赛区评阅编号（由赛区组委会填写）：

2017 高教社杯全国大学生数学建模竞赛

承　诺　书

我们仔细阅读了《全国大学生数学建模竞赛章程》和《全国大学生数学建模竞赛参赛规则》(以下简称为“竞赛章程和参赛规则”,可从全国大学生数学建模竞赛网站下载)。

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章程和参赛规则的，如果引用别人的成果或资料（包括网上资料），必须按照规定的参考文献的表述方式列出，并在正文引用处予以标注。在网上交流和下载他人的论文是严重违规违纪行为。

我们以中国大学生名誉和诚信郑重承诺，严格遵守竞赛章程和参赛规则，以保证竞赛的公正、公平性。如有违反竞赛章程和参赛规则的行为，我们将受到严肃处理。

我们授权全国大学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（从 A/B/C/D 中选择一项填写）：B

我们的报名参赛队号（12 位数字全国统一编号）：4321

参赛学校（完整的学校全称，不含院系名）：浙江大学

参赛队员 (打印并签名) ：1. 刘昊洋

2. 曹子浩

3. 徐国瑞

指导教师或指导教师组负责人 (打印并签名)：谈之奕

（指导教师签名意味着对参赛队的行为和论文的真实性负责）

日期: 2017 年 08 月 15 日

（请勿改动此页内容和格式。此承诺书打印签名后作为纸质论文的封面，注意电子版论文中不得出现此页。以上内容请仔细核对，如填写错误，论文可能被取消评奖资格。）

赛区评阅编号（由赛区组委会填写）：

2017 高教社杯全国大学生数学建模竞赛

编 号 专 用 页

赛区评阅记录（可供赛区评阅时使用）：

评阅人						
备注						

送全国评阅统一编号（由赛区组委会填写）：

全国评阅随机编号（由全国组委会填写）：

（请勿改动此页内容和格式。此编号专用页仅供赛区和全国评阅使用，参赛队打印后装订到纸质论文的第二页上。注意电子版论文中不得出现此页。）

交巡警服务平台的设置与调度

目录

摘要

“有困难找警察”，是家喻户晓的一句流行语。警察肩负着刑事执法、治安管理、交通管理、服务群众四大职能。为了更有效地贯彻实施这些职能，需要在市区的一些交通要道和重要部位设置交巡警服务平台。每个交巡警服务平台的职能和警力配备基本相同。由于警务资源是有限的，如何根据城市的实际情况与需求合理地设置交巡警服务平台、分配各平台的管辖范围、调度警务资源是警务部门面临的一个实际课题。

为了解决问题一，我们将道路分布情况以及交巡警服务平台分布情况抽象成为图，道路交点作为节点，道路为节点之间的线段。根据已经给出的数据，我们首先用坐标与编号描述了各个节点的位置，之后用矩阵来刻画各点之间的距离关系。生成图上各节点的邻接矩阵后，按照 Dijkstra 算法找到每个交巡警服务平台到各个节点的最短路程，再进行删选就可以得到各交巡警服务平台所管辖的节点，由此得到管辖的区域。

为了解决问题二，由于实际中一个平台的警力最多封锁一个路口，为在突发情况时得到合理的的调度方案，我们引入运筹学中 0-1 整数规划模型，通过解数学规划的方法得出封锁 13 个路口的合理封锁方案。

对于问题三，从每个交巡警平台的工作负担以及出警时间长短考虑，分别筛选出负担较重的平台以及所需出警时间较长的节点，综合考虑后选择了五个节点作为新的平台，在缓解了原有平台的工作负担的同时也缩短了出警的总时间。

针对问题四，首先从全市范围内考虑，以能否 3 分钟到达以及工作负担的平均分配为依据评价该市当前服务平台安排的合理情况，类似于问题三对当前交巡警服务平台安排进行优化。

针对问题五，在犯罪嫌疑人已逃窜 3 分钟的情况下，基于警力占用最少，封锁盲点最少，封锁时间最短的原则，本文在图论的基础上建立了围捕逃犯模型，通过考虑罪犯的可能逃逸路径，围捕节点选择，围捕警力分配选择三个方面，采用了分层围堵的方案，在节省警力，争取的时间的原则下，给出调度全市交巡警服务平台警力资源的围堵方案。

本模型建立了在理想条件的交巡警平台的最优设置，可给生活中交巡警平台的设置予与参考，有一定的实际价值，可使交巡警在接到任务后更好的利用较短时间分配救援力量和选择最佳行进路径。并且该模型也可运用到其他最优选址问题中去，比如关于消防救援工作最优路径问题、重大生产安全事故应急救援问题、公共交通的最优路径问题等。

关键字： Dijkstra 算法　0-1 规划　匹配问题　围捕模型

一、 问题重述

关于交巡警平台主要需要解决以下几个问题：

(1) 为该市中心城区 A 各交巡警服务平台分配管辖范围，使其在所管辖的范围内出现突发事件时，尽量能在 3 分钟内有交巡警到达事发地。

(2) 对于发生重大突发事件，需要调度全区 20 个交巡警服务平台的警力资源，进出该区的 13 条交通要道实现快速全封锁。实际中一个平台的警力最多封锁一个路口，请给出该区交巡警服务平台警力合理的调度方案。

(3) 根据现有交巡警服务平台的工作量不均衡和有些地方出警时间过长的实际情况，拟在该区内再增加 2 至 5 个平台，请确定需要增加平台的具体个数和位置。

(4) 针对全市的具体情况，按照设置交巡警服务平台的原则和任务，分析研究该市现有交巡警服务平台设置方案的合理性。如果有明显不合理，请给出解决方案。

(5) 如果该市地点 P（第 32 个节点）处发生了重大刑事案件，在案发 3 分钟后接到报警，犯罪嫌疑人已驾车逃跑。为了快速搜捕嫌疑犯，请给出调度全市交巡警服务平台警力资源的最佳围堵方案。

二、 问题分析

首先，从条件可以知道，警车的速度是恒定的，所以在本题中，我们可以用路程来考察时间。而道路分布情况以及交巡警服务平台的布置情况已经在图中给出，我们第一步要做的就是把各点的坐标确定出来。

问题一要求求解各交巡警服务平台分配管辖范围，使其在所管辖的范围内出现突发事件时，尽量能在 3 分钟内有交巡警（警车的时速为 60km/h）到达事发地。我们首先求出从各个节点到其他节点间的邻接矩阵，再根据图论中的 Dijkstra 算法，可以计算出第 i 个节点到 j 第个节点的最短距离进而得到有路口节点到交通巡警服务平台节点间的最短距离矩阵，最后用指派模型对交巡警各个服务平台管辖范围进行约束安排。

