交通警力责任区分配问题

摘要

在本文中，我们首先依据交通划分存在的约束条件，建立了描述警力分配情况的数学规划模型，并且通过深度优先搜索方法得到了模型的精确解。在多划分的情况下，我们对比了在不同情况下采取贪心策略的搜索算法和启发式算法的求解精度，确定了两种算法的适用范围，并且把两种算法在考虑概率分布的实际情况下进行了合理的应用。

对于问题 1，我们通过分析辖区交通划分的实际情况，确定了点集连通等约束条件，同时把划分的各个集合内部元素之和的方差作为目标函数来表示警情总量的均衡程度，建立了使目标函数最小的划分数学规划模型。并使用深度优先搜索的方法得到了模型在划分数 $k = 2$ 时的部分精确解。

对于问题 2，我们可以按照类似问题 1 的方法划分求解，但由于我们在问题 1 中的求解方法只能保证进行第一次划分时的局部最优，在需要进行多划分的情景中按照贪心策略使用此算法有一定的局限性，无法获得全局最优解。考虑到图划分问题是一个 NP 问题，直接求解问题的计算规模较大，我们同时使用了启发式算法来求得近似最优解，并对两种算法的效率和精度进行了对比。

对于问题 3，我们认为问题中的数据信息服从特定的统计分布，且只考虑对应分布发生概率在 90% 以上的部分，这会在一定程度上影响我们的在问题 1 和 2 中使用的各个交通事故易发点数据的数学期望。由于事故点发生事故可以认为是一次伯努利实验的结果，这种情况下可以认为发生事故的次数近似服从泊松分布。我们根据各个顶点的已知数据对泊松分布的参数 λ 进行估计。利用得到的泊松分布模型进行对警情数量进行区间估计。得到了对应各个事故点发生事故次数的数学期望。并利用修正后的数学期望按照问题 1 和 2 的方法计算划分情况。

我们的模型综合了能求得局部精确解的搜索模型以及时间复杂度较低，能求得近似解的启发式模型。并在文中分析了各种模型的优缺点和适用范围。在实际的图的划分问题中有一定的应用价值。

关键字：图的 k 划分　启发式算法　泊松分布　深度优先搜索

一、 问题重述

随着道路交通的快速发展，交通勤务的科学化、精细化管理，是交通管理部门的重视。某交警中队根据其辖区交通实际状况，把负责的辖区划分为若干责任区，每个责任区都委派一名交警进行交通事故处理，每名交警仅负责其责任区内的所有交通事故的警情处理。为均衡每名交警的工作量，交警中队管理部门需要把该辖区划分为若干警情总量尽可能均衡的责任区。

1. 假设该辖区内共有若干处交通事故易发点，我们已知每一处交通事故易发点的年均警情量。若交警中队管理部门拟把该辖区划分为 2 个年均警情量尽可能均衡的责任区，建立此问题的数学模型并给出高效的求解算法。
2. 在问题 1 的基础上，若交警中队管理部门拟把该辖区划分为 k 个年均警情量尽可能均衡的责任区，建立此问题的一般数学模型并给出高效的求解算法。
3. 交通事故的发生都具有一定的随机性，假设每个交通事故易发点年均警情量的发生概率都不低于 90%。若交警中队管理部门拟把该辖区划分为 k 个年均警情量尽可能均衡的责任区，建立此问题的一般数学模型并给出高效的求解算法。

二、 问题分析

对于问题 1，我们可以分析辖区交通划分的实际情况，确定“集合的发生事故数之和与各个集合发生事故数的均值的偏差最小”，“划分后的图必须连通”等限制关系，得到对应的约束条件，同时把划分集合元素和的方差作为目标函数，建立使目标函数最小的划分数学规划模型。可以使用深度优先搜索的方法求精确解。

对于问题 2，由于我们在问题 1 中的求解方法只能保证局部最优，在需要进行多划分的情景可以按照贪心策略，在每次划分时试图使划分部分各个点的权重之和接近方差最小情况对应的均值最优解，但由于在以局部最优进行划分时没有考虑到全局情况，使用此算法不一定能得到全局最优。另外，考虑到图划分问题是一个 NP 问题，直接搜索所有可行情况求解问题的计算规模较大，我们可以考虑使用启发式算法来求得近似最优解。同时可以把两种求解结果进行对比以确定算法的使用范围。

对于问题 3，问题中的数据信息服从特定的分布，且只考虑对应分布发生概率在 90% 以上的部分，这会在一定程度上改变我们的在问题 1 和 2 中使用的各个交通事故易发点数据的数学期望。我们参考文献 [2] 的数据，采用负二项分布模拟各个事故点交通事故的发生情况，使用重新获得的数学期望进行求解。

三、 模型假设及符号说明

3.1 模型假设

1. 每条道路对交通事故的影响作用相同，即认为图中各条边的边权相等
2. 忽略其他外界因素对交通事故发生的影响
3. 交通事故的发生量服从泊松分布

3.2 符号说明

符号	意义
V	交通事故易发点组成的点集
G	反映各个交通事故点和它们之间的连接情况的图
$W(v_i)$	v_i 的年均警情量
C_A	划分后点集对应的连通矩阵
$E(V)$	负责区年均警情量总和的数学期望
$D(V)$	负责区年均警情量总和的方差

四、 模型准备

我们可以把辖区中的各个交通事故易发点用点集 $V=(v_1,v_2,\cdots,v_n)$ 表示，辖区内各个交通事故点和它们之间的连接情况可以用图 $G=(V,E)$ 来表示。每个点的年均警情量可以用 $W(v_i),i=1,2,\cdots,n$ 表示，它对应图中每个点的点权。

则 G 对应的邻接矩阵 C 为:

$$C=\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix}$$

其中

$$c_{pq}=\begin{cases} 1(v_p \text{ 与 } v_q \text{ 邻接}) \\ 0(v_p \text{ 与 } v_q \text{ 不邻接}) \end{cases}$$

