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）：

我们的队号为（填写完整的队号）：

所属学校（请填写完整的全名）：

参赛队员（打印并签名）：

1.

2.

3.

指导教师或指导教师组负责人(打印并签名)：

日期：　　年　　月　　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 **出租车合乘方案优化模型**

摘　　要：

本文针对某出租车公司的出租车合乘业务的编排问题，分析了影响合乘编排的主要因素，为了满足乘客等待时间尽量短和所需出租车数量尽量少的前提条件，建立最优化模型，并且计算出合理的车费费率计算模式。利用图论算法，将优化模型转换为出租车和乘车人点对点的规划问题进行求解，最后得到带有“鲁棒性”约束的最优合乘计划。

在解决第1问的过程中，本文充分考虑现实环境下打车人和司机的利益因素，制定出高效的基于离子群优化算法和遗传算法的合乘方案及其相应算法。本文以公平性为基本原则，综合考虑驾驶员与出行者利益 ，以出行者时间费用成本最小为目标函数 ， 以保障驾驶员合理收益为约束，设计出了基于遗传算法的动态出租车合乘路径动态决策模型，该模型考虑了驾驶员收益、乘客合乘收益、时间窗、运行时间、载客容量、路径合理性、可行域七个方面的约束条件，保障了合乘出行者的时间费用成本最低，使得系统资源效益最大化。

针对问题2，我们权衡了现有的两类合乘定价模型（基于百分比和基于公平性）的基础上，加入运用了帕累托最优原则，提出了基于公平性分摊合乘模式的帕累托出租车定价模型，通过对驾驶员收益与乘客费用变化情况的分析，尊重先上者权益，采用线上者优先权原则，同时在合乘计价标准上，依据帕累托效率改进原则，在不损害任何一方利益的情况下使得多方受益。最终得到人性化的合理出租车合乘定价模型。

针对问题三，首先对提供的乘车人和出租车位置信息进行预处理，将抽象的网格化地点信息映射到网格上，并求解每两点之间的最短行车路线和距离。把抽象的合乘方案设计问题转化为数组上的填数问题，此填数问题既要基于使目标函数值最小化的原则，又要在合理的约束条件之下。目标函数有两个，一是本方案中全部合乘出行者的时间成本，二是出行者乘坐出租车的总费用。约束条件有六点，一是司机的合乘收益应该大于非合乘收益之和，二是乘客合乘费用要少于单独乘车支付的费用，三是乘客等待出租车的时间不能过大，四是乘客合乘所耗费的时间不要过大，五是出租车实时搭载乘客数不能超过3个，六是出租车行车途中不能同时经过一个服务点两次。在以上目标函数和约束条件下，设置好合适的变异因子，在迭代变异的过程中，每一步变异都引起目标函数的变动，动态的采用最有利于减小目标函数值的变异方式，动态决策使得局部最优向全局最优演化发展，逐步得到最优化解决方案。

关键字:出租车合乘编排,动态决策,多目标规划,遗传算法

1 问题的重述

1.1 问题的背景

近年来随着生活水平的不断提高，出租车以其灵活、便捷、舒适的特性给人们带来了诸多便利,尽管如此,出租车一天24小时扫荡式寻客的经营方式和 “一人一车”乘客可单独使用的特点，也暴露出许多问题和不足：出租车空车率高、实载率低，严重浪费交通资源，造成城市交通拥堵，污染空气环境等。在城市未来交通的发展中，由于能源的日益紧缺与保护环境的需要，我们要更加注重提高城市道路的运输效率，于是出租车的合乘业务便应运而生。

1.2 问题的提出

目前合乘业务主要有相同起点与一口价两种模式。其中，相同起点模式中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，而中途不允许一人先上车，中途再招揽其他乘客。很明显，该模式很容易实现。但该模式构成合乘的条件较严格，合乘比例偏低，往往带来生意的亏损等影响。而“一口价”模式是 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，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。容易发现，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。但该模式也有其明显的不足：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。

为应对上述问题，我们提出了出租车合乘业务，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。

1.3 待解决的问题

1）请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；

2）请设计与合乘方案相应的合理的车费计算方法；

3)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。

2 模型假设与符号说明

2.1 模型假设

1）乘客的出发地与目的地之间道路选择可以是任意的，即可以选择出发地与目的地之间的直线路径；

2）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;

3)允许部分出租车仍保持空驰状态；

4）将司机看成一类特殊的乘客，每辆车最大载客4人。

2.2 符号说明

m：出租车的数量；

K：所有出租车的集合，有$K=\{1,2,\dots,m\}$；

n：乘客的数量；

P：所有需要搭乘的乘客的集合，有$P=\{1,2,\dots,n\}$；

$2n$：所有的上、下车站点的数量，包括乘客的出发地和目的地；

S：站点的集合，有$S=\{1,2,3,\dots,2n\}$；

Z：所研究区域内所有点的集合。$Z=S\cup Z'$，其中Z'表示除上下车点外，出租车所经过的道路节点的集合；

B：所有合乘乘客的费率总和（元/km）；

$\mathrm{C_i}$：第i位合乘乘客的单位里程费率（元/ km）；

$\mathrm{d_n}$：合乘模式下，最后下车乘客的出行距离（km）；

$\mathrm{N_i}$：第i位合乘乘客的非直线出行系数；

H：乘客各需求点的集合；

q_{uv}^{k}：u点上第k台车至v的客流量；

V_{k}：出租车k行驶过程中的最大载客容量；

$[\mathrm{a_x},\mathrm{I_x}]$：乘客x在所研究区域中上、下车点的服务时间窗口，$\mathrm{a_x},\mathrm{I_x}$分别为时间窗口的上限与下限；

$TA_{\mathrm{x}}^{\mathrm{k}}$：具体的时刻变量，表示出租车k到达站点x的精确时刻；

TL_x^k：具体的时刻变量，表示出租车离开站点x的精确时刻；

$X_{x,y}^k$：二进制标识变量，说明某辆车j运行过程中是否经过某点。

3 问题分析

3.1 问题分析

为了解决多车辆合乘问题，我们采用一种基于吸引粒子群算法与改进遗传算法的两级算法模型，运用转化思想，先将多车合成问题转化为单车合乘问题，再通过单车内对乘客上下车顺序进行排序，进而得到最优解。具体的求解过程如下：

初次分配乘客：该过程利用吸引粒子群算法，找到车辆行驶过程中的最佳调整半径，将半径内满足条件的乘客划分到某辆合适的车辆中，进而进行下一步的优化过程；

单车路径优化：利用改进的遗传算法，对已经划分到同一辆车上的乘客进行上下车顺序的排序，即规划出一条合适的行车路线，以满足乘客的合乘需求。

在把多车辆合乘匹配问题转换为单车辆合乘匹配问题后，采用遗传算法对其进行求解，将单车乘客内部排序，得到优化的行驶路线。

3.2 理论基础

3.2.1 粒子群优化算法

粒子群优化算法是通过模拟鸟群觅食行为而发展起来的一种基于群体协作的随机搜索算法。通常认为它是群集智能的一种。它可以被纳入多主体优化系统。粒子群优化算法是由Eberhart博士和kennedy博士发明。

PSO模拟鸟群的捕食行为。一群鸟在随机搜索食物，在这个区域里只有一块食物。所有的鸟都不知道食物在那里。但是他们知道当前的位置离食物还有多远。那么找到食物的最优策略是什么呢。最简单有效的就是搜寻目前离食物最近的鸟的周围区域

PSO 从这种模型中得到启示并用于解决优化问题。PSO 中，每个优化问题的解都是搜索空间中的一只鸟。我们称之为“粒子”。所有的粒子都有一个由被优化的函数决定的适应值，每个粒子还有一个速度决定他们飞翔的方向和距离。然后粒子们就追随当前的最优粒子在解空间中搜索。PSO 初始化为一群随机粒子(随机解)，然后通过迭代找到最优解，在每一次叠代中，粒子通过跟踪两个“极值”来更新自己。第一个就是粒子本身所找到的最优解，这个解叫做个体极值 pBest，另一个极值是整个种群目前找到的最优解，这个极值是全局极值 gBest。另外也可以不用整个种群而只是用其中一部分最优粒子的邻居，那么在所有邻居中的极值就是局部极值。

在找到这两个最优值时，粒子根据如下的公式来更新自己的速度和新的位置

$$\mathrm{v}[\ \ ] = \mathrm{v}[\ \ ] + c_1 \times rand(\ \ ) \times (pbest[\ \ ] - present[\ \ ]) + c_2 \times rand() \times (gbest[\ \ ] - present[\ \ ])$$

$$\mathrm{present}[\ \ ] = \mathrm{present}[\ \ ] + \mathrm{v}[\ \ ]$$

v[ ]是粒子的速度，present[ ]是当前粒子的位置. $pbest[\ \ ]$和$gbest[\ \ ]$如前定义$rand(\ \ )$是介于（0，1）之间的随机数. c_1，c_2 是学习因子. 通常 $c_1 = c_2 = 2$.

