机器人协同巡逻问题

摘要

利用机器人对场所进行定点监视或不间断地巡逻监视是安全自动化的方法之一，我们需要在不同情况下合理的分配巡逻机器人来达到机器人巡逻效率的最大化。在问题一中，我们首先确定了图的最小生成树，并以最小化结点空闲度的均值和方差为目标函数，建立了数学规划模型来对最小生成树进行划分，获得了可行的安排方式。在问题二中，我们采用泊松分布和均匀分布的复合模型来模拟街道不同位置发生事故的概率，并依据其概率分布进行具体的任务划分。在问题三中，我们对有向图的边权、机器人的扫描范围等因素进行了合理简化，最终把问题转化为与问题一相似的无向图结点巡逻问题。

在问题一中，由于路径长度直接影响巡逻的时间间隔进而影响空闲度，我们建立了使所有子图的总路径长度最小的以及使各个子图路径长度的方差最小的多目标规划模型。为了让总路径长度最小最小，我们可以通过建立**最小生成树**来让连接各个各个子图的的连通子图的总路径长度最小，并且将最小生成树进行划分，从而让结点被巡逻的时间间隔最短，空闲度最小。之后，我们根据划分子图必须连通等约束条件对最小生成树进行多次随机划分，并选择路径长度最短且各条路径的方差最小的分配方案，从而得到在机器人数量不同时针对于每个机器人的巡逻方案。

在问题二中，我们综合**泊松分布和均匀分布**建立了描述不同地点发生事故的概率分布模型，确定了在发生事故时事故发生在各个具体地点的概率，并以均匀化各个辖区内的概率分布为目标进行巡逻区域的划分。在模型求解中，我们通过**随机模拟**方法模拟机器人的巡逻过程，搜索了所以可能的划分情况。最终得到在只存在单个事故点的情况下得到的最佳划分点为 40、50、121、211；在存在两个个事故点的情况下对应的最佳划分点为 42、62、134、162。

在问题三中，我们把单元格等价为问题一中的结点，并且根据单元格大小和巡逻速度确定各条边的权值。从而把考虑了单元格大小和扫描范围的模型简化为问题一中的图连接模型。我们证明了选择“线形路径”才能使得时间间隔最短这一结论，根据这一结论使用网络中各条边在两个方向的均值来表示各条边的权值。并采用“先寻找最小生成树，后根据约束随机划分生成树并比较”这一与问题一相似的思路进行求解。但由于问题三有初始状态，我们在问题三的划分中增加了把 8 个初始结点划分入不同子图的约束条件，并且把最大化效用函数作为目标函数，最终得到了使得效用函数最大的分配方式，该分配方式对应的效用函数值为 3.20。

关键字：最小生成树　泊松分布　随机模拟

一、 问题重述

随着社会经济的发展，社会对于安全自动化的需求越来越迫切。巡逻机器人是一个多功能集成系统，它可以感知环境，进行规划路线，做出动态决策。为了保证场所的安全，可以利用机器人对场所进行定点监视或不间断地巡逻监视。

在问题 1 中，假设你们在尝试在边权不同的两个地区（1 和 2）进行机器人巡逻试验，为了给两个地区分配合理的巡逻机器人数量能够达到巡逻效率的最大化，分别在 8，12，16 个机器人的情况下对两个地区进行巡逻。在问题 1 中我们需要对这些机器人的巡逻问题建立数学模型并给出高效的求解算法，从而达到巡逻效率的最大化。

在问题 2 中，我们需要对可能发生处突发紧急事件的步行街进行建模。巡逻过程中需要保证对整条街道的覆盖，同时，当街道上的某事件高发处突发紧急事件时，距离最近的机器人会立刻前往该处位置进行定位后通知安保团队，随后继续巡逻。我们需要对这些机器人的巡逻问题建立数学模型并给出高效的求解算法，在实现对整条街道的巡逻效率最大化的同时，在应对紧急事件时，提高对事件高发处的巡逻频率。并且给出在有 5 个机器人的情况下，对以下两种特定的情况的处理方案：

1. 假设位于距离街道左端 42 米的事件高发处发生紧急事件
2. 假设分别位于距离街道左端 42 和 147 米的两个事件高发处发生紧急事件

在问题 3 中我们需要考虑具有高低起伏地形的地区（相邻单元格间不同方向移动的速度不一致）进行巡逻的任务。在问题中，每个机器人有一个大小为 D 的方形巡逻范围，巡逻区域包含大小为 D 的 80 个单元格。并且问题给出了 8 个机器人的初始位置，在巡逻过程中需要保证对所有单元格的覆盖。请对这些机器人的巡逻问题建立数学模型并给出高效的求解算法，实现整个区域的巡逻效率最大化。

二、 问题分析

题目中的三个问题有着明显的相关性，在问题 1 中，我们需要在无边权影响和有边权影响（地区 2）的前提下，考虑模型在顶点连接情况不同的前提下，针对机器人的巡逻问题建立数学模型并给出高效的求解算法。而在问题二中，我们需要考虑地区有随机因素时的求解方法。在问题三中，问题中的图变为了正反向巡逻速率有差异的有向图，并且搜索方式相对问题一有一定的变化。

在问题一中，我们首先根据机器人调度中的实际情况，修正了结点空闲度的计算方法，建立了使所有结点的总空闲度最小以及使所有结点的空闲度的方差最小的多目标规划模型。由于路径长度直接影响巡逻的时间间隔进而影响空闲度，为了让总空闲度最

小，我们可以通过建立最小生成树来让连接各个结点的连通子图的路径长度最小，从而让结点被巡逻的时间间隔最短，空闲度最小。之后，我们根据划分子图必须连通等约束条件对最小生成树进行多次随机划分并选择路径长度最短且各条路径的方差最小的分配方案，从而得到针对于每个机器人的巡逻方案。

在问题二中，我们考虑到事故发生的随机性，综合泊松分布和均匀分布建立了描述不同地点发生事故的概率分布模型，确定了在发生事故时事故发生在各个具体地点的概率，并以减小各个辖区内发生事故次数的方差为目标进行机器人的划分，从而得到在考虑随机因素的分配方案。在有多个紧急情况点时，我们可以使用多个单个事故点的概率分布模型进行叠加，从而得到对应多事故点的概率分布情况。由于模型中各个分布所占权重这一参数取值无法确定，我们可以对分布权重 W 进行敏感性分析来确定概率分布对于划分的影响。在确定了各个位置发生事故的概率后，我们可以对机器人的巡逻过程进行随机模拟来得到最优解。

在问题三中，首先我们可以把单元格等价为问题一中的结点，并且根据单元格大小和巡逻速度确定各条边的权值。从而把考虑了单元格大小和扫描范围的模型简化为问题一中的图连接模型。我们证明了选择“线形路径”才能使得时间间隔最短这一结论，根据这一结论使用网络中各条边在两个方向的均值来表示各条边的权值。并采用“先寻找最小生成树，后根据约束随机划分生成树并比较”这一与问题一相似的方法进行求解。由于题目中的各个结点有初始位置，因此我们需要选择将所有结点划分入不同集合的划分方案，并在划分完成后检验划分路径是否可连通，以及在可连通情况下检测连通后沿着某一方向移动可否降低总边权。