将该辖区分成 k 个划分 $V_1,V_2,\cdots,V_k$，每个划分对应的权值为

$$w(V_i) = \sum_{v \in V_i} w(v), i = 1, 2, \cdots, k$$

划分后的点集对应的邻接矩阵 C' 为

$$C' = \begin{bmatrix} c'_{11} & c'_{12} & \cdots & c'_{1n} \\ c'_{21} & c'_{22} & \cdots & c'_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c'_{n1} & c'_{n2} & \cdots & c'_{nn} \end{bmatrix}$$

$$c'_{pq} = \begin{cases} 1, (c_{pq} = 1) \wedge \exists i (v_p \in V_i \wedge v_q \in V_i) \\ 0, \text{其他} \end{cases}, i \in 1, 2, \cdots, k$$

令

$$C'^0 + C'^1 + C'^2 + \cdots + C'^{n-1} = \begin{bmatrix} con_{11} & con_{12} & \cdots & con_{1n} \\ con_{21} & con_{22} & \cdots & con_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ con_{n1} & con_{n2} & \cdots & con_{nn} \end{bmatrix}$$

则划分后的点集对应的连通矩阵 C_A 满足：

$$C_{Aij} = \begin{cases} 1, con_{ij} \neq 0 \\ 0, \text{其他} \end{cases}$$

划分后，每个划分 V_n 中由点集 V 的部分点组成。各个点集对对应的点权之和为 $\sum W(V_n)$。所有责任曲警情量的平均值 $E(V)$ 可表示为：

$$E(V) = \frac{1}{k} \sum_{j=1}^{k} W(V_j) \tag{1}$$

五、 问题一

5.1 模型建立

根据题目，我们的目标是使负责区年均警情量总和的方差 $D(V)$ 最小，即

$$\min D(V) = \frac{1}{k} \sum_{i=1}^{k} (W(V_i) - E(V))^2 \tag{2}$$

我们发现在划分辖区时，需要考虑子集间不连通约束和子集内连通约束。

（1）子集间不连通约束

在模型准备过程中，通过将连接两个不同子集的边在点集 G 的邻接矩阵中对应的项置为 0，保证了划分后的任意子集间不会相连，因此子集间不连通约束始终成立，无须单独写出。

（2）子集内连通约束

对负责区的划分应保证每个责任区是联通的，因此，划分后辖区的连通矩阵 C_A 需满足对划分得到的任意一个子集 V_i 中的任意两个不相同的点 v_p, v_q，都有 $C_{Apq} = C_{Aqp} = 1$，从而得到子集内连通性约束：

$$\forall i \in \{1, 2, \cdots, k\} \forall v_p, v_q \in V_i, C_{Apq} = C_{Aqp} = 1, (p \neq q) \tag{3}$$

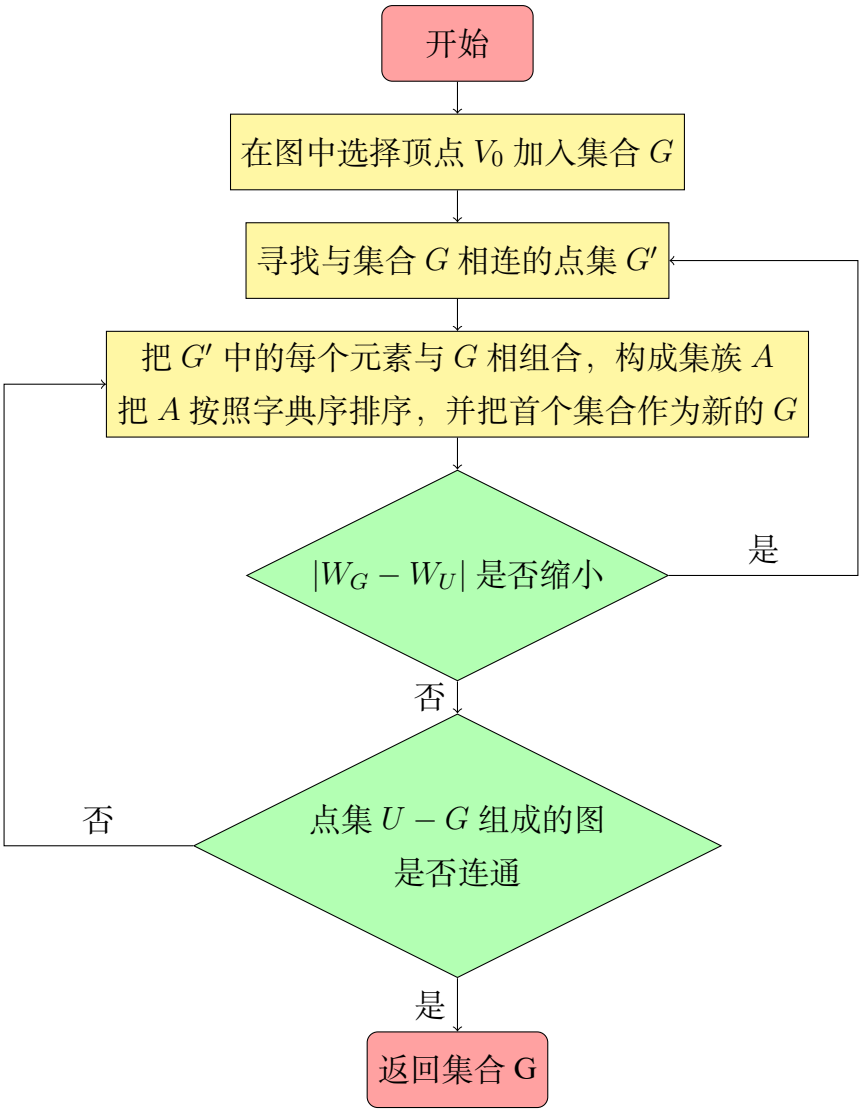
在 $k = 2$ 时，可简化为划分的每个点集的发生事故数之和 $\sum W(V_i)$ 与各个集合发生事故数的均值 $E(V)$ 的差的绝对值最小，在这种情况下我们可以建立如下的数学规划模型：

$$\min |W(V_i) - E(V)| \tag{4}$$

$$s.t. \forall n \in \{1, 2, \cdots, k\} \forall i, j \in V_n, C_{Aij} = C_{Aji} = 1 (i \neq j) \tag{5}$$