问题二要求在突发情况下，给出 A 区的 13 个路口的最佳封锁方案，基于“一个平台最多封锁一个路口”的限制，从警力分配最优，到达时间路口最短的原则出发，我们引入 0-1 整数规划模型，通过对第 i 路口节点到第 j 个交通路口的最短距离 a_{ij} 的分析，得出对 13 个路口的最佳封锁方案。

问题三的关键在于解决平台的工作量不均以及部分节点出警时间较长这两个问题。在前两问中，我们已经得出了出警时间较长节点的集合，同时通过比较各平台管辖节点发案率与平台平均发案率，找到工作负荷较大的平台。结合这两个集合的情况又增设了五个交巡警平台，使得总出警时间减少并且减少了各平台的工作负担。

问题四要求评价该市现有交巡警服务平台设置方案的合理性，首先从各区分配平台数的均衡性和能不能保证全市 3 分钟内可达，分析其合理性，之后在对平台设置方案进行优化，以全市 3 分钟内可达为要求，得到最少需要平台为 105 个，且按区内工作量均衡原则将各区节点分配给各区平台。

问题五要求给出调度全市交巡警服务平台警力资源的最佳围堵方案，基于此我们在图论的基础上建立了围捕模型，通过考虑罪犯的可能逃逸路径，围捕节点选择，围捕警力分配选择三个方面，同时尽可能使封锁盲点最少，封锁时间最短，采用了分层围堵的方案，在节省警力，争取的时间的原则下，给出调度全市交巡警服务平台警力资源的围堵方案。

三、 模型的假设

- 所有道路均为双行道。
- 出警时道路通畅（无交通事故、交通阻塞等发生），警车以恒定时速 60 km/h 行驶。
- 警车行驶的整个路途中，转弯处不需要花费时间。
- 发案点仅在各个节点上。
- 犯罪嫌疑人的逃窜速度为 60 km/h。

四、 符号说明

五、 建立模型、分析与求解

5.1 交巡警服务平台分配管辖范围

5.1.1 求最短距离矩阵

因为每个交巡警服务平台的职能和警力配备基本相同，所以要考虑每个平台在工作量均衡的条件下能最短时间内到达突发事件现场，主要考虑的方向是各个平台管辖范围内的总的时间最短（最短时间可转化为出警的最短路程）。由各节点间的距离矩阵和图论中的 Dijkstra 算法，可以计算出第 i 路口节点到第 j 个服务平台的最短距离 a_{ij}，是一个 92×92 的邻接矩阵，然后从中抽出 92 个节点分别到 20 个服务平台的最短距离，进而得到有路口节点到交通巡警服务平台节点间的最短距离矩阵。

符号	意义
a_{ij}	第 i 个节点到第 j 个巡警平台的最短距离
x_{ij}	第 i 个节点是否由第 j 个巡警平台管辖
P	巡警平台所构成的集合
V	A 区所有节点所构成的集合
A_1	节点 i 就是巡警平台
A_2	节点 i 到任何巡警平台所需时间大于 3 分钟
A_3	至少一个巡警平台 j 到节点 i 的时间不超过 3 分钟
$\bar{b}$	A 区每天平均案发率
b_i	i 节点每天的案发率

代码实现 所给的附件是一个表格，我们使用了 Python 读取数据，并转化为 numpy 对象。在此基础上，我们定义了几个可复用的函数，可以方标地读取两个节点之间地距离。

我们求最短距离矩阵的问题基本上是一个简单的图论问题。再结合使用表格中的道路信息，即可生成一个表征当前图的带权邻接矩阵，记为 M, 元素为 m_{ij}。若节点 i 到节点 j 有直接的路相连接，则 $m_{ij} = m_{ji} = Distance(i, j)$，如果没有，则为定义的 -1，表示无穷大。[3]

在得到邻接矩阵的基础上，我们可以进行下一步的得到最短距离的操作。对该邻接矩阵使用 Dijkstra 算法，可以得到任意两个节点之间的最短距离以及这个距离是怎样的路径达到的。自此，我们就完成了求最短距离矩阵的操作，得到了一个获得任意两个节点之间最短距离及路径的函数。

5.1.2 最优分配方案的确定

根据题目中“尽量能在 3 分钟内有交巡警到达出发地”和“警车的时速为 60 km/h”的假设，可以算出车程对应的是 3 km 。即若某个路口的节点到某个交巡警服务平台节点的距离小于等于 3 km ，则该路口就有可能被该交巡警服务台所管辖。由此可以根据最短距离矩阵可以将 A 区的节点分为三类:[1]

(1) $A_1 = \{i \in V | \min\limits_{j \in P} a_{ij} = 0\}$，即节点 i 就是交巡警服务平台节点，这样的节点有 $1, 2, \cdots, 20$。则 $A_1 = \{1, 2, \cdots, 20\}$ 。V 为 A 区所有路口节点所构成的集合。

(2) $A_2 = \{i \in V | \min_{j \in P} a_{ij} > 3\}$，即节点到任何交巡警服务平台节点的最短距离大于3km，这样的节点有 28, 29, 38, 39, 61, 92。

(3) $A_3 = \{i \in V | \min_{j \in P} 0 < a_{ij} <= 3\}$，即至少存在一个交巡警服务平台节点 i 到节点 j 的距离大于 0 且不超过 3km，除了第一类和第二类以外的所有节点都属于这样的节点。

根据原始数据整理我们得出交巡警服务平台节点所构成的集合 P, $P = \{1, 2, \cdots, 20\}$, 为了确定约束条件，运用运筹学上的指派模型建立 0-1 变量：

$$x_{ij} = \begin{cases} 1 & \text{第 i 个路口由第 j 个交巡警服务平台管辖} \\ 0 & \text{第 i 个路口不由第 j 个交巡警服务平台管辖} \end{cases} \tag{1}$$

其中 $i \in V, j \in P$，得出了三种分类情况下各个路口的管辖情况：

(1) 若 $i \in A_1$，则

$$x_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{2}$$

即交巡警服务平台由自己管辖。

(2) 若 $i \in A_2$，则根据时间最先的原则，节点 i 应由它最近的服务平台管辖，不妨设该交巡警服务平台的节点数 j_0，即

$$x_{ij} = \begin{cases} 1 & i = j_0 \\ 0 & i \neq j_0 \end{cases} \tag{3}$$

(3) 若 $i \in A_3$，则至少存在一个交巡警服务平台节点到节点 i 的距离大于 0 且不超过 3km 不妨设满足该条件的服务平台有 k 个，记为 $j_1, j_2, \cdots, j_k$，那么节点最终肯定由其中一个服务平台所管辖，所以有：

$$x_{ij_1} + x_{ij_2} + \cdots, x_{ij_k} = 1 \tag{4}$$

以上就是变量 x_{ij} 的约束。

目标函数的建立主要是依据使得各交巡警服务平台到其管辖区内个路口节点的时间总和尽可能的短。根据这一思想，构造如下规划：[2]

$$z = \min \sum_{i=1}^{92} \sum_{j=1}^{20} c_{ij} \times x_{ij} \tag{5}$$

$$s.t=\begin{cases} x_{ij}=\begin{cases}1 & i=j\\ 0 & i\neq j\end{cases}, i\in A_1 \\ x_{ij}=\begin{cases}1 & i=j_0\\ 0 & i\neq j_0\end{cases}, i\in A_2 \\ x_{ij_1}+x_{ij_2}+\cdots, x_{ij_k}=1, i\in A_3 \end{cases} \tag{6}$$

