队伍编号	905488
题号	A

数据驱动的城市轨道交通网络优化

摘 要

截至2018 年12 月31 日，中国内地累计共有35 座城市建成并投运城市轨道交通，里程共计5766.6 公里。据2017 年统计，北京城市轨道交通年乘客量全年达到45.3 亿人次，日均客流为1241.1 万人次，单日客运量最高达1327.46 万人次。可见，城市轨道交通已成为大城市居民出行的主要载体，也是城市发展的重要支撑。本次赛题即在此背景下，通过大数据手段来解决在城市轨道交通网络优化的问题。

第一问题目要求我们对几大附件进行探索性数据分析。在附件1的探索性分析中，我们发现了数据集的稀疏性、数据集的收集时间段以及地铁拥挤滞留的现象，总结了人们出行的基本规律。在附件2的探索性数据分析中我们建立了北京地铁路网的无向图模型，并得到了一些与地铁线的信息，例如换乘站等。在附件3的探索性数据分析中我们建立了论证了附件3的数据集的3个主要论段：附件三是以12小时制度记录；附件三中有‘幽灵车’的数据分为两个部分：一部分是地铁早班的数据，一部分是半夜地铁无人的调度数据。附件三中的北京地铁运行图的数据是从前一天晚上的调度运行图到次日中午的运行图数据：即缺失了北京市下午（12-23:50左右）的地铁的运行图数据。

第二问题目要求我们设置一些算法，来解决对特定乘客的路线追踪及乘客辅助路径选择服务。由于第一问我们已经得出地铁运行图数据的缺失，我们首先利用朴素贝叶斯分类器将整个地铁的运行图数据补充完整。接下来根据人的理性假设，设计了一个通用算法来解决定乘客的路线追踪及乘客辅助路径选择服务：先利用Dijkstra算法搜索最短路，再利用BSF算法搜索最短路长度的110%长度的路径作为备选路径。最后结合地铁运行图数据，我们开发了一种综合数据结构用来储存地铁运行安排，并提供时间搜索算法对备选路径花费的时间进行计算与对比。

第三问根据题目要求，我们建立了以乘客出行时间（包括出行时间和滞留时间）最短为目标的单一限流模型，并考虑了站台乘客上下车行为约束、列车位置约束、进站口乘客出行行为约束、单线多站服务系统能力约束，并利用粒子群算法求解，给在第四问出限流方案。

关键词：图论、贝叶斯方法、Dijkstra 算法、BSF 算法、时间搜索算法、乘客限流模型、目标规划、粒子群算法

目录

一、问题重述

截至 2018 年 12 月 31 日，中国内地累计共有 35 座城市建成并投运城市轨道交通，里程共计 5766.6 公里。进入“十三五”以来，三年累计新增运营线路长度为 2148.7 公里，年均新增线路长度为 716.2 公里（2018 中国城市轨道交通协会快报）。表 1 统计了 2018 年中国内地城轨交通运营线路长度排名前 5 的各大城市。以北京市为例，其轨道交通覆盖 11 个市辖区，运营里程约 714 公里，共设车站 391 座，开通里程居中国第二位。此外，据 2017 年统计，北京城市轨道交通年乘客量全年达到 45.3 亿人次，日均客流为 1241.1 万人次，单日客运量最高达 1327.46 万人次。可见，城市轨道交通已成为大城市居民出行的主要载体，也是城市发展的重要支撑。

问题 1:附件 1 给出了北京市某时段部分城市轨道交通线网的乘客 0-D 数据，附件 2 为基础信息数据，附件 3 为该时段的列车运行图数据。依据北京城市轨道交通线网图 (附件 4)，试分析基于以上数据的乘客出行特征，包括出行时段分布、出行距离分布、出行时长分布等。

问题 2:基于问题 1 的路径选择结果，设计一套算法还原乘客出行的准确信息，即 乘客在何时何站搭乘何辆地铁列车（如有换乘，需计算）并在何时何地出站，完成其一次完整的地铁出行，并完整填写表 2(计算乘客编号为 2、7、19、31、41、71、83、89、101、113、2845、124801、140610、164834、193196、223919、275403、286898、314976、315621 的完整出行线路）。另外，设计一套智能算法，以辅助并优化乘客的在轨道交通路网中的路径选择，如通过优化路径可缩短行程、减少拥挤等。

问题 3:假设地铁八通线每列列车容量为 1428 人，列车座位数为 256 座，限流时段长度可根据需要任选，且以 7:00 为首班列车发车时刻，在减小列车超载现象的基础上，尽可能缩短乘客出行时间（包括出行时间和滞留时间），并以此为目标建立城市轨道交通单一线路乘客限流模型。对模型求解后给出具体限流措施以改进八通线的服务水平，具体包括：

问题 3.1:若八通线不限制限流车站个数，试分析限流前后的总出行时间、平均出行时间对比，结果如表 3 所示。

问题 3.2:若八通线限制限流车站个数（分别取限流车站数为 1-5 个车站），试分析限流前后的总出行时间、平均出行时间对比，结果如表 4 所示。

问题 3.3:根据以上分析结果，举例说明八通线两个限流效果最好的车站，并阐述原因。

问题 4: 结合问题 1-3,给出具体限流措施以改进城市轨道交通的服务水平，如动态限流方案、限流车站的选取、限流时段的选择、限流强度等。

二、问题分析

第一问，在基于本经某时段部分城市轨道交通网线乘客的 O-D 数据，结合北京地铁的整体线路图和运行数据的条件上，要求分析乘客的出行特征。这显然是一个探索性数据分析问题。通过对于几个附件数据的研究：生成图论中的无向图表示北京地铁路网、生成距离权重、分析地铁的运行时间表、分析乘客的出行数据等等。

第二问，前半部分要求我们根据乘客的出行数据：上车点、下车点、刷卡进站时间、刷卡出站时间来推断乘客经过的站点。在基于人的理性的假设下，我们假定其必定选择基于距离的最短路。但是在站点中乘客人数造成滞留、多条最短路的影响下，我们需要一套基于列车运行表的时间搜索算法，来对比多条路线在多种情况下所需花费的时间和数据中的时间问题。后半部分要求我们设计一套智能算法，来辅助优化乘客在轨道交通路网中的路径选择：在时间搜索算法与最短路算法的基础上，我们可以提出更多的约束条件，形成一个目标规划模型并选择合适的算法进行优化。

第三问则是要求从管理者的角度出发，以八通线为例设计一个城市轨道交通单一线路乘客限流模型。以最小化乘客的出行时间和滞留时间为目标，加以实际数据的制定相关约束：乘客行为约束、列车状态约束等。最终形成一个目标规划模型并选择合适的算法进行优化

三、模型假设

1. 理性人假设：乘客均是“理性人”。

2. 数据真实性假设：附件数据均为真实数据

3. 关于附件一的假设：将乘客刷卡出站时间等同于乘客下车时间，而刷卡进站时间不等同于乘客上车时间。

4.关于附件一的假设：附件三是以 12 小时制度记录。附件三中有‘幽灵车’数据分为两个部分：一部分是地铁早班的数据，一部分是半夜地铁无人的调度数据。附件三中的北京地铁运行图的数据是从前一天晚上的调度运行图到次日中午的运行图数据：即缺失了北京市下午（12-23:50 左右）的地铁的运行图数据。

5. 注：以上两点关于数据集的假设均在 **5.1 问题一定的模型建立与求解** 均有论证。

6. 附件 1 的乘客数据是未限流下的情况。

7. 以上假定使用范围为前文，对于第三问更多的模型假定在正文中。

四、符号与约定

序号	参数	意义
1	s	为线路单方向上车站顺序编号，s=1,2,3…,S
2	r	为线路单方向上列车顺序编号，r=1,2,3…,R
3	t	为各限流时段顺序编号，t=1,2,3…TE
4	Δt	为每一个限流时段持续时间,单位：min
5	$R_s^{ar}(t)$	为 t 时刻，车站 s 的乘客到达率,单位：人/min
6	Q_s^t	为车站 s 在时间粒度 T 内的总进站量,单位：人
7	$P_{\mathrm{i},s}$	为从车站 i 到车站 s 的乘客占 i 站上车总乘客的比例
8	ρ_s	车站 s 总进站量中，选择本文研究方向出行乘客所占比例
9	γ	为进站客流量放大系数，%
10	f	为线路上列车发车间隔时间，单位：min
11	ΔT_s^{s+1}	为列车在车站 s 和 s+1 的区间运行时分，单位：min
12	Q_s^p	车站 s 的站台可容纳的最大聚集人数能力，单位：人
13	T_{delay}	为乘客总出行延误时间，单位：人· min
14	T_{ql}	为因进站口限流导致乘客延误时间，单位：人· min
15	T_{qr}	为因站台留乘导致乘客延误时间，单位：人· min
16	θ_1	表示因进站口限流导致乘客延误的延误时间惩罚系数
17	θ_2	表示因留乘导致乘客延误的延误时间惩罚系数
18	$q_s^o(t)$	为在 t 时段内，新到达车站 s 进站口等候乘车的乘客人数，单位：人
19	$q_s^h(t)$	为在 t 时段内，车站 s 进站口等候进站的聚集乘客人数，单位：人
20	$q_s^l(t)$	为在 t 时段内，车站 s 进站口被限制未能进入车站的乘客人数，单位：人
21	$q_s^e(t)$	为在 t 时段内，车站 s 进站口进入车站站台的乘客人数，单位：人·min
22	$q_s^p(t)$	为在 t 时段内，车站 s 进站口进入车站站台的乘客人数，单位：人·min

23	$q_s^a(t)$	为在 t 时段内，在车站站台下车的乘客人数，单位：人·min
24	$q_s^b(t)$	为在 t 时段内，在车站 s 站台候车乘客中的上车人数，单位：人·min
25	$q_s^d(t)$	为在 t 时段结束时，车站 s 站台候车乘客中的留乘人数，单位：人·min
26	$q_s^m(t)$	为在 t 时段结束时，列车 r 内的载客人数，单位：人·min
27	$q_r^{av}(t)$	为在 t 时段结束时，列车 r 可允许上车的人数，单位：人·min
28	$L_{s,r}(t)$	为在 t 时段开始时，列车 r 是否位于车站 s 内，为 0-1 变量
29	C	列车额定载客人数，单位：人
30	$\eta^{\max}$	最大列车满载率系数，%

五、模型的建立与求解

5.1 问题一的模型建立与求解

5.1.1 附件一的探索性数据分析

对于附件一，给出的数据是有关于线网乘客 O-D 数据，来源的时间段是某个北京市的时间段，我们首先统计乘客出行的时间段，如图 1 所示。注意附件一记录时间是 24 小时制。

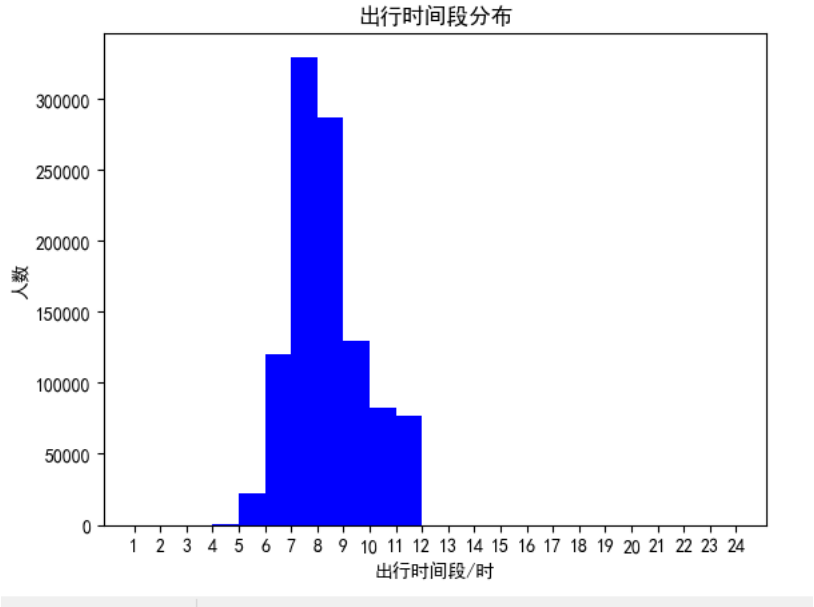


图 1. 附件一中乘客出行时间段

从图 1 可以看出，附件二的数据主要集中 4 点到 12 点。基本成高斯分布在此区间内。值得提出的是据统计，附件一一共给出约 105 万出行数据，其中有 100 万数据是上午的出行数据，时间区间在 5-12 时；5 万出行数据是下午的出行数据，时间区间在 12-24 时之间。此数据集有十分大的稀疏性存在：上午的数据占了十分大的比例，但是下午的数据并不是非常多。从理论上来说，关于乘客的分布应该如图 2[1] 所示。

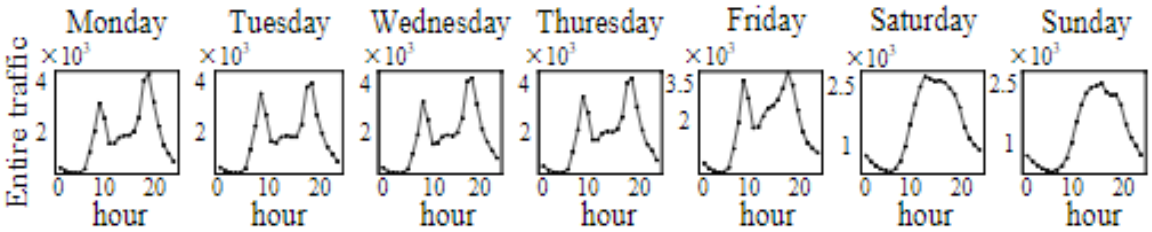


图 2. 公共出行交通系统乘客流量的一般分布

公共出行交通系统乘客的一般在一周的工作日（周一至周五）呈双峰分布，在非工作日呈单峰分布：工作日人们早上出行上班，晚上下班回家；周末一般中午至下午是出行高峰期。地铁作为公共出行交通系统的一部分，也必然符合该规律。

据此，对于附件一我们有以下推断：

1. 附件一的数据采集于北京市某一工作日（周一至周五）。
2. 附件一的数据极度稀疏，不足以用来描述下午乘客的出行状态。

附件一中的数据除了能够进行出行时间段的统计，还能做出出行时间的统计，还有出行距离的数据，结果如图 3 与 图 4 所示。

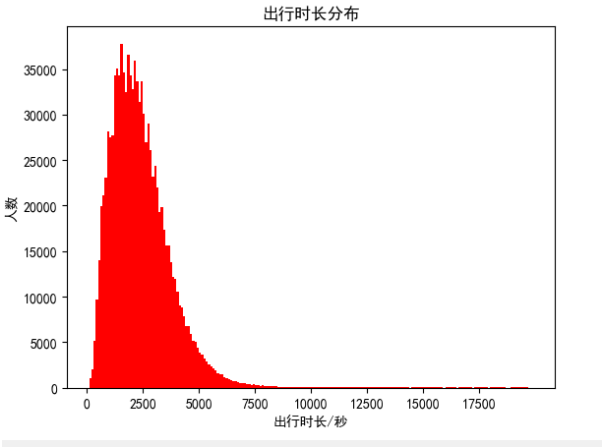


图 3. 乘客出行时长分布

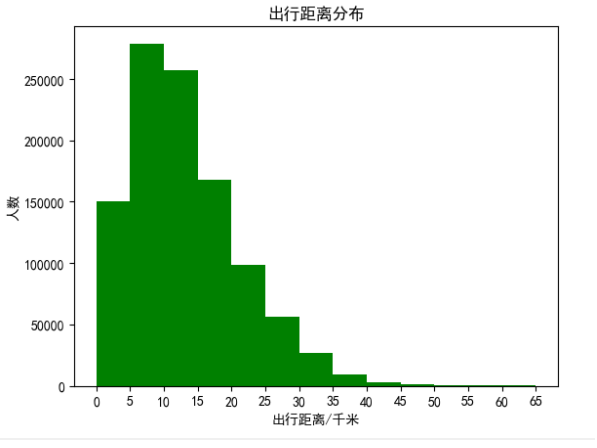


图 4. 乘客出行距离分布

这里的出行距离的计算方式是乘客的进站点和出站点的直线距离，我们从【2】获取北京地铁各个站点的经纬度，并利用以下球面三角距离公式计算两点之间的球面距离：

$$D=\mathrm{R}\cdot\arccos[\cos\beta_1\cos\beta_2\cos(\alpha_1-\alpha_2)+\sin\beta_1\sin\beta_2]$$

其中: D 是两地铁站直线距离，R 是地球的半径，在这里取 6371 km；(β_i,α_i) 是 i 地的经度角、纬度角二元数组。

图 3 与 图 4 的分布具有一定的类似性，原因在于出行距离一般和出行时间成正比：距离越长，在路上花费的时间就越长。但是我们发现最高峰的频数不同，此处可以得出在出行过程中，地铁站出现由于拥挤造成的滞留现象导致了相同的出行公里数不对应相同的出行时间：分布频数发生了偏移，两个分布的方差是不同的。当然需要指出的是这也可能由换乘站的数量发生一定的偏移，但是我们的数据频数大概有 10000 的偏移，这个偏移量过大，因此可以判断地铁必定出现了拥挤滞留现象。于是我们提出了如下对附件一的数据假设：将乘客刷卡出站时间等同于乘客下车时间，而刷卡进站时间不等同于乘客上车时间。