我们可以从图中的某一个点出发，使用搜素方法按照一定顺序对相连的点集进行遍历，选择能够让 $|\sum W(V_n) - \frac{1}{k} \sum W(V)|$ 缩小的点加入集合，从而穷举得到能够达到局部最优的点集。

如果用 W_G 代表某个点集 G 内所有点的权重之和，用 U 代表全集，上述算法可以用流程图表示为：



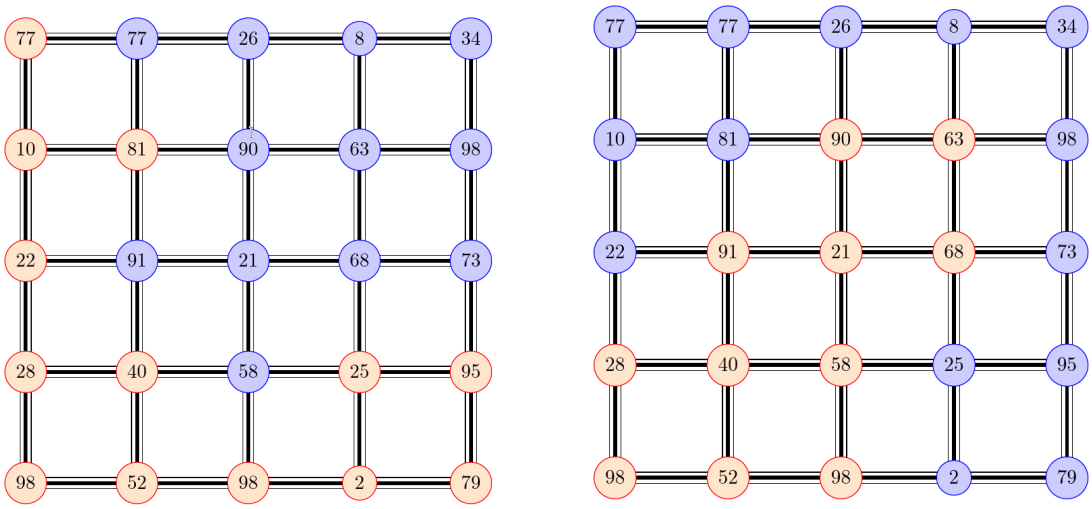
5.2 模型求解

我们采取了深度优先搜索的思路，在每次添加元素后按照字典序进行排序，这样可以使得我们的搜索朝着一个特定的方向进行，以尽快的获得结果。

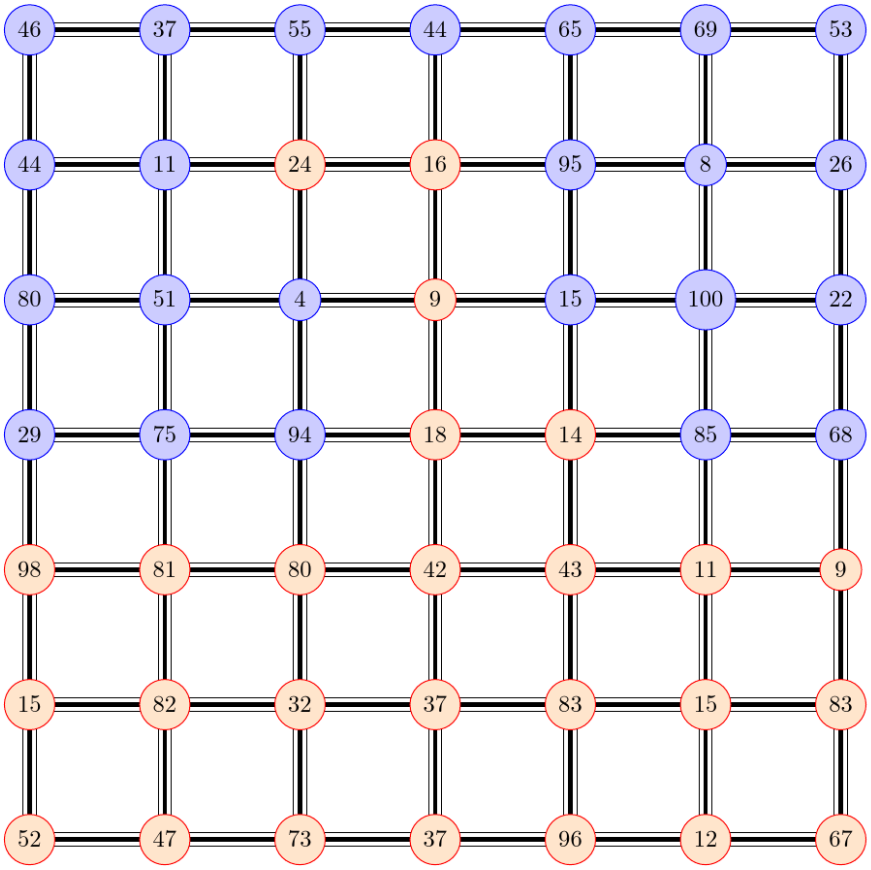
为了在搜索中排除部分不可行的情况，提高搜索效率，我们采取了剪枝策略，并在搜索中排除了以下情况：

1. 加入新的点的时候判断 $|\sum W(V_n) - \frac{1}{k}\sum W(V)|$ 是否减小，排除了会导致偏差增大的情况
2. 判断点集是否连通，从而排除了部分会导致划分的点集不连通的情况
3. 在获得第一组结果后停止搜索

对于数据 1，使用 C++ 编程求解得到了 18 组最优解（求解详情请见附录），其中两种最优解的情况如下：



对于数据 2，由于数据规模的进一步增大，数据间的组合方式也快速增加，我们求解发现在在数据 2 上进行均衡 2 划分时有超过 1000 组最优解。由于计算资源限制，我们只求解了部分最优解，其中一组最优解的分组情况如下图所示：



