基于用电可靠性的配电网规划模型

摘要

本题是基于用电可靠性的 10KV 低压配电网规划问题。我们建立了**双层规划**模型、用户可靠性评估模型和孤岛划分模型并通过**遗传-模拟退火算法**、最小生成树算法、动态规划算法解决了不同情况下的配电网规划问题。

针对问题一：以配电网建造费用作为总规划目标，建立**双层规划模型**。对于上层主线规划模型，首先通过 **K-means 聚类算法**对负荷进行分类，以此确定分支数量，并将聚类中心作为初始二级分叉点。通过**遗传-模拟退火算法**得到主线的最佳规划方案，并得到一级分叉点位置；对于下层支线规划模型，进行**二次嵌套聚类**，运用等效电源和加权距离的思想建立模型，得到支线的最佳规划和二三级分叉点的位置。通过双层规划迭代优化二级分叉点位置，得到最佳的单供配电网规划。运用苏州工业园实例中的数据，得到单供配电网**最低建设费用结果为 3503.13 千元**，并计算出各个用户的用电可靠性。

针对问题二：综合考虑聚类中心点的聚集程度以及电源的相对位置关系，采用**最小生成树算法**来确定各个负荷点的供电归属问题。随后使用问题一建立的模型和苏州工业园实例中的数据得到两个单供配电网最低建设总费用为 **3720.14 千元，并计算出各用户的用电可靠性。**

针对问题三：以提高配电网中最低的用电可靠性为目标；以建造总费用上限为 X，双供配电网用户供电调度原则，主线限流条件为约束条件建立优化模型。首先对配电网进行**最小区域划分**。然后，针对不同的故障情况进行动态孤岛划分；建立了**孤岛划分模型，**采用**动态规划**的算法，得到表征不同故障发生时负荷点供电情况的关联矩阵。最后，我们改进了问题一二中的可靠性评估模型，得到**双供配电网用电可靠性评估模型**，以提高配电网中最低的用电可靠性为目标，讨论不同联络线情况下的最佳方案。

针对问题四：延用双供配电网用电可靠性评估模型，以建设费用最小为优化目标；以每个用户的用户可靠性不低于 Y%，双供配电网用户供电调度原则，主线限流条件为约束条件建立模型，得到最佳方案。

关键词：双层规划 遗传-模拟退火算法 配电网规划 动态规划 用电可靠性

1 问题重述

1.1 背景分析

随着我国发展，供电的需求越来越大，配电网规划的重要性与日俱增。配电网在电力系统中起着承上启下的作用，承担着从发电站获得电源并向不同的用户分配的作用。在已知电站和用户地理位置的条件下，合理地规划电网能极大地减少线路和开关的铺设费用并提高用户的用电可靠性。

1.2 相关信息

在实际的配电网规划中，常常会出现多级分叉点问题，即在分叉点的某条支路上仍存在其他分叉点。为了更好地分析问题，现对分叉点进行简单的定义，在后文中会有更加详细的讨论，详见 5.1.1 节。现定义，在主线上的分叉点为一级分叉点；由一级分叉点引出的支线与其他支线的交点为二级分叉点；同理可得三级分叉点的定义。

1.3 问题提出

本题要求建立数学模型并回答以下问题：

问题一：已知一个电源和一批用户的平面坐标、每个用户用电功率需求、每个设备单元建造费用。以费用最低为目标求单供树状配电网，给出树状配电网的分叉点坐标，并计算每个用户的用电可靠性。

问题二：在问题一的基础上增加一个电源。以费用最低为目标设计两个单供树状配电网，给出树状配电网的分叉点坐标，并计算每个用户的用电可靠性。

问题三：在问题二的两个单供配电网之间建立若干条联络线形成双供配电网，并扩充供电功率。以建造双供配电网总花费上限为 X 为约束条件，以最低的用电可靠性达到最大值为优化目标，设计联络线和开关。画出联络线拓扑简略图，并计算双供配电网中每个用户的用电可靠性。

问题四：在问题二的两个单供配电网之间建立若干条联络线形成双供配电网，并扩充供电功率。以双供配电网中每个用户的用电可靠性不低于 Y%为约束条件，

以建设费用最低为优化目标，设计双供配电网。画出联络线拓扑简略图，并计算双供配电网中每个用户的用电可靠性。

2 问题分析

2.1 问题一的分析

这是一个单电源配电网规划问题。配电网建造费用作为总规划目标；开关设置原则，开关价格与限流，单位线路造价，变电站与负荷的平面坐标为约束条件。为简化问题，我们将配电网规划拆解成两部分规划：主线规划和支线规划，建立双层规划模型。

上层规划以主线为优化对象；基于负荷的坐标，我们通过聚类算法对负荷进行分类，以此确定分支数量，并将聚类中心作为初始二级分叉点。在此基础上，通过遗传-模拟退火算法得到主线的最佳规划，并得到一级分叉点位置。

下层规划以支线为优化对象，对每个簇进行二次嵌套聚类分析，运用加权距离和等效电源的思想建立模型，得到支线的最佳规划和二三级分叉点的位置。

上层规划和下层规划并不是严格独立的，上下层的结果会相互影响。我们通过不断迭代优化二级分叉点的位置，最终得到最佳的配电网规划和最小建设费用。

最后我们建立用户可靠性评估模型，计算出每个故障单元的故障率，根据故障单元的独立性，最终算出每个用户的用电可靠性。

2.2 问题二的分析

问题二要求设计两个单供配电网，使得总建造费用最低。我们需要先解决负荷的供电归属问题，再延用问题一的思路和模型设计两个单供配电网。

首先，我们要解决负荷的供电归属问题，即确定负荷点由电源 1 还是电源 2 供电。当聚类簇数为k时，总共会出现2^k种情况，排除由某一个电源为所有负荷供电的情况，仍有2^k-2种情况。若直接采用第一问的模型对所有情况进行遍历计算，计算量将十分巨大。由于最小生成树的边长总和一定程度上能反映负荷与电源的聚集程度，因此可以采取比较最小生成树长度总和的方法来简化计算量，

选取出几种总长度较小的情况做后续的优化。

然后，我们分别对第一步筛选出几种负荷归属方案，使用问题一的双层规划模型，得到多种规划方案。我们将总造价最低的方案确定为最优方案。

最后，我们应用第一问的可靠性评估模型计算出各个用户的用电可靠性。

2.3 问题三的分析

在问题二两个单供配电网规划的基础上，新增若干条联络线和开关，形成双供配电网。我们以提高配电网中最低的用电可靠性为目标；以建造总费用上限为X，双供配电网用户供电调度原则和主线限流条件为约束条件建立模型。

首先，我们需要确定联络线的个数。由于总花费上限没有给出具体数据，因此我们对一条联络线，两条联络线，三条联络线的情况分别进行优化并得出建造总费用。

其次，我们需要确定联络线的位置。我们通过配电网的最小区域划分，对联络线的连接位置进行遍历。

对每种联络线设计方案，我们针对不同的故障情况进行动态孤岛划分。为了满足双供配电网用户供电调度原则，依据划分的最小配电区域，我们通过关联矩阵表征不同故障发生时负荷点的供电情况，采用动态规划的算法，实现孤岛划分。

最后，我们建立双供配电网可靠性评估模型。针对双供配电网多种供电线路方案，对问题一，二中的用户可靠性模型进行改进，得到双供配电网的可靠性评估模型，计算出每种联络线规划的最低的用户可靠性，选择最低用户可靠性最大的方案作为最佳方案。

2.4 问题四的分析

问题四和问题三同理，改变了问题三的约束条件与优化目标。以降低建设费用为优化目标；以每个用户的用户可靠性不低于 Y%，双供配电网用户供电调度原则和主线总限流为约束条件。

首先，我们需要确定联络线的个数。同样由于最低的用户可靠性数据未定，因此我们只讨论连接一条联络线和两条联络线的情况并分别得出建造总费用。

其次，在确定了联络线的数量后，我们仍按照问题三的思路对联络线的连接

位置进行遍历，用关联矩阵去表征不同故障发生时的负荷点供电情况。

最后，我们通过双供配电网用户用电可靠性评估模型，计算出每种联络线规划方案的最低建造费用，选择建造费用最低的方案作为最佳方案。

3 模型假设

为了简化模型，本文做出以下合理的假设或条件约束：

1. 假设在电网运行过程中不产生线路损耗的费用。
2. 假设存在一个分叉点下可能还会有其他分叉点的情况，为了简化模型本文只考虑到三级分叉点。
3. 假设每个单元设备是否故障是相互独立的。
4. 假设开关在线路中可视为一个理想的质点。
5. 假设联络线只能从主线连接到主线。
6. 假设一级分叉点连接两个二级分叉点时，将该一级分叉点视为两个无限接近的一级分叉点。

4 符号约定

参数名称	说明	单位
k	聚类簇数	个
n	用户(负荷)的总数	个
$C_{mainline}$	建造主线的总费用	千元
$C_{spurline}$	建设支线的总费用	千元
C_{switch}	建设开关的总费用	千元
C_{single}	单供树状配电网的最小建设费用	千元
$x_{mainline}$	主线的总长度	千米
L_j	第j个最小供电单元	
P_i	第i个用户的功率需求量	兆瓦
m	因故障由备用电源供电的用户数目	个
P_{extend}	备用电源的扩展功率	兆瓦
M	供电关联矩阵	
R_i	第i个用户的用电可靠性	
$C_{connect}$	建设联络线的费用	千元
C	总建设费用	千元

注：在具体的模型中，将对符号进行分别说明

5 模型的建立与求解

5.1 问题一模型的建立与求解

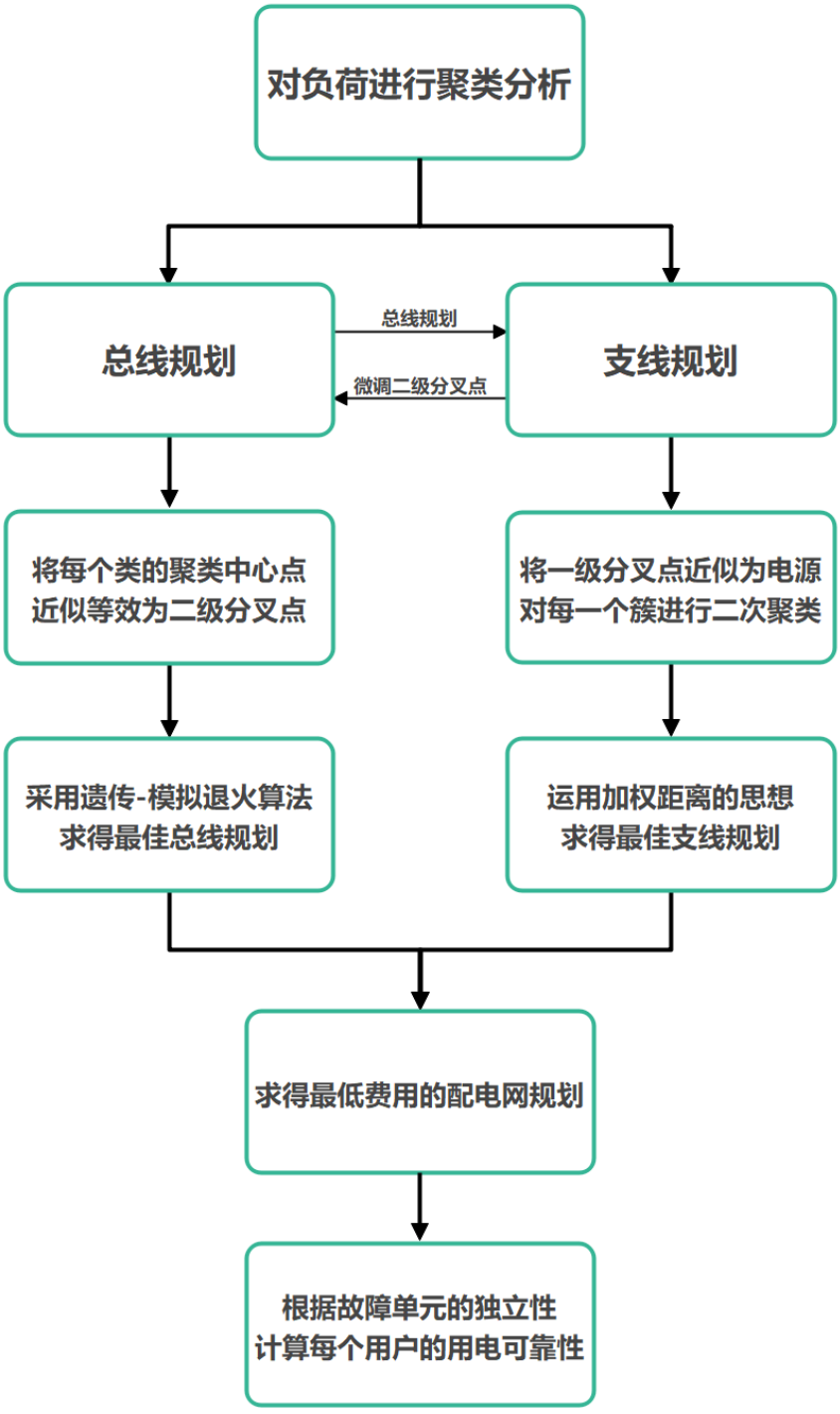


图 1 问题一思路图

5.1.1 多级分叉点情况讨论

为方便叙述，以图 2 为例，引入一级分叉点，二级分叉点，三级分叉点，一级支路，二级支路，三级支路的概念。

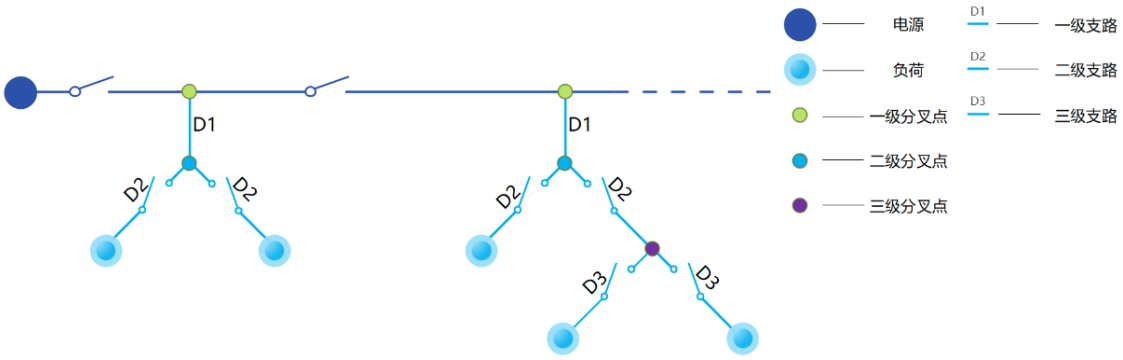


图 2 多级分叉点示意图

本题中，一个单供电源最多只能为 50 个负荷供电，因此我们认为当一级分叉点、二级分叉点和三级分叉点数量较多时，连接的负荷数量大于 50，出现四级分叉点的概率非常小。本模型能够处理多级分叉点问题，但会增加计算量和求解的时间。因此，在本文中为了简化模型并提高算法的效率，假设最多只存在三级分叉点。

5.1.2 数据获取与预处理

首先，我们结合相关文献[1]75，参考苏州工业园区的实例，得到了 3 个电源变电站和 24 个负荷的具体坐标，并在此基础上增加了 11 个负荷点，详见附录 1。当研究单供问题时，则使用其中一个电源变电站的具体坐标数据。

由于配电网规划需要考虑电源和负荷点的坐标，因此我们考虑先对负荷点进行 K-means 聚类分析。根据聚类中心点的定义，簇内各点到它的距离平方和是最小的，我们认为聚类中心点对于决定分叉点的位置和数目有一定的参考价值，因此不妨将其设为初始的二级分叉点。

在确定聚类簇数时，由于轮廓系数的大小只能反映聚类的效果，聚类最佳结果并不能与最小建设费用等价。因此我们对不同的聚类簇数分别应用问题一的模型进行计算，最后根据费用确定最佳的聚类簇数。

5.1.3 双层规划模型的框架

本文针对单供配电网建立双层规划模型[1]30。

上层为主线规划模型，以主线建造费用，主线开关费用以及一级支路的建造费用之和最低为目标，决策变量为主线规划方案和一级分叉点的位置方案。

下层为支线规划模型，以支线建造费用和负荷前开关建造费用综合之和最低为目标函数，决策变量为支线规划方案。

上层规划确定了主线规划方案和一级分叉点的位置，并将其传递至下层；下层在上层的规划基础上，对二级分叉点的位置进行优化，并将优化结果反馈给上层，上层模型再根据下层的反馈调整主线规划方案和一级分叉点的位置，最终求得最优单供配电网规划方案。

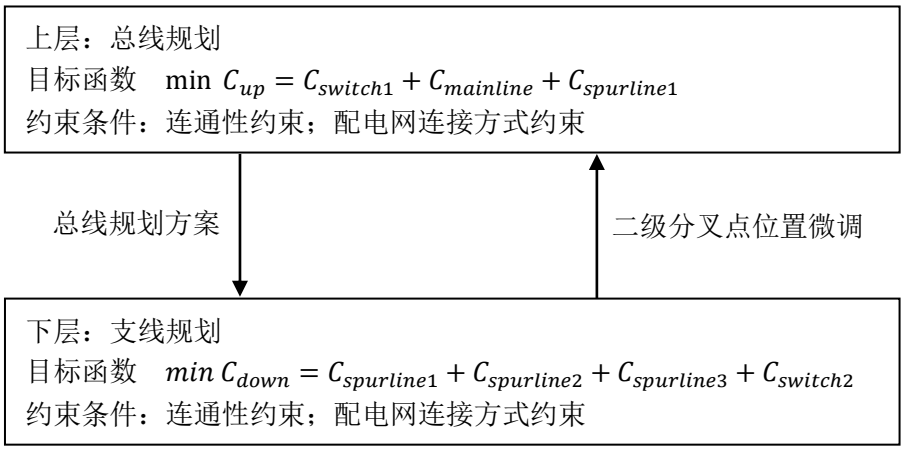


图 3 双层规划框架图

5.1.4 双层规划模型的建立

(1) 主线规划模型的建立

上层模型即主线规划模型负责确定主线的长度和一级分叉点的位置，并将规划方案传递给下层模型。其具体的优化函数如下：

$$min \quad C_{up} = C_{switch1} + C_{mainline} + C_{spurline1} \qquad (5-1)$$

其中，各个费用的含义如下表所示：

表 1 主线规划模型变量含义及计算表

变量公式	含义
$C_{switch1} = k \cdot U_{switch1}$	主线上开关的总费用
$C_{mainline} = U_{mainline} \cdot x_{mainline}$	建造主线的总费用
$C_{spurline1} = \sum U_{spurline} \cdot x_{i,1}$	建造一级支路总费用
$U_{spurline} = \begin{cases} 188.6, & n_{spurline} \le 2 \\ 239.4, & n_{spurline} \ge 3 \end{cases}$	每公里支线的建造费用
k	聚类簇数
$U_{switch1}$	一个主线开关的建造费用
$U_{mainline}$	每公里主线的建造费用
$x_{mainline}$	主线的总长度
$x_{i,1}$	某条支线一级支路的长度
$n_{spurline}$	某条支线承载的负荷数量

结合配电网规划的实际情况和开关的设置规则，上层模型的约束条件包括：

① 连通性约束。要求在配电网上任意一点到电源的所有路中，可以通过设置开关状态使得仅有一条是通路。

② 配电网接线方式的约束。最终要形成树状配电网。

(2) 支线规划模型的建立

下层模型即支线规划模型负责确定二级分叉点和三级分叉点的位置，并将规划后的结果反馈给上层，使上层的模型根据更优的二级分叉点的位置继续规划，反复迭代后即可找到最优解。其具体的优化函数如下：

$$min \quad C_{down} = C_{spurline1} + C_{spurline2} + C_{spurline3} + C_{switch2} \qquad (5-2)$$

其中， 各个费用的含义如下表所示：

表 2 支线规划模型变量含义及计算表

变量公式	含义
$C_{spurline1} = \sum U_{spurline} \cdot x_{i,1}$	建造一级支路总费用
$C_{spurline2} = \sum U_{spurline} \cdot x_{i,2}$	建造二级支路总费用
$C_{spurline3} = \sum U_{spurline} \cdot x_{i,3}$	建造三级支路总费用

变量公式	含义
$C_{switch2} = n \cdot U_{switch2}$	用户前开关的总费用
$x_{i,2}$	某条支线二级支路的长度
$x_{i,3}$	某条支线三级支路的长度
n	总负荷数
$U_{switch2}$	一套负荷前开关的建造费用

结合配电网规划的实际情况和开关的设置规则，下层模型的约束包括：

① 连通性约束。要求在配电网上任意一点到电源的所有路中，可以通过设置开关状态使得仅有一条是通路。

② 配电网接线方式的约束。最终要形成树状配电网。

双层规划模型具体步骤如下：

上层规划：

Step1:对负荷根据坐标点进行聚类。

Step2:将聚类中心近似等效为二级分叉点，通过遗传-模拟退火算法得到主线规划。

下层规划：

Step3:对负荷二次嵌套聚类，通过加权距离计算，得到一级支线的规划。

Step4:将二次聚类中心近似等效为三级分叉点，得到二级支线的规划。

Step5:连接负荷与三级分叉点，得到初步的主线规划和支线规划。

综合迭代：

Step6:微调二级分叉点坐标，迭代双层规划，得到最佳单供配电网规划。

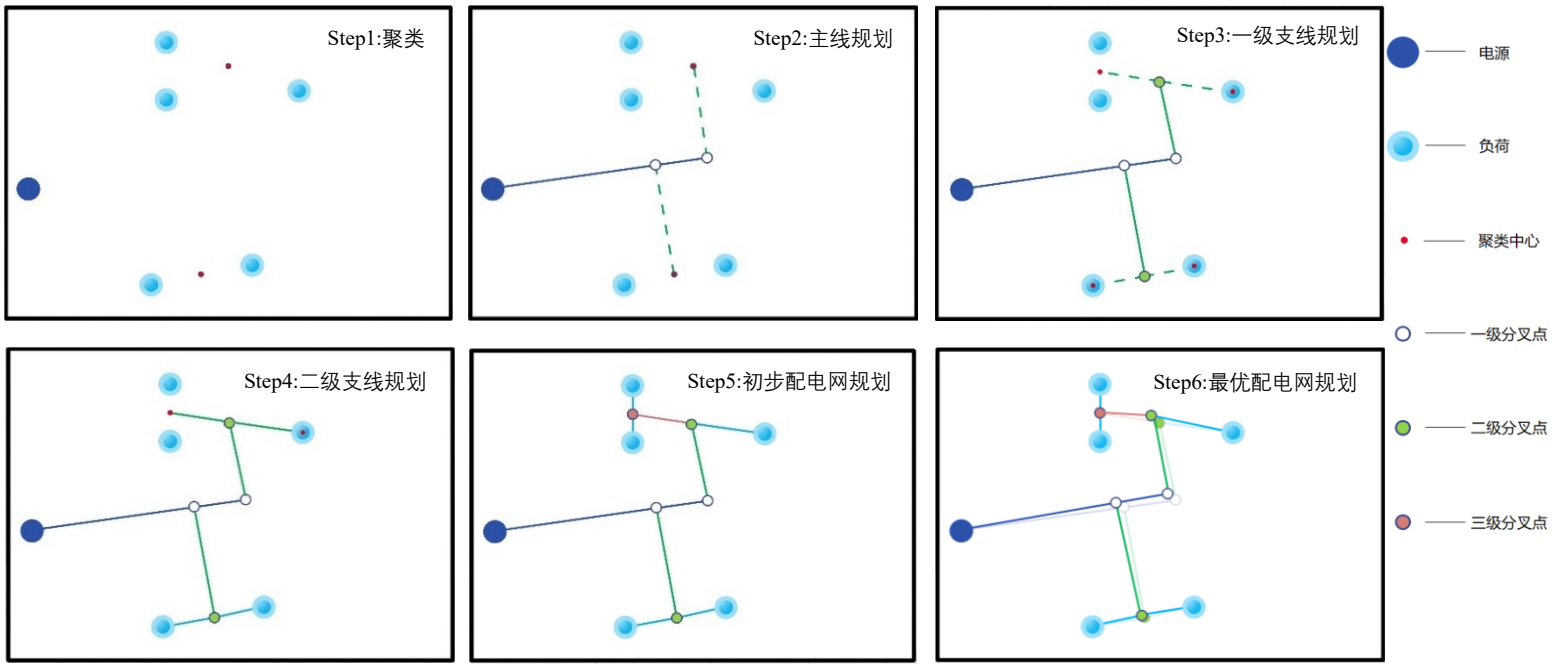


图 4 双层规划过程图

注：六张图分别对应六个步骤。

5.1.5 单供树状配电网最小建设费用模型

根据双层模型的最优解，我们可以求得单供树状配电网的最小建设费用。其表达式如下：

$$C_{single} = C_{switch} + C_{mainline} + C_{spurline} \quad (5-3)$$

各类费用的具体计算公式如下：

$$C_{switch} = C_{switch1} + C_{switch2} \quad (5-4)$$

$$C_{spurline} = C_{spurline1} + C_{spurline2} + C_{spurline3} \quad (5-5)$$

各个公式的具体内容详见 5.1.4，表 1，表 2。

5.1.6 单供配电网用电可靠性评估模型

我们假定每个单元设备是否发生故障是相互独立的，因此，对于某个用户，我们需要找出电源到该用户之间的唯一供电通路。该用户的用电可靠性即为供电通路上的所有故障单元设备正常运行的概率的乘积。

因此，第i个用户的用电可靠性计算公式如下：

$$R_i = R_{power} \cdot R_{user} \cdot R_{switch} \cdot R_{line} \qquad (5-6)$$

其中，变量的含义如下：

表 3 单供可靠性模型变量含义及计算表

变量公式	含义
$R_{power} = 1 - 0.5\% = 0.995$	电源正常运行的概率
$R_{user} = 1 - 0.5\% = 0.995$	用户正常运行的概率
$R_{switch} = (1 - 0.2\%)^{t_i}$	电源到第i个用户的供电通路上所有开关都正常运行的概率
$R_{line} = \prod_{j=1}^{y}(1 - s_{i,j} \cdot 0.002)$	电源到第i个用户的供电通路上所有故障单元路都正常运行的概率
t_i	电源到第i个用户的供电通路上所有的开关个数
$s_{i,j}$	第i个用户第j条故障单元边的长度

5.1.7 问题一模型的求解

(1) K-means 聚类分析的结果

我们采用 K-means 算法求解，起初把轮廓系数作为评判标准来决定聚类后的簇数。但我们同时也考虑到聚类的最佳结果并不能与最小建设费用等价。

于是我们采用遍历的思想，探究了不同k值情况下建设费用的变化情况。由于在求解过程中我们发现双层规划两次后建设费用的波动较小，故只比较双层规划两次后的建设费用的变化情况。最终，当k值取到 7 时，其建设费用明显小于其他情况。如下图所示：

