundant}$	合乘时乘客对于绕路的容忍时间
$T_{redundant\_per}$	合乘时乘客对于绕路的容忍时间门限

4 模型的建立、分析与求解

4.1 合乘方案及其相应算法设计

正如本文第一章问题分析所提及，减少乘客的等待时间与减少出租车的使用量之间存在一定矛盾。但考虑到调度平台设计目的在于提高乘客出行率，不能为了减少出租车使用量而让部分乘客不能乘车或者等待乘车时间过长。其次，从司机的角度讲，司机希望自己空载的时间越短越好。因此，我们首先考虑让乘客尽快上车，在此基础上考虑减少出租车的使用量。

为了减少乘客的等待的时间，调度平台应尽量安排出租车搭乘与之距离最近

的乘客，即出租车根据就近原则接客。为了减小出租车的使用量，调度平台应尽量采取合乘的方式，将满足合乘条件的乘客安排到同一辆车。因此，合乘方案的总体设计思路是：基于贪心算法的思想，调度平台依据就近原则，安排每辆空载出租车首先搭乘距离其最近的乘客（即第一批上车的乘客），比如出租车 a 首先依就近原则接乘客 A。接到乘客 A 后，出租车 a 在其当前位置（即乘客 A 起点）再根据就近原则，去接距离其最近的乘客 B，但此时需要先进行一次判断，即判断乘客 B 能否与车上的乘客 A 合乘，如果能够合乘，则乘客 B 上车，否则乘客 B 不上车。该车不断按此法搭载乘客，直到所有座位坐满或没有乘客满足合乘条件。同时，调度平台不断安排出租车按此法搭载乘客，直到所有的乘客都安排完毕。合乘方案总体设计思路如图 1 所示。

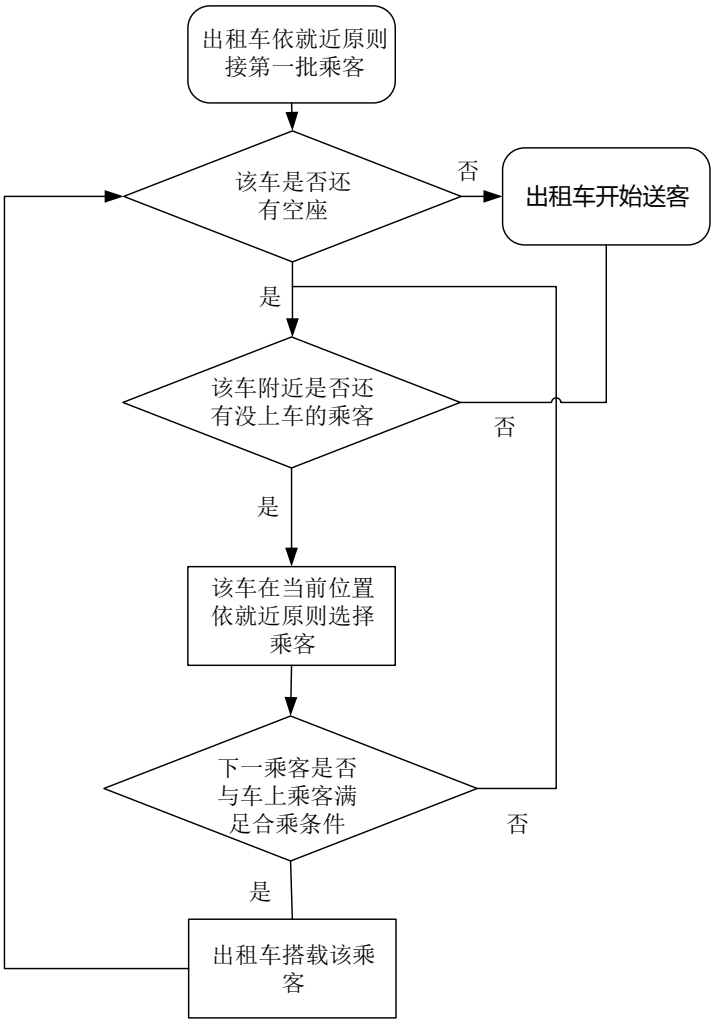

图 1 合乘方案总体设计思路

上述的合乘方案的总体设计思路，在具体实现时还需要解决以下四个具体问题：

◆ **多车“竞争”问题**：在安排出租车去接距离其最近的乘客时，可能会出现“竞

争”问题，即距离两辆或者多辆出租车最近的乘客是同一个人，这时应该安排哪一辆车去接该乘客？

- **合乘条件判定问题**：合乘方案及其相应算法的基础在于合乘，那么就需要确定哪些乘客可以合乘，即不同乘客能够合乘的条件是什么？
- **第一批乘客分类问题**：根据合乘方案总体思路，第一批上车的乘客是根据就近原则确定的，但是并没有判断这些乘客是否可以合乘。如果第一批乘客中存在可以合乘的乘客，但是调度平台会将这些乘客分配到不同车租车，从而增加所需出租车数量，进而造成出租车资源的浪费。因此需要对第一批上车的乘客进行分类，挑选可以合乘的乘客，将他们分配到同一辆出租车，提高出租车利用率。
- **最短路径搜索问题**：在出租车合乘时，有多条不同的乘客接送路径。相同情况下出租车应该选择最短的路径，以减少乘客的等待时间及路上行驶时间，那么怎样找到所有可能路径中最短的路径？

4.1.1 多车“竞争”问题

由于我们采用“车找人”的策略，距离两辆或者多辆出租车最近的乘客可能是同一个人。但一名乘客只能搭乘一辆出租车，此时采用就近原则就会出现多车竞争同一乘客的问题。

采用双向确认策略可以避免“竞争”问题。假设距离出租车 c 最近的乘客是乘客 A，则需要判断距离乘客 A 最近的出租车是否为出租车 c。如果判断为真，则完成了双向确认，调度平台安排出租车 c 去接乘客 A。如果判断为假，则出租车 c 暂时不做安排。双向确认的策略的示意图如图 2 下所示。

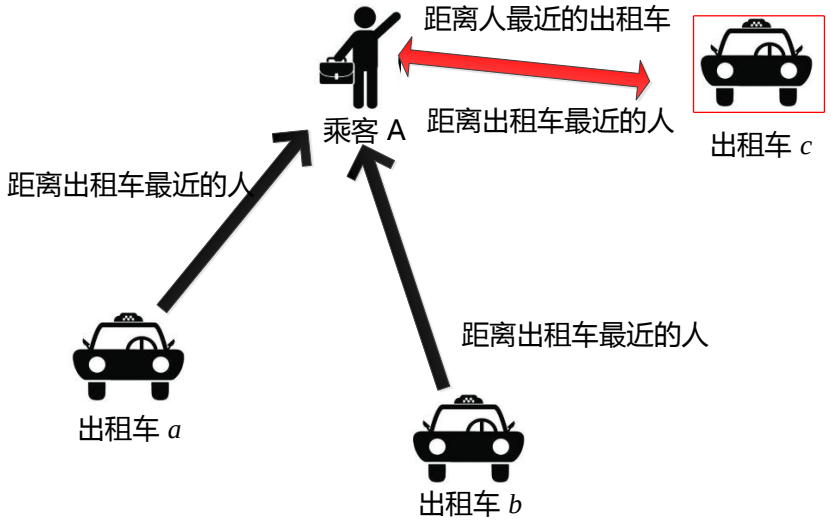

图 2 双向确认示意图

4.1.2 合乘条件判定问题

在考虑合乘条件时，需要考虑乘客与司机双方的利益诉求，只有双方利益诉求都得到满足时，才具备合乘的条件。乘客的利益诉求包括乘车费用和乘车时间。对于乘车费用，乘客希望自己合乘时的乘车费用 Fee_{share} 小于自己单独乘车时的乘车费用 Fee_{alone}；对于乘车时间，首先乘客不会允许等车时间过长，即等车时间 $T_{waitting}$ 要在等待容忍阈值 $T_{waitting\_per}$ 内，其次，乘客也不会允许因绕路耽误的时间过长，即由于绕路而所耽误的时间 $T_{redundant}$ 要在乘客的绕路容忍时间 $T_{redundant\_per}$ 内。司机的利益诉求比较简单，即载客收入。司机希望有乘客合乘时收入 $Income_{share}$ 能够高于单独载客时的收入 $Income_{alone}$。由于乘客利益诉求与司机利益诉求相互关联，为便于分析，下文中，我们将乘客与司机的利益诉求分为经济利益诉求和时间利益诉求两类（如表 1 所示）。

表 1 乘客司机双方利益诉求表

	经济利益诉求	**时间利益诉求**
乘客	乘车费用低	耽误时间少
司机	载客收入高	--

4.1.2.1 经济利益诉求

以两个乘客 i 与乘客 j 合乘为例。乘客 i 和乘客 j 的经济利益诉求为

$$\begin{aligned} Fee_{i\_share} &< Fee_{i\_alone} \\ Fee_{j\_share} &< Fee_{j\_alone} \end{aligned} \tag{1}$$

司机 k 的经济利益诉求为

$$Income_{k\_share} > Income_{k\_alone} \tag{2}$$

乘客 i 与 j 合乘时的总乘车费用正好是司机 k 的收入，即

$$Fee_{i\_share} + Fee_{j\_share} = Income_{k\_share} \tag{3}$$

由乘客的经济利益诉求（1）可以得出：

$$Fee_{i\_share} + Fee_{j\_share} < Fee_{i\_alone} + Fee_{j\_alone} \tag{4}$$

结合式（2）、（3）、（4）可以得出：

$$Income_{k\_alone} < Income_{k\_share} < Fee_{i\_alone} + Fee_{j\_alone} \tag{5}$$

式（5）即达到合乘条件时必须满足的经济利益诉求。

假定乘客 i 与 j 单独乘车时车辆行驶的距离分别为 $Distance_{i\_alone}$ 与 $Distance_{j\_alone}$，两人合乘时车辆行驶的总距离为 $Distance_{ij\_share}$（单位 Km，该路径与合乘时路径规划有关，在 4.1.4 节我们将讨论如何找到搭载合乘乘客前往目的地的最短路径），出租车每公里的收费为 α（单位元，这里暂时不考虑起步价）。则由式（5）右半边可得：

$$\alpha Distance_{ij\_share} < \alpha\left(Distance_{i\_alone} + Distance_{j\_alone}\right) \tag{6}$$

进一步可简化为

$$Distance_{ij\_share} < Distance_{i\_alone} + Distance_{j\_alone} \tag{7}$$

式（7）明确的表明：要满足经济利益诉求，合乘的行驶路线要小于非合乘时的各条路线之和。因此，在设计合乘算法时，我们将满足式（7）作为合乘条件之一。

两个乘客合乘的情况可拓展到三个乘客合乘的情况。这里直接给出三人合乘时，要满足的经济利益诉求条件：

$$Distance_{ij\_share} < Distance_{i\_alone} + Distance_{j\_alone} + Distance_{r\_alone} \tag{8}$$

4.1.2.2 时间利益诉求

以乘客 i 与 j 合乘为例。对于等待时间，两名乘客的时间利益诉求为

$$\begin{aligned} T_{i\_waitting} &< T_{waitting\_per} \\ T_{j\_waitting} &< T_{waitting\_per} \end{aligned} \tag{9}$$

对于绕路浪费的时间，两名乘客的时间利益诉求为

$$\begin{aligned} T_{i\_redudant} &= \frac{Distance_{i\_share} - Distance_{i\_alone}}{v} < T_{i\_redudant\_per} \\ T_{j\_redudant} &= \frac{Distance_{j\_share} - Distance_{j\_alone}}{v} < T_{j\_redudant\_per} \end{aligned} \tag{10}$$