3.2.2 遗传算法

遗传算法是一类借鉴生物界的进化规律（适者生存，优胜劣汰遗传机制）演化而来的随机化搜索方法。它是由美国的 J.Holland 教授 1975 年首先提出，其主要特点是直接对结构对象进行操作，不存在求导和函数连续性的限定；具有内在的隐并行性和更好的全局寻优能力；采用概率化的寻优方法，能自动获取和指导优化的搜索空间，自适应地调整搜索方向，不需要确定的规则。

遗传算法的基本运算过程如下：

1）初始化：设置进化代数计数器$t = 0$，设置最大进化代数T，随机生成M个个体作为初始群体$P(0)$。

2）个体评价：计算群体$P(t)$中各个个体的适应度。

3）选择运算:将选择算子作用于群体。选择的目的是把优化的个体直接遗传到下一代或通过配对交叉产生新的个体再遗传到下一代。选择操作是建立在群体中个体的适应度评估基础上的。

4）交叉运算：将交叉算子作用于群体。遗传算法中起核心作用的就是交叉算子。

5）变异运算：将变异算子作用于群体。即是对群体中的个体串的某些基因座上的基因值作变动。群体$P(t)$经过选择、交叉、变异运算之后得到下一代群体$P(t+1)$。

6）终止条件判断:若$t = T$,则以进化过程中所得到的具有最大适应度个体作为最优解输出，终止计算。

4 模型的建立与求解

4.1 问题1的建模与求解

从系统最优的角度考虑，模型应合理进行路径安排，提高运输组织效率，尽可能地降低所有出行者时间费用总成本，并保障驾驶员收益。因此以全部出行者时间费用总成本最低作为系统最优的目标，以费率制定的合理性做为公平性指标。

对于本问题，我们需要用最少的车辆数和乘客等待的时间最少的方案，来安排乘客的搭乘需求，城市交通图是很复杂的，乘客点又密密麻麻的覆盖在地图上，车辆点也是不规则分布。乘客的最近邻点可能还是别的乘客的最近邻点，司机对周边的乘客选择也是多样的，而且还要合乘，计算复杂性瞬间上升一个数量级，局部最优，并不等于全局最优，所以，我们需要一种智能化的搜索方式。

粒子群算法和遗传算法恰好适用于本问题的求解。粒子群算法能帮助我们从局部最优来考虑每个乘车人或每辆出租车的选择，而遗传学算法通过迭代变异，

不断向全局最优靠拢，最终生成合乘的总方案。

为了实现粒子群算法和遗传算法，我们必须设置严格的粒子群优化规则和遗传变异规则。规则主要有以下两点，一是目标函数，二是约束规则。目标函数和约束规则的选择对最终合乘方案的正确性起决定性作用。以下将详细说明针对这两种算法实现的目标函数，决策变量和约束规则。

4.1.1 目标函数

人们在出行中选择出租车作为出行方式，主要是方便、快捷。而在合乘时，对于单个出行而言，合乘时间消耗未必是最低，但是对整体要保证全部合乘出行者的时间费用成本最低。在合乘中主要是优化资源配置，提高系统运行效率和服务水平，使得系统资源效益最大化。

所以，针对本问题，具体有两个因素值得注意，也就产生两个目标函数：

1）时间成本

目标函数中的时间成本由系统中全部合乘出行者的时间成本共同构成。这里的时间成本主要是指车辆运行中的时间：

$$\sum_{k\in K}\sum_{u\in H}\sum_{v\in H}\sum_{i\in G}\sum_{j\in G} x_{ij}^{k} t_{ij} q_{uv}^{k}$$

式中：K表示车辆的集合，H为乘客各需求点的集合，G为路网中各节点的集合，t_{ij}为$(i-j)$的运行时间，q_{uv}^{k}为u点上第k台车至v的客流量。

2）总费用最小

目标函数中的费用成本是指出行者乘坐出租车需要支付的费用。但是在合乘时，由于出租车搭载多个乘客，各个乘客出行目的各异，合乘费用应由合乘者共同分担。对于不同出行者需要制定不同的费率，保障出行者之间的公平性。

$$\sum_{k\in K}\sum_{u\in H}\sum_{v\in H} R_{uv}^{k} q_{uv}^{k} P_{uv}$$

$$P_{uv} = r_0(D_{uv} - L_0) + C_0$$

$$R_{uv}^{k} = f(D_{uv}^{kr}, \left(\frac{D_{uv}^{kr}}{\sum_{u\in H}\sum_{v\in H} D_{uv}^{kr}}\right), N_k, (r_0\left(\sum_{i\in G}\sum_{j\in G} x_{ij}^{k} D_{ij} - L_0\right) + C_0 N_k))$$

$$D_{uv}^{kr} = D_{uv}^{k} - D_{uv}$$

式中，P_{uv}为常规出租车的价格，C_0为起步价，L_0为起步价公里数，r_0为公里单价，D_{uv}^{kr}为第k辆车在需求点$(u-v)$的绕行距离。

为了将两者统一起来，设定一个时间费用转化系数α，通过转化系数α的转化，最终构成以时间费用总成本为目标的目标函数：

$$\mathrm{MinZ}=\sum_{\mathrm{k}\in\mathrm{K}}\sum_{\mathrm{u}\in\mathrm{H}}\sum_{\mathrm{v}\in\mathrm{H}}\sum_{\mathrm{i}\in\mathrm{G}}\sum_{\mathrm{j}\in\mathrm{G}}\mathrm{x}_{\mathrm{ij}}^{\mathrm{k}}\mathrm{t}_{\mathrm{ij}}\mathrm{q}_{\mathrm{uv}}^{\mathrm{k}}+\alpha\sum_{\mathrm{k}\in\mathrm{K}}\sum_{\mathrm{u}\in\mathrm{H}}\sum_{\mathrm{v}\in\mathrm{H}}\mathrm{R}_{\mathrm{uv}}^{\mathrm{k}}\mathrm{q}_{\mathrm{uv}}^{\mathrm{k}}\mathrm{P}_{\mathrm{uv}}$$

4.1.2 决策变量

对于本问题，有两个决策变量，驾驶员的路径选择和乘车人的费用计算：

1）路径变量

设$\mathrm{x}_{\mathrm{ij}}^{\mathrm{k}}$为路径变量，为 1 时，表示第$k$台车经过边$(i-j)$；为 0 时，表示不经过。

2）费用变量

设$\mathrm{R}_{\mathrm{uv}}^{\mathrm{k}}$为费用决策变量，表示第$k$台车$u$点上车到$v$点下车的乘客的合成费率。

4.1.3 约束条件

本文题需要实现出租车数量尽量少，乘客等待时间少这两个明显的约束，但是还有很多隐性的约束值得注意，否则就得不出最优解，综合对本问题的理解和对现实生活中的隐性要求，我们主要总结出以下七点需要注意的约束：

1）驾驶员收益约束

合乘既要满足乘客的成本利益，同时也要考虑驾驶员或运营者的盈利情况。如果合乘只顾乘客利益而不顾运营者的盈利，则驾驶员的积极性会被打消。所以通过合乘，驾驶员的收入情况至少要等于或者大于非合乘时的收入，这样才能保证运营者的盈利，促使合乘顺利实施。

$$r_0(D_k^0-N_k^0L_0)+C_0N_k^0\le\sum_{u\in\mathrm{H}}\sum_{v\in\mathrm{H}}R_{uv}^kq_{uv}^kP_{uv}$$

$$\sum_{u\in\mathrm{H}}\sum_{v\in\mathrm{H}}R_{uv}^kq_{uv}^kP_{uv}\le r_0\left(\sum_{u\in\mathrm{H}}\sum_{v\in\mathrm{H}}D_{uv}^k-N_kL_0\right)+C_0N_k$$

式中，N_k^0为第k辆车在当前路径距离下非合乘载客人数；D_k^0为第k辆车在当前路径距离下非合乘载客距离，N_k为第k辆车接受预约的乘客总数，D_{uv}^k为第k辆车经过需求点$(u-v)$的距离。

2)乘客合乘收益约束

对于单个合乘出行乘客，最后的总费用要低于自己独自打车出行的费用，如果不能满足这一点，合乘对于乘客而言就没有明显的吸引力，同时还可能花费更多的时间成本。所以，每一辆车的每一位乘客在合乘时的费用要小于非合乘时的费用，即折扣率小于 1：

$$R_{uv}^k\le1$$

3）时间窗约束

在车辆到达之前，乘客等待时间过长或者车辆未准时到达都会影响服务质量。目前乘客对服务要求越来越高，通过对时间的限制要求既能保证其服务质量，又

能提高出租车的运输效率。所以，合乘车辆须满足乘客要求，在其要求的时间范围内到达乘客候车地点：

$$\mathrm{e}_{uv} \le W_{uv}^{k} \le l_{uv}$$

式中，e_{uv}为出租车最早抵达时间，l_{uv}为出租车最晚抵达时间。

4）运行时间约束

合乘车辆在满足乘客时间要求的同时，在行驶过程中也需要高效快速地运行。即使车辆存在绕行，其形式时间都不得超过规定的最长时间，只有这样才能满足快速的要求。同样，在出租车辆等待乘客的同时也提高了对乘客的要求 ，即等待时间不能过长，否则就会影响其后乘客的正常时间。车辆必须在下一乘客规定的等候时间内完成等待、载客等任务：

$$A_{uv}^{k} \le a_{uv}$$

$$B_{j}^{k} \ge (B_{i}^{k} + d_{i} + t_{ij})x_{ij}^{k}$$

式中，$\mathrm{A}_{\mathrm{uv}}^{\mathrm{k}}$为搭乘第$k$台出租车从$u$至$v$的运行时间，$\mathrm{a}_{\mathrm{uv}}$为最长运行时间，$\mathrm{B}_{\mathrm{j}}^{\mathrm{k}}$为第$k$台车在$j$载客点上的开始服务时间，$\mathrm{d}_{\mathrm{i}}$为车在$i$点的停留时间。