附件一的乘客数据理论上是对背景一天出行数据的一个随机抽样，在最后我们讨论一下这个随机抽样与原始分布的规格大小。由题目我们可知：据 2017 年统计，北京城市轨道交通年乘客量全年达到 45.3 亿人次，日均客流为 1241.1 万人次，单日客运量最高

达 1327.46 万人次。由【1】我们已知人数分布为双峰分布，于是我们的 4-12 时的 100 万数据约是整体分布的 $\frac{100}{\frac{12-4+1}{24-4+1}\times 1241}\approx\frac{1}{5}$ 的抽样。于是我们引入扩大因子 λ 以求获得更好的人群总体：其中上午（4-12 时）的扩大因子 λ_{5-12}=5，下午（12-24 时）的扩大因子 λ_{12-24}=150 。

5.1.2 附件二的探索性数据分析

附件二一共被分成两个部分，第一个部分是北京的地铁站编号，第二个部分是北京的地铁路线。我们对附件二中的数据的主要处理就是生成北京地铁的网络结构图。由于北京的地铁中：非环线的均为双向地铁，环线（如 10 号线）均为内外环结构。所以我们采用的是无向图。利用 Python 的图论计算包——NetWorkX，我们可以构建北京的地铁网络，如图 5 所示。

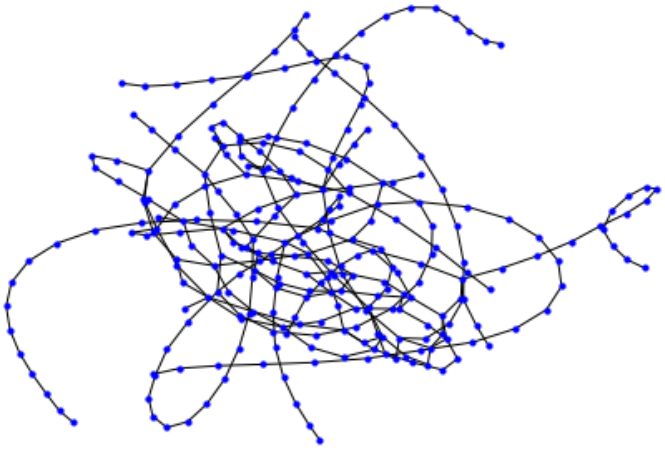


图 5. 北京地铁无向图路网

通过附件二我们还可以获得一些关于地铁路线的信息，如换乘站等。由于换乘站是要做特殊处理的，属于有用的数据，在这里列出举例列出各个换乘站：

['北土城', '角门西', '军事博物馆', '三元桥', '慈寿寺', '霍营', '奥林匹克公园', '双井', '望京', '望京西', '西二旗', '七里庄', '芍药居', '朝阳门', '国家图书馆', '立水桥', '海淀黄庄', '惠新西街南口', '四惠', '菜市口', '西单', '六里桥', '白石桥南', '公主坟', '大屯路东', '建国门', '

朱辛庄', '金台路', '宣武门', '宋家庄', '北京西站', '平安里', '复兴门', '崇文门', '东四', '郭公庄', '西直门', '车公庄', '四惠东', '东单', '国贸', '鼓楼大街', '南锣鼓巷', '西局', '磁器口', '呼家楼', '知春路', '东直门', '雍和宫']

5.1.3 附件三的探索性数据分析

附件三给出的是北京地铁的运行图数据，在这里我们在进行探索性数据分析后给出以下几点，后面我们将一一论证，并列为数据假设。先列出：

1.附件三是以 12 小时制度记录。

2.附件三中有‘幽灵车’数据，可以分为两个部分：一部分是地铁首班车的数据，一部分是半夜地铁无人的调度数据。

3.附件三中的北京地铁运行图的数据是从前一天晚上的调度运行图到次日中午的运行图数据：即缺失了北京市下午（12-23:50 左右）的地铁的运行图数据。

现在我们对以上三点进行一个连续性论证：

对于附件三，我们先统计了所有车子离开站点的时间分布图，如图 6 所示：

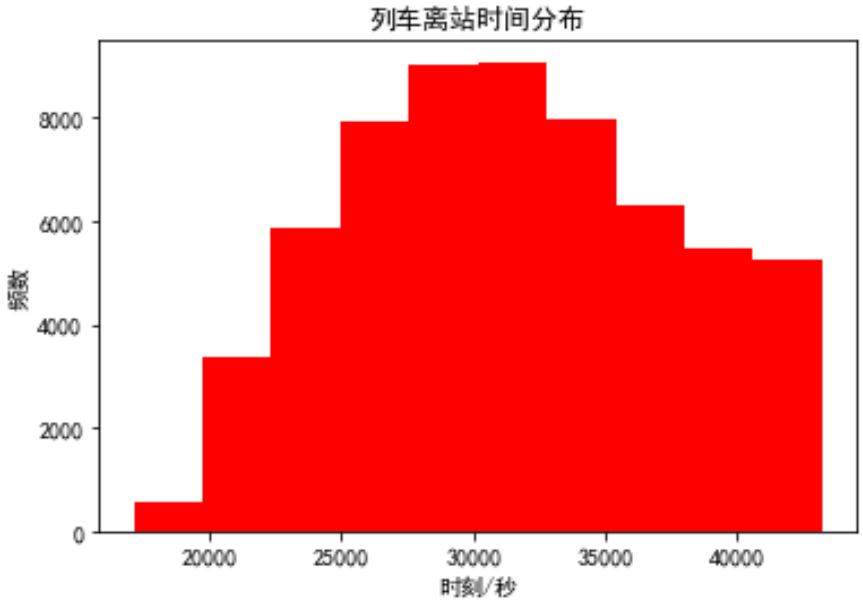
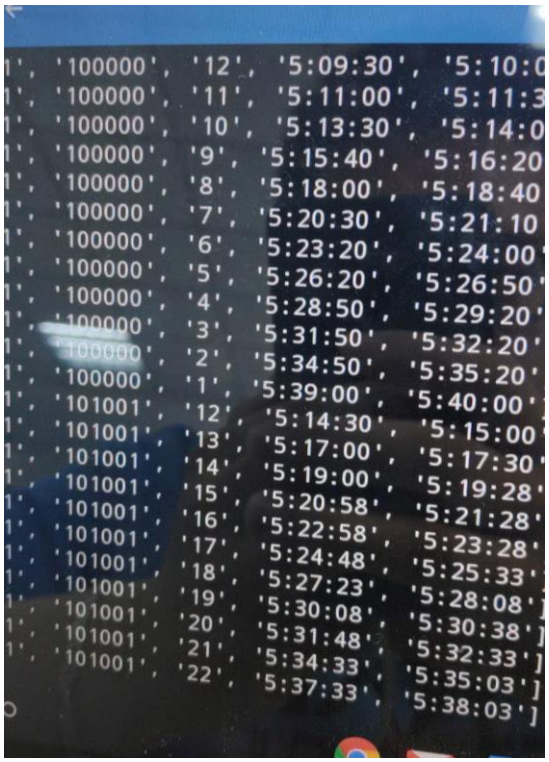


图 6. 列车离站时刻分布

从分布图中我们可以看见列车离开站点的时刻分布，最大的离开时刻约在第 43000，也即 12 时左右，由此可以推论，本数据的时间记录方式是以 12 小时为制度记录。由此第一个结论被论证了。

接下来我们从数据里面发现了一部分“幽灵车”的出现。在这里先给出幽灵车的定义：对于一般的地铁来说，都是从首站发车到尾站，也即每一个车次走过的站点数均和地铁路线中的站点数是一致的，我们把站点数不一致的车次成为“幽灵车”，我们以一号线为例给出一些典型的“幽灵车”的例子，如图 7 和 图 8 所示。



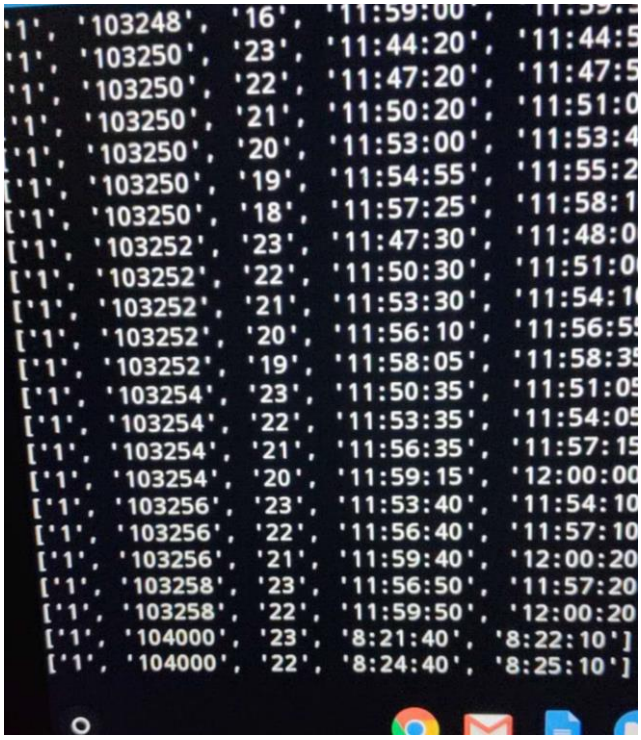


图 7. 早上的“幽灵车”　　图 8. 晚上 / 中午的“幽灵车”

在这里我们以下算法将“幽灵车”的车次遴选出来：

1. 利用附件 2 统计每一地铁路线的总站数，记录为 $Num_{(i)}$ ， i 指代地铁编码。
2. 利用附件 3 将统一车次编号的地铁放在同一个集合， $T_{(i)}^{(\overline{abcdef})}$ 。其中 $\overline{abcdef}$ 是车次号， i 指代地铁编码。
3. 利用以下判断将“幽灵车”至于同一集合， $GhostTrain$ 。

   If $\text{len}(T_{(i)}^{(\overline{abcdef})}) \neq Num_{(i)}$:

   $GhostTrain.\text{add}(T_{(i)}^{(\overline{abcdef})})$

在初步遴选完成“幽灵车”后，我们统计了每一条线上“幽灵车”的比例，如图 9 所示。

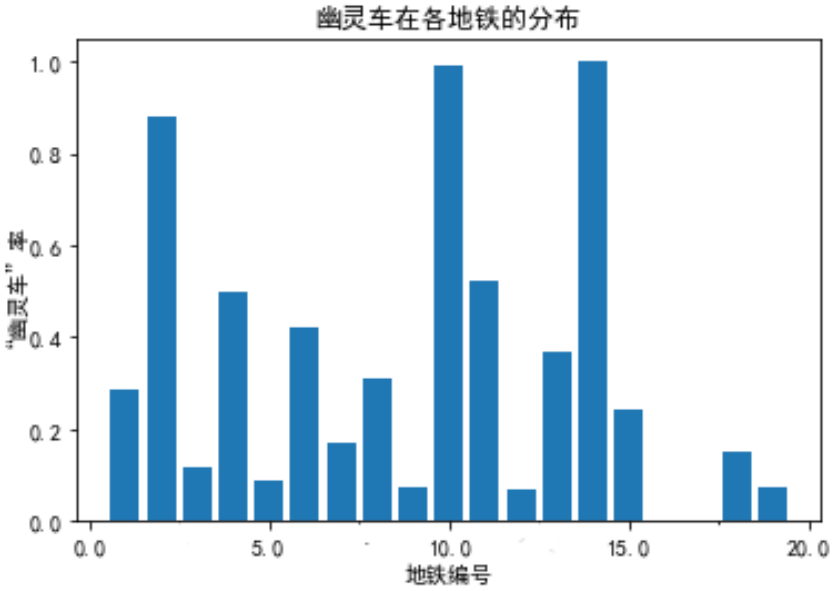


图 9. 幽灵车在各线的分布

我们可以从图 9 中发现以下线路的“幽灵车”率非常之高（超过 50%）：地铁 2、地铁 4、地铁 10、地铁 11、地铁 14。经过线路和数据研究我们将以上超过 50%的地铁分成以下几个部分：

① 地铁 2 和 地铁 10：这两条线均为环线，在设计的时候使用的是分段机制，即需要下车换成其他其他车次的同路线地铁才能完成整条环线[3].

② 地铁 4 、地铁 11、地铁 14：这三条线均有分叉口或者是有部分路线不开启导致附件 2 给出的地铁站数不精确。地铁 4 是两条路线：地铁四号线和大兴线的合体，因为有支路；11 号线是机场线，有支路：可以不经过 2 号航站楼。地铁 14 号线的七里庄部分不开放。

因此寻找“幽灵车”的算法应该更新为：

1. 利用附件 2 统计每一地铁路线的总站数，记录为 $Num_{(i)}$ ， i 指代地铁编码。

2. 利用附件 3 将统一车次编号的地铁放在同一个集合， $T_{(i)}^{(\overline{abcdef})}$ 。其中 $\overline{abcdef}$ 是车次号， i 指代地铁编码。

3. 利用以下判断将“幽灵车”至于同一集合，*G*host*T*rain 。

If $i=4$ ：

If $\text{len}(T_{(i)}^{(\overline{abcdef})}) \neq Num_{(i)}$ and $\text{len}(T_{(i)}^{(\overline{abcdef})}) \neq Num_{(i)}-10$:

*G*host*T*rain.add($T_{(i)}^{(\overline{abcdef})}$)

elif $i=11$:

If $\text{len}(T_{(i)}^{(\overline{abcdef})}) \neq Num_{(i)}$ and $\text{len}(T_{(i)}^{(\overline{abcdef})}) \neq Num_{(i)}-1$:

*G*host*T*rain.add($T_{(i)}^{(\overline{abcdef})}$)

elif $i=14$:

If $\text{len}(T_{(i)}^{(\overline{abcdef})}) \neq Num_{(i)}-12$:

*G*host*T*rain.add($T_{(i)}^{(\overline{abcdef})}$)

elif $i=2$ or 10:

If $\text{len}(T_{(i)}^{(\overline{abcdef})}) \neq Num_{(i)}$:

*G*host*T*rain.add($T_{(i)}^{(\overline{abcdef})}$)

*G*host*T*rain del max(*count*[*len*(*G*host*T*rain$_i$)])

*G*host*T*rain del max(*count*[*len*(*G*host*T*rain$_i$)])

*G*host*T*rain del max(*count*[*len*(*G*host*T*rain$_i$)])

……

*G*host*T*rain del max(*count*[*len*(*G*host*T*rain$_i$)])

else:

If $\text{len}(T_{(i)}^{(\overline{abcdef})}) \neq Num_{(i)}$:

*G*host*T*rain.add($T_{(i)}^{(\overline{abcdef})}$)

经更新算法修改后我们可以得到“幽灵车”的新分布，如图 10.