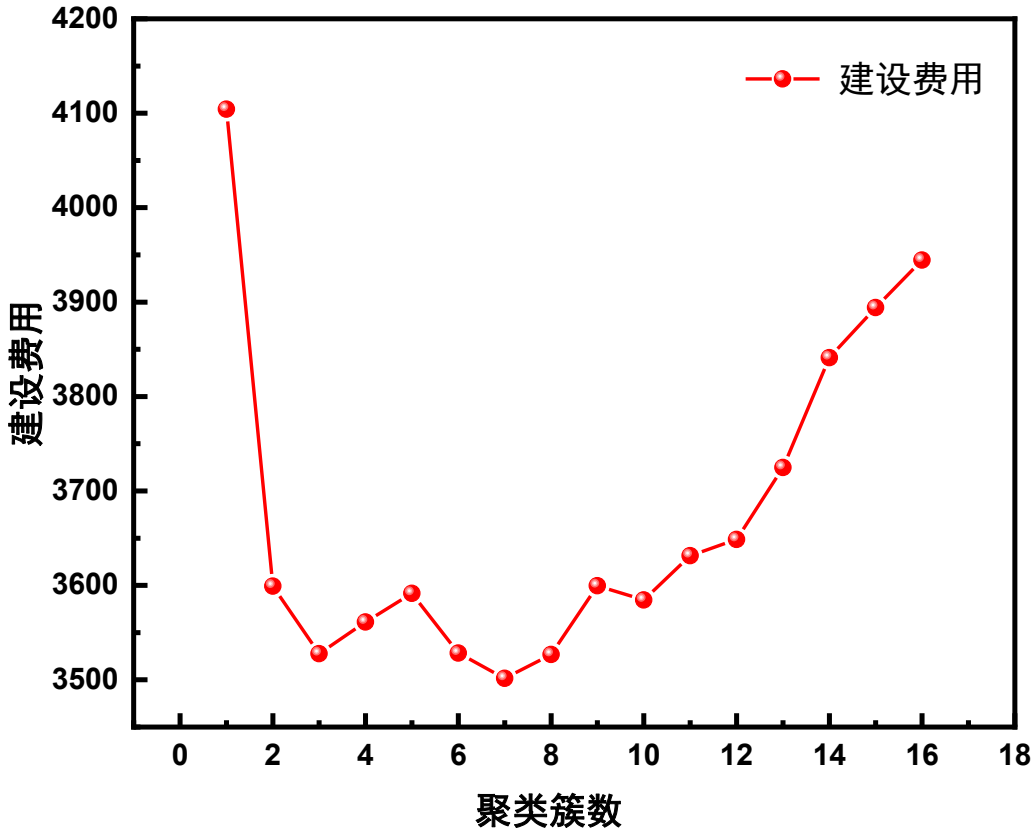


图 5 聚类簇数与建设费用关系图

(2) 双层规划模型与单供树状配电网最小建设费用的求解

双层规划问题是一个 NP-hard 问题，普通的算法求解起来较为复杂，而模拟退火算法和遗传算法是启发式算法，是当前求解双层规划问题的有效手段[2]。但对于上层主线规划模型，两个单一的算法存在着各自的缺陷，为了更快地得到更优的解，我们将遗传算法和模拟退火算法相结合，用遗传-模拟退火算法去求解。它综合了模拟退火算法搜索面广而遗传算法收敛较快的优点，有效地克服了模拟退火算法收敛速度较慢而遗传算法收敛易早熟的问题[3]。

先用遗传算法快速搜索出较优的解，再用模拟退火算法对局部进行搜索，得到上层的最优规划。其具体步骤如下：

Step1:初始化种群的规模、遗传的代数、交叉和变异的概率。

Step2:初始化第一代的种群，并计算第一代种群中每个个体的适应度函数值，将其从大到小排序。

Step3:按适应度从大到小确定染色体产生第二代的种群。

Step4:按一般的遗传算法对第二代进行交叉变异更新种群，选取最优的种群。

Step5:选取种群中最佳的个体，并采用模拟退火算法优化，直至获得最优解。

我们分别用模拟退火算法、遗传算法和遗传-模拟退火算法进行了计算。并对其优缺点、运行时间和规划结果进行比较，如表 4 所示：

表 4 三种算法优缺点比较表

算法	稳定性	全局搜索能力	局部搜索能力	收敛时间	规划结果(千元)
遗传算法	弱	强	弱	短	1678.14
模拟退火算法	较弱	较弱	强	较长	1646.01
遗传-模拟退火算法	强	强	强	短	1613.39

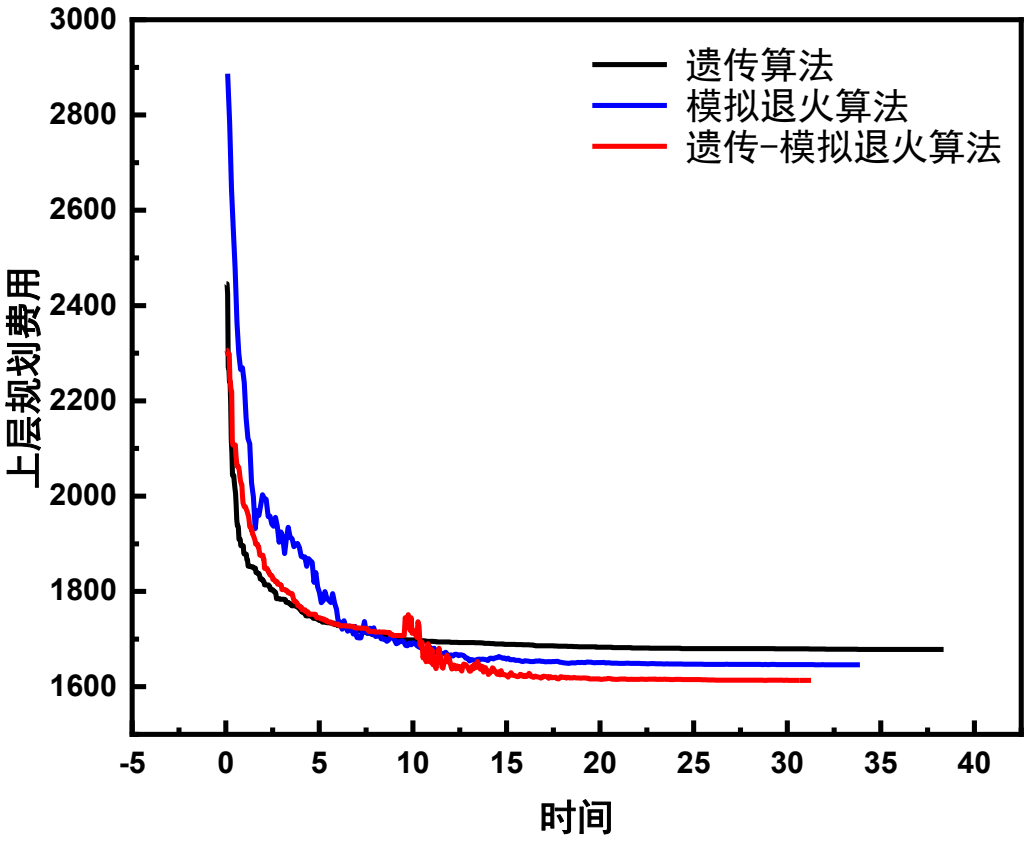


图 6 三种算法比较

可以看出遗传-模拟退火算法可以在更短的运行时间中，得到的更优更精确的解，比单纯的模拟退火算法和遗传算法搜索能力更强。

对于下层支线规划模型，我们采用二次嵌套聚类以及加权距离的方法求解。我们分析了上层模型的结果，大部分一级分叉点只会与一个二级分叉点连接；存在个别一级分叉点会与两个二级分叉点连接，此时我们视为两个一级分叉点重合

了。我们采用“二次嵌套聚类”、“等效电源”和“加权距离”思想来求解下层模型。同时，我们在求解时直接设出优化后的二级分叉点坐标，用其来表示下层建设费用，大大地减少了计算量。其具体步骤如下：

Step1:将一级分叉点视为“电源”，对每个簇进行二次嵌套聚类分析。

Step2:在二次嵌套聚类分析后，得到的聚类中心点为三级分叉点。

Step3:设出各个优化后的二级分叉点坐标。

Step4:通过每条线段承载的负荷数量来确定其权重。

Step5:计算加权距离下的下层最优规划。

下层得到最优规划后，将信息反馈给上层，上层根据信息调整并重新规划。将上述过程反复迭代，得到最优解，最后依次将所有费用加到一起，即可得到问题一的总建设费用。

根据以上的算法，我们得到问题一的结果如下：

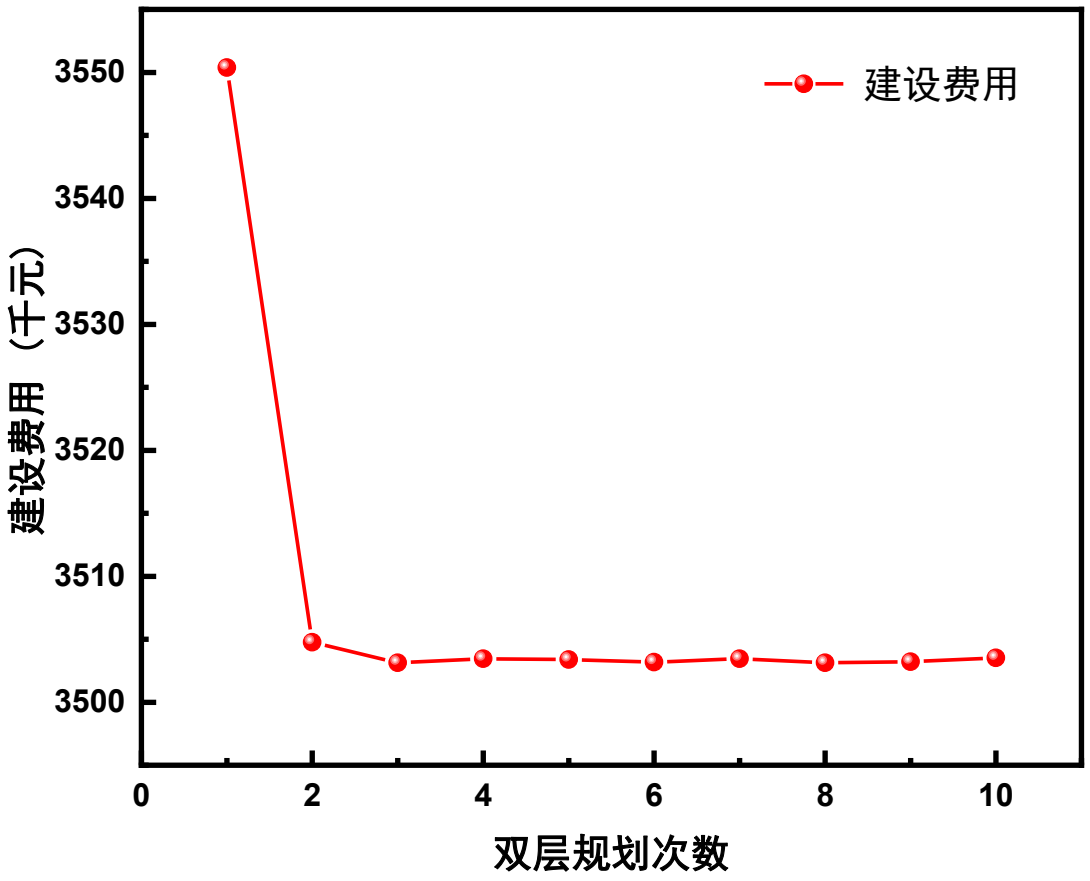


图 7 双层规划结果图

由图 7 可知，第一次双层规划后能明显使建设费用下降，之后的双层规划会让建设费用在 3503 千元附近上下波动，这是启发式算法的本身不稳定的特点导致的。且波动的范围较小，引起的误差小。最终，我们在规划十次的结果中选出最小的建设费用，为 3503.13 千元。

该结果下对应的分叉点坐标如下表所示：

表 5 问题一分叉点坐标表

分叉点序号	分叉点等级	分叉点横坐标	分叉点纵坐标	分叉点序号	分叉点等级	分叉点横坐标	分叉点纵坐标
1	一级	-0.3185	1.3341	19	三级	-0.1000	1.0000
2	一级	0.1388	1.0972	20	三级	0.9400	0.9800
3	一级	0.4464	0.8790	21	三级	0.5400	0.9400
4	一级	0.5050	0.4040	22	三级	1.0700	1.1400
5	一级	0.2093	0.0009	23	三级	0.7500	1.2050
6	一级	0.1127	-0.2552	24	三级	0.9800	1.3500
7	二级	-0.6225	1.0725	25	三级	0.4650	-0.9300
8	二级	0.1390	1.0976	26	三级	0.7600	-0.4700
9	二级	0.8835	1.1328	27	三级	0.3500	-1.2800
10	二级	0.4650	-0.9300	28	三级	0.0000	0.1500
11	二级	0.2008	0.0012	29	三级	0.2000	-0.0900
12	二级	-0.5000	-0.5000	30	三级	0.2500	0.0200
13	二级	0.7633	0.3433	31	三级	-0.4500	-0.8600
14	三级	-0.6225	1.0725	32	三级	-0.0300	-0.5350
15	三级	-0.4800	0.5550	33	三级	-0.6000	-0.2400
16	三级	0.3900	1.2000	34	三级	-0.5000	-0.5000
17	三级	0.1150	1.4200	35	三级	0.7633	0.3433
18	三级	0.1500	0.9500	36	三级	0.7800	-0.1800

(3) 用户用电可靠性模型的求解

在双层规划模型求解出最优解后，可以直接求出各个用户的用电可靠性，具体结果如下表所示：

表 6 问题一用户用电可靠性表

用户序号	用电可靠性	用户序号	用电可靠性
1	98.316%	19	96.612%
2	98.057%	20	97.422%
3	98.084%	21	96.782%
4	98.090%	22	96.643%
5	98.082%	23	96.869%
6	98.090%	24	96.816%
7	97.689%	25	96.713%
8	97.722%	26	96.869%
9	97.722%	27	98.350%
10	97.719%	28	98.319%
11	97.716%	29	98.343%
12	97.716%	30	98.223%
13	97.734%	31	98.057%
14	98.223%	32	96.612%
15	97.509%	33	97.505%
16	96.659%	34	97.274%
17	97.233%	35	97.509%
18	97.265%		

5.2 问题二模型的建立与求解

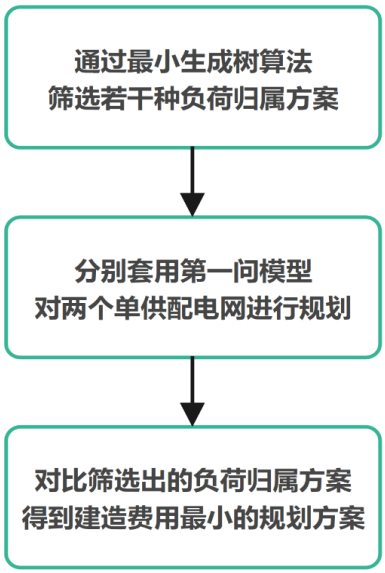


图 8 问题二思路图

5.2.1 数据获取与预处理

问题二考虑的是两个电源的情况，我们依旧采用苏州工业园的案例，我们从三个电源中随机选取两个，仍选择原来的 35 个负荷。同理，我们对 35 个负荷进行聚类分析，把聚类中心点假设为初始的二级分叉点。若出现某个聚类中心点为一级分叉点的情况，只需视为一二级分叉点之间的距离为 0。

5.2.2 问题二模型的建立

问题二中最关键的问题负荷的供电归属问题。当聚类簇数为k时，总共会出现2^k种情况，排除由某一个电源为所有负荷供电的情况，仍有2^k-2种情况。若直接对这2^k-2种情况运用问题一的模型进行优化，计算量将呈指数增长。因此，我们提出一种比较最小生成树长度总和的方法来简化计算量。我们用两个最小生成树将两个电源与其对应的聚类簇中心相连接，计算其两个树的边长长度总和。总和长度在一定程度上反映各个聚类中心点之间的聚集程度以及与电源的相对位置关系，总和长度较短的情况下更有可能得到最小的建设费用，其具体的优化函数如下：

$$min \quad X = X_1 + X_2 \tag{5-7}$$

其中，X_1和X_2分别表示电源 1 和电源 2 的最小生成树长度总和。

我们比较不同情况下的长度，选取距离较短的几种情况做后续的优化，这能极大地减少计算量。

当确定了负荷点由哪个电源供电后，只需对两个电源分别使用问题一的模型，最后将两个电源的费用相加就可得到整个配电网的建设费用：

$$C = C_1 + C_2 \tag{5-8}$$

其中，C_1和C_2分别表示电源 1 和电源 2 的最小建设费用详见 5.1.5 节。

同理，也可以得到各个用户的用电可靠性。

5.2.3 问题二模型的求解

(1) K-means 聚类分析的结果

与问题一相似，我们对 35 个负荷进行了聚类分析，同样采用遍历的思想，探究了不同k值情况下双供电源双层规划两次后的建设费用的变化情况。

(2) 最小生成树的求解

为确定负荷点由电源 1 还是电源 2 供电的问题，我们引入了最小生成树算法，并以树边长的总和作为评判标准。其具体的步骤如下：

Step1:通过排列组合，得到$2^k - 2$种情况。

Step2:对每种情况下的两个电源做最小生成树，使其与对应的聚类中心点相连。

Step3:计算每种情况下最小生成树的边长长度总和。

Step4:比较长度总和，选取长度总和小的几种情况做后续的优化。

(3) 双层规划模型的求解：

对双电源情况运用问题一的模型和算法，我们在最小建设费用的方案发现了两种情况：

① 当聚类簇数k较大时，某条主线会比另一条主线短很多。

② 当聚类簇数k较小时，一级支路的长度比主线的总长更长。

我们认为这个是不合理的，违背了实际情况。为了让模型更加贴近实际情况并保证两条主线的可扩展性，我们对最小生成树的目标函数进行修正，如下所示：

$$min \quad X = X_1 + X_2 + |X_1 - X_2| \tag{5-9}$$

该式要求在长度总和尽可能小的情况下，两个最小生成树的长度不能相差太大，经过我们的测试，新的目标函数能很好地解决情况①中的问题，使两条主线长度更加合理。

此外，我们增加了新的约束条件，要求每条主线上至少有两个聚类簇，这能较好地解决情况②中的问题。

因此，在遍历k值时，选择更加合理的规划，这可能会导致费用变高，但却能让模型更加贴近现实情况。得到的结果如下图所示：

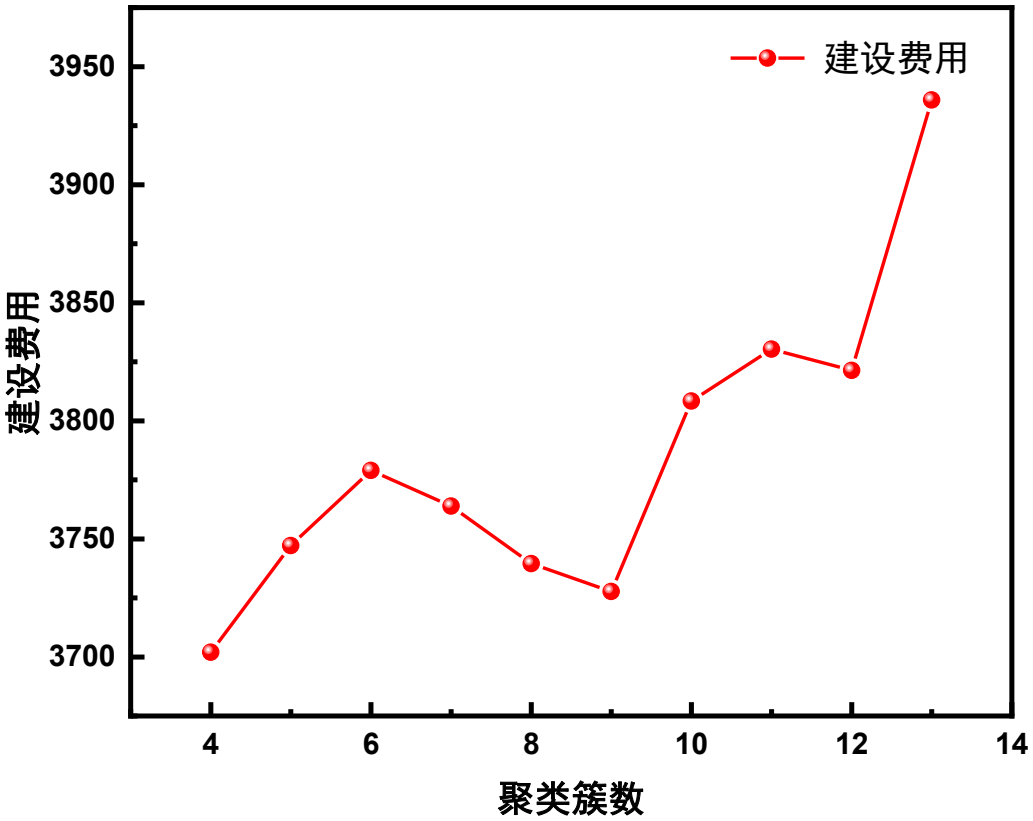


图 9 问题二聚类簇数与建设费用关系

注：由于聚类簇数为 4 和 5 时建设费用的波动较大，因此选择建设费用的平均值作图。

由图 9 可知，当k为 4 或 9 时，建设费用较小。由于k为 4 时建设费用不太稳定，某些情况中依然出现了一级支路长度长于主线总长的问题，因此我们选择k为 9 作为我们的最佳方案。我们选择k为 9，进行多次双层规划得到：

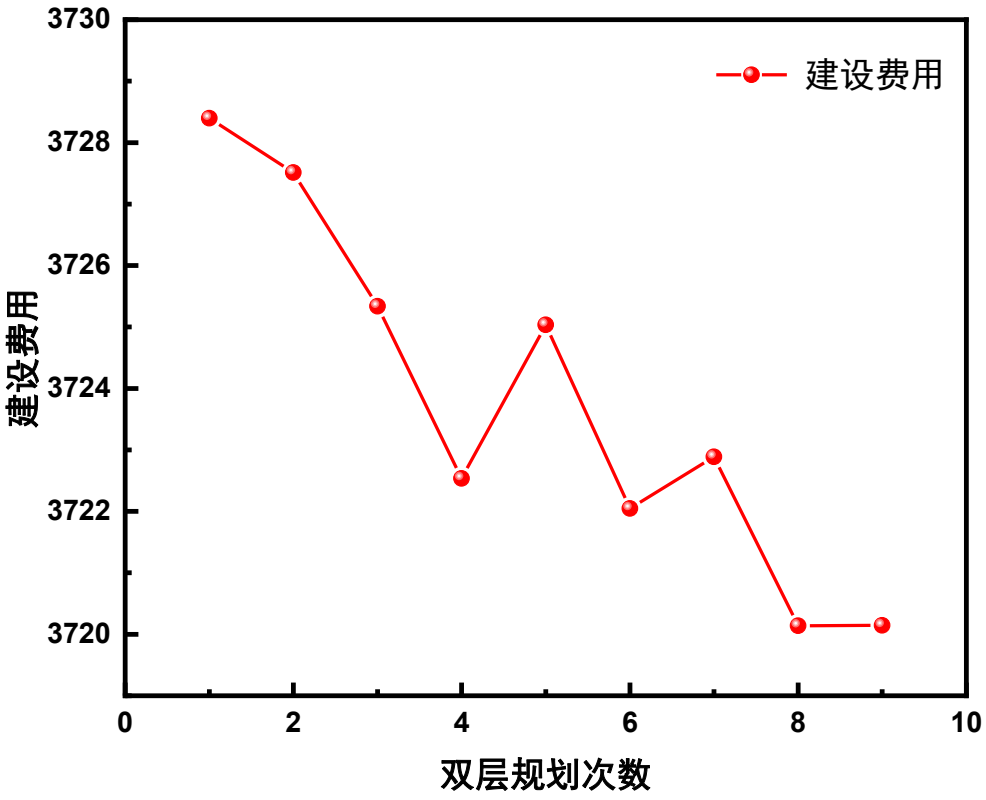


图 10 问题二双层规划结果图

得到最小建设费用为 3720.14 千元。该结果下对应的分叉点坐标如下表所示：

表 7 电源 1 分叉点坐标表

分叉点序号	分叉点等级	分叉点横坐标	分叉点纵坐标	分叉点序号	分叉点等级	分叉点横坐标	分叉点纵坐标
1	一级	-0.5959	1.0801	11	三级	-0.0300	-0.5350
2	一级	-0.4650	0.9122	12	三级	-0.5000	-0.5000
3	一级	-0.4844	0.5611	13	三级	-0.4500	-0.8600
4	二级	-0.6000	1.0800	14	三级	-0.6000	-0.2400
5	二级	-0.5000	-0.5000	15	三级	-0.1000	1.0000
6	二级	0.1390	1.0976	16	三级	0.1150	1.4200
7	二级	-0.4799	0.5548	17	三级	0.3900	1.2000
8	三级	-0.5200	0.9300	18	三级	0.1500	0.9500
9	三级	-0.7700	1.2000	19	三级	-0.4200	0.4400
10	三级	-0.6000	1.0800	20	三级	-0.5400	0.6700

表 8 电源 2 分叉点坐标表

分叉点序号	分叉点等级	分叉点横坐标	分叉点纵坐标	分叉点序号	分叉点等级	分叉点横坐标	分叉点纵坐标
1	一级	0.8834	1.1328	14	三级	0.3500	-1.2800
2	一级	0.7731	0.3512	15	三级	0.5300	-0.8600
3	一级	0.6129	-0.0083	16	三级	1.0700	1.1400
4	一级	0.7253	-0.3059	17	三级	0.7500	1.2050
5	二级	0.7732	0.3512	18	三级	0.9800	1.3500
6	二级	0.4000	-1.0000	19	三级	0.9400	0.9800
7	二级	0.8835	1.1328	20	三级	0.5400	0.9400
8	二级	0.7698	-0.3243	21	三级	0.7800	-0.1800
9	二级	0.2088	0.0012	22	三级	0.7600	-0.4700
10	三级	0.7600	0.4200	23	三级	0.0000	0.1500
11	三级	0.8400	0.3300	24	三级	0.2000	-0.0900
12	三级	0.6900	0.2800	25	三级	0.2500	0.0200
13	三级	0.4000	-1.0000				

(4) 用户可靠性模型的求解：

使用问题一中的模型，求出各个用户的用电可靠性，具体结果如下表所示：

表 9 问题二用户用电可靠性表

用户序号	用电可靠性	用户序号	用电可靠性
1	98.350%	19	97.578%
2	97.952%	20	97.288%
3	97.979%	21	97.288%
4	97.984%	22	97.609%
5	97.976%	23	97.334%
6	97.984%	24	97.316%
7	98.326%	25	97.680%

用户序号	用电可靠性	用户序号	用电可靠性
8	98.359%	26	97.371%
9	98.359%	27	98.384%
10	98.357%	28	98.358%
11	98.354%	29	98.384%
12	98.354%	30	97.665%
13	98.371%	31	97.952%
14	97.666%	32	97.578%
15	98.038%	33	98.030%
16	97.626%	34	97.691%
17	97.649%	35	98.038%
18	97.682%		

5.3 问题三模型的建立与求解

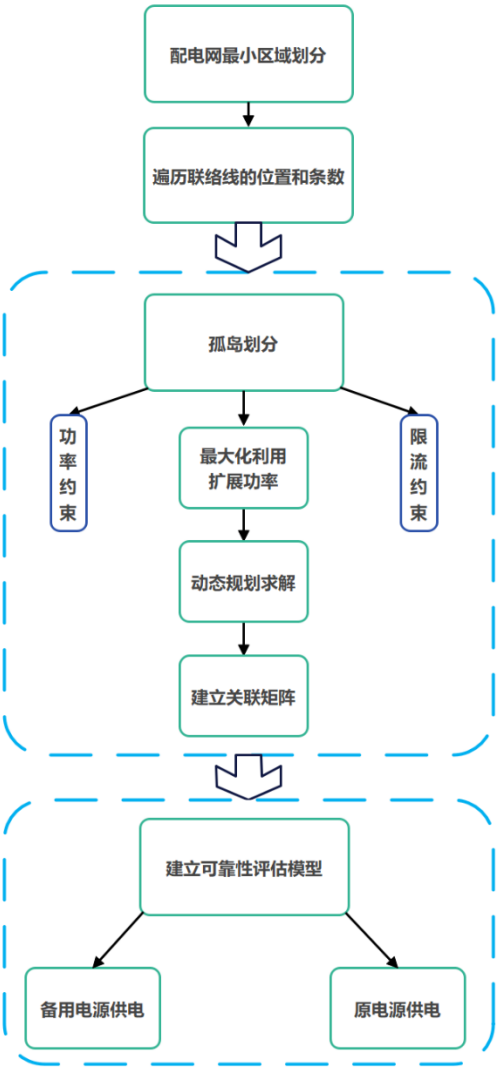


图 11 问题三思路图

5.3.1 配电网最小区域划分

最小配电区域：每个负荷以及开关到负荷前的线路称之为为**最小配电区域，如图 12 中的**Z_1、Z_2、Z_3、Z_4。由于配电网的故障扩散和供电恢复都以开关为基础，因此我们根据开关设置原则进行最小配电区域划分。这是故障发生时配电网中所能隔离的最小单元。