将出警时间考虑进去，使得各交巡警服务平台到其管辖区内个路口节点的时间总和尽可能的短。由于题目中假设警车的速度恒定，所以时间问题可以转化为距离问题进而得到如下优化安排模型：

编程求得最优值为 1032.19，再根据最优解 x_{ij} 的取值可以得到管辖范围分配的合理方案如表 1 所示。

表 **1**　交巡警管辖范围的合理分配方案

交警平台	管辖路口	交警平台	管辖路口
1	1 67 68 69 71 73 74 75 76 78	11	11 26 27
2	2 39 40 43 44 70 72	12	12 25
3	3 45 55 65 66	13	13 21 22 23 24
4	4 57 60 62 63 64	14	14
5	5 49 50 51 52 53 56 58 59	15	15 28 29
6	6	16	16 36 37 38
7	7 30 32 47 48 61	17	17 41 42
8	8 33 46	18	18 80 81 82 83
9	9 31 32 34 35 45	19	19 77 79
10	10	20	20 84 85 86 87 88 89 90 91 92

由上述结果可看出，在案发当时交巡警服务平台即接到报警并立即派出警力前往的情况下，仍不能保证在 3 分钟内有交巡警到达事发地的路口有 28,29,38,39,61,92 共 6 个；管辖范围仅有一个路口的有平台 6，10，14，且为它们本身；平台 20,1,5 的管辖路口数较多，分别为 10 个，10 个，9 个。由此可以看出存在交巡警平台的工作量不均衡和有些地方出警时间过长的问题。

代码实现 我们使用了 Python 的 Pulp 库进行 0-1 规划的求解，在定义最小封锁时间的情况下进行对该 0-1 规划的求解，查看规划是否可解。如果解集为空集，测说明当前定义的最小封锁时间太小，无法满足，再逐步加大最小封锁时间。最终可以得到一组具有最小封锁时间的，且满足约束条件的一组解。

5.2 警力合理的封锁调度方案

由问题一可知，A 区交警服务平台有 20 个，将交警服务平台的个数记为 i，则 $i = 1, 2, \cdots, 20$，记要封锁的出口数为 j，$j = 1, 2, \cdots, 13$。基于“一个平台最多封锁一个路口”的限制，从警力分配最优，到达时间路口最短的原则出发，可以将 20 个交警服务平台封锁 13 个出口问题转化为 13 个交警服务平台调度到 13 个节点的问题。基于以上分析，引入 0-1 整数规划模型，记 0-1 变量 x_{ij}，x_{ij} 表示巡警服务平台 i 对要道 j 进行封锁情况，若巡警服务平台 i 对节点 j 进行封锁，记 $x_{ij} = 1$，否则记为 $x_{ij} = 1$，即：

$$x_{ij} = \begin{cases} 1 & \text{第 i 个路口由第 j 个交巡警服务平台封锁} \\ 0 & \text{第 i 个路口不由第 j 个交巡警服务平台封锁} \end{cases} \tag{7}$$

其决策变量有 260 个。

本题要求对 13 条要道进行快速封锁，即要求巡警服务台对 13 条交通要道进行全部封锁所需时间最短的调度方案。从警力分配最优，到达时间路口最短的原则出发，在假设警车行驶速度相同的条件下，可转化为求巡警服务台与要道最大距离最短的调度方案。则本题目标函数 $f = \max(c_{ij}x_{ij})$，其中 c_{ij} 为 20 个服务平台到 13 个交通要道的最短距离，为一个 20×13 的矩阵。则目标函数为: $\min f = \max\limits_{\substack{1<=i<=20 \\ 1<=j<=13}} (c_{ij}x_{ij})$。

根据问题的要求，每个交通要道必须有一个交警服务平台对其进行封锁，即对于 $j = 1, 2, \cdots, 13$，应有 $\sum\limits_{j=1}^{13} x_{ij} = 1$，对于 $i = 1, 2, \cdots, 20$，应该有 $\sum\limits_{i=1}^{20} x_{ij} <= 1$。

综上所述，可以得到此问题的优化模型为:

$$\min f = \max_{\substack{1<=i<=20 \\ 1<=j<=13}} (c_{ij}x_{ij}) \tag{8}$$

$$s.t. = \begin{cases} \sum\limits_{j=1}^{13} x_{ij} = 1, j = 1, 2, \cdots, 13 \\ \sum\limits_{i=1}^{20} x_{ij} <= 1, i = 1, 2, \cdots, 20 \\ x_{ij} = 0 \text{ 或 } 1 \end{cases} \tag{9}$$

按照附件 2 中 20 个巡警服务台和 13 条交通要道的顺序进行编号，引入决策变量，根据已经建立的模型中的约束条件和目标函数。

从求解结果显示，目标函数的最小值为 80.15，即封锁 13 条交通要道的最少时间为 8.015 分钟，基于此，给出了 A 区交巡警服务平台警力合理的调度方案如下表 2。

表 **2**　**A** 区交巡警服务平台警力封锁 **13** 条交通要道的调度方案

交巡警平台位置标号	出入 A 区的路口标号	到达路口的距离
7	29	80.155
10	12	75.866
12	22	68.825
16	14	67.417
14	23	64.733
6	16	62.586
11	21	50.723
1	62	48.852
4	38	48.610
15	28	47.518
8	30	30.608
5	48	24.758
13	24	23.854

5.3 交巡警服务平台的合理增添方案

用 $\overline{b}$ 表示整个 A 区服务平台每天的平均案发率，即服务平台每天的平均工作量，b_i 表示 i 路口节点每天的案发率，则有 $\overline{b}=\frac{1}{20}\sum_{i=1}^{92}b_i$ ，考察每个平台管辖范围内每天的案发率总和。记某平台每天发案率与平均水平之差为该平台对应的偏差。根据偏差的定义可知，若偏差大于零，则说明该平台有工作量负荷的情况，反之说明没有工作量负荷的情况。利用附件所给数据和已求出各平台所管辖的范围，可以计算出全区服务平台的平均工作量 $\overline{b}=6.225$。为了更详细的了解各服务平台的工作量与平均工作量的差别，通过计算求出了全区各服务平台工作量的偏差，如表 4。