式中，$Distance_{i\_share}$ 表示乘客 i 合乘时从起点到终点经过的路径。该路径与合乘时路径规划有关，在 4.1.4 节我们将讨论如何找到搭载合乘乘客前往目的地的最短路径。v 表示车辆的行驶速度。需要注意的是，每个乘客的绕路容忍时间可能是不同的，比如对于直达时间需要 2 小时的乘客，20 分钟的绕路耽误时间一般

是可以容忍的，但对于直达时间仅需要 5 分钟的乘客，20 分钟的绕路耽误时间一般就无法接受。在系统设计时，我们使用乘客直达时间的 30%作为绕路容忍时间门限。

将上述的分析过程可推广到三名乘客合乘时需要满足的时间利益诉求，故不再赘述。

4.1.2.3 合乘条件总结

针对 4.1.2.1 和 4.1.2.2 的分析，不同乘客能够同时满足经济利益诉求（即式（7）或（8））和时间利益诉（即式（9）和（10）），才能达到合乘条件。用伪代码可将判断乘客是否满足合乘条件的算法表示如下：

算法 1 合乘条件判定算法

Input：

乘客 1，乘客 2，…

合乘路线 L

Output：

合乘标志 flag，flag 为 1 表示可合乘，为 0 表示不能合乘

0 Begin

1 根据 L，计算各乘客等待时间：$T_{1\_watting}$， $T_{2\_watting}$， …

2 if $\{T_{1\_watting}, T_{2\_watting}, ...\} < T_{waitting\_per}$

3 if $\{T_{1\_redudant}, T_{2\_redudant}, ...\} < T_{redudant\_per}$

4 if $\{Distance_{1\_alone} + Distance_{2\_alone} + \cdots\} > L$

5 输出：flag = 1，退出

6 else

7 输出：flag = 0，退出

8 else

9 输出：flag = 0，退出

10 else

11 输出：flag = 0，退出

12 End

4.1.3 第一批乘客分类问题

第一批上车的乘客根据双向确认的策略选择。但是如果直接安排出租车接客，而不考虑这些乘客中是否存在满足合乘条件的乘客，那么就会造成出租车的浪费。

而后续安排出租车接客时系统都会对后续上车乘客进行合乘条件判断，故该过程不会产生车辆浪费的现象，因此需要对第一批上车的乘客进行分类以避免浪费。

一般情况下，每辆出租车最多可以坐三人。因此，将第一批乘客划分为三类：第一类为只能单独乘车的乘客；第二类为只能够两人合乘的乘客；第三个类为可以三人合乘的乘客。通过将第一批上车的乘客分类，优化出租车调度方案，可以最大限度的减少出租车的使用量，提高使用效率。分类方法借鉴层次聚类的思想，分三步进行（如图 3 所示）：

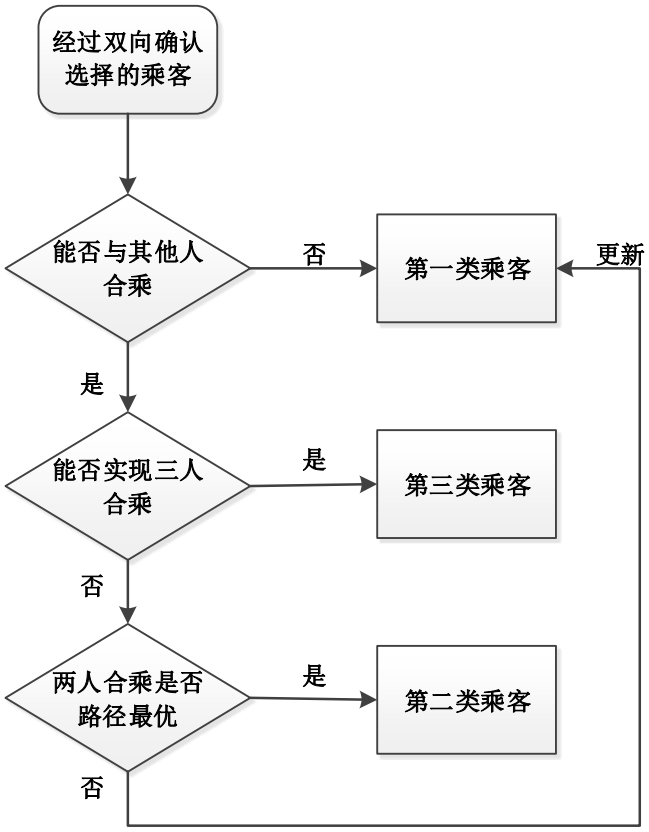

图 3 第一批乘客分类流程图

- **第一步，确定第一类乘客。**因为根据双向确认策略，针对某辆出租车选择的第一批上车乘客总人数远小于乘客总人数，因此遍历这些乘客不会导致过大的计算量。所以我们遍历该车的第一批乘客，找到不能与其他人合乘的乘客即第一类乘客。
- **第二步，确定第三类乘客。**优先考虑第三类乘客，是因为（满足每位乘客时间利益诉求和经济利益诉求的）第三类乘客越多，节省的车辆就越多，因此我们优先找到第三类乘客。与第一次分类相似，采取遍历的方法找到第三类乘客。但是为了减少计算量，我们采用如下方法：假设能够与乘客 i 合乘的所有乘客构成集合 R，能够与乘客 i 达成三人合乘条件的乘客肯定包含在 R

中，即因此只需要在集合 R 中依次枚举两个乘客，根据三人合乘的判定条件，即可找出能够与乘客 i 实现三人合乘的所有第三类乘客。相比于遍历所有人的方法，这种方法大大缩小了搜索范围。

- **第三步，确定第二类乘客。**经过第一次和第二次筛选后，剩余的乘客均为第二类乘客，但可能存在“重复”情况，比如：乘客 i 与乘客 j 可以合乘，乘客 i 与乘客 k 也可以合乘，这就需要判断乘客 i 与谁合乘是组合最优。为此，我们以节省的路程 $Distance_{save}$ 作为判断指标，当乘客 i 与乘客 j 合乘时节约的路程大于乘客 i 与乘客 k 合乘时节约的路程，即

$$Distance_{ij\_save} > Distance_{ik\_save} \tag{11}$$

其中

$$Distance_{ij\_save} = Distance_{i\_alone} + Distance_{j\_alone} - Distance_{ij\_share} \tag{12}$$

$$Distance_{ik\_save} = Distance_{i\_alone} + Distance_{k\_alone} - Distance_{ik\_share} \tag{13}$$

则安排乘客 i 与乘客 j 合乘，否则安排乘客 i 与乘客 k 合乘。这样的分类方法虽然不能保证每个人行驶的路程最短，但是能够保证第二类乘客整体行驶的路程最短。需要注意的是当安排乘客 i 与乘客 j 合乘，乘客 k 可能找不到合乘的对象，这时需要把乘客 k 重新划入到第一类乘客中，即对第一类乘客进行更新。

分类具体步骤如下：

Step 1：根据双向确认规则，得到第一批上车的乘客的集合 $Q\{p_1, p_2, \cdots, p_N\}$。

Step 2：对于乘客 $p_i \in Q, (i = 1, 2, \cdots, N)$，找到集合 Q 中所有能与 p_i 合乘的乘客，并将这些乘客构成集合 Q_i。

Step 3：找出所有等于空集的集合 Q_i，将这些乘客划为第一类乘客，并从集合 Q 中删除这些乘客。

Step 4：对于所有非空且集合中元素个数大于等于 2 的集合 $Q_i (i = 1, 2, \cdots, N)$，从 Q_i 中任选选择 2 个乘客 p_x、p_y，判断这两名乘客能否与乘客 p_i 实现三人合乘，直到遍历所有可能的组合。如果三人能够合乘，将 $\{p_x, p_y, p_i\}$ 作为一个元素存入集合 P 中。

Step 5：对集合 P 中的所有组合按照合乘时节约的路程排序，保留节约路程

最多的组合$\{p_x, p_y, p_i\}$，同时删除P中任何包含了乘客p_x或p_y或p_i的组合。重复 Step 5，直到选出最后一个组合。

Step 6：Step 5 中产生的所有三人组合构成第三类乘客，从Q中删除第三类乘客。对当前Q中的所有乘客按照合乘条件两两组合，并将所有组合存入集合Φ中。

Step 7：对集合Φ中的所有组合按照合乘时节约的路程排序，保留节约路程最多的组合$\{p_x, p_y\}$，并且删除Φ中任何包含了乘客p_x或p_y的乘客，重复 Step 7，直到选出最后一个组合。

Step 8：将 Step 7 中产生的所有二人组合构成第二类乘客，从Q中删除第二类乘客，余下的乘客重新归入到第一类乘客中，整个过程结束。

4.1.4 最短路径搜索问题

路径选择关系到乘客搭乘出租的路程。在设计合乘方案时，判断是否满足合乘条件问题以及对第一批乘客分类时都需要考虑乘客的合乘时路径的距离$Distance_{share}$。本文以出租车能够完成接送乘客任务的最短路径作为乘客的合乘路径L，其距离为$Distance_{share}$。

合乘时出租车可以选择不同的路线将乘客依次送达目的地，当有两个乘客合乘时，存在 2 种可能的路径（如图 4 所示，即先送乘客 1 或先送乘客 2）。当有三个乘客合乘时，假设上车顺序确定，存在 6 种可能的送客路径（如图 5 所示，即先送乘客 1、再送乘客 2、最后送乘客 3，先送乘客 1、再送乘客 3、最后送乘客 2，先送乘客 2、再送乘客 1、最后送乘客 3，先送乘客 2、再送乘客 3、最后送乘客 1，先送乘客 3、再送乘客 1、最后送乘客 2，先送乘客 3、再送乘客 2、最后送乘客 1）。