三、 模型假设及符号说明

3.1 模型假设

1. 机器人的巡逻以米为单位
2. 假设机器人在巡逻时不受到外界因素（例如道路环境）的影响
3. 在没有事故点的情况下，事故的发生位置服从均匀分布；有事故点时，在事故点附近事故的发生概率服从泊松分布

3.2 符号说明

符号	意义
G	表示结点间相互关系的无向图
W_{ij}	结点 i 和 j 的权重
A_{ij}	表示结点的分配情况，$A_{ij}=1$ 时，表示第 i 个结点被分到了第 j 条路径
l_k	第 k 条路径的长度
P_x	位置 x 处发生事故的概率

四、 问题一

4.1 模型准备

4.1.1 目标函数的确定

我们可以使用空闲度这一评估标准来判断某一个巡逻策略的好坏，结点的空闲度是结点上相邻两次巡逻的时间间隔。在实际情况中，由于存在，所以结点相邻两次巡逻的时间间隔并不一定相同，在这种情况下，我们用结点各次被巡逻的时间间隔的**叠加**代表巡逻的时间间隔。它可以表示为情况如下：

$$\frac{1}{T_n}=\sum_{i=0}^{n}\frac{1}{t_{n,i}} \tag{1}$$

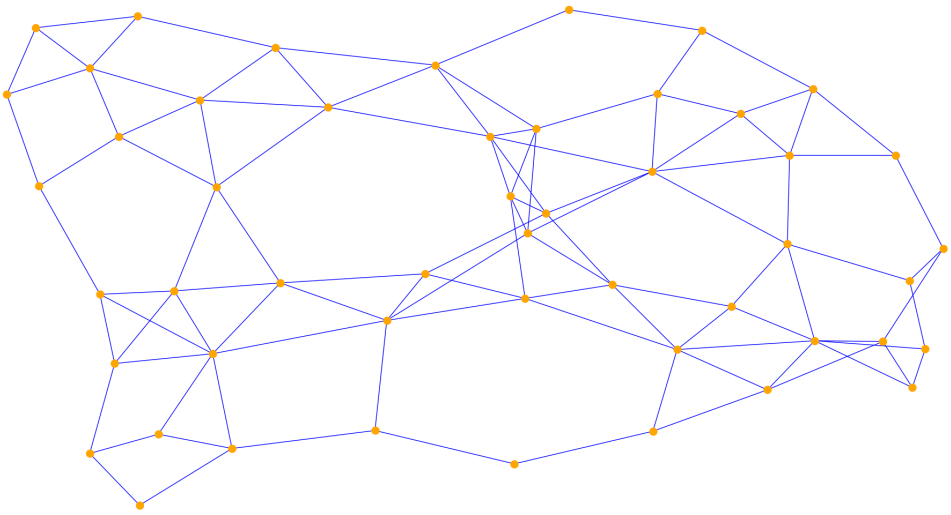
其中 $t_{n,i}$ 代表经过某个结点的某条特定路径 i 相邻两次巡逻的时间间隔。

由于各个机器人的速度是恒定的，结点上相邻两次巡逻的时间间隔可以表示为在相邻两次巡逻间走过的路程。因此总空闲度最小这一目标可以转化为巡逻走过的路程最小。

在我们的模型中，我们需要把各条路程分配给各个特定的机器人，因而我们需要让各条路径的长度差尽可能小从而使各个机器人的任务量均等。也就是我们要最小化路径长度的**方差**。

4.2 模型建立

地区 1 的路径连接情况如下：



我们可以用一个无向图 $G=(V,E)$ 来表示各个结点和结点之间的连接情况。为了让所有机器人走过的路程最小，我们可以寻找 G 的一个能连通所有结点的总权重最小的的子集。为了把任务均匀的分配在各个机器人上，我们需要把这一子集划分为多个路径长度方差最小的独立集合。也就是首先寻找图 G 的**最小生成树**，并且根据题目中给出的等限制条件，建立数学规划模型来对最小生成树进行平均划分，从而得到针对各个机器人的任务分配。

我们用可以图 G 的连通矩阵 Q 和邻接矩阵 W 记录最小生成树的边信息。其中 Q 表示连通矩阵，$Q_{ij}=1$ 表示结点 i 和结点 j 连通；$Q_{ij}=\infty$ 表示结点 i 和结点 j 不连通，当 $i=j$ 时，$Q_{ij}=0$。

W 表示权重矩阵，当结点 i 和结点 j 连通时，W_{ij} 表示两结点之间的边权，同理，$W_{ij}=\infty$ 表示结点 i 和结点 j 不连通，当 $i=j$ 时，$W_{ij}=0$。

我们把最小生成树分为 $n=8,12,16$ 条路径，为最小生成树的划分构造一个维度为 $m\times n$ 的 A 矩阵，其中 m 为结点数。当 $A_{ij}=1$ 时，表示第 i 个结点被分到了第 j 条路径。

由于巡逻过程中需要保证对地区中所有结点的覆盖，我们需要约束条件来限制所有的结点都至少在一条路径之中。这一约束条件可以用以下的关系式来保证：

$$\sum_{j=1}^{n} A_{ij} \geq 1, i=(1,2,\cdots,m) \tag{2}$$

以下约束保证了在第 k 条路径中，如果第 i 个结点和第 j 个结点包含在其中，那么结点 i 和结点 j 连通。具体讨论如下：

1. 如果结点 i 和结点 j 都包含在路径 k 中，那么 $A_{ik}A_{jk}=1$。如果 i，j 连通，那么 $Q_{ij}=1$，而约束小于 ∞ 保证了结点 i 只能与和它相连的结点连通，符合路径结点连通的要求；如果 i，j 不连通，则 $Q_{ij}=\infty$，显然不符合约束。

2. 如果结点 i 和结点 j 至少有一个不包含在路径 k 中，那么 $A_{ik}A_{jk}=0$，符合约束。

我们的确保划分后的路径内结点连通的约束条件可表示为：

$$\sum_{j=1}^{m} A_{ik}A_{jk}W_{ij} < \infty, (i=1,2,\cdots,m; k=1,2,\cdots,n) \tag{3}$$

由上述约束，第 k 条路径的长度可以表示为：

$$l_k = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} A_{ik}A_{jk}W_{ij} \tag{4}$$

根据模型准备部分，我们的规划模型的目标函数为最小化路径总长以及最小化路径长度的方差。这两个目标函数可表示为：

$$\begin{cases} \min & \sum_{k=1}^{n} l_k \\ \min & \sum_{k=1}^{n} (l_k - \bar{l})^2 \end{cases} \tag{5}$$

其中 $\bar{l}$ 代表 l 的均值，可表示为 $\bar{l} = \frac{\sum_{k=1}^{n} l_k}{n}$。

综合上文的分析，我们用于求解最小生成树的最优划分的规划模型可表示为：

$$\begin{aligned} & \min \quad \sum_{k=1}^{n} l_k \\ & \min \quad \sum_{k=1}^{n} (l_k - \bar{l})^2 \\ \text{s.t.} \quad & \sum_{j=1}^{n} A_{ij} \geq 1, i = (1,2,\cdots,m) \\ & \sum_{j=1}^{m} A_{ik}A_{jk}W_{ij} < \infty, (i=1,2,\cdots,m; k=1,2,\cdots,n) \end{aligned} \tag{6}$$