5）容量约束

在车辆运行过程中，出租车的载客量为小于等于最大载客容量，所以在调度系统实行出租车的调度与管理的时候，要确定出租车剩余的载客容量。只有保证当前服务的情况下才能满足下一组乘客的需求：

$$Q_{j}^{k} \ge (Q_{i}^{k} + q_{j}^{k})x_{ij}^{k}$$

$$max\ \{0, q_{u}^{k}\} \le Q_{u}^{k} \le \min\{Q_{k}, Q_{k} + q_{u}^{k}\}$$

式中，Q_{j}^{k}为第k台车从j出发时的载客量，Q_{k}为出租车的最大容量。

6）车辆合理路径约束

车辆在运行过程中可以存在绕行，但是不允许在一次服务中到达一个地点两次，即在所有需求点有且只能有一辆车访问一次：

$$\sum_{\mathrm{j}} x_{ij}^{k} = 1$$

$$\sum_{\mathrm{j}} x_{ij}^{k} = \sum_{i} x_{ji}^{k}$$

7）可行域约束

$$x_{ij}^{k} = \{0，1\}$$

4.1.4 算法设计

由于出租车合乘问题是一个混合整数规划问题，涉及到的约束条件较多，一般的方法虽然能够对其求解，但往往结果不够理想。通常对于这样的NP问题，采用启发式算法容易编程求解，针对本问题的性质，我们需要使用改进的粒子群算法和遗传算法相结合来进行求解。

粒子群算法原理很好理解，但只能实现局部最优，但目标是实现总体合乘方案的最优。所以只能以遗传算法为主，粒子群算法只用在遗传算法的原子优化上，也就是说，对每辆车选择路线这个小问题上我们采用粒子群算法来优化变异规则。

以下是对出租车乘车方案的遗传算法预处理和步骤：

1）变量编码与遗传算子

采用实数编码的方式，编码主要是给出各辆出租车经过需求点和需求点上客数量。每个初始解包含按照出租车顺序所排列的各个出租车所经过的出行需求点和搭载出行者人数。对于遗传算法中的选择计算，采用轮盘赌的方式，交叉计算采用两点交叉的方式，变异采用单点变异的方式，同时为了更好地求得最优解，采用自适应交叉变异概率，通过设定目标函数最大累计伪变化代数，提高交叉变异概率，加速种群的进化，避免陷入局部最优点。

2）约束处理

在计算中，首先考虑容量约束，如果该约束不满足，则不需要继续计算，让该个体的适应度函数值直接排在最后一位。其次要考虑公平性约束，即驾驶员的收益约束与乘客合乘收益约束，如果不能满足约束条件，要对目标函数给予惩罚，使其在进化过程中自动被淘汰掉。而对于不能满足时间窗约束、运行时间约束以及车辆路径合理性约束的个体，给予适当的惩罚。

3）算法步骤

步骤 1：输入路网结构、路网特性以及出租车最初位置等基础数据；

步骤 2：设置初始进化代数$GEN = 0$和累计目标函数未变化代数$n = 0$，设置最大进化代数$GENmax$和最大累计目标函数未变化代数$namx$，并设定选择、交叉、变异概率和其他计算参数，根据OD出行需求矩阵，随机产生初始种群；

步骤 3：利用$Floyd$计算最短路径，确定个体中各辆出租车运行路径；

步骤 4：判断个体方案是否满足约束条件，如果不满足要给予目标函数惩罚；

步骤 5：计算种群中各个个体目标函数以及适应度函数，对比前后 2 次的目标函数，如果相等，则$n = n + 1$；否则，$n = 0$；

步骤 6：运用轮盘赌方式进行选择操作，如果$n > nmax$，则提高交叉和变异概率；否者，就按照初 始设定交叉和变异概率进行交叉、变异操作，更新种群；

步骤 7：如果$GEN = ENmax$，转到步骤 8；否则，$GEN = GEN + 1$，转到步骤 3；

步骤 8：结束，输出结果。

4.2 问题2的建模与求解

4.2.1 合乘模式的出租车定价制定原则

合乘模式的出租车定价需根据出租车行业的实际情况，结合城市的经济发展水平，遵循以下原则确定：

1） 符合目前我国及各城市对于出租车发展所实施的相关法律和政策；
2） 充分考虑地方的经济发展水平以及居民收入水平，维护出租车行业的稳定性；
3） 避免与轨道交通和常规公共交通形成不公平竞争，优化城市交通结构，充分利用道路资源；
4） 保障出租车驾驶员的利润收益；
5） 为保障乘客乘坐出租车的舒适性，每辆出租车最多允许搭载三位乘客；
6） 模型算法实际操作性强，便于广泛应用。

4.2.2 现有的出租车合乘定价模型分析

现有的出租车合乘定价模型有两类：百分比分摊合乘出租车定价模型和公平性分摊合乘出租车定价模型。

4.2.2.1 百分比分摊合乘出租车定价模型

百分比计费方式主要分为三种：

（1） 合乘路段的百分比

在合乘路段，乘客只需支付合乘路段费用的一定百分比；在乘客单独出行路段，乘客支付全部费用。

（2） 单位里程费率的百分比

乘客支付单位里程费率的一定百分比，单位起步价格全额支付。

（3） 总费用的百分比

只要有合乘行为，乘客就支付出行总费用的一定百分比，而不考虑是否全程合乘。

4.2.2.2 公平性分摊合乘出租车定价模型

公平性要求使出租车驾驶员和乘客利益达到均衡为主要目标。在考虑出租车驾驶员收益的同时，也要兼顾乘客的承受能力。这就考虑到在实施出租车合乘的过程中，不可避免的会出现绕道，而需要弥补长途乘客所受的损失。

4.2.3 基于公平性分摊的出租车合乘定价模型

使出租车驾驶员和乘客的利益达到均衡是合乘模式出租车定价的主要目标。在考虑出租车驾驶员收益的同时，也要兼顾乘客的承受能力。由于乘客出行目的地不同，在实施出租车合乘的过程中，不可避免的会出现绕道，因此考虑弥补长途乘客所受到的损失。

假设每位合乘乘客的费率与非直达系数成反比，出行距离越长，费率越低。基于公平性分摊的合乘模式出租车定价模型为：

1） 在$d_i \le d_0$的情况下，即各合乘乘客的出行距离均在起步距离之内时，

收费标准为：

$$P_i = \frac{C_0}{n}$$

2） 在$d_i > d_0$的情况下，即各合乘乘客的出行距离均在起步距离之外时，考虑以所有合乘乘客的总费率最小为目标：

$$\mathrm{minB} = \sum_{i=1}^{n} C_i$$

实施出租车合乘后，出租车驾驶员的收入应有所增加，即：

$$\sum_{i=1}^{n}[C_i \times (d_i - d_0)] + C_0 \geq C \times (d_n - d_0) + C_0$$

实施出租车合乘后，每位合乘乘客的出行费用应较非合乘模式降低：

$$C_i \times (d_i - d_0) + \frac{C_0}{n} \leq C_i \times (d_i - d_0) + C_0$$

假设每位合乘乘客的费率与非直达系数成反比，即：

$$\frac{C_i}{C_{i+1}} = \frac{N_{i+1}}{N_i}$$

则第i位合乘乘客的出行费用为：

$$P_i = \frac{C_0}{n} + C_i \times (d_i - d_0)$$

4.2.4 合乘模型的优化--基于公平性分摊的帕累托合乘出租车定价模型

针对基于公平性分摊模型进行优化，我们引入驾驶员补偿系数，以确保合乘模式出租车驾驶员收入的增加。

出租车驾驶员补偿系数是考虑在合乘状态下，乘客乘坐出租车的总体需求有可能减少，为保障驾驶员收入增加而建立的出租车驾驶员补偿措施。本文中假设出租车驾驶员收入和出租车行驶距离有关，并且是行驶距离的线性函数。

实施出租车合乘前，出租车驾驶员的总收入为：

$$\mathrm{F_0} = \mathrm{g_0} \times \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{d_i}$$

考虑实施出租车合乘后，出租车驾驶员的总收入最大化。由于搭载的乘客数量越多，出租车驾驶员的总收入越大，因此假设合乘中每一合乘路段均有三位合乘乘客，则实施出租车合乘后，出租车驾驶员的总收入为：

$$\mathrm{F'} = \mathrm{g_1} \times \left(\sum_{\mathrm{i=1}}^{\mathrm{n-n_1}} \mathrm{d_i} + \sum_{\mathrm{j=1}}^{\mathrm{n_1}} \mathrm{d_j} - \frac{2}{3} \sum_{\mathrm{j=1}}^{\mathrm{n_1}} \Delta \mathrm{d_j} \right) = \mathrm{g_1} \times \left(\sum_{\mathrm{i=1}}^{\mathrm{n_1}} \mathrm{d_i} - \frac{2}{3} \sum_{\mathrm{j=1}}^{\mathrm{n_1}} \Delta \mathrm{d_j} \right)$$

根据实施出租车合乘后，出租车驾驶员收入不能减少的原则，得到下式：

$$\mathrm{g_1} \times \left(\sum_{\mathrm{i=1}}^{\mathrm{n_1}} \mathrm{d_i} - \frac{2}{3} \sum_{\mathrm{j=1}}^{\mathrm{n_1}} \Delta \mathrm{d_j} \right) \geq \mathrm{g_0} \times \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{d_i}$$