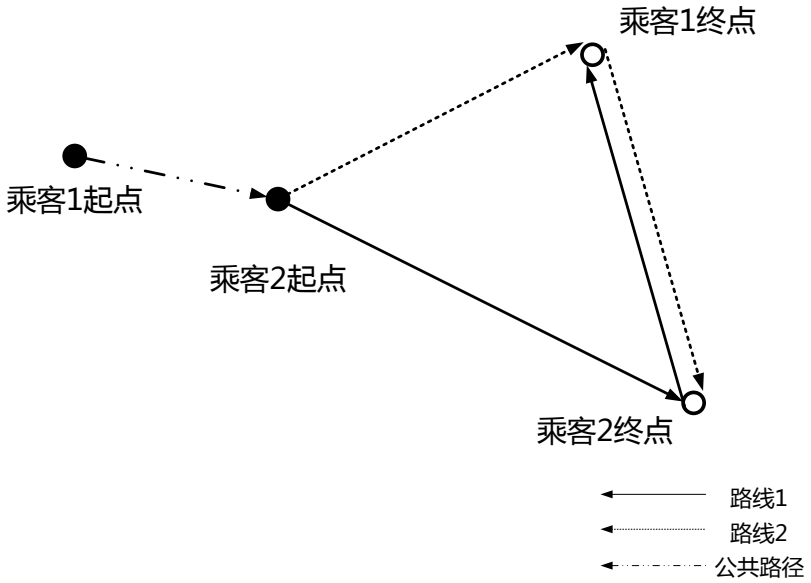

图 4 两人合乘时路线示意图

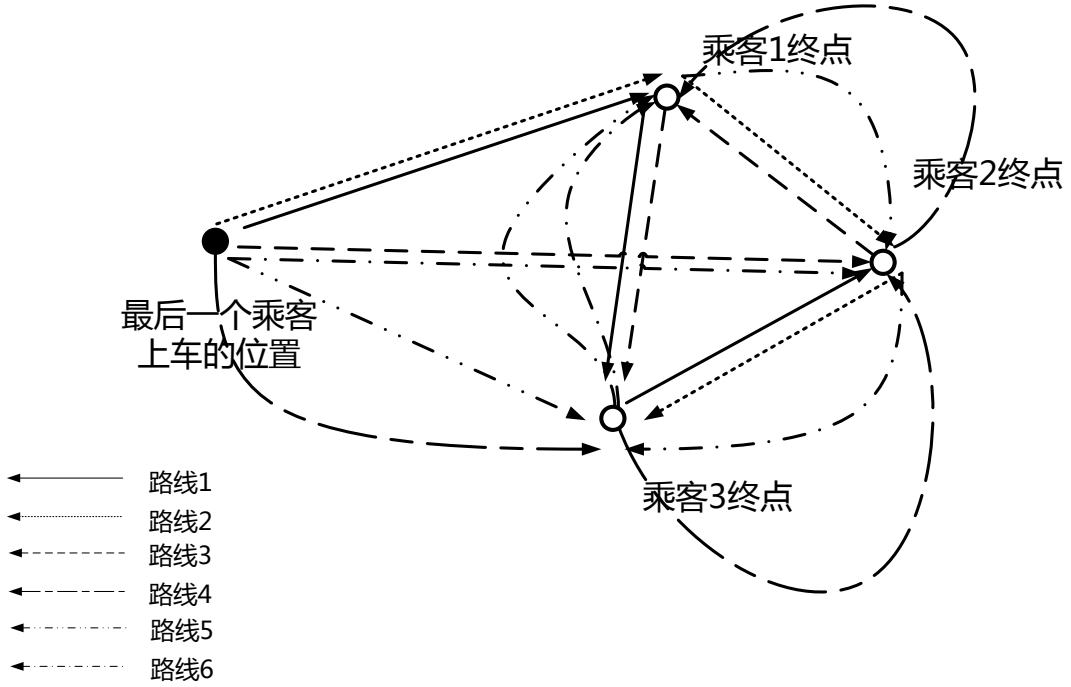

图 5 三人合乘时路线示意图

目前，求解最短路径的经典算法有模拟退火算法、蚁群算法等。这些算法适用于大型的路径规划问题，得到的解也能够逼近最优解。本问题中，由于出租车最多乘坐 3 名乘客，出租车接乘客与送乘客可能的路径有限。考虑到这个特殊性，我们直接采取遍历的方法确定两人合乘时出租车接送乘客的最短路径 L_{ij}（其距离为 $Distance_{ij\_share}$）和三人合乘时出租车接送乘客的最短路径 L_{ijk}（其距离为

$Distance_{ijk\_share}$）。

4.1.4 合乘方案及其算法总结

充分讨论多车“竞争”问题、合乘条件判定问题、第一批乘客分类问题和最短路径搜索问题的解决方案后，现将图 1 所示的合乘总体方案细化，其具体步骤为：

Step 1：假设当前所有等待乘车的乘客集合为$Q_0\{p_1,p_2,\cdots,p_M\}$，当前空载出租车集合为$T_0\{t_1,t_2,\cdots,t_L\}$，当前非满载出租车集合为$TF\{t_1,t_2,\cdots,t_K\}$（$T_0 \subseteq TF$），根据双向确认规则，得到第一批上车的乘客的集合$Q\{p_1,p_2,\cdots,p_N\}(M>N)$。

Step 2：根据第一批乘客分类算法将Q分为三类，即第一类乘客Q_1、第二类乘客Q_2、第三类乘客Q_3。

Step 3：调度平台安排出租车依据最短合乘路径接第三类乘客Q_3上车，并依据最短路径将这些乘客分别送往目的地。同时更新Q_0、T_0、TF。

Step 4：安排TF中出租车依据就近原则接Q_0中乘客上车，如果出租车空载，则乘客直接上车，反之乘客上车前需要根据算法 1 判定是否满足与该车上乘客合乘条件。如果该车坐满或附近没有等待上车的乘客，则依据最短合乘路径将这些乘客分别送往目的地。同时更新Q_0、T_0、TF。

Step 5：重复 Step 4，直至没有出租车可以安排或没有等待上车的乘客。

4.2 车费计算方法

车费计算是合乘系统设计中非常关键的问题，直接影响着乘客与司机的经济利益分配。在车费计算时需要遵循两条基本的原则：

- 不能损害乘客的合乘积极性，即合乘时乘客需要支付的费用不能高于单独乘车时需要支付的费用；
- 不能损害司机的合乘积极性，即合乘时司机的收入不能低于非合乘时的收入。

以两名乘客 i 与 j 合乘为例，假定这两名乘客已经满足了合乘条件。乘客 i 单独乘车时车辆行驶距离为$Distance_{i_alone}$，乘客 j 单独乘车时车辆行驶距离为$Distance_{j_alone}$，将两人合乘时车辆行驶最短距离作为合乘路径$Distance_{ij_share}$。假

设出租车每公里的收费为α（每公里单价不因是否合乘改变）。那么由于合乘而节约的总费用为：

$$Save_{share}=\alpha\left(Distance_{i\_alone}+Distance_{j\_alone}-Distance_{ij\_share}\right) \tag{14}$$

约定司机与乘客之间的利益分配比例因子为$\beta(0<\beta<1)$。司机收获的利益为

$$Income_{Driver}=\beta\bullet Save \tag{15}$$

两名乘客收获的总利益为

$$Income_{Passenger}=(1-\beta)\bullet Save \tag{16}$$

两名乘客各自获益按照他们的路程比例进行分配，即

$$\begin{aligned} Income_{i\_share}&=(1-\beta)\bullet Save\bullet\frac{Distance_{i\_alone}}{Distance_{i\_alone}+Distance_{j\_alone}}\\ Income_{j\_share}&=(1-\beta)\bullet Save\bullet\frac{Distance_{j\_alone}}{Distance_{i\_alone}+Distance_{j\_alone}} \end{aligned} \tag{17}$$

经过上述的利益分配后，乘客 *i* 实际支付的乘车费用为

$$Fee_{i\_share}=\alpha\bullet Distance_{i\_alone}-Income_{i\_share} \tag{18}$$

乘客 *j* 实际支付的乘车费用为：

$$Fee_{j\_share}=\alpha\bullet Distance_{j\_alone}-Income_{j\_share} \tag{19}$$

司机实际的收入为：

$$Income_{Driver\_share}=\alpha\bullet Distance_{ij\_share}+\alpha\bullet\beta\bullet Save \tag{20}$$

在满足合乘条件时，式（7）总是成立的，因此式（14）中$Save>0$。考察式（18）、（19）、（20），可得：

$$\begin{aligned} &Fee_{i\_share}<\alpha\bullet Distance_{i\_alone}=Fee_{i\_alone}\\ &Fee_{j\_share}<\alpha\bullet Distance_{j\_alone}=Fee_{j\_alone}\\ &Income_{Driver\_share}>\alpha\bullet Distance_{ij\_share}=Income_{Driver\_alone} \end{aligned} \tag{21}$$

由式（21）可得，按照式（18）（19）设计的收费方案，合乘时每个乘客的支出均小于单独乘车时的支出，司机的收入大于单独行驶相同距离时的收入。因此这种计费方案能够提高乘客与司机的合乘积极性，是一种合理的车费计算方式。上述计费方式可拓展到 3 人合乘的情况，这里不再赘述。

车费计算的具体步骤如下：

Step 1：根据合乘乘客的起点终点，规划合乘的最短路径 L，并计算得出合乘路径的距离 $Distance_{share}$；

Step 2：根据式（14）计算合乘节省费用 $Save_{share}$；

Step 3：根据式（17）和利益分配因子 β，计算各个乘客的收益 $Income_{share}$；

Step 4：根据式（18）和（19）得到每名乘客实际支付费用 Fee_{share}

4.3 合乘方案求解与数据分析

4.3.1 求解某情境下合乘方案

假设某城市的路网为正方形网格，网格边长 500 米，道路均可双向行驶。据附件 1 打车需求信息和附件 2 空驰出租车位置信息，求解具体的分配方案。首先对乘客需求信息进行预处理，忽略出发地和目的地之间距离小于 20 米的乘车需求信息，共忽略 4 名乘客的请求信息。经计算共得到分配方案 1 个（详细分配方案见附件），该方案用车 541 辆，其中 226 辆出租车搭载 1 人，174 辆出租车搭载 2 人，141 辆出租车搭载 3 人，乘客平均等待时间为 0.26 分钟。乘客等待时间分布如图 6 所示。

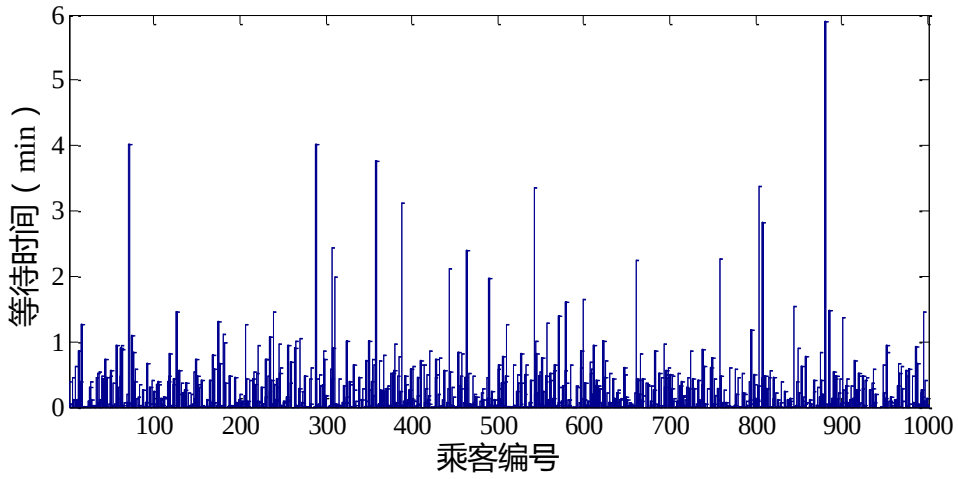

图 6 乘客等待时间分布图

4.3.2 合乘方案分析

在 4.1 和 4.2 节中，我们详细分析和描述了出租车合乘方案及其算法。现在我们对获得的分配方案进行分析。

4.3.2.1 多车“竞争”问题数据分析

假设出租车只根据就近原则搭乘附件 1 中发出请求信息的所有乘客，会出现多车“竞争”问题，竞争情况如图 7 所示。

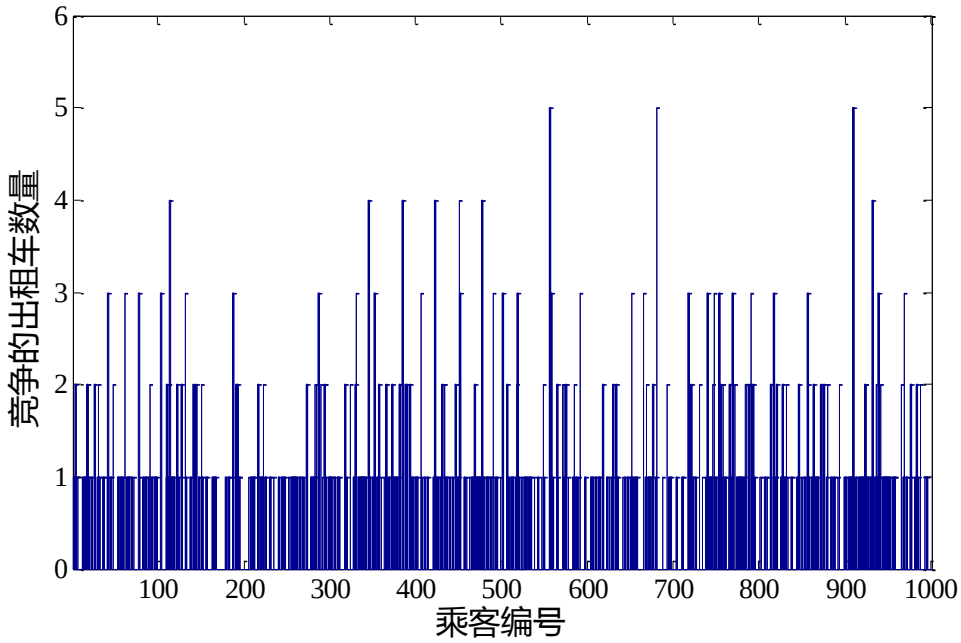

图 7 多车“竞争”数量分布

从图 7 可以看出，相当一部分乘客成为多辆出租车“竞争”的对象，最多时出现了 5 辆车竞争同一个乘客的情况。据统计，全地图中出现“竞争”的出租车达到 303 辆，占到出租车总数的 45%。说明该竞争问题不是个别现象，而是一种常见现象，因此设计系统时必须要考虑多车“竞争”问题。本文利用双向确认策略，有效地解决了该问题，同时尽量保证了乘客以最快时间上车。