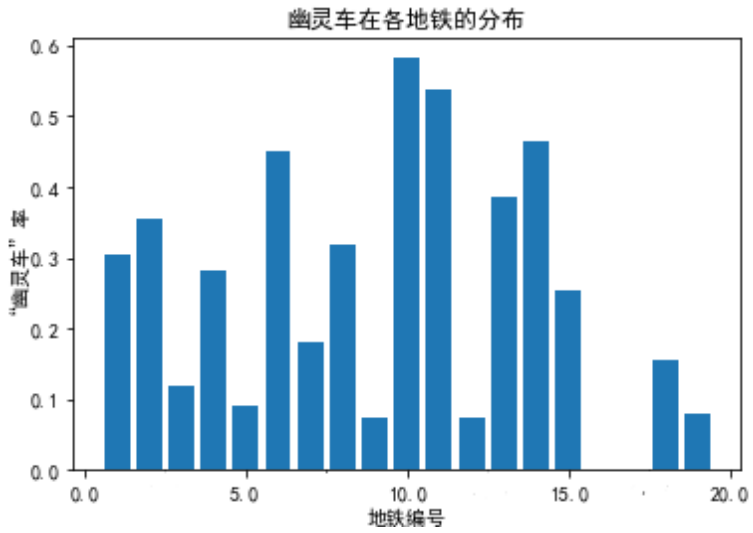


图 10. 幽灵车在各线的分布_改

比之前的一场比例要来得低得多，现在我们统计一下现有“幽灵车”的离站时间分布，如图 11 所示。

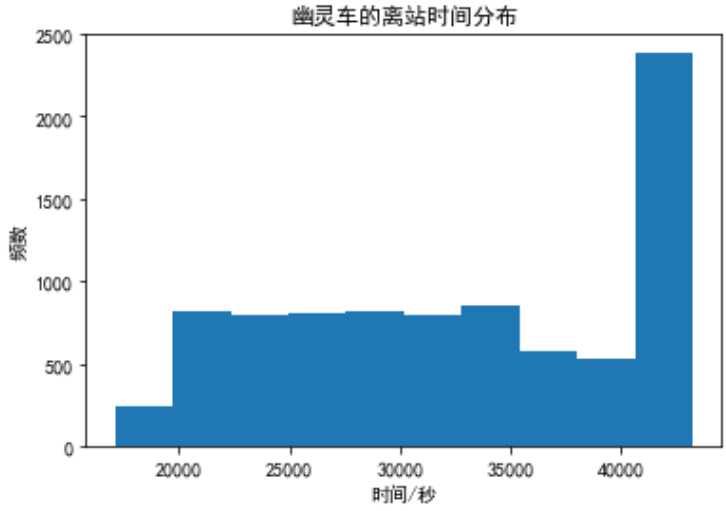


图 11 “幽灵车”的离站时间分布

在这里我们有显然可得的结果：有一半左右的幽灵车集中在 12 点左右，而其他时间则是较为平均的分布。

现在我们以 1 号线的幽灵车来说明我们的第二条推论：‘幽灵车’数据，可以分为两个部分：一部分是地铁首班车的数据，一部分是半夜地铁无人的调度数据。

图 12 显示的是 1 号线“幽灵车”离站时间。

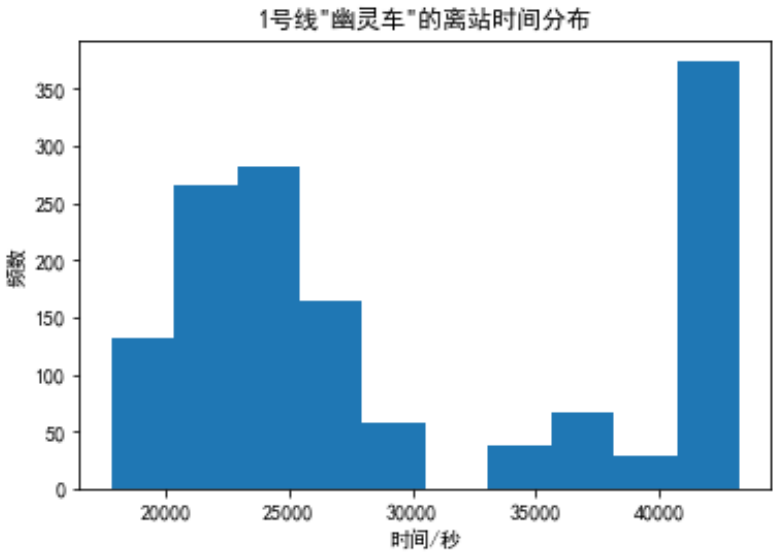


图 12. 1 号线“幽灵车”离站时间

现在就可以清楚得看出 1 号线的“幽灵车”被很明显得分为两个部分，分别是早上 5-7 点和 12 点左右。现在我们结合图 7、图 8 和下面的图 13 来证明这两个部分分别是地铁首班车数据和半夜地铁无人的调度数据。图 13 显示的是地铁 1 号线的早班和末班时间表。

1号线首末车时刻表

车站名称	往四惠东方向		往苹果园方向	
	首车时间	末车时间	首车时间	末车时间
苹果园	5:10	23:30	——	——
古城	4:58	23:33	5:36	0:23
八角游乐园	5:01	23:36	5:32	0:20
八宝山	5:04	23:39	5:29	0:17
玉泉路	5:06	23:42	5:26	0:14
五棵松	5:09	23:45	5:23	0:11
万寿路	5:12	23:48	5:21	0:08
公主坟	5:14	23:50	5:18	0:06
军事博物馆	5:17	23:53	5:16	0:04
木樨地	5:19	23:55	5:13	0:01
南礼士路	5:21	23:57	5:11	23:59
复兴门	5:15	0:00	5:10	23:57
西单	5:17	0:02	5:17	23:54
天安门西	5:19	0:04	5:15	23:52
天安门东	5:21	0:06	5:13	23:50
王府井	5:23	0:08	5:11	23:48
东单	5:25	0:11	5:09	23:46
建国门	5:28	0:13	5:07	23:44
永安里	5:30	0:16	5:04	23:41
国贸	5:32	0:18	5:02	23:39
大望路	5:35	0:21	5:00	23:36
四惠	5:38	0:23	4:57	23:33
四惠东	——	——	5:05	23:30

图 13[4] 地铁 1 号线的早班和末班时间表

我们可以很清楚地看到有一些站的早班时间是一样的：往四惠东方向是“军事博物馆”和“西单”等；往苹果园方向是“王府井”和“南礼士路”等。同一车次地铁不可能在同一时间出现在两个地点，说明早班车是在多个地点同时有多辆车发车。而对应图 7、图 8 的早上“幽灵车”，晚上“幽灵车”恰好可以发现，早上的“幽灵车”的出发点恰好是早班时间出发相同的站，而晚上“幽灵车”的终点有恰好是那些站点。这说明了这些幽灵车主要是可以分为两个部分：一部分是地铁首班车的数据，一部分是半夜地铁无人的调度数据。由此第二个结论被论证了。

现在证明第三个结论，也即附件三中的北京地铁运行图的数据是从前一天晚上的调度运行图到次日中午的运行图数据：即缺失了北京市下午（12-23:50 左右）的地铁的运行图数据。在证明之前我们需要先把幽灵车中半夜调度的部分去掉，避免影响。用以下算法可以解决：

```
For j in GhostTrain:
    If Last_Time(j) > Last_Time_i:
        GhostTrain del j
```

其中 $Last\_Time(j)$ 表示该车次的最早离站时间，$Last\_Time_i$ 表示地铁 i 的末班时间，可在【4】获得。算法即排除在地铁 i 停运后仍然运行的车次。

再处理完后，我们来证明结论三，即数据集是从前一天晚上的调度运行图到次日中午的运行图数据。利用反证法来证明。

假设数据集不是如我们假设的，那么根据图 1 我们可以得到：早上 11-12 点和晚上 11-12 点，某个站点的列车时间间隔是不一样。早上 11-12 点的间隔应该比晚上 11-12 点的距离要更加密集一点，因为早上 11-12 点恰是高峰期，而晚上则是结束期间。则其列车时间间隔分布应该是一个双峰分布。

图 14 显示了地铁 1 号线一非换乘站的 11-12 点之间的列车时间间隔，我们可以很明显的发现这是一个单峰分布，和我们的推论矛盾。因此结论三获得证明。

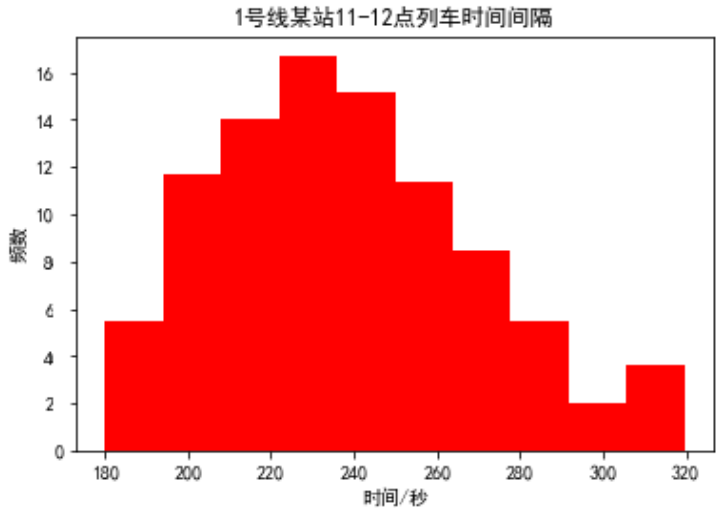


图 14. 地铁 1 号线一非换乘站的 11-12 点之间的列车时间间隔

5.1.4 总结

在这一部分，我们分别对附件 1、附件 2、附件 3 的数据进行了探索性数据分析。在附件 1 的探索性分析中，我们发现了数据集的稀疏性、数据集的时间段以及地铁拥挤滞留的现象，总结了人们出行的基本规律。在附件 2 的探索性数据分析中我们建立了北京地铁路网的无向图模型，并得到了一些与地铁线的信息，例如换乘站等。在附件 3 的探索性数据分析中我们建立了对于附件 3 的数据集的 3 个主要论证，并作出证明。

5.2 问题二的模型建立与求解

5.2.1 地铁系统分析

在开始第二问的研究之前，我们先对已有的地铁系统进行一个彻底的地分析：对于一个地铁系统，我们要保持对其动态的掌握必须有两项最为基础的数据：地铁的无向图与列车时刻表。对于地铁站之间的关系生成的无向图我们已经由附件二转化而得。但是对于列车时刻表我们仍然需要进行补充：根据我们对于附件 3 的探索性数据分析，我们清楚而明白地知道我们缺失了一半的数据（12-24 时）的数据，现在我们不需补充我们的数据。我们选用的方法是贝叶斯分类器。

5.2.2 贝叶斯分类器与拉普拉斯平滑

朴素贝叶斯分类器[5]是一种简单的分类算法。其基本的思想思想基础是遮掩的：对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪一个最大那就将其分属于哪一类。其算法过程如下：

1.设 $x=\{a_1,a_2,\cdots a_m\}$ 是一个待分类的项， a_i 是 x 的一个特征属性。

2.有类的集合 $C=\{\mathrm{y}_1,\mathrm{y}_2,\cdots \mathrm{y}_n\}$ 。

3.计算 $P(y_1\mid x)$ 、 $P(y_2\mid x)$ ····· $P(y_n\mid x)$

4.若 $P(y_k\mid x)=\max\{P(y_1\mid x),P(y_2\mid x)\cdots P(y_n\mid x)\}$ ，则 $x\in y_k$ 。

其中，第三步可由贝叶斯公式计算为：

$$P(y_i\mid x)=\frac{P(x\mid y_i)\cdot P(y_i)}{P(x)}$$

若 a_i 之间是相互条件独立的，则计算可以简化为：

$$P(y_i\mid x)=\frac{P(x\mid y_i)\cdot P(y_i)}{P(x)}=\frac{\prod_{i=1}^{m}P(a_i\mid y_i)\cdot P(y_i)}{P(x)}.$$

当某个特征是连续值时，我们可以使用概率密度代替概率。

当某个特征是离散时，我们使用正常的概率公式进行计算；但是这里我们使用拉普拉斯平滑处理某个分类从未出现过的情况。使用的公式如下：

$$P(a_i\mid y_i)=\frac{\sum_{i=1}^{n}I(a_i,y_i)+1}{\sum_{i=1}^{n}I(y_i)+m}$$

$$P(y_i)=\frac{\sum_{i=1}^{n}I(y_i)+1}{\mathrm{n}+m}$$

5.2.3 特征的选取与地铁时间表的补全

为了开始使用朴素贝叶斯分类器，我们必须把我们整个地铁系统抽象为一个分类问题。我们的讨论基础就基于如下依据：一个车站的人数和这个车站的车次是有关系的。这是显然易见的：以某个车站进站人数递增肯定是将要来地铁的车次了。两者之间是有关的，因此我们将地铁时间表补全问题解释为一个分类问题。

对于我们已有的数据，我们拥有比较完整的对于人群的抽样，但是对于地铁的时间表的数据是不全的。于是我们站点的人群数据去推断站点的地铁状态。

我们首先考虑某个站点i，我们考虑某个时刻j（单位：秒）：该站点有的人流特征是：在此刻有m 人进入i 站，有n 人离开i 站；该站点拥有的地铁状态是：在该时刻离下一班地铁来到还有t 秒，该车次是属于地铁k ，其为正向（1）或者反向（0）行驶的地铁等。在这里我们规定如果车站编码依次增大则为正向，反之则为方向。以地铁1号线为例，从苹果园向四惠东的地铁则为正向，从四惠东到苹果园的地铁则为反向。

于是整个问题变成:

$$(m,n)\to(t,k,0/1)$$

在这里我们视m 、n 两个特征均为连续变量，方便计算。显然进站人数m和出站人数n两者必是相互独立的，符合朴素贝叶斯分类器的基本假设。我们均假设其满足正态分布，即：

$$m\sim\frac{1}{\sqrt{2\pi}\sigma_m}e^{-\frac{(m-\mu_m)^2}{2\sigma_m^2}}$$

$$n\sim\frac{1}{\sqrt{2\pi}\sigma_n}e^{-\frac{(n-\mu_n)^2}{2\sigma_n^2}}$$

对于$(t,k,0/1)$的标签，k 和 0/1 显然是分类数据。对于t，理论上并不是一个离散值，但是我们为了适应贝叶斯分类器的理论将离散化：$t\in[\min(t_j),\max(t_j)]$，$t\in Z^*$。

将数据集分为训练集（80%）与测试集（20%），得到整体正确率如下表：

表 1. 朴素贝叶斯分类器训练结果

数据集	训练集	测试集
正确率	95.43%	83.1%

在这里我们记我们训练好的第j 个站点的贝叶斯分类型为$Model^{(j)}_{Bay_Sta}$ 。

5.2.4 问题二分析与算法设计

在这里我们以乘客为理性人的假设为前提，那么对于基于我们在对 **5.1.1 附件二的探索性数据分析** 中的结论之一：乘客出行时长分布和乘客出行距离分布具有相似性，因此乘客最后的出行路径一定是在物理上的最短路，因此不管是我们需要一个算法追踪

指定乘客的路径还是我们设计一个算法来辅助乘客的路径选择，我们选定的路径一定是最短路，或者是相对靠近最短路的路径（可能由于拥挤导致乘客选择支线），因此我们设计这样的一个算法来进行对这个问题进行研究：

我们的算法分成以下几步：

1. 现将 **5.1.2 附件二的探索性数据分析** 中生成的无向图赋予权重，两个站点间的权重是由两点的直线距离来决定。
2. 对于给定的一个出发点和一个终点，我们先使用 Dijkstra 算法计算最短路，并且将该最短路的距离记为 $L_{\min}$
3. 利用 BSF 算法开始进行图的遍历，每找到一条路就计算其长度 L_i ，若 $L_i \le 110\% \cdot L_{\min}$ ，则将该路添加进候选路径集 *Paths* 。当找到第一条路 $L_i > 110\% \cdot L_{\min}$ 时即停止遍历整个图。
4. 在候选路径集 *Paths* 中，利用列车时间表计算录成所需时间。
5. 最后一步对于两种不同的情况：若是追踪指定乘客的路径，则是选择与附件 1 指定乘客所花费时间最相近的候选路径集 *Paths* 中的路径，则该路径是指定乘客的路径；若是辅助乘客的路径选择则向乘客推荐候选路径集 *Paths* 中的用时最短的路径。

接下来的两节主要描述 Dijkstra 算法和 BSF 算法 与 如何利用列车时间表计算某指定路径所需时间的时间搜索算法。

5.2.5 Dijkstra 算法和 BSF 算法

Dijkstra 算法和 BSF 算法是关于图论的两个经典算法。下面利用图 15 来讨论这两个算法。

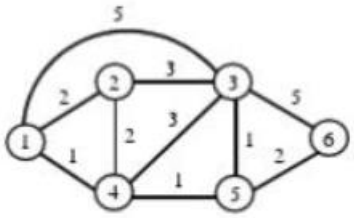


图 15．某图

Dijkstra 算法[7]是有权无向图中寻找最短路的方法，其表现为寻找源节点到网络中其他各节点的最短路径。为了方便期间，我们设源节点为节点 1，然后一步一步寻找直

到找到源节点到左右接待你的最短路位置。

令 $D(v)$ 为源节点（记为节点 1）带某节点 v 的距离，就是从节点内 1 沿着某一路径到节点 v 的所有链路的长度之和。再令 $l(i,j)$ 为节点 i 到节点 j 支架你的距离。整个算法只有
以下两个部分。

（1） 初始化

令 N 表示网络节点的集合。先令 $N=\{1\}$。对所有不在 N 的节点 v ，写出

$$D(v)=\begin{cases} l(1,v) & 1,v\text{直接连接} \\ \infty & 1,v\text{不直接连接} \end{cases}$$

（2）寻找一个不在 N 中的节点 w ，其 $D(w)$ 最小，把节点 w 加入 N 。然后令所有不在 N 的节点 v，令：

$$D(v)=\min\{D(v),D(w)+l(w,v)\}$$

（2） 重复第二步知道所有节点都在 N 中。

BSF 算法[6]，又被成为广度优先算法，是一种图遍历算法，与 DSF 算法相对立。BSF 算法强调以宽度有限进行搜索，即指定一个源节点，率先搜索完距源节点深度相同的节点然后再进行更深层次的节点搜索。以图 15 为例，节点一为源节点，利用 BSF 搜索全图的过程是：1-2-3-4-5-6。

由于 BSF 算法强调广度，因此每一条找到的路，其长度随着算法的进行而慢慢变长，因此我们在设计上一个算法时可以一看见超过最短路长度 10%直接终止算法，节约计算资源。