六、 问题二

6.1 模型建立

我们可以使用与问题 1 相似的数学规划方法得到年均警情量总和均衡的辖区 k 划分模型：

$$\min \quad D(V) = \frac{1}{k}\sum_{i=1}^{k}(w(V_i) - E(V))^2, \tag{6}$$

$$\text{s.t.} \quad \forall i \in \{1, 2, \cdots, k\} \forall v_p, v_q \in V_i, C_{Apq} = C_{Aqp} = 1, (p \neq q) \tag{7}$$

这一模型给出了我们在 $k > 2$ 时的优化目标和约束条件。

6.2 模型求解

对该规划问题的求解，若直接采取贪心策略，逐步寻找局部最优，则最终得到的解并非是全局最优的，甚至可能不是一个较优解，故采取启发式算法 PART[] 并加以改进，实现该辖区的划分，实现步骤如下：

（1）辖区对应图的预处理

根据启发式算法的思路，在保证每个子集的点权值和均衡的情况下，可通过最小化割边权值的方式实现每个子集内部边权和最大，当所有边权均为某一常数时，最终求出的解在点权和均衡和顶点数均衡上都是较优的，故可将每一条边的权值赋为 1，得到 $n = 25$ 和 $n = 49$ 时对应的带权图：

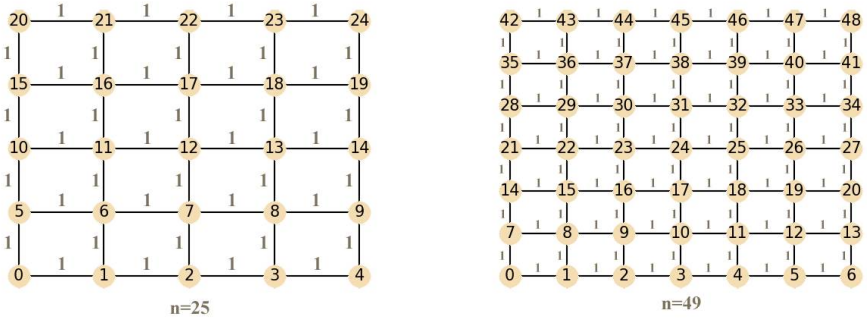


图 1　辖区带权图

（2）已选点集扩展和候选点集更新

在得到的带权图中随机选取一个点作为起始点，加入到已选点集 S 中，将与已选点集中所有点相邻的未被选过的点加入到候选点集 $Cand$ 中。

设所有未选点组成的集合为 $USel$，已选点集 S 中与候选点 v 邻接的点的边权之和为 Sum_{SC}，未选点集 $USel$ 中与候选点 v 邻接的点的边权之和为 Sum_{UC}，则有增益

$$gain(v) = Sum_{SC} - Sum_{UC} \tag{8}$$

在候选点集中找到具有最大增益 $gain$ 的点，加入到已选点集 S 中。

重复执行候选点的选取操作，直至已选点集 S 的点权之和与 $E(V)$ 之差的绝对值最小，然后判断未选点集的连通性，若连通，则将当前已选点集作为划分得到的子集，反之，则清空已选点集和候选点集，重新开始这一子集的构造。

(3) 已选点集族生成

对辖区带权图重复 k 次已选点集扩展和候选点集更新操作，得到已选点集的集族，即年均警情量总和均衡的辖区 k 划分的较优解对应的划分如下图所示：

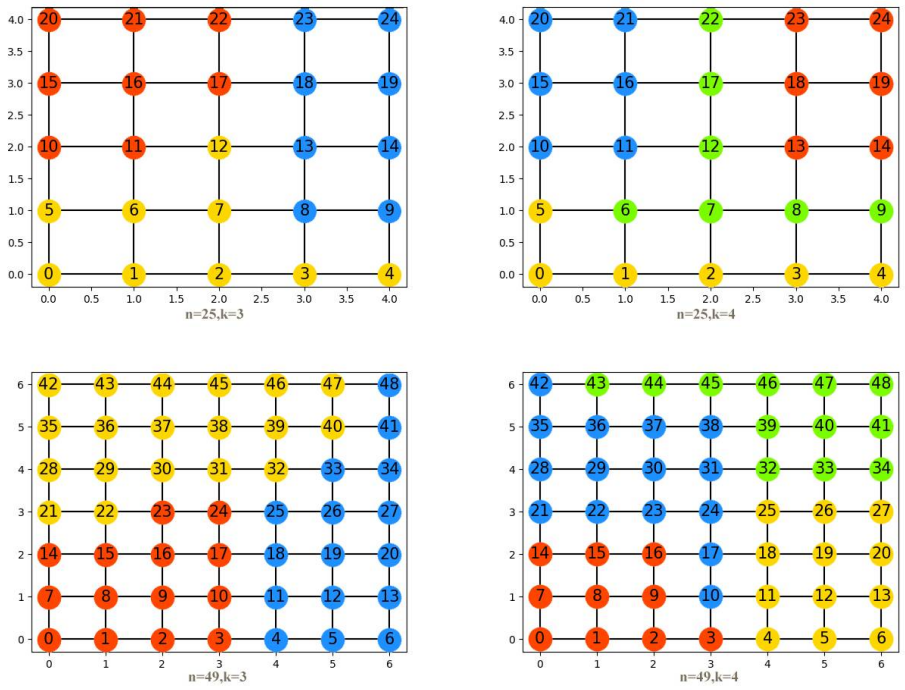


图 2　年均警情量总和均衡的辖区 k 划分的较优解

每组解对应的方差 $D(V)$ 分别为：

	k=3	k=4
n=25	27.555587	31.250000
n=49	24.666667	37.500000

6.3 启发式算法与基于贪心策略的搜索算法的对比

根据上面的求解结果我们可以发现，在绝大多数情况下，启发式算法的求解情况都明显优于基于贪心策略的搜索算法。但在划分组数较少，划分范围较小时 (例如数据 1 在 k=3 时的情况)，基于贪心策略的搜索模型的求解结果与启发式算法的误差接近，但基于贪心策略的搜索模型的时间复杂度明显更高。这是因为在数据较小时，贪心算法缺少对全局的考虑的问题被缩小了。