表 **4**　各交巡警服务平台的工作量偏差

巡警点	1	2	3	4	5	6	7	8	9	10
偏差	4.075	3.475	-0.625	0.375	3.475	-3.725	3.375	-1.225	1.975	-4.625
巡警点	11	12	13	14	15	16	17	18	19	20
偏差	-1.625	-2.225	2.275	-3.725	-1.425	-1.225	-0.925	-0.125	-2.825	5.275

通过以上统计，我们可以进行第一步筛选，从表 3 中可找出 20 个交巡警服务平台中工作量负荷的服务台，将各服务台工作量负荷程度由大到小进行排列有：20 号、1 号、2 号、5 号、7 号、13 号，这里我们只筛选了一些工作量负荷相对较大的平台。

通过各平台工作量不均衡进行了第一步筛选，然后以各路口节点出警时间不同进行第二步筛选。根据各交巡警服务平台到各节点的最短距离矩阵 C，我们筛选出了一些出警时间超过三分钟的路口节点，即节点距离管辖自己的交巡警服务平台超过 3km。得到的目标有：$d_{2(39)}=3.68, d_{7(61)}=4.19, d_{2(39)}=3.68, d_{15(28)}=4.75, d_{15(29)}=5.70, d_{16(38)}=3.41, d_{20(92)}=3.60$。

此筛选表明 39、61、28、29、38、92 这些节点的出警时间过长，需要通过增加交巡警服务平台来解决此问题。

综合前两步的筛选结果，就可以进行综合考虑确定增添服务平台的数量与位置了。由于工作量不均衡没有定性的评价指标，本文引入了偏差来衡量工作量不均衡的程度，因此，在设置交巡警服务平台的时候，优先考虑偏差，进而考虑出警时间控制在三分钟以内。

在 A 区的交通网络示意图中标识出工作量负荷的交巡警服务平台位置，再标识出出警时间超过三分钟的路口节点位置。在工作量负荷与出警时间过长的两个方面，对临近标记点进行试点和博弈，可以发现，合理增加服务平台不但可以解决工作量负荷的问题，而且可以解决出警时间过长的问题，当然，也要遵循增加量最小的原则，以控制成本。

经过逐步的试点与修正，最终我们确定了在 49，30，23，86，69 号路口节点增加服务平台。在增加五个服务平台的基础上，我们再次对各平台管辖的范围进行划分，不但使全区的工作量有所下降，而且也使各平台工作量的不平衡度明显下降，均值减小，极差变小。如表 5 所示。

表 5　增加节点后各交巡警服务平台的工作量偏差

巡警点	1	2	3	4	5	6	7	8	9	10
偏差	1.42	3.02	-0.18	1.62	1.02	-2.48	0.52	0.02	3.22	-3.38
巡警点	11	12	13	14	15	16	17	18	19	20
偏差	-0.38	-0.98	1.12	-2.48	-0.18	0.02	0.32	1.12	-1.58	1.42
巡警点	49	30	23	86	69					
偏差	0.12	-2.58	1.42	-0.88	-1.28					

5.4 其他区域交巡警服务平台的合理性分析

对于其他平台，我们可以通过类似前述分析 A 区的方法进行分析。在与平均工作量偏差，以及最短到达时间上进行评价。为了节省篇幅，我们在以下的图标只列出了具有代表意义和决定性的条目。

5.4.1 B 区

工作量评价 工作量偏差表格如表 6 所示。

表 6 B 区工作量偏差 (部分)

巡警点	94	96	98
偏差	4.7	4.4	4.8

效率性评价 出警时间最长节点分配表格如表 7 所示。

表 7 B 区出警分配 (部分)

出入 B 区的路口标号	交巡警平台位置标号	到达路口的距离
153	99	44.703
152	99	33.658
123	94	33.653
122	94	32.853
124	96	32.098
151	96	31.937

在工作量评价方面，96，94，98 节点的工作量较其他的更大一些，可以适当在 96，94，98 节点附近增加少量巡查点。

可以看到，所有的节点都能在 5 分钟之内达到，除 153 节点之外也都能在 4 分钟之内达到，在效率性方面，B 区现有的警点分配较为合理。

5.4.2 C 区

工作量评价 工作量偏差表格如表 9 所示。

表 9 C 区工作量偏差 (部分)

巡警点	180	179	171	173
偏差	15.088	7.388	4.488	3.988

效率性评价 出警时间最长节点分配表格如表 10 所示。

表 10 C 区出警分配 (部分)

出入 C 区的路口标号	交巡警平台位置标号	到达路口的距离
207	178	68.605
264	166	66.223
239	173	65.686
202	177	64.609
206	178	59.605
253	171	58.259
208	178	57.102
199	175	55.091
317	181	54.752
316	180	54.07
301	180	53.583
287	178	53.313
200	177	52.011
263	166	51.223

在工作量方面，180 与 179 节点的工作量较平均水平来说极大，需要在附近或者其他位置适当增加巡查点。

在效率性方面，有 14 个节点需要最近的巡查点 5 分钟以上才能到达，因此设置巡查点不够合理，还需要进行调整。

5.4.3 D 区

工作量评价 工作量偏差表格如表 12 所示。

表 12 D 区工作量偏差 (部分)

巡警点	321	320
偏差	5.667	3.867

效率性评价 出警时间最长节点分配表格如表 13 所示。

表 13 D 区出警分配 (部分)

出入 D 区的路口标号	交巡警平台位置标号	到达路口的距离
332	328	160.628
330	328	118.456
329	328	111.073
331	328	90.974
362	323	81.069
370	320	78.085
371	320	73.613
344	323	48.407
339	328	47.244
369	321	41.85

在工作量方面，321 与 320 节点的工作量较平均水平来说较大，需要在附近或者其他位置适当增加巡查点。

在效率性方面，有 7 个节点需要最近的巡查点 5 分钟以上才能到达，因此设置巡查点不够合理，还需要进行调整。除此之外，332，330，329 节点需要 10 分钟以上才能到达，说明距离最近的巡查点太远，有必要在附近设置一个新的巡查点。

5.4.4 E 区

工作量评价 工作量偏差表格如表 15 所示。

表 15　E 区工作量偏差 (部分)

巡警点	385	382	383	379
偏差	6.24	5.84	5.64	5.14

效率性评价　出警时间最长节点分配表格如表 16 所示。

表 16　C 区出警分配 (部分)

出入 E 区的路口标号	交巡警平台位置标号	到达路口的距离
582	479	84.798
541	484	70.418
510	483	66.068
574	480	65.832
575	479	61.636
578	479	57.575
513	478	55.735
515	478	55.697
523	477	52.395
524	478	50.643