则出租车驾驶员补偿系数ω的临界值为：

$$\omega = \frac{\mathrm{g_1}}{\mathrm{g_0}}$$

当有乘客上下车时，记为一个状态，乘客合乘出租车状态如下图所示。

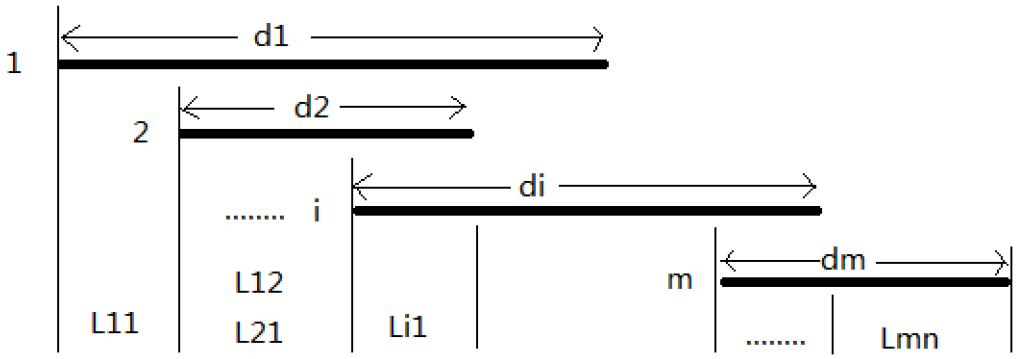

图 4.1 乘客合乘出租车状态图

S_i为第 i个乘客的出行距离，m为在某一合乘过程中车上的乘客数量，n_i为第 i个乘客在合乘出行的过程中合乘人数产生变化的状态的数量；d_{ij}为第 i个乘客在状态j下的出行距离。

对于出租车驾驶员来说，乘客在合乘的过程中，上下车次数增加。因此，考虑将乘客上下车过程中所浪费的时间转化为乘客应付的费用C_t，具体计算公式如下：

$$C_t = \frac{B_m}{30 \times h \times 60}$$

实施出租车合乘后，考虑引进出租车驾驶员补偿因子的情况下，第i个乘客的合乘出行费用为：

$$P_i = \begin{cases} \omega \times \left(\dfrac{C_0}{X_i} + \sum_{j=1}^{n_i} C_{ij} \times d_{ij} + C_t \right) & \sum_{j=1}^{n_i} d_{ij} > d_0 \\ \omega \times \left(\dfrac{C_0}{X_i} + C_t \right) & \sum_{j=1}^{n_i} d_{ij} \le d_0 \end{cases}$$

C_t为合乘中乘客需花费的时间转换费用，B_m为出租车驾驶员人均月收入，h为出租车驾驶员平均每日工作时间，x_{ij}为第 i个乘客合乘时的最大人数（$X_i = \max\{x_{ij}\}$，C_{ij}为第 i个乘客在状态j下的单位里程费率。

为了直观的理解，加入帕累托最优原理后，考虑出租车司机、乘客、社会、出租车公司四方利益的合成部分费用计算方法帕累托模型如下：

		各种合成情况下各个乘客所付费率与原来费率的百分比					
		A		B		C	
上车	1	●	100%				
	2	●	70%	●	70%		

可能出现的情况	3	●	60%	◆	80%		
	4	●	40%	◆	60%	■	80%
	5	●	50%	●	50%	◆	80%
	6	●	50%	◆	65%	◆	65%
	7	●	60%	●	60%	●	60%

表 4.1 分时分段合乘帕累托最优合乘计价模型

注：A、B、C、D 分别表示 A、B、C、D 四个不同的人。特别提示：当前面人下车后，后面的乘客要依次向前转换角色。如果乘客实际行程不超过起步价行程，乘客费用为当前状态下所享受折扣乘以起步价。

● 表示第一组乘客情况

◆ 表示第二组乘客情况

■ 表示第三组乘客情况

4.3 问题3的算法流程

步骤 1： 数据预处理

确定输入路网结构 、路网特性以及出租车最初位置等基础数据。

根据题目描述，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。

观察数据发现，很多乘车人的点并不在网格上，根据乘车只能在主干道上行驶和点的就近原则，我们需要将乘车人的点投影到距离最近的主干道上。

如图所示：

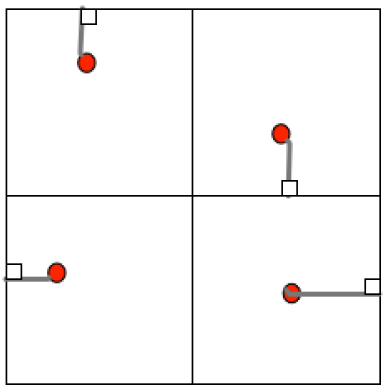

图 4.2 乘车人投影规则

现在来看看怎样在网格限制下，求得两点之间的最短距离，设两点为 A 和 B，坐标分别为 A(x1,y1)，B(x2,y2)。

第一种情况，A 点横坐标和 B 点纵坐标的值恰好为边界值，或者 A 点纵坐标和 B 点纵坐标的值恰好为边界值：

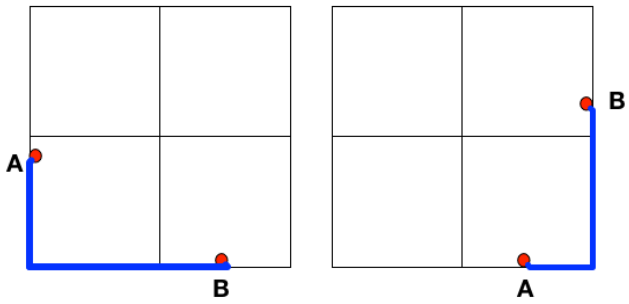

图 4.3 最短距离情况 1

则 A、B 的最短距离为 abs(x1-x2)+abs(y1-y2)。

第二种情况是 A 和 B 的横坐标不在同一网格区间且纵坐标不在同一网格区间：

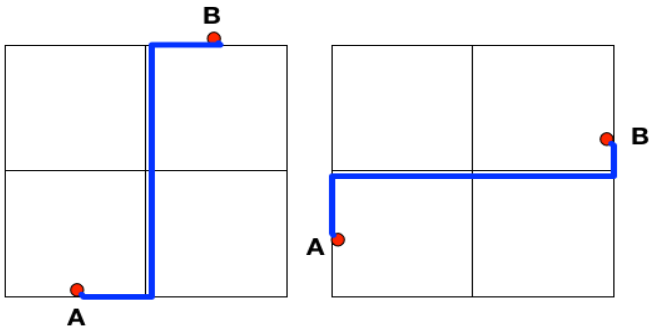

图 4.4 最短距离情况 2

则 A、B 的最短距离为 abs(x1-x2)+abs(y1-y2)。

第三种情况是 A 和 B 的横坐标在同一网格区间或纵坐标在同一网格区间

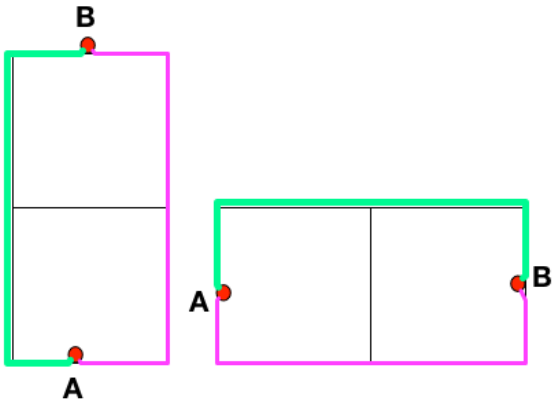

图 4.5 最短距离情况 3

这种情况下就需要比较绿色的路径和红色路径哪一条更短，我们选取更短的那一条作为两点之间的距离。

经过如上分析，我们已经能够将网格上图形化的点转化为能够处理的图了。将每个人的起始出发点、到达点和出租车所在的位置进行编号，总点数是1001*2+667=2669。我们只需记录下每两个点之间的距离就能反映整个乘车人、出租车位置的信息了，路径在网格上的实际路径是我们不必关心的数据。当然，出租车到乘客到达点的距离和出租车到出租车的距离是在求解过程中没有意义的数据，但是，我们不必取用即可。

最后我们用 point_arr 二维数组来表示每个点在网格上对应的点，并用 distance_arr 二维数组来表示没两个点之间的最短距离（假定司机会选择最短路径到达另一个点）。

路网结构：已省去，以每两点之间最短距离来表示。

出租车、乘车人上车点、乘车人目的地均已点的形式保存于 point_arr 二维数组上。

步骤 2：编码规则

我们以 7*667 长度的数组来表示出租车乘坐座位和上下车点的信息，把该一维数组命名为 Variation。

如图所示：

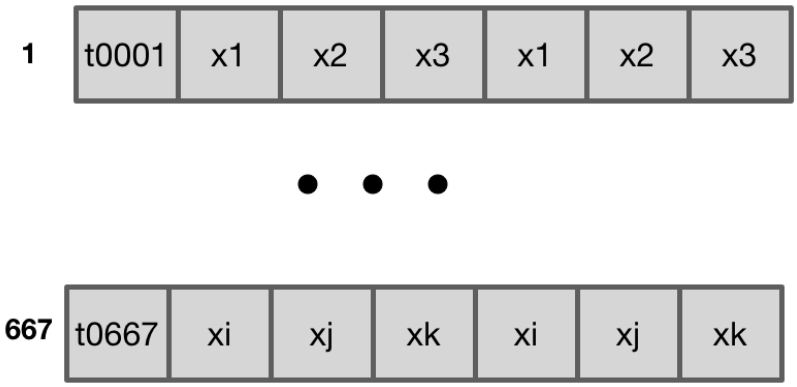

图 4.6 编码规则

该编码 Variation 内每个 7 位中，第 1 位表示出租车编号，后 6 位表示乘客上车或下车，头一次出现表示上车，再次出现表示下车。后 6 位即表示了座位最多不超过 3 个的信息，也表示了上下车的顺序信息，这些数也可以为 0，表示没有人上下车。之所以设置 6 位，是基于一个假设，如果搭载了 3 个以上的人，就会导致乘客等待时间过久，这样对乘客来说，他还需要等前一个人被送到某地址才能来接他，这是不合理的。

步骤 3：变异规则

保证 1001 个人都填入 Variation 的数组内，变异规则就是交换规则。可以是人和人之间的交换，也可以是人和空座位的交换，因为用变量代替，所以不必考虑。存在两种变异情况，如图所示：