最小输电单元：主线上的每个故障单元线路与其到电源侧的主线开关称之为为最小输电单元，如图 12 中的L_1、L_2、L_3、L_4。以一级分叉点的位置为基础，对主线进行划分，并对各个最小输电单元进行编号以便后续建立关联矩阵。

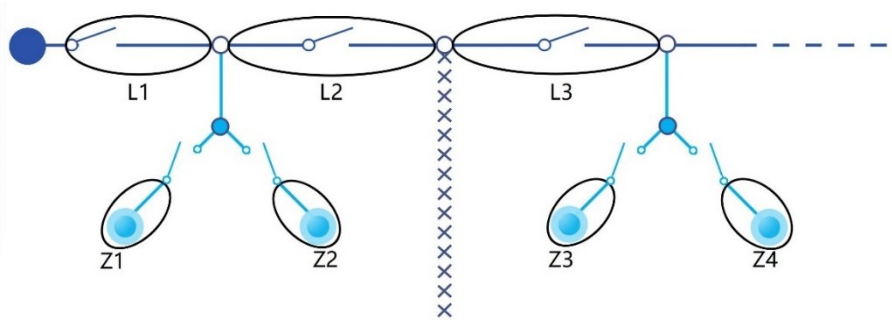


图 12 最小区域划分图

L_1、L_2、L_3、L_4是最小输电区域，Z_1、Z_2、Z_3、Z_4是最小配电区域。由交叉组成的线为联络线。

5.3.2 孤岛划分方法

我们将双供配电网的某个电源视为另一侧负荷的备用电源，当配电网发生故障时，备用电源会为另一侧的负荷供电。当故障发生时，会发生孤岛现象，即一部分负荷与主电网断开，该部分负荷由备用电源供电。

由于配电网的结构以及备用电源的功率有限，因此配电网故障后的孤岛划分是根据故障点的位置，联络线的连接位置，用户所需功率以及备用电源的扩展功率动态生成的。我们可以通过 5.3.4 节的动态规划模型求得孤岛划分方案，用关联矩阵表示不同故障发生时的孤岛划分方案。

进行孤岛划分的目标是最大化利用备用电源的扩展功率；这需要考虑两个约束原则，一是故障用户供电的功率总和不能超出备用电源多余的功率，二是备用电源的总电流不能超过其主线限流。

我们根据以上条件建立了孤岛划分模型：

孤岛划分模型目标函数：

$$max \quad \sum_{i}^{m} P_i \tag{5-10}$$

$$s.t. \begin{cases} \sum_{i}^{m} P_i \le P_{extend} & \text{功率约束} \\ \sum_{i}^{a} \frac{P_i}{10kV} \le \frac{2 \cdot P_{total}}{10kV} & \text{电流约束} \end{cases}$$

其中，

m为因故障由备用电源供电的用户数目；

P_i为第i个用户的功率需求量；

P_{extend}为备用电源扩展的功率；

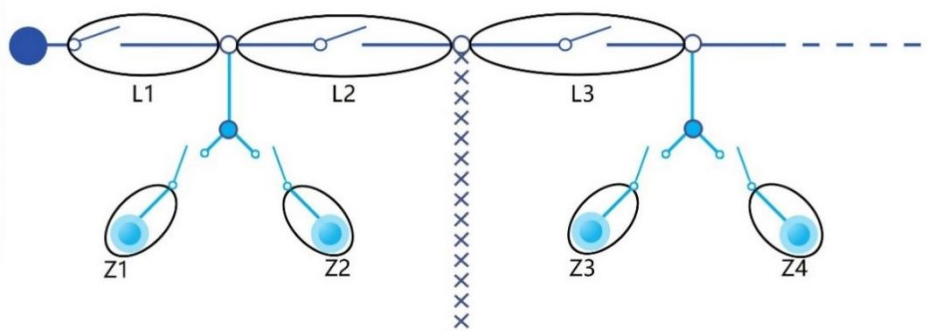
P_{total}为备用电源的额定功率(不包括扩展功率)；

a为备用电源供电的用户总数(额外供电用户和自身单供配电网用户之和)。

最终孤岛划分的结果由关联矩阵表示，关联矩阵是一个由 0 和 1 构成的矩阵，用于描述负荷是否由备用电源供电情况。矩阵中的元素a_{ij}表示，当最小供电单元L_j发生故障时，第i个负荷的供电情况。

$$a_{ij}=\begin{cases}1 & \text{该负荷纳入孤岛，由备用电源供电}\\ 0 & \text{该负荷不纳入孤岛}\end{cases}$$

为帮助理解关联矩阵的含义，下面以图 12 最小区域划分图作为例子分析：



假设 5.3.4 的动态规划模型求解出的关联矩阵如下式所示：

$$M=\begin{bmatrix}1 & 0 & 0\\ 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 1 & 0\end{bmatrix}$$

观察该矩阵的第一列，a_{11}和a_{21}为 1，a_{31}和a_{41}为 0，这一列元素表示，当L_1区域发生故障时，Z_1、Z_2区域由备用电源供电，Z_3、Z_4区域不由备用电源供电，即Z_3、Z_4区域断供。

观察该矩阵的第二列，a_{12}和a_{11}为 0，a_{32}和a_{42}为 1，这一列元素表示，当L_2区域发生故障时，Z_1、Z_2区域依旧由原电源供电，Z_3、Z_4区域由备用电源供电。

观察该矩阵的第三列，所有元素都为 0，这是因为故障点发生在L_2区域时，Z_1、Z_2区域依旧由原电源供电，而Z_3、Z_4区域由于联络线位置的原因无法由备用电源供电。

5.3.3 动态规划模型

确定孤岛划分策略，实际上就是求解出若干个负荷点，将其归入孤岛，以最大程度去利用备用电源的扩展功率。对于每个负荷点来说，会有两种结果，供电和不供电，如果暴力枚举，算法运行时间将会随输入规模呈指数级上升。因此我们采用动态规划的算法求解。

(1) 确定动态规划矩阵以及下标的含义

我们建立动态规划矩阵来表示供电功率，其中的元素a_{ij}表示，在前i个负荷中，对于功率上限为j的孤岛，供电功率的最大值。

当前状态依赖于之前的状态，初始状态$a_{00}=0$，总共有 N 个负荷，则需要进行 N 次决策，决定负荷能否由备用电源供电，每一次对第i个负荷的决策，由之前的决策进行更新。

(2) 确定状态转移方程

① 当前功率上限为j的孤岛不能为第i个负荷供电时，前i个负荷的最优选取方案即为前$i-1$个负荷的最优选取方案。

$$a_{ij}=a_{\{i-1\}\{j\}}$$

② 当前功率上限为j的孤岛能为第i个负荷供电时，此时需要决策是否为第i个负荷进行供电。

$$a_{ij}=\begin{cases}a_{\{i-1\}\{j-p_i\}}+P_i & \text{供电时}\\ a_{\{i-1\}\{j\}} & \text{不供电时}\end{cases}$$

由于我们需要最大化利用备用电源的功率，因此综合以上两种情况：

$$a_{ij}=max\left(a_{\{i-1\}\{j-p_i\}}+P_i\ ,\ a_{\{i-1\}\{j\}}\right)$$

(3) 得到最终孤岛划分结果

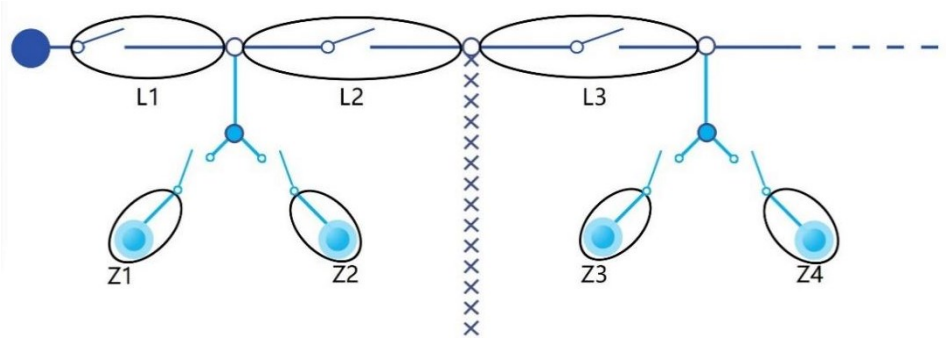
对之前的每次决策进行记录，最后得到最佳的孤岛划分方案，记录在关联矩阵中。

5.3.4 双供配电网用电可靠性评估模型

由于单个故障单元发生故障的概率基本上都处于 0.3%~0.5%，同时发生两多处故障的概率大约在 0.0009%~0.0025%，概率十分低，且同时发生多处故障的情况比较复杂。因此为了简化模型，我们把同时发生多处故障的概率视为高阶的无穷小项，在计算时予以忽略。这样计算出来的可靠性会偏低，但误差不会超过 0.0025%，我们认为是合理的。

(1) 引入例子

由于双供配电网的用电可靠性计算比较复杂，下面先通过图 12 最小区域划分图为例子来解释公式：



不妨设

L_1、L_2、L_3的正常运行的可靠性分别为$P(L_1)=95\%$、$P(L_2)=96\%$和$P(L_3)=97\%$；

用户 3 从一级分叉点到Z_3的所有单元设备正常运行的可靠性为$P(Z_3)=98\%$

从联络线到备用电源所有单元设备正常运行的可靠性为$P(L_{connect})=90\%$。

以下情况下用户 3 可以正常用电：

① 通过自身的电源供电。L_1、L_2、L_3和用户 3 从一级分叉点到Z_3的所有单元设备正常运行，此时恰好为单供情况下的用电可靠性，对应项为：

$$P(L_1)\cdot P(L_2)\cdot P(L_3)\cdot P(Z_3)$$

② 通过备用电源供电。L_1发生故障，但L_3和用户 3 从一级分叉点到Z_3的所有单元设备正常运行。且备用电源可以给用户 3 供电，联络线到备用电源所有单元设备正常运行。对应项为：

$$(1-P(L_1))\cdot a_{31}\cdot P(L_3)\cdot P(Z_3)\cdot P(L_{connect})$$

③ 通过备用电源供电。L_2发生故障，但L_3和用户 3 从一级分叉点到Z_3的所有单元设备正常运行。且备用电源可以给用户 3 供电，联络线到备用电源所有单元设备正常运行。对应项为：

$$(1-P(L_2))\cdot a_{32}\cdot P(L_3)\cdot P(Z_3)\cdot P(L_{connect})$$

如果是L_3发生故障时，根据动态规划结果$a_{33}=0$，用户 3 不能用电。为公式推广方便，可记作为：

$$(1-P(L_3))\cdot a_{33}$$

忽略高阶无穷小项，运用全概率公式可得用户 3 的用电可靠性为：

$$R_3=P(L_1)\cdot P(L_2)\cdot P(L_3)\cdot P(Z_3)+$$

$$\big(1-P(L_1)\big)\cdot a_{31}\cdot P(L_3)\cdot P(Z_3)\cdot P(L_{connect})+$$

$$\big(1-P(L_2)\big)\cdot a_{32}\cdot P(L_3)\cdot P(Z_3)\cdot P(L_{connect})+$$

$$\big(1-P(L_3)\big)\cdot a_{33}$$

(2) 简化公式

通过以上例子，不难看出双供配电网的用电可靠性计算十分复杂，这与故障发生的位置、联络线的位置和联络线的条数相关。为了将上述例子推广到更复杂的情况下，我们做出以下变量替换：

$$R_{31}=P(L_1)\cdot P(L_2)\cdot P(L_3)\cdot P(Z_3)$$

$$R_{32}=(1-P(L_1))\cdot a_{31}\cdot P(L_3)\cdot P(Z_3)\cdot P(L_{connect})+$$

$$(1-P(L_2))\cdot a_{32}\cdot P(L_3)\cdot P(Z_3)\cdot P(L_{connect})+(1-P(L_3))\cdot a_{33}$$

R_{31}表示用户 3 自身所在电源的用电可靠性；R_{32}表示用户 3 接入联络线后提升的用电可靠性。

不妨引入求和符号和新的符号对R_{32}进行进一步的化简：

$$R_{32}=\sum_{j=1}^{3}\left(1-R_{L_j}\right)\cdot a_{3j}\cdot R_{connect}$$

用$1-R_{L_j}$示第j条最小供电单元发生故障的概率，即$1-P(L_1)$、$1-P(L_2)$和$1-P(L_3)$；

用$R_{connect}$为第i个用户从其最小配电区域Z_i通过联络线连接到备用电源的概率，即三项共同拥有的公因子$P(L_3)\cdot P(Z_3)\cdot P(L_{connect})$，由于第三项中$a_{33}=0$，

也可等效视为其有公因子$P(L_3)\cdot P(Z_3)\cdot P(L_{connect})$。

再用求和符号把所有的用到联络线的 3 种情况相加。

(3) 推广公式

通过上述操作，我们成功将具体、复杂的情况化为一个抽象、简单的通式，下面将这个通式推广到普适的情况。

我们通过全概率公式，得到双供配电网的可靠性计算公式如下：

$$R_i = R_{i1} + R_{i2} \tag{5-11}$$

其中，

R_{i1}为第i个用户自身所在电源的用电可靠性，详见 5.1.6 节；

R_{i2}为接入联络线后提升的用电可靠性，其具体的计算公式如下：

$$R_{i2} = \sum_{j=1}^{b}\left(1 - R_{L_j}\right)\cdot a_{ij}\cdot R_{connect} \tag{5-12}$$

其中，

b是用户i供电的所有情况

$1 - R_{L_j}$为第j条最小供电单元发生故障的概率；

a_{ij}为关联矩阵的元素，用于判断故障时负荷能否被备用电源供电；

$R_{connect}$为第i个用户从其最小配电区域Z_i通过联络线连接到备用电源的可靠性，即能成功连接到备用电源的概率。其具体的计算公式与用户的位置、故障发生的位置、联络线的接入位置、联络线的条数有关，即与i和j的取值有关；求和符号表示将所有的最小供电单元发生故障时的情况相加。

综上，在得到了双供配电网用电可靠性模型和关联矩阵后，求总建设费用不超过X的情况下，使双供配电网中最低的用电可靠性达到最大的联络线和开关设计。其目标函数如下：

$$max \quad \{min[R_1, R_2, \dots, R_n]\} \tag{5-13}$$

$$s.t. \begin{cases} C \le X & \text{费用约束} \\ \sum_{i}^{m} P_i \le P_{extend} & \text{功率约束} \\ \sum_{i}^{a} \dfrac{P_i}{10kV} \le \dfrac{2\cdot P_{power}}{10kV} & \text{电流约束} \end{cases}$$

5.3.5 问题三模型的求解

为减少计算量，在连接联络线时，我们确定了联络线所在的区域，但没有遍历区域上的所有点。在两个一级分叉点的中点处插入一个点，通过遍历一级分叉点和插入点的位置来决定联络线的位置。使用苏州工业园的用户数据，问题三得到如下结果。

由于建设费用的上限X未知，我们讨论了不同联络线情况下的可靠性，如下表所示：

表 10 问题三结果表

联络线条数	最低用电可靠性	联络线费用上限(千元)	建设费用上限(千元)
一条	97.8349%	500	5108.95
两条	97.8439%	1100	5708.95
三条	97.8440%	1700	6308.95

(1) 一条联络线的情况

表 11 问题三用户用电可靠性表（一条联络线）

用户序号	用电可靠性	用户序号	用电可靠性
1	99.237%	19	98.069%
2	99.071%	20	97.894%
3	99.098%	21	97.890%
4	99.104%	22	98.100%
5	99.096%	23	98.817%
6	99.104%	24	97.934%
7	99.201%	25	98.172%
8	99.234%	26	97.990%
9	99.234%	27	99.271%
10	99.232%	28	99.245%
11	99.229%	29	99.271%
12	99.229%	30	98.356%
13	99.247%	31	99.071%

用户序号	用电可靠性	用户序号	用电可靠性
14	99.243%	32	97.835%
15	99.258%	33	99.250%
16	99.002%	34	99.174%
17	99.132%	35	99.258%
18	99.165%		

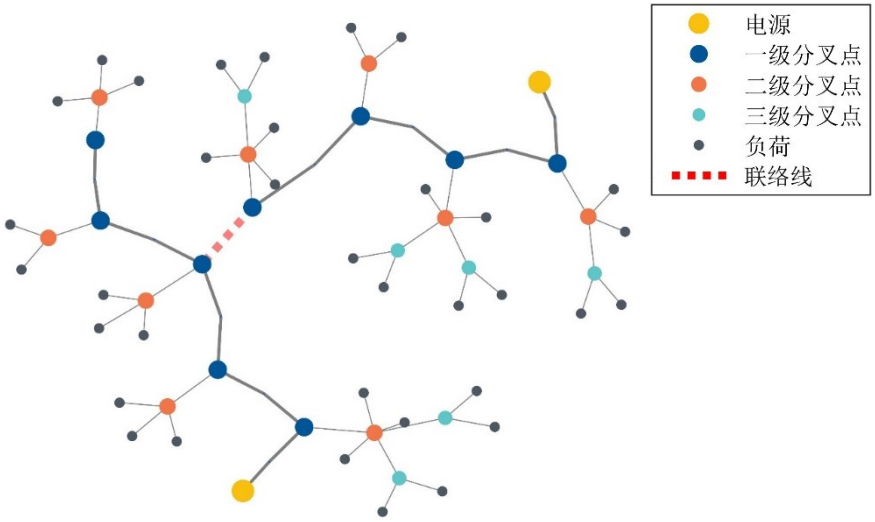


图 13 问题三 单联络线电网结构拓扑图

(2) 两条联络线的情况

表 12 问题三 用户用电可靠性（两条联络线）

用户序号	用电可靠性	用户序号	用电可靠性
1	99.259%	19	98.080%
2	99.096%	20	97.894%
3	99.123%	21	97.890%
4	99.129%	22	98.112%
5	99.120%	23	98.839%
6	99.129%	24	97.934%
7	99.222%	25	98.184%
8	99.255%	26	97.990%
9	99.255%	27	99.293%

用户序号	用电可靠性	用户序号	用电可靠性
10	99.253%	28	99.266%
11	99.250%	29	99.293%
12	99.250%	30	98.366%
13	99.268%	31	99.096%
14	99.269%	32	97.844%
15	99.276%	33	99.268%
16	99.028%	34	99.194%
17	99.152%	35	99.276%
18	99.185%		

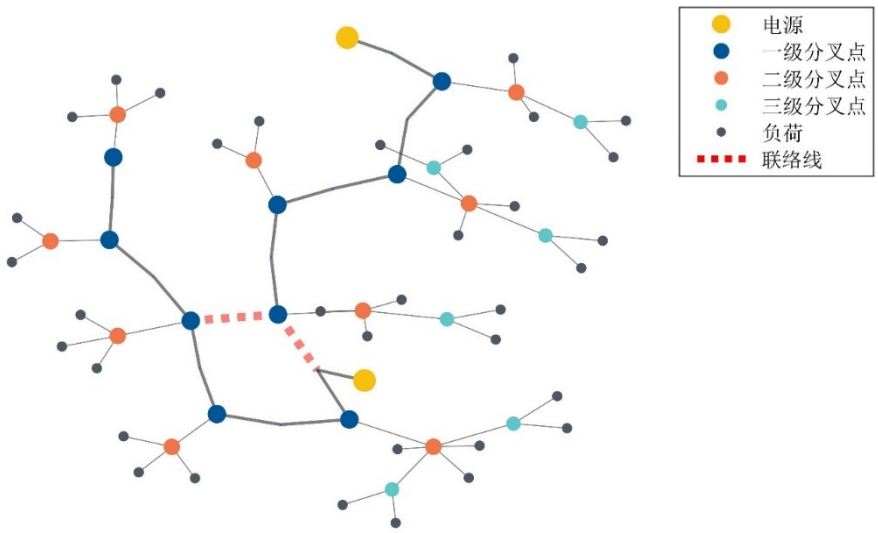


图 14 问题三 双联络线电网结构拓扑图

(3) 三条联络线的情况

表 13 问题三 用户用电可靠性表（三条联络线）

用户序号	用电可靠性	用户序号	用电可靠性
1	99.259%	19	98.080%
2	99.096%	20	98.331%
3	99.123%	21	98.328%
4	99.129%	22	98.112%

用户序号	用电可靠性	用户序号	用电可靠性
5	99.121%	23	99.085%
6	99.129%	24	98.179%
7	99.222%	25	98.184%
8	99.256%	26	98.235%
9	99.256%	27	99.293%
10	99.253%	28	99.267%
11	99.250%	29	99.293%
12	99.250%	30	98.366%
13	99.268%	31	99.096%
14	99.270%	32	97.844%
15	99.283%	33	99.276%
16	99.029%	34	99.203%
17	99.161%	35	99.284%
18	99.194%		

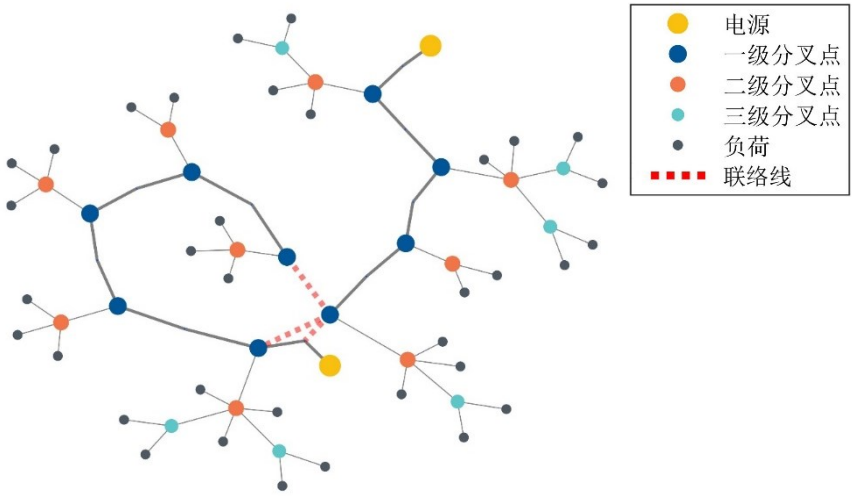


图 15 问题三 三条联络线电网结构拓扑图

5.4 问题四模型的建立与求解

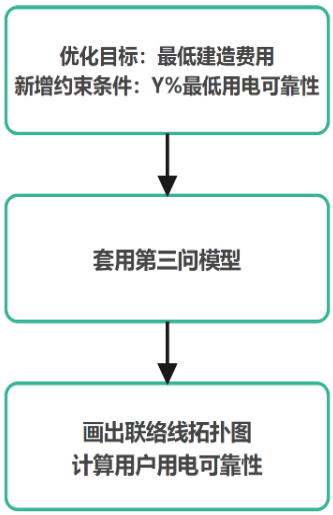


图 16 问题四思路图

5.4.1 问题四模型的建立

在问题四中，仍使用问题三中的动态规划算法得到关联矩阵去表征故障发生时的负荷点供电情况。以配电网总的建设费用最小为优化目标，以每个用户的用户可靠性不低于 Y%，双供配电网用户供电调度原则和主线总限流为约束条件建立以下模型：

$$min \quad C = C_1 + C_2 + C_{connect} + C_{expend} \tag{5-14}$$

$$s.t. \begin{cases} min[R_1, R_2, \dots, R_n] \geq Y\% & \text{可靠性约束} \\ \sum_{i}^{m} P_i \leq P_{extend} & \text{功率约束} \\ \sum_{i}^{a} \frac{P_i}{10kV} \leq \frac{2 \cdot P_{power}}{10kV} & \text{电流约束} \end{cases}$$

其中，

C_1和C_2分别为电源 1 和电源 2 的最小建设费用，详见 5.1.5 节；

C_{expend}为扩展供量带来的费用；

$C_{connect}$为联络线的建设费用，其计算公式如下：

$$C_{connect} = U_{mainline} \cdot x_{connect} + n_c \cdot U_{switch1}$$

其中，$x_{connect}$为联络线的总长度；n_c为联络线的条数；$U_{mainline}$和$U_{switch1}$详见5.1.4节。

5.4.2 问题四模型的求解

在第二问的结果下，最低的用户可靠性为97.288%。由于最低的用户用电可靠性Y%未知，我们讨论了不同联络线情况下的可靠性，如下表所示：

表14 问题四结果表

最低的用电可靠性	联络线条数	建设费用(千元)
97.3000%	一条	5069.10
97.8400%	两条	5529.55
97.8439%	三条	6070.20

(1) 一条联络线的情况

表15 问题四 用户用电可靠性表（一条联络线）

用户序号	用电可靠性	用户序号	用电可靠性
1	99.237%	19	98.064%
2	99.065%	20	98.961%
3	99.095%	21	98.092%
4	99.102%	22	98.096%
5	99.091%	23	97.743%
6	99.102%	24	97.725%
7	99.201%	25	98.168%
8	99.234%	26	97.780%
9	99.234%	27	99.272%
10	99.232%	28	99.245%
11	99.229%	29	99.272%
12	99.229%	30	98.355%
13	99.247%	31	99.065%
14	99.242%	32	97.834%
15	99.256%	33	99.249%

用户序号	用电可靠性	用户序号	用电可靠性
16	98.994%	34	99.172%
17	99.130%	35	99.257%
18	99.163%		

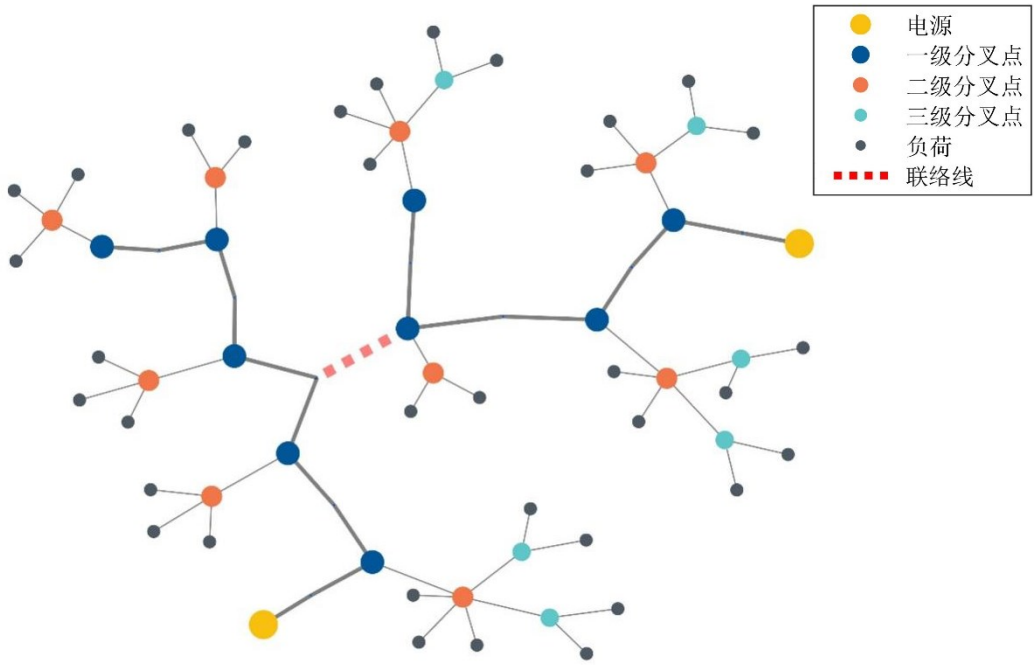


图 17 问题四 单联络线电网结构拓扑图

(2) 两条联络线的情况

表 16 问题四 用户用电可靠性表（两条联络线）

用户序号	用电可靠性	用户序号	用电可靠性
1	99.258%	19	98.080%
2	99.095%	20	98.145%
3	99.123%	21	98.141%
4	99.128%	22	98.112%
5	99.120%	23	99.090%
6	99.128%	24	98.184%
7	99.222%	25	98.184%
8	99.255%	26	98.240%
9	99.255%	27	99.293%