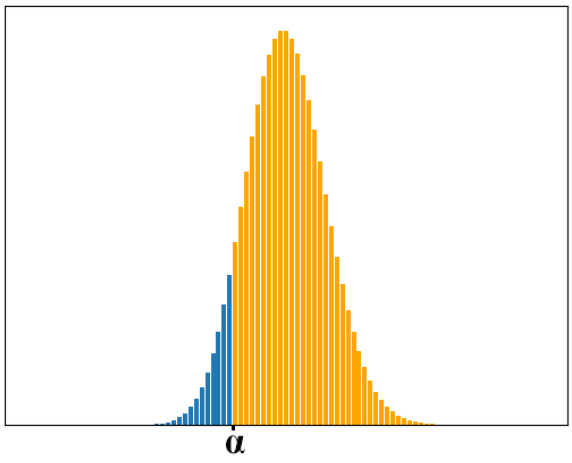
另外，由于贪心策略具有“局部最优”的特性，在某些只对于某一个辖区的划分均衡性要求较高，而对其他辖区的划分的均衡性要求不强的情况下有一定优势。

七、 问题三

7.1 模型建立

考虑到在实际情况下年均警情量并不是一个定值，而是可以在一个范围内变动的随机变量。根据区间用来划分不同的责任区，可以求得更多的解，得到更多的分配方案，也更加适用于实际的情况。我们据此对模型二进行改动，将每个节点的权值（警情量）由定值变为一个区间，并重新求解模型。

根据文献 [2]，每一辆车通过某段道路时, 它要么成功通过, 要么发生交通事故。对于道路交通而言, 发生事故的概率是很低的, 而伯努利试验的次数又是巨大的。在这种情况下，可以认为事故点年均警情量发生的次数 X 近似服从泊松分布，即 $X \sim P(\lambda)$，由于每个交通事故易发点年均警情量的发生概率都不低于 90%，可以将数据中给出的年均警情量看做泊松分布的一个“上 α 分位点”，其中 $\alpha = 0.9$。泊松分布的概率函数如下图所示，其中橙色的面积应该大于总面积的 90%。



根据泊松分布的定义：

$$P(X=n)=\frac{\lambda^n e^{-\lambda}}{n!} \tag{9}$$

假设 a 是所给数据的某点的警情量，根据模型要求：

$$P(X\geq a)=\sum_{i=a}^{\infty}\frac{e^{-\lambda}\lambda^i}{i!}\geq 0.9 \tag{10}$$

令 λ 是使上式成立的最小值，根据泊松分布参数的含义，λ 是事故发生量的期望。求得实例 1 各节点的 λ 参数为 (实例 2 的数据见附件)：

序号	λ	序号	λ	序号	λ	序号	λ	序号	λ
0	111	5	35	10	28	15	14	20	88
1	61	6	48	11	103	16	93	21	88
2	111	7	68	12	27	17	102	22	33
3	4	8	32	13	79	18	73	23	12
4	90	9	108	14	84	19	111	24	42

因为每年事故的发生量符合泊松分布，我们希望求得关于警情发生量的一个置信区间 $[\theta_1,\theta_2]$，使得这个区间尽量小，并且置信系数至少为 β，即：

$$\begin{aligned}&\min \quad \theta_1-\theta_2\\ &s.t.\sum_{i=\theta_1}^{\theta_2}\frac{e^{-\lambda}\lambda^i}{i!}\geq\beta\end{aligned} \tag{11}$$

取 $\beta=0.95$。用 R 语言求得实例 1 的各节点警情发生量的置信区间为：

序号	置信区间	序号	置信区间	序号	置信区间	序号	置信区间	序号	置信区间
0	[91,134]	5	[24,49]	10	[19,40]	15	[8,23]	20	[71,108]
1	[47,78]	6	[35,64]	11	[84,125]	16	[75,114]	21	[71,108]
2	[91,134]	7	[53,86]	12	[18,39]	17	[83,124]	22	[23,46]
3	[1,10]	8	[22,45]	13	[63,98]	18	[57,92]	23	[6,21]
4	[72,111]	9	[89,130]	14	[67,104]	19	[91,134]	24	[30,57]

7.2 模型求解

根据以上求解结果，我们建立如下的节点权值区间化模型。

令 $\hat{\theta}_i = [\theta_{i1}, \theta_{i2}]$，表示第 i 个节点权值的区间范围，S_k 表示第 k 个划分的节点集合。

为了使得每个区域的警情发生量均衡，我们建立目标函数使得不同区域的警情量的方差最小，目标函数如下：

$$\min \quad \sum_{i=1}^{k}(\sum_{\hat{\theta}_t \in S_i} \hat{\theta}_t - \frac{\sum_{j=1}^{n} \lambda_j}{k})^2 \tag{12}$$