4.3.2.1 “第一批乘客”的分类结果及分析

图 8 为利用双向确认策略选择的第一批上车乘客的分布情况，总人数为 457 人。如果不对这些乘客进行分类，则接第一批乘客需要使用出租车 457 辆。

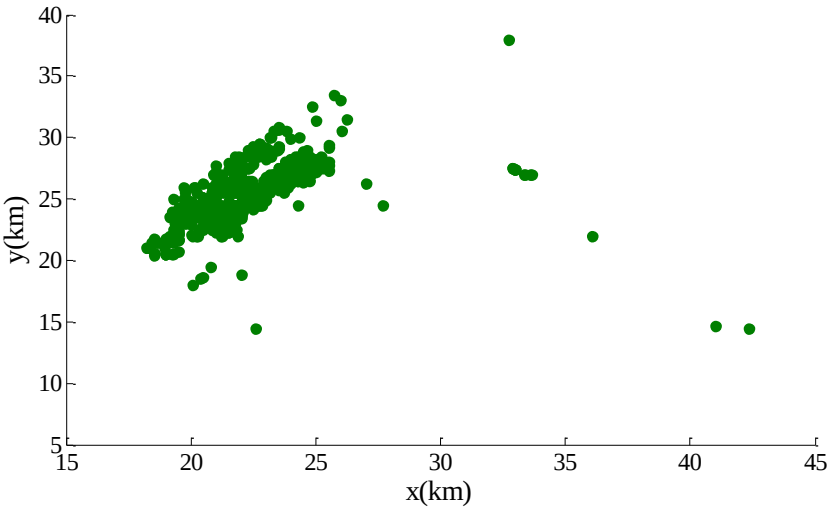

图 8 “第一批乘客”分布情况

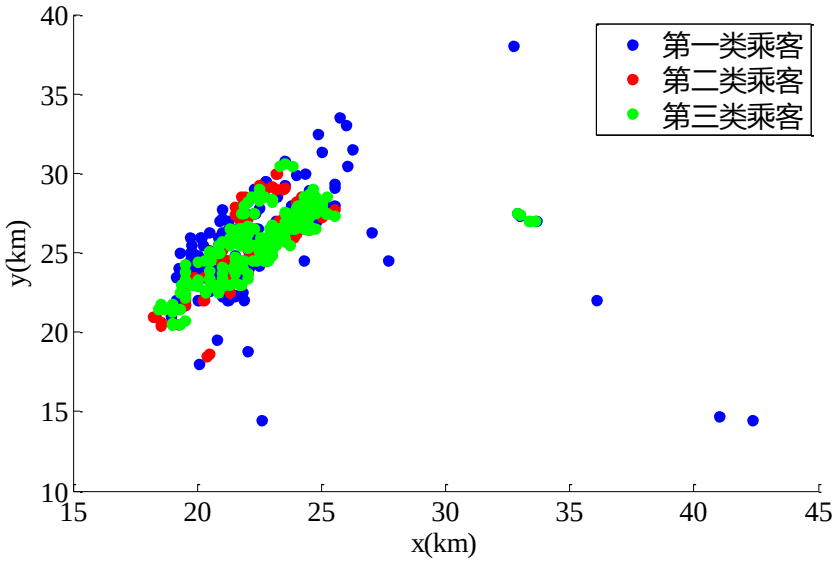

图 9 “第一批乘客”分类结果

利用本文提出的分类方法，对第一批上车的乘客进行分类，其结果如图 9 所示。由图 9 可得第一批乘客中包含大量可以合乘的乘客，其中可以三人合乘的乘客有 228 人，可以两人合乘的乘客有 96 人。经计算，经过分类后，接第一批乘客需要使用的出租车数量为 257 辆，相比于不分类时，节省 200 辆出租车。如果将所有乘客考虑在内，不对第一批乘客进行分类时，将所有的乘客安排完毕需要使用 593 辆车，乘客平均等待时间为 0.19 分钟，分类后只需要使用 541 辆车，乘客平均等待时间为 0.26 分钟。对比结果如表 2、表 3 所示。可以看出对第一类乘客进行分类后，能够在不明显增加平均等待时间的前提下，增加乘客合乘的比例，进而有效节省出租车的使用量。

表 2 第一批乘客分类前后所需出租车数量对比

	安排第一批乘客需要车辆数	安排所有乘客需要车辆数
第一批乘客不分类	457	593
第一批乘客分类	257	541

表 3 第一批乘客分类前后乘客平均等待时间对比

	乘客平均等待时间（分钟）
第一批乘客不分类	0.19
第一批乘客分类	0.26

4.3.3 车费计算结果及分析

根据本文提出的合乘方案以及车费计算方法，假设出租车每公里收费 3 元，合乘方案中利益分配因子 β=0.5。得到合乘乘客乘车费用分布情况如图 10 所示。

图 10 还将单独乘车时的车费列出，为了便于观察，图 11 将图 10 中 600 号至 640 号乘客的支出进行放大显示。

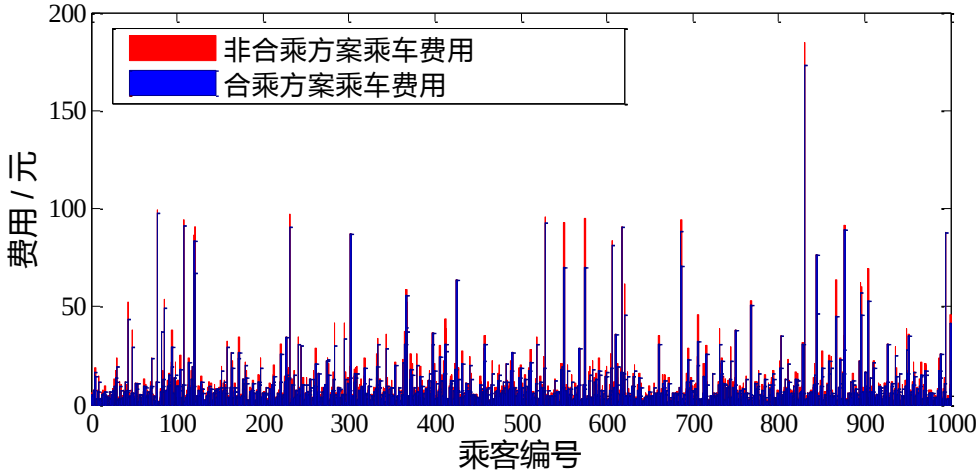

图 10 合乘时与单独乘车时费用支出对比

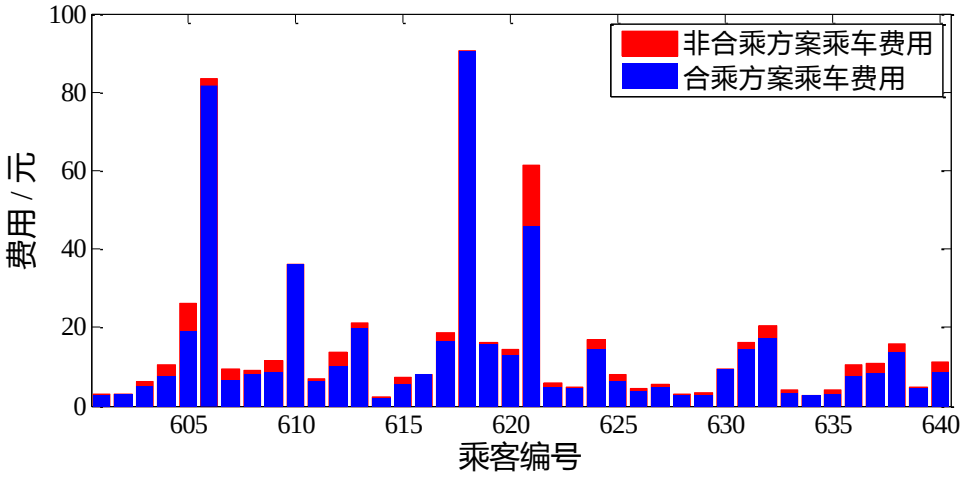

图 11 合乘时与单独乘车时部分乘客费用支出对比

从图 10 和 11 可以看出：对于所有参与合乘的乘客，合乘时支出的费用均小于单独乘车时指出的费用。

图 12 中给出了根据本文提出的车费计算方法，司机在合乘时的收入情况。为了形成对比，图 12 也给出了司机在没有乘客合乘时的收入情况。同时为了便于观察，图 13 将图 12 中 200 号至 260 号司机的收入进行放大显示。

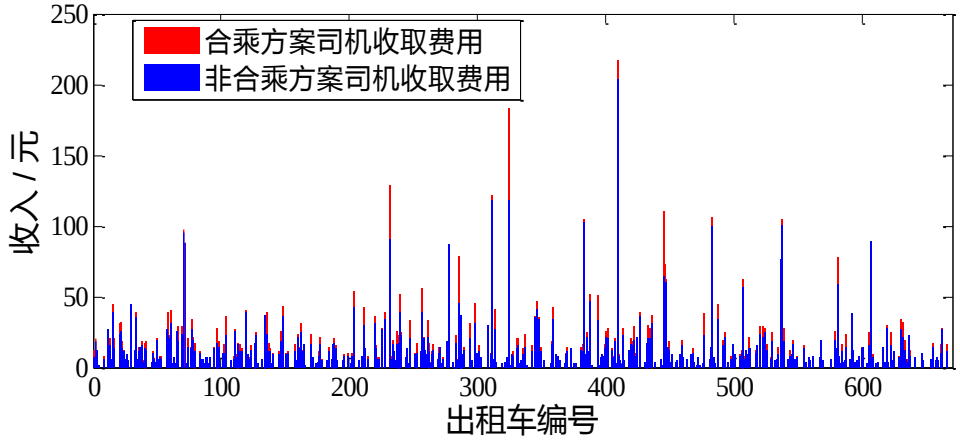

图 12 司机收入情况对比

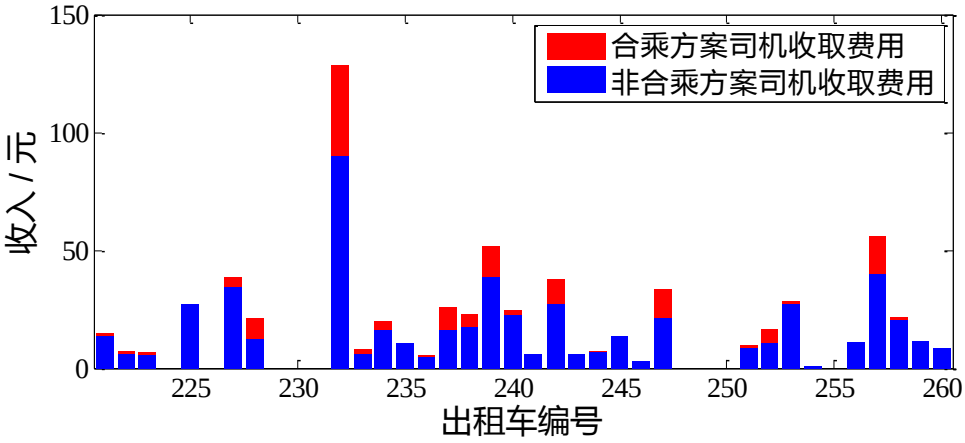

图 13 部分司机收入情况对比

从图 12 和 13 可以看出：在有乘客合乘时，司机的收入总是大于没有乘客合乘时的收入。综合图 10 至 13 可以得出结论：本文提出的方法符合司机与乘客的共同利益，有助于提升司机与乘客的合乘积极性。