在工作量方面，385，382，383，379 节点的工作量较平均水平来说极大，需要在附近或者其他位置适当增加巡查点。

在效率性方面，有 10 个节点需要最近的巡查点 5 分钟以上才能到达，因此设置巡查点不够合理，还需要进行调整。

5.4.5 F 区

工作量评价　工作量偏差表格如表 18 所示。

表 18　F 区工作量偏差 (部分)

巡警点	477	475	478	476
偏差	8.373	5.973	5.673	3.673

效率性评价　出警时间最长节点分配表格如表 19 所示。

表 **19**　**F** 区出警分配 (部分)

出入 F 区的路口标号	交巡警平台位置标号	到达路口的距离
153	99	44.703
152	99	33.658
123	94	33.653
122	94	32.853
124	96	32.098
151	96	31.937

在工作量方面，477，475，478，379 节点的工作量较平均水平来说较大，需要在附近或者其他位置适当增加巡查点。

在效率性方面，所有节点在最近的巡查点 5 分钟之内都能到达，因此设置巡查点较为合理，在进行其他调整后进行微调即可。

5.5 对 32 节点罪犯的围堵方案

5.5.1 罪犯的可能逃逸路径

根据之前的邻接矩阵，我们能够比较方便地得到罪犯在一定时间，一定速度限制下所到达地所有路口节点。表 21 列出了罪犯在十分钟内可能到达的节点。

表 **21**　罪犯可能到达的节点

时间 / min	比之前增加的节点
1	32, 33
2	34, 7, 8, 9, 30, 31
3	35, 36, 45, 46, 47, 48
4	37, 5, 6, 237, 16
5	235, 236, 173, 15, 49, 50
6	232, 233, 234, 247, 51, 52, 53, 245, 55, 56, 59, 61
7	3, 38, 39, 231, 10, 238, 560, 244, 54, 246, 57, 58
8	230, 40, 171, 44, 242, 243, 60
9	64, 65, 2, 66, 4, 67, 228, 229, 488, 561, 241, 216, 28, 63

10	227, 68, 69, 70, 218, 43, 75, 76, 77, 172, 239, 217, 26, 29, 62

可以看到，罪犯在 5 分钟之后其可能的逃逸路径将会以较大的斜率增大，也就意味着追捕难度的提高，我们需要要尽量在 5 分钟之内将罪犯围捕。

5.5.2 围捕节点选择

由于罪犯是处于不断的逃亡的，我们的目的是将他的逃亡路线封堵。同样通过上述中提到的 Dijkstra 算法，我们可以得到从 32 节点到其他所有节点的最短路径。有理由相信，罪犯有比较大的概率使用最短路径进行逃亡。

因此我们需要尽可能围堵住概率最大的前述节点的最短路径。所得的路径如表 23 所示。

表 **23** 罪犯可能逃逸路径

节点	路径
32	32
33	$32 \rightarrow 33$
34	$32 \rightarrow 33 \rightarrow 34$
7	$32 \rightarrow 7$
8	$32 \rightarrow 33 \rightarrow 8$
9	$32 \rightarrow 33 \rightarrow 34 \rightarrow 9$
30	$32 \rightarrow 7 \rightarrow 30$
31	$32 \rightarrow 31$
35	$32 \rightarrow 33 \rightarrow 34 \rightarrow 9 \rightarrow 35$
36	$32 \rightarrow 33 \rightarrow 34 \rightarrow 9 \rightarrow 35 \rightarrow 36$
45	$32 \rightarrow 33 \rightarrow 34 \rightarrow 9 \rightarrow 35 \rightarrow 45$
46	$32 \rightarrow 33 \rightarrow 8 \rightarrow 46$
47	$32 \rightarrow 7 \rightarrow 47$
48	$32 \rightarrow 7 \rightarrow 30 \rightarrow 48$
37	$32 \rightarrow 33 \rightarrow 34 \rightarrow 9 \rightarrow 35 \rightarrow 36 \rightarrow 37$
5	$32 \rightarrow 7 \rightarrow 47 \rightarrow 5$
6	$32 \rightarrow 7 \rightarrow 47 \rightarrow 6$
237	$32 \rightarrow 7 \rightarrow 30 \rightarrow 237$
16	$32 \rightarrow 33 \rightarrow 34 \rightarrow 9 \rightarrow 35 \rightarrow 36 \rightarrow 16$

235	$32 \to 7 \to 30 \to 48 \to 235$
236	$32 \to 7 \to 30 \to 237 \to 236$
172	$32 \to 7 \to 30 \to 48 \to 235 \to 173$
15	$32 \to 31 \to 15$
49	$32 \to 7 \to 47 \to 5 \to 49$
50	$32 \to 7 \to 47 \to 5 \to 50$

经过上述分析，我们将外围围捕节点的集合设为 {37, 5, 6, 237, 16, 35, 36 }，以尽可能地围捕罪犯。

5.5.3 围捕警力分配选择

同样，利用上述封锁路口的结论和代码，可以得到对于该方案的警力分配。警力分配如表 25 所示。由于时长最长为 3 分钟，因此可以即时进行上述节点的围捕。

表 **25** 围捕罪犯全市警力调度方案

交巡警平台位置标号	围捕路口标号	到达路口的距离
7	37	30.414
8	16	26.923
173	237	11.325
16	36	6.083
9	35	4.243
5	5	0.000
6	6	0.000

但实际上，由于我们这里只取了相当少的一部分警力进行罪犯的抓捕。如果我们能够实时调用更多的警力，在前述分配好的情况下进行复分配，并且结合警车移动位置进行实时的规划，以使得警车的移动距离最短，这样就可以达到围捕效果的最大化。[4] 在此基础上，我们就能继续得到完整的围捕方案。

六、 模型的评价和改进

6.1 模型的评价

6.1.1 问题一模型的评价

针对问题一，利用图论中 Dijkstra 算法和运筹学中的指派模型可以很好地解决指定地区交巡警服务平台管辖范围的合理分配问题，得到了总移动时间最小原则下的最优分配方案，为管辖范围的合理分配提供了一个参考，同时也为进一步研究交巡警服务平台的合理设置问题做了准备。

6.1.2 问题四模型的评价

针对问题四，我们对其他几个区的平台分别从能否在三分钟以内到达节点以及各工作平台工作负担两个维度进行评价，即效率性评价以及工作量评价，以此为依据说明现有平台依然存在着缺陷。运用类似于问题三的调整方法，最终给出了部分调整的方案，使得尽可能覆盖所有节点并且工作负担分配相对均衡。

6.1.3 问题五模型的评价

针对问题五，我们利用图论和整数规划的理论建立了一种在市区内围捕罪犯的方法，通过了分层围堵的方案，在节省警力，争取的时间的原则下，对嫌疑人的逃跑路径进行分析，得出了合理的警力调用围堵方案，快速围堵犯罪嫌疑人提供了一种有效的方法。

6.2 模型的改进

本文中的模型也有一定的局限性，如现实中不能时刻保证道路的时刻畅通，也不能保证出警时间和罪犯逃逸的速度，同时犯罪发生地点也是随机的。本文中忽略了现实中的一些不定因素，问题四中忽略了人口密度对平台设置可能带来的影响，忽略了距离较近的平台可能带来的资源重复浪费，问题五中忽略了嫌疑犯可能躲藏的情况。

七、 参考文献

[1] 于晶贤，李金秋，田秋菊. 交巡警服务平台管辖范围的合理分配研究 [J]. 第 34 期，2011 年 12 月.