用户序号	用电可靠性	用户序号	用电可靠性
10	99.253%	28	99.266%
11	99.250%	29	99.293%
12	99.250%	30	98.366%
13	99.268%	31	99.095%
14	99.269%	32	97.843%
15	99.283%	33	99.275%
16	99.028%	34	99.209%
17	99.167%	35	99.283%
18	99.200%		

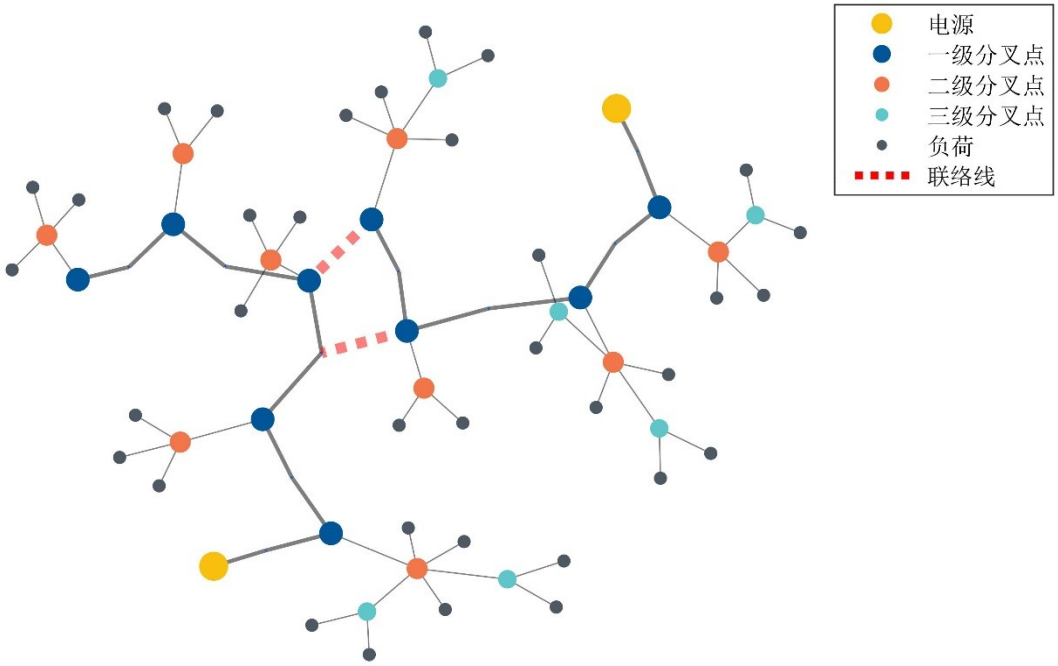


图 18 问题四 双联络线电网结构拓扑图

(3) 三条联络线的情况

表 17 问题四 用户用电可靠性表（三条联络线）

用户序号	用电可靠性	用户序号	用电可靠性
1	99.260%	19	98.079%
2	99.090%	20	99.248%
3	99.120%	21	98.358%

用户序号	用电可靠性	用户序号	用电可靠性
4	99.127%	22	98.111%
5	99.116%	23	98.199%
6	99.127%	24	98.181%
7	99.223%	25	98.183%
8	99.256%	26	98.237%
9	99.256%	27	99.294%
10	99.254%	28	99.267%
11	99.251%	29	99.294%
12	99.251%	30	98.365%
13	99.269%	31	99.090%
14	99.269%	32	97.844%
15	99.283%	33	99.275%
16	99.028%	34	99.204%
17	99.162%	35	99.283%
18	99.194%		

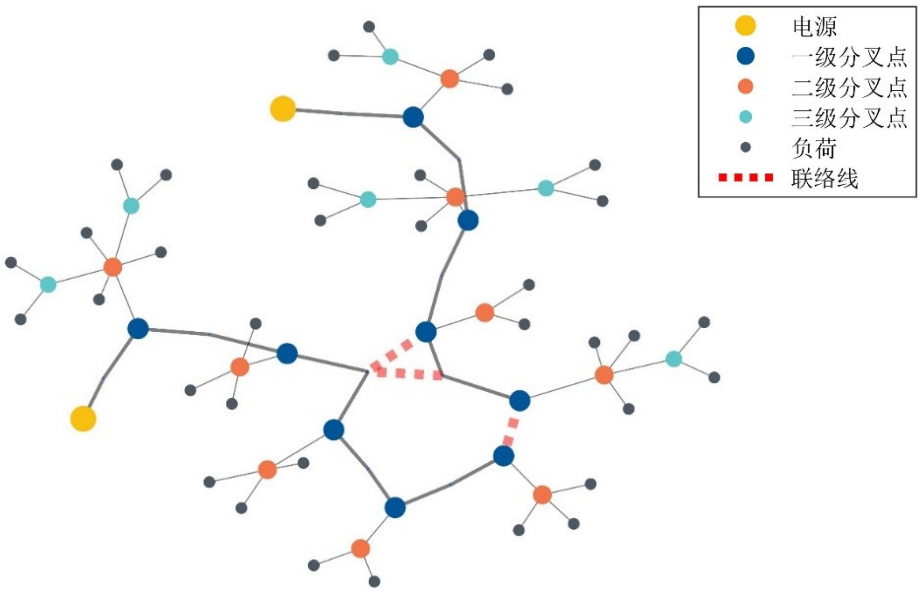


图 19 问题四 三条联络线电网结构拓扑图

6 模型的评价

6.1 灵敏度分析

我们对问题一建立的模型进行灵敏度分析，令电源的横坐标和纵坐标上下波动 5%，探究建设费用的变化情况，如下图所示：

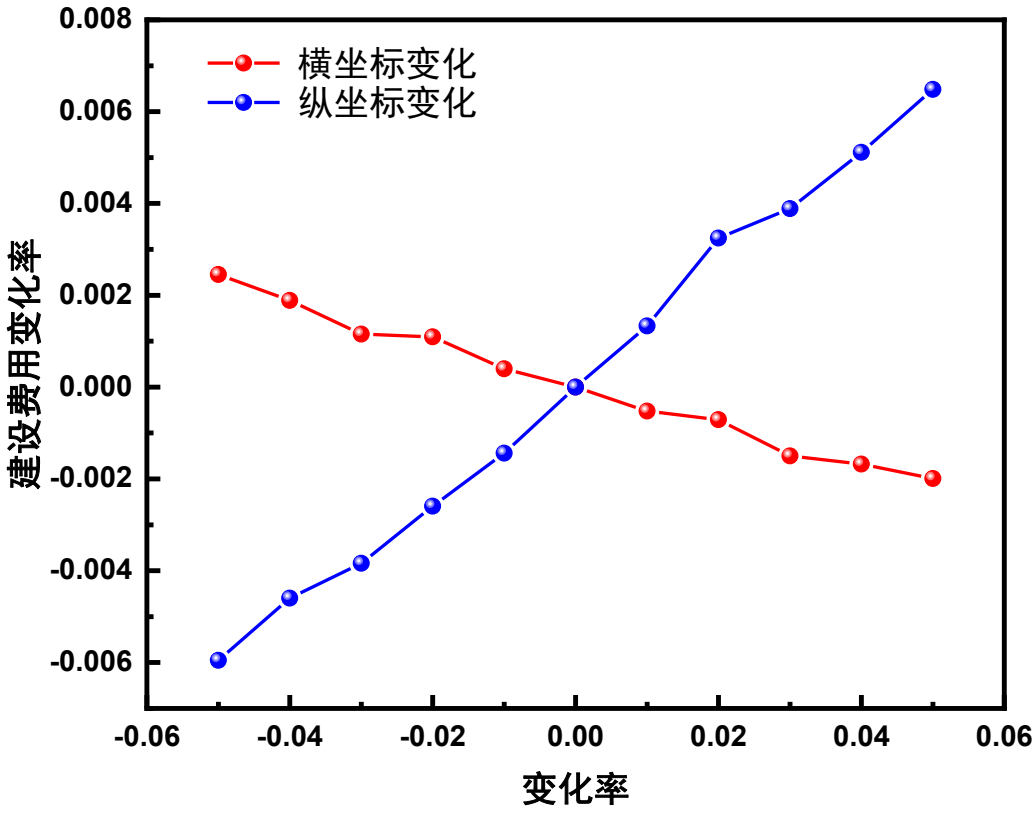


图 20 灵敏度分析图

由图 20 可知，无论是横坐标变化还是纵坐标变化，建设费用的变化率均处于 0.2%~0.6%，说明模型受电源坐标变化的影响较小，稳定性强，能够分析不同坐标数据下的配电网规划问题。

6.2 模型的优点

1. 相比于传统的聚类分析，考虑到聚类效果最好并不代表着建设费用最小，采用遍历的方法找到最优的聚类簇数，解决了聚类分析k值选取的问题。
2. 通过双层规划互相传递分叉点信息，反复迭代找到了最优的分叉点位置。

3. 将遗传算法和模拟退火算法相结合，得到更加高效稳定的遗传-模拟退火算法。
4. 在问题二中引入最小生成树算法，决定负荷点由哪个电源供电，大大减少了计算量；在得到结果后进一步分析，优化了最小生成树算法的优化函数，让模型更加接近实际生活。
5. 对配电网进行最小区域划分，用关联矩阵去表示负荷供电情况。

6.3 模型的缺点

1. 延用本模型的思路能够处理更加多级的分叉点问题，但为了简化模型，仅考虑到了三级分叉点的情况，未考虑更加多级分叉点的问题。
2. 由于使用了启发式算法，需要调整参数，且计算的结果有一定的波动。
3. 考虑到三级分叉点的微小调整对建设费用影响较小，为了减少计算量直接把二次嵌套聚类的中心点视为三级分叉点。
4. 为减少模型的计算量，在确定联络线时确定了联络线所在的区域，但没有遍历区域上的所有点。

7 参考文献

[1]孙若萱. 计及分布式电源的高可靠性配电网规划方法研究[D].东南大学,2019.

[2] 查伟雄,孙敬. 基于模拟退火算法的危险货物道路运输路径优化双层规划模型[J]. 公路交通科技,2012,29(04):101-106.

[3]解晨,韦雄奕.模拟退火算法和遗传算法的比较与思考[J].电脑知识与技术,2013,9(19):4418-4419.

8 附录

附录 1：算例数据表

电源数据如下表：

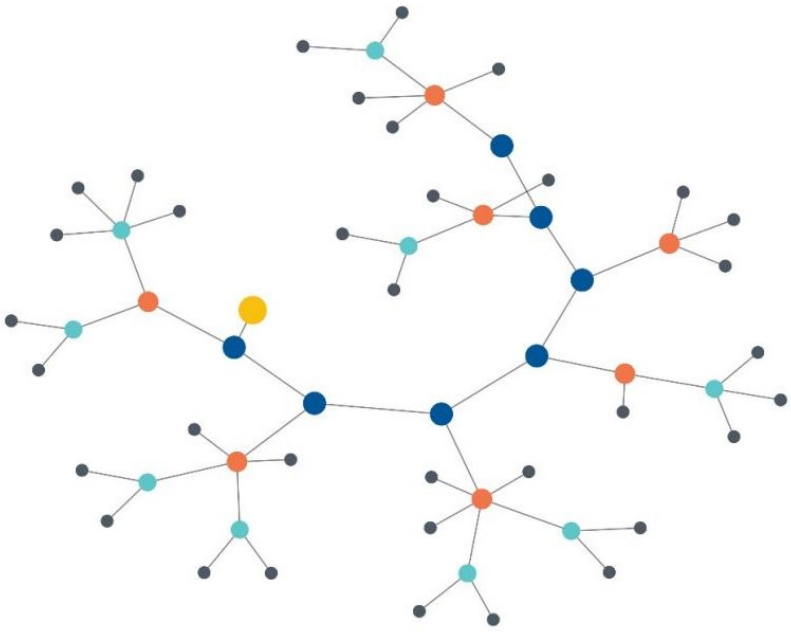
序号	出线电压	额定供电量(兆瓦)	可扩展供电量	扩展供量价格(千元)
1	10kV	负荷需求之和*1.1	50%	扩展功率*16.5
2	10kV	负荷需求之和*1.1	50%	扩展功率*16.5

本文在苏州工业园实例的基础上，增加了 11 个负荷点，具体数据如下表所示：

电源(负荷)序号	横坐标(千米)	纵坐标(千米)	功率需求量(兆瓦)
电源 1	-0.6	2.17	
电源 2	0.62	2.13	
1	-0.77	1.2	2.7
2	0.05	1.45	2.42
3	-0.1	1	3.41
4	0.07	0.94	3.37
5	0.39	1.2	2.16
6	0.23	0.96	2.6
7	0.54	0.94	3.5
8	0.7	1.15	2.81
9	0.8	1.26	2.8
10	0.98	1.35	3.6
11	1.1	1.2	3.1
12	1.04	1.08	2.9
13	0.94	0.98	3.9
14	-0.42	0.44	1.97
15	0.76	0.42	2.33
16	-0.6	-0.24	2.71
17	0	0.15	1.35

18	0.2	-0.09	3
19	0	-0.49	2.73
20	0.78	-0.18	3.45
21	0.76	-0.47	2.7
22	-0.45	-0.86	2.84
23	0.53	-0.86	3.2
24	0.35	-1.28	2.5
25	-0.5	-0.5	1.76
26	0.4	-1	2.27
27	-0.62	1.05	2.16
28	-0.52	0.93	3.19
29	-0.58	1.11	3.68
30	-0.54	0.67	3.41
31	0.18	1.39	3.56
32	-0.06	-0.58	1.99
33	0.69	0.28	3.42
34	0.25	0.02	2.4
35	0.84	0.33	2.05

附录 2：图片补充(单位：千米)



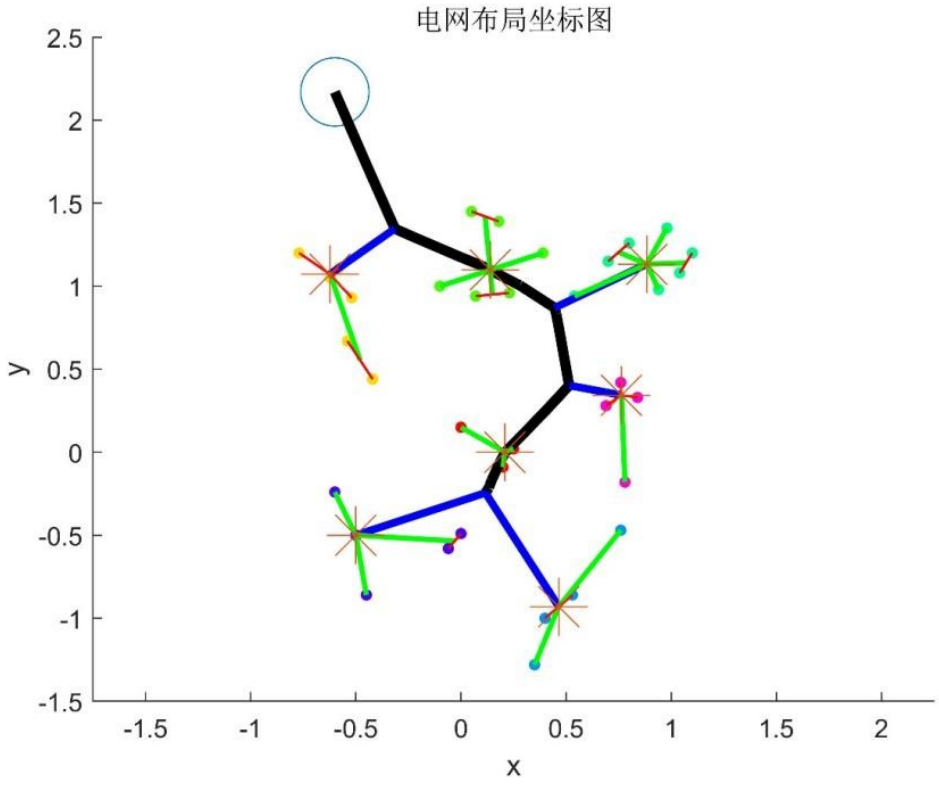


图 20 问题一 拓扑图与坐标图

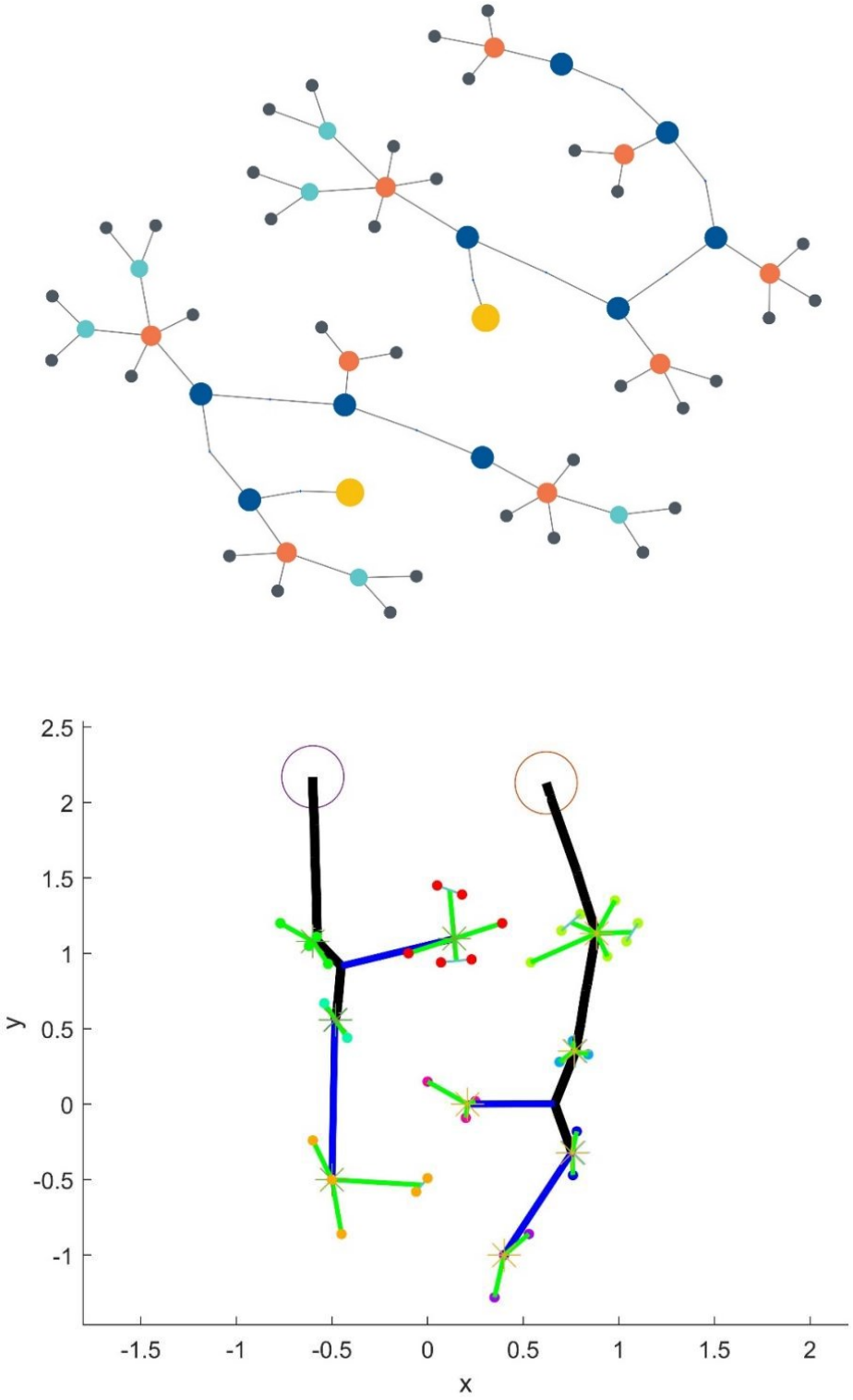
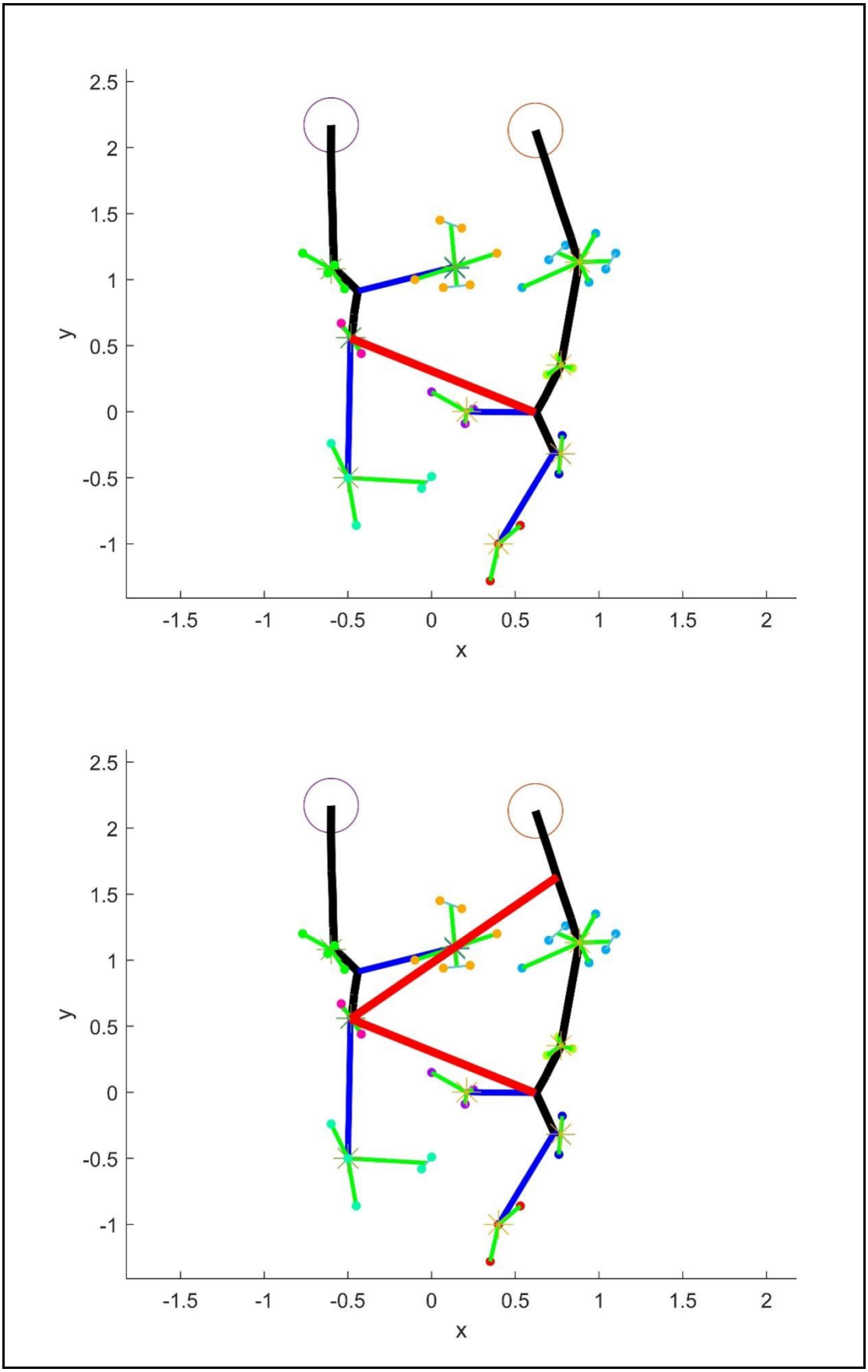


图 21 问题二 拓扑图与坐标图


2.5
2
1.5
1
0.5
0
-0.5
-1
y
-1.5
-1
-0.5
0
0.5
1
1.5
2
x
2.5
2
1.5
1
0.5
0
-0.5
-1
y
-1.5
-1
-0.5
0
0.5
1
1.5
2
x

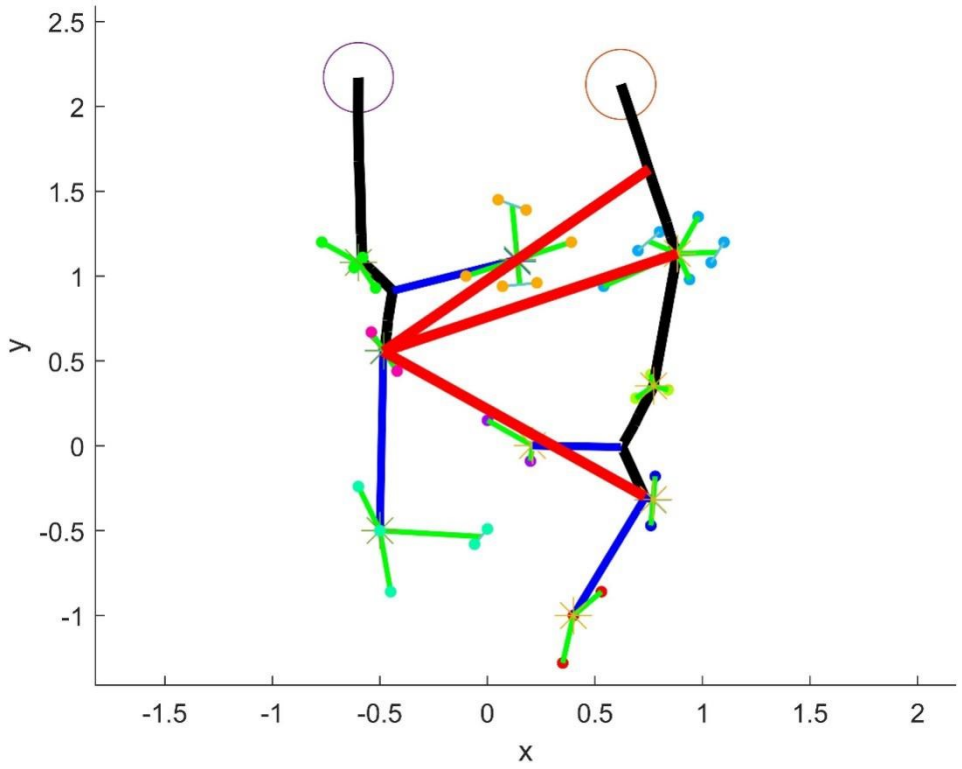
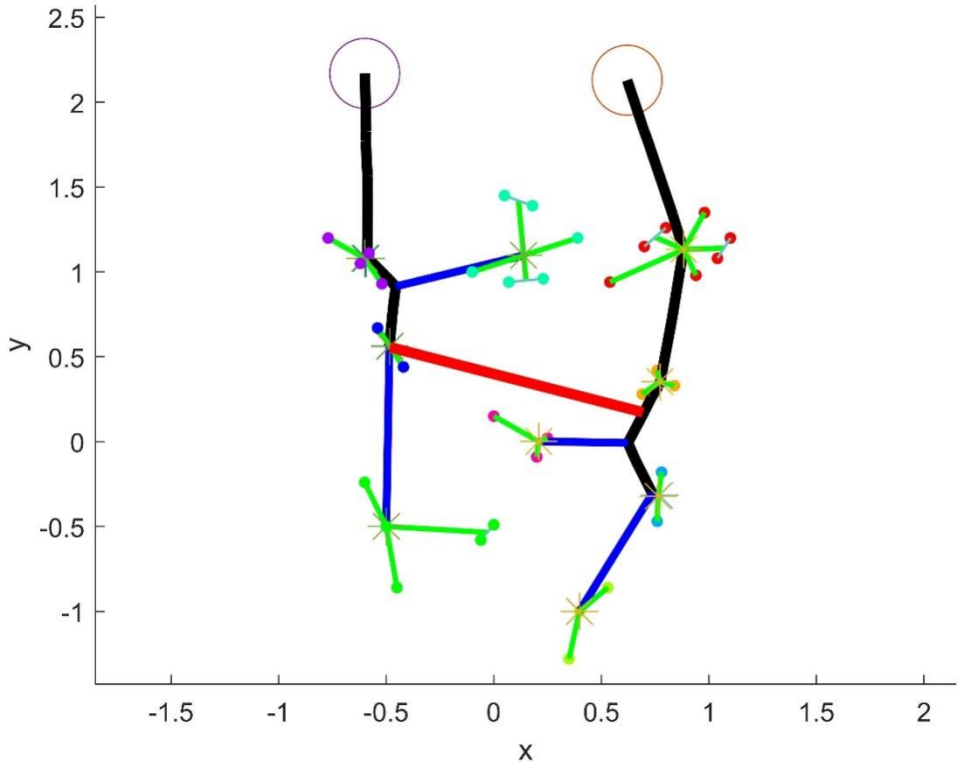