5.2.6 时间搜索算法

为了技术在候选路径集 *Paths* 中每条路的所用的时间，我们创建了一种新的数据结构来表示地铁运行时刻表，并使用了一种新型时间搜索算法。

新的数据结构类似图 16 所示。

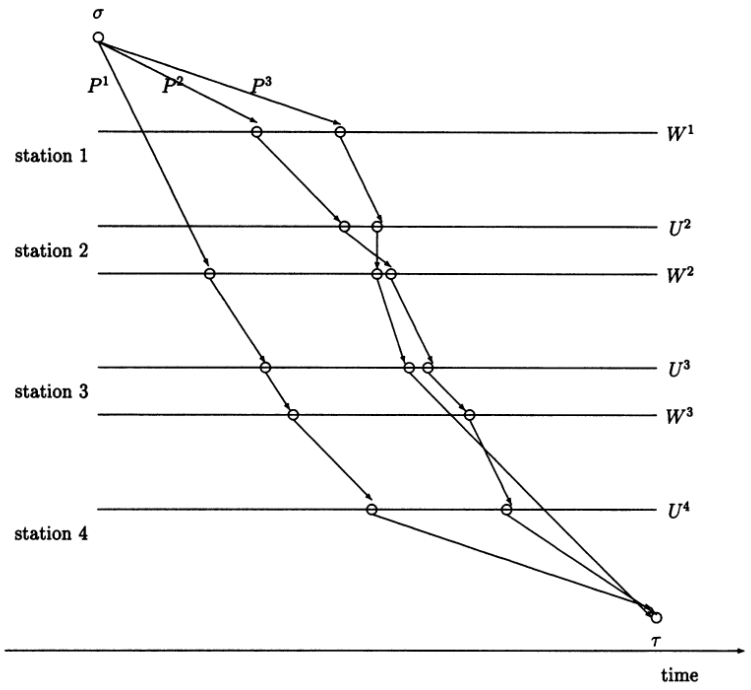


图 16. 新的数据结构的图示

具体用来储存的数据的是一个高维矩阵（86400*277*4），我们现在记这个矩阵为TT，这是一个类链表结构。

$$TT=\begin{bmatrix} [[[t_{now},Line,NextSatation,t_{NextStation}]_1\cdots[\ ]_{277}]_1 \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ [[[[t_{now},Line,NextSatation,t_{NextStation}]_1\cdots[\ ]_{277}]_1]_{86400} \end{bmatrix}$$

TT 矩阵的第一个层是每天时间的离散化（按秒记录，共 86400 秒一天），第二层是北京地铁站的总站点数（277 站），第三层是在某时刻某个站点的状态（4 个参数）：t_{now}，是现在的时刻，$Line$ 是现在在该站点的停留地铁的地铁编号，$NextSatation$ 是该车次的下一站，$t_{NextStation}$ 是下一站的到达时间。如果该站点此刻没有地铁，则值均为零。这个矩阵主要是由附件 2 和 $Model^{(j)}{}_{Bay\_Sta}$ 生成的时刻表构成的。

利用这个矩阵我们就可以任意计算某个人在某一时间点进入某一个车站，要根据某个给定路线到达某个斩断所需的时间，用以下时间搜索算法可以实现。

我们假设某乘客在a 时刻进入b 站，要前往c 站所需的时间可以由以下算法：

1. 检查TT矩阵的$TT[a,b,1]$是否为 0。若不为零，则到下一站的时间为$TT[a,b,3]-b$。若为 0，则进入循环: 每次都对b递增 1 到b'，直到$TT[a,b',1]$不为零，则记录下到一站的时间$TT[a,b,3]-b$。
2. 接着可以不断进行迭代，我们可以根据$NextSatation, t_{NextStation}$的数据检索到下一站中记录的下一站的信息，直到到达$c$ 站。

这个算法借助的是链表的思维，整个检索的过程可以由图 17 表示。

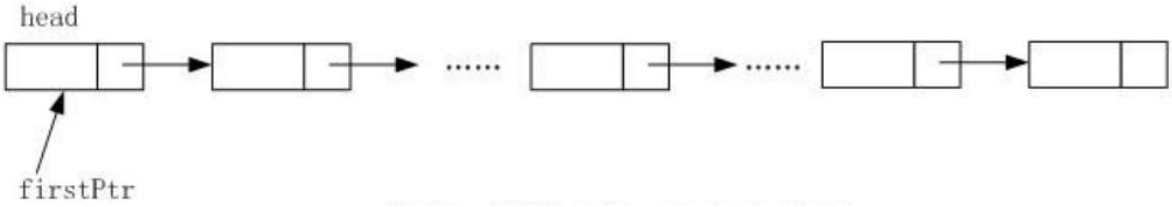


图 17. 时间搜索算法

5.2.7 问题二的模型求解

有了以上的新型数据结构和算法，我们可以使用其对问题进行求解。获得的乘客形成路线如表所示。

表 2. 乘客选择的路线

乘客	选择的路线
1	['高米店北', '西红门', '新宫', '公益西桥', '角门西', '马家堡', '北京南站', '陶然亭', '菜市口', '宣武门', '和平门', '前门', '崇文门', '东单', '建国门']
7	['霍营', '立水桥', '北苑', '望京西', '芍药居', '太阳宫', '三元桥', '东直门']
19	['柳芳', '东直门', '东四十条', '朝阳门', '建国门', '北京站', '崇文门']
31	['天通苑南', '立水桥', '霍营', '回龙观', '龙泽', '西二旗', '上地', '五道口', '知春路', '大钟寺', '西直门', '车公庄']
41	['古城', '八角游乐园', '八宝山', '玉泉路', '五棵松', '万寿路', '公主坟', '军事博物馆', '白堆子', '白石桥南', '车公庄西', '车公庄', '平安里', '北海北', '南锣鼓巷', '什刹海', '鼓楼大街', '安定门']
71	['天通苑', '天通苑南', '立水桥', '北苑', '望京西', '芍药居', '太阳宫', '三元桥', '东

	直门', '东四十条', '朝阳门', '建国门']
83	['立水桥', '北苑', '望京西', '芍药居', '太阳宫', '三元桥', '东直门', '东四十条', '朝阳门', '建国门', '北京站', '崇文门']
89	['青年路', '十里堡', '金台路', '呼家楼', '东大桥', '朝阳门', '东四', '南锣鼓巷', '北海北', '平安里', '车公庄', '阜成门']
101	['大井', '七里庄', '六里桥', '六里桥东', '北京西站', '军事博物馆', '木樨地', '南礼士路', '复兴门', '西单', '天安门西', '天安门东', '王府井', '东单', '建国门']
113	['北京西站', '湾子', '达官营', '广安门内', '菜市口', '宣武门', '和平门', '前门', '崇文门', '东单', '建国门']
2845	['天通苑北', '天通苑', '天通苑南', '立水桥', '北苑', '望京西', '芍药居', '太阳宫', '三元桥', '东直门']
124801	['北宫门', '西苑', '圆明园', '北京大学东门', '中关村', '海淀黄庄', '知春里', '知春路', '西土城', '牡丹园', '健德门', '北土城', '安贞门', '惠新西街南口', '芍药居', '望京西', '望京']
140610	['宋家庄', '刘家窑', '蒲黄榆', '天坛东门', '磁器口', '崇文门', '北京站', '建国门', '朝阳门', '东四十条', '东直门', '三元桥', '太阳宫', '芍药居', '望京西', '望京']
164834	['复兴门', '阜成门', '车公庄', '西直门', '大钟寺', '知春路', '西土城', '牡丹园', '健德门', '北土城', '安贞门', '惠新西街南口', '惠新西街北口', '大屯路东', '关庄', '俸伯']
193196	['临河里', '梨园', '九棵树', '果园', '通州北苑', '八里桥', '管庄', '双桥', '传媒大学', '高碑店', '四惠东', '四惠', '大望路', '国贸', '永安里', '建国门', '朝阳门', '东四十条', '东直门', '雍和宫', '和平里北街', '和平西桥', '惠新西街南口', '惠新西街北口', '大屯路东', '关庄', '俸伯', '顺义', '石门', '南法信']
223919	['角门西', '马家堡', '北京南站', '陶然亭', '菜市口', '宣武门', '西单', '复兴门', '阜成门', '车公庄', '西直门']
275403	['南锣鼓巷', '北海北', '平安里', '车公庄', '西直门', '大钟寺', '知春路', '五道口', '上地']
286898	['枣园', '高米店南', '高米店北', '西红门', '新宫', '公益西桥', '角门西', '马家堡', '北京南站', '陶然亭', '菜市口', '宣武门', '长椿街', '复兴门', '阜成门', '车公庄', '西直门', '大钟寺', '知春路', '五道口', '上地', '西二旗', '龙泽', '回龙观']

314976	['八宝山', '玉泉路', '五棵松', '万寿路', '公主坟', '军事博物馆', '白堆子', '白石桥南', '车公庄西', '车公庄', '西直门', '大钟寺', '知春路', '五道口', '上地']
315621	['国贸', '永安里', '建国门', '朝阳门', '东四十条', '东直门', '柳芳']

至于智能算法协助乘客选择路径的设计已经在 **5.2.4 问题二的分析和算法设计** 中阐述清楚，这里不再赘述。

5.3 问题三的模型建立与求解

5.3.1 问题三的分析与假设扩充

问题三要我们从管理者的角度出发，建立单交通限流模型，这显示是一个目标规划问题，我们这里先对问题的假设进行更加完整而仔细的扩充。

为了简化模型构建要素，对城市轨道交通单线协同限流策略优化模型进行如下基本假定：

（1）在总限流时段内，车站进站口及站台的客流，都符合均匀到达规律，视为均匀分布；

（2)进站口到达乘客仅有被限流和进入站台两种状态,不存在放弃出行需求的行为；

（3）简化乘客在车站付费区的走行过程。不考虑进出站通道对乘客的影响，模型主要从线路层面考虑协同限流策略，因此忽略站内基础设施的影响，即经过进站口的乘客，在限流过后，可直接到达车站站台，简化乘客站内走行时间；

（4）在实际运营中，出站客流能快速离开车站，模型忽略其对站台及进站口客流的影响；

（5）进站乘客进入站台后，仅可通过上车接受运输服务离开其出发车站，不会再次返回其出发车站进站口离站;

（6）列车发车间隔时间固定，且严格按照计划时刻表运行。列车为乘客实现在线路上位移的载体，其运行时间对客流的状态产生了极大的影响。在实际运营中，在不发生故障的情况下，城市轨道交通列车准点率极高，可假定为按照计划时刻表运行；

（7）乘客不存在主观留乘行为。当列车能力未达饱和，乘客可继续上车时，站台乘客不会选择等候下一趟列车；

（8）研究单向的列车运行过程。由于城市轨道交通常态化限流时段的客流潮汐特征明显，因此本文仅研究单一方向上的协同限流策略，模型输入参数如未进行特殊说明，均

为处理后的单方向输入参数。

5.3.2 模型的建立

决策变量设置为车站限流人数$q_s^l(t)$，$q_s^l(t)$表示的是在 t 时段内，车站进站口聚集人数中，因受到限流影响而未能进入车站站台的人数。城市轨道交通运营方迫于站台能力和列车能力的限制，当线路客流量较大时，分别在各站进站口采取一定措施，限制部分乘客的进站，从而对站内候车人数进行一定规模的控制，使整条线路的运营达到最优状态。

所建模型[8]的目标函数为最小化乘客总出行延误时间（T_{delay}），乘客总出行延误时间应力在总限流时段内全线各站乘客出行延误时间的总和，为一维数值形式。总限流时段内各站的乘客出行延误时间包括乘客因限流导致总延误时间（T_{ql}）和因留乘导致总延误时间（T_{qr}），具体计算公式如式 3-1、3-2、3-3 所示。由于在实际运营中，乘客对于因限流和留乘所造成的出行延误较为敏感，当乘客在进站口因限流延误时间或者在站台因留乘延误时间过长时，会极大的影响城市轨道交通车站运营安全，降低乘客出行满意程度，因此，对乘客因限流延误时间和乘客因留乘延误时间设定延误时间惩罚系数θ_1，θ_2。

$$T_{ql}=\sum_{t=1}^{TE}\sum_{s=1}^{S-1}[q_s^l(t)\cdot\frac{\Delta t}{2}] \tag{3-1}$$

$$T_{qr}=\sum_{t=1}^{TE}\sum_{s=1}^{S-1}[q_s^d(t)\cdot f] \tag{3-2}$$

$$T_{delay}=[\theta_1 T_{ql}+\theta_2 T_{qr}]=\sum_{t=1}^{TE}\sum_{s=1}^{S-1}\theta_1[q_s^l(t)\cdot\frac{\Delta t}{2}]+\sum_{t=1}^{TE}\sum_{s=1}^{S-1}\theta_2[q_s^d(t)\cdot f] \tag{3-3}$$

模型所制定单线协同限流策略其目的是通过控制线路各站的限流人数，改变乘客到达站台的时间。在车站层面，随着限流人数的增多，乘客因限流导致的延误时间增多，但进站乘客和站台滞留乘客减少，乘客因留乘延误时间相应的减少；在线路层面，随着上游车站限流人数的增多，乘客因限流延误时间增多，但预留的列车剩余能力增多，使下游车站输送乘客人数增多，相应限流和留乘人数减少。因此，可通过数学方法，制定使全线乘客总出行延误时间最小化的单线协同限流策略。

单线协同限流策略优化模型主要分析车站初始客流输入和客流在车站与列车间的转移过程两个部分，因此约束条件主要围绕以下四个方面进行描述，分别为进站口乘客出行行为约束、列车位置信息约束、站台乘客上下车行为约束和单线多站服务系统能力约束。

车站进站口往往是城市轨道交通运营方实施限流措施的主要地点之一。车站进站口是乘客到达车站后的首要聚集地点，是本文研究单线协同限流策略的开端，在客流需求大的高峰时段，乘客大量聚集在车站进站口，等候进入车站站台乘车，为避免站台乘客的过度拥挤，有必要对进站口进站人数采取一定的限流措施。因此，车站内乘客人数是由乘客到达规律以及制定的限流策略共同决定的，具体进站口乘客出行行为约束如式 3-4、3-5、3-6 所示：

$$q_s^o(t) = R_s^{ar}(t) \cdot \Delta t \quad (3-4)$$

$$q_s^b(t) = q_s^o(t) + q_s^l(t-1) \quad (3-5)$$

$$q_s^e(t) = q_s^h(t) - q_s^l(t) \quad (3-6)$$

式 3-4 为在 t 时段内，进站口到达的客流人数约束，表示乘客已经到达车站进站口等待，为乘客出行需求输入；式 3-5 为在 t 时段内车站进站口聚集人数约束，用 t 时段内新到达的乘客人数与 t-1 时段内的限流人数之和表示；式 3-6 为能够进入车站站台的乘客人数约束，用进站口在 t 时段内的聚集人数与当前限流人数的差值表示，为车站进站口与车站站台的链接环节。

在模型限流时段不断推进的过程中，列车在线路上运行，不能保证恰好在一个限流时段内有列车进入车站，触发乘客上下车行为。本模型设定列车位置信息约束，使用 0-1 变量 $L_{s,r}(t)$ 表示在 t 限流时段开始时，列车 r 是否处于车站 s 的状态信息，具体表达式见式 3-7 所示。当 $L_{s,r}(t)$ 变量值为 0 时，表示此时列车 r 未处于车站 s；当 $L_{s,r}(t)$ 变量值为 1 时，表示此时列车 r 正处于车站车站 s 站台乘客可触发乘客上下车行为。

$$L_{s,r}(t) = \begin{cases} 0(\text{列车r未处于车站s内}) \\ 1(\text{列车r处于车站s内}) \end{cases} \quad (3-7)$$

乘客在站台的出行行为包括列车内乘客下车进入站台和站台乘客进入列车两个过

程。当在限流时段内，有列车到达车站时，乘客开始上下车行为。首先，根据出行 OD 矩阵计算在此站下车的客流量，完成列车内乘客下车过程；接着，在站台乘客上车行为方面，若此时站台候车人数不超过下车后列车剩余载客能力时，站台乘客全部上车；否则将发生乘客留乘现象，超过列车剩余载客能力的乘客将继续在站台等候下一次列车进站。有关乘客上下车行为约束的具体表达式，见式 3-8 至式 3-13 所示。

$$q_s^p(t)=\begin{cases} q_s^p(t-1)+q_s^e(t), \sum_{r=1}^{R} L_{s,r}(t)=0 \\ q_s^d(t), \sum_{r=1}^{R} L_{s,r}(t)=1 \end{cases} \tag{3-8}$$

$$q_s^a(t)=\sum_{r=`}^{R}\sum_{t=1}^{s-1} q_s^b(t)\cdot P_{\mathrm{i},s}\cdot L_{s,r}(t) \tag{3-9}$$

$$q_s^b(t)=\begin{cases} \sum_{r=1}^{R} L_{s,r}(t)\cdot[q_s^p(t-1)+q_s^e(t)], q_s^p(t-1)+q_s^e(t)\le q_r^{av}(t-1)+q_s^a(t) \\ \sum_{r=1}^{R} L_{s,r}(t)\cdot[q_r^{av}(t-1)+q_s^a(t)], q_s^p(t-1)+q_s^e(t)\ge q_r^{av}(t-1)+q_s^a(t) \end{cases} \tag{3-10}$$

$$q_s^d(t)=\sum_{r=1}^{R} L_{s,r}(t)\cdot[q_s^p(t-1)+q_s^e(t)-q_s^b(t)] \tag{3-11}$$

$$q_r^m(t)=q_r^m(t-1)-q_s^a(t)+q_s^b(t) \tag{3-12}$$

$$q_r^{av}(t)=\eta^{\max}\cdot C-q_r^m(t) \tag{3-13}$$

式 3-8 是对 t 时段结束后站台人数的计算，分为是否有列车进站两种情况分别计算。式 3-9 是对乘客下车过程的描述，根据之前车站上车人数及乘客出行 OD 概率矩阵计算求得。式 3-10 是对乘客上车过程的描述，对站台候车人数与列车载客能力的关系分情况处理，上车人数取二者中的较小值。式 3-11 是对乘客留乘现象的描述，是对 t 时段站台候车人数与上车人数的关系进行分析，当站台人数全部上车时，$q_s^p(t-1)+q_s^e(t)=q_s^b(t)$，可知 $q_s^d(t)$=0；当站台人数未能全部上车，出现留乘现象时，$q_s^p(t-1)+q_s^e(t)>q_s^b(t)$，则 $q_s^d(t)$>0。在式 3-9、3-10 和 3-11 中，当任何列车均不在车站 s 内，有 $\sum_{r=1}^{R} L_{s,r}$=0，则下车人数、上车人数和留乘人数均为 0。式 3-12 是对（时段结束后列荤载客人数的计算，当列车到达车站时，乘客进行上下车过程，列车载客人数发生变化；反之，当列车未到达车站时，上下车人数均为 0，则有 $q_r^m(t)=q_r^m(t-1)$。式 3-13 是对 t 时段结束后列车剩余载客能力的计算，为列车最大载客人数与列车当前载客人数的差值。