[2] 姜启源，谢金星，叶俊. 数学模型 [M]. 高等教育出版社，2008.

[3] 严蔚敏，吴伟民. 数据结构 [M]. 清华大学出版社，2006 年 12 月.

[4] 董金哲，罪犯围捕中的数学模型 [J]. 第 18 期，2012 年.

附录 A

1.1 apps.py

```
# Frequently used functions

import os
import platform
from xlrd import open_workbook
from xlrd import XL_CELL_TEXT, XL_CELL_NUMBER, XL_CELL_DATE, XL_CELL_BOOLEAN
import numpy as np
import math

INFINITY = -1
UNDIFINED = -2


def sheet_to_array(
      filename,
      sheet_number,
      first_col=0,
      last_col=None,
      header=True):
   """Return a floating-point numpy array from sheet in an Excel spreadsheet.

   Notes:
   0. The array is empty by default; and any non-numeric data in the sheet will
     be skipped.
   1. If first_col is 0 and last_col is None, then all columns will be used,
   2. If header is True, only one header row is assumed.
   3. All rows are loaded.
   """
   DEBUG = False
   # sheet
   book = open_workbook(filename)
   sheet0 = book.sheet_by_index(sheet_number)
   rows = sheet0.nrows
   # cols
   if not last_col:
      last_col = sheet0.ncols
   if first_col > last_col:
      raise Exception("First column must be smaller than last column!")
   cols = [col for col in range(first_col, last_col + 1)]
   # rows
   skip = 0
   if header:
```

```
      skip = 1
   data = np.empty([len(cols), rows - skip])

   for row in range(skip, sheet0.nrows):
      row_values = sheet0.row(row)
      for col, cell in enumerate(row_values):
         if DEBUG and row < 2:
            print(row, col, cell.ctype, cell.value, '\n')
         if col in cols and cell.ctype == XL_CELL_NUMBER:
            data[col - first_col, row - skip] = cell.value
   return data


def get_row_index(x):
   return {
      'A': [0, 92],
      'B': [92, 165],
      'C': [165, 319],
      'D': [319, 371],
      'E': [371, 474],
      'F': [474, 582],
      'ALL': [0, 582]
   }[x]

def get_police_index(x):
   return {
      'A': [0, 20],
      'B': [20, 28],
      'C': [28, 45],
      'D': [45, 54],
      'E': [54, 69],
      'F': [69, 80],
      'ALL': [0, 80]
   }[x]

def map_region(data, lines, name):
   row_index = get_row_index(name)
   nodes = data[0:3, row_index[0]:row_index[1]]
   j = row_index[1] - row_index[0]
   mat = np.ones([j, j]) * INFINITY

   for i in range(mat.shape[1]):
      mat[i, i] = 0
   for i in range(lines.shape[1]):
      if ((lines[0, i] >= nodes[0, 0]) and (lines[0, i] <= nodes[0, j - 1])
            and (lines[1, i] >= nodes[0, 0]) and (lines[1, i] <= nodes[0, j - 1])):
         # print(lines[0, i], lines[1, i])
```

```
            # print(lines[0, i] - row_index[0] - 1,
            #     lines[1, i] - row_index[0] - 1)
            # print((data[1, int(lines[0, i] - 1)], data[1, int(lines[1, i] - 1)],
            #     data[2, int(lines[0, i] - 1)], data[2, int(lines[1, i] - 1)]))
            distance = math.hypot(data[1, (lines[0, i] - 1)] - data[1, (
                lines[1, i] - 1)], data[2, (lines[0, i] - 1)] - data[2, (lines[1, i] -
                    1)])
            # print(distance)
            mat[(lines[0, i] - row_index[0] - 1),
                (lines[1, i] - row_index[0] - 1)] = distance
            mat[(lines[1, i] - row_index[0] - 1),
                (lines[0, i] - row_index[0] - 1)] = distance
    return mat


def dijkstra(mat, node):
    # intial
    size = mat.shape[0]
    dist = np.ones(size) * INFINITY
    prev = np.ones(size) * UNDIFINED
    dist[node] = 0

    vertex_set = set(np.arange(0, size))
    # vertex_set.discard('')

    # dijkstra
    while len(vertex_set):
        for i in range(size):
            if (i in vertex_set and dist[i] >= 0):
                u = i
                break
        for i in vertex_set:
            if (dist[i] < dist[u] and dist[i] != -1):
                u = i
        vertex_set.discard(u)

        for v in range(size):
            if (mat[v, u] >= 0):
                alt = dist[u] + mat[v, u]
                if (alt < dist[v] or dist[v] == INFINITY):
                    dist[v] = alt
                    prev[v] = u

    # return result
    return dist, prev

def police_distance(data, mat, name, police=None):
```

```
    police_row = get_police_index(name)
    row = get_row_index(name)
    offset = row[0] + 1

    # get police indexes
    if (police == None):
        police = data[1, police_row[0]:police_row[1]]
        police = police.astype(int)
    else:
        tmp = data[1, police_row[0]:police_row[1]]
        tmp = set(tmp.astype(int))
        police = tmp.union(police)

    dis_arr = []
    pre_arr = []

    # dijastra
    for i in police:
        dist, prev = dijkstra(mat, i - offset)
        dis_arr.append(dist)
        pre_arr.append(prev)
    return police, dis_arr, pre_arr, offset

def police_inout(data, mat, input, name):
    police_row = get_police_index(name)
    row = get_row_index(name)
    offset = row[0] +1

    # get police indexes
    police = data[1, police_row[0]:police_row[1]]
    police = police.astype(int)

    dis_arr = []
    pre_arr = []
    for i in police:
        dist, prev = dijkstra(mat, i - offset)
        dis_arr.append(dist)
        pre_arr.append(prev)
    return police, dis_arr, pre_arr, offset

def near_index(dist, offset, i):
    arr = []
    for item in dist:
        arr.append(item[i])
    return arr.index(min(arr))

def suit_index(suited, i, offset):
```

```
    k = 0
    result = []
    for item in suited:
        if (i+offset) in item:
            result.append(k)
        k += 1
    return result

def myabs(num):
    return num if num >= 0 else -1 * num

def mymax(arr):
    max_tmp = arr[0]
    for i in arr:
        if (i >= max_tmp):
            max_tmp = i
    return max_tmp
```