4.3 模型求解

4.3.1 Prim 算法

我们可以使用 Prim 算法来进行最小生成树的求解。算法从图中的任意一个结点开始，每一步执行时从连接目前已有节点的边中选取权值最小的边加入路径集合，知道找到所有路径为止。这一算法的伪代码可以表示如下：

Algorithm 1: 求解最小生成树的 Prim 算法

Data: G：待求最小生成树的连通图，r：连通图的根结点

Result: A:G 的最小生成树，

初始化所有节点;
foreach $u \in G.V$ **do**
    u.key=∞; u.par=null;
end
r.key=0; Q=G.V;
寻找权重最小的边并加入 A;
while $G \neq \Phi$ **do**
    u=getMin(Q);
    foreach $v \in$G.adj[u] **do**
        if $v \in Q$ &$w(u,v) < v.key$ **then**
            v.par=u;v.key=w(u,v)
        end
    end
end

在每次求解可能的划分时，我们随机选取生成树的根结点，并使用 Prim 算法得到对应该根结点的最小生成树。

4.4 规划模型求解

在通过 Prim 算法得到最小生成树后，我们需要对最小生成树进行合理的划分来使得总路径长度和路径长度的方差最小。考虑到在划分时受到“必须覆盖所有结点”、“划分的子图必须连通”等约束条件，可以依据规划模型6来求解最优划分方式。

由于模型的约束条件相对较多，不易于按照规划模型的方法进行直接求解，我们可以考虑随机删除最小生成树中的数条边，从而把最小生成树划分为多个连通子图，并选取符合约束条件的连通子图来求可行解。

对于地区一，我们采用随机模拟的方法得到的划分方式为：

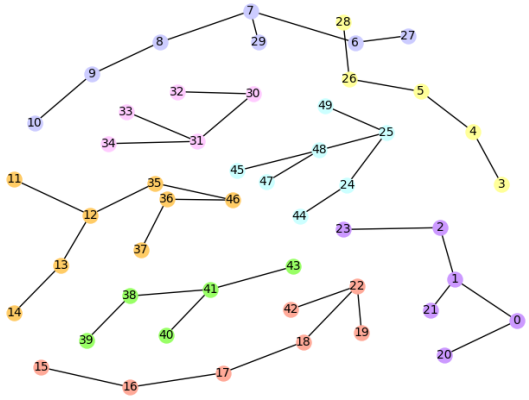


图 1　地区一在 8 个机器人时的划分情况

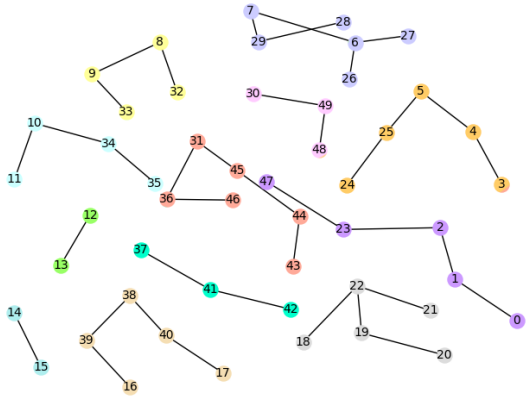


图 2　地区一在 12 个机器人时的划分情况

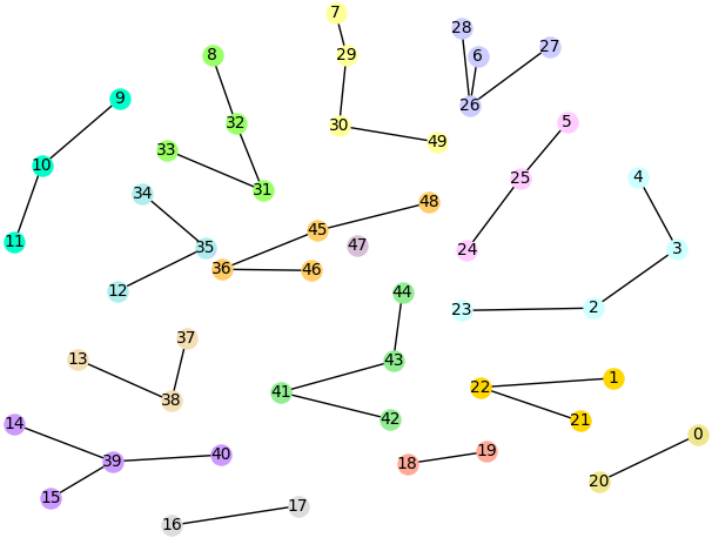


图 3　地区一在 16 个机器人时的划分情况

对于地区二，我们采用随机模拟的方法得到的划分方式为：

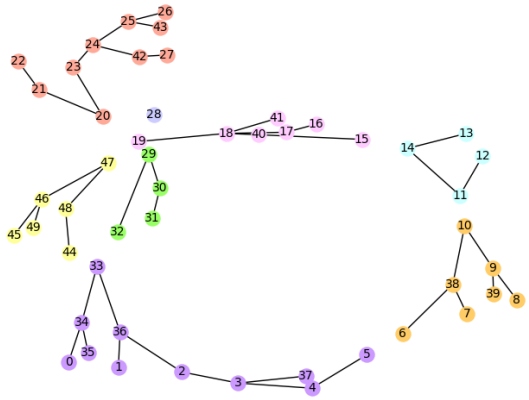


图 4　地区二在 8 个机器人时的划分情况

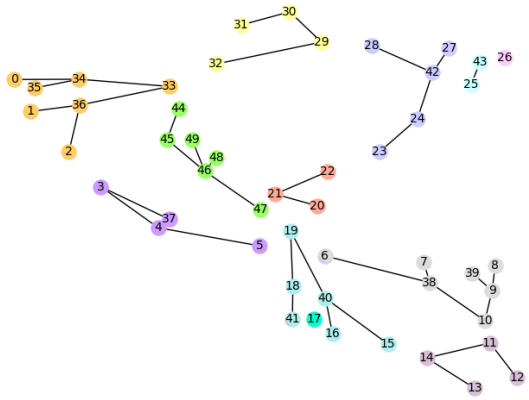


图 5　地区二在 12 个机器人时的划分情况

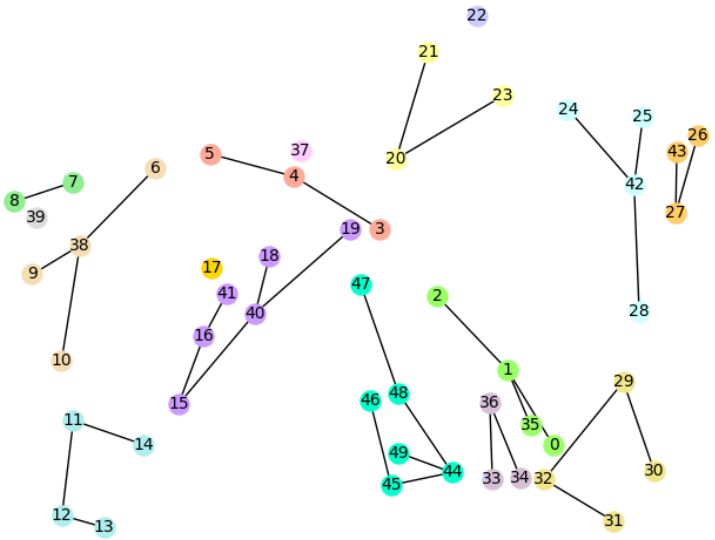


图 6　地区二在 16 个机器人时的划分情况

五、 问题二

5.1 模型建立

根据实际情况我们假设机器人在沿着步行街巡逻时可以获得步行街沿线的所有信息，这样我们可以将机器人在步行街的巡逻过程简化为机器人沿着直线往复进行巡逻，并在某处突发紧急事件时，选择距离最近的机器人会前往该处进行定位。