在目前已建成的城市轨道交通线路中，车站以及列车等设备的能力有其自身的限制，城市轨道交通客流控制必须要求在不超过其能力限制的条件下进行控制管理，因此需加入约束条件对单线多站服务系统能力进行约束，具体表达式见式 3-14 至式 3-17 所示。

$$q_r^m(t) \le C * \eta^{\max} \quad (3\text{-}14)$$

$$q_s^p(t) \le Q_s^p \quad (3\text{-}15)$$

$$q_s^l(t) \le q_s^o(t) + q_s^l(t-1) \quad (3\text{-}16)$$

$$q_s^b(t) \le q_s^p(t) \quad (3\text{-}17)$$

式 3-14 表示列车能力约束，列车内载客人数不能超过列车最大载客人数；式 3-15 表示站台能力约束，站台候车人数不能超过站台允许的最大聚集人数能力；式 3-16 表示限流人数不能超过进站口聚集人数；式 3-17 表示站台上车人数不能超过站台候车人数。

5.3.3 八通线问题的数据准备

八通线线路呈东西走向，上行方向为四惠站至土桥站，运行时间为 6:00-23:50，下行方向为土桥站至四惠站，运行时间为 5:20-23:10。全线共设 13 个车站，其中四惠站和四惠东站为 1 号线的换乘车站，站台形式为岛式，其他车站均为单线运营，站台形式均为侧式。

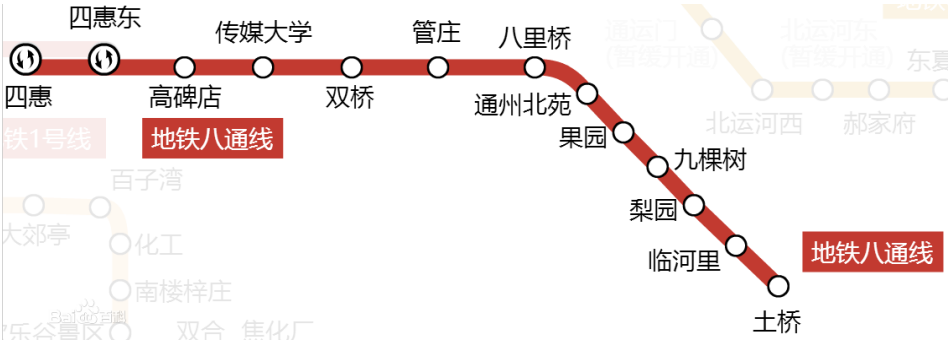


图 18. 北京地铁八通线线路示意图

针对问题三，选取北京地铁八通线下行方向，早高峰 7:00-9:00 时段，构建模型，制定最优化单线协同限流策略，并对制定的限流策略效果进行对比分析，试图得出限流效果较好的车站。

需要用到的基础数据主要有三种类型：线路各站客流数据、列车及线路相关数据、模型及算法相关数据。研究范围为早高峰 7:00-9:00 时段，主要包括各站进站口客流量和乘客出行 OD 概率矩阵。下面是有关列车的一些数据：

1）列车车辆相关数据

每列列车的容量为 1428 人/列，列车座位数为 256 座/列

2）列车发车间隔

八通线早高峰时段（7:00-9:00），下行方向采用 min/列的固定发车间隔模式，本题的列车发车间隔取 $f=30s$

3）列车计划时刻表数据

查阅八通线列车在早高峰时段的各区间运行时分，得出列车在线路运行计划时刻表。由于八通线列车在早高峰时期停站时间极短，且停站时间不影响模型求解结果，所以可以忽略列车在车站的停留时间。

列出首班车到达时刻表，如表 所示，后续列车可依次按照发车间隔时间递推得到。

表 3 八通线 7:00-9:00 时段首班车到达时刻表

车站	首班车到达时刻
土桥站	7:00
临河里站	7:01
梨园站	7:04
九棵树站	7:06
果园站	7:08
通州北苑站	7:11
八里桥站	7:13
管庄站	7:17
双桥站	7:20
传媒大学站	7:23
高碑店站	7:26
四惠东站	7:28
四惠站	7:31

1） 车站站台最大聚集人数能力设置

不考虑上行方向客流对车站站台能力的占用，按照车站站台人均占地最小面积 $0.5m^2$ / 人 和八通线各车车站站台面积，计算得出各车站站台乘客最大聚集人数能力。如表 4 所示

表 4 八通线各车站站台乘客最大聚集人数能力

车站	首站台最大聚集人数（单位：人）
土桥站	894
临河里站	867
梨园站	834
九棵树站	867
果园站	857
通州北苑站	844
八里桥站	936
管庄站	810
双桥站	793
传媒大学站	900
高碑店站	806
四惠东站	1106
四惠站	1029

综上所述是所有所需的参数。

5.3.4 粒子群算法

问题三归根结底是一个目标规划题目，这里我们使用粒子群算法求解 **5.3.2** 中提出的规划及约束问题。

粒子群是模拟群鸟觅食的模型，它在寻找全局最优的应用上具有十分广泛的作用。在群鸟觅食的模型中。我们把每一只鸟儿都看作一个粒子，那么整个鸟群可以被看成一

个粒子群。设在某一个D 维的目标搜索空间中，有m 个粒子所组成的粒子群，其中第i （i=1,2，…，m）个粒子正处于位置$x_i=(x_i^1,x_i^2,\cdots,x_i^D)$ ，这个位置可能正好是一个最优解，将X_i带入目标函数就可以得出其适应值，通过适应值的大小可以判断出这个位置的优劣。假设个体粒子经过的最好位置是$P_i=(p_i^1, p_i^2, \cdots, p_i^D)$ ，整体粒子群所经过最好的位置是$P_g=(p_g^1, p_g^2, \cdots, p_g^D)$。第 i 粒子的速度记为$S_i=(s_i^1, s_i^2, \cdots, s_i^D)$ 。

在粒子群算法运行中，用以下公式对粒子的位置进行不断的更新：

$$s_i^d=\omega s_i^d+c_1r_1(p_i^d-x_i^d)+c_2r_2(p_g^d-x_i^d)$$

$$x_i^d=x_i^d+\alpha s_i^d$$

其中，i=1，2，…，m；d=1,2，…，D；惯性因子$\omega\geq 0$；加速常数c_1、$c_2\geq 0$；r_1、r_2是在$[0,1]$内变化的随机数；约束因子α控制速度的权重；$s_i^d\in\left[-s_{max}^d, s_{max}^d\right]$，表明粒子的速度被最大速度限制，这个最大速度可以人为设定。最后，粒子群位置的改变，也即迭代次数可以被人为设定。

5.3.5 问题三的模型求解

由于粒子群算法的参数设置问题，亦或是数据问题，此处粒子群优化算法一直不能达到全局最优化值，此处有待未来探究。

5.4 限流措施的给出

虽然我们没有计算出问题三的具体模型解，但是根据一些局部最优解和统计数据来看，我们仍然能给出如下限流措施的建议。

制定的最优单线协同限流策略，应保证乘客总在车旅行时间与限流前的一样，达到最大值，使系统无客流的损失，满足乘客总在车旅行时间指标最优的条件。模型所制定的单线协同限流策略，通过调控地铁路线下行方向各车站在早高峰时段的控制力度，对线路能力进行了动态配置，在不造成客流损失的条件下，使得列车运输能力得到充分的利用，实现为线路拥挤车站预留更多列车剩余能力的功能，缓解线路各车站的拥挤，保证线路各车站乘客的安全出行，缩短全线乘客总出行延误时间和乘客总出行等待时间，

达到系统优化的效果。

由问题三的历史解决方案可以得出：在早高峰 7:00-9:00 时段，共有 33 列车到达传媒大学站进行运输服务。在站台乘客上车人数方面，最优单线协同限流策略相比于未协同限流策略影响下，传媒大学车站在前 5 列车中的列车 2、列车 3 的上车人数有了显著增加，获得了较多的列车剩余能力，极大的缓解了传媒大学车站的客流拥挤程度；后续在列车 5~列车 15 运营期间，上车人数差异较小；在列车 15~列车 25 运营期间，八通线各车站客流到达率逐渐降低，传媒大学站上游车站对列车能力的占用降低，使传媒大学站站台乘客上车人数呈现上升趋势；在列车 25 之后，限流措施解除，站台上车人数趋于定值 120 人。在站台乘客留乘人数方面，最优单线协同限流策略较未协同限流策略极大的减少了站台乘客留乘人数，缩短了八通线乘客总体因留乘导致的延误时间。而且在列车 25 到达传媒大学站时，已无留乘乘客，提前结束了八通线线路的乘客留乘现象，使线路提前恢复正常运营。

制定的最优单线协同限流策略，应保证乘客总在车旅行时间与限流前的一样，达到最大值，使系统无客流的损失，满足乘客总在车旅行时间指标最优的条件。模型所制定的单线协同限流策略，通过调控地铁路线下行方向各车站在早高峰时段的控制力度，对线路能力进行了动态配置，在不造成客流损失的条件下，使得列车运输能力得到充分的利用，实现为线路拥挤车站预留更多列车剩余能力的功能，缓解线路各车站的拥挤，保证线路各车站乘客的安全出行，缩短全线乘客总出行延误时间和乘客总出行等待时间，达到系统优化的效果

六、模型的评价与改进

6.1 模型的优点

（1）清楚扎实地从数据中看到存在的问题，并且论证我们自己的结论。

（2）贝叶斯模型简单方便，准确率高。

（3）仿照链表开发的 TT 矩阵十分可靠高效，是一种新的高效的数据结构。

6.2 模型的缺点

（1）第三问模型过于复杂，暂时不能求解出最优解。

（2）第二问模型过于理想化。

参考文献

【1】 Chen L , Zhang D , Wang L , et al. Dynamic cluster-based over-demand prediction in bike sharing systems[C]// Acm International Joint Conference on Pervasive & Ubiquitous Computing. ACM, 2016.

【2】 北京地铁站经纬度信息 - 百度坐标系 - 图文 - 百度文库

【3】 路宗存，王琦．北京地铁 10 号线一期工程车站选型总结与思考[J]．铁道标准设计, 2009 (12): 46-49.

【4】 首末车时间 | 北京地铁官方网站

【5】 Friedman, N., Geiger, D. & Goldszmidt, M. Machine Learning (1997) 29: 131. https://doi.org/10.1023/A:1007465528199

【6】 Dijkstra 算法描述 - 百度文库

【7】 BSF 与 DSF - qianji little boy 的博客 - CSDN 博客

【8】 赵锴．城市轨道交通单线协同限流策略优化研究[D]. 2018.

附录

Code1

```
"""
Created on Fri Apr 12 20:26:08 2019

@author: 30517
"""

import numpy as np


labelpostive = []
labelnegative = []
```

```
for i in range(86400):
    labelpostive.append([])
for i in range(86400):
    for j in range(276):
        labelpostive[i].append([[]])

for i in range(86400):
    labelnegative.append([])
for i in range(86400):
    for j in range(276):
        labelnegative[i].append([[i,'0',0,0]])



train_code_list=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

counter = 0

for i in train_number_list:
    for key in i:
        train_code_list[counter].append(key)
    counter = counter + 1



for i in train_code_list:
    for j in i:
        a = train_number_list[train_code_list.index(i)][j][0][2]
        b = train_number_list[train_code_list.index(i)][j][1][2]
        if int(a) < int(b):
            counter = 0
            for k in train_number_list[train_code_list.index(i)][j]:
```

```
                    if counter != (len(train_number_list[train_code_list.index(i)][j])-1):
                        for p in range(k[3],k[4]+1):
                            #placeholder = []

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(p)        #添加此刻时间

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(k[0])        #添加从属线路

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train
_code_list.index(i)][j][counter+1][2])-1])+1))        #添加下一站地标

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter+1][3])
#添加下一站到达时间

#labelpostive[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1]))] = placeholder
                        counter = counter + 1
                    else:
                        for p in range(k[3],k[4]+1):
                            #placeholder = []

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(p)        #添加此刻时间

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(k[0])        #添加从属线路
```

```
labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])+1))        #添加下一站地标

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter][3])
#添加下一站到达时间

#labelpostive[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1]))] = placeholder
                        counter = counter + 1
            elif int(a) > int(b):
                counter = 0
                for k in train_number_list[train_code_list.index(i)][j]:
                    if counter != (len(train_number_list[train_code_list.index(i)][j])-1):
                        for p in range(k[3],k[4]+1):
                            #placeholder = []

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(p)        #添加此刻时间

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(k[0])        #添加从属线路

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter+1][2])-1])+1))        #添加下一站地标

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter+1][3])
#添加下一站到达时间
```

```
#labelnegative[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(
i)][j][counter][2])-1]))] = placeholder
                    counter = counter + 1
                else:
                    for p in range(k[3],k[4]+1):
                        #placeholder = []

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1])][0].append(p)        #添加此刻时间

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1])][0].append(k[0])        #添加从属线路

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train
_code_list.index(i)][j][counter][2])-1])+1))        #添加下一站地标

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter][3])
#添加下一站到达时间

#labelnegative[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(
i)][j][counter][2])-1]))] = placeholder
                    counter = counter + 1
'''
labelpostive_np = np.array(labelpostive)
labelnegative_np = np.array(labelnegative)
'''
for i in range(86400):
    for j in range(276):
```

```
            labelpostive[i][j].append(labelnegative[i][j][0])


label = np.array(labelpostive)
```

Code2

```
"""
Created on Sat Apr 13 11:54:28 2019

@author: 30517
"""


people_on = []
people_off = []


g = open('1.txt',mode='r',encoding='gbk')
for line in g:
    a = line.replace('\n', '').split(',')
    b = a[3].split(':')
    c = a[4].split(':')
    a[3] = int(b[0])*3600 + int(b[1])*60 + int(b[2])
    a[4] = int(c[0])*3600 + int(c[1])*60 + int(c[2])
    on = [a[1],a[3]]
    off = [a[2],a[4]]
    if ((a[4]-a[3]) > 0) and (0 < int(a[1]) < 329) and (0 < int(a[2]) < 329): #数据清洗，失误数据：站点错误和时间记录错误
        people_on.append(on)
        people_off.append(off)
g.close()
```

```
import numpy as np


feature = []

for i in range(86400):
    feature.append([])
for i in range(86400):
    for j in range(276):
        feature[i].append([[i,0,0,0]])

for i in people_on:
    feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][2] =
feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][2] + 1
for i in people_off:
    feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][3] =
feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][3] + 1




for i in range(86400):
    for j in range(276):
        if i != 0:
            feature[i][j][0][1] = feature[i-1][j][0][1] + feature[i-1][j][0][2] -
feature[i-1][j][0][3]
```

```
feature = np.array(feature)
```