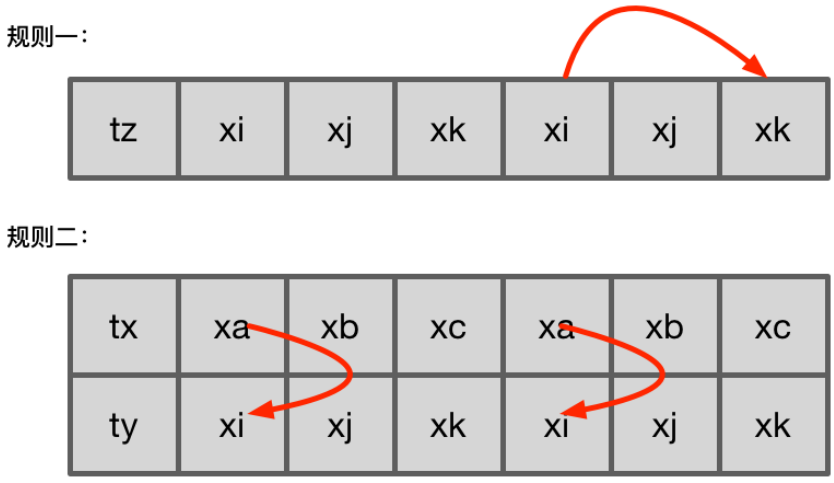

图 4.7 变异规则

这两种情况足以展示所有的变异规则，一是同一辆出租车内部的上下车顺序关系，可以任意两个乘客根据到达目的地的不同来决定谁先下车，同一个司机可

以根据自己确定要接送的某几个乘客决定先接哪一个和先送哪一个。二是不同出租车之间，对同一个乘客，决定他应该坐哪一辆车

步骤 4：设置参数和数据准备

设置初始进化代数 GEN = 0 和累计目标函数未变化代数 n = 0 ， 设置最大进化代数 GENmax 和最大累计目标函数未变化代数 nmax ,并 设定选择、交叉、变异概率和其他计算参数,根据 OD 出行需求矩阵 ,随机产生初始种群。

出租车运行路径：由于已经计算出了两点间的最短路径，并假设出租车会走最短路径，所以可忽略。

步骤 5：约束条件设置

如果不满足约束条件，对目标函数进行惩罚。

除了这些约束外，有一个最明显的约束，两人合乘的条件是两人分别乘车的路线距离之和不能大于合乘总长度。

步骤 6：计算种群中各个个体目标函数以及适应度函数,对比前后 2 次的目标函数，如果相等, 则 n= n+1；否则，n=0。

步骤 7：运用轮盘赌方式进行选择操作 ， 如果 n> nmax ,则提高交叉和变异概率；否者,就按照初 始设定交叉和变异概率进行交叉、变异操作,更新种群。

步骤 8：如果 GEN = GENmax ,转到步骤 8，否则，GEN=GEN+1，一直进行循环迭代变异。

步骤 9：结束 ,输出结果。

结果分析：

发现绝大多数人都能够两人合乘，这样花费较少，且出租车司机也能够赚的更多。

但是发现有 7 人必须单独乘车,现在将所有人所在位置投射到二维坐标系上。这 7 条路线是：

车编号	t0101	t0135	t0296	t0397	t0443	t0444	t0628
上　车	p0288u	p0126u	p0575u	p0287u	p0606u	p0691u	p0573u
下　车	p0288d	p0126d	p0575d	p0287d	p0606d	p0691d	p0573d

7 辆车坐标：

taxi_ID	t0101	t0135	t0296	t0397	t0443	t0444	t0628
now_x	20.6355	22.0081	36.0652	25.7152	22.1683	26.0000	32.7457
now_y	19.5000	18.5000	22.0000	33.5000	26.5000	33.0254	38.0000

7 个人的坐标：

person_ID	p0288	p0126	p0575	p0287	p0606	p0691	p0573
pickup_x	20.0854	21.5488	42.3259	25.7152	21.7525	26.0000	32.7457
pickup_y	16.0000	17.5000	14.5000	33.5000	26.5000	33.0254	38.0000
dropoff_x	20.6516	22.0081	22.5000	26.1417	1.5000	26.3293	31.0000
dropoff_y	18.0000	18.5000	26.2428	31.0000	18.9133	33.0000	36.2064

可以发现，这 7 个人中，有 5 个人的起始坐标到目的地的路线，肉眼看起来都有些不寻常，所以这样更加映证了结果的正确性。

代码说明：

initial.py：将乘客和出租车所在点投射到最近的网格点上。输出 requests_deal.csv 和 point_arr.txt

distance.txt，表示任意两点的最短距离

two_passenger.py：求两乘客最优路线和距离，输出 two_passenger_72508.txt

three_passenger.py:求三乘客最优路线和距离,输出 three_passenger.txt

solver.py：求解出租车的接送路线，输出 answer.txt，也即最终的接送方案

solver2_car_distance.py：求解所有出租车行驶距离，输出 car_distance.txt

4.4 模型评价

出租车合乘的概念在提出以后迅速得到很多人的关注，2000 年以来，相关学者对出租车合乘的方法、理论和技术路线的研究逐渐增多，目前,国内外学者对于出租车合乘主要研究的是合乘的基本理论，本文中，我们以三个已有的比较成熟的出租车合乘模型为例，简单分析之后，从乘客等待时间、所用总车辆数和乘客总花费数三个方面与我们的模型进行比较。最终得出结论：我们在充分考虑要合乘的乘客和司机的利益的基础上,从出租车路径和合乘计费两个方面进行建模,适应性强。我们的模型在尽可能满足乘客等待时间最短的基础上，最大化乘客和司机的利益，同时可以采用尽量少的出租车，使得合乘更加“有效”。

4.4.1 已有出租车合乘理论与技术方案的比较

经过近几年的研究，出租车合乘理论有了一定的创新和发展，本文选择基于矩阵迭代算法的出租车合乘最短路径选择、基于 Floyd 算法的出租车合乘模式的探讨、Z-CJ 出租车合乘计价器设计这三个成果为例，进行简单比较。他们都有各自的创新点，但各自又有各自明显的缺陷，都没有基于消费者和司机的心理选择、缺少技术系统支持等因素，最终导致没有落实。

4.4.1.1 基于矩阵迭代算法的出租车合乘最短路径选择

这一模型利用了运筹学中最短路径的矩阵迭代法建立交通网络模型，从理论上解决出租车在合乘时的路线选择问题。基本满足简单条件下合乘的要求。但是他有明显的缺陷：

首先，该模型对乘客的限制条件太高。它要求乘客必须是在合乘站点上车并且是去往同一方向的两个不同目的地，且目的地之间距离相近。现实生活中这样严格的要求很难满足,很可能导致客流量过低,甚至无法找到满足的乘客等问题。所以该模型对出租车司机的利益问题考虑不够周到，同时，对具有合乘愿景的乘客的需求分析不够到位。

其次，该模型的适应性也很低。它计算出出租车合乘的最短路径之后，要求司机按照该走法并且行驶过程中不再搭载其他乘客。这一要求非常苛刻，试想，如果途中遇到路线非常合适的愿意合乘的乘客，却无法搭载，这对乘客和司机都是不好的要求。

4.1.1.2 出租车合乘模型的探讨

这一模型中，作者创新的提出了合成查询系统，并且利用 Floyd 算法求解了出租车路线中任意两点之间的最短距离。同时，模型中也包括了合成费率的计算，增大的模型的可适用性。但同样，它的缺陷也很明显：

首先，该模式仍是静态模式下的合乘模式。而实际生活中静态分配只是初始状态，运行期间很可能动态发生变化。如果只考虑静态组合，很难适应实际应用需求，造成效率低下，乘客满意度下降等很多问题。

其次，该成果中设计的费用模型，在合乘增加了司机运营成本的情况下，对司机增加的最小收入百分比较少，只有 0～20%，这在实际操作中司机很难认同，从而导致合乘的失败。

4.1.1.3 Z-CJ出租车合乘计价器设计

该模型主要对合乘计价问题进行了创新性的研究，通过增加目的地输入和输出系统，使得模型不局限于静态组合的合成原则，可以在任何地点参与实施合乘行为。可以说，该模型是离产业化阶段最近的研发成果，但是它最致命的缺陷就是他的合乘计价方法，严重不满足出租车司机的利益，导致最终应用的失败。

下表综合展示了上述 3 个成果的分析：

序号	成果名称	发表作者	主要内容	创新点	不足
1	基于矩阵迭代算法的出租车合乘最短路径选择	大连交通大学郭瑞军交通运输工程学院的王晚香	出租车合乘静态组合模式下，运用矩阵迭代算法求出两人合乘最短路径	利用矩阵迭代算法求出了从起点到两个重点的最短路径	局限于动态组合下两人以上不同起点低的路径选择
2	出租车合乘模式的探讨	合肥工业大学机械与汽车工程学院覃运梅和广西工学院汽车工程系石琴	提出建立出租车合乘运营查询系统，并运用 Floyd 算法求出在合乘情况下乘客的不同费率	提出建立合乘查询系统，利用 Floyd 算法实现任意两点之间最短路径的查询，建立了较为有效的合成费率模型	局限于静态组合下多人合乘，在增加了司机运营成本情况下，对司机因合乘运营而增加的最小收入百分比建议是 0～20%，在实际操作中司机不认同。

3	Z-CJ 出租车合乘计价器设计	海军航空工程学院的应朝龙、山东烟台中正新技术有限公司赵剑峰总经理和山东烟台市质量技术监督局刘海滨	介绍了动态情况下具体的合乘计价器的电路组成和主要工作原理，并设计开发目的地中文输入、输出系统。对合乘各方分别打折计费。	在计价器系统中增加中文目的地输入和输出系统，使合乘行为不局限于静态形式。	计价原则有悖于心理行为学，在增加了出租车司机运营成本情况下，没有增加其收入，不利于合乘机制的实施。