由于机器人的巡逻速度为 1 米每秒，我们可以将总长为 300 米的步行街划分为 300 个离散点组成的集合。考虑到我们已经把机器人的巡逻过程简化为沿着直线巡逻，因而我们可以把机器人分配任务理解为把街道划分为连续的 5 组点集，每组点集代表对每个巡逻机器人分配的巡逻区域。

在不考虑事故高发点的情况下，可以认为事故在整个路段上服从离散均匀分布。用 X 表示巡逻路径的总长度，则在发生事故时，事故发生在路段上各个位置的概率均为 $\frac{1}{X}$。

由于单个事故点在某一时间发生事故的概率极小，整个巡逻过程的持续时间很长，因而事故点附近发生事故的概率可以近似表示为一个泊松分布模型。泊松分布的参数 λ 对应事故高发地的位置。

在单个事故点的情况下，我们可以使用一个描述无事故高发地时事故发生概率的均匀分布，与描述事故点及其附近的事故发生概率的泊松分布进行复合，从而得到描述总体情况的泊松分布。在多个事故高发点的情况下，我们可以转而为每个事故高发点分配不同参数的泊松分布进行表示。在这种情况下，位置 x 处发生事故的概率 P_x 为:

$$P_x = p * \frac{1}{X} + \sum_{i=1}^{n} p_i \frac{\lambda_i^x}{x!} e^{-\lambda_i} \tag{7}$$

上式中的均匀分布的权重 p 和代表各个事故发生点的权重 p_i，影响事故点的危险性，λ_i 为泊松分布的参数。

根据上文的假设，机器人巡逻区域的划分可以由四个划分点 x_1, x_2, x_3, x_4 表示。从 x_i 到 x_{i+1} 的巡逻段发生事故的次数的数学期望可表示为：

$$E(x_i, x_{i+1}) = \sum_{i=a}^{b} P_x \tag{8}$$

我们应该让各个巡逻段发生事故的次数的数学期望尽可能均等，这可以表现为让各个巡逻地段的发生事故的次数方差最小，我们的优化目标可以表示为:

$$\min D(E) = \frac{1}{5}\sum_{i=0}^{4}(E(x_i, x_{i+1}) - \bar{E})^2 \tag{9}$$

5.2 模型求解

在确定了当发生一起事故时事故发生在各个位置的概率后，我们可以使用随机模拟的方式来模拟各个辖区内的事故发生情况。具体方法可表示为：

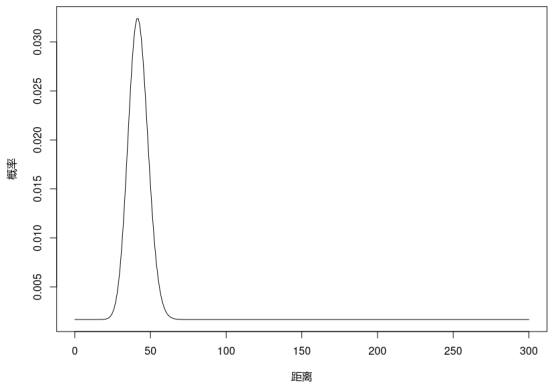
1. 每间隔一秒，机器人移动到相邻的下一个结点
2. 每隔服从正态分布 $N(dt, D)$ 的一个随机生成的时间后，发生一次紧急事件
3. 突发紧急事件时，距离最近的机器人立刻前往该处位置进行定位，其它机器人进行巡逻
4. 模拟一段时间，计算按照此种该种划分方案巡逻时机器人的总巡逻长度

根据上文的假设，机器人巡逻区域的划分可以由四个划分点表示，我们可以通过搜索的方式列举机器人可能的分配情况，对各种情况进行随机模拟来获得对应该种分配方案的总巡逻路程。最终选取巡逻路程最大的方案作为最优巡逻方案。

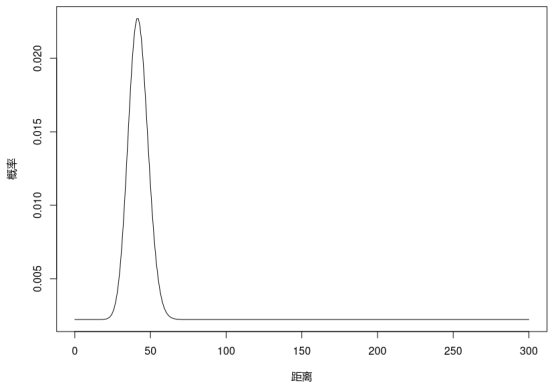
5.2.1 敏感性分析

根据上文，我们可以使用离散均匀分布和泊松分布的复合分布来表示街道上各点发生事故的概率。由于无法根据数据预先确定各个分布的权重，我们可以首先通过搜索方法列举不同权值时街道上各点发生事故概率的分布情况。之后可以对比以上的各个分布情况图来分析各个边的权重对于分布的影响。

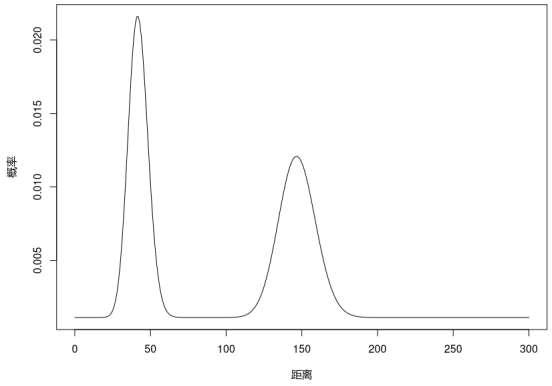
我们选取 $p : p_1 = 1 : 1$、$p : p_1 = 1 : 2$、$p : p_1 : p_2 = 1 : 1 : 1$、$p : p_1 : p_2 = 2 : 1 : 1$ 四种情况，并且做出各个情况下的概率分布图如下：