1.2 init.py

```
import apps

nodes = apps.sheet_to_array(
    './attachment_02.xls',
    0,
    first_col=0,
    last_col=None,
    header=True)
lines = apps.sheet_to_array(
    './attachment_02.xls',
    1,
    first_col=0,
    last_col=1,
    header=True)
lines = lines.astype(int)
police = apps.sheet_to_array(
    './attachment_02.xls',
    2,
    first_col=0,
    last_col=None,
    header=True)
inoutCity = apps.sheet_to_array(
    './attachment_02.xls',
    3,
    first_col=1,
```

```
   last_col=1,
   header=True)

inoutA = apps.sheet_to_array(
   './attachment_02.xls',
   3,
   first_col=2,
   last_col=2,
   header=True)

base = apps.sheet_to_array(
   './attachment_02.xls',
   4,
   first_col=0,
   last_col=None,
   header=True)
```

1.3 source_01.py

```
# To get the suited vertices and matrix for further use

from init import *
from apps import *
import numpy


STD_DIS = 3 / 60 * 60 * 1000 / 100

# define const name, A, B, C, D, E, F or All
name = 'D'

# get the adjacency matrix
mat = map_region(nodes, lines, name)

# get the shortest distance array of every poliction station to every node
police, dist, prev , offset = police_distance(police, mat, name, )

V = len(mat)
P = len(police)

# find who suits the case
j = 0
suited = []
for i in dist:
 k = 0
```

```
  node_set = set()
  for dis in i:
    if (dis <= STD_DIS):
      node_set.add(k+offset)
    k=k+1
  suited.append(node_set)
  j = j+1

rows = police.shape[0] + 1
clos = dist[0].shape[0] + 1
output = numpy.zeros([rows, clos])
for i in range(rows):
  if (i >= 1):
    output[i, 1:] = dist[i-1]
node_index = numpy.arange(1, 583)
ran = get_row_index(name)
output[0, 1:] = node_index[ran[0]:ran[1]]
output[1:, 0] = police.astype(int)

numpy.savetxt("output/" + name + ".csv", output, delimiter=",")

# Type 1
set_01 = set(police)

# Type 2
node_set = set()
for item in suited:
  node_set = node_set.union(item)
set_02 = node_set - set(police)

# Type 3
set_03 = set(range(offset, offset + V)) - node_set

rate = nodes[4,:]
```

1.4 source_02.py

```
from init import *
from apps import *

# define const name to 'A', for we only consider the rigion A
name = 'ALL'

# # get the adjacency matrix
mat = map_region(nodes, lines, name)
```

```
# # get the shortest distance array of every poliction station to every node
police, dist, prev , offset = police_distance(police, mat, name)

# format and validation
inoutA = inoutA[0].astype(int)
inoutCity = inoutCity[0].astype(int)
tmp = []
for i in inoutA:
  if (i > 0):
    tmp.append(i)
inoutA = list(tmp)

tmp = []
for i in inoutCity:
  if (i != 0):
    tmp.append(i)
inoutCity = tmp

inoutCus = [37, 5, 6, 237, 16, 35, 36]


inout = inoutCus

I = len(inout)
P = len(police)
V = len(mat)
```

1.5 source_03.py

```
# 0-1 Programming

from pulp import *
import numpy as np
from source_01 import *
import csv
# from source_02 import *

X_PRE = 'x_'

# 设置对象
model = LpProblem('model', LpMinimize)
X = np.empty([V, P]).astype(LpVariable)

keys = []
```

```
for i in range(V):
  for j in range(P):
    key = X_PRE + str(i) + '_' + str(j)
    X[i][j] = LpVariable(key, cat='Binary')
    keys.append(key)

# 目标函数
model += sum((sum((X[i][j] * dist[j][i]) for i in range(V))) for j in range(P)),
    'profit'


# 载入约束变量
for i in range(V):
    flag = 1
    for j in range(P):
        if (i+offset in set_01):
            model += X[i][j] == (i == j)
        elif (i+offset in set_02):
            if (flag):
                set_tmp = set(suit_index(suited, i, offset))
                model += sum(X[i][j] for j in set_tmp) == 1
                model += sum(X[i][j] for j in set(range(P)) - set_tmp) == 0
                flag = 0
        elif (i+offset in set_03):
            model += X[i][j] == (j == near_index(dist, offset, i))

# Solve
GLPK().solve(model)

keys = set()
rate_average = sum(rate[offset-1:V+offset-1]) / P
jurisdiction = [set() for _ in range(P)]

each_node = []
for i in model.variables():
    if i.varValue:
        arr = i.name.split('_')
        i ,j = int(arr[1]), int(arr[2])
        jurisdiction[j].add(i+offset)
        each_node.append([i + offset, j+offset, dist[j][i]])

each_node.sort(key=lambda x: x[2], reverse=True)

k = offset
su = 0
each_police = []
for i in jurisdiction:
```

```
    diff = (sum(rate[k-1] for k in i) - rate_average)
    each_police.append([k, diff])
    k+= 1
    su += diff

each_police.sort(key=lambda x: x[1], reverse=True)


# Output

text_01 = []
text_02 = []

for i in each_node:
    text_01.append(('Node', i[0],'Police', i[1], 'Distance', "%.3f" % i[2]))

for i in each_police:
    text_02.append(("Police", i[0] , 'Diff', "%+.3f" % i[1] , 'Ave', rate_average))


text_02

with open('output/nodes/' + name + '_nodes.csv','w') as f:
    f_csv = csv.writer(f)
    f_csv.writerows(text_01)
    f_csv.writerows(text_02)
    f_csv.writerow(('Sum Distance', value(model.objective)))
```

1.6 source_04.py

```
# 0-1 Programming for another

from pulp import *
import numpy as np
from source_02 import *

X_PRE = 'x_'
MAX = 40

# Set objects
model = LpProblem('model', LpMinimize)
X = np.empty([I, P]).astype(LpVariable)

keys = []
for i in range(I):
```

```
  for j in range(P):
    key = X_PRE + str(i) + '_' + str(j)
    X[i][j] = LpVariable(key, cat='Binary')
    keys.append(key)

# Load the constrains
for j in range(P):
  model += sum(X[i][j] for i in range(I)) <= 1

for i in range(I):
  model += sum(X[i][j] for j in range(P)) == 1



# Find max
for j in list(range(P)):
  for i in list(range(I)):
    if (dist[j][inout[i] - offset]):
      model += X[i][j] * dist[j][inout[i] - offset]<= MAX

# Solve
GLPK().solve(model)

# Show the result
result = []
for i in model.variables():
   if i.varValue:
      arr = i.name.split('_')
      i ,j = int(arr[1]), int(arr[2])
      result.append([police[j], inout[i], value(X[i][j] * dist[j][inout[i] -
          offset])])


#Sort the result and out put
result.sort(key=lambda x: x[2], reverse=True)
for i in result:
  print(i[0],'&', i[1], '&', "%.3f" % i[2], '\\\\')
print("Max Distance: ", "%.3f" % result[0][2])
```