Code3

```
'''
class Graph(object):

    def __init__(self, graph_dict=None):
        """ initializes a graph object
            If no dictionary or None is given,
            an empty dictionary will be used
        """
        if graph_dict == None:
            graph_dict = {}
        self.graph_dict = graph_dict

    def vertices(self):
        """ returns the vertices of a graph """
        return list(self.graph_dict.keys())

    def edges(self):
        """ returns the edges of a graph """
        return self.__generate_edges()

    def add_vertex(self, vertex):
        """ If the vertex "vertex" is not in
            self.__graph_dict, a key "vertex" with an empty
            list as a value is added to the dictionary.
            Otherwise nothing has to be done.
        """
        if vertex not in self.graph_dict:
            self.graph_dict[vertex] = []
```

```
def add_edge(self, edge):
    """ assumes that edge is of type set, tuple or list;
        between two vertices can be multiple edges!
    """
    edge = set(edge)
    (vertex1, vertex2) = tuple(edge)
    if vertex1 in self.graph_dict:
        self.graph_dict[vertex1].append(vertex2)
    else:
        self.graph_dict[vertex1] = [vertex2]

def generate_edges(self):
    """ A static method generating the edges of the
        graph "graph". Edges are represented as sets
        with one (a loop back to the vertex) or two
        vertices
    """
    edges = []
    for vertex in self.graph_dict:
        for neighbour in self.graph_dict[vertex]:
            if {neighbour, vertex} not in edges:
                edges.append({vertex, neighbour})
    return edges

def __str__(self):
    res = "vertices: "
    for k in self.graph_dict:
        res += str(k) + " "
    res += "\nedges: "
    for edge in self.generate_edges():
```

```
            res += str(edge) + " "
        return res


def find_path(self, start_vertex, end_vertex, path=None):
        """ 寻找从一个 v 到另一个 v 的路径，给出一条 """
        if path == None:
            path = []
        graph = self.graph_dict
        path = path + [start_vertex]
        if start_vertex == end_vertex:
            return path
        if start_vertex not in graph:
            return None
        for vertex in graph[start_vertex]:
            if vertex not in path:
                extended_path = self.find_path(vertex,
                                               end_vertex,
                                               path)
                if extended_path:
                    return extended_path
        return None


def find_all_paths(self, start_vertex, end_vertex, path=[]):
        """ 寻找所有从一个 v 到另一个 v 的路径 """
        graph = self.graph_dict
        path = path + [start_vertex]
        if start_vertex == end_vertex:
            return [path]
        if start_vertex not in graph:
```

```
            return []
        paths = []
        for vertex in graph[start_vertex]:
            if vertex not in path:
                extended_paths = self.find_all_paths(vertex,
                                                     end_vertex,
                                                     path)
                for p in extended_paths:
                    paths.append(p)
        return paths
'''

'''获取站点和基本连接图位置'''

from matplotlib import pyplot as plt
import networkx as nx
import matplotlib as mpl
mpl.rcParams['axes.unicode_minus']=False #用来显示负数的
mpl.rcParams['font.family']='SimHei'
mpl.rcParams['font.sans-serif']=['SimHei']
G=nx.Graph()


'''
G.add_nodes_from([1,2,3])
G.add_edges_from([(1,2),(1,3)])
'''
```

```
g = open('2-1.txt',mode='r',encoding='gbk')
station = set()
for line in g:
    b = line.replace('\n', '').split(',')
    station.add(b[1])
g.close()

station = list(station)

station_dict = {}

for sta in station:
    station_dict[sta]=[]

for i in station:
    a = []
    g = open('2-1.txt',mode='r',encoding='gbk')
    for line in g:
        c = line.replace('\n', '').split(',')
        if c[1] == i:
            a.append([c[0],c[2]])
    station_dict[i] = a
    g.close()


'''
计算换乘站
'''
```

```
change_station=[]

for j in station:
    if not(len(station_dict[j]) == 1):
        change_station.append(j)


'''
收集按照排序的站点
'''

station_in_index=[]
g = open('2-1.txt',mode='r',encoding='gbk')
for line in g:
    c = line.replace('\n', '').split(',')
    station_in_index.append(c[1])
g.close()




'''按照交通线分类地铁线'''

transportline=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
f = open('2-1.txt',mode='r',encoding='gbk')
for line in f:

    a = line.replace('\n', '').split(',')
    transportline[eval(a[2])-1].append(a)

f.close()
```

```
for i in station_in_index:
    G.add_node(i)



for i in transportline:
    for j in range(len(i)):
        if j == 0:
            G.add_edge(i[j][1],i[j+1][1])
        elif j == (len(i)-1):
            G.add_edge(i[j][1],i[j-1][1])
        else:
            G.add_edge(i[j][1],i[j+1][1])
            G.add_edge(i[j][1],i[j-1][1])
```

```
print(nx.shortest_path(G, source='国贸', target='柳芳'))
```

Code4

```
""" A Python Class
A simple Python graph class, demonstrating the essential
facts and functionalities of graphs.
"""
```

```
class Graph(object):

    def __init__(self, graph_dict=None):
        """ initializes a graph object
            If no dictionary or None is given,
            an empty dictionary will be used
        """
        if graph_dict == None:
            graph_dict = {}
        self.__graph_dict = graph_dict

    def vertices(self):
        """ returns the vertices of a graph """
        return list(self.__graph_dict.keys())

    def edges(self):
        """ returns the edges of a graph """
        return self.__generate_edges()

    def add_vertex(self, vertex):
        """ If the vertex "vertex" is not in
            self.__graph_dict, a key "vertex" with an empty
            list as a value is added to the dictionary.
            Otherwise nothing has to be done.
        """
        if vertex not in self.__graph_dict:
            self.__graph_dict[vertex] = []

    def add_edge(self, edge):
```

```
        """ assumes that edge is of type set, tuple or list;
            between two vertices can be multiple edges!
        """
        edge = set(edge)
        (vertex1, vertex2) = tuple(edge)
        if vertex1 in self.__graph_dict:
            self.__graph_dict[vertex1].append(vertex2)
        else:
            self.__graph_dict[vertex1] = [vertex2]

    def __generate_edges(self):
        """ A static method generating the edges of the
            graph "graph". Edges are represented as sets
            with one (a loop back to the vertex) or two
            vertices
        """
        edges = []
        for vertex in self.__graph_dict:
            for neighbour in self.__graph_dict[vertex]:
                if {neighbour, vertex} not in edges:
                    edges.append({vertex, neighbour})
        return edges

    def __str__(self):
        res = "vertices: "
        for k in self.__graph_dict:
            res += str(k) + " "
        res += "\nedges: "
        for edge in self.__generate_edges():
            res += str(edge) + " "
        return res
```

```
def find_path(self, start_vertex, end_vertex, path=None):
    """ 寻找从一个 v 到另一个 v 的路径，给出一条 """
    if path == None:
        path = []
    graph = self.__graph_dict
    path = path + [start_vertex]
    if start_vertex == end_vertex:
        return path
    if start_vertex not in graph:
        return None
    for vertex in graph[start_vertex]:
        if vertex not in path:
            extended_path = self.find_path(vertex,
                                           end_vertex,
                                           path)
            if extended_path:
                return extended_path
    return None


def find_all_paths(self, start_vertex, end_vertex, path=[]):
    """ 寻找所有从一个 v 到另一个 v 的路径 """
    graph = self.__graph_dict
    path = path + [start_vertex]
    if start_vertex == end_vertex:
        return [path]
    if start_vertex not in graph:
        return []
    paths = []
```

```
            for vertex in graph[start_vertex]:
                if vertex not in path:
                    extended_paths = self.find_all_paths(vertex,
                                                         end_vertex,
                                                         path)
                    for p in extended_paths:
                        paths.append(p)
            return paths


if __name__ == "__main__":

    g = { "a" : ["d"],
          "b" : ["c"],
          "c" : ["b", "c", "d", "e"],
          "d" : ["a", "c"],
          "e" : ["c"],
          "f" : []
        }


    graph = Graph(g)

    print("Vertices of graph:")
    print(graph.vertices())

    print("Edges of graph:")
    print(graph.edges())

    print("Add vertex:")
    graph.add_vertex("z")
```

```
print("Vertices of graph:")
print(graph.vertices())

print("Add an edge:")
graph.add_edge({"a","z"})

print("Vertices of graph:")
print(graph.vertices())

print("Edges of graph:")
print(graph.edges())

print('Adding an edge {"x","y"} with new vertices:')
graph.add_edge({"x","y"})
print("Vertices of graph:")
print(graph.vertices())
print("Edges of graph:")
print(graph.edges())


print('The path from vertex "a" to vertex "b":')
path = graph.find_path("a", "b")
print(path)

print('The path from vertex "a" to vertex "f":')
path = graph.find_path("a", "f")
print(path)

print('The path from vertex "c" to vertex "c":')
path = graph.find_path("c", "c")
```

```
    print(path)
```

Code5

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 14 00:21:08 2019

@author: 30517
"""
import matplotlib.pyplot as plt
import numpy as np


Exp=[]
Var=[]

X = feature[:,0,0,0]
Y = feature[:,0,0,1]

plt.bar(Y)
plt.show()
```

Code6.

```
'''站点计算'''
'''
g = open('2-1.txt',mode='r',encoding='gbk')
station = set()
```

```
for line in g:
    b = line.replace('\n', '').split(',')
    station.add(b[1])
g.close()
'''
```

station = ['青年路', '西小口', '通州北关', '六道口', '虎坊桥', '北土城', '回龙观', '苏庄', '湾子', '管庄', '新宫', '角门西', '军事博物馆', '三元桥', '慈寿寺', '园博园', '潘家园', '善各庄', '大郊亭', '天通苑北', '广阳城', '和平门', '东风北桥', '北京站', '霍营', '什刹海', '奥林匹克公园', '健德门', '牡丹园', '阜成门', '大红门', '双井', '清源路', '稻田', '纪家庙', '珠市口', '五棵松', '劲松', '北宫门', '小红门', '安定门', '五道口', '望京东', '望京', '通州北苑', '安河桥北', '望京西', '沙河', '西二旗', '车公庄西', '蒲黄榆', '草房', '七里庄', '通运门', '枣园', '潞城', '木樨地', '芍药居', '亮马桥', '北海北', '黄渠', '回龙观东大街', '圆明园', '八角游乐园', '安贞门', '北京南站', '望京南', '北京大学东门', '朝阳门', '国家图书馆', '双桥', '育知路', '亦庄桥', '立水桥', '海淀黄庄', '北新桥', '长春桥', '崔各庄', '郝家府', '惠新西街南口', '荣昌东街', '四惠', '菜市口', '刘家窑', '大钟寺', '同济南路', '西红门', '六里桥东', '西单', '科怡路', '平西府', '梨园', '六里桥', '良乡南关', '巩华城', '安立路', '百子湾', '玉泉路', '生物医药基地', '大瓦窑', '丰台东大街', '垡头', '黄村火车站', '荣京东街', '农业展览馆', '北苑', '中关村', '成寿寺', '上地', '东夏园', '丰台科技园', '苏州街', '白石桥南', '公主坟', '长椿街', '九龙山', '天安门西', '俸伯', '南法信', '海淀五路居', '广渠门外', '化工', '林萃桥', '大屯路东', '永安里', '角门东', '和平西桥', '旧宫', '褡裢坡', '朝阳公园', '大望路', '建国门', '达官营', '泥洼', '大井', '肖村', '苹果园', '东大桥', '首经贸', '育新', '清华东路西口', '枣营', '东湖渠', '积水潭', '顺义', '物资学院路', '亦庄文化园', '丰台站', '石门', '草桥', '巴沟', '黄村西大街', 'T3 航站楼', '郭庄子', '九棵树', '新街口', '朱辛庄', '动物园', '金台路', '南礼士路', '丰台南路', '篱笆房', '桥湾', '后沙峪', '宣武门', '高米店北', '西钓鱼台', '马泉营', '万源街', '宋家庄', '北京西站', '北运河东', '前门', '南邵', '平安里', '立水桥南', '焦化厂', '良乡大学城', '复兴门', '崇文门', '广安门内', '安华桥', '将台', '惠新西街北口', '国展', '北沙滩', '白堆子', '安德里北街', '知春里', '灵境胡同', '陶然亭', '和平里北街', '团结湖', '双合', '龙泽', '魏公村', '天安门东', '良乡大学城西', '广渠门内', '西四', '马家堡', '南楼梓庄', '阜通', '东四', '郭公庄', '义和庄', '经海路', '张郭庄', '欢乐谷景区', '西直门', '车公庄', '四惠东', '次渠', '森林公园南门', '东单', '光熙门', '八宝山', '古城', '大葆台', '莲花桥', '天通苑南', '长阳', '国贸', '花园桥', '临河里', '车道沟', '北运河西', 'T2 航站楼', '鼓楼大街', '人民大学', '八里桥', '孙河', '常营', '西苑', '柳芳', '南锣鼓巷', '良乡大学

城北', '火器营', '万寿路', '西局', '花梨坎', '亦庄火车站', '公益西桥', '十里堡', '天通苑', '天宫院', '太阳宫', '沙河高教园', '王府井', '生命科学园', '灯市口', '果园', '十里河', '磁器口', '来广营', '传媒大学', '天坛东门', '呼家楼', '金台夕照', '知春路', '西土城', '高家园', '张自忠路', '北苑路北', '永泰庄', '石榴庄', '奥体中心', '分钟寺', '关庄', '土桥', '东直门', '高碑店', '雍和宫', '高米店南', '东四十条', '次渠南']

```
station_dict = {}

for sta in station:
    station_dict[sta]=[]

for i in station:
    a = []
    g = open('2-1.txt',mode='r',encoding='gbk')
    for line in g:
        c = line.replace('\n', '').split(',')
        if c[1] == i:
            a.append([c[0],c[2]])
    station_dict[i] = a
    g.close()


'''
计算换乘站
'''


change_station=[]

for j in station:
    if not(len(station_dict[j]) == 1):
```

```
            change_station.append(j)


'''
收集按照排序的站点
'''

station_in_index=[]
g = open('2-1.txt',mode='r',encoding='gbk')
for line in g:
    c = line.replace('\n', '').split(',')
    station_in_index.append(c[1])
g.close()


'''
计算每条路所需要的站点
'''
LineStationNumber=[]

for i in range(0,19):
    counter = 0
    g = open('2-1.txt',mode='r',encoding='gbk')
    for line in g:
        c = line.replace('\n', '').split(',')
        if (int(c[2])-1) == i:
            counter = counter + 1
    LineStationNumber.append(counter)
    g.close()

LineStationNumber[13] = 7        #排除十四号线七里庄问题
```

'''寻找幽灵车'''

```
GhostTrain = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

transportline=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

f = open('3.txt',mode='r',encoding='gbk')
for line in f:
    a = line.replace('\n', '').split(',')
    b = a[3].split(':')
    c = a[4].split(':')
    a[3] = int(b[0])*3600 + int(b[1])*60 + int(b[2])
    a[4] = int(c[0])*3600 + int(c[1])*60 + int(c[2])
    transportline[eval(a[0])-1].append(a)
f.close()


train_number_list = [{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{}]


for ii in range(len(transportline)):

    train_number=set()

    for i in transportline[ii]:
        train_number.add(i[1])
```

```
    train_number=list(train_number)

    for i in train_number:
        train_number_list[ii][i]=[]

    for i in transportline[ii]:
        train_number_list[ii][i[1]].append(i)

    for i in train_number:
        if ii == 3:             #四号线合并问题
            if (len(train_number_list[ii][i]) != LineStationNumber[ii]) and
(len(train_number_list[ii][i]) != 25):
                GhostTrain[ii].append(train_number_list[ii][i])


        else:
            if len(train_number_list[ii][i]) != LineStationNumber[ii]:
                GhostTrain[ii].append(train_number_list[ii][i])


length_Ten=[]
StationCountForTen={}
for i in GhostTrain[9]:
    length_Ten.append(len(i))
```

```
for i in length_Ten:
    StationCountForTen[i]=0


for i in length_Ten:
    for j in GhostTrain[9]:
        if len(j) == i:
            StationCountForTen[i] = StationCountForTen[i] + 1
counter = 0
for j in GhostTrain[9]:
    if len(j) == 46 or len(j) == 37 or len(j) == 13 or len(j) == 30 or len(j) == 27:
        del GhostTrain[9][counter]
    counter = counter + 1




length_Two=[]
StationCountForTwo={}
for i in GhostTrain[1]:
    length_Two.append(len(i))

for i in length_Two:
    StationCountForTwo[i]=0

for i in length_Two:
    for j in GhostTrain[1]:
        if len(j) == i:
            StationCountForTwo[i] = StationCountForTwo[i] + 1

counter = 0
for j in GhostTrain[1]:
    if len(j) == 19:
```

```
            del GhostTrain[1][counter]
        counter = counter + 1



length_ele=[]
StationCountForele={}
for i in GhostTrain[10]:
        length_ele.append(len(i))

for i in length_ele:
        StationCountForele[i]=0

for i in length_ele:
        for j in GhostTrain[10]:
            if len(j) == i:
                StationCountForele[i] = StationCountForele[i] + 1

counter = 0
for j in GhostTrain[10]:
        if len(j) == 3:
            del GhostTrain[1][counter]
        counter = counter + 1



Train_Move= [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
for i in GhostTrain:
        for j in i:
            if (int(j[-1][4]) > 40500) and (len(j) < (LineStationNumber[int(j[0][0])-1])*(0.5)):
                Train_Move[int(j[0][0])-1].append(j)
```

```
for i in Train_Move:
    for j in i:
        del train_number_list[int(j[0][0])-1][j[0][1]]
```

Code7

```
'''

f = open('2-1.txt',mode='r',encoding='gbk')
for line in f:
    a = line.replace('\n', '').split(',')
    print(a)


g = open('2-2.txt',mode='r',encoding='gbk')
for line in g:
    b = line.replace('\n', '').split(',')
    print(b)

'''


'''

class Graph(object):

    def __init__(self, graph_dict=None):
        """ initializes a graph object
            If no dictionary or None is given,
```