图 22 问题三 各情况坐标图



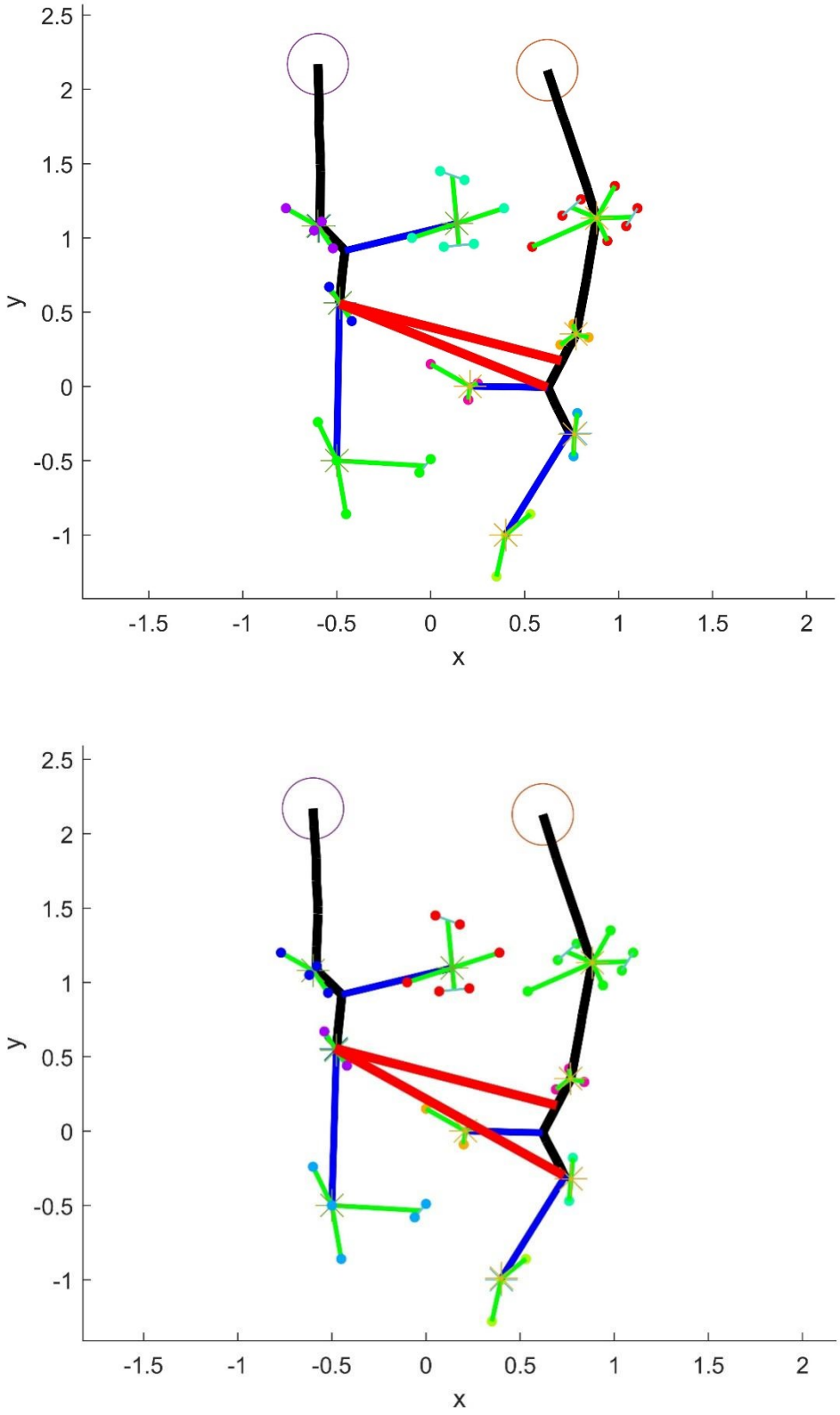


图 23 问题四 各情况坐标图

注：图 23 中最后一张图对应的是三条联络线的情况，由于有两个一级分叉点重合，我们认为其无限接近；导致图上联络线也重合了，实际上连接的点不相同。

附录 3：Matlab 代码

第一问要用到程序一到十七

第二问要用到程序二到二十

第三问要用到程序二十到二十六

第四问要用到程序二十到二十六

程序一：第一问主程序

B.m

```
clear; clc;
global center2; global center2_load; global edge;
global i;  global dott; global center_load;


% 读取/生成数据部分
data = readmatrix("电网数据.xlsx");
supply = data(1, 1:2);
x = data(4:size(data, 1), 1)'; % 点的 x 坐标
y = data(4:size(data, 1), 2)'; % 点的 y 坐标
len = length(x);


% 第一层聚类部分，idx 为各负荷所属簇的下标，center 为各簇中心坐标
cluster_num = 7; % 聚类簇数 k
[idx, center] = kmeans(cat(1, x, y)', cluster_num, 'Replicates', 1000); % 聚类 1000 次，选 SSD 最小的结果
center_load = count_num(idx, cluster_num); % 每簇对应的负荷数
% gscatter(x, y, idx); % 根据聚类结果，用不同颜色画出各点


% 画出 二级分叉点和电源 的外接矩形，生成均匀点集
box_tmp = cat(1, center, supply);
spacing = 0.01 * min( max(box_tmp) - min(box_tmp) ); % 生成点的间距为外接矩形的宽度的 1%
[boxx, boxy] = bounding_box(box_tmp(:, 1), box_tmp(:, 2), spacing); % 画出外接矩形，生成均匀点集
clear box_tmp;
% hold on;
% scatter(boxx, boxy, 2);
```

```
times = 2;
color = [0,0,0; 0,0,255; 0,255,0; 255,0,0] / 255;
for time = 1:times % 双层规划次数
    % 遗传算法
    epoch = 400;
    pop_size = 30; % 群体大小
    individual_size = 20; % 个体长度
    pm = 0.1; % 变异概率

    [best_individual, fit, run_time1] = Genetic_algorithm(epoch, pop_size,
individual_size, pm, supply, center, center_load);


%     % 这是不用模拟退火，单用遗传的代码
%          [dot,  fit,  run_time1]  =  Genetic_algorithm(1000,  pop_size,
individual_size, pm, supply, center, center_load);


    % 进行模拟退火
    temterature = 5;  % 初始温度
    innerloop = 100;    % 马尔科夫链长 （内层循环次数）
    dc = 0.99; % 退温系数 Dewarming_coefficient
    spacing = 0.01 * min( max(center) - min(center) ); % 扰动步长为外接矩形
的宽度的 1%

    % 将遗传算法的最优解作为初始点的位置
    dot_index = dsearchn([boxx', boxy'], best_individual);
    dot = [boxx(dot_index)', boxy(dot_index)'];
    previous_cost = cost_fun(cat(1, supply, dot), center, center_load);

    [dot, cost, run_time2] = Simulated_annealing(temterature, innerloop,
dc, dot, supply, center, center_load, spacing, boxx, boxy);

    if time == 1
        figure;
        plot(run_time1, fit);
        hold on;
        plot(max(run_time1)+run_time2, cost);
        hold on;
        line([max(run_time1),  max(run_time1)+run_time2(1)],  [fit(epoch),
cost(1)], 'Color', [1, 0, 0]);
        title("遗传模拟退火算法下费用的变动情况");
```

```
        xlabel('运行时间 / s');
        ylabel('费用 / 千元');
    end

%     fit = cat(2, fit, cost);
%     run_time = cat(2, run_time1, max(run_time1)+run_time2);
%     writematrix(fit, "result5");


%     % 这是不用遗传，单用模拟退火的代码
%     % 初始化扰动：随机改变 n 个点的位置
%     dot_num = 20; % 点的数量，扰动方式改进后，点的数量影响不大，但要控制在 10-20 个左右
%       dot = ceil(length(boxx) * rand(1, dot_num) ); % 随机从所有点中选 dot_num 个点
%     dot = [boxx(dot)', boxy(dot)']; % 得到点的坐标
%       [dot, cost, run_time2] = Simulated_annealing(10, 150, 0.99, dot, supply, center, center_load, spacing, boxx, boxy);


    % 画图检查结果
    % scatter(center(:, 1), center(:, 2), 125); % 画出各簇中心
    % hold on;
    % scatter(supply(1), supply(2), 200);

    dot = cat(1, dot, supply);
    dotx = dot(:, 1)';
    doty = dot(:, 2)';
    [~, edge] = minspan(dot);


    % hold on;
    % scatter(dotx, doty, 50);
    % hold on;
    %  line([dotx(edge(:,  1));  dotx(edge(:,  2))],  [doty(edge(:,  1));  doty(edge(:, 2))] );


    % 画图
    if time == times
        figure;
        title("电网布局坐标图");
        hold on;
        gscatter(x, y, idx);
```

```
        hold on;
        % scatter(center(:, 1), center(:, 2), 200, "diamond");
        hold on;
        scatter(supply(1), supply(2), 700);
        hold on;
        line([dotx(edge(:, 1)); dotx(edge(:, 2))], [doty(edge(:, 1));
doty(edge(:, 2))], 'Color', color(1, :), 'LineWidth', 4)
        hold on;
    end


    dott = dot;
    % 第二层规划
    cluster = cell(1, cluster_num); % 第一次聚类中的簇
    idx2 = cell(1, cluster_num); % 第二次聚类中各点的归属
    cluster2 = cell(1, cluster_num); % 第二次聚类中各簇
    center2 = cell(1, cluster_num); % 第二次聚类中各中心
    center2_load = cell(1, cluster_num); % 第二次聚类中各簇点数
    point = zeros(cluster_num, 2); % 优化后的二级分叉点坐标
    nearest = zeros(size(edge, 1), 2);

    cost_1 = 0; cost_2 = 0; cost_3 = 0; cost_main = 0;
    cost_4 = 2.6 * len + 56.8 * cluster_num;

    for i = 1:cluster_num  % 对第一层的每个聚类中心
        % 将第一层聚类的每个簇提取出来
        cluster{i} = [x(idx == i)', y(idx == i)'];

        % 根据每簇的点个数 进行 第二层聚类
        clear SilhouetteCoefficient;
        for n = 1:ceil(center_load(i) / 2 + 1)
            [idx2{i}, center2{i}] = kmeans(cluster{i}, n, 'Replicates',
200);
            SilhouetteCoefficient(n)     =     mean(silhouette(cluster{i},
idx2{i}) );
            if center_load(i) == 1
                break;
            end
        end

        [~, cluster2_num] = max(SilhouetteCoefficient);
        [idx2{i},    center2{i}]    =    kmeans(cluster{i},    cluster2_num,
'Replicates', 500);
        center2_load{i} = count_num(idx2{i}, cluster2_num);
```

```
        cluster2{i} = cell(1, cluster2_num);
        for t = 1:cluster2_num
            cluster2{i}{t} = cluster{i}(idx2{i} == t, :);
        end



        hold on;
        %     scatter(center2{i}(:, 1), center2{i}(:, 2), 100);


        % 以 一级支线的费用和二级支线的费用最小为目标 优化二级分叉点
        point(i, :) = fminsearch(@optimize_fun, rand(1, 2));
        for n = 1:99
            point(i, :) = point(i, :) + fminsearch(@optimize_fun, rand(1,
2));
        end
        point(i, :) = point(i, :) / 100;



        % 计算费用
        % 计算一级支线费用
        d = zeros(1, size(edge, 1));
        for j = 1:size(edge, 1)
            % 二级分叉点到总线每段的距离
            [d(1, j), nearest(j, :)] = distance_fun(point(i, :), dot(edge(j,
1), :), dot(edge(j, 2), :) );
        end
        % 二级分叉点到总线的最短距离
        [distance, min_index] = min(d);
        if time == times
            hold on;
            line([point(i,  1),  nearest(min_index,  1)],  [point(i,  2),
nearest(min_index, 2)], 'Color', color(2, :), 'LineWidth', 3);
        end

        if center_load(i) >= 3
            weight = 239.4; % 支线 B
        else
            weight = 188.6; % 支线 A
        end
        % 加上一级支线的价格
        cost_1 = cost_1 + weight * distance;
```

```
        % 计算二级支线费用
        for j = 1:cluster2_num % 对第二层聚类的每簇
            % 第 i 簇里面的第 j 簇的负荷数
            if center2_load{i}(j) >= 3
                weight = 239.4; % 支线 B
            else
                weight = 188.6; % 支线 A
            end
            cost_2 = cost_2 + weight * dist(point(i, :), center2{i}(j, :)');
            if time == times
                hold on;
                line([point(i,  1),  center2{i}(j,  1)],  [point(i,  2),
center2{i}(j, 2)], 'Color', color(3, :), 'LineWidth', 2);
            end

            % 计算三级支线费用
            weight = 188.6; % 支线 A
            distance = dist(cluster{i}(idx2{i} == j, :), center2{i}(j, :)');
            cost_3 = cost_3 + weight * sum(distance);

            for k = 1:center2_load{i}(j) % 对第二层每簇的每个点
                box_tmp = cluster{i}(idx2{i} == j, :);
                if time == times
                    hold on; % 画图，连线
                    line([box_tmp(k, 1), center2{i}(j, 1)], [box_tmp(k, 2),
center2{i}(j, 2)], 'Color', color(4, :), 'LineWidth', 1);
                end
            end
        end
    end

    center = point;
    [~, ~, main_line_length] = cost_fun(dot, center, center_load);
    cost_main = 325.7 * main_line_length;
    fprintf("最终花费%f 千元\n", cost_1 + cost_2 + cost_3 + cost_4 +
cost_main);
    total_cost(time) = cost_1 + cost_2 + cost_3 + cost_4 + cost_main;

    if time == times
        hold on;
        scatter(point(:, 1), point(:, 2), 500, '*');
        axis([-1.75, 2.25, -1.5, 2.5]);
    end
```

```
end


figure;
plot(total_cost);
title("双层规划下费用的变动情况");
xlabel("双层规划次数");
ylabel("费用");

reliability = zeros(1, len);
for i = 1:len
    reliability(1, i) = reliability_fun(0, [x(1, i), y(1, i)],
dot(1:individual_size, :), supply, cluster, center, cluster2, center2,
center2_load);
end

figure;
[~, ~, ~, ~, ~, ~, sorted_index] = reliability_fun(0, [x(1), y(1)], dot,
supply, cluster, center, cluster2, center2, center2_load);

topology_plot(center(sorted_index, :), cluster2(sorted_index),
center2(sorted_index) );

title("电网结构拓扑图");
```

程序二：聚类信息分析程序

count_num.m

```
function [flag] = count_num(index, k)

flag = zeros(k, 1);
for i = 1:length(index)
    for j = 1:k
        if index(i) == j
            flag(j) = flag(j) + 1;
            break;
        end
    end
end


end
```

程序三：外接矩形生成程序

bounding_box.m

```
function [boxx, boxy] = bounding_box(x, y, spacing);
% 此函数画出点集 X（坐标为 x 和 y）的外接矩形，在矩形中均匀生成一系列点

xx = min(x) : spacing : max(x); % 为生成点做准备
yy = min(y) : spacing : max(y); % 为生成点做准备

xlen = length(xx); % box 在 x 方向的长度（元素数量）
ylen = length(yy); % box 在 y 方向的长度（元素数量）

boxx = zeros(1, xlen * ylen); % 预分配内存
boxy = zeros(1, xlen * ylen); % 预分配内存

for i = 1 : ylen % 对每一行
    for j = 1 : xlen % 对每一列
        boxx(xlen * (i-1) + j) = xx(j);
    end
end

for i = 1 : ylen % 对每一行
    for j = 1 : xlen % 对每一列
        boxy(xlen * (i-1) + j) = yy(i);
    end
end

return;
```

程序四：遗传算法程序

Genetic_algorithm.m

```
function [best_individual, fit, run_time] = Genetic_algorithm(epoch,
pop_size, individual_size, pm, supply, center, center_load)

box_tmp = cat(1, center, supply);
xrange = max(box_tmp) - min(box_tmp);
yrange = xrange(2);
xrange = xrange(1);
minimun = min(box_tmp);

pop(:, 1, :) = xrange * rand([individual_size, 1, pop_size]) + minimun(1); %
```

```
随机产生初始群体
pop(:, 2, :) = yrange * rand([individual_size, 1, pop_size]) + minimun(2); %
随机产生初始群体

tmp(:, :, 1) = supply;
for i = 1:2*pop_size-1
    tmp = cat(3, tmp, supply); % 计算适应度时要带上电源
end
fitvalue = fitvalue_fun(cat(1, pop, tmp(:, :, 1:pop_size)), center,
center_load);

tic;
for i = 1:epoch % 遗传代数

    newpop = crossover(pop, fitvalue); % 交叉
    newpop = mutation(newpop, pm, 0.05 * min(xrange, yrange) ); % 变异
    newpop = cat(3, pop, newpop);

    fitvalue = fitvalue_fun(cat(1, newpop, tmp), center, center_load); %
计算群体中每个个体的适应度
    [pop, fitvalue] = selection(newpop, fitvalue); % 自然选择

    [~, bestfit] = best(pop, fitvalue); % 求出群体中最优的个体及其适应值
    fit(i) = bestfit;
    run_time(i) = toc;
    if mod(i, 50) == 0
        fprintf("---遗传第%d 轮迭代---\n", i);
        fprintf(" 目前花费为%.2f\n", fit(i) );
    end
end
toc;

% 挑出遗传算法结果中最好的解作为模拟退火的初始解
[best_individual, ~] = best(pop, fitvalue);

% figure;
% plot(fit);
% title('遗传算法下解的变动情况');
% xlabel('迭代次数');
% ylabel('费用');
```

程序五：遗传算法子程序——变异

mutation.m

```
function [pop] = mutation(pop, pm, pace)
% 值传递
[sx, sy, sz] = size(pop);
for i = 1:sz
    for j = 1:sx
        for k = 1:sy
            if rand(1) < pm
                pop(j, k, i) = pop(j, k, i) + 2*pace * rand(1) - pace;
            end
        end
    end
end

return;
```

程序六：遗传算法子程序——交叉

crossover.m

```
function [newpop] = crossover(pop, fitvalue)
% 交叉，更好的个体应该优先享有繁衍的权利
% 交叉规则：1：更好的个体参与交叉的概率更大，
% 2：对两个参与交叉的个体，随机选择对应的一段染色体片段，对这段上的每一位对应的数字做出如下操作：
% 个体 A 的某个染色体数字 a 对应着 个体 B 的某个染色体数字 b
% 更新 a 和 b 都等于 mean(a + b) + 1.5 * abs(a - b) * (rand - 0.5)
% 其中 rand 为[0, 1]中均匀分布的随机数

[sx, sy, sz] = size(pop);

newpop = ones(sx, sy, sz);
for i = 1:2:sz
    a = ceil(sz * rand(1) );
    b = ceil(sz * rand(1) );

    for j = 1:sy
        % 交叉的染色体长度
        cross_length = ceil(sx / 2 * rand(1) );
        % 交叉的染色体片段的起始索引
```

```
        cross_index = ceil(rand(1) * (sx - cross_length + 1) );
        % 两个个体在 交叉的染色体片段上的 每个染色体的均值
        mean_value = (pop(cross_index:cross_index+cross_length-1, j, a) +
pop(cross_index:cross_index+cross_length-1, j, b) ) / 2;
        % 两个个体 交叉的染色体片段上的 每个染色体的变化大小
        value_length = abs(pop(cross_index:cross_index+cross_length-1, j,
a) - pop(cross_index:cross_index+cross_length-1, j, b) );
        % 不交叉的位置直接复制
        newpop(:, :, i) = pop(:, :, a);
        newpop(:, :, i+1) = pop(:, :, b);
        % 对交叉方式进行试验
%                 newpop(cross_index:cross_index+cross_length-1, j, i) =
pop(cross_index:cross_index+cross_length-1, j, b);
%               newpop(cross_index:cross_index+cross_length-1, j, i+1) =
pop(cross_index:cross_index+cross_length-1, j, a);
        % 交叉得出新染色体片段
        for k = cross_index:cross_index+cross_length-1
            newpop(k, j, i) = 1.5 * value_length(k - cross_index + 1) *
(rand(1) - 0.5) + mean_value(k - cross_index + 1);
            newpop(k, j, i+1) = 1.5 * value_length(k - cross_index + 1) *
(rand(1) - 0.5) + mean_value(k - cross_index + 1);
        end
    end
end

return;
```

程序七：遗传算法子程序——选择

selection.m

```
function [newpop, fitvalue]=selection(pop, fitvalue)
% 自然选择优胜劣汰，选择最好的一半个体组成新群体

[~, ~, sz] = size(pop);
[~, idx] = sort(fitvalue, 'ascend'); % 从小到大排列

newpop = pop(:, :, idx(1:sz/2) );
fitvalue = fitvalue(idx(1:sz/2) );

return;
```

程序八：遗传算法子程序——适应度函数

fitvalue_fun.m

```
% 计算适应值,此适应值对应线路的花费，应越小越好

fitvalue = zeros(1, size(pop, 3) );
for i = 1:size(pop, 3)
    fitvalue(i) = cost_fun(pop(:, :, i), center, center_load);
end

return;
```

程序九：遗传算法子程序——选择最佳个体

best.m

```
function [best_individual, best_fitval] = best(pop, fitvalue)

[~, ~, sz] = size(pop);
best_individual = pop(:, :, 1);
best_fitval = fitvalue(1);
for i = 2:sz
    if fitvalue(i) < best_fitval
        best_individual = pop(:, :, i);
        best_fitval = fitvalue(i);
    end
end

return;
```

程序十：模拟退火算法程序

Simulated_annealing.m

```
function [dot, cost, run_time] = Simulated_annealing(T, innerloop, dc, dot,
supply, center, center_load, spacing, boxx, boxy)
% 模拟退火算法

previous_cost = cost_fun(cat(1, supply, dot), center, center_load);
```

```
epoch = 0; % 外层循环计数器

% 开始退火
tic;
while T > 0.01
    for i = 1:innerloop
        % 扰动:
        new_dot = dot_disturb(dot, boxx, boxy, spacing);
        % 计算扰动后的 cost
        new_cost = cost_fun(cat(1, supply, new_dot), center, center_load) ;
        delta_cost = new_cost - previous_cost;
        if delta_cost < 0
            dot = new_dot;
            previous_cost = new_cost;
        else
            if exp(-delta_cost/T)>rand()
               dot = new_dot;
               previous_cost = new_cost;
            end
        end
    end
    epoch = epoch + 1;

    cost(epoch) = previous_cost;
    run_time(epoch) = toc;
    T = T * dc; % 退温系数
    if mod(epoch, 50) == 0
        fprintf("---退火第%d 轮迭代---\n", epoch);
        fprintf("  目前温度为%.2f\n", T);
        fprintf("  目前花费为%.2f\n", previous_cost);
    end
end
toc;

% figure;
% plot(cost);
% title('模拟退火算法下解的变动情况');
% xlabel('迭代次数');
% ylabel('费用');

return;
```

程序十一：模拟退火算法子程序——扰动

dot_disturb.m

```
function dot = dot_disturb(dot, boxx, boxy, spacing)
% 此函数对点进行扰动

size = length(dot);
disturb_num = ceil(size * rand(1) );
disturb_index = ceil(size * rand(1, disturb_num) ); % 扰动的点的索引
% disturbed = ceil(length(boxx) * rand(1, disturb_num) ); % 扰动后点的 box 索引 % 一开始用的方法
for i = 1:disturb_num
    % 对每个要扰动的点，在 x 方向上走[-2, 2]个 spacing
    dot(disturb_index(i), 1) = dot(disturb_index(i), 1) + (floor(5 * rand(1) ) - 2) * spacing;
    % 对每个要扰动的点，在 y 方向上走[-2, 2]个 spacing
    dot(disturb_index(i), 2) = dot(disturb_index(i), 2) + (floor(5 * rand(1) ) - 2) * spacing;
end
% dot(disturb_index, 1:2) = [boxx(disturbed)', boxy(disturbed)']; % 一开始用的方法

% matlab 函数默认值传递，改变传入形参对 main 函数的实参没有影响
return;
```

程序十二：模拟退火算法子程序——代价函数

cost_fun.m

```
function [money, node, L] = cost_fun(dot, center, center_load)
% 此函数算出总线和一级支线的费用之和

[L, node] = minspan(dot); % 用一条最短的折线把 dot 连接起来

D = zeros(1, size(center, 1) ); % 一级支线长度
distance = zeros(1, size(node, 1) ); % 某中心到总线每条线段的距离
for i = 1:size(center, 1)
    for j = 1:size(node, 1)
        distance(j) = distance_fun(center(i, :), dot(node(j, 1), :), dot(node(j, 2), :) ); % 第 i 个中心到所有线段的距离
    end
    D(i) = min(distance); % 第 i 个中心到总线的最短距离
```

```
end

money = 325.7 * L;
for i = 1:size(center, 1)
    if center_load(i) >= 3
        money = money + 239.4 * D(i);
    else
        money = money + 188.6 * D(i);
    end
end

return;
```

程序十三：最小生成树程序

minspan.m

```
function [length, endnode, weight] = minspan(dot)
% 函数目的：用一条最短的折线把 dot 连接起来

len = size(dot, 1);

weight = zeros(1, len^2); % 预分配内存
for i = 1:len
    for j = 1:len
        % 计算权重，值为各点间的欧氏距离
        weight(len*(i-1) + j) = sqrt( (dot(i, 1)-dot(j, 1))^2 + (dot(i, 2)-
dot(j, 2))^2 );
    end
end

a = zeros(1, len^2); % 预分配内存
for i = 1:len
    for j = 1:len
        % 为了方便图论计算，a 为各边起点
        a(len*(i-1) + j) = i;
    end
end

b = zeros(1, len^2); % 预分配内存
for i = 1:len:len^2 -len+1
    % 为了方便图论计算，b 为各边终点
```

```
    b(i:i+len-1) = 1:len;
end


G = graph(a, b, weight); % 根据 各边起点 a 和 各边起点 b 创建图论图
T = minspantree(G); % 计算 最小生成树，目的是用一条最短的折线把 dot 连接起来
endnode = T.Edges.EndNodes;
weight = T.Edges.Weight;
length = sum(weight);

return;
```

程序十四：距离计算程序

distance_fun.m

```
function [distance, nearest_dot] = distance_fun(p, a, b)
% 计算点 p 到一条线段的距离，这条线段以 a 和 b 为端点

if a == b
    distance = norm(p - a);
    nearest_dot = a;
    return;
end
ap = p - a; % 向量 ap
ab = b - a; % 向量 ab
projection = ap * ab' / norm(ab)^2 * ab; % ap 向 ab 作投影
factor = projection / ab;
if factor <= 0
    distance = norm(p - a);
    nearest_dot = a;
elseif factor >= 1
    distance = norm(p - b);
    nearest_dot = b;
else
    distance = norm(ap - projection);
    nearest_dot = p - ap + projection;
end

return;
```