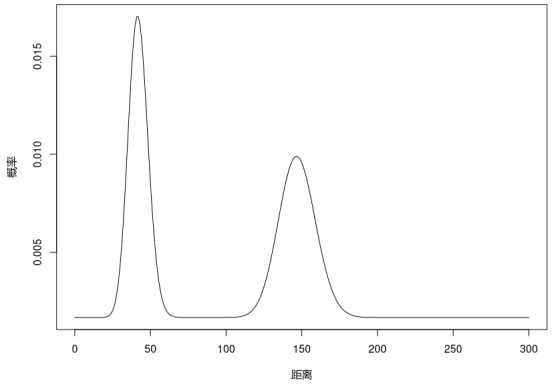
(a) 权重均为 1/2 时的单事故点模型



(b) 事故点权重为 1/3 的单事故点模型



(c) 权重均为 1/3 的双事故点模型



(d) 事故点权重为 1/4 的双事故点模型

图 7　参数不同情况下的概率分布

观察上图可以发现，在只有单个事故高发点的情况下，调整均匀分布的权重 p 只会对发生事故概率的峰值影响较大。而对分布整体影响不大。在有多个事故高发点的情况下，调整均匀分布的权重 p 对分布的影响与单个事故点的情况相似，而调整某个事故点发生事故的权重 p_i 时会整体影响两个事故点发生事故的相对概率。

5.2.2 描述巡逻过程的随机模拟方法

观察上面的敏感性分析的结果，$p : p_1 = 1 : 1$ 以及 $p : p_1 : p_2 = 1 : 1 : 1$ 能够较好的反映事故点附近的概率分布特性，因此在下面的模拟分析中，我们在单个事故点的情况下选取 $p : p_1 = 1 : 1$ 的情况，在多个事故点的情况下选取权重 $p : p_1 : p_2 = 1 : 1 : 1$ 的情况。

由于我们已经把问题简化成了为在一维直线上巡逻的机器人分配相连的巡逻路径的问题。在为 5 个机器人划分巡逻路径时，我们可以简化问题为在路径中合理的插入 4 个节点，并且让插入节点后进行模拟时的总巡逻距离最长。

我们对机器人的移动过程进行模拟，在模拟中每间隔一秒，机器人移动到相邻的下一个结点，每隔服从正态分布 $N(dt, D)$ 的一个随机生成的时间后，模拟一次紧急事件的发生。在紧急事件发生时，距离最近的机器人立刻前往该处位置进行定位，其它机器人进行巡逻。我们可以对四个划分点的可行情况进行搜索，并确定最优的划分点。

我们通过对可行情况进行搜索，在只存在单个事故点的情况下得到的最佳划分点为 40、50、121、211；在存在多个事故点的情况下得到的划分点为 42、62、134、162。在这种情况下对应的机器人的巡逻路径如下表所示：

单个事故点		两个事故点	
机器人 1 的巡逻路径	(0,0),(40,40),(80,0)	机器人 1	(0,0),(42,42),(84,0)
机器人 2 的巡逻路径	(0,40),(10,50),(20,40)	机器人 2	(0,42),(20,62),(40,42)
机器人 3 的巡逻路径	(0,50),(71,121),(142,50)	机器人 3	(0,62),(40,134),(80,62)
机器人 4 的巡逻路径	(0,121),(90,211),(180,121)	机器人 4	(0,134),(28,162),(56,134)
机器人 5 的巡逻路径	(0,211),(89,300),(178,211)	机器人 5	(0,162),(138,300),(276,162)

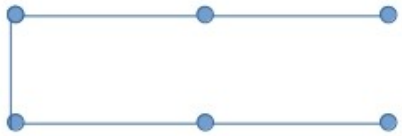
通过观察上述情况，我们发现我们的考虑事故的概率分布的划分模型较好的考虑到了事故高发点处概率密度的影响，在事故高发点附近合理的减小了对应机器人的巡逻范围。从而使得事故高发地附近的机器人在发生事故时能够较快的前往事故点，减少了由于应对紧急事件而损失的巡逻效率。

六、 问题三

6.1 模型建立

在问题三的情境中，每个机器人具有一个大小为 D 的方块形状的巡逻范围，，并且已经将巡逻区域划分成了大小为 D 的 80 个单元格。在机器人巡逻时，在经过某一个单元格时会逐步扫描单元格范围内的所有情况。因此我们可以把机器人**恰好扫描**某个单元格的情形等价为机器人处在问题一的结点上。而机器人在各个结点间的移动过程可以等价为问题一机器人在边上移动的过程。地区的高低起伏的地形导致了正反向移动的速度不同，因而在问题三中结点和它们之间的关系可以用一个**有向图**表示。

一些图中的结点可能有两类连接方式——首尾结点不相连的“线形”（左图）和首尾结点相连的“环形”（右图）：



对于无向图的情况，由于“线形”连接经过的边更少，因而在途径特定数量的点时，选择“线形”连接边的总权重较小，方案更优秀。因而我们可以采取与问题一相似的方法，考虑到“线形”连接会从两个方向经过各条边，我们可以使用两个方向边权的均值作为边权。采用与问题一相似的先求解最小生成树，之后后在约束条件内进行随机模拟来求解。

在问题三中，由于问题中的图是一个有向图，这可能导致在某些情况下采取沿着特定方向移动的“环形路径”变成最优情况。在这种情况下，机器人会沿着特定的方向经过环形的各条边，应该使用对应的特定方向的权值来代表结点间边的权重。我们可以在通过最小生成树确定划分方案后检测划分部分的首尾是否相连，在相连的情况下判断组成环形路径后的总权重是否减小。在总权重减小的前提下选择环形路径。

问题三要求我们选取效用函数作为目标函数，它代表所有单元格在单位时间内被访问的总次数，用 N 是代表单元格数，T 代表巡逻周期，n_i 是单元格 i 在周期内被访问的总次数。由于问题的目标只包括最大化所有单元格在单位时间内被访问的总次数，我们可以排除方差因素对划分方案的影响。效用函数的具体表示如下：

$$f = \sum_{i}^{N} \frac{n_i}{T} \tag{10}$$

由于问题三给出了初始情况下各个机器人的位置，所以我们在选取划分时应当增加一个使得各个机器人的初始位置从属于不同划分的约束条件。用 Z 表示所有初始结点组成的集合，J 表示所有路径组成的集合，这一约束条件可表示为：