```
            an empty dictionary will be used
        """
        if graph_dict == None:
            graph_dict = {}
        self.__graph_dict = graph_dict

    def vertices(self):
        """ returns the vertices of a graph """
        return list(self.__graph_dict.keys())

    def edges(self):
        """ returns the edges of a graph """
        return self.__generate_edges()

    def add_vertex(self, vertex):
        """ If the vertex "vertex" is not in
            self.__graph_dict, a key "vertex" with an empty
            list as a value is added to the dictionary.
            Otherwise nothing has to be done.
        """
        if vertex not in self.__graph_dict:
            self.__graph_dict[vertex] = []

    def add_edge(self, edge):
        """ assumes that edge is of type set, tuple or list;
            between two vertices can be multiple edges!
        """
        edge = set(edge)
        (vertex1, vertex2) = tuple(edge)
        if vertex1 in self.__graph_dict:
            self.__graph_dict[vertex1].append(vertex2)
```

```
        else:
            self.__graph_dict[vertex1] = [vertex2]

    def __generate_edges(self):
        """ A static method generating the edges of the
            graph "graph". Edges are represented as sets
            with one (a loop back to the vertex) or two
            vertices
        """
        edges = []
        for vertex in self.__graph_dict:
            for neighbour in self.__graph_dict[vertex]:
                if {neighbour, vertex} not in edges:
                    edges.append({vertex, neighbour})
        return edges

    def __str__(self):
        res = "vertices: "
        for k in self.__graph_dict:
            res += str(k) + " "
        res += "\nedges: "
        for edge in self.__generate_edges():
            res += str(edge) + " "
        return res


    def find_path(self, start_vertex, end_vertex, path=None):
        """ 寻找从一个 v 到另一个 v 的路径，给出一条 """
        if path == None:
            path = []
        graph = self.__graph_dict
```

```
        path = path + [start_vertex]
        if start_vertex == end_vertex:
            return path
        if start_vertex not in graph:
            return None
        for vertex in graph[start_vertex]:
            if vertex not in path:
                extended_path = self.find_path(vertex,
                                               end_vertex,
                                               path)
                if extended_path:
                    return extended_path
        return None


def find_all_paths(self, start_vertex, end_vertex, path=[]):
        """ 寻找所有从一个 v 到另一个 v 的路径 """
        graph = self.__graph_dict
        path = path + [start_vertex]
        if start_vertex == end_vertex:
            return [path]
        if start_vertex not in graph:
            return []
        paths = []
        for vertex in graph[start_vertex]:
            if vertex not in path:
                extended_paths = self.find_all_paths(vertex,
                                                     end_vertex,
                                                     path)
                for p in extended_paths:
                    paths.append(p)
```

```
        return paths


    def vertex_degree(self, vertex):
        """ The degree of a vertex is the number of edges connecting
            it, i.e. the number of adjacent vertices. Loops are counted
            double, i.e. every occurence of vertex in the list
            of adjacent vertices. """
        adj_vertices =  self.__graph_dict[vertex]
        degree = len(adj_vertices) + adj_vertices.count(vertex)
        return degree


    def find_isolated_vertices(self):
        """ returns a list of isolated vertices. """
        graph = self.__graph_dict
        isolated = []
        for vertex in graph:
            print(isolated, vertex)
            if not graph[vertex]:
                isolated += [vertex]
        return isolated


    def delta(self):
        """ the minimum degree of the vertices """
        min = 100000000
        for vertex in self.__graph_dict:
            vertex_degree = self.vertex_degree(vertex)
            if vertex_degree < min:
                min = vertex_degree
```

```
        return min


    def Delta(self):
        """ the maximum degree of the vertices """
        max = 0
        for vertex in self.__graph_dict:
            vertex_degree = self.vertex_degree(vertex)
            if vertex_degree > max:
                max = vertex_degree
        return max


    def degree_sequence(self):
        """ calculates the degree sequence """
        seq = []
        for vertex in self.__graph_dict:
            seq.append(self.vertex_degree(vertex))
        seq.sort(reverse=True)
        return tuple(seq)




    def is_connected(self,
                     vertices_encountered = None,
                     start_vertex=None):
        """ determines if the graph is connected """
        if vertices_encountered is None:
            vertices_encountered = set()
        gdict = self.__graph_dict
        vertices = list(gdict.keys()) # "list" necessary in Python 3
        if not start_vertex:
            # chosse a vertex from graph as a starting point
```

```
            start_vertex = vertices[0]
        vertices_encountered.add(start_vertex)
        if len(vertices_encountered) != len(vertices):
            for vertex in gdict[start_vertex]:
                if vertex not in vertices_encountered:
                    if self.is_connected(vertices_encountered, vertex):
                        return True
        else:
            return True
        return False

def diameter(self):
        """ calculates the diameter of the graph """

        v = self.vertices()
        pairs = [ (v[i],v[j]) for i in range(len(v)-1) for j in range(i+1, len(v))]
        smallest_paths = []
        for (s,e) in pairs:
            paths = self.find_all_paths(s,e)
            smallest = sorted(paths, key=len)[0]
            smallest_paths.append(smallest)

        smallest_paths.sort(key=len)

        # longest path is at the end of list,
        # i.e. diameter corresponds to the length of this path
        diameter = len(smallest_paths[-1]) - 1
        return diameter
```

```
g = { "a" : ["d"],
                "b" : ["c"],
                "c" : ["b", "c", "d", "e"],
                "d" : ["a", "c"],
                "e" : ["c"],
                "f" : []
              }

graph = Graph(g)
print("Vertices of graph:")
print(graph.vertices())
print("Edges of graph:")
print(graph.edges())
print("Add vertex:")
graph.add_vertex("z")
print("Vertices of graph:")
print(graph.vertices())
print("Add an edge:")
graph.add_edge({"a","z"})
```

```
print("Vertices of graph:")
print(graph.vertices())
print("Edges of graph:")
print(graph.edges())
print('Adding an edge {"x","y"} with new vertices:')
graph.add_edge({"x","y"})
print("Vertices of graph:")
print(graph.vertices())
print("Edges of graph:")
print(graph.edges())



'''



'''基本数据集处理'''



g = open('2-1.txt',mode='r',encoding='gbk')
station = set()
for line in g:
    b = line.replace('\n', '').split(',')
    station.add(b[1])
g.close()

station = list(station)
```

```
station_dict = {}

for sta in station:
    station_dict[sta]=[]

for i in station:
    a = []
    g = open('2-1.txt',mode='r',encoding='gbk')
    for line in g:
        c = line.replace('\n', '').split(',')
        if c[1] == i:
            a.append([c[0],c[2]])
    station_dict[i] = a
    g.close()


'''
计算换乘站
'''


change_station=[]

for j in station:
    if not(len(station_dict[j]) == 1):
        change_station.append(j)


'''
收集按照排序的站点
```

```
'''

station_in_index=[]
g = open('2-1.txt',mode='r',encoding='gbk')
for line in g:
    c = line.replace('\n', '').split(',')
    station_in_index.append(c[1])
g.close()




transportline=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
f = open('3.txt',mode='r',encoding='gbk')
for line in f:

    '''分割数据集'''
    a = line.replace('\n', '').split(',')
    '''


    b = a[3].split(':')
    c = a[4].split(':')
    a[3] = int(b[0])*3600 + int(b[1])*60 + int(b[2])
    a[4] = int(c[0])*3600 + int(c[1])*60 + int(c[2])


    '''
    transportline[eval(a[0])-1].append(a)
```

```
'''
for i in transportline:
    a = len(i)
    for x in range(a-1):
        for y in range(a-1-x):
            if i[y][4]>i[y+1][4]:
                i[y],i[y+1] = i[y+1],i[y]
    print(i)
'''

for i in transportline[0]:
    print(i)
```

Code8

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 14 16:45:51 2019

@author: 30517
"""

plt.title('1 号线某站 11-12 点列车时间间隔')
plt.ylabel('频数')
plt.xlabel('时间/秒')
plt.hist(time,color='r')
plt.show()
```

```
'''
import matplotlib.pyplot as plt




x=[1,2,3,4,5]
y=[159.08,77.30,69.99,55.93,48.99]
plt.xticks([1,2,3,4,5],
       ['WT','Q104A','Y176F','F155A','H29A'])

e=[15.9,3.5,3.5,5.5,4.8]



plt.bar(x,y)

plt.ylabel('Expression Level of sfGFP')

ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

'''
```

Code9

```
# -*- coding: utf-8 -*-
```

"""

Created on Sat Apr 13 17:43:52 2019

@author: 30517

"""

import xgboost as xgb

from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.metrics import r2_score

from sklearn.metrics import mean_squared_error

import numpy as np

station = ['青年路', '西小口', '通州北关', '六道口', '虎坊桥', '北土城', '回龙观', '苏庄', '湾子', '管庄', '新宫', '角门西', '军事博物馆', '三元桥', '慈寿寺', '园博园', '潘家园', '善各庄', '大郊亭', '天通苑北', '广阳城', '和平门', '东风北桥', '北京站', '霍营', '什刹海', '奥林匹克公园', '健德门', '牡丹园', '阜成门', '大红门', '双井', '清源路', '稻田', '纪家庙', '珠市口', '五棵松', '劲松', '北宫门', '小红门', '安定门', '五道口', '望京东', '望京', '通州北苑', '安河桥北', '望京西', '沙河', '西二旗', '车公庄西', '蒲黄榆', '草房', '七里庄', '通运门', '枣园', '潞城', '木樨地', '芍药居', '亮马桥', '北海北', '黄渠', '回龙观东大街', '圆明园', '八角游乐园', '安贞门', '北京南站', '望京南', '北京大学东门', '朝阳门', '国家图书馆', '双桥', '育知路', '亦庄桥', '立水桥', '海淀黄庄', '北新桥', '长春桥', '崔各庄', '郝家府', '惠新西街南口', '荣昌东街', '四惠', '菜市口', '刘家窑', '大钟寺', '同济南路', '西红门', '六里桥东', '西单', '科怡路', '平西府', '梨园', '六里桥', '良乡南关', '巩华城', '安立路', '百子湾', '玉泉路', '生物医药基地', '大瓦窑', '丰台东大街', '垡头', '黄村火车站', '荣京东街', '农业展览馆', '北苑', '中关村', '成寿寺', '上地', '东夏园', '丰台科技园', '苏州街', '白石桥南', '公主坟', '长椿街', '九龙山', '天安门西', '俸伯', '南法信', '海淀五路居', '广渠门外', '化工', '林萃桥', '大屯路东', '永安里', '角门东', '和平西桥', '旧宫', '褡裢坡', '朝阳公园', '大望路', '建国门', '达官营', '泥洼', '大井', '肖村', '苹果园', '东大桥', '首经贸', '育新', '清华东路西口', '枣营', '东湖渠', '积水潭', '顺义', '物资学院路', '亦庄文化园', '丰台站', '石门', '草桥', '巴沟', '黄村西大街', 'T3 航站楼', '郭庄子', '九棵树', '

新街口', '朱辛庄', '动物园', '金台路', '南礼士路', '丰台南路', '篱笆房', '桥湾', '后沙峪', '宣武门', '高米店北', '西钓鱼台', '马泉营', '万源街', '宋家庄', '北京西站', '北运河东', '前门', '南邵', '平安里', '立水桥南', '焦化厂', '良乡大学城', '复兴门', '崇文门', '广安门内', '安华桥', '将台', '惠新西街北口', '国展', '北沙滩', '白堆子', '安德里北街', '知春里', '灵境胡同', '陶然亭', '和平里北街', '团结湖', '双合', '龙泽', '魏公村', '天安门东', '良乡大学城西', '广渠门内', '西四', '马家堡', '南楼梓庄', '阜通', '东四', '郭公庄', '义和庄', '经海路', '张郭庄', '欢乐谷景区', '西直门', '车公庄', '四惠东', '次渠', '森林公园南门', '东单', '光熙门', '八宝山', '古城', '大葆台', '莲花桥', '天通苑南', '长阳', '国贸', '花园桥', '临河里', '车道沟', '北运河西', 'T2 航站楼', '鼓楼大街', '人民大学', '八里桥', '孙河', '常营', '西苑', '柳芳', '南锣鼓巷', '良乡大学城北', '火器营', '万寿路', '西局', '花梨坎', '亦庄火车站', '公益西桥', '十里堡', '天通苑', '天宫院', '太阳宫', '沙河高教园', '王府井', '生命科学园', '灯市口', '果园', '十里河', '磁器口', '来广营', '传媒大学', '天坛东门', '呼家楼', '金台夕照', '知春路', '西土城', '高家园', '张自忠路', '北苑路北', '永泰庄', '石榴庄', '奥体中心', '分钟寺', '关庄', '土桥', '东直门', '高碑店', '雍和宫', '高米店南', '东四十条', '次渠南']

```
station_dict = {}

for sta in station:
    station_dict[sta]=[]

for i in station:
    a = []
    g = open('2-1.txt',mode='r',encoding='gbk')
    for line in g:
        c = line.replace('\n', '').split(',')
        if c[1] == i:
            a.append([c[0],c[2]])
    station_dict[i] = a
    g.close()
```

```
'''
计算换乘站
'''


change_station=[]

for j in station:
    if not(len(station_dict[j]) == 1):
        change_station.append(j)


'''
收集按照排序的站点
'''

station_in_index=[]
g = open('2-1.txt',mode='r',encoding='gbk')
for line in g:
    c = line.replace('\n', '').split(',')
    station_in_index.append(c[1])
g.close()


'''
计算每条路所需要的站点
'''
LineStationNumber=[]

for i in range(0,19):
    counter = 0
```

```
        g = open('2-1.txt',mode='r',encoding='gbk')
        for line in g:
            c = line.replace('\n', '').split(',')
            if (int(c[2])-1) == i:
                counter = counter + 1
        LineStationNumber.append(counter)
        g.close()

#LineStationNumber[13] = 7          #排除十四号线七里庄问题



'''寻找幽灵车'''


GhostTrain = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

transportline=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

f = open('3.txt',mode='r',encoding='gbk')
for line in f:
        a = line.replace('\n', '').split(',')
        b = a[3].split(':')
        c = a[4].split(':')
        a[3] = int(b[0])*3600 + int(b[1])*60 + int(b[2])
        a[4] = int(c[0])*3600 + int(c[1])*60 + int(c[2])
        transportline[eval(a[0])-1].append(a)
f.close()
```

```
train_number_list = [{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{}]


for ii in range(len(transportline)):

    train_number=set()

    for i in transportline[ii]:
        train_number.add(i[1])

    train_number=list(train_number)

    for i in train_number:
        train_number_list[ii][i]=[]

    for i in transportline[ii]:
        train_number_list[ii][i[1]].append(i)

    for i in train_number:
        if ii == 3:             #四号线合并问题
            if (len(train_number_list[ii][i]) != LineStationNumber[ii]) and (len(train_number_list[ii][i]) != 25):
                GhostTrain[ii].append(train_number_list[ii][i])


        else:
            if len(train_number_list[ii][i]) != LineStationNumber[ii]:
                GhostTrain[ii].append(train_number_list[ii][i])
```

```
'''处理十号线的问题'''
length_Ten=[]
StationCountForTen={}
for i in GhostTrain[9]:
    length_Ten.append(len(i))

for i in length_Ten:
    StationCountForTen[i]=0

for i in length_Ten:
    for j in GhostTrain[9]:
        if len(j) == i:
            StationCountForTen[i] = StationCountForTen[i] + 1
counter = 0
for j in GhostTrain[9]:
    if len(j) == 46 or len(j) == 37 or len(j) == 13 or len(j) == 30 or len(j) == 27:
        del GhostTrain[9][counter]
    counter = counter + 1



'''半夜调度'''

Train_Move= [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
for i in GhostTrain:
    for j in i:
        if (int(j[-1][4]) > 40500) and (len(j) < (LineStationNumber[int(j[0][0])-1])*(0.5)):
            Train_Move[int(j[0][0])-1].append(j)
```

```
'''去掉半夜调度车'''

for i in Train_Move:
    for j in i:
        del train_number_list[int(j[0][0])-1][j[0][1]]



'''现在 train_number_list 是已知的所有运人的车子'''



labelpostive = []
labelnegative = []

for i in range(86400):
    labelpostive.append([])
for i in range(86400):
    for j in range(276):
        labelpostive[i].append([[]])

for i in range(86400):
    labelnegative.append([])
for i in range(86400):
    for j in range(276):
        labelnegative[i].append([[i,'0',0,0]])



train_code_list=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
```

```
counter = 0

for i in train_number_list:
    for key in i:
        train_code_list[counter].append(key)
    counter = counter + 1



for i in train_code_list:
    for j in i:
        a = train_number_list[train_code_list.index(i)][j][0][2]
        b = train_number_list[train_code_list.index(i)][j][1][2]
        if int(a) < int(b):
            counter = 0
            for k in train_number_list[train_code_list.index(i)][j]:
                if counter != (len(train_number_list[train_code_list.index(i)][j])-1):
                    for p in range(k[3],k[4]+1):
                        #placeholder = []

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(p)        #添加此刻时间

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(k[0])        #添加从属线路

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train
_code_list.index(i)][j][counter+1][2])-1])+1))        #添加下一站地标

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter+1][3])
```

```
#添加下一站到达时间

#labelpostive[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1]))] = placeholder
                        counter = counter + 1
                    else:
                        for p in range(k[3],k[4]+1):
                            #placeholder = []

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(p)        #添加此刻时间

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(k[0])        #添加从属线路

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train
_code_list.index(i)][j][counter][2])-1])+1))        #添加下一站地标

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter][3])
#添加下一站到达时间

#labelpostive[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1]))] = placeholder
                        counter = counter + 1
            elif int(a) > int(b):
                counter = 0
                for k in train_number_list[train_code_list.index(i)][j]:
                    if counter != (len(train_number_list[train_code_list.index(i)][j])-1):
                        for p in range(k[3],k[4]+1):
```

```
                    #placeholder = []

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(p)        #添加此刻时间

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(k[0])        #添加从属线路

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter+1][2])-1])+1))        #添加下一站地标

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter+1][3])
#添加下一站到达时间

#labelnegative[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1]))] = placeholder
                    counter = counter + 1
                else:
                    for p in range(k[3],k[4]+1):
                        #placeholder = []

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(p)        #添加此刻时间

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(k[0])        #添加从属线路

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train
```

```
_code_list.index(i)][j][counter][2])-1])+1))        #添加下一站地标

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter][3])
#添加下一站到达时间

#labelnegative[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(
i)][j][counter][2])-1]))] = placeholder
                                counter = counter + 1



for i in range(86400):
    for j in range(276):
        labelpostive[i][j].append(labelnegative[i][j][0])


label = np.array(labelpostive)









people_on = []
people_off = []


g = open('1.txt',mode='r',encoding='gbk')
for line in g:
```

```
        a = line.replace('\n', '').split(',')
        b = a[3].split(':')
        c = a[4].split(':')
        a[3] = int(b[0])*3600 + int(b[1])*60 + int(b[2])
        a[4] = int(c[0])*3600 + int(c[1])*60 + int(c[2])
        on = [a[1],a[3]]
        off = [a[2],a[4]]
        if ((a[4]-a[3]) > 0) and (0 < int(a[1]) < 329) and (0 < int(a[2]) < 329): #数据清洗，失误数据：站点错误和时间记录错误
            people_on.append(on)
            people_off.append(off)
g.close()



feature = []

for i in range(86400):
    feature.append([])
for i in range(86400):
    for j in range(276):
        feature[i].append([[i,0,0,0]])

for i in people_on:
    feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][2] = feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][2] + 1
for i in people_off:
```

```
        feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][3] =
feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][3] + 1




for i in range(86400):
        for j in range(276):
                if i != 0:
                        feature[i][j][0][1] = feature[i-1][j][0][1] + feature[i-1][j][0][2] -
feature[i-1][j][0][3]

feature = np.array(feature)









X=[]
Y=[]


# 读取文件原始数据

for i in range(43200):
        for j in range(276):
                for k in range(4):
```

```
                if i != 1:
                    x = []
                    x.append(feature[i][j][0][k])
                    X=np.r_[X,x]
                else:
                    x = []
                    x.append(feature[i][j][0][k])
                    X=x


for i in range(43200):
    for j in range(276):
        for k in range(4):
            if i != 1:
                y = []
                y.append(label[i][j][0][k])
                Y=np.r_[Y,y]
            else:
                y = []
                y.append(label[i][j][0][k])
                Y=y

'''
X= feature[0:43200,:,:][0].reshape((1,-1))
Y= label[0:43200,:,:][0].reshape((1,-1))
'''
```

```
# XGBoost 训练过程
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2, random_state=0)

model = xgb.XGBRegressor(max_depth=3,
                         learning_rate=0.1,
                         n_estimators=100,
                         silent=True,
                         objective='reg:linear',
                         nthread=-1,
                         gamma=0,
                         min_child_weight=1,
                         max_delta_step=0,
                         subsample=0.85,
                         colsample_bytree=0.7,
                         colsample_bylevel=1,
                         reg_alpha=0,
                         reg_lambda=1,
                         scale_pos_weight=1,
                         seed=1440,
                         missing=None)

model.fit(X_train, y_train)

#  对测试集进行预测
ans = model.predict(X_test)
```

```
print(r2_score(ans,y_test))
print(mean_squared_error(ans,y_test))
#  显示重要特征
```