表4.2 现有典型合乘成果分析

4.4.2 模型评价

综合分析了上述三个模型，我们从以下几个方面，设计出租车合乘模型。

首先，路径选择，出租车调度方面，在不改变现有的计价标准和规则的前提下，研究出静态和动态组合情况下都适用的路径选择方案。利用启发式算法中成熟的遗传算法，进行出租车对乘客选择以及调度。

其次，在合乘费用计算方面，尊重先上者权益，采用先上者优先权原则。在合乘计价标准上，依据帕累托效率改进原则，在不损害任何一方利益的情况下使得多方受益。

为了更好的体现我们模型的适用性，我们以第三题的数据为例，从乘客总的时间花费、所用总车辆数和乘客金钱总花费三个方面进行比较。选择这三个方面作为评价因素的原因如下：

首先，**乘客总时间花费**。人们在出行中选择出租车作为出行方式，主要是方便、快捷。而在合乘时,对于单个出行而言，合乘时间消耗未必是最低，但是对整体要保证全部合乘出行者的时间费用成本最低。这体现了合乘对所有乘客以及司机的“效率性”。也就是说，在相同数量乘客需求下，所花费总时间越少，说明合乘策略的效率越高。

其次，**使用的总车辆数**。在乘客数量一定的前提下，使用的出租车总数越少，说明合乘的乘客越多，每辆车的利用价值就越高，出租车司机因此获利也会较高。当然，合乘人数越多，会造成对某个乘客的绕行，时间花费增加等问题，所以综合考虑所有乘客的总利益。在车辆总数和总乘客花费时间之间有一个较好的权衡是很重要的。

最后，**乘客金钱总花费**。金钱花费涉及到合乘费率计算模型，如何合理的设计合乘费率是合乘模型中很重要的一部分。合乘价格决定了乘客与司机的参与度。对于司机来说，在同样长度的路线中可以收益高于原本的价格，甚至将近 2-3 倍的收益，是一件很好的事情，而对乘客来说，他们牺牲自己舒适度和时间，所希望换取的就是较少的金钱花费。因此在固定总用户乘客数量的条件下，金钱花费越少，乘客满意度越高。

4.4.2.1 基于遗传算法的出租车合乘路径动态决策模型评价

由于问题三的简化，我们认为出租车是匀速行驶并且各路段没有堵车、红绿

灯等现象，另外我们忽略乘客上下车、付钱等行为所花费的时间，因此，出租车的行驶距离可以直接用来分析乘客时间花费。

如下图，通过比较各个算法下出租车的总运行距离和平均运行距离，我们可以很清楚的看出，相较于矩阵迭代和 Floyd 算法，我们的算法（动态遗传）所使用的出租车在运行距离方面是最短的。无论是总运行距离还是平均每辆车的运行距离，都体现出了较大的优势，从而反映出了我们模型在乘客总时间花费方面是较好的。

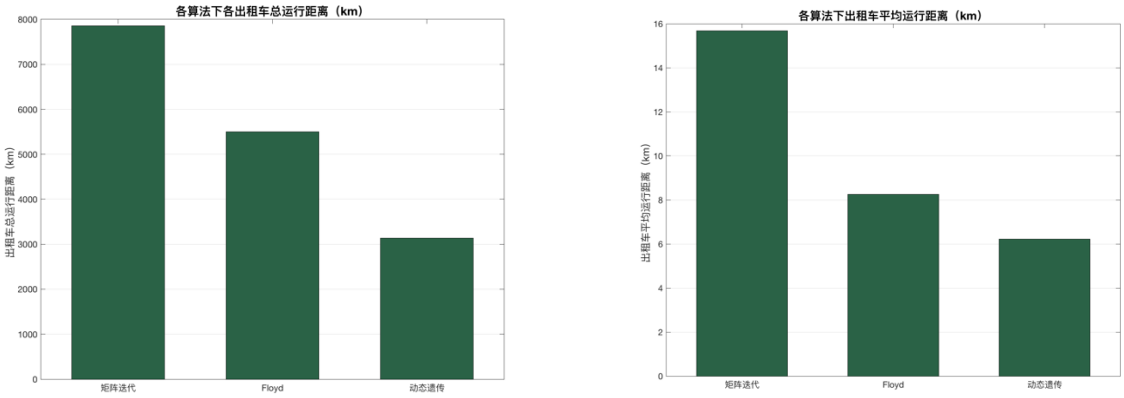

图 4.8 各算法下各出租车运行距离比较

固定了乘客数量的基础上，运用各个模型，得到各个模型下为满足所有乘客的乘车需求所使用的总出租车数量，如下图。从图中可以很清晰的看出，基于 Floyd 算法的模型得到的出租车总数明显高于其他两种算法，实际是使用了能够使用的全部车辆（667 辆）。而矩阵迭代算法和我们基于遗传算法的动态调度均使用了较少的车辆（501 辆和 504 辆），矩阵迭代算法并没有实现动态的乘客增减，而是静态的给每辆车分配了 2 名乘客，然后计算其最短路线。而我们算法在分配之后实时的计算每次分配结果的在我们定义的目标函数下的合适度，而不是死板的固定乘客数量，更不是利用所有车。在我们得到的结果中，发现有部分乘客其要求特殊于其他乘客，很难找到与之相配的其他合乘乘客，因此单独给他分配一辆车，从而最大化整理乘客的利益，实现了动态调度的目的。

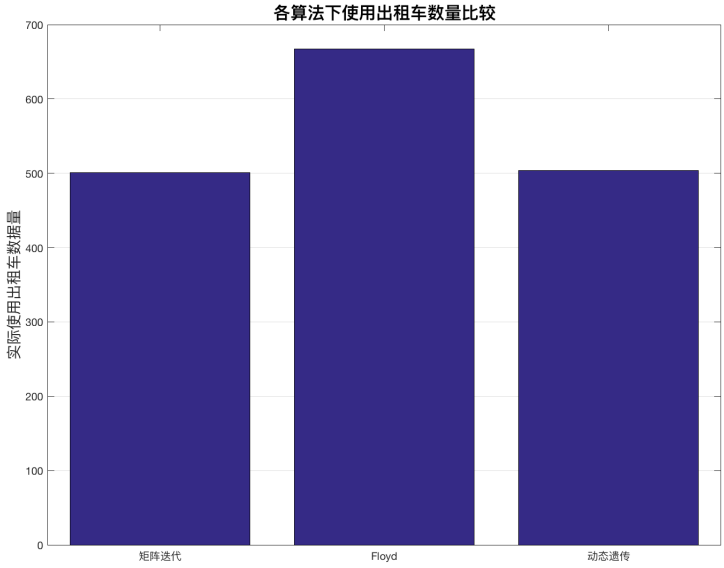

图 4.9 各算法模型下出租车使用数量比较

4.4.2.2 基于公平性分摊的帕累托合乘出租车定价模型评价

经过相关资料的查询，发现现有成熟的出租车定价模型主主要有两大类：基于百分比分摊的出租车合乘定价模型和基于公平性的出租车合乘定价模型。这两类模型拥有各自的优缺点。基于百分比分摊的原理简单，易于实现，但其忽略的一条严重的影响因素，即因为合乘而带来的绕行。基于公平性分摊的原理相较于前者较为复杂，考虑了由于合乘带来的乘客费用的减少同时和司机收益的增加，但是其将合乘绕行与合乘价格的关系考虑的过于简单，容易导致不同乘客。

为了更形象的体会我们基于帕累托最优原理对公平性分摊模型的优化从而得到的新模型的效果，在固定使用基于遗传算法的动态调度路径选择的基础上，我们比较各个模型在问题三中运用后的总乘客的金钱花费，结果如下：
（假设以长沙为例：起步价$C_0 = 8$（元），起步价公里数$L_0 = 2$（km），公里单价$r_0 = 2.5$（元），固定的百分比率为 80%）

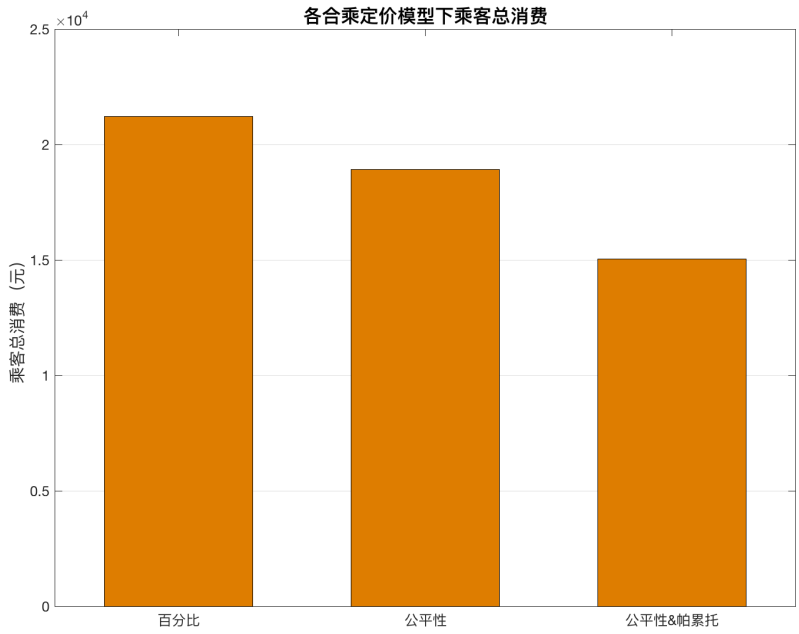

图 4.10 各定价模型下乘客总消费比较

通过上图，可以很清楚的看到，我们的定价模型，在满足出租车司机利益的基础上，产生了最少的乘客消费，最大化提高了乘客的利益。

通过以上三个方面的分析，我们设计的出租车合乘模型在路径选择，动态调度以及合乘定价方面相较于已有的一些成果有了较为明显的改进。在实现动态调度满足现有乘客的基础上，充分考虑影响合乘价格的多项因素，在公平性分摊定价模型的基础上运用了帕累托最优原理，使得乘客和司机双方的利益得到最优的调解。最终得到了较为成熟的出租车合乘模型。