程序十五：下层优化函数

optimizie_fun.m

```
function cost = optimize_fun(x)

global center2; global center2_load; global edge;
global i;  global dott; global center_load;

cost = 0;
d = zeros(1, size(edge, 1) );
for k = 1:size(edge, 1)
    % 点 x 到总线各段的距离
    d(i) = distance_fun(x, dott(edge(k, 1), :), dott(edge(k, 2), :) );
end
distance = min(d); % 点 x 到总线的最短距离

if center_load(i) >= 3
    weight = 239.4; % 支线 B
else
    weight = 188.6; % 支线 A
end

cost = weight * distance;
for j = 1:size(center2{i}, 1)
    if center2_load{i}(j) >= 3
        weight = 239.4; % 支线 B
    else
        weight = 188.6; % 支线 A
    end
    cost = cost + weight * sqrt( ( center2{i}(j, 1) - x(1) )^2 +
( center2{i}(j, 2) - x(2) )^2 );
end

return;
```

程序十六：可靠性计算程序

reliability_fun.m

```
function [reliability, sub_line_reliability, main_line_reliability, p,
length, sorted_nearest_dot, sorted_index] = reliability_fun(option,
target, dot, supply, cluster, center, cluster2, center2, center2_load)
% 计算单供网络的可靠性
```

```
main_line = cat(1, dot, supply);  % 总线上的点
[~, endnode] = minspan(main_line);  % 总线
tmp_distance = zeros(1, size(endnode, 1) );  % 某个中心到总线每段的距离
tmp_nearest_dot = zeros(size(endnode, 1), 2);  % 某个中心到总线每段最近的点
distance = zeros(1, size(center, 1) );   % 某个中心到总线的最短距离
nearest_dot = zeros(size(center, 1), 2);  % 总线到某个中心的最近的点

% 计算每个中心到总线的最短距离和最近点（一级分叉点）
for i = 1:size(center, 1)
    for j = 1:size(endnode, 1)
        % 第 i 个中心到总线各段的最短距离
        [tmp_distance(j),          tmp_nearest_dot(j,          :)]          =
distance_fun(center(i,      :),      main_line(endnode(j,      1),      :),
main_line(endnode(j, 2), :) );
    end
    [distance(i), min_index] = min(tmp_distance); % 第 i 个中心到总线的最短距
离
    nearest_dot(i, :) = tmp_nearest_dot(min_index, :); % 第 i 个中心对应的一
级分叉点
end

dist_from_supply = dist(nearest_dot, supply');  % 一级分叉点到电源的欧式距离
[~, sorted_index] = sort(dist_from_supply, 'ascend');  % 距离从小到大排序
sorted_nearest_dot = nearest_dot(sorted_index, :);
length = zeros(1, size(sorted_index, 1) ); % 总线各段长度
length(1, 1) = dist(nearest_dot(sorted_index(1), :), supply'); % 第一段是
最近的一级分叉点到电源的距离
for i = 2:size(sorted_index, 1)
    % 总线各段长度：各两个一级分叉点间的距离
    length(1,     i)     =     dist(nearest_dot(sorted_index(i-1),     :),
nearest_dot(sorted_index(i), :)' );
end

possibility = cumprod(1 - 0.002 * length);  % 第 i 个一级分叉点处故障概率
for i = 1:size(possibility, 2)
    possibility(i) = possibility(i) * (1 - 0.002) ^ i; % 开关故障率 0.002
    possibility(i) = possibility(i) * (1 - 0.005); % 电源故障率 0.005
end

if option == 0
    for i = 1:size(cluster, 2)
        for j = 1:size(cluster2{i}, 2)
            for k = 1:center2_load{i}(j)
                if  target(1)  ==  cluster2{i}{j}(k,  1)  &&  target(2)  ==
```

```
cluster2{i}{j}(k, 2)
                    length2 = dist(target, center2{i}(j, :)' ); % 三级支线
                    length2    =    length2    +    dist(center2{i}(j,    :),
center(i, :)' ); % 二级支线
                    length2    =    length2    +    dist(center(i,    :),
nearest_dot(i, :)' ); % 一级支线
                    % 负荷可靠性 = 总线部分可靠性 * 支线部分可靠性（开关故障率
0.002 用户故障率 0.005）
                    main_line_reliability = possibility(sorted_index == i);
                    sub_line_reliability = (1 - 0.002 * length2) * 0.998 *
0.995;
                    reliability       =       main_line_reliability       *
sub_line_reliability;
                    p = nearest_dot(i, :); % p 是对应的一级分叉点
                    return;
                end
            end
        end
    end
end

if option == 1
    for i = 1:size(center, 1)
        if nearest_dot(sorted_index(i), :) == target
            reliability = possibility(i);
            main_line_reliability = possibility(i);
            sub_line_reliability = 1;
            p = target;
            return;
        end
    end
end

reliability = -1; % 报错
return;
```

程序十七：单供电网拓扑图绘制程序

topology_plot.m

```
function topology_plot(center, cluster2, center2)

Color = [250 192 15; 1 86 153; 243 118 74; 95 198 201; 79 89 100] / 255;
```

```
Size = [10, 8, 7, 6, 4];
main_line = ["supply"];
name = ["supply", "P", "C", "c", "n"];
NodeSize = Size(1);
Node_Color = Color(1, :);

for i = 1 : size(center, 1)
    % 一级分叉点
    main_line = cat(2, main_line, name(2)+num2str(i) );
    NodeSize = cat(2, NodeSize, Size(2) );
    Node_Color = cat(1, Node_Color, Color(2, :) );
end
G = graph(diag(ones(1, size(center, 1) ), 1), main_line, "upper");

counter = 1;
for i = 1 : size(center, 1)
    G = G.addnode(name(3)+num2str(i) );
    G = G.addedge(name(2)+num2str(i), name(3)+num2str(i), 1);
    Node_Color = cat(1, Node_Color, Color(3, :) );
    NodeSize = cat(2, NodeSize, Size(3) );
    for j = 1 : size(center2{i}, 1)
        if size(cell2mat(cluster2{i}(j) ), 1) == 1
            G = G.addnode(name(5)+num2str(counter) );
            G = G.addedge(name(3)+num2str(i), name(5)+num2str(counter), 1);
            counter = counter + 1;
            NodeSize = cat(2, NodeSize, Size(5) );
            Node_Color = cat(1, Node_Color, Color(5, :) );
        else
            G = G.addnode(name(4)+num2str(i)+num2str(j) );
            G                =                G.addedge(name(3)+num2str(i),
name(4)+num2str(i)+num2str(j), 1);
            Node_Color = cat(1, Node_Color, Color(4, :) );
            NodeSize = cat(2, NodeSize, Size(4) );
            for k = 1 : size(cell2mat(cluster2{i}(j) ), 1)
                G = G.addnode(name(5)+num2str(counter) );
                G        =        G.addedge(name(4)+num2str(i)+num2str(j),
name(5)+num2str(counter), 1);
                counter = counter + 1;
                NodeSize = cat(2, NodeSize, Size(5) );
                Node_Color = cat(1, Node_Color, Color(5, :) );
            end
        end
    end
end
```

```
hold on;
h = plot(G, "Layout", "force");
h.MarkerSize = NodeSize;
layout(h,'force','UseGravity',true);
tmp = cell(1, size(h.MarkerSize, 2) );
for i = 1:size(h.MarkerSize, 2)
     tmp{i} = ' ';
end
h.NodeLabel = tmp;
h.NodeColor = Node_Color;
h.EdgeColor = [0 0 0];
```

程序十八：双供电网拓扑图绘制程序

topology_plot2.m

```
function h = topology_plot2(center, cluster2, center2, center_2,
cluster2_2, center2_2, contact)

Color = [250 192 15; 1 86 153; 243 118 74; 95 198 201; 79 89 100; 1 86
240; ] / 255;
Size = [10, 8, 7, 6, 4, 0.01];
main_line = ["电源 1"];
name = ["电源 1", "P", "C", "c", "n", "m"];
main_line_2 = ["电源 2"];
name_2 = ["电源 2", "P.", "C.", "c.", "n.", "m."];
NodeSize = Size(1);
Node_Color = Color(1, :);

for i = 1 : size(center, 1)
    % 一级分叉点
    main_line = cat(2, main_line, name(6)+num2str(i) );
    main_line = cat(2, main_line, name(2)+num2str(i) );
    NodeSize = cat(2, NodeSize, Size(6) );
    NodeSize = cat(2, NodeSize, Size(2) );
    Node_Color = cat(1, Node_Color, Color(6, :) );
    Node_Color = cat(1, Node_Color, Color(2, :) );
end
NodeSize = cat(2, NodeSize, Size(1) );
Node_Color = cat(1, Node_Color, Color(1, :) );
for i = 1 : size(center_2, 1)
    % 一级分叉点
```

```
    main_line_2 = cat(2, main_line_2, name_2(6)+num2str(i) );
    main_line_2 = cat(2, main_line_2, name_2(2)+num2str(i) );
    NodeSize = cat(2, NodeSize, Size(6) );
    NodeSize = cat(2, NodeSize, Size(2) );
    Node_Color = cat(1, Node_Color, Color(6, :) );
    Node_Color = cat(1, Node_Color, Color(2, :) );
end

tmp = diag(ones(1, 2*size(cat(1, center, center_2), 1)+1 ), 1);
tmp(1 + 2*size(center, 1), 2 + 2*size(center, 1) ) = 0;
G = graph(tmp, cat(2, main_line, main_line_2), "upper");

for i = 1:size(contact, 1)
    if mod(contact(i, 1), 2) == 0
        left = name(6) + num2str(contact(i, 1) / 2);
    else
        left = name(2) + num2str(floor(contact(i, 1) / 2) );
    end
    if mod(contact(i, 2), 2) == 0
        right = name_2(6) + num2str(contact(i, 2) / 2);
    else
        right = name_2(2) + num2str(floor(contact(i, 2) / 2) );
    end
    G = G.addedge(left, right, 1);
end



counter = 1;
for i = 1 : size(center, 1)
    G = G.addnode(name(3)+num2str(i) );
    G = G.addedge(name(2)+num2str(i), name(3)+num2str(i), 1);
    NodeSize = cat(2, NodeSize, Size(3) );
    Node_Color = cat(1, Node_Color, Color(3, :) );
    for j = 1 : size(center2{i}, 1)
        if size(cell2mat(cluster2{i}(j) ), 1) == 1
            G = G.addnode(name(5)+num2str(counter) );
            G = G.addedge(name(3)+num2str(i), name(5)+num2str(counter), 1);
            counter = counter + 1;
            NodeSize = cat(2, NodeSize, Size(5) );
            Node_Color = cat(1, Node_Color, Color(5, :) );
        else
            G = G.addnode(name(4)+num2str(i)+num2str(j) );
            G                =                G.addedge(name(3)+num2str(i),
name(4)+num2str(i)+num2str(j), 1);
```

```
            NodeSize = cat(2, NodeSize, Size(4) );
            Node_Color = cat(1, Node_Color, Color(4, :) );
            for k = 1 : size(cell2mat(cluster2{i}(j) ), 1)
                G = G.addnode(name(5)+num2str(counter) );
                G        =        G.addedge(name(4)+num2str(i)+num2str(j),
name(5)+num2str(counter), 1);
                counter = counter + 1;
                NodeSize = cat(2, NodeSize, Size(5) );
                Node_Color = cat(1, Node_Color, Color(5, :) );
            end
        end
    end
end


counter = 1;
for i = 1 : size(center_2, 1)
    G = G.addnode(name_2(3)+num2str(i) );
    G = G.addedge(name_2(2)+num2str(i), name_2(3)+num2str(i), 1);
    NodeSize = cat(2, NodeSize, Size(3) );
    Node_Color = cat(1, Node_Color, Color(3, :) );
    for j = 1 : size(center2_2{i}, 1)
        if size(cell2mat(cluster2_2{i}(j) ), 1) == 1
            G = G.addnode(name_2(5)+num2str(counter) );
            G = G.addedge(name_2(3)+num2str(i), name_2(5)+num2str(counter),
1);
            counter = counter + 1;
            NodeSize = cat(2, NodeSize, Size(5) );
            Node_Color = cat(1, Node_Color, Color(5, :) );
        else
            G = G.addnode(name_2(4)+num2str(i)+num2str(j) );
            G               =               G.addedge(name_2(3)+num2str(i),
name_2(4)+num2str(i)+num2str(j), 1);
            NodeSize = cat(2, NodeSize, Size(4) );
            Node_Color = cat(1, Node_Color, Color(4, :) );
            for k = 1 : size(cell2mat(cluster2_2{i}(j) ), 1)
                G = G.addnode(name_2(5)+num2str(counter) );
                G       =       G.addedge(name_2(4)+num2str(i)+num2str(j),
name_2(5)+num2str(counter), 1);
                counter = counter + 1;
                NodeSize = cat(2, NodeSize, Size(5) );
                Node_Color = cat(1, Node_Color, Color(5, :) );
            end
        end
```

```
    end
end

hold on;
h = plot(G, "Layout", "force");
% h = plot(G, "Layout", "layered");

h.MarkerSize = NodeSize;
h.NodeColor = Node_Color;
layout(h,'force','UseGravity',true)


tmp = cell(1, size(h.MarkerSize, 2) );
% for i = 1:size(h.MarkerSize, 2)
%     if i <= 2*size(center, 1)+1 && mod(i, 2) == 1
%         tmp{i} = h.NodeLabel{i};
%        elseif i > 2*size(center, 1)+1 && i <= 2*size(cat(1, center,
center_2), 1)+2 && mod(i, 2) == 0
%         tmp{i} = h.NodeLabel{i};
%     else
%         tmp{i} = ' ';
%     end
% end
for i = 1:size(h.MarkerSize, 2)
     tmp{i} = ' ';
end
h.NodeLabel = tmp;
h.NodeFontSize = 6;
h.EdgeColor = [0 0 0];
h.LineWidth = 0.5;
```

程序十九：第一问的过程函数

B1_fun.m

```
function [cost, info] = B1_fun(x, y, supply, cluster_num, idx, center,
idxidx, power, ttt)

global center2; global center2_load; global edge;
global i;  global dott; global center_load;
len = size(x, 1);
```

```
% 画出外接矩形，生成均匀点集
box_tmp = cat(1, center, supply);
spacing = 0.01 * min( max(box_tmp) - min(box_tmp) ); % 生成点的间距为外接矩形的宽度的 1%
[boxx, boxy] = bounding_box(box_tmp(:, 1), box_tmp(:, 2), spacing); % 画出外接矩形，生成均匀点集


color = [0 0 0; 0 0 255; 0 255 0; 95 198 201] / 255;
for kkk = 1:ttt
    % 遗传算法
    epoch = 500;
    pop_size = 30; % 群体大小
    individual_size = 20; % 个体长度
    pm = 0.1; % 变异概率
    [best_individual,    ~]    =    Genetic_algorithm(epoch,    pop_size,
individual_size, pm, supply, center, center_load);



    % 进行模拟退火
    temterature = 5;  % 初始温度
    innerloop = 150;    % 马尔科夫链长 （内层循环次数）
    dc = 0.99; % 退温系数 Dewarming_coefficient
    spacing = 0.01 * min( max(center) - min(center) ); % 扰动步长为外接矩形的宽度的 1%

    % 将遗传算法的最优解作为初始点的位置
    dot_index = dsearchn([boxx', boxy'], best_individual);
    dot = [boxx(dot_index)', boxy(dot_index)'];
    previous_cost = cost_fun(cat(1, supply, dot), center, center_load);

    [dot, ~] = Simulated_annealing(temterature, innerloop, dc, dot, supply,
center, center_load, spacing, boxx, boxy);




    % 画图检查结果
    hold on;
    % scatter(center(:, 1), center(:, 2), 125); % 画出各簇中心
    % hold on;
    % scatter(supply(1), supply(2), 200);

    dot = cat(1, dot, supply);
    dotx = dot(:, 1)';
```

```
    doty = dot(:, 2)';
    [~, edge] = cost_fun(dot, center, center_load);

    % hold on;
    % scatter(dotx, doty, 50);
    % hold on;
    %  line([dotx(edge(:,  1));  dotx(edge(:,  2))],  [doty(edge(:,  1));
doty(edge(:, 2))] );


    % 画图
    if kkk == ttt
        hold on;
        gscatter(x, y, idx);
        hold on;
        % scatter(center(:, 1), center(:, 2), 200, "diamond");
        hold on;
        scatter(supply(:, 1), supply(:, 2), 700);
        hold on;
        line([dotx(edge(:,  1));  dotx(edge(:,  2))],  [doty(edge(:,  1));
doty(edge(:, 2))], 'Color', color(1, :), 'LineWidth', 4)
        hold on;
    end


    dott = dot;
    % 第二层规划
    cluster = cell(1, cluster_num); % 第一次聚类中的簇
    idx2 = cell(1, cluster_num); % 第二次聚类中各点的归属
    center2 = cell(1, cluster_num); % 第二次聚类中各中心
    center2_load = cell(1, cluster_num); % 第二次聚类中各簇点数
    cluster2 = cell(1, cluster_num); % 第二次聚类中各簇
    point = zeros(cluster_num, 2); % 优化后的二级分叉点坐标
    nearest = zeros(size(edge, 1), 2);

    cost_1 = 0; cost_2 = 0; cost_3 = 0; cost_main = 0;
    cost_4 = 2.6 * len + 56.8 * cluster_num;

    for i = 1:cluster_num  % 对第一层的每个聚类中心
        % 将第一层聚类的每个簇提取出来
        cluster{i} = [x(idx == idxidx(i))', y(idx == idxidx(i))'];

        % 根据每簇的点个数 进行 第二层聚类
        clear SilhouetteCoefficient;
```

```
        for n = 1:ceil(center_load(idxidx(i) ) / 2 + 1)
            [idx2{i}, center2{i}] = kmeans(cluster{i}, n, 'Replicates',
200);
            SilhouetteCoefficient(n)     =     mean(silhouette(cluster{i},
idx2{i}) );
            if center_load(idxidx(i) ) == 1
                break;
            end
        end
        [~, cluster2_num] = max(SilhouetteCoefficient);
        [idx2{i},    center2{i}]    =    kmeans(cluster{i},    cluster2_num,
'Replicates', 500);
        center2_load{i} = count_num(idx2{i}, cluster2_num);
        cluster2{i} = cell(1, cluster2_num);
        for t = 1:cluster2_num
            cluster2{i}{t} = cluster{i}(idx2{i} == t, :);
        end



        hold on;
        %     scatter(center2{i}(:, 1), center2{i}(:, 2), 100);


        % 以 一级支线的费用和二级支线的费用最小为目标 优化二级分叉点
        point(i, :) = fminsearch(@optimize_fun, rand(1, 2));
        for n = 1:99
            point(i, :) = point(i, :) + fminsearch(@optimize_fun, rand(1,
2));
        end
        point(i, :) = point(i, :) / 100;

        % 计算费用
        % 计算一级支线费用
        d = zeros(1, size(edge, 1));
        for j = 1:size(edge, 1)
            % 二级分叉点到总线每段的距离
            [d(1, j), nearest(j, :)] = distance_fun(point(i, :), dot(edge(j,
1), :), dot(edge(j, 2), :) );
        end
        % 二级分叉点到总线的最短距离
        [distance, min_index] = min(d);
        hold on;
        if kkk == ttt
            line([point(i,  1),  nearest(min_index,  1)],  [point(i,  2),
```

```
nearest(min_index, 2)], 'Color', color(2, :), 'LineWidth', 3);
        end

        if center_load(idxidx(i) ) >= 3
            weight = 239.4; % 支线 B
        else
            weight = 188.6; % 支线 A
        end
        % 加上一级支线的价格
        cost_1 = cost_1 + weight * distance;

        % 计算二级支线费用
        for j = 1:cluster2_num % 对第二层聚类的每簇
            % 第 i 簇里面的第 j 簇的负荷数
            if center2_load{i}(j) >= 3
                weight = 239.4; % 支线 B
            else
                weight = 188.6; % 支线 A
            end
            cost_2 = cost_2 + weight * dist(point(i, :), center2{i}(j, :)');
            hold on;
            if kkk == ttt
                line([point(i,  1),  center2{i}(j,  1)],  [point(i,  2),
center2{i}(j, 2)], 'Color', color(3, :), 'LineWidth', 2);
            end

            % 计算三级支线费用
            weight = 188.6; % 支线 A
            distance = dist(cluster{i}(idx2{i} == j, :), center2{i}(j, :)');
            cost_3 = cost_3 + weight * sum(distance);

            for k = 1:center2_load{i}(j) % 对第二层每簇的每个点
                box_tmp = cluster{i}(idx2{i} == j, :);
                hold on; % 画图，连线
                if kkk == ttt
                    line([box_tmp(k, 1), center2{i}(j, 1)], [box_tmp(k, 2),
center2{i}(j, 2)], 'Color', color(4, :), 'LineWidth', 1);
                end
            end

        end

    end
```

```
    center = point;
    [~, ~, main_line_length] = cost_fun(dot, center, center_load);
    cost_main = 325.7 * main_line_length;
    cost = cost_1 + cost_2 + cost_3 + cost_4 + cost_main;

    hold on;
    scatter(point(:, 1), point(:, 2), 200, '*');
    axis([-2, 2, -1, 3]);

end

info = cell(1, 7);
info{1} = dot(1:individual_size, :);
info{2} = cluster;
info{3} = center;
info{4} = cluster2;
info{5} = center2;
info{6} = center2_load;
for i = 1:size(info{2}, 2)
    for j = 1:size(info{2}{i}, 1)
        for k = 1:size(x, 2)
            if x(k) == info{2}{i}(j, 1) && y(k) == info{2}{i}(j, 2)
                break;
            end
        end
        info{7}{i}(j) = power(k);
    end
end
```

程序二十：第二三四问主程序

B_2.m

```
clear; clc;

global center2; global center2_load; global edge;
global i;  global dott; global center_load;

% 读取/生成数据部分
data = readmatrix("电网数据.xlsx");
supply = data(1:2, 1:2);
x = data(4:size(data, 1), 1)'; % 点的 x 坐标
```

```
y = data(4:size(data, 1), 2)'; % 点的 y 坐标
power = data(4:size(data, 1), 3); % 各个负荷的功率
len = length(x);


% 第一层聚类部分，idx 为各负荷所属簇的下标，center 为各簇中心坐标
cluster_num = 9; % 聚类簇数 k
[idx, center] = kmeans(cat(1, x, y)', cluster_num, 'Replicates', 2000); %
聚类 2000 次，选 SSD 最小的结果
center_load = count_num(idx, cluster_num); % 每簇对应的负荷数
% gscatter(x, y, idx); % 根据聚类结果，用不同颜色画出各点


% 将所有簇分成两组，分别对应 0，1 两个电源
% 用二进制串表示每个簇的归属，0 表示此簇归属到电源 0
idxx = 1:cluster_num;
belong = zeros(1, cluster_num); % 长度为 cluster_num 的二进制串
L = zeros(1, 2^cluster_num-2);

% 算出最有可能的组合情况
for n = 1:2^cluster_num-2
    num = n;
    for m = cluster_num:-1:1  % 利用反复除 2 的方法得到二进制串
        belong(1, m) = mod(num, 2);
        num = floor(num / 2);
    end

    L0 = minspan(cat(1, center(belong==0, :), supply(1, :) ) );
    L1 = minspan(cat(1, center(belong==1, :), supply(2, :)) );

    L(n) = max(L1, L0);  % 等于 (abs(L0 -L1) + L0 + L1) / 2

%     L(n) = L0 + L1;
    if sum(belong == 1) < 2 || sum(belong == 0) < 2  % 簇数约束
        L(n) = 99999;
    end
end

[~, sorted_index] = sort(L, 'ascend');


% 对可能的组合情况运用第一问的算法
ttt = 2; % 双层规划次数
for n = 4:4
```

```
    num = sorted_index(n);
    for m = cluster_num:-1:1  % 利用反复除 2 的方法得到二进制串
        belong(1, m) = mod(num, 2);
        num = floor(num / 2);
    end
    figure;
    idxidx = idxx(belong == 0);
    [cost0, info0] = B1_fun(x, y, supply(1, :), cluster_num-sum(belong),
idx, center(belong==0, :), idxidx, power, ttt);
    idxidx = idxx(belong == 1);
    [cost1, info1] = B1_fun(x, y, supply(2, :), sum(belong), idx,
center(belong==1, :), idxidx, power, ttt);
    total_cost(n) = cost0 + cost1;
    fprintf("最终花费%f 千元\n", total_cost(n) );


    % 化整，方便背包问题求解
    for k = 1:size(info0{7}, 2)
        info0{7}{k} = round(info0{7}{k} * 100);
    end
    for k = 1:size(info1{7}, 2)
        info1{7}{k} = round(info1{7}{k} * 100);
    end
end


min_re = 1;  % 最低可靠性
sum_re = 0;
for i = 1:35
    for m = 1:size(info0{2}, 2) % 簇
        for n = 1:size(info0{2}{m}, 1)
            if x(i) == info0{2}{m}(n, 1) && y(i) == info0{2}{m}(n, 2)
                re(i) = reliability_fun(0, info0{2}{m}(n, :), info0{1},
supply(1, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6});
                sum_re = sum_re + re(i);
                if re(i) < min_re
                    min_re = re(i);
                end
            end
        end
    end
    for m = 1:size(info1{2}, 2)
        for n = 1:size(info1{2}{m}, 1)
            if x(i) == info1{2}{m}(n, 1) && y(i) == info1{2}{m}(n, 2)
```

```
                re(i) = reliability_fun(0, info1{2}{m}(n, :), info1{1},
supply(2, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6});
                sum_re = sum_re + re(i);
                if re(i) < min_re
                    min_re = re(i);
                end
            end
        end
    end
end

fprintf("不加联络线的情况下最低可靠性为%f\n", min_re);


% 第三第四问
info0 = load("info0.mat");
info1 = load("info1.mat");
info0 = struct2cell(info0);
info1 = struct2cell(info1);
info0 = info0{1};
info1 = info1{1};

% % 第三问 一条线的情况
% [val, min_reliability, len1, len2, len3, cost, contact, h] =
reliability_fun_3(supply, info0, info1, 500, 0.98, 3, [1 0 0], 1);
% % 第三问 两条线的情况
% [val, min_reliability, len1, len2, len3, cost, contact, h] =
reliability_fun_3(supply, info0, info1, 1100, 0.98, 3, [1 0 0], 1);
% % 第三问 三条线的情况
% [val, min_reliability, len1, len2, len3, cost, contact, h] =
reliability_fun_3(supply, info0, info1, 1700, 0.98, 3, [1 0 0], 1);
% % 第四问 一条线的情况
% [val, min_reliability, len, cost, contact, h] = reliability_fun_1(supply,
info0, info1, 10000, 0.973, 4, [1 0 0], 1);
% % 第四问 两条线的情况
% [val, min_reliability, len, cost, contact, h] = reliability_fun_1(supply,
info0, info1, 10000, 0.9784, 4, [1 0 0], 1);
% % 第四问 三条线的情况
% [val, min_reliability, len, cost, contact, h] = reliability_fun_1(supply,
info0, info1, 10000, 0.978439, 4, [1 0 0], 1);
```