Code10

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 13 13:45:24 2019

@author: 30517
"""

import xgboost as xgb
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import numpy as np


station = ['青年路', '西小口', '通州北关', '六道口', '虎坊桥', '北土城', '回龙观', '苏庄', '湾子', '管庄', '新宫', '角门西', '军事博物馆', '三元桥', '慈寿寺', '园博园', '潘家园', '善各庄', '大郊亭', '天通苑北', '广阳城', '和平门', '东风北桥', '北京站', '霍营', '什刹海', '奥林匹克公园', '健德门', '牡丹园', '阜成门', '大红门', '双井', '清源路', '稻田', '纪家庙', '珠市口', '五棵松', '劲松', '北宫门', '小红门', '安定门', '五道口', '望京东', '望京', '通州北苑', '安河桥北', '望京西', '沙河', '西二旗', '车公庄西', '蒲黄榆', '草房', '七里庄', '通运门', '枣园', '潞城', '木樨地', '芍药居', '亮马桥', '北海北', '黄渠', '回龙观东大街', '圆明园', '八角游乐园', '安贞门', '北京南站', '望京南', '北京大学东门', '朝阳门', '国家图书馆', '双桥', '育知路', '亦庄桥', '立水桥', '海淀黄庄', '北新桥', '长春桥', '崔各庄', '郝家府', '惠新西街南口', '荣昌东街', '四惠', '菜市
```

口', '刘家窑', '大钟寺', '同济南路', '西红门', '六里桥东', '西单', '科怡路', '平西府', '梨园', '六里桥', '良乡南关', '巩华城', '安立路', '百子湾', '玉泉路', '生物医药基地', '大瓦窑', '丰台东大街', '垡头', '黄村火车站', '荣京东街', '农业展览馆', '北苑', '中关村', '成寿寺', '上地', '东夏园', '丰台科技园', '苏州街', '白石桥南', '公主坟', '长椿街', '九龙山', '天安门西', '俸伯', '南法信', '海淀五路居', '广渠门外', '化工', '林萃桥', '大屯路东', '永安里', '角门东', '和平西桥', '旧宫', '褡裢坡', '朝阳公园', '大望路', '建国门', '达官营', '泥洼', '大井', '肖村', '苹果园', '东大桥', '首经贸', '育新', '清华东路西口', '枣营', '东湖渠', '积水潭', '顺义', '物资学院路', '亦庄文化园', '丰台站', '石门', '草桥', '巴沟', '黄村西大街', 'T3 航站楼', '郭庄子', '九棵树', '新街口', '朱辛庄', '动物园', '金台路', '南礼士路', '丰台南路', '篱笆房', '桥湾', '后沙峪', '宣武门', '高米店北', '西钓鱼台', '马泉营', '万源街', '宋家庄', '北京西站', '北运河东', '前门', '南邵', '平安里', '立水桥南', '焦化厂', '良乡大学城', '复兴门', '崇文门', '广安门内', '安华桥', '将台', '惠新西街北口', '国展', '北沙滩', '白堆子', '安德里北街', '知春里', '灵境胡同', '陶然亭', '和平里北街', '团结湖', '双合', '龙泽', '魏公村', '天安门东', '良乡大学城西', '广渠门内', '西四', '马家堡', '南楼梓庄', '阜通', '东四', '郭公庄', '义和庄', '经海路', '张郭庄', '欢乐谷景区', '西直门', '车公庄', '四惠东', '次渠', '森林公园南门', '东单', '光熙门', '八宝山', '古城', '大葆台', '莲花桥', '天通苑南', '长阳', '国贸', '花园桥', '临河里', '车道沟', '北运河西', 'T2 航站楼', '鼓楼大街', '人民大学', '八里桥', '孙河', '常营', '西苑', '柳芳', '南锣鼓巷', '良乡大学城北', '火器营', '万寿路', '西局', '花梨坎', '亦庄火车站', '公益西桥', '十里堡', '天通苑', '天宫院', '太阳宫', '沙河高教园', '王府井', '生命科学园', '灯市口', '果园', '十里河', '磁器口', '来广营', '传媒大学', '天坛东门', '呼家楼', '金台夕照', '知春路', '西土城', '高家园', '张自忠路', '北苑路北', '永泰庄', '石榴庄', '奥体中心', '分钟寺', '关庄', '土桥', '东直门', '高碑店', '雍和宫', '高米店南', '东四十条', '次渠南']

```
station_dict = {}

for sta in station:
    station_dict[sta]=[]

for i in station:
    a = []
    g = open('2-1.txt',mode='r',encoding='gbk')
```

```
    for line in g:
        c = line.replace('\n', '').split(',')
        if c[1] == i:
            a.append([c[0],c[2]])
    station_dict[i] = a
    g.close()


'''
计算换乘站
'''


change_station=[]

for j in station:
    if not(len(station_dict[j]) == 1):
        change_station.append(j)


'''
收集按照排序的站点
'''

station_in_index=[]
g = open('2-1.txt',mode='r',encoding='gbk')
for line in g:
    c = line.replace('\n', '').split(',')
    station_in_index.append(c[1])
g.close()
```

```
'''
计算每条路所需要的站点
'''
LineStationNumber=[]

for i in range(0,19):
    counter = 0
    g = open('2-1.txt',mode='r',encoding='gbk')
    for line in g:
        c = line.replace('\n', '').split(',')
        if (int(c[2])-1) == i:
            counter = counter + 1
    LineStationNumber.append(counter)
    g.close()

#LineStationNumber[13] = 7        #排除十四号线七里庄问题




'''寻找幽灵车'''


GhostTrain = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

transportline=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

f = open('3.txt',mode='r',encoding='gbk')
for line in f:
    a = line.replace('\n', '').split(',')
```

```
        b = a[3].split(':')
        c = a[4].split(':')
        a[3] = int(b[0])*3600 + int(b[1])*60 + int(b[2])
        a[4] = int(c[0])*3600 + int(c[1])*60 + int(c[2])
        transportline[eval(a[0])-1].append(a)
f.close()


train_number_list = [{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{}]


for ii in range(len(transportline)):

        train_number=set()

        for i in transportline[ii]:
                train_number.add(i[1])

        train_number=list(train_number)

        for i in train_number:
                train_number_list[ii][i]=[]

        for i in transportline[ii]:
                train_number_list[ii][i[1]].append(i)

        for i in train_number:
                if ii == 3:             #四号线合并问题
                        if      (len(train_number_list[ii][i])      !=      LineStationNumber[ii])      and
(len(train_number_list[ii][i]) != 25):
                                GhostTrain[ii].append(train_number_list[ii][i])
```

```
            else:
                if len(train_number_list[ii][i]) != LineStationNumber[ii]:
                    GhostTrain[ii].append(train_number_list[ii][i])



'''处理十号线的问题'''
length_Ten=[]
StationCountForTen={}
for i in GhostTrain[9]:
    length_Ten.append(len(i))

for i in length_Ten:
    StationCountForTen[i]=0

for i in length_Ten:
    for j in GhostTrain[9]:
        if len(j) == i:
            StationCountForTen[i] = StationCountForTen[i] + 1
counter = 0
for j in GhostTrain[9]:
    if len(j) == 46 or len(j) == 37 or len(j) == 13 or len(j) == 30 or len(j) == 27:
        del GhostTrain[9][counter]
    counter = counter + 1
```

```
'''半夜调度'''

Train_Move= [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
for i in GhostTrain:
    for j in i:
        if (int(j[-1][4]) > 40500) and (len(j) < (LineStationNumber[int(j[0][0])-1])*(0.5)):
            Train_Move[int(j[0][0])-1].append(j)

'''去掉半夜调度车'''

for i in Train_Move:
    for j in i:
        del train_number_list[int(j[0][0])-1][j[0][1]]



'''现在 train_number_list 是已知的所有运人的车子'''



labelpostive = []
labelnegative = []

for i in range(86400):
    labelpostive.append([])
for i in range(86400):
    for j in range(276):
        labelpostive[i].append([[]])

for i in range(86400):
```

```
    labelnegative.append([])
for i in range(86400):
    for j in range(276):
        labelnegative[i].append([[i,'0',0,0]])


train_code_list=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

counter = 0

for i in train_number_list:
    for key in i:
        train_code_list[counter].append(key)
    counter = counter + 1


for i in train_code_list:
    for j in i:
        a = train_number_list[train_code_list.index(i)][j][0][2]
        b = train_number_list[train_code_list.index(i)][j][1][2]
        if int(a) < int(b):
            counter = 0
            for k in train_number_list[train_code_list.index(i)][j]:
                if counter != (len(train_number_list[train_code_list.index(i)][j])-1):
                    for p in range(k[3],k[4]+1):
                        #placeholder = []

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(p)        #添加此刻时间

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
```

```
[j][counter][2])-1])][0].append(k[0])        #添加从属线路

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train
_code_list.index(i)][j][counter+1][2])-1])+1))        #添加下一站地标

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter+1][3])
#添加下一站到达时间

#labelpostive[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1]))] = placeholder
                    counter = counter + 1
                else:
                    for p in range(k[3],k[4]+1):
                        #placeholder = []

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(p)        #添加此刻时间

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(k[0])        #添加从属线路

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train
_code_list.index(i)][j][counter][2])-1])+1))        #添加下一站地标

labelpostive[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)]
[j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter][3])
#添加下一站到达时间
```

```
#labelpostive[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1]))] = placeholder
                                counter = counter + 1
            elif int(a) > int(b):
                counter = 0
                for k in train_number_list[train_code_list.index(i)][j]:
                    if counter != (len(train_number_list[train_code_list.index(i)][j])-1):
                        for p in range(k[3],k[4]+1):
                            #placeholder = []

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1])][0].append(p)          #添加此刻时间

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1])][0].append(k[0])          #添加从属线路

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train
_code_list.index(i)][j][counter+1][2])-1])+1))          #添加下一站地标

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)
][j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter+1][3])
#添加下一站到达时间

#labelnegative[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(
i)][j][counter][2])-1]))] = placeholder
                                counter = counter + 1
                    else:
                        for p in range(k[3],k[4]+1):
                            #placeholder = []
```

```
labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(p)        #添加此刻时间

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(k[0])        #添加从属线路

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(str(station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])+1))        #添加下一站地标

labelnegative[p][station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1])][0].append(train_number_list[train_code_list.index(i)][j][counter][3])
#添加下一站到达时间

#labelnegative[p,(station.index(station_in_index[int(train_number_list[train_code_list.index(i)][j][counter][2])-1]))] = placeholder
                                counter = counter + 1



for i in range(86400):
    for j in range(276):
        labelpostive[i][j].append(labelnegative[i][j][0])


label = np.array(labelpostive)
```

```
people_on = []
people_off = []


g = open('1.txt',mode='r',encoding='gbk')
for line in g:
    a = line.replace('\n', '').split(',')
    b = a[3].split(':')
    c = a[4].split(':')
    a[3] = int(b[0])*3600 + int(b[1])*60 + int(b[2])
    a[4] = int(c[0])*3600 + int(c[1])*60 + int(c[2])
    on = [a[1],a[3]]
    off = [a[2],a[4]]
    if ((a[4]-a[3]) > 0) and (0 < int(a[1]) < 329) and (0 < int(a[2]) < 329):
#数据清洗，失误数据：站点错误和时间记录错误
        people_on.append(on)
        people_off.append(off)
g.close()



feature = []

for i in range(86400):
    feature.append([])
```

```
for i in range(86400):
    for j in range(276):
        feature[i].append([[i,0,0,0]])

for i in people_on:
    feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][2] =
feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][2] + 1
for i in people_off:
    feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][3] =
feature[i[1]][station.index(station_in_index[int(i[0])-1])][0][3] + 1




for i in range(86400):
    for j in range(276):
        if i != 0:
            feature[i][j][0][1] = feature[i-1][j][0][1] + feature[i-1][j][0][2] -
feature[i-1][j][0][3]

feature = np.array(feature)




X=[]
```

```
Y=[]


# 读取文件原始数据

for i in range(43200):
    for j in range(276):
        for k in range(4):
            if i != 1:
                x = []
                x.append(feature[i][j][0][k])
                X=np.r_[X,x]
            else:
                x = []
                x.append(feature[i][j][0][k])
                X=x


for i in range(43200):
    for j in range(276):
        for k in range(4):
            if i != 1:
                y = []
                y.append(label[i][j][0][k])
                Y=np.r_[Y,y]
            else:
                y = []
                y.append(label[i][j][0][k])
                Y=y

'''
```

```
X= feature[0:43200,:,:][0].reshape((1,-1))
Y= label[0:43200,:,:][0].reshape((1,-1))
'''



# XGBoost 训练过程
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2, random_state=0)

model = xgb.XGBRegressor(max_depth=2,
                         learning_rate=0.1,
                         n_estimators=100,
                         silent=True,
                         objective='reg:linear',
                         nthread=-1,
                         gamma=0,
                         min_child_weight=1,
                         max_delta_step=0,
                         subsample=0.85,
                         colsample_bytree=0.7,
                         colsample_bylevel=1,
                         reg_alpha=0,
                         reg_lambda=1,
                         scale_pos_weight=1,
                         seed=1440,
                         missing=None)
```

```
model.fit(X_train, y_train)


# 对测试集进行预测
ans = model.predict(X_test)
print(r2_score(ans,y_test))
print(mean_squared_error(ans,y_test))
# 显示重要特征
```