5 总结

实施出租车合乘运营能有效挖掘出租车运力潜能，提高出租汽车服务效率；能有效促进节能减排、有助于降低城市空气 PM 值；能有效缓解交通拥堵和打车难，降低居民出行成本，缩短候车时长，提高社会工作效率；能有效增加出租车司机收入，维持出租车行业的稳定。为了满足乘客等待时间尽量短和所需出租车数量尽量少的前提条件，建立最优化模型，并且计算出合理的车费费率计算模式。根据现有的一些出租车合乘模式和方案，我们从合乘路径选择，出租车调度以及合乘定价等问题入手，利用图论算法，将优化模型转换为出租车和乘车人点对点的规划问题进行求解，建立了基于遗传算法的出租车合乘路径动态决策模型和基于公平性分摊合成模式的帕累托出租车定价模型相结合的出租车合乘模型，并对其进行了一定方面的评价，证明其是一个较为成熟可行的出租车合乘模型。

针对第一个问题，本文充分考虑现实环境下打车人和司机的利益因素，制定出高效的基于离子群优化算法和遗传算法的合乘方案及其相应算法。本文以公平性为基本原则，综合考虑驾驶员与出行者利益 ，以出行者时间费用成本最小为目标函数 ， 以保障驾驶员合理收益为约束，设计出了基于遗传算法的动态出租车合乘路径动态决策模型，该模型考虑了驾驶员收益、乘客合乘收益、时间窗、运行时间、载客容量、路径合理性、可行域七个方面的约束条件，保障了合乘出行者的时间费用成本最低，优化了资源配置，提高系统运行效率和服务水平，使得系统资源效益最大化。同时，把我们的模型和现有的一些比较成熟模型进行了比较，通过实验证明了我们的路径选择模型在具有“鲁棒性”约束的条件下，展现出了较高的“效率”。

针对问题 2，本文充分分析了基于百分比分摊的合乘定价模型和基于公平性分摊的合乘定价模型这两类模型各自的优缺点。在此基础上，我们选择考虑较为充分的基于公平性的定价方法，考虑了帕累托效率，对原始模型进行了一定程度的改进。同时也相应的制定了一些原则，比如先上者优先权原则，即必须尊重先搭乘出租车人的意愿。在合乘费率的计价算法上，采用“共乘分摊，多方受益，不公不变，简单方便”的原则。只对共同合乘行程部分费用进行改变，在原来的计价标准上采取打折计算。以在不损害任何一方利益的情况下使得多方受益为目标，最终得到人性化的合理出租车合乘定价模型，实验证明该模型在不损害司机的基础上，最大化了乘客的利益。

在问题 3 的处理上，由于所给数据及题设的特殊性，我们首先对提供的乘车人和出租车位置信息进行预处理，将抽象的网格化地点信息映射到网格上，并求解每两点之间的最短行车路线和距离。把抽象的合乘方案设计问题转化为数组上的填数问题，此填数问题既要基于使目标函数值最小化的原则，又要在合理的约束条件之下。目标函数有两个，一是本方案中全部合乘出行者的时间成本（即车辆运行中的时间），二是出行者乘坐出租车的总费用。约束条件有六点，一是司机的合乘收益应该大于非合乘收益之和，二是乘客合乘费用要少于单独乘车支付的费用，三是乘客等待出租车的时间不能过大，四是乘客合乘所耗费的时间不要过大（即不要绕太多的远路），五是出租车实时搭载乘客数不能超过 3 个，六是出租车行车途中不能同时经过一个服务点两次。在以上目标函数和约束条件下，采用严格的遗传算法，设置好合适的变异因子，在迭代变异的过程中，每一步变异都引起目标函数的变动，动态的采用最有利于减小目标函数值的变异方式，动态决策使得局部最优向全局最优演化发展，逐步得到最优化解决方案。

6 参考文献

[1]吴芳, 李志成, & 徐琛. (2009). 出租车合乘制调度优化模型研究. *兰州交通大学学报*.

[2]周和平,钟壁樯,彭霞花,&夏西.(2011).出租车合乘路径选择与费率优化模型. *长沙理工大学学报: 自然科学版*, *8*(1), 20-24.

[3] 张亦楠, 魏志强, & 刘昊. (2014). 出租车多人合乘匹配问题的研究. *信息通信*, (3), 70-71.

[4] 李坤远, 马梦崎, 崔乐天, & 韩敬微. 出租车合乘的智能调度及算法研究.

[5] 张薇何瑞春. (2015). 出租车合乘路径演化博弈分析. *兰州交通大学学报*, *34*(6), 70-76.

[6] 邓向林. (2013). 基于动态规划算法的出租车合乘模式研究. *微型机与应用*, *32*(8), 79-81.

[7] 洪麟琳. (2012). *基于合乘模式的出租车定价研究* (Master's thesis, 哈尔滨工业大学).

[8] 于匡贝. (2015). *基于预约模式的出租车合乘路径优化* (Master's thesis, 哈尔滨工业大学).

[9] 傅丽叶. (2003). 智能出租车方案与多人多车搭乘算法. *微型机与应用*, *22*(1), 42-45.

[10] 刘彦蕊. (2003). 出租车最优合乘模式探析及政策建议

7 附录

程序代码

语言：python

程序 1——initial.py：

```
# coding: utf-8
#初始化数据，把乘客位和出租车置点映射到网格上，并求得所有点之间的最短路径
import csv

border=[]
x=0
for i in range(80*2):
    border.append(x)
    x+=0.5

def low_border(x):  #求位置点的边界值
    for i in range(len(border)):
        if border[i]>x:
            return border[i-1]

def near_point(a,b):#求最近的垂直边界点，以将点映射到网格上
    border_a=low_border(a)
    border_b=low_border(b)
    min_d=float(1)
    min_index=0
    #print "a=",a
    #print "border_a=",border_a
    aa=a-border_a
    bb=border_a+0.5-a
    cc=b-border_b
    dd=border_b+0.5-b
    if aa<min_d:
        min_d=aa
        min_index=1
    if bb<min_d:
        min_d=bb
        min_index=2
    if cc<min_d:
        min_d=cc
        min_index=3
    if dd<min_d:
        min_d=dd
```

```
            min_index=4
        if min_index==1:
            return border_a,b
        elif min_index==2:
            return border_a+0.5,b
        elif min_index==3:
            return a,border_b
        elif min_index==4:
            return a,border_b+0.5
def if_on_border(x):    #判断某点是否在边界上
        for i in range(len(border)):
            if border[i]==x:
                return True
        return False
def if_same_border(a,b):#判断两个点是否在同一个边界区间上
        if low_border(a)==low_border(b):
            return True
        else:
            return False

def distance(a,b,c,d):#求两点的最短距离
        #print a,b,c,d
        if ((if_on_border(a) and if_on_border(d)) or (if_on_border(b) and if_on_border(c))): #如果两点在垂直边界上
            #print '1'
            return abs(c-a)+abs(d-b)
        if ((not if_same_border(a,c)) and (not if_same_border(b,d))):    #如果两点都不在同一边界区间内
            #print '2'
            return abs(c-a)+abs(d-b)
        if if_same_border(a,c):#如果在同一边界区间内
            #print '3'
            xx=low_border(a)
            #print 'xx=',xx
            xx1=a+c-xx-xx
            #print 'xx1=',xx1
            xx2=xx+xx+1-a-c
            #print 'xx2=',xx2
            if xx1>xx2:
                xx1=xx2
            #print 'xx1=',xx1
            return xx1+abs(b-d)
        if if_same_border(b,d):
            xx=low_border(b)
```

```
        xx1=b+d-xx-xx
        xx2=xx+xx+1-b-d
        if xx1>xx2:
            xx1=xx2
        return xx1+abs(a-c)

def initial():
    passenger=[]
    num=0
    taxi=[]
    num2=0
    point_arr=[]
    point_arr.append([])
    num_point=0
    csvfile = file('requests.csv', 'rb')
    reader = csv.reader(csvfile)     #读取乘客的乘车点和目的地信息
    for line in reader:
        if line[0]!='person_ID':
            passenger.append([])
            a,b=near_point(float(line[1]),float(line[2]))
            c,d=near_point(float(line[3]),float(line[4]))
            passenger[num].append(line[0]) #存入 passenger 数组中
            passenger[num].append(a)
            passenger[num].append(b)
            passenger[num].append(c)
            passenger[num].append(d)

            num+=1
    csvfile.close()
    #print passenger

    csvfile = file('taxi.csv', 'rb') #读取出租车位置信息
    reader2 = csv.reader(csvfile)
    for line in reader2:
        if line[0]!='taxi_ID':
            taxi.append([])
            a,b=near_point(float(line[1]),float(line[2]))
            taxi[num2].append(line[0]) #存入 taxi 数组中
            taxi[num2].append(a)
            taxi[num2].append(b)
            num2+=1
    csvfile.close()
    #print taxi
```

```
    for i in range(2669):       #把乘客上下车地点和出租车地点放在同一数组内
        point_arr.append([])
    for line in passenger:
        num_point+=1
        point_arr[num_point].append(line[0])
        point_arr[num_point].append(line[1])
        point_arr[num_point].append(line[2])
        point_arr[num_point+1001].append(line[0])
        point_arr[num_point+1001].append(line[3])
        point_arr[num_point+1001].append(line[4])
    num_point=0
    for line in taxi:
        num_point+=1
        point_arr[num_point+2002].append(line[0])
        point_arr[num_point+2002].append(line[1])
        point_arr[num_point+2002].append(line[2])
    #print point_arr
    output=open('point_arr.txt','w')
    for i in range(2669):
        line=point_arr[i+1]
        output.write(str(line[0]))
        output.write(' ')
        output.write(str(line[1]))
        output.write(' ')
        output.write(str(line[2]))
        output.write('\n')
    output.close()

    #print distance(25.3,27.5,25.4,28.5)
    #print distance(25.3,27.5,25.5,28.1)
    print 'ok!'


    distance_arr=[]         #求任意两点之间最短距离
    for i in range(2669+1):
        distance_arr.append([])
        for j in range(2669+1):
            distance_arr[i].append(1.0)

    for i in range(2669):
        for j in range(2669):

    distance_arr[i+1][j+1]=distance(point_arr[i+1][1],point_arr[i+1][2],point_arr[j+1][1],point_arr[
j+1][2])
```

```
        print distance_arr
        output=open('distance.txt','w')#最短距离存入文件当中
        for i in range(2669):
                for j in range(2669):
                        output.write(str(distance_arr[i+1][j+1]))
                        if j!=2669:
                                output.write(' ')
                output.write('\n')
        output.close()


def main():
        initial()


if __name__ == '__main__':
        main()
```