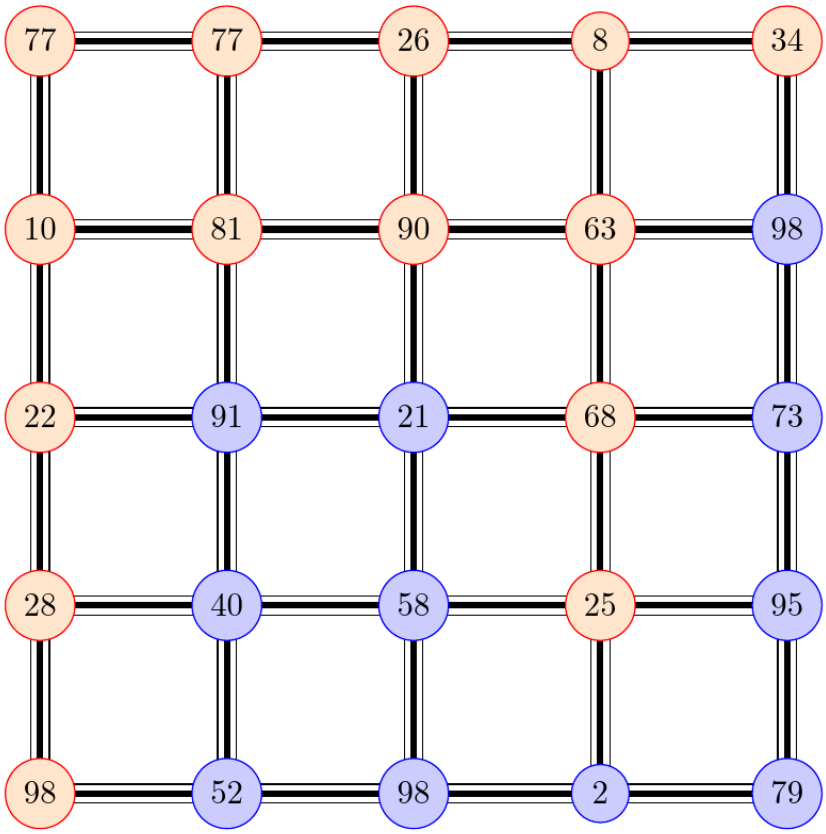
在这种情况下，新的规划模型可表示为

$$\min \quad \sum_{i=1}^{k}(\sum_{\hat{\theta}_t \in S_i} \hat{\theta}_t - \frac{\sum_{j=1}^{n} \lambda_j}{k})^2 \tag{13}$$

$$s.t. \forall n \in \{1, 2, \cdots, k\} \forall i, j \in V_n, C_{Aij} = C_{Aji} = 1 (i \neq j) \tag{14}$$

其中 k 为分区的数量，n 是节点数量，$\hat{\theta}$ 是节点权值和的区间范围。

在这种情况下，对于数据 1 我们求得的一组最优解如下：



八、 结果分析与检验

8.1 模型优点

1. 模型使用了基于搜索算法和贪心策略求局部最优解的算法和启发式算法来解决图的划分问题，并对比了两种算法的优劣和使用范围。
2. 基于原理，实现了求解图划分问题的较高效的启发式算法

8.2 模型缺陷

1. 搜索算法基于贪心策略，在划分的组别较少时会有比较明显的误差
2. 启发式算法的结果有不确定性，可能需要多次运行以达到较好的结果

参考文献

[1] 郑丽丽, 武继刚, 陈勇, 朱梅霞. 带权图的均衡 k 划分 [J]. 计算机研究与发展,2015,52(03):769-776.

[2] 孟祥海, 覃薇, 霍晓艳. 基于统计与假设检验的高速公路交通事故数据分布特性 [J]. 交通运输工程学报,2018,18(01):139-149.

附录 A　问题 1 求解结果

由于问题只需要划分为两个集合。我们在这里对没一组解只展示某一个集合的对应情况，另一个集合是其对应的补集。数据 1：

```
0 1 2 3 4 5 6 8 9 10 15 16 20
0 1 2 3 4 5 6 9 10 11 12 16
0 1 2 3 4 5 6 10 15 17 20 21 22 23
0 1 2 3 4 5 7 8 10 15 16 20 21
0 1 2 3 4 5 8 10 11 13 16 18
0 1 2 3 4 6 7 8 12 16 17 18
0 1 2 3 4 6 7 9 11 12 14
0 1 2 3 5 6 8 10 15 16 18 20 21 22 23
0 1 2 3 5 10 11 12 13 15 18 20 21
0 1 2 5 6 7 11 12 13 17 18
0 1 2 5 6 10 15 16 17 20 21 22 23
0 1 2 5 7 10 12 15 19 20 21 22 23 24
0 1 2 5 7 10 15 17 18 20 21 22 23
0 1 5 6 10 11 15 16 18 20 21 22 23 24
0 1 5 8 10 12 13 15 18 19 20 21 22 23 24
0 5 7 10 11 12 13 15 16 17 18 20
0 5 8 10 13 15 16 17 18 20 21 22 23 24
0 5 8 10 13 14 15 18 19 20 21 22 23 24
```

由于数据 2 对应的解较多，这里只展示几组：

```
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,16,19,20,23,26,27
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,17,18,21,28,30,31,32,35,36,37,42
```

附录 B　问题 2 求解结果

```
//搜索算法,数据1,k=3
0 5 7 10 12 15 16 17 18
1 2 3 4 6 8 9 11
13 14
//搜索算法,数据2,k=4
0 5 8 10 11 12 13
1 2 3 4 6 7
9 14 18 19 24
15 16 17 20 21 22 23
//启发式算法（选取100次求解的最优解）
//数据1,k=3
10,11,15,16,17,20,21,22, 474.000000
0,1,2,3,4,5,6,7,12, 476.000000
8,9,13,14,18,19,23,24, 464.000000
```

```
//数据1,k=4
13,14,18,19,23,24, 344.000000
10,11,15,16,20,21, 358.000000
0,1,2,3,4,5, 357.000000
6,7,8,9,12,17,22, 355.000000
//数据2,k=3
0,1,2,3,7,8,9,10,14,15,16,17,23,24, 788.000000
21,22,28,29,30,31,32,35,36,37,38,39,40,42,43,44,45,46,47, 777.000000
4,5,6,11,12,13,18,19,20,25,26,27,33,34,41,48, 787.000000
//数据2,k=4
0,1,2,3,7,8,9,14,15,16, 597.000000
4,5,6,11,12,13,18,19,20,25,26,27, 586.000000
31,32,33,34,38,39,40,41,44,45,46,47,48, 577.000000
10,17,21,22,23,24,28,29,30,35,36,37,42,43, 592.000000
```

附录 C　对数据 2 的 λ 求解结果

```
62,56,85,45,109,17,78,21,94,40,45,95,21,95,111,93,92,51,52,16,13,37,87,107,24,
19,98,79,92,61,7,13,21,114,29,53,16,31,22,108,12,33,55,45,65,53,76,80,63
```