5 结论

本文利用贪心算法以及层次聚类的思想设计了一种出租车合乘系统，该系统能够保证在一定的容忍条件下，尽量减少乘客的等车时间，并且充分节省了出租车资源。同时，该出租车合乘系统的车费计算方法能够保证乘客与司机的合乘积极性。利用题目中给出的数据计算表明：本文设计的系统能够保证在运送所有乘客的前提下，节省出租车 116 辆，乘客最大等待时间小于 6 分钟，乘客平均等待时间 0.26 分钟。

6 模型评价

模型优点：

1）在安排出租车接客时，将贪心算法与最短路径规划算法结合使用，最大限度的减少了乘客的等车时间

2）利用层次聚类的思想，最大限度减少了出租车数量的消耗

3）车费计算方式比较合理，保证了乘客与司机的合乘积极性

模型缺点：

1）在进行人车分配时，基于的是静态模型，并没有考虑乘客需求的动态变化以及出租车状态的变化

参考文献

[1] 刘鑫. 城市车辆动态拼车调度机制的研究[D]. 南京大学, 2013.

[2] 郭会, 王丽侠. 基于个性化需求的拼车路径匹配算法研究[J]. 计算机技术与发展, 2017, 27(1):57-60.

[3] 张瑾. 出租车“拼车”问题研究及其服务系统设计实现[D]. 兰州交通大学, 2009.

[4] 吕红瑾, 夏士雄, 杨旭,等. 基于区域划分的出租车统一推荐算法[J]. 计算机应用, 2016, 36(8):2109-2113.

[5] 董映杉, 魏清媛, 任媛媛,等. 基于智能化调度拼车出行方案[J]. 科技与企业, 2016(10):60-61.

[6] 谭家美, 朱丽叶, 南香兰,等. 网络拼车成功的因素分析[J]. 上海海事大学学报, 2013, 34(2):89-94.

[7] 段小艺. 实时出租车拼车关键技术研究[D]. 华东师范大学, 2016.

[8] 唐方慧. 出租车合乘路径选择及费率优化问题研究[D]. 兰州交通大学, 2016.

[9] 王丽珍. 大城市出租车静态和动态合乘模式的探讨[D]. 长沙理工大学, 2012.

[10]于匡员. 基于预约模式的出租车合乘路径优化[D]. 哈尔滨工业大学, 2015.