程序 2——two_passengers.py：

```
#!/usr/bin/python
#coding=utf-8
#求个乘车人的最优化路线
distance=[]
distance.append([])
for i in range(2669):
        distance.append([])
line_arr=[]
line_arr.append([])
num_line=0

def read_data():#读取距离信息
        fin=open("distance.txt","r")
        str1=fin.read()
        fin.close()
        list1=str1.split('\n')
        for i in range(2669):
                tmp=list1[i].split(' ')
                distance[i+1].append(float(0))
                for j in range(2669):
                        distance[i+1].append(float(tmp[j]))
                        #print i+1,':=',len(distance[i+1])
        print distance[2669][2669]
```

```
def min_line(x1,x2):        #两人乘车，共有四种乘车方案,返回最优化乘车方案
    a=distance[x1][x2]+distance[x2][x1+1001]+distance[x1+1001][x2+1001]#上下车顺序 a,b,a,b
    b=distance[x1][x2]+distance[x2][x2+1001]+distance[x2+1001][x1+1001]#上下车顺序 a,b,b,a
    c=distance[x2][x1]+distance[x1][x2+1001]+distance[x2+1001][x1+1001]#上下车顺序 b,a,b,a
    d=distance[x2][x1]+distance[x1][x1+1001]+distance[x1+1001][x2+1001]#上下车顺序 b,a,a,b
    min_len=float(5000)
    min_index=0
    if a<min_len:
        min_len=a
        min_index=1
    if b<min_len:
        min_len=b
        min_index=2
    if c<min_len:
        min_len=c
        min_index=3
    if d<min_len:
        min_len=d
        min_index=4
    if min_index==1:
        return x1,x2,x1+1001,x2+1001,min_len
    if min_index==2:
        return x1,x2,x2+1001,x1+1001,min_len
    if min_index==3:
        return x2,x1,x2+1001,x1+1001,min_len
    if min_index==4:
        return x2,x1,x1+1001,x2+1001,min_len

def way_line():        #求出所有多人组合的最优化方案
    global num_line
    global line_arr
    for i in range(1001): #多人组合
        for j in range(i+1,1001):
            a,b,c,d,e=min_line(i+1,j+1)
            if e<(distance[i+1][i+1+1001]+distance[j+1][j+1+1001]):
                num_line+=1
                line_arr.append([])
                line_arr[num_line].append(e)
                line_arr[num_line].append(a)
                line_arr[num_line].append(b)
                line_arr[num_line].append(c)
                line_arr[num_line].append(d)
def print_results():        #打印所有多人组最优化乘车方案和该方案的最短距离
    global num_line
```

```
    global line_arr
    output=open('two_passenger.txt','w')
    for i in range(1,num_line+1):
        for j in range(5):
            output.write(str(line_arr[i][j]))
            if j!=4:
                output.write(' ')
        output.write('\n')
    output.close()


def main():
    read_data()
    way_line()
    print_results()
    print 'num_line=',num_line

if __name__ == '__main__':
    main()
```

程序 3——solver.py：

```
#!/usr/bin/python
#coding=utf-8
#求解程序
import csv

distance=[]
distance.append([])
for i in range(2669):
    distance.append([])
line_arr=[]
line_arr.append([])
num_line=72508

flag_person=[]
for i in range(1002):
    flag_person.append(0)
num_flag_person=0

flag_car=[]
for i in range(668):
    flag_car.append(0)
```

```
answer=[]
for i in range(668):
    answer.append([])

taxi=[]
taxi.append('0')
person=[]
person.append('0')

def read_data():                    #读取距离信息，乘车人信息，出租车信息，多人最优路径信息
    global distance
    global line_arr
    fin=open("distance.txt","r")
    str1=fin.read()
    fin.close()
    list1=str1.split('\n')
    for i in range(2669):
        tmp=list1[i].split(' ')
        distance[i+1].append(float(0))
        for j in range(2669):
            distance[i+1].append(float(tmp[j]))
            #print i+1,':=',len(distance[i+1])
    print distance[2669][2669]
    fin=open("two_passenger_72508.txt","r")
    str1=fin.read()
    fin.close()
    list1=str1.split('\n')
    for i in range(72508):
        tmp=list1[i].split(' ')
        #num_line+=1
        line_arr.append([])
        line_arr[i+1].append(float(tmp[0]))
        for j in range(4):
            line_arr[i+1].append(int(tmp[j+1]))
    print line_arr[72508][3],line_arr[72508][4]

    csvfile = file('taxi.csv', 'rb')
    reader = csv.reader(csvfile)
    for line in reader:
        if line[0]!='taxi_ID':
            taxi.append(line[0])
    csvfile.close()
    print taxi[667]
```

```
    csvfile = file('requests.csv', 'rb')
    reader = csv.reader(csvfile)
    for line in reader:
        if line[0]!='person_ID':
            person.append(line[0])
    csvfile.close()
    print person[1001]


def find_least_persons():   #找到当前最短线路
    global distance
    global line_arr
    global num_line
    min_dis=5000.0
    min_index=0
    for i in range(1,num_line+1):   #num_line 线路数在减少
        if line_arr[i][0]<min_dis:
            min_dis=line_arr[i][0]
            min_index=i
    print 'min_index=',min_index,' len=',len(line_arr[min_index])
    return
line_arr[min_index][0],line_arr[min_index][1],line_arr[min_index][2],line_arr[min_index][3],line_arr[
min_index][4],min_index

def delete_line(x,y):    #已上车的人的线路要去掉
    global distance
    global line_arr
    global num_line
    i=1
    while i<=num_line:
        if (line_arr[i][1]==x) or (line_arr[i][1]==y) or (line_arr[i][2]==x) or (line_arr[i][2]==y):
            del line_arr[i]
            num_line-=1
        else:
            i+=1

def find_least_car(x):
    global distance
    global line_arr
    global flag_car
    min_dis=5000.0
    min_index=0
    for i in range(1,667+1):
```

```
            if (not(flag_car[i]) and distance[i+2002][x]<min_dis):
                min_dis=distance[i+2002][x]
                min_index=i
        return min_index

def solver():
        global distance
        global line_arr
        global num_line
        global answer
        #global num_flag_person
        global flag_car
        car=0
        while num_line>0:                                   #两人线路搜索完毕即停止
            e,a,b,c,d,index=find_least_persons()   #返回当前最短线路
            delete_line(a,b)                                  #人已经上车，删除含这 2 人的线路
            flag_person[a]=1   #标记已上车
            flag_person[b]=1
            #num_flag_person+=2

            index=find_least_car(a)         #距离 a 的最近的车
            flag_car[index]=1                 #标记车已用

            answer[index].append(taxi[index])     #线路存入结果数组
            answer[index].append(person[a])
            answer[index].append(person[b])
            answer[index].append(person[c-1001])
            answer[index].append(person[d-1001])
            car+=1
        remain=0
        ji=0
        for i in range(1,1002):                 #剩余没上车的人单独乘坐
            if not(flag_person[i]):
                remain+=1

        while remain>0:
            for i in range(1,1002):
                if not(flag_person[i]):
                    index=i
            print "loop_death"
            single_car=find_least_car(index)
            flag_car[single_car]=1
            answer[single_car].append(taxi[single_car])
            answer[single_car].append(person[index])
```

```
                answer[single_car].append(person[index])
                remain-=1
                car+=1
        return car

def print_result(car):    #打印乘车方案
        output=open('answer.txt','w')
        output.write(str(car)+'\n')
        for i in range(1,668):
                if len(answer[i])==5:
                        output.write(str(answer[i][0]))
                        output.write(' ')
                        output.write(str(answer[i][1]))
                        output.write(' ')
                        output.write(str(answer[i][2]))
                        output.write(' ')
                        output.write(str(answer[i][3]))
                        output.write(' ')
                        output.write(str(answer[i][4]))
                        output.write('\n')
                elif len(answer[i])==3:
                        output.write(str(answer[i][0]))
                        output.write(' ')
                        output.write(str(answer[i][1]))
                        output.write(' ')
                        output.write(str(answer[i][2]))
                        output.write('\n')
        output.close()
        print "Got the answer!"

def main():
        read_data()
        car=solver()
        print_result(car)
if __name__ == '__main__':
        main()
```