程序二十一：可靠性计算程序

reliability_fun0.m

```
function [nearest_dot, sorted_index, possibility] = reliability_fun0(dot, supply, center)
% 计算单供网络的可靠性

main_line = cat(1, dot, supply);  % 总线上的点
[~, endnode] = minspan(main_line);  % 总线
tmp_distance = zeros(1, size(endnode, 1) );  % 某个中心到总线每段的距离
tmp_nearest_dot = zeros(size(endnode, 1), 2);  % 某个中心到总线每段最近的点
distance = zeros(1, size(center, 1) );   % 某个中心到总线的最短距离
nearest_dot = zeros(size(center, 1), 2);  % 总线到某个中心的最近的点

% 计算每个中心到总线的最短距离和最近点（一级分叉点）
for i = 1:size(center, 1)
    for j = 1:size(endnode, 1)
        % 第 i 个中心到总线各段的最短距离
        [tmp_distance(j),          tmp_nearest_dot(j,          :)]          =
distance_fun(center(i,      :),      main_line(endnode(j,      1),      :),
main_line(endnode(j, 2), :) );
    end
    [distance(i), min_index] = min(tmp_distance); % 第 i 个中心到总线的最短距离
    nearest_dot(i, :) = tmp_nearest_dot(min_index, :); % 第 i 个中心对应的一级分叉点
end

dist_from_supply = dist(nearest_dot, supply');  % 一级分叉点到电源的欧式距离
[~, sorted_index] = sort(dist_from_supply, 'ascend');  % 距离从小到大排序
length = zeros(1, size(sorted_index, 1) ); % 总线各段长度
length(1, 1) = dist(nearest_dot(sorted_index(1), :), supply'); % 第一段是最近的一级分叉点到电源的距离
for i = 2:size(sorted_index, 1)
    % 总线各段长度：各两个一级分叉点间的距离
    length(1,     i)     =     dist(nearest_dot(sorted_index(i-1),     :),
nearest_dot(sorted_index(i), :)' );
end

possibility = cumprod(1 - 0.002 * length);  % 第 i 个一级分叉点处故障概率
for i = 1:size(possibility, 2)
    possibility(i) = possibility(i) * (1 - 0.002) ^ i; % 开关故障率 0.002
    possibility(i) = possibility(i) * (1 - 0.005); % 电源故障率 0.005
end
```

```
nearest_dot = nearest_dot(sorted_index, :);

return;
```

程序二十二：可靠性计算程序

reliability_fun00.m

```
function [reliability, sub_line_reliability, main_line_reliability] = reliability_fun00(option, target, cluster, center, cluster2, center2, center2_load, nearest_dot, sorted_index, possibility)
if option == 0
    for i = 1:size(cluster, 2)
        for j = 1:size(cluster2{i}, 2)
            for k = 1:center2_load{i}(j)
                if target(1) == cluster2{i}{j}(k, 1) && target(2) == cluster2{i}{j}(k, 2)
                    length2 = dist(target, center2{i}(j, :)' ); % 三级支线
                    length2 = length2 + dist(center2{i}(j, :), center(i, :)' ); % 二级支线
                    length2 = length2 + dist(center(i, :), nearest_dot(i, :)' ); % 一级支线
                    % 负荷可靠性 = 总线部分可靠性 * 支线部分可靠性（开关故障率 0.002 用户故障率 0.005）
                    main_line_reliability = possibility(i);
                    sub_line_reliability = (1 - 0.002 * length2) * 0.998 * 0.995;
                    reliability = main_line_reliability * sub_line_reliability;
                    p = nearest_dot(i, :); % p 是对应的一级分叉点
                    return;
                end
            end
        end
    end
end
if option == 1
    for i = 1:size(center, 1)
        if nearest_dot(i, :) == target
            reliability = possibility(i);
            main_line_reliability = possibility(i);
            sub_line_reliability = 1;
            p = target;
```

```
            return;
        end
    end
end
reliability = -1; % 报错
return;
```

程序二十三：可靠性计算程序——双供单联络线

reliability_fun_1.m

```
function [val, min_reliability, len1, cost, contact, h, avg] = reliability_fun_1(supply, info0, info1, money, request_min_reliability, target, color, invoke_time)
% 计算双供网络，一条联络线情况下的可靠性


% 求电源 0 的各个一级分叉点
[~, ~, ~, ~, length0, nearest_dot0, sorted_index0] = reliability_fun(0, info0{2}{1}(1, :), info0{1}, supply(1, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6});

% 求电源 1 的各个一级分叉点
[~, ~, ~, ~, length1, nearest_dot1, sorted_index1] = reliability_fun(0, info1{2}{1}(1, :), info1{1}, supply(2, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6});


% 计算可供功率上限
capacity0 = 0; capacity1 = 0;
for k = 1:size(info0{7}, 2)
    capacity0 = capacity0 + sum(info0{7}{k});
end
for k = 1:size(info1{7}, 2)
    capacity1 = capacity1 + sum(info1{7}{k});
end
capacity0 = capacity0 * 1.1 * 0.5;
capacity1 = capacity1 * 1.1 * 0.5;
capacity = round([capacity0, capacity1]);


if invoke_time == 1
    % 换顺序
```

```
    info0{2} = info0{2}(sorted_index0);
    info1{2} = info1{2}(sorted_index1);
    info0{3} = info0{3}(sorted_index0, :);
    info1{3} = info1{3}(sorted_index1, :);
    for i = 4:7
        info0{i} = info0{i}(sorted_index0);
        info1{i} = info1{i}(sorted_index1);
    end
end



% 在每两个相邻的一级分叉点间插入点
nearest_dot0 = cat(1, supply(1, :), nearest_dot0);
nearest_dot1 = cat(1, supply(2, :), nearest_dot1);
tmp_var = zeros(2*size(nearest_dot0, 1)-1, 2);
tmp_var(1:2:2*size(nearest_dot0, 1)-1, :) = nearest_dot0;
nearest_dot0 = tmp_var;
for i = 2:2:size(nearest_dot0, 1)-1
    nearest_dot0(i, :) = (nearest_dot0(i-1, :) + nearest_dot0(i+1, :) ) /
2;
end
tmp_var = zeros(2*size(nearest_dot1, 1)-1, 2);
tmp_var(1:2:2*size(nearest_dot1, 1)-1, :) = nearest_dot1;
nearest_dot1 = tmp_var;
for i = 2:2:size(nearest_dot1, 1)-1
    nearest_dot1(i, :) = (nearest_dot1(i-1, :) + nearest_dot1(i+1, :) ) /
2;
end


% 故障发生时决策对哪些负荷供电 0
possibility0    =    rot90(tril(ones(size(info0{2},    2)    )    )    );
decesion_mat0_size2 = 0;
for m = 1:size(info0{2}, 2) % 簇
    if size(info0{2}{m}, 1) > decesion_mat0_size2
        decesion_mat0_size2 = size(info0{2}{m}, 1);
    end
end   % 故障发生时决策对哪些负荷供电
decision_mat0    =    zeros(size(info0{2},    2),    decesion_mat0_size2,
size(info0{2}, 2));
candidate0 = cell(1, size(info0{2}, 2) ); % 故障发生时待供电的负荷的功率
for e = 1:size(info0{2}, 2)  % 对每种故障情况
```

```
    candidate0{e} = info0{7}{size(info0{7}, 2)}; % 待供电的
    for time = 2:e
        % 待供电的负荷
        candidate0{e} = cat(2, info0{7}{size(info0{2}, 2) - time + 1},
candidate0{e});
    end

    for cluster = 1:size(info0{2}, 2)
        for time = 1:cluster
            % 算出对应的故障概率
            if time == 1 && time == size(info0{2}, 2) - e  + 1
                possibility0(cluster, e) = possibility0(cluster, e) *  (1 -
0.998 * (1 - 0.002 * length0(time) ) * 0.995 );
            elseif time == 1
                possibility0(cluster, e) = possibility0(cluster, e) * 0.998
* (1 - 0.002 * length0(time) ) * 0.995;
            elseif time == size(info0{2}, 2) - e  + 1
                possibility0(cluster, e) = possibility0(cluster, e) *  (1 -
0.998 * (1 - 0.002 * length0(time) ) );
            else
                possibility0(cluster, e) = possibility0(cluster, e) * 0.998
* (1 - 0.002 * length0(time) );
            end
        end
    end

    [max_volumn0(e),    choices0{e}]    =    package_problem2(capacity(1),
candidate0{e}, candidate0{e});
    tmp_counter = 1;
    for m = size(info0{2}, 2)-e+1:size(info0{2}, 2)
        for n = 1:size(info0{2}{m}, 1)
            if sum(tmp_counter == choices0{e}) == 1
                decision_mat0(m, n, e) = 1;
            end
            tmp_counter = tmp_counter + 1;
        end
    end
end


% 故障发生时决策对哪些负荷供电 1
possibility1    =    rot90(tril(ones(size(info1{2},    2)    )    )    );
decesion_mat1_size2 = 0;
for m = 1:size(info1{2}, 2) % 簇
```

```
    if size(info1{2}{m}, 1) > decesion_mat1_size2
        decesion_mat1_size2 = size(info1{2}{m}, 1);
    end
end   % 故障发生时决策对哪些负荷供电
decision_mat1    =    zeros(size(info1{2},    2),    decesion_mat1_size2,
size(info1{2}, 2));
candidate1 = cell(1, size(info1{2}, 2) ); % 故障发生时待供电的负荷的功率
for e = 1:size(info1{2}, 2)  % 对每种故障情况（最近的故障有 e 种情况）

    candidate1{e} = info1{7}{size(info1{7}, 2)}; % 待供电的
    for time = 2:e
        % 待供电的负荷
        candidate1{e}  =  cat(2,  info1{7}{size(info1{2},  2)  -  time  +  1},
candidate1{e});
    end

    for cluster = 1:size(info1{2}, 2)
        for time = 1:cluster
            % 算出对应的故障概率
            if time == 1 && time == size(info1{2}, 2) - e  + 1
                possibility1(cluster, e) = possibility1(cluster, e) *  (1 -
0.998 * (1 - 0.002 * length1(time) ) * 0.995 );
            elseif time == 1
                possibility1(cluster, e) = possibility1(cluster, e) * 0.998
* (1 - 0.002 * length1(time) ) * 0.995;
            elseif time == size(info1{2}, 2) - e  + 1
                possibility1(cluster, e) = possibility1(cluster, e) *  (1 -
0.998 * (1 - 0.002 * length1(time) ) );
            else
                possibility1(cluster, e) = possibility1(cluster, e) * 0.998
* (1 - 0.002 * length1(time) );
            end
        end
    end

    [max_volumn1(e),    choices1{e}]    =    package_problem2(capacity(2),
candidate1{e}, candidate1{e});
    tmp_counter = 1;
    for m = size(info1{2}, 2)-e+1:size(info1{2}, 2)
        for n = 1:size(info1{2}{m}, 1)
            if sum(tmp_counter == choices1{e}) == 1
                decision_mat1(m, n, e) = 1;
            end
            tmp_counter = tmp_counter + 1;
```

```
        end
    end
end


sum_reliability = zeros(1, size(nearest_dot0, 1) * size(nearest_dot1, 1) );
min_reliability = zeros(1, size(nearest_dot0, 1) * size(nearest_dot1, 1) );
[nd0, si0, po0] = reliability_fun0(info0{1}, supply(1, :), info0{3} );
[nd1, si1, po1] = reliability_fun0(info1{1}, supply(2, :), info1{3} );
len1 = 99999 * ones(1, size(nearest_dot0, 1) * size(nearest_dot1, 1) );
% 电源 0 的第 i 个一级分叉点 连接到 电源 1 的第 j 个一级分叉点
for i = 2:size(nearest_dot0, 1)
    for j = 2:size(nearest_dot1, 1)
        contact_line_length        =        dist(nearest_dot0(i,        :),
nearest_dot1(j, :)' );
        main_counter = (i-1) * size(nearest_dot1, 1) + j;
        if 325.7 * contact_line_length  + 56.8 > money
            continue;
        end


        clear reliability0; counter = 1;
        for m = 1:size(info0{2}, 2) % 簇
            for n = 1:size(info0{2}{m}, 1)
                % 本树的主线可靠性和支线可靠性
                [~,  sub_line_re0,  main_line_re0]  =  reliability_fun00(0,
info0{2}{m}(n, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6}, nd0,
si0, po0);

                % 联络线在另一树的端点的可靠性
                if j == 1
                    contact_line_re0 = 0.995;
                elseif mod(j, 2) == 1
                    contact_line_re0         =         reliability_fun00(1,
nearest_dot1(j, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6}, nd1,
si1, po1);
                else
                    contact_line_re0         =         reliability_fun00(1,
nearest_dot1(j+1, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6},
nd1, si1, po1);
                    contact_line_re0 = contact_line_re0 / (1 - 0.002 * 0.5
* length1(j/2) );
                end
                if contact_line_re0 == -1
```

```
                    disp("ERROR"); % 报错
                end


                % 联络线在本树的端点 到 每个一级分叉点 的距离
                tmp1 = 2*m+1;
                tmp2 = i;
                tmp_reliability = 1;
                % line1
                tmp_len = 0;
                for o = min(tmp1, tmp2):max(tmp1, tmp2)-1
                    tmp_len = tmp_len + 0.5 * length0(ceil(o / 2) );
                    if mod(o, 2) == 0
                        tmp_reliability = tmp_reliability * 0.998; % 开关
                        tmp_reliability = tmp_reliability * (1 - 0.002 *
tmp_len);
                        tmp_len = 0;
                    elseif o == max(tmp1, tmp2)-1 && mod(o, 2) == 1
                        tmp_reliability = tmp_reliability * 0.998; % 开关
                        tmp_reliability = tmp_reliability * (1 - 0.002 *
tmp_len);
                        tmp_len = 0;
                    end
                end

                % 算上联络线本身的可靠性
                contact_line_re0 = contact_line_re0 * 0.998 * (1 - 0.002 *
contact_line_length);
                % 算上联络线到一级分叉点的可靠性
                contact_line_re0 = contact_line_re0 * tmp_reliability;


                % 负荷可靠性计算
                tmp_re = 0;
                for e = 1:size(info0{2}, 2)
                    tmp_contact_re = 1;
                    if i >= 2 * (size(info0{2}, 2) - e) + 1 % 故障发生在联络
线 1 之前，可靠性要计算联络线 1
                        tmp_contact_re = tmp_contact_re * (1 -
contact_line_re0);
                    end
                    tmp_re = tmp_re + decision_mat0(m, n, e) *
possibility0(m, e) * (1 - tmp_contact_re );
                end
```

```
                tmp_re = tmp_re + main_line_re0;
                reliability0(counter) = sub_line_re0 * tmp_re;
                counter = counter + 1;
            end
        end


        clear reliability1; counter = 1;
        for m = 1:size(info1{2}, 2)
            for n = 1:size(info1{2}{m}, 1)
                % 本树的主线可靠性和支线可靠性
                [~, sub_line_re1, main_line_re1] = reliability_fun00(0,
info1{2}{m}(n, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6}, nd1,
si1, po1);

                % 联络线在另一树的端点的可靠性
                if i == 1
                    contact_line_re1 = 0.995;
                elseif mod(i, 2) == 1
                    contact_line_re1          =          reliability_fun00(1,
nearest_dot0(i, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6}, nd0,
si0, po0);
                else
                    contact_line_re1          =          reliability_fun00(1,
nearest_dot0(i+1, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6},
nd0, si0, po0);
                    contact_line_re1 = contact_line_re1 / (1 - 0.002 * 0.5
* length0(i/2) );
                end
                if contact_line_re1 == -1
                    disp("ERROR"); % 报错
                end

                % 联络线在本树的端点 到 每个一级分叉点 的距离
                tmp1 = 2*m+1;
                tmp2 = j;
                tmp_reliability = 1;
                % line1
                tmp_len = 0;
                for o = min(tmp1, tmp2):max(tmp1, tmp2)-1
                    tmp_len = tmp_len + 0.5 * length1(ceil(o / 2) );
                    if mod(o, 2) == 0
                        tmp_reliability = tmp_reliability * 0.998; % 开关
                        tmp_reliability = tmp_reliability * (1 - 0.002 *
```

```
tmp_len);
                    tmp_len = 0;
                elseif o == max(tmp1, tmp2)-1 && mod(o, 2) == 1
                    tmp_reliability = tmp_reliability * 0.998; % 开关
                    tmp_reliability = tmp_reliability * (1 - 0.002 *
tmp_len);
                    tmp_len = 0;
                end
            end
            % 算上联络线本身的可靠性
            contact_line_re1 = contact_line_re1 * 0.998 * (1 - 0.002 *
contact_line_length);
            % 算上联络线到一级分叉点的可靠性
            contact_line_re1 = contact_line_re1 * tmp_reliability;

            % 负荷可靠性计算
            tmp_re = 0;
            for e = 1:size(info1{2}, 2)
                tmp_contact_re = 1;
                if j >= 2 * (size(info1{2}, 2) - e) + 1 % 故障发生在联络
线 1 之前，可靠性要计算联络线 1
                    tmp_contact_re = tmp_contact_re * (1 -
contact_line_re1);
                end
                tmp_re = tmp_re + decision_mat1(m, n, e) *
possibility1(m, e) * (1 - tmp_contact_re );
            end
            tmp_re = tmp_re + main_line_re1;
            reliability1(counter) = sub_line_re1 * tmp_re;
            counter = counter + 1;
        end
    end
    sum_reliability(main_counter) = sum(reliability0) +
sum(reliability1);
    min_reliability(main_counter) = min(min(reliability0),
min(reliability1) );
    len1(main_counter) = dist(nearest_dot0(i, :),
nearest_dot1(j, :)' );

  end
end

if sum(min_reliability) == 0
  val = -1; % 费用不够建设任何一条联络线
```

```
    cost = -1; h = -1;
    disp("费用不够建设任何一条联络线");
    return;
end

if target == 3 % 第 3 问最低可靠性最大的方案
    [val, max_index1] = max(min_reliability); % 最低可靠性最大
    avg = sum_reliability(max_index1) / 35;
    fprintf("一条联络线的情况下最大的最低可靠性是%f\n", val);
    fprintf("此方案用户平均可靠性是%f\n", avg);

    % 得到方案对应的 index
    col1 = mod(max_index1, size(nearest_dot1, 1) );
    if col1 == 0
        col1 = size(nearest_dot1, 1);
    end
    row1 = ceil(max_index1 / size(nearest_dot1, 1) );

    len1 = len1(max_index1);
    cost = 56.8 + 325.7 * len1; % 联络线花费

    hold on; % 画图
    line([nearest_dot0(row1,       1),       nearest_dot1(col1,       1)],
[nearest_dot0(row1,   2),   nearest_dot1(col1,   2)],   'Color',   color,
'LineWidth', 4);
    contact = [row1, col1];
    figure;
    h = topology_plot2(info0{3}, info0{4}, info0{5}, info1{3}, info1{4},
info1{5}, contact);

elseif target == 4 % 第 4 问费用最低的方案
    while true
        [min_len, min_index] = min(len1); % 联络线最短的方案
        if min_len == 99999
            % 如果连一条线怎么样都达不到可靠性的要求
            disp("一条联络线无法达到可靠性要求");
            disp("正在尝试连接两条联络线");
            [val,  min_reliability,  len1,  ~,  cost,  contact,  h,  avg]  =
reliability_fun_2(supply,  info0,  info1,  money,  request_min_reliability,
target, color, invoke_time+1);
            return;
        elseif min_reliability(min_index) < request_min_reliability % 如果
不满足最低可靠性的要求
            len1(min_index) = 99999; % 则舍去这个方案
```

```
        else
            cost = 56.8 + 325.7 * min_len; % 总花费
            val = min_reliability(min_index); % 本方案最低可靠性
            avg = sum_reliability(min_index) / 35;
            disp("一条联络线的情况下满足了可靠性要求");
            fprintf("最低费用为%f\n", cost);

            % 得到方案对应的 index
            col1 = mod(min_index, size(nearest_dot1, 1) );
            if col1 == 0
                col1 = size(nearest_dot1, 1);
            end
            row1 = ceil(min_index / size(nearest_dot1, 1) );
            contact = [row1, col1];

            hold on; % 画图
            line([nearest_dot0(row1,    1),    nearest_dot1(col1,    1)],
[nearest_dot0(row1,   2),   nearest_dot1(col1,   2)],   'Color',   color,
'LineWidth', 4);
            figure;
            h  =  topology_plot2(info0{3},  info0{4},  info0{5},  info1{3},
info1{4}, info1{5}, contact);
            return;
        end
    end
end

return;
```