$$t_i = \begin{cases} 0 & (i \notin Z) \\ 1 & (i \in Z) \end{cases}$$

$$\forall j \in J, \sum t_i A_{ij} = 1$$

其中 t_i，用于判断 i 是否属于结合 Z，通过求和的方式确保一条路径存在且仅存在一个初始顶点。

在增加约束条件后，问题三的优化模型可以修正为：

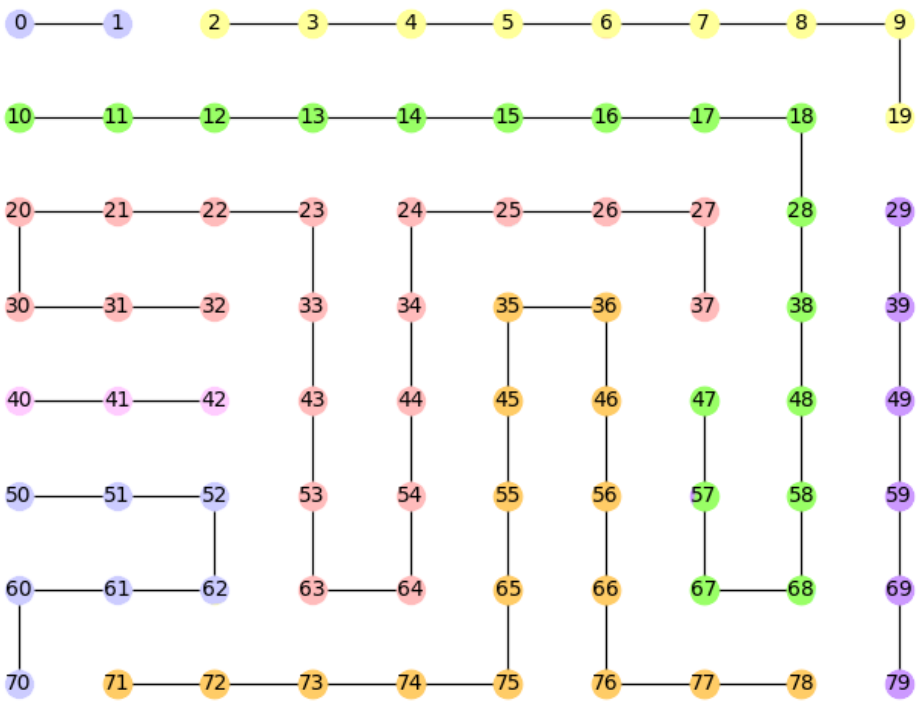
$$
\begin{aligned}
&\max \quad f=\sum_{i}^{N}\frac{n_i}{T}\\
&\text{s.t.} \quad \sum_{j=1}^{n}A_{ij}\geq 1, i=(1,2,\cdots,m)\\
&\sum_{j=1}^{m}A_{ik}A_{jk}W_{ij}<\infty, (i=1,2,\cdots,m; k=1,2,\cdots,n) \qquad (11)\\
&t_i=\begin{cases}0 & (i\notin Z)\\ 1 & (i\in Z)\end{cases}\\
&\forall j\in J, \sum t_iA_{ij}=1
\end{aligned}
$$

6.2 问题求解

我们首先依据题目中给出的单元格间的距离 D 和机器人在不同单元格间移动的速度，计算了各个顶点间沿不同方向的边权，从而得到了表示单元格间相关关系的有向带权图。根据模型建立部分的分析，我们假设路径以总长度较短的“线性”路径为主，使用两个方向边权的均值作为新的权重，构造最小生成树。

在通过 Prim 算法得到最小生成树后，我们需要对最小生成树进行合理的划分来使得总路径长度和路径长度的方差最小。考虑到在划分时受到“必须覆盖所有结点”、“划分的子图必须连通”、“一条路径存在且仅存在一个初始顶点”等约束条件，可以依据规划模型11来求解最优划分方式。

由于模型的条件相对偏多，不易于按照规划模型的方法进行直接求解，我们可以考虑随机删除最小生成树中的数条边，从而把最小生成树划分为多个连通子图，并选取符合约束条件的连通子图作为可行解。得到的一组可行结果如下（图中不同颜色代表不同的划分情况）：



七、 敏感性分析

在问题一中，我们考虑到机器人的巡逻速度一定，为了使得总空闲度最小，我们需要让路径总长度最小；为了使得划分后的各条路径能尽量平衡的分配到各个机器人，我们需要让各条路径长度的方差最小，因而建立了双目标规划模型。为了确定两个目标函数的影响，我们需要对两个权值进行敏感性分析。我们选取 8 个机器人的情况进行下面的敏感性分析。

在双目标规划模型中，路径总长度和各条路径长度的方差是影响决策的两个变量。我们可以列举不同的路径长度和方差的比值来反应权重对于模型的影响。

我们在关于路径长度部分的权重和关于方差部分的权重的比值为 3:1,1:1,1:3 时，对此时的路径总长度 $L = \sum l_k$ 和路径方差 $D = \sum(l_k - \bar{l})$ 进行了计算，计算结果如下：

$W_L : W_D$	L	D
3:1	48	7.50
1:1	46	7.44
1:3	44	7.75

观察上述结果可以发现路径长度和方差的比值对于模型求解结果的影响较小。

八、 模型评价与推广

8.1 模型优点

- 模型通过求解最小生成树并进行划分的方法，通过设计合理的约束条件实现了点的平衡划分
- 模型依据泊松分布和均匀分布模拟了事故在不同位置的分布情况，并进行随机模拟，能较好的反映事故发生后的具体反应过程