附录

源程序代码

本论文研究中所使用的全部程序均为自己独立撰写，均为 MATLAB 代码。程序包括主程序及调用函数，具体源代码如下：

1)主函数（main.m）

```
%% 主程序 %%
%侯振宇
%2017 数模省赛
%2017.4.21
%2017.4.24 修改，将由直接第一人上车到接完所有人
%2017.4.24 修改，可以改变阈值，并观察结果变化

%% 初始化
close all;
clear;
clc;
%% 导入数据
path='D:\工作文件\硕士研究生\数学建模\2017 省赛\2017 湖南研究生数学建模竞赛题目\2017 湖南研究生数学建模竞赛题目\A 题';
passenger_name='requests.csv';
taxi_name='taxi.csv';
p=csvread([path,'\',passenger_name],1,1);
t=csvread([path,'\',taxi_name],1,1);

%乘客
start_x=p(:,1);
start_y=p(:,2);
start_position=[start_x,start_y];
end_x=p(:,3);
end_y=p(:,4);
end_position=[end_x,end_y];

%出租车
taxi_x=t(:,1);
```

```
taxi_y=t(:,2);
taxi_position=[taxi_x,taxi_y];
t_position_now=taxi_position;
p_num=length(start_x);
t_num=length(taxi_x);

%出租车行驶速度（km/min）
v=1;

%%%%%%%%%%%%%%%%%% 设置改变相关阈值，观察最终结果的变化%%%%%%%%%%%%%%%%%%
d_start2end_p_num=1;%改变次数
d_farrest_t_num=1;%次数
max_total_time=zeros(d_start2end_p_num,d_farrest_t_num);%从出粗车现在位置到把乘客送完的最长时间
max_total_beside_taxi=zeros(d_start2end_p_num,d_farrest_t_num);%从接了第一个乘客开始，到把乘客送完的最长时间
max_wait_time=zeros(d_start2end_p_num,d_farrest_t_num);%乘客的最长等待时间
max_total_p=zeros(d_start2end_p_num,d_farrest_t_num);%乘客最长上车到下车时间
taxi_used_all=zeros(d_start2end_p_num,d_farrest_t_num);%所用出租车数
int_flag_all=zeros(d_start2end_p_num,d_farrest_t_num);%迭代次数
sum_total_time=zeros(d_start2end_p_num,d_farrest_t_num);%每辆车把所有乘客送完的总用时
sum_total_beside_taxi=zeros(d_start2end_p_num,d_farrest_t_num);%不算接第一个乘客时间，把所有乘客用完总时间
sum_wait_time=zeros(d_start2end_p_num,d_farrest_t_num);%所有乘客等待时间的总和
sum_total_p=zeros(d_start2end_p_num,d_farrest_t_num);%所有乘客上车到下车时间的总和
p_no_taxi=zeros(d_start2end_p_num,d_farrest_t_num);%未上车的人
d_start2end_p_change=linspace(0,0.1,d_start2end_p_num);
d_farrest_t_change=linspace(1,50,d_farrest_t_num);

for d_start2end_p_count=1:d_start2end_p_num
    for d_farrest_t_count=1:d_farrest_t_num
        d_start2end_p=d_start2end_p_change(d_start2end_p_count);%乘客的起点到终点阈值
        d_farrest_t=d_farrest_t_change(d_farrest_t_count);%出租车最远距离阈值
```

```
%% 定义相关参数
flag_p=zeros(p_num,1);%每个乘客对应的车号，0 为未上车，-1 为不派车
flag_t=zeros(t_num,3);%每个车载的乘客，共三个位置，0 为该位置没有人,-1 为不用这辆车
wait_time=99999*ones(p_num,1);%每个乘客的等待时间
total_p=zeros(p_num,1);%每个乘客到达目的地的总用时
d_same_taxi=0.5;%同乘一辆车的聚类条件
d_start2end_p=0.02;%乘客的起点到终点阈值
d_farrest_t=10;%出租车最远距离阈值
%% 初始情况画图
figure;
scatter(start_x,start_y,'r','fill');
hold on;
% scatter(end_x,end_y,'g','fill');
% hold on;
scatter(taxi_x,taxi_y,'b','fill');
legend('start','taxi');

%% 第一步，如果目的地与起点距离小于 500 米，不派车%每个人起点和终点的距离
d_p=abs(start_x-end_x)+abs(start_y-end_y);
flag_p(d_p<d_start2end_p)=-1;
wait_time(d_p<d_start2end_p)=-1;%更新等待时间
d_p2p=d(start_position,start_position);
%% 寻找距离每辆车最近的人与距离每个人最近的车
distance=d(start_position,taxi_position);
[d_nearest_p,nearest_p]=sort(distance,1);%距车最近的人
[d_nearest_t,nearest_t]=sort(distance,2);%距人最近的车

%% 第二步，如果所有人距离某辆车都很远(10km 以上)，则不用这辆车
if ~isempty(find(d_nearest_p(1,:)>d_farrest_t))
    flag_t(find(d_nearest_p(1,:)>d_farrest_t),1)=-1;
    flag_t(find(d_nearest_p(1,:)>d_farrest_t),2)=-1;
    flag_t(find(d_nearest_p(1,:)>d_farrest_t),3)=-1;
```

```
        end
        %% 进行人车匹配，并筛选能拼车的人
        t_remain=find(flag_t(:,1)==0);
        p_remain=find(flag_p==0);
        distance_remain=d(start_position(p_remain,:),taxi_position(t_remain,:));
        [d_nearest_p_remain,nearest_p_remain]=sort(distance_remain,1);%距车最近的人
        [d_nearest_t_remain,nearest_t_remain]=sort(distance_remain,2);%距人最近的车
        if ~isempty(find(d_nearest_p_remain(1,:)>d_farrest_t, 1))
            flag_t(t_remain(find(d_nearest_p_remain(1,:)>d_farrest_t)),:)=[-1,-1,-
1]; %#ok<*FNDSB>
        end
        p_special=[];
        t_special=[];
        for t_count=1:length(t_remain)
            p_temp=nearest_p_remain(1,t_count);%距车最近的人
            if t_count==nearest_t_remain(p_temp,1)&&flag_t(t_remain(t_count),1)==0%距人最
近的车，同时车人距离小于 500 米
                p_special=[p_special,p_remain(p_temp)];
                t_special=[t_special,t_remain(t_count)];
            end
        end
        matrix_special=zeros(length(p_special));%每个人及其可以同乘的人
        matrix_special(:,1)=p_special;
        for special_count=1:length(p_special)
            temp=1;
            for count=1:length(p_special)
                if
d(start_position(p_special(special_count),:),start_position(p_special(count),:))<d_same_taxi&&cou
nt~=special_count

[path,total_wait,total]=istogether(0,p_special(special_count),p_special(count),t,p);
                    if ~isempty(path)
                        temp=temp+1;
                        matrix_special(special_count,temp)=p_special(count);
                    end
```

```
                end
            end
        end

        %% 对能拼车的人判断能否三个人拼车
        plan=[];%方案，共 12 列，出租车，上车顺序，等待时间，下车顺序，总用时，总节省路程
        p_together=p_special(find(matrix_special(:,2)~=0));
        t_together=t_special(find(matrix_special(:,2)~=0));
        p_pairs=zeros(length(p_together),2);
        p_pairs(:,1)=1:length(p_together);
        p_pairs(:,2)=p_together;
        matrix_together=matrix_special(find(matrix_special(:,2)~=0),:);
        for together_count=1:length(p_together)
            first_p=matrix_together(together_count,1);
            second_num=length(find(matrix_together(together_count,:)~=0));
            for second_count=2:second_num
                second_p=matrix_together(together_count,second_count);
                second_vector=matrix_together(p_pairs(p_pairs(:,2)==second_p,1),:);
                third_num=length(find(second_vector~=0));
                for third_count=1:third_num
                    third_p=second_vector(third_count);
                    if third_p~=second_p&&third_p~=first_p
                        taxi=t_together(p_pairs(:,2)==first_p);

[path,total_wait,total]=shorestpathV3(taxi,first_p,second_p,third_p,t,p);
                        if ~isempty(path)&&isempty(find(total_wait>10))
                            st=d(taxi_position(taxi,:),start_position(first_p,:));
                            s1=d(start_position(first_p,:),end_position(first_p,:));
                            s2=d(start_position(second_p,:),end_position(second_p,:));
                            s3=d(start_position(third_p,:),end_position(third_p,:));

plan=[plan;[taxi,first_p,second_p,third_p,total_wait,path,total,s1+s2+s3+st-total(4)]];
                        end
                    end
```

```
                end
            end
        end

        %% 根据拼车能节省的最多的路程来选择哪些乘客能够三人拼车及上车、下车顺序
        %按照节省路程数量进行从大到小排序
        [sorted_data,sort_index]=sort(plan(:,size(plan,2)),'descend');
        plan_temp=plan(sort_index,:);
        %循环，每次选择节省路程最多的组合上车，然后将包含上车人的组合去掉，直到所有组合都排除完
        x=[];y=[];
        while ~isempty(plan_temp)
            %让节省最多路程的人上车
            chosen_group=plan_temp(1,2:4);
            flag_p(chosen_group(1))=plan_temp(1,1);
            wait_time(chosen_group(1))=plan_temp(1,5);%更新等待时间
            flag_p(chosen_group(2))=plan_temp(1,1);
            wait_time(chosen_group(2))=plan_temp(1,6);%更新等待时间
            flag_p(chosen_group(3))=plan_temp(1,1);
            wait_time(chosen_group(3))=plan_temp(1,7);%更新等待时间
            flag_t(plan_temp(1,1),:)=chosen_group;
            t_position_now(plan_temp(1,1),:)=start_position(chosen_group(1),:);
            y=[y;chosen_group];
            x=[x;plan_temp(1,1)];
            %去掉包含上了车的乘客的组合
            index_temp=[];
            for count_temp=2:size(plan_temp,1)
                temp_group=plan_temp(count_temp,2:4);
                if
isempty(find(temp_group==chosen_group(1)))&&isempty(find(temp_group==chosen_group(2)))&
&isempty(find(temp_group==chosen_group(3))) %#ok<*EFIND>
                    index_temp=[index_temp,count_temp]; %#ok<*AGROW>
                end
            end
            plan_temp=plan_temp(index_temp,:);
```

```
        end
        % 画图展示
        figure;
        scatter(taxi_x(flag_t(:,1)==0),taxi_y(flag_t(:,1)==0),'fill');
        hold on;
        scatter(start_x(flag_p==0),start_y(flag_p==0),'r','fill');
        hold on;
        scatter(start_x(flag_p>0),start_y(flag_p>0),'g','fill');
        legend(['未载人的车，共',num2str(length(taxi_x(flag_t(:,1)==0))),'辆'],['未上车的人，共',num2str(length(start_x(flag_p==0))),'人'],['已上车的人，共',num2str(length(start_x(flag_p>0))),'人']);
        title('第一部分三个拼车的人上车后');
        axis([0,45,10,40])

        %% 根据上述结果判断匹配后剩下的人乘车方式
        p_after=p_together(flag_p(p_together)==0);
        t_after=t_together(flag_p(p_together)==0);

        matrix_after=zeros(length(p_after));% 每个人及其可以同乘的人
matrix_after(:,1)=p_after;
        matrix_after(:,1)=p_after;
        for after_count=1:length(p_after)
            temp=1;
            for count=1:length(p_after)
                if
d(start_position(p_after(after_count),:),start_position(p_after(count),:))<d_same_taxi&&count~=after_count
                    [path,total_wait,total]=istogether(0,p_after(after_count),p_after(count),t,p);
                    if ~isempty(path)
                        temp=temp+1;
                        matrix_after(after_count,temp)=p_after(count);
                    end
                end
            end
        end
```

```
        %确定能够两拼的
        p_together_after=p_after(matrix_after(:,2)~=0);

        if ~isempty(p_together_after)

            t_together_after=t_after(matrix_after(:,2)~=0);
            matrix_together_after=matrix_after(matrix_after(:,2)~=0,:);

            p_pairs_after=zeros(length(p_together_after),2);
            p_pairs_after(:,1)=1:length(p_together_after);
            p_pairs_after(:,2)=p_together_after;

            plan=[];
            for together_count=1:length(p_together_after)
                first_p=matrix_together_after(together_count,1);
                second_num=length(find(matrix_together_after(together_count,:)~=0));
                for second_count=2:second_num
                    second_p=matrix_together_after(together_count,second_count);
                    taxi=t_together_after(p_pairs_after(:,2)==first_p);
                    [path,total_wait,total]=istogether(taxi,first_p,second_p,t,p);
                    if ~isempty(path)&&isempty(find(total_wait>10))
                        st=d(taxi_position(taxi,:),start_position(first_p,:));
                        s1=d(start_position(first_p,:),end_position(first_p,:));
                        s2=d(start_position(second_p,:),end_position(second_p,:));
                        plan=[plan;[taxi,first_p,second_p,total_wait,path,total,s1+s2+st-
total(3)]];
                    end
                end
            end

            [sorted_data,sort_index]=sort(plan(:,size(plan,2)),'descend');
            plan_temp=plan(sort_index,:);
            %循环，每次选择节省路程最多的组合上车，然后将包含上车人的组合去掉，直
到所有组合都排除完
```

```
            while ~isempty(plan_temp)
                %让节省最多路程的人上车
                chosen_group=plan_temp(1,2:3);
                flag_p(chosen_group(1))=plan_temp(1,1);
                wait_time(chosen_group(1))=plan_temp(1,4);%更新等待时间
                flag_p(chosen_group(2))=plan_temp(1,1);
                wait_time(chosen_group(2))=plan_temp(1,5);%更新等待时间
                flag_t(plan_temp(1,1),1:2)=chosen_group;
                t_position_now(plan_temp(1,1),:)=start_position(chosen_group(1),:);
                %去掉包含上了车的乘客的组合
                index_temp=[];
                for count_temp=2:size(plan_temp,1)
                    temp_group=plan_temp(count_temp,2:3);
                    if
isempty(find(temp_group==chosen_group(1)))&&isempty(find(temp_group==chosen_group(2)))
                        index_temp=[index_temp,count_temp];
                    end
                end
                plan_temp=plan_temp(index_temp,:);
            end
            %画图展示
            figure;
            scatter(taxi_x(flag_t(:,1)==0),taxi_y(flag_t(:,1)==0),'fill');
            hold on;
            scatter(start_x(flag_p==0),start_y(flag_p==0),'r','fill');
            hold on;
            scatter(start_x(flag_p>0),start_y(flag_p>0),'g','fill');
            legend(['未载人的车，共',num2str(length(taxi_x(flag_t(:,1)==0))),'辆'],['未上车的
人，共',num2str(length(start_x(flag_p==0))),'人'],['已上车的人，共
',num2str(length(start_x(flag_p>0))),'人']);
            title('两拼的人上车后');
            axis([0,45,10,40])

            %不能拼车的情况
            if ~isempty(find(flag_p(p_special)==0));
```

```
                    p_still_alone=p_special(find(flag_p(p_special)==0));
                    t_still_alone=t_special(find(flag_p(p_special)==0));
                    flag_p(p_still_alone)=t_still_alone;

wait_time(p_still_alone)=diag(d(start_position(p_still_alone,:),taxi_position(t_still_alone,:)));%  更新等待时间
                    flag_t(t_still_alone,1)=p_still_alone;
                    t_position_now(t_still_alone,:)=start_position(p_still_alone,:);
                end

            end
            %画图展示
            figure;
            scatter(taxi_x(flag_t(:,1)==0),taxi_y(flag_t(:,1)==0),'fill');
            hold on;
            scatter(start_x(flag_p==0),start_y(flag_p==0),'r','fill');
            hold on;
            scatter(start_x(flag_p>0),start_y(flag_p>0),'g','fill');
            legend(['未载人的车，共',num2str(length(taxi_x(flag_t(:,1)==0))),'辆'],['未上车的人，共',num2str(length(start_x(flag_p==0))),'人'],['已上车的人，共',num2str(length(start_x(flag_p>0))),'人']);
            title('第一次匹配的所有乘客都上车后');
            axis([0,45,10,40])

            %% 对剩下的所有的人和车循环匹配，直到没有能够匹配的为止
            whether_flag=ones(t_num,p_num);
            int_flag=0;
            t_remain=find(flag_t(:,3)==0);
            p_remain=find(flag_p==0);
            if ~isempty(t_remain)&&~isempty(p_remain)

                distance_remain=d(start_position(p_remain,:),taxi_position(t_remain,:));
                [d_nearest_p_remain,nearest_p_remain]=sort(distance_remain,1);%距车最近的人
                [d_nearest_t_remain,nearest_t_remain]=sort(distance_remain,2);%距人最近的车
                %进行人车匹配
```

```
                p_special=[];
                t_special=[];
                for t_count=1:length(t_remain)
                    p_temp=nearest_p_remain(1,t_count);%距车最近的人
                    if t_count==nearest_t_remain(p_temp,1)&&flag_t(t_remain(t_count),3)==0%距
人最近的车，同时车人距离小于 500 米
                        p_special=[p_special,p_remain(p_temp)];
                        t_special=[t_special,t_remain(t_count)];
                    end
                end
                while ~isempty(p_special)

                    num=length(p_special);
                    for count=1:num
                        if
d(t_position_now(t_special(count),:),start_position(p_special(count),:))<d_farrest_t
                            switch length(find(flag_t(t_special(count),:)==0))
                                case 1%车上已经有两个人了
                                    taxi_now=t_special(count);
                                    first_p=flag_t(taxi_now,1);
                                    second_p=flag_t(taxi_now,2);
                                    third_p=p_special(count);

[path,total_wait,total]=shorestpathV3(taxi_now,first_p,second_p,third_p,t,p);
                                    if ~isempty(path)&&isempty(find(total_wait>d_farrest_t))%
能上车的情况
                                        flag_p(third_p)=taxi_now;
                                        wait_time(third_p)=total_wait(3);
                                        flag_t(taxi_now,3)=third_p;
                                    else
                                        %如果不能上车，则标记这辆车和这个人不能拼车
                                        whether_flag(taxi_now,third_p)=-1;
                                    end
                                case 2%车上已经有一个人了
                                    taxi_now=t_special(count);
```

```
                                    first_p=flag_t(taxi_now,1);
                                    second_p=p_special(count);

[path,total_wait,total]=istogether(taxi_now,first_p,second_p,t,p);
                                    if ~isempty(path)&&isempty(find(total_wait>d_farrest_t))%
能上车的情况
                                        flag_p(second_p)=taxi_now;
                                        wait_time(second_p)=total_wait(2);
                                        flag_t(taxi_now,2)=second_p;
                                    else
                                        %如果不能上车，则标记这辆车和这个人不能拼车
                                        whether_flag(taxi_now,second_p)=-1;
                                    end
                                case 3%车上没有人
                                    %直接上车
                                    taxi_now=t_special(count);
                                    p_now=p_special(count);
                                    flag_p(p_now)=taxi_now;

wait_time(p_now)=d(taxi_position(taxi_now,:),start_position(p_now,:));
                                    flag_t(taxi_now,1)=p_now;
                                    t_position_now(taxi_now,:)=start_position(p_now,:);
                            end
                        else
                            whether_flag(t_special(count),p_special(count))=-1;
                        end
                    end
                    int_flag=int_flag+1;

                    %画图展示
                    figure;
                    scatter(taxi_x(flag_t(:,1)==0),taxi_y(flag_t(:,1)==0),'fill');
                    hold on;
                    scatter(start_x(flag_p==0),start_y(flag_p==0),'r','fill');
                    hold on;
```

```
                scatter(start_x(flag_p>0),start_y(flag_p>0),'g','fill');
                legend(['未载人的车，共',num2str(length(taxi_x(flag_t(:,1)==0))),'辆'],['未上车的人，共',num2str(length(start_x(flag_p==0))),'人'],['已上车的人，共',num2str(length(start_x(flag_p>0))),'人']);
                title(['第',num2str(int_flag),'次循环后']);
                axis([0,45,10,40])

                %更新 p_special
                t_remain=find(flag_t(:,3)==0);
                p_remain=find(flag_p==0);
                if isempty(t_remain)||isempty(p_remain)
                    break;
                end
                distance_remain=d(start_position(p_remain,:),taxi_position(t_remain,:));
                [d_nearest_p_remain,nearest_p_remain]=sort(distance_remain,1);%距车最近的人
                [d_nearest_t_remain,nearest_t_remain]=sort(distance_remain,2);%距人最近的车
                %进行人车匹配
                p_special=[];
                t_special=[];
                p_special_index=[];
                for t_count=1:length(t_remain)
                    p_temp=nearest_p_remain(1,t_count);%距车最近的人
                    if t_count==nearest_t_remain(p_temp,1)&&flag_t(t_remain(t_count),3)==0%距人最近的车
                        p_special=[p_special,p_remain(p_temp)];
                        t_special=[t_special,t_remain(t_count)];
                        p_special_index=[p_special_index,p_temp];
                    end
                end

                %==================判断匹配上的人能否上车（根据判断过的上车与否来决定）====================%
                for judge_count=1:length(p_special)
```

```
                    p_wait=p_special(judge_count);
                    t_wait=t_special(judge_count);
                    p_wait_index=p_special_index(judge_count);
                    if whether_flag(t_wait,p_wait)==-1
                        t_vector=nearest_t_remain(p_wait_index,:);
                        for t_wait_index=2:length(t_vector)
                            t_next=t_remain(t_vector(t_wait_index));
                            if whether_flag(t_next,p_wait)~=-1&&isempty(find(t_special==t_next))
                                t_special(judge_count)=t_next;
                                break;
                            end
                        end
                    end
                end
                %=========================================================================================%
            end
        end

        %画图展示
        figure;
        scatter(taxi_x(flag_t(:,1)==0),taxi_y(flag_t(:,1)==0),'fill');
        hold on;
        scatter(start_x(flag_p==0),start_y(flag_p==0),'r','fill');
        hold on;
        scatter(start_x(flag_p>0),start_y(flag_p>0),'g','fill');
        legend(['未载人的车，共',num2str(length(taxi_x(flag_t(:,1)==0))),'辆'],['未上车的人，共',num2str(length(start_x(flag_p==0))),'人'],['已上车的人，共',num2str(length(start_x(flag_p>0))),'人']);
        title(['最终结果']);
        axis([0,45,10,40])

        %% 计算总时间
        filename='plan.txt';
```

```
[total_time,total_beside_taxi,taxi_used,total_p]=result(filename,start_position,end_position,taxi_position,flag_t,t,p);

        %% 进行对比，衡量优劣
        p_no_taxi(d_start2end_p_count,d_farrest_t_count)=length(start_x(flag_p<=0));% 未上车的人
        max_total_time(d_start2end_p_count,d_farrest_t_count)=max(total_time);% 从出粗车现在位置到把乘客送完的最长时间
        max_total_beside_taxi(d_start2end_p_count,d_farrest_t_count)=max(total_beside_taxi);%从接了第一个乘客开始，到把乘客送完的最长时间
        max_wait_time(d_start2end_p_count,d_farrest_t_count)=max(wait_time);% 乘客的最长等待时间
        max_total_p(d_start2end_p_count,d_farrest_t_count)=max(total_p);%乘客最长上车到下车时间
        taxi_used_all(d_start2end_p_count,d_farrest_t_count)=taxi_used;%所用出租车数
        int_flag_all(d_start2end_p_count,d_farrest_t_count)=int_flag;%迭代次数

sum_total_time(d_start2end_p_count,d_farrest_t_count)=sum(total_time)+sum(total_p(find(flag_p<=0)));%每辆车把所有乘客送完的总用时

sum_total_beside_taxi(d_start2end_p_count,d_farrest_t_count)=sum(total_beside_taxi)+sum(total_p(find(flag_p<=0)));%不算接第一个乘客时间，把所有乘客用完总时间
        sum_wait_time(d_start2end_p_count,d_farrest_t_count)=sum(wait_time);% 所有乘客等待时间的总和
        sum_total_p(d_start2end_p_count,d_farrest_t_count)=sum(total_p);%所有乘客上车到下车时间的总和

    end
end



wait_time_new=wait_time;
wait_time_new(find(wait_time_new<0))=0;
```

```
figure;
bar(wait_time_new);
axis([1,1001,0,6]);
xlabel('等待时间（min）');
ylabel('乘客编号');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2)计算距离函数（d.m）