附录 D　基于贪心策略的搜索算法——C++ 源代码

```
//divide into two
#include <bits/stdc++.h>
using namespace std;

set<pair<set<int>, int>> status_set;
pair<set<int>, int> annser;
int tt = 0;
set<set<int>> searched_set; //the set of vertex which have searched, to prune
int number = 0;

struct Graph
{
    //the sum of the bellow array is 1414
    int vertex_weight[25] = {98, 52, 98, 2, 79, 28, 40, 58, 25, 95, 22, 91, 21, 68, 73,
     10, 81, 90, 63, 98, 77, 77, 26, 8, 34};
    vector<vector<int>> a_list;//adjacent list;
}a;

int vis[sizeof(a.vertex_weight) / sizeof(int)]; //if the node has been traversed
int have_in_set[sizeof(a.vertex_weight) / sizeof(int)]; //vertex i have be in a set
```

```
void construct_edges()
{
    int num = sizeof(a.vertex_weight)/sizeof(int);
    a.a_list.resize(num+5);
    int from[40] = {11,8,10,22,5,7,15,8,10,15,1,16,18,1,16,18,4,6,
    12,20,12,3,9,6,14,19,0,2,17,11,23,0,2,17,5,13,21,3,13,7};
    int to[40] = {16,13,11,23,6,12,20,9,15,16,6,21,19,2,17,23,9,7,
    13,21,17,4,14,11,19,24,5,3,22,12,24,1,7,18,10,14,22,8,18,8};
    for(int i = 0; i < (sizeof(from)/sizeof(int)); i++)
    {
        a.a_list[from[i]].push_back(to[i]);
        a.a_list[to[i]].push_back(from[i]);
    }
}


void dfs(int i) //to traverse the graph
{
    if (vis[i])
        return;
    vis[i] = 1;
    for (auto it = a.a_list[i].begin(); it != a.a_list[i].end(); it++)
    {
        dfs(*it);
    }
}

bool is_connected_graph(pair<set<int>, int> status) //to judge the remain graph is or not the
     connected graph
{
    memset(vis, 0, sizeof(vis));
    for (auto it = status.first.begin(); it != status.first.end(); it++)
    {
        vis[*it] = 1;
    }
    for (int i = 0; i < sizeof(a.vertex_weight) / sizeof(int); i++)
    {
        if (!vis[i])
        {
            dfs(i);
            break;
        }
    }
    for (int i = 0; i < sizeof(a.vertex_weight) / sizeof(int); i++)
    {
        if (!vis[i])
```

```
            return false;
    }
    return true;
}

void force_search()
{
    bool have_found = false;
    pair<set<int>, int> first_status;
    first_status.first.insert(20);
    int sum = 0;
    for (int i = 0; i < (sizeof(a.vertex_weight) / sizeof(int)); i++)
        sum += a.vertex_weight[i];
    first_status.second = abs(sum / 2 - a.vertex_weight[20]);
    status_set.insert(first_status);
    searched_set.insert(first_status.first);
    annser = first_status;
    while (!status_set.empty())
    {
        //cout<<tt++<<'\n';
        pair<set<int>, int> current_status = *(status_set.begin());

        /*for(auto it = current_status.first.begin(); it != current_status.first.end(); it++)
            cout<<*it<<' ';
        cout<<'\n';*/

        status_set.erase(status_set.begin());
        set<int> to_insert_vertex; //the vertex in complementary set of current_status which
            connected with it
        for (auto it = current_status.first.begin(); it != current_status.first.end(); it++)
        {
            //cout<<111<<*it<<111;
            //cout<<a.a_list[*it].size();
            for (auto it1 = a.a_list[*it].begin(); it1 != a.a_list[*it].end(); it1++)
            {
                //cout<<*it1<<'\n';
                auto it2 = find(current_status.first.begin(), current_status.first.end(), *it1);
                //if(current_status.first.find(*it1) == current_status.first.end())//if not in
                    the current set
                if (it2 == current_status.first.end())
                    to_insert_vertex.insert(*it1);
            }
        }
        //cout<<111<<to_insert_vertex.size();
        for (auto it = to_insert_vertex.begin(); it != to_insert_vertex.end(); it++)
        {
            pair<set<int>, int> new_status;
```

```
        new_status.first = current_status.first;
        new_status.first.insert(*it);
        new_status.second = current_status.second - a.vertex_weight[*it];

        if (new_status.second == 0)
        {
            //cout<<'1';
            if (searched_set.find(new_status.first) == searched_set.end()){
                searched_set.insert(new_status.first);
                if (is_connected_graph(new_status))
                {
                    cout<<number++<<' ';
                    //have_found = true;
                    annser = new_status;
                    //break;
                    /*for (auto it = new_status.first.begin(); it != new_status.first.end();
                         it++)
                        cout << *it << ' ';
                    cout << '\n';*/
                }
            }
        }
        else if (new_status.second > 0)
        {
            //cout<<'2';
            if (searched_set.find(new_status.first) == searched_set.end())
            {
                status_set.insert(new_status);
                searched_set.insert(new_status.first);
            }
        }
        else
        {
            //cout<<'3';
            if (searched_set.find(new_status.first) == searched_set.end())
            {
                searched_set.insert(new_status.first);
                if (abs(new_status.second) < current_status.second)
                {
                    if (abs(new_status.second) < annser.second)
                        if (is_connected_graph(new_status))
                            annser = new_status;
                }
                else
                {
                    if (current_status.second < annser.second)
                        if (is_connected_graph(current_status))
```

```
                            annser = current_status;
                    }
                }
            }
        }
        if (have_found)
            break;
    }
}

int main()
{
    memset(have_in_set, 0, sizeof(have_in_set));
    construct_edges();
    force_search();
    /*for(int i = 0; i < 25; i++)
    {
        for(int j = 0; j < 25; j++)
        {
            cout<<a.edges[i][j]<<' ';
        }
        cout<<'\n';
    }*/
    /*for(int i = 0; i < 25; i++)
    {
        for(auto it = a.a_list[i].begin(); it != a.a_list[i].end(); it++)
            cout<<*it<<' ';
        cout<<'\n';
    }*/
    for (auto it = annser.first.begin(); it != annser.first.end(); it++)
        cout << *it << ' ';
    cout << '\n';
    cout << annser.second;

    system("pause");
}
```