8.2 模型不足

- 问题一和二中使用的随机模拟方法得到的结果有一定的随机性，不一定能得到最优解
- 问题三中，可能忽略了部分“环形路径”的分配方案

附录 A　求解问题一的 Python 程序

```
import numpy as np
import networkx as nx
import math
import random

class Graph:
    def __init__(self,nodes,edges):
        self.nodes=nodes
        self.edges=edges
        self.map=[]
        self.distmap=[]
        self.primmap=[]
        self.visited = []
        self.totdist=0
        l=len(nodes)
        for i in range(l):
            self.map.append([])
            self.distmap.append([])
            self.primmap.append([])
            self.visited.append(0)
            for j in range(l):
                self.map[i].append(float(math.inf))
                self.primmap[i].append(float(math.inf))
                self.distmap[i].append(0)
        for item in self.edges:
            self.map[item[0]][item[1]]=item[2]
            self.map[item[1]][item[0]]=item[2]

    def generateCand(self,sel_nodes):
        cand_nodes=[]
        for item in sel_nodes:
            for i in range(len(self.map[item])):
                if not math.isinf(self.map[item][i]):
                    cand_nodes.append(i)
        return cand_nodes

    def Prim(self):
        sel_nodes=[]
        sel_edges=[]
        ininode=random.randint(min(self.nodes),max(self.nodes))
        sel_nodes.append(ininode)
        cand_nodes=self.nodes.copy()
        cand_nodes.remove(ininode)
        while len(cand_nodes)!=0:
```

```
            st=0;ed=0;dist=math.inf
            for stn in sel_nodes:
                for edn in cand_nodes:
                    if dist>self.map[stn][edn]:
                        st=stn;ed=edn
                        dist=self.map[stn][edn]
            sel_nodes.append(ed)
            cand_nodes.remove(ed)
            sel_edges.append((st,ed,dist))
        for item in sel_edges:
            self.primmap[item[0]][item[1]]=item[2]
            self.primmap[item[1]][item[0]]=item[2]
        for edge in sel_edges: self.totdist+=edge[2]
        return sel_edges

    def dfs(self,i,j,cur):
        for nex in self.nodes:
            if nex==i: continue
            if not math.isinf(self.primmap[cur][nex]) and self.visited[nex]!=1:
                self.visited[nex]=1
                dfs(i,j,nex)
                self.visited[nex]=0
                self.distmap[i][j]+=self.edges[cur][nex]

    def subdfs(self,cur,cut_map,visitedn):
        visitedn[cur]=1
        for nex in self.nodes:
            if not math.isinf(cut_map[cur][nex]) and visitedn[nex]!=1 and self.visited[nex]!=1:
                self.visited[nex]=1
                self.subdfs(nex,cut_map,visitedn)
                self.visited[nex]=0
                self.temdist+=cut_map[cur][nex]

    def generateChoice(self,sel_edges,del_num):
        res_edges=sel_edges.copy()
        for i in range(del_num):
            index=random.randint(0,len(res_edges)-1)
            res_edges.remove(res_edges[index])
        return res_edges

    def generateSubgraph(self,cutted_edges):
        cutted_map=[]
        visited_nodes=[]
        res_dists=[]
        l=len(self.nodes)
        for i in range(l):
            cutted_map.append([])
```

```
            visited_nodes.append(0)
            for j in range(l):
                cutted_map[i].append(float(math.inf))
        for item in cutted_edges:
            cutted_map[item[0]][item[1]]=item[2]
            cutted_map[item[1]][item[0]]=item[2]
        for node in self.nodes:
            if visited_nodes[node]==0:
                self.temdist=0
                self.visited[node]=1
                self.subdfs(node,cutted_map,visited_nodes)
                self.visited[node]=0
                res_dists.append(self.temdist)
        return res_dists

    def cutedge(self,sel_edges):
        # for item in sel_edges:
        #     self.visited[item[0]]=1
        #     self.visited[item[1]]=1
        #     self.dfs(item[0],item[1],item[1])
        #     self.visited[item[0]]=0
        #     self.dfs(item[1],item[0],item[0])
        #     self.visited[item[1]]=0
        itetime=1000
        i=0
        mintotdist=1000000
        minvar=1000000
        minweight=100000
        rescut=[]
        resdist=[]
        while i<itetime:
            temp_choice=self.generateChoice(sel_edges,7)
            res_dist=self.generateSubgraph(temp_choice)
            if 0 in res_dist:
                continue
            # print(i)
            # print(temp_choice)
            # print(res_dist)
            tempdist=sum(res_dist)
            tempvar=np.var(res_dist)
            # print('总和：',tempdist)
            # print('方差：',tempvar)
            mintotdist=min(mintotdist,tempdist)
            minvar=min(minvar,tempvar)
            if minvar==tempvar:
                rescut=temp_choice
                resdist=res_dist
```

```
            i+=1
        # print(mintotdist,minvar)
        # print(rescut)
        # print(resdist)
        return mintotdist,minvar,resdist,rescut

nodes=[0,1,2,3,4,5,6,7,8,9,
       10,11,12,13,14,15,16,17,18,19,
       20,21,22,23,24,25,26,27,28,29,
       30,31,32,33,34,35,36,37,38,39,
       40,41,42,43,44,45,46,47,48,49]

edgelist=[(0,1,1),(0,20,1),(1,2,1),(1,21,1),(1,22,1),(2,3,1),(2,23,1),(3,4,1),(4,5,1),(4,25,1),
(5,6,1),(5,25,1),(5,26,1),(6,7,1),(6,26,1),(6,27,1),(7,8,1),(7,29,1),(8,9,1),(8,32,1),
(9,10,1),(9,32,1),(9,33,1),(10,11,1),(10,34,1),(11,12,1),(12,13,1),(12,35,1),(12,36,1),(12,37,1),
(13,14,1),(13,37,1),(13,38,1),(14,15,1),(14,39,1),(15,16,1),(15,39,1),(16,17,1),(16,39,1),(17,18,1),
(17,40,1),(18,19,1),(18,22,1),(18,42,1),(19,20,1),(19,22,1),(19,42,1),(20,21,1),(21,22,1),(22,23,1),
(22,42,1),(22,43,1),(23,24,1),(23,43,1),(23,44,1),(23,47,1),(24,25,1),(24,44,1),(24,46,1),(24,48,1),
(25,26,1),(25,48,1),(25,49,1),(26,27,1),(26,28,1),(26,30,1),(26,49,1),(27,28,1),(28,29,1),(29,30,1),
(30,31,1),(30,32,1),(30,49,1),(31,32,1),(31,33,1),(31,34,1),(31,36,1),(31,45,1),(31,47,1),(32,33,1),
(33,34,1),(34,35,1),(35,36,1),(35,46,1),(35,47,1),(36,37,1),(36,45,1),(36,46,1),(37,38,1),(37,41,1),
(38,39,1),(38,40,1),(38,41,1),(39,40,1),(40,41,1),(41,42,1),(41,43,1),(42,43,1),(43,44,1),(44,45,1),
(45,46,1),(45,48,1),(46,47,1),(47,48,1),(48,49,1)]

g=Graph(nodes,edgelist)
md=1000000
mv=1000000
rescut=[]
resd=[]
#print(g.Prim())
for i in range(100):
    print(i)
    td,tv,tcut,tdis=g.cutedge(g.Prim())
    if tv<mv:
        md=td;mv=tv;rescut=tcut;resd=tdis
print(rescut)
print(resd)
print(md,mv)
print(1)
```

附录 B　用于模拟事故分布的 R 语言程序

```
#3
f <- function(x, p1 =1/3, p2 =1/3,p3=1/3 ,m1, m2)
    #p1 * ppois(x, m1) + p2 * ppois(x, m2)+p3*punif(x,0,300)
```

```
   p1 * dpois(x, m1) + p2 * dpois(x, m2)+p3*dunif(x,0,300)
#double
d  <- function(x, p1 =1/2, p2 =1/2,m1)
   #p1 * ppois(x, m1) + p2 * punif(x,0,300)
   p1 * dpois(x, m1) +p2*dunif(x,0,300)
x <- seq(0, 300, len=301)
dens <- f(x, p1 = 1/3,p2=1/3,p3=1/3, m1 = 42, m2 = 147)
#dens<- d(x,p1=1/3,p2=2/3,m1=42)
print(dens)
plot(x, dens, type = "l",ylab="概率",xlab="距离")
#write.csv(dens,"density2-3.csv")
```