```
%% 计算出租车和乘客之间的街区距离 %%
%侯振宇
%2017 数模省赛
%2017.4.21

function distance=d(p_position,t_position)
%输入
%p_position    乘客位置坐标，p_num*2
%t_position    出租车位置坐标，t_num*2

%输出
%distance                   每个乘客和每辆车之间的距离，p_num*t_num

p_num=size(p_position,1);
t_num=size(t_position,1);

taxi_x=t_position(:,1);
taxi_y=t_position(:,2);
start_x=p_position(:,1);
start_y=p_position(:,2);
if p_num>1||t_num>1
    distance=abs(ones(p_num,1)*taxi_x'-start_x*ones(1,t_num))+abs(ones(p_num,1)*taxi_y'-start_y*ones(1,t_num));
else
    distance=abs(taxi_x-start_x)+abs(taxi_y-start_y);
end
```

3)判断两个乘客能否共乘函数（istogether.m）

```
%% 判断两个人能否上同一辆车 %%
function [path,total_wait,total]=istogether(taxi,first,second,t,p);
%输入
%s1                第一个乘客的出发点
%e1                第一个乘客的终点
%s2                第二个乘客的出发点
%e2                第二个乘客的终点

%输出
%path              同乘方案，先送谁再送谁
%total_wait     两个人的等待时间
%total             总用时,第一个、第二个乘客上车到下车的用时、第一个乘客上车到所有乘客下车的总时间

%% 初始化
path=[];total_wait=Inf;total=Inf;

%% 参数计算
s1=p(first,1:2);
e1=p(first,3:4);
s2=p(second,1:2);
e2=p(second,3:4);
if taxi==0;
    taxi_first=0;
else
    taxi_now=t(taxi,:);
    taxi_first=d(taxi_now,s1);
end
%% 判断
beta=1.3;%可以容忍的绕路比例
% beta=Inf;%可以容忍的绕路比例

if d(s2,e1)>d(s2,e2)%先送乘客 2
    if d(s1,s2)+d(s2,e2)+d(e2,e1)<d(s1,e1)+d(s2,e2)
        if d(s2,e2)/d(s2,e2)<=beta&&(d(s1,s2)+d(s2,e2)+d(e2,e1))/d(s1,e1)<=beta
```

```
                path=[second,first];
                total_wait=[taxi_first,taxi_first+d(s1,s2)];
                total=[d(s1,s2)+d(s2,e2)+d(e2,e1),d(s2,e2),d(s1,s2)+d(s2,e2)+d(e2,e1)];
            end
        end
else%先送乘客 1
        if d(s1,s2)+d(s2,e1)+d(e1,e2)<d(s1,e1)+d(s2,e2)
            if (d(s1,s2)+d(s2,e1))/d(s1,e1)<=beta&&(d(s2,e1)+d(e1,e2))/d(s2,e2)<=beta
                path=[first,second];
                total_wait=[taxi_first,taxi_first+d(s1,s2)];
                total=[d(s1,s2)+d(s2,e1),d(s2,e1)+d(e1,e2),d(s1,s2)+d(s2,e1)+d(e1,e2)];
            end
        end
end
end
```

4)判断三个乘客能否共乘函数（whether3IsOK.m）

```
function [index,distance,path]=whether3IsOk(first,second,third,p,distancebefore)
beta=1.3;%可以容忍的绕路比例

startingPoint1=[p(first,1),p(first,2)];%location where the 1st passenger take the taxi
startingPoint2=[p(second,1),p(second,2)];%location where the 2nd passenger take the taxi
startingPoint3=[p(third,1),p(third,2)];%location where the 3rd passenger take the taxi
position1=[p(first,3),p(first,4)]; %1st passenger's destination
position2=[p(second,3),p(second,4)]; %2nd passenger's destination
position3=[p(third,3),p(third,4)]; %3rd passenger's destination

originalDistance1=d(startingPoint1,position1);
originalDistance2=d(startingPoint2,position2);
originalDistance3=d(startingPoint3,position3);
positionOf4=[startingPoint3;position1;position2;position3];

distanceOf4=zeros(4,4);
for i=1:4
    for j=1:4
        tmp1=positionOf4(i,:);
```

```
            tmp2=positionOf4(j,:);
            distanceOf4(i,j)=abs(tmp1(1)-tmp2(1))+abs(tmp1(2)-tmp2(2));
        end
end
possibilityPath=[first,second,third;    %all possible path
        first,third,second
        second,first,third;
        second,third,first;
        third,first,second;
        third,second,first];

overpath=[(d(startingPoint1,startingPoint2)+d(startingPoint2,startingPoint3)+d(startingPoint3,positi
on1))/originalDistance1,...%绕路比例

(d(startingPoint2,startingPoint3)+d(startingPoint3,position1)+d(position1,position2))/originalDistan
ce2,...
        (d(startingPoint3,position1)+d(position1,position2)+d(position2,position3))/originalDistance3;

(d(startingPoint1,startingPoint2)+d(startingPoint2,startingPoint3)+d(startingPoint3,position1))/origi
nalDistance1,...

(d(startingPoint2,startingPoint3)+d(startingPoint3,position1)+d(position1,position3)+d(position3,po
sition2))/originalDistance2,...
        (d(startingPoint3,position1)+d(position1,position3))/originalDistance3;

(d(startingPoint1,startingPoint2)+d(startingPoint2,startingPoint3)+d(startingPoint3,position2)+d(pos
ition2,position1))/originalDistance1,...
        (d(startingPoint2,startingPoint3)+d(startingPoint3,position2))/originalDistance2,...
        (d(startingPoint3,position2)+d(position2,position2)+d(position1,position3))/originalDistance3;

(d(startingPoint1,startingPoint2)+d(startingPoint2,startingPoint3)+d(startingPoint3,position2)+d(pos
ition2,position3)+d(position3,position1))/originalDistance1,...
        (d(startingPoint2,startingPoint3)+d(startingPoint3,position2))/originalDistance2,...
        (d(startingPoint3,position2)+d(position2,position3))/originalDistance3;
```

```
(d(startingPoint1,startingPoint2)+d(startingPoint2,startingPoint3)+d(startingPoint3,position3)+d(position3,position1))/originalDistance1,...

(d(startingPoint2,startingPoint3)+d(startingPoint3,position3)+d(position3,position1)+d(position1,position2))/originalDistance2,...
    (d(startingPoint3,position3))/originalDistance3;

(d(startingPoint1,startingPoint2)+d(startingPoint2,startingPoint3)+d(startingPoint3,position3)+d(position3,position2)+d(position2,position1))/originalDistance1,...