1.7 source_05.py

```
# from source_01 import *
from apps import *
from init import *
```

```
THEFT = 32
NAME = 'ALL'
STD_DIS = 10

mat = map_region(nodes, lines, NAME)
offset = get_row_index(NAME)[0]+1

dist, prev = dijkstra(mat, THEFT - offset)

time_dis = list(range(1,6))

all_where = []

prev_where = set()
for k in time_dis:
 now_where = set()
 for i,n in enumerate(dist):
  if (n < k * STD_DIS):
   now_where.add(i+offset)
 now_where = now_where - prev_where
 prev_where = now_where.union(prev_where)
 all_where.append(now_where)

for i,n in enumerate(all_where):
 print(i+1, ' & ' ,n)
 for i in n:
  m = prev[i-offset]
  arr = [i]
  string = str(i)
  while(m != -2):
   arr.append(int(m+offset))
   m = prev[int(m)]
  arr.reverse()
  # string = '$'
  # for i in range(len(arr)-1):
  #  string += str(arr[i]) + '\t\\to\t'
  # string += str(arr[len(arr)-1]) + '\t$'
  # print(string)
```

1.8 source_06.py

```
# 0-1 Programming for another

from pulp import *
```

```
from apps import *
from init import *
import numpy as np

THEFT = 32
NAME = 'A'
STD_DIS = 10
X_PRE = 'x_'
MAX = 82

where_to_push = [1, 2, 3, 4, 5, 6, 7]

mat = map_region(nodes, lines, NAME)

# get the shortest distance array of every poliction station to every node
police, dist, prev , offset = police_distance(police, mat, NAME)

V = len(mat)
P = len(police)
I = len(where_to_push)

# Set objects
model = LpProblem('model', LpMinimize)
X = np.empty([I, P]).astype(LpVariable)


keys = []
for i in range(I):
 for j in range(P):
  key = X_PRE + str(i) + '_' + str(j)
  X[i][j] = LpVariable(key, cat='Binary')
  keys.append(key)

# 载入约束变量
for j in range(P):
 model += sum(X[i][j] for i in range(I)) <= 1

for i in range(I):
 model += sum(X[i][j] for j in range(P)) == 1


# Find max
for j in list(range(P)):
 for i in list(range(I)):
  if (dist[j][where_to_push[i] - offset]):
   model += X[i][j] * dist[j][where_to_push[i] - offset]<= MAX
```

```
# Solve
print(model)
GLPK().solve(model)

# 显示结果

for i in model.variables():
   if i.varValue:
      arr = i.name.split('_')
      i ,j = int(arr[1]), int(arr[2])
      print('Police: ', police[j],'\tInout: ', where_to_push[i], '\tDistance: ',
          value(X[i][j] * dist[j][where_to_push[i] - offset]))
```

1.9 source_07.py

```
# 0-1 Programming

from pulp import *
import numpy as np
import csv
from init import *
from apps import *
# from source_02 import *

X_PRE = 'x_'
STD_DIS = 3 / 60 * 60 * 1000 / 100

name = 'A'

# get the adjacency matrix
mat = map_region(nodes, lines, name)

# get the shortest distance array of every poliction station to every node

police, dist, prev , offset = police_distance(police, mat, name, police=set([49
    ,30 ,23 ,86 ,69]))

police = list(police)

V = len(mat)
P = len(police)

j = 0
suited = []
for i in dist:
```

```
  k = 0
  node_set = set()
  for dis in i:
    if (dis <= STD_DIS):
      node_set.add(k+offset)
    k=k+1
  suited.append(node_set)
  j = j+1

# Type 1
set_01 = set(police)

# Type 2
node_set = set()
for item in suited:
  node_set = node_set.union(item)
set_02 = node_set - set(police)

# Type 3
set_03 = set(range(offset, offset + V)) - node_set

rate = nodes[4,:]

X_PRE = 'x_'

# 设置对象
model = LpProblem('model', LpMinimize)
X = np.empty([V, P]).astype(LpVariable)

keys = []
for i in range(V):
  for j in range(P):
    key = X_PRE + str(i) + '_' + str(j)
    X[i][j] = LpVariable(key, cat='Binary')
    keys.append(key)


# 目标函数
model += sum((sum((X[i][j] * dist[j][i]) for i in range(V))) for j in range(P)),
    'profit'


# 载入约束变量
for i in range(V):
    flag = 1
    for j in range(P):
       if (i+offset in set_01):
```

```
            model += X[i][j] == (i == police[j]-offset)
        elif (i+offset in set_02):
            if (flag):
                set_tmp = set(suit_index(suited, i, offset))
                model += sum(X[i][j] for j in set_tmp) == 1
                model += sum(X[i][j] for j in set(range(P)) - set_tmp) == 0
                flag = 0
        elif (i+offset in set_03):
            model += X[i][j] == (j == near_index(dist, offset, i))

# Solve
GLPK().solve(model)

keys = set()
rate_average = sum(rate[offset-1:V+offset-1]) / P
jurisdiction = [set() for _ in range(P)]

each_node = []
for i in model.variables():
    if i.varValue:
        arr = i.name.split('_')
        i ,j = int(arr[1]), int(arr[2])
        jurisdiction[j].add(i+offset)
        each_node.append([i + offset, j+offset, dist[j][i]])

each_node.sort(key=lambda x: x[2], reverse=True)

k = offset
su = 0
each_police = []
for i in jurisdiction:
    diff = (sum(rate[k-1] for k in i) - rate_average)
    each_police.append([k, diff])
    k+= 1
    su += diff

each_police.sort(key=lambda x: x[1], reverse=True)

text_01 = []
text_02 = []

for i in each_node:
    text_01.append(('Node', i[0],'Police', i[1], 'Distance', "%.3f" % i[2]))

for i in each_police:
    text_02.append(("Police", i[0] , 'Diff', "%+.3f" % i[1] , 'Ave', rate_average))
```

```
# print()

text_02

with open('output/added/' + name + '_nodes.csv','w') as f:
    f_csv = csv.writer(f)
    # f_csv.writerow(headers)
    f_csv.writerows(text_01)
    f_csv.writerows(text_02)
    f_csv.writerow(('Sum Distance', value(model.objective)))
```