附录 C　用于求解问题 3 的 python 程序

```
import numpy as np
import networkx as nx
import math
import random

class Graph:
   def __init__(self,nodes,edges):
      self.nodes=nodes
      self.edges=edges
      self.map=[]
      self.distmap=[]
      self.primmap=[]
      self.visited=[]
      self.degree=[]
      self.cutnode = []
      self.tempcutnode=[]
      self.keynode=[0,9,24,40,45,58,70,79]
      self.totdist=0
      l=len(nodes)
      for i in range(l):
         self.map.append([])
         self.distmap.append([])
         self.primmap.append([])
         self.visited.append(0)
         self.degree.append(0)
         for j in range(l):
            self.map[i].append(float(math.inf))
            self.primmap[i].append(float(math.inf))
            self.distmap[i].append(0)
      for item in self.edges:
         self.map[item[0]][item[1]]=item[2]
```

```
        self.map[item[1]][item[0]]=item[2]

    def generateCand(self,sel_nodes):
        cand_nodes=[]
        for item in sel_nodes:
            for i in range(len(self.map[item])):
                if not math.isinf(self.map[item][i]):
                    cand_nodes.append(i)
        return cand_nodes

    def Prim(self):
        sel_nodes=[]
        sel_edges=[]
        ininode=random.randint(min(self.nodes),max(self.nodes))
        sel_nodes.append(ininode)
        cand_nodes=self.nodes.copy()
        cand_nodes.remove(ininode)
        while len(cand_nodes)!=0:
            st=0;ed=0;dist=math.inf
            for stn in sel_nodes:
                for edn in cand_nodes:
                    if dist>self.map[stn][edn]:
                        st=stn;ed=edn
                        dist=self.map[stn][edn]
            sel_nodes.append(ed)
            cand_nodes.remove(ed)
            sel_edges.append((st,ed,dist))
        for item in sel_edges:
            self.primmap[item[0]][item[1]]=item[2]
            self.primmap[item[1]][item[0]]=item[2]
        for edge in sel_edges: self.totdist+=edge[2]
        return sel_edges

    def dfs(self,i,j,cur):
        for nex in self.nodes:
            if nex==i: continue
            if not math.isinf(self.primmap[cur][nex]) and self.visited[nex]!=1:
                self.visited[nex]=1
                dfs(i,j,nex)
                self.visited[nex]=0
                self.distmap[i][j]+=self.edges[cur][nex]

    def subdfs(self,cur,cut_map,visitedn):
        visitedn[cur]=1
        for nex in self.nodes:
            if not math.isinf(cut_map[cur][nex]) and visitedn[nex]!=1 and self.visited[nex]!=1:
                self.tempcutnode.append(nex)
```

```
            self.visited[nex]=1
            self.subdfs(nex,cut_map,visitedn)
            self.visited[nex]=0
            self.temdist+=cut_map[cur][nex]

    def judgeavai(self):
        for lis in self.cutnode:
            flag=0
            for item in self.keynode:
                if item in lis and flag==0:
                    flag=1
                elif item in lis and flag==1:
                    return False
        return True

    def effunc(self,res_dist):
        eff=0
        for lis in self.cutnode:
            for item in lis:
                eff+=self.degree[item]/res_dist[item]
        return eff

    def generateChoice(self,sel_edges,del_num):
        res_edges=sel_edges.copy()
        for i in range(del_num):
            index=random.randint(0,len(res_edges)-1)
            res_edges.remove(res_edges[index])
        return res_edges

    def generateSubgraph(self,cutted_edges):
        cutted_map=[]
        visited_nodes=[]
        res_dists={}
        self.cutnode=[]
        l=len(self.nodes)
        for i in range(l):
            cutted_map.append([])
            visited_nodes.append(0)
            self.degree[i]=0
            for j in range(l):
                cutted_map[i].append(float(math.inf))
        for item in cutted_edges:
            self.degree[item[0]]+=1
            self.degree[item[1]]+=1
            cutted_map[item[0]][item[1]]=item[2]
            cutted_map[item[1]][item[0]]=item[2]
        for node in self.nodes:
```

```
            self.tempcutnode=[]
            if visited_nodes[node]==0:
                self.tempcutnode.append(node)
                self.temdist=0
                self.visited[node]=1
                self.subdfs(node,cutted_map,visited_nodes)
                self.visited[node]=0
                for itm in self.tempcutnode:
                    res_dists.update({itm:self.temdist})
                self.cutnode.append(self.tempcutnode.copy())
        return res_dists

    def cutedge(self,sel_edges):
        itetime=100
        i=0
        tar=0
        rescut=[]
        resdist={}
        while i<itetime:
            temp_choice=self.generateChoice(sel_edges,7)
            res_dist=self.generateSubgraph(temp_choice)
            if (0 in res_dist.values()) or (not self.judgeavai()):
                continue
            if self.effunc(res_dist)>tar:
                tar=self.effunc(res_dist)
                rescut=temp_choice
                resdist=res_dist
            i+=1
        print(tar)
        print(rescut)
        print(resdist)
        return tar,resdist,rescut

nodes=[0,1,2,3,4,5,6,7,8,9,
       10,11,12,13,14,15,16,17,18,19,
       20,21,22,23,24,25,26,27,28,29,
       30,31,32,33,34,35,36,37,38,39,
       40,41,42,43,44,45,46,47,48,49,
       50,51,52,53,54,55,56,57,58,59,
       60,61,62,63,64,65,66,67,68,69,
       70,71,72,73,74,75,76,77,78,79]

edgelist=[(0,1,5),(0,10,5),(1,2,5),(1,11,9),(2,3,5),(2,12,9),(3,4,5),(3,13,9),
          (4,5,5),(4,14,9),(5,6,5),(5,15,9),(6,7,5),(6,16,9),(7,8,5),(7,17,9),
          (8,9,5),(8,18,9),(9,19,5),(10,11,5),(10,20,6),(11,12,5),(11,21,6),
          (12,13,5),(12,22,6),(13,14,5),(13,23,6),(14,15,5),(14,24,6),(15,16,5),
          (15,25,6),(16,17,5),(16,26,6),(17,18,5),(17,27,6),(18,19,9),(18,28,5),
```

```
        (19,29,5),(20,21,5),(20,30,5),(21,22,5),(21,31,9),(22,23,5),(22,32,9),
        (23,24,6),(23,33,5),(24,25,5),(24,34,5),(25,26,5),(25,35,9),(26,27,5),
        (26,36,9),(27,28,6),(27,37,5),(28,29,9),(28,38,5),(29,39,5),(30,31,5),
        (30,40,6),(31,32,5),(31,41,6),(32,33,9),(32,42,5),(33,34,6),(33,43,5),
        (34,35,9),(34,44,5),(35,36,5),(35,45,5),(36,37,9),(36,46,5),(37,38,6),
        (37,47,5),(38,39,9),(38,48,5),(39,49,5),(40,41,5),(40,50,5),(41,42,5),
        (41,51,9),(42,43,9),(42,52,9),(43,44,6),(43,53,5),(44,45,9),(44,54,5),
        (45,46,6),(45,55,5),(46,47,9),(46,56,5),(47,48,6),(47,57,5),(48,49,9),
        (48,58,5),(49,59,5),(50,51,5),(50,60,6),(51,52,5),(51,61,6),(52,53,9),
        (52,62,5),(53,54,6),(53,63,5),(54,55,9),(54,64,5),(55,56,6),(55,65,5),
        (56,57,9),(56,66,5),(57,58,6),(57,67,5),(58,59,9),(58,68,5),(59,69,5),
        (60,61,5),(60,70,5),(61,62,5),(61,71,9),(62,63,9),(62,72,9),(63,64,5),
        (63,73,9),(64,65,9),(64,74,9),(65,66,6),(65,75,5),(66,67,9),(66,76,5),
        (67,68,5),(67,77,9),(68,69,9),(68,78,9),(69,79,5),(70,71,5),(71,72,5),
        (72,73,5),(73,74,5),(74,75,5),(75,76,6),(76,77,5),(77,78,5),(78,79,5)]

g=Graph(nodes,edgelist)
mtar=0
rescut=[]
resd=[]
for i in range(100):
    print(i)
    tar,tcut,tdis=g.cutedge(g.Prim())
    if tar>mtar:
        mtar=tar;rescut=tcut;resd=tdis
print(rescut)
print(resd)
print(mtar)
print(1)
```