(d(startingPoint2,startingPoint3)+d(startingPoint3,position3)+d(position3,position2))/originalDistance2,...
    (d(startingPoint3,position3))/originalDistance3];

total_p=zeros(size(overpath));%每个乘客自己的上车到下车时间
for count=1:6
    total_p(count,:)=overpath(count,:).*[originalDistance1,originalDistance2,originalDistance3];
end
possibilityDistance=[distanceOf4(1,2)+distanceOf4(2,3)+distanceOf4(3,4);
    distanceOf4(1,2)+distanceOf4(2,4)+distanceOf4(4,3);
    distanceOf4(1,3)+distanceOf4(3,2)+distanceOf4(2,4);
    distanceOf4(1,3)+distanceOf4(3,4)+distanceOf4(4,2);
    distanceOf4(1,4)+distanceOf4(4,2)+distanceOf4(2,3);
    distanceOf4(1,4)+distanceOf4(4,3)+distanceOf4(3,2)];
possibilityDistance_new=[];
possibilityPath_new=[];
total_p_new=[];
for count=1:6
  if  isempty(find(overpath(count,:)>=beta))
      possibilityDistance_new=[possibilityDistance_new;possibilityDistance(count,:)];
      possibilityPath_new=[possibilityPath_new;possibilityPath(count,:)];
      total_p_new=[total_p_new;total_p(count,:)];
  end
end
%shorst=find(possibilityDistance==min(possibilityDistance));
```

```
[value,shorst]=min(possibilityDistance_new);
if (value+distancebefore) < (originalDistance1+originalDistance2+originalDistance3)
        index=1;
        path=possibilityPath_new(shorst,:);
        distance=[total_p_new(shorst,:),value];
else
        index=0;
        path=[];
        distance=inf;
end
```

5)输出三个乘客共乘最短路径函数（shorestpathV3.m）

```
%%%%%%%%%%%%%%%%%%%shorestpathV2                                                        release
note%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function shorestpathV2 inherits main part from function shorestpath.
%compared with previous version(shorestpath),shorestpathV2 can also
%(1)show the entire path from the 1st, 2nd and 3rd passenger take the taxi
%one by one to drive them to their destinations via shorestpath.
%(2)calculate the total distance of the entire path(in km)
%(3)calculate the total waiting time of the 3 passengers(but in km,if you want
%to know the accurate time(in sec or hour) ,you can get it by a certain taxi
%speed).
%23rd April,2017
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%input the taxi number and a sequence contains 3 passengers who take the same taxi in order
%output a sequence indicate the shortest path via their destinations
function [path,totalWating,entirePathDistance]=shorestpathV3(taxiNum,first,second,third,t,p)

%standardEntirePath=[strcat('taxi_',int2str(taxiNum)),strcat('p',int2str(first),'u'),strcat('p',int2str(seco
nd),'u'),strcat('p',int2str(third),'u')]
%standard format
entirePath=[taxiNum,first,second,third];%initiate entirePath by a sequence how to take the taxi

taxiStartingPoint=[t(taxiNum,1),t(taxiNum,2)];%location where the No.taxiNum taxi is
startingPoint1=[p(first,1),p(first,2)];%location where the 1st passenger take the taxi
```

```
startingPoint2=[p(second,1),p(second,2)];%location where the 2nd passenger take the taxi
startingPoint3=[p(third,1),p(third,2)];%location where the 3rd passenger take the taxi

waitingDistance1=d(taxiStartingPoint,startingPoint1);
waitingDistance2=d(taxiStartingPoint,startingPoint1)+d(startingPoint1,startingPoint2);
waitingDistance3=d(taxiStartingPoint,startingPoint1)+d(startingPoint1,startingPoint2)+d(startingPoint2,startingPoint3);
% totalWating=waitingDistance1+waitingDistance2+waitingDistance3 ;%the total waiting time of the 3 passengers(in km)
totalWating=[waitingDistance1,waitingDistance2,waitingDistance3];
entirePathDistance=waitingDistance3;%initiate entirePathDistance

position1=[p(first,3),p(first,4)]; %1st passenger's destination
position2=[p(second,3),p(second,4)]; %2nd passenger's destination
position3=[p(third,3),p(third,4)]; %3rd passenger's destination
distancebefore=waitingDistance3-waitingDistance1;
[index,distance,path]=whether3IsOk(first,second,third,p,distancebefore);
% index=1
if index==1
    %disp('Great! These 3 guys are willing to share a taxi. ')
    %possibilityDistance(shorst,:)%to test the codes
    entirePath=[entirePath,path];%the entire path from the 1st, 2nd and 3rd passenger take the taxi
    %one by one to drive them to their destinations via shorestpath
    total_all=distancebefore+distance(4);
    entirePathDistance=[distance(1:3),total_all];
    %distanceOf4 %to test the codes
else
    path=[];
    entirePathDistance=0;
    entirePathDistance=0;
    %disp('Sorry...These 3 guys can not share a taxi.')
end
```

6)计算费用函数（calculate_fee.m）

```
%% 计算费用 %%
```

```
function [fee_oral,fee_over]=calculate_fee(start_position,end_position,taxi_position,flag_t,t,p);

%% 参数设定
t_num=size(taxi_position,1);
p_num=size(start_position,1);
alpha=3;%每公里单价
beta=0.5;%司机所占利益比
cost_taxi_oral=zeros(t_num,1);%之前每辆车的花费
cost_taxi_now=zeros(t_num,1);%之后每辆车的花费

fee_oral=zeros(p_num,1);%乘客独自乘坐应付的钱数
fee_now=zeros(p_num,1);%乘客实际应付的钱数
fee_over=zeros(p_num,1)%乘客剩下的钱数
d_p=abs(start_position(:,1)-end_position(:,1))+abs(start_position(:,2)-end_position(:,2));%每个乘客的原始距离

[total_time,total_beside_taxi,taxi_used,total_p]=result(0,start_position,end_position,taxi_position,flag_t,t,p);
%% 车费计算
for t_count=1:t_num
    switch length(find(flag_t(t_count,:)>0))%判断坐了几个人
        case 1 %独自乘坐
            first=flag_t(t_count,1);
            fee_oral(first)=alpha*d_p(first);
            fee_over(first)=0;
            cost_taxi_oral(t_count)=fee_oral(first);
            cost_taxi_now(t_count)=cost_taxi_oral(t_count);
        case 2 %两人拼车
            first=flag_t(t_count,1);
            second=flag_t(t_count,2);
            cost_taxi_oral(t_count)=alpha*total_beside_taxi(t_count);
            fee_oral(first)=alpha*d_p(first);
            fee_oral(second)=alpha*d_p(second);
            fee_over(first)=(1-beta)*(fee_oral(first)+fee_oral(second)-
cost_taxi_oral(t_count))*d_p(first)/(d_p(first)+d_p(second));
```

```
                fee_over(second)=(1-beta)*(fee_oral(first)+fee_oral(second)-
cost_taxi_oral(t_count))*d_p(second)/(d_p(first)+d_p(second));
                cost_taxi_now(t_count)=beta*(fee_oral(first)+fee_oral(second)-
cost_taxi_oral(t_count))+cost_taxi_oral(t_count);
            case 3 %三人拼车
                first=flag_t(t_count,1);
                second=flag_t(t_count,2);
                third=flag_t(t_count,3);
                cost_taxi_oral(t_count)=alpha*total_beside_taxi(t_count);
                fee_oral(first)=alpha*d_p(first);
                fee_oral(second)=alpha*d_p(second);
                fee_oral(third)=alpha*d_p(third);
                fee_over(first)=(1-beta)*(fee_oral(first)+fee_oral(second)+fee_oral(third)-
cost_taxi_oral(t_count))*d_p(first)/(d_p(first)+d_p(second)+d_p(third));
                fee_over(second)=(1-beta)*(fee_oral(first)+fee_oral(second)+fee_oral(third)-
cost_taxi_oral(t_count))*d_p(second)/(d_p(first)+d_p(second)+d_p(third));
                fee_over(third)=(1-beta)*(fee_oral(first)+fee_oral(second)+fee_oral(third)-
cost_taxi_oral(t_count))*d_p(third)/(d_p(first)+d_p(second)+d_p(third));
                cost_taxi_now(t_count)=beta*(fee_oral(first)+fee_oral(second)+fee_oral(third)-
cost_taxi_oral(t_count))+cost_taxi_oral(t_count);
        end
end

fee_now=fee_oral-fee_over;



fee_over_together=fee_over(fee_over>0);
fee_oral_together=fee_oral(fee_over>0);
fee_now_together=fee_now(fee_over>0);

figure;
bar(fee_oral,'r');
hold on;
bar(fee_now,'b');
xlabel('乘客编号');
```

```
ylabel('金额');
legend('非合乘方案乘车费用','合乘方案乘车费用');
figure;
bar(fee_oral,'r');
hold on;
bar(fee_now,'b');
xlabel('乘客编号');
ylabel('金额');
axis([600.5,640.5,0,100]);
legend('非合乘方案乘车费用','合乘方案乘车费用');

figure;
bar(cost_taxi_now,'r');
hold on;
bar(cost_taxi_oral,'b');
xlabel('出租车编号');
ylabel('金额');
legend('合乘方案司机收取费用','非合乘方案司机收取费用');
figure;
bar(cost_taxi_now,'r');
hold on;
bar(cost_taxi_oral,'b');
xlabel('出租车编号');
ylabel('金额');
axis([220.5,260.5,0,150]);
legend('合乘方案司机收取费用','非合乘方案司机收取费用');

end
```

7)计算总时间、等待时间等,并输出最终结果函数（result.m）

```
%% 计算总用时，并按要求输出文件 %%

function
[total_time,total_beside_taxi,taxi_used,total_p]=result(filename,start_position,end_position,taxi_pos
ition,flag_t,t,p);
%% 参数定义
```

```
if filename==0;filename='plan.txt'; end
path=['D:\工作文件\硕士研究生\数学建模\2017 省赛\结果\',filename];
t_num=size(taxi_position,1);
p_num=size(start_position,1);

total_time=-1e-10*ones(t_num,1);
total_p=-ones(p_num,1);%每个乘客从上车到下车总用时
total_beside_taxi=-1e-10*ones(t_num,1);%每辆车从接第一个乘客到送完所有乘客的总路程
%% 计算总用车并输出
taxi_used=length(find(flag_t(:,1)>0));
fid=fopen(path,'wt');

fprintf(fid,'%d\n',taxi_used);

%% 开始计算
for t_count=1:t_num
    switch length(find(flag_t(t_count,:)>0))%判断坐了几个人
        case 1
            first_p=flag_t(t_count,1);

total_time(t_count)=d(taxi_position(t_count,:),start_position(first_p,:))+d(start_position(first_p,:),en
d_position(first_p,:));
            total_beside_taxi(t_count)=d(start_position(first_p,:),end_position(first_p,:));
            total_p(first_p)=d(start_position(first_p,:),end_position(first_p,:));
            fprintf(fid,'%d,%d,%d\n',t_count,first_p,first_p);
        case 2
            first_p=flag_t(t_count,1);
            second_p=flag_t(t_count,2);
            [path,wait,total]=istogether(t_count,first_p,second_p,t,p);
            if ~isempty(path)
                total_time(t_count)=total(3)+wait(1);
                total_beside_taxi(t_count)=total(3);
                total_p(first_p)=total(1);
                total_p(second_p)=total(2);
                fprintf(fid,'%d,%d,%d,%d,%d\n',t_count,first_p,second_p,path(1),path(2));
```

```
                else
                    t_count
                end
            case 3
                first_p=flag_t(t_count,1);
                second_p=flag_t(t_count,2);
                third_p=flag_t(t_count,3);
                [path,wait,total]=shorestpathV3(t_count,first_p,second_p,third_p,t,p);
                if ~isempty(path)
                    total_time(t_count)=total(4)+wait(1);
                    total_beside_taxi(t_count)=total(4);
                    total_p(first_p)=total(1);
                    total_p(second_p)=total(2);
                    total_p(third_p)=total(3);

fprintf(fid,'%d,%d,%d,%d,%d,%d,%d\n',t_count,first_p,second_p,third_p,path(1),path(2),path(3));
                else
                    t_count
                end

        end
end

fclose(fid);

p_no_taxi=find(total_p==-1);
for count=1:length(p_no_taxi)

total_p(p_no_taxi(count))=d(start_position(p_no_taxi(count),:),end_position(p_no_taxi(count),:));
end
```