附录 E　启发式搜索算法——C++ 源代码

```
#include <cstdio>
#include <vector>
#include <set>
#include <cmath>
#include <map>
#include <random>
```

```
#include <ctime>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;

map<vector<int>,double> mp;
int vertex2[26]={98,52,98,2,79,28,40,58,25,95,22,91,21,68,73,10,81,90,63,98,77,77,26,8,34};
int vertex1[50]={52,47,73,37,96,12,67,15,82,32,37,83,15,83,98,81,80,42,43,11,9,29,75,94,18,
14,85,68,80,51,4,9,15,100,22,44,11,24,16,95,8,26,46,37,55,44,65,69,53};
int direct[4][2]={{0,1},{1,0},{0,-1},{-1,0}};
bool chosen[50];
bool visit[50];
int cnt=0;

vector<vector<int>> vec;

bool isDup(vector<int> ve,int t){
    int i;
    for(i=0;i<(int)ve.size();i++){
        if(t==ve[i]) return true;
    }
    return false;
}

bool isDup(vector<vector<int>> ve,vector<int> t){
    int i;
    for(i=0;i<(int)ve.size();i++){
        if(t==ve[i]) return true;
    }
    return false;
}

bool isOK(int dx,int dy,int pos){
    int nx=pos%7+dx,ny=pos/7+dy;
    if(nx<0||ny<0||nx>=7||ny>=7) return false;
    else return true;
}

bool isreachable(vector<int> inp,int pos){
    int i,dx,dy;
    for(i=0;i<4;i++){
        dx=direct[i][0];dy=direct[i][1];
        if(isOK(dx,dy,pos)){
            if(isDup(inp,pos+dx+7*dy)) return true;
        }
    }
```

```
    return false;
}

void dfs(int cur){
    int pos,dir;
    for(dir=0;dir<4;dir++){
        if(!isOK(direct[dir][0],direct[dir][1],cur)) continue;
        pos=cur+direct[dir][0]+direct[dir][1]*7;
        if(visit[pos]) continue;
        cnt++;
        visit[pos]=true;
        dfs(pos);
    }
    return ;
}

bool isAvail(){
    int i,j,cn=0;
    vector<int> temp;
    for(i=0;i<49;i++){
        if(chosen[i]){
            visit[i]=true;cn++;
        }
        else temp.push_back(i);
    }
    cnt=0;
    dfs(temp[0]);
    if(cnt==49-cn) return true;
    else return false;
}

int thegain(vector<int> inp,int pos){
    int i,dx,dy,sum=0;
    for(i=0;i<4;i++){
        dx=direct[i][0];dy=direct[i][1];
        if(isOK(dx,dy,pos)){
            if(isDup(inp,pos+dx+7*dy)) sum++;
            else sum--;
        }
    }
    return sum;
}

int main(){
    int tes=100,i,j,k,n=2,index,mgain,mpos,gain,weight;
    vector<vector<int>> fin;
    double avg=1176,res=1176*1176; //49-2
```

```
//double avg=784,res=784*784; //49-3
//double avg=588,res=588*588; //49-4
srand((unsigned)time(NULL));
while(tes--){
    memset(chosen,0,sizeof(chosen));
    vec={};
    for(i=0;i<n-1;i++){
        vector<int> temp;
        vector<int> cand;
        while(temp.empty()||mp[temp]<avg){
            if(cand.empty()){
                index=rand()%49;
                while(chosen[index]){
                    index=rand()%49;
                }
                temp.push_back(index);
                mp[temp]=vertex1[index];
                chosen[index]=true;
            }
            else{
                mgain=-4;
                weight=mp[temp];
                for(j=0;j<cand.size();j++){
                    index=cand[j];
                    gain=thegain(temp,index);
                    if(gain>mgain&&!chosen[index]){
                        mgain=gain;
                        mpos=index;
                    }
                }
                if(abs(avg-weight-vertex1[mpos])<abs(avg-weight)){
                    temp.push_back(mpos);
                    chosen[mpos]=true;
                    sort(temp.begin(),temp.end());
                    mp[temp]=(double)weight+vertex1[mpos];
                }
                else break;
            }
            for(j=0;j<49;j++){
                if(!isDup(temp,j)&&isreachable(temp,j)){
                    if(cand.empty()) cand.push_back(j);
                    else if(!isDup(cand,j)) cand.push_back(j);;
                }
            }
            sort(cand.begin(),cand.end());
        }
        memset(visit,0,sizeof(visit));
```

```
            cnt=0;
            if(!isAvail()){
                for(j=0;j<temp.size();j++) chosen[temp[j]]=false;
                i--;
            }
            else vec.push_back(temp);
        }
        double tar=2352,dev=0;
        printf("%d:\n",100-tes);
        for(i=0;i<n-1;i++){
            for(j=0;j<vec[i].size();j++){
                printf("%d ",vec[i][j]);
            }
            printf(" %f\n",mp[vec[i]]);
            tar-=mp[vec[i]];
            dev+=mp[vec[i]]*mp[vec[i]];
        }
        dev+=tar*tar;
        for(i=0;i<49;i++){
            if(!chosen[i]) printf("%d ",i);
        }
        printf(" %f\n",tar);
        if(abs(dev/n-avg*avg)<res){
            res=abs(dev/n-avg*avg);
            fin=vec;
        }
    }
    double tar=2352;
    memset(chosen,0,sizeof(chosen));
    printf("best:\n");
    for(i=0;i<n-1;i++){
        for(j=0;j<fin[i].size();j++){
            printf("%d,",fin[i][j]);
            chosen[fin[i][j]]=true;
        }
        printf(" %f\n",mp[fin[i]]);
        tar-=mp[fin[i]];
    }
    for(i=0;i<49;i++){
        if(!chosen[i]) printf("%d,",i);
    }
    printf(" %f\ndev:%f",tar,res);
    return 0;
}
```

附录 F　求解置信区间的 R 语言代码

```
setwd("/home/lly/hduMCM/thesis/model2/")
data<-read.csv("data1.csv")

a<-data["年均警情量"]
a<-as.matrix(a)
a<-na.omit(a)
a<-qpois(0.9,a)
a<-as.vector(t(a))
print(a)
for (i in 1:length(a))
   print(poisson.test(a[i]))
```