程序二十四：可靠性计算程序——双供双联络线

reliability_fun_2.m

```
function [val, min_reliability, len1, len2, cost, contact, h, avg] =
reliability_fun_2(supply, info0, info1, money, request_min_reliability,
target, color, invoke_time)
% 计算双供网络，两条联络线情况下的可靠性


% 求电源 0 的各个一级分叉点
[~, ~, ~, ~, length0, nearest_dot0, sorted_index0] = reliability_fun(0,
info0{2}{1}(1, :), info0{1}, supply(1, :), info0{2}, info0{3}, info0{4},
info0{5}, info0{6});
```

```
% 求电源 1 的各个一级分叉点
[~, ~, ~, ~, length1, nearest_dot1, sorted_index1] = reliability_fun(0,
info1{2}{1}(1, :), info1{1}, supply(2, :), info1{2}, info1{3}, info1{4},
info1{5}, info1{6});


% 计算可供功率上限
capacity0 = 0; capacity1 = 0;
for k = 1:size(info0{7}, 2)
    capacity0 = capacity0 + sum(info0{7}{k});
end
for k = 1:size(info1{7}, 2)
    capacity1 = capacity1 + sum(info1{7}{k});
end
capacity0 = capacity0 * 1.1 * 0.5;
capacity1 = capacity1 * 1.1 * 0.5;
capacity = round([capacity0, capacity1]);


if invoke_time == 1
    % 换顺序
    info0{2} = info0{2}(sorted_index0);
    info1{2} = info1{2}(sorted_index1);
    info0{3} = info0{3}(sorted_index0, :);
    info1{3} = info1{3}(sorted_index1, :);
    for i = 4:7
        info0{i} = info0{i}(sorted_index0);
        info1{i} = info1{i}(sorted_index1);
    end
end



% 在每两个相邻的一级分叉点间插入点
nearest_dot0 = cat(1, supply(1, :), nearest_dot0);
nearest_dot1 = cat(1, supply(2, :), nearest_dot1);
tmp_var = zeros(2*size(nearest_dot0, 1)-1, 2);
tmp_var(1:2:2*size(nearest_dot0, 1)-1, :) = nearest_dot0;
nearest_dot0 = tmp_var;
for i = 2:2:size(nearest_dot0, 1)-1
    nearest_dot0(i, :) = (nearest_dot0(i-1, :) + nearest_dot0(i+1, :) ) /
2;
end
```

```
tmp_var = zeros(2*size(nearest_dot1, 1)-1, 2);
tmp_var(1:2:2*size(nearest_dot1, 1)-1, :) = nearest_dot1;
nearest_dot1 = tmp_var;
for i = 2:2:size(nearest_dot1, 1)-1
    nearest_dot1(i, :) = (nearest_dot1(i-1, :) + nearest_dot1(i+1, :) ) /
2;
end


% 故障发生时决策对哪些负荷供电 0
possibility0    =    rot90(tril(ones(size(info0{2},    2)    )    )    );
decesion_mat0_size2 = 0;
for m = 1:size(info0{2}, 2) % 簇
    if size(info0{2}{m}, 1) > decesion_mat0_size2
        decesion_mat0_size2 = size(info0{2}{m}, 1);
    end
end   % 故障发生时决策对哪些负荷供电
decision_mat0    =    zeros(size(info0{2},    2),    decesion_mat0_size2,
size(info0{2}, 2));
candidate0 = cell(1, size(info0{2}, 2) ); % 故障发生时待供电的负荷的功率
for e = 1:size(info0{2}, 2)  % 对每种故障情况

    candidate0{e} = info0{7}{size(info0{7}, 2)}; % 待供电的
    for time = 2:e
        % 待供电的负荷
        candidate0{e}  =  cat(2,  info0{7}{size(info0{2},  2)  -  time  +  1},
candidate0{e});
    end

    for cluster = 1:size(info0{2}, 2)
        for time = 1:cluster
            % 算出对应的故障概率
            if time == 1 && time == size(info0{2}, 2) - e  + 1
                possibility0(cluster, e) = possibility0(cluster, e) *  (1 -
0.998 * (1 - 0.002 * length0(time) ) * 0.995 );
            elseif time == 1
                possibility0(cluster, e) = possibility0(cluster, e) * 0.998
* (1 - 0.002 * length0(time) ) * 0.995;
            elseif time == size(info0{2}, 2) - e  + 1
                possibility0(cluster, e) = possibility0(cluster, e) *  (1 -
0.998 * (1 - 0.002 * length0(time) ) );
            else
                possibility0(cluster, e) = possibility0(cluster, e) * 0.998
* (1 - 0.002 * length0(time) );
```

```
            end
        end
    end

    [max_volumn0(e),    choices0{e}]    =    package_problem2(capacity(1),
candidate0{e}, candidate0{e});
    tmp_counter = 1;
    for m = size(info0{2}, 2)-e+1:size(info0{2}, 2)
        for n = 1:size(info0{2}{m}, 1)
            if sum(tmp_counter == choices0{e}) == 1
                decision_mat0(m, n, e) = 1;
            end
            tmp_counter = tmp_counter + 1;
        end
    end
end



% 故障发生时决策对哪些负荷供电 1
possibility1    =    rot90(tril(ones(size(info1{2},    2)    )    )    );
decesion_mat1_size2 = 0;
for m = 1:size(info1{2}, 2) % 簇
    if size(info1{2}{m}, 1) > decesion_mat1_size2
        decesion_mat1_size2 = size(info1{2}{m}, 1);
    end
end   % 故障发生时决策对哪些负荷供电
decision_mat1    =    zeros(size(info1{2},    2),    decesion_mat1_size2,
size(info1{2}, 2));
candidate1 = cell(1, size(info1{2}, 2) ); % 故障发生时待供电的负荷的功率
for e = 1:size(info1{2}, 2)  % 对每种故障情况（最近的故障有 e 种情况）

    candidate1{e} = info1{7}{size(info1{7}, 2)}; % 待供电的
    for time = 2:e
        % 待供电的负荷
        candidate1{e}  =  cat(2,  info1{7}{size(info1{2},  2)  -  time  +  1},
candidate1{e});
    end

    for cluster = 1:size(info1{2}, 2)
        for time = 1:cluster
            % 算出对应的故障概率
            if time == 1 && time == size(info1{2}, 2) - e  + 1
                possibility1(cluster, e) = possibility1(cluster, e) *  (1 -
```

```
0.998 * (1 - 0.002 * length1(time) ) * 0.995 );
            elseif time == 1
                possibility1(cluster, e) = possibility1(cluster, e) * 0.998
* (1 - 0.002 * length1(time) ) * 0.995;
            elseif time == size(info1{2}, 2) - e  + 1
                possibility1(cluster, e) = possibility1(cluster, e) *  (1 -
0.998 * (1 - 0.002 * length1(time) ) );
            else
                possibility1(cluster, e) = possibility1(cluster, e) * 0.998
* (1 - 0.002 * length1(time) );
            end
        end
    end

    [max_volumn1(e),    choices1{e}]    =    package_problem2(capacity(2),
candidate1{e}, candidate1{e});
    tmp_counter = 1;
    for m = size(info1{2}, 2)-e+1:size(info1{2}, 2)
        for n = 1:size(info1{2}{m}, 1)
            if sum(tmp_counter == choices1{e}) == 1
                decision_mat1(m, n, e) = 1;
            end
            tmp_counter = tmp_counter + 1;
        end
    end
end



sum_reliability = zeros(1, size(nearest_dot0, 1) * size(nearest_dot1, 1) *
size(nearest_dot0, 1) * size(nearest_dot1, 1));
min_reliability = zeros(1, size(nearest_dot0, 1) * size(nearest_dot1, 1) *
size(nearest_dot0, 1) * size(nearest_dot1, 1));
[nd0, si0, po0] = reliability_fun0(info0{1}, supply(1, :), info0{3} );
[nd1, si1, po1] = reliability_fun0(info1{1}, supply(2, :), info1{3} );
len1 = 10000 * ones(1, size(nearest_dot0, 1) * size(nearest_dot1, 1) *
size(nearest_dot0, 1) * size(nearest_dot1, 1) );
len2 = 10000 * ones(1, size(nearest_dot0, 1) * size(nearest_dot1, 1) *
size(nearest_dot0, 1) * size(nearest_dot1, 1) );
% 电源 0 的第 i 个一级分叉点 连接到 电源 1 的第 j 个一级分叉点
for i = 2:size(nearest_dot0, 1)
    for j = 2:size(nearest_dot1, 1)
        for i2 = 2:size(nearest_dot0, 1)
            for j2 = 2:size(nearest_dot1, 1)
```

```
                contact_line_length     =     dist(nearest_dot0(i,     :),
nearest_dot1(j, :)' );
                contact_line2_length    =     dist(nearest_dot0(i2,    :),
nearest_dot1(j2, :)' );
                main_counter   =   (i-1)   *     size(nearest_dot1,   1)   *
size(nearest_dot0, 1) * size(nearest_dot1, 1)...
                    + (j-1) * size(nearest_dot0, 1) * size(nearest_dot1, 1)
+ (i2 - 1) * size(nearest_dot1, 1) + j2;
                if 325.7 * (contact_line_length + contact_line2_length) + 2
* 56.8 > money
                    continue;
                end


                clear reliability0; counter = 1;
                for m = 1:size(info0{2}, 2) % 簇
                    for n = 1:size(info0{2}{m}, 1)
                        % 本树的主线可靠性和支线可靠性
                        [~,       sub_line_re0,       main_line_re0]       =
reliability_fun00(0,  info0{2}{m}(n,  :),  info0{2},  info0{3},  info0{4},
info0{5}, info0{6}, nd0, si0, po0);

                        % 联络线在另一树的端点的可靠性
                        if j == 1
                            contact_line_re0 = 0.995;
                        elseif mod(j, 2) == 1
                            contact_line_re0     =     reliability_fun00(1,
nearest_dot1(j, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6}, nd1,
si1, po1);
                        else
                            contact_line_re0     =     reliability_fun00(1,
nearest_dot1(j+1,  :),  info1{2},  info1{3},  info1{4},  info1{5},  info1{6},
nd1, si1, po1);
                            contact_line_re0 = contact_line_re0 / (1 - 0.002
* 0.5 * length1(j/2) );
                        end
                        if contact_line_re0 == -1
                            disp("ERROR"); % 报错
                        end

                        if j2 == 1
                            contact_line2_re0 = 0.995;
                        elseif mod(j2, 2) == 1
                            contact_line2_re0     =     reliability_fun00(1,
```

```
nearest_dot1(j2, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6}, nd1,
si1, po1);
                    else
                        contact_line2_re0     =     reliability_fun00(1,
nearest_dot1(j2+1, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6},
nd1, si1, po1);
                        contact_line2_re0  =  contact_line2_re0  /  (1  -
0.002 * 0.5 * length1(j2/2) );
                    end
                    if contact_line2_re0 == -1
                        disp("ERROR"); % 报错
                    end

                    % 联络线在本树的端点 到 每个一级分叉点 的距离
                    tmp1 = 2*m+1;
                    tmp2 = i;
                    tmp_reliability = 1;
                    % line1
                    tmp_len = 0;
                    for o = min(tmp1, tmp2):max(tmp1, tmp2)-1
                        tmp_len = tmp_len + 0.5 * length0(ceil(o / 2) );
                        if mod(o, 2) == 0
                            tmp_reliability = tmp_reliability * 0.998; %
开关
                            tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                            tmp_len = 0;
                        elseif o == max(tmp1, tmp2)-1 && mod(o, 2) == 1
                            tmp_reliability = tmp_reliability * 0.998; %
开关
                            tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                            tmp_len = 0;
                        end
                    end

                    % 算上联络线本身的可靠性
                    contact_line_re0 = contact_line_re0 * 0.998 * (1 -
0.002 * contact_line_length);
                    % 算上联络线到一级分叉点的可靠性
                    contact_line_re0      =      contact_line_re0      *
tmp_reliability;
```

```
                        % 第二联络线在本树的端点 到 每个一级分叉点 的距离
                        tmp1 = 2*m+1;
                        tmp2 = i2;
                        tmp_reliability = 1;
                        % line2
                        tmp_len = 0;
                        for o = min(tmp1, tmp2):max(tmp1, tmp2)-1
                            tmp_len = tmp_len + 0.5 * length0(ceil(o / 2) );
                            if mod(o, 2) == 0
                                tmp_reliability = tmp_reliability * 0.998; %
开关
                                tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                                tmp_len = 0;
                            elseif o == max(tmp1, tmp2)-1 && mod(o, 2) == 1
                                tmp_reliability = tmp_reliability * 0.998; %
开关
                                tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                                tmp_len = 0;
                            end
                        end

                        % 算上联络线本身的可靠性
                        contact_line2_re0 = contact_line2_re0 * 0.998 * (1 -
0.002 * contact_line2_length);
                        % 算上联络线到一级分叉点的可靠性
                        contact_line2_re0     =     contact_line2_re0     *
tmp_reliability;



                        % 负荷可靠性计算
                        tmp_re = 0;
                        for e = 1:size(info0{2}, 2)
                            tmp_contact_re = 1;
                            if i >= 2 * (size(info0{2}, 2) - e) + 1 % 故障发
生在联络线 1 之前，可靠性要计算联络线 1
                                tmp_contact_re  =  tmp_contact_re  *  (1  -
contact_line_re0);
                            end
                            if i2 >= 2 * (size(info0{2}, 2) - e) + 1 % 故障发
生在联络线 2 之前，可靠性要计算联络线 2
                                tmp_contact_re  =  tmp_contact_re  *  (1  -
contact_line2_re0);
```

```
                        end
                        tmp_re = tmp_re + decision_mat0(m, n, e) *
possibility0(m, e) * (1 - tmp_contact_re );
                    end
                    tmp_re = tmp_re + main_line_re0;
                    reliability0(counter) = sub_line_re0 * tmp_re;
                    counter = counter + 1;
                end
            end


            clear reliability1; counter = 1;
            for m = 1:size(info1{2}, 2)
                for n = 1:size(info1{2}{m}, 1)
                    % 本树的主线可靠性和支线可靠性
                    [~,       sub_line_re1,       main_line_re1]       =
reliability_fun00(0, info1{2}{m}(n, :), info1{2}, info1{3}, info1{4},
info1{5}, info1{6}, nd1, si1, po1);

                    % 联络线在另一树的端点的可靠性
                    if i == 1
                        contact_line_re1 = 0.995;
                    elseif mod(i, 2) == 1
                        contact_line_re1     =     reliability_fun00(1,
nearest_dot0(i, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6}, nd0,
si0, po0);
                    else
                        contact_line_re1     =     reliability_fun00(1,
nearest_dot0(i+1, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6},
nd0, si0, po0);
                        contact_line_re1 = contact_line_re1 / (1 - 0.002
* 0.5 * length0(i/2) );
                    end
                    if contact_line_re1 == -1
                        disp("ERROR"); % 报错
                    end

                    if i2 == 1
                        contact_line2_re1 = 0.995;
                    elseif mod(i2, 2) == 1
                        contact_line2_re1     =     reliability_fun00(1,
nearest_dot0(i2, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6}, nd0,
si0, po0);
                    else
```

```
                        contact_line2_re1     =     reliability_fun00(1,
nearest_dot0(i2+1, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6},
nd0, si0, po0);
                        contact_line2_re1  =  contact_line2_re1  /  (1  -
0.002 * 0.5 * length0(i2/2) );
                    end
                    if contact_line2_re1 == -1
                        disp("ERROR"); % 报错
                    end


                    % 联络线在本树的端点 到 每个一级分叉点 的距离
                    tmp1 = 2*m+1;
                    tmp2 = j;
                    tmp_reliability = 1;
                    % line1
                    tmp_len = 0;
                    for o = min(tmp1, tmp2):max(tmp1, tmp2)-1
                        tmp_len = tmp_len + 0.5 * length1(ceil(o / 2) );
                        if mod(o, 2) == 0
                            tmp_reliability = tmp_reliability * 0.998; %
开关
                            tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                            tmp_len = 0;
                        elseif o == max(tmp1, tmp2)-1 && mod(o, 2) == 1
                            tmp_reliability = tmp_reliability * 0.998; %
开关
                            tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                            tmp_len = 0;
                        end
                    end
                    % 算上联络线本身的可靠性
                    contact_line_re1 = contact_line_re1 * 0.998 * (1 -
0.002 * contact_line_length);
                    % 算上联络线到一级分叉点的可靠性
                    contact_line_re1      =      contact_line_re1      *
tmp_reliability;


                    % 联络线在本树的端点 到 每个一级分叉点 的距离
                    tmp1 = 2*m+1;
                    tmp2 = j2;
```

```
                        tmp_reliability = 1;
                        % line2
                        tmp_len = 0;
                        for o = min(tmp1, tmp2):max(tmp1, tmp2)-1
                            tmp_len = tmp_len + 0.5 * length1(ceil(o / 2) );
                            if mod(o, 2) == 0
                                tmp_reliability = tmp_reliability * 0.998; %
开关
                                tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                                tmp_len = 0;
                            elseif o == max(tmp1, tmp2)-1 && mod(o, 2) == 1
                                tmp_reliability = tmp_reliability * 0.998; %
开关
                                tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                                tmp_len = 0;
                            end
                        end
                        % 算上联络线本身的可靠性
                        contact_line2_re1 = contact_line2_re1 * 0.998 * (1 -
0.002 * contact_line2_length);
                        % 算上联络线到一级分叉点的可靠性
                        contact_line2_re1     =     contact_line2_re1     *
tmp_reliability;



                        % 负荷可靠性计算
                        tmp_re = 0;
                        for e = 1:size(info1{2}, 2)
                            tmp_contact_re = 1;
                            if j >= 2 * (size(info1{2}, 2) - e) + 1 % 故障发
生在联络线 1 之前，可靠性要计算联络线 1
                                tmp_contact_re  =  tmp_contact_re  *  (1  -
contact_line_re1);
                            end
                            if j2 >= 2 * (size(info1{2}, 2) - e) + 1 % 故障发
生在联络线 2 之前，可靠性要计算联络线 2
                                tmp_contact_re  =  tmp_contact_re  *  (1  -
contact_line2_re1);
                            end
                            tmp_re  =  tmp_re  +  decision_mat1(m,  n,  e)  *
possibility1(m, e) * (1 - tmp_contact_re );
                        end
```

```
                    tmp_re = tmp_re + main_line_re1;
                    reliability1(counter) = sub_line_re1 * tmp_re;
                    counter = counter + 1;
                end
            end
            sum_reliability(main_counter)    =    sum(reliability0)    +
sum(reliability1);
            min_reliability(main_counter)    =    min(min(reliability0),
min(reliability1) );
            len1(main_counter)      =      dist(nearest_dot0(i,      :),
nearest_dot1(j, :)' );
            len2(main_counter)      =      dist(nearest_dot0(i2,      :),
nearest_dot1(j2, :)' );
        end
    end
end



if target == 3 % 第 3 问最低可靠性最大的方案
    if sum(min_reliability) == 0
        clear len1; clear len2;
        disp("费用不够建设两条联络线");
        % 只连一条联络线
        disp("正在尝试只连一条联络线");
        [val,   min_reliability,   len1,   cost,   contact,   h,   avg]   =
reliability_fun_1(supply, info0, info1, money, request_min_reliability,
target, color, invoke_time+1);
        len2 = -1;
        return;
    end
    while true
        [val, max_index] = max(min_reliability); % 最低可靠性最大
        avg = sum_reliability(max_index) / 35;

        max_index1   =   ceil(max_index   /   (size(nearest_dot1,   1)   *
size(nearest_dot0, 1) ) );
        max_index2    =    mod(max_index,    size(nearest_dot1,    1)    *
size(nearest_dot0, 1) );
        % 得到方案对应的 index
        col1 = mod(max_index1, size(nearest_dot1, 1) );
        if col1 == 0
            col1 = size(nearest_dot1, 1);
```

```
        end
        row1 = ceil(max_index1 / size(nearest_dot1, 1) );

        col2 = mod(max_index2, size(nearest_dot1, 1) );
        if col2 == 0
            col2 = size(nearest_dot1, 1);
        end
        row2 = ceil(max_index2 / size(nearest_dot1, 1) );
        if row1 ~= row2 || col1 ~= col2
            break
        else
            min_reliability(max_index) = 0;
        end
    end
    contact = [row1, col1; row2, col2];
    fprintf("两条联络线的情况下最大的最低可靠性是%f\n", val);
    fprintf("此方案用户平均可靠性是%f\n", avg);


    len2 = len2(max_index); % 第二条联络线长度
    len1 = len1(max_index);
    cost = 2 * 56.8 + 325.7 * (len1 + len2); % 联络线花费


    hold on; % 画图
    line([nearest_dot0(row1,       1),       nearest_dot1(col1,       1)],
[nearest_dot0(row1,   2),   nearest_dot1(col1,   2)],   'Color',   color,
'LineWidth', 4);
    hold on;
    line([nearest_dot0(row2,       1),       nearest_dot1(col2,       1)],
[nearest_dot0(row2,   2),   nearest_dot1(col2,   2)],   'Color',   color,
'LineWidth', 4);
    figure;
    h = topology_plot2(info0{3}, info0{4}, info0{5}, info1{3}, info1{4},
info1{5}, contact);


elseif target == 4 % 第 4 问费用最低的方案
    len = len1 + len2;
    while true
        [min_len, min_index] = min(len); % 第二条联络线最短的方案
        if min_len == 20000
            % 如果连两条线怎么样都达不到可靠性的要求
            disp("两条联络线无法达到可靠性要求");
```

```
            disp("正在尝试连接三条联络线");
            [val, min_reliability, len1, len2, ~, cost, contact, h, avg] =
reliability_fun_3(supply, info0, info1, money, request_min_reliability,
target, color, invoke_time+1);
            return;
        elseif min_reliability(min_index) < request_min_reliability % 如果
不满足最低可靠性的要求
            len(min_index) = 20000; % 则舍去这个方案
        else


            % 得到方案对应的 index
            min_index1 = ceil(min_index / (size(nearest_dot1, 1) *
size(nearest_dot0, 1) ) );
            min_index2 = mod(min_index, size(nearest_dot1, 1) *
size(nearest_dot0, 1) );
            col1 = mod(min_index1, size(nearest_dot1, 1) );
            if col1 == 0
                col1 = size(nearest_dot1, 1);
            end
            row1 = ceil(min_index1 / size(nearest_dot1, 1) );
            col2 = mod(min_index2, size(nearest_dot1, 1) );
            if col2 == 0
                col2 = size(nearest_dot1, 1);
            end
            row2 = ceil(min_index2 / size(nearest_dot1, 1) );

            if row1 == row2 && col1 == col2
                len(min_index) = 20000;
                continue;
            end

            cost = 2 * 56.8 + 325.7 * min_len; % 总花费
            val = min_reliability(min_index); % 本方案最低可靠性
            avg = sum_reliability(min_index) / 35;
            contact = [row1, col1; row2, col2];
            disp("两条联络线的情况下满足了可靠性要求");
            fprintf("最低费用为%f\n", cost);

            hold on; % 画图
            line([nearest_dot0(row1, 1), nearest_dot1(col1, 1)],
[nearest_dot0(row1, 2), nearest_dot1(col1, 2)], 'Color', color,
'LineWidth', 4);
            hold on; % 画图
```

```
            line([nearest_dot0(row2,    1),    nearest_dot1(col2,    1)],
[nearest_dot0(row2,   2),   nearest_dot1(col2,   2)],   'Color',   color,
'LineWidth', 4);
            figure;
            h  =  topology_plot2(info0{3},  info0{4},  info0{5},  info1{3},
info1{4}, info1{5}, contact);
            return;
        end
    end
end

return;
```

程序二十五：可靠性计算程序——双供三联络线

reliability_fun_3.m

```
function [val, min_reliability, len1, len2, len3, cost, contact, h, avg] =
reliability_fun_3(supply,  info0,  info1,  money,  request_min_reliability,
target, color, invoke_time)
% 计算双供网络，两条联络线情况下的可靠性


% 求电源 0 的各个一级分叉点
[~, ~, ~, ~, length0, nearest_dot0, sorted_index0] = reliability_fun(0,
info0{2}{1}(1, :), info0{1}, supply(1, :), info0{2}, info0{3}, info0{4},
info0{5}, info0{6});

% 求电源 1 的各个一级分叉点
[~, ~, ~, ~, length1, nearest_dot1, sorted_index1] = reliability_fun(0,
info1{2}{1}(1, :), info1{1}, supply(2, :), info1{2}, info1{3}, info1{4},
info1{5}, info1{6});


% 计算可供功率上限
capacity0 = 0; capacity1 = 0;
for k = 1:size(info0{7}, 2)
    capacity0 = capacity0 + sum(info0{7}{k});
end
for k = 1:size(info1{7}, 2)
    capacity1 = capacity1 + sum(info1{7}{k});
end
capacity0 = capacity0 * 1.1 * 0.5;
```

```
capacity1 = capacity1 * 1.1 * 0.5;
capacity = round([capacity0, capacity1]);


if invoke_time == 1
    % 换顺序
    info0{2} = info0{2}(sorted_index0);
    info1{2} = info1{2}(sorted_index1);
    info0{3} = info0{3}(sorted_index0, :);
    info1{3} = info1{3}(sorted_index1, :);
    for i = 4:7
        info0{i} = info0{i}(sorted_index0);
        info1{i} = info1{i}(sorted_index1);
    end
end



% 在每两个相邻的一级分叉点间插入点
nearest_dot0 = cat(1, supply(1, :), nearest_dot0);
nearest_dot1 = cat(1, supply(2, :), nearest_dot1);
tmp_var = zeros(2*size(nearest_dot0, 1)-1, 2);
tmp_var(1:2:2*size(nearest_dot0, 1)-1, :) = nearest_dot0;
nearest_dot0 = tmp_var;
for i = 2:2:size(nearest_dot0, 1)-1
    nearest_dot0(i, :) = (nearest_dot0(i-1, :) + nearest_dot0(i+1, :) ) /
2;
end
tmp_var = zeros(2*size(nearest_dot1, 1)-1, 2);
tmp_var(1:2:2*size(nearest_dot1, 1)-1, :) = nearest_dot1;
nearest_dot1 = tmp_var;
for i = 2:2:size(nearest_dot1, 1)-1
    nearest_dot1(i, :) = (nearest_dot1(i-1, :) + nearest_dot1(i+1, :) ) /
2;
end


% 两个离电源最远的一级分叉点间的长度，如果符合要求，则连接
len3     =     dist(nearest_dot0(size(nearest_dot0,     1),     :     ),
nearest_dot1(size(nearest_dot1, 1), : )' );
contact_line_3_re0  =  reliability_fun(1,  nearest_dot1(size(nearest_dot1,
1), : ), info1{1}, supply(2, :), info1{2}, info1{3}, info1{4}, info1{5},
info1{6});
contact_line_3_re1  =  reliability_fun(1,  nearest_dot0(size(nearest_dot0,
```

```
1), : ), info0{1}, supply(1, :), info0{2}, info0{3}, info0{4}, info0{5},
info0{6});
contact_line_3_re0 = contact_line_3_re0 * (1 - 0.002 * len3) * 0.998; % 联
络线本身的可靠性
contact_line_3_re1 = contact_line_3_re1 * (1 - 0.002 * len3) * 0.998; % 联
络线本身的可靠性


% 故障发生时决策对哪些负荷供电 0
possibility0    =    rot90(tril(ones(size(info0{2},    2)    )    )    );
decesion_mat0_size2 = 0;
for m = 1:size(info0{2}, 2) % 簇
    if size(info0{2}{m}, 1) > decesion_mat0_size2
        decesion_mat0_size2 = size(info0{2}{m}, 1);
    end
end   % 故障发生时决策对哪些负荷供电
decision_mat0    =    zeros(size(info0{2},    2),    decesion_mat0_size2,
size(info0{2}, 2));
candidate0 = cell(1, size(info0{2}, 2) ); % 故障发生时待供电的负荷的功率
for e = 1:size(info0{2}, 2)  % 对每种故障情况

    candidate0{e} = info0{7}{size(info0{7}, 2)}; % 待供电的
    for time = 2:e
        % 待供电的负荷
        candidate0{e} = cat(2, info0{7}{size(info0{2}, 2) - time + 1},
candidate0{e});
    end

    for cluster = 1:size(info0{2}, 2)
        for time = 1:cluster
            % 算出对应的故障概率
            if time == 1 && time == size(info0{2}, 2) - e  + 1
                possibility0(cluster, e) = possibility0(cluster, e) *  (1 -
0.998 * (1 - 0.002 * length0(time) ) * 0.995 );
            elseif time == 1
                possibility0(cluster, e) = possibility0(cluster, e) * 0.998
* (1 - 0.002 * length0(time) ) * 0.995;
            elseif time == size(info0{2}, 2) - e  + 1
                possibility0(cluster, e) = possibility0(cluster, e) *  (1 -
0.998 * (1 - 0.002 * length0(time) ) );
            else
                possibility0(cluster, e) = possibility0(cluster, e) * 0.998
* (1 - 0.002 * length0(time) );
            end
```

```
        end
    end

    [max_volumn0(e),    choices0{e}]    =    package_problem2(capacity(1),
candidate0{e}, candidate0{e});
    tmp_counter = 1;
    for m = size(info0{2}, 2)-e+1:size(info0{2}, 2)
        for n = 1:size(info0{2}{m}, 1)
            if sum(tmp_counter == choices0{e}) == 1
                decision_mat0(m, n, e) = 1;
            end
            tmp_counter = tmp_counter + 1;
        end
    end
end



% 故障发生时决策对哪些负荷供电 1
possibility1    =    rot90(tril(ones(size(info1{2},    2)    )    )    );
decesion_mat1_size2 = 0;
for m = 1:size(info1{2}, 2) % 簇
    if size(info1{2}{m}, 1) > decesion_mat1_size2
        decesion_mat1_size2 = size(info1{2}{m}, 1);
    end
end   % 故障发生时决策对哪些负荷供电
decision_mat1    =    zeros(size(info1{2},    2),    decesion_mat1_size2,
size(info1{2}, 2));
candidate1 = cell(1, size(info1{2}, 2) ); % 故障发生时待供电的负荷的功率
for e = 1:size(info1{2}, 2)  % 对每种故障情况（最近的故障有 e 种情况）

    candidate1{e} = info1{7}{size(info1{7}, 2)}; % 待供电的
    for time = 2:e
        % 待供电的负荷
        candidate1{e}  =  cat(2,  info1{7}{size(info1{2},  2)  -  time  +  1},
candidate1{e});
    end

    for cluster = 1:size(info1{2}, 2)
        for time = 1:cluster
            % 算出对应的故障概率
            if time == 1 && time == size(info1{2}, 2) - e  + 1
                possibility1(cluster, e) = possibility1(cluster, e) *  (1 -
0.998 * (1 - 0.002 * length1(time) ) * 0.995 );
```

```
            elseif time == 1
                possibility1(cluster, e) = possibility1(cluster, e) * 0.998
* (1 - 0.002 * length1(time) ) * 0.995;
            elseif time == size(info1{2}, 2) - e  + 1
                possibility1(cluster, e) = possibility1(cluster, e) *  (1 -
0.998 * (1 - 0.002 * length1(time) ) );
            else
                possibility1(cluster, e) = possibility1(cluster, e) * 0.998
* (1 - 0.002 * length1(time) );
            end
        end
    end

    [max_volumn1(e),    choices1{e}]    =    package_problem2(capacity(2),
candidate1{e}, candidate1{e});
    tmp_counter = 1;
    for m = size(info1{2}, 2)-e+1:size(info1{2}, 2)
        for n = 1:size(info1{2}{m}, 1)
            if sum(tmp_counter == choices1{e}) == 1
                decision_mat1(m, n, e) = 1;
            end
            tmp_counter = tmp_counter + 1;
        end
    end
end



sum_reliability = zeros(1, size(nearest_dot0, 1) * size(nearest_dot1, 1) *
size(nearest_dot0, 1) * size(nearest_dot1, 1));
min_reliability = zeros(1, size(nearest_dot0, 1) * size(nearest_dot1, 1) *
size(nearest_dot0, 1) * size(nearest_dot1, 1));
[nd0, si0, po0] = reliability_fun0(info0{1}, supply(1, :), info0{3} );
[nd1, si1, po1] = reliability_fun0(info1{1}, supply(2, :), info1{3} );
len1 = 10000 * ones(1, size(nearest_dot0, 1) * size(nearest_dot1, 1) *
size(nearest_dot0, 1) * size(nearest_dot1, 1) );
len2 = 10000 * ones(1, size(nearest_dot0, 1) * size(nearest_dot1, 1) *
size(nearest_dot0, 1) * size(nearest_dot1, 1) );
% 电源 0 的第 i 个一级分叉点 连接到 电源 1 的第 j 个一级分叉点
for i = 2:size(nearest_dot0, 1)
    for j = 2:size(nearest_dot1, 1)
        for i2 = 2:size(nearest_dot0, 1)
            for j2 = 2:size(nearest_dot1, 1)
                contact_line_length     =     dist(nearest_dot0(i,     :),
```

```
nearest_dot1(j, :)' );
                contact_line2_length    =    dist(nearest_dot0(i2,    :),
nearest_dot1(j2, :)' );
                main_counter   =   (i-1)   *     size(nearest_dot1,   1)   *
size(nearest_dot0, 1) * size(nearest_dot1, 1)...
                    + (j-1) * size(nearest_dot0, 1) * size(nearest_dot1, 1)
+ (i2 - 1) * size(nearest_dot1, 1) + j2;
                if 325.7 * (contact_line_length + contact_line2_length +
len3) + 3 * 56.8 > money
                    continue;
                end


                clear reliability0; counter = 1;
                for m = 1:size(info0{2}, 2) % 簇
                    for n = 1:size(info0{2}{m}, 1)
                        % 本树的主线可靠性和支线可靠性
                        [~,       sub_line_re0,       main_line_re0]       =
reliability_fun00(0,  info0{2}{m}(n,  :),  info0{2},  info0{3},  info0{4},
info0{5}, info0{6}, nd0, si0, po0);

                        % 联络线在另一树的端点的可靠性
                        if j == 1
                            contact_line_re0 = 0.995;
                        elseif mod(j, 2) == 1
                            contact_line_re0     =     reliability_fun00(1,
nearest_dot1(j, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6}, nd1,
si1, po1);
                        else
                            contact_line_re0     =     reliability_fun00(1,
nearest_dot1(j+1, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6},
nd1, si1, po1);
                            contact_line_re0 = contact_line_re0 / (1 - 0.002
* 0.5 * length1(j/2) );
                        end
                        if contact_line_re0 == -1
                            disp("ERROR"); % 报错
                        end

                        if j2 == 1
                            contact_line2_re0 = 0.995;
                        elseif mod(j2, 2) == 1
                            contact_line2_re0     =     reliability_fun00(1,
nearest_dot1(j2, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6}, nd1,
```

```
si1, po1);
                        else
                            contact_line2_re0     =     reliability_fun00(1,
nearest_dot1(j2+1, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6},
nd1, si1, po1);
                            contact_line2_re0  =  contact_line2_re0  /  (1  -
0.002 * 0.5 * length1(j2/2) );
                        end
                        if contact_line2_re0 == -1
                            disp("ERROR"); % 报错
                        end

                        % 联络线在本树的端点 到 每个一级分叉点 的距离
                        tmp1 = 2*m+1;
                        tmp2 = i;
                        tmp_reliability = 1;
                        % line1
                        tmp_len = 0;
                        for o = min(tmp1, tmp2):max(tmp1, tmp2)-1
                            tmp_len = tmp_len + 0.5 * length0(ceil(o / 2) );
                            if mod(o, 2) == 0
                                tmp_reliability = tmp_reliability * 0.998; %
开关
                                tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                                tmp_len = 0;
                            elseif o == max(tmp1, tmp2)-1 && mod(o, 2) == 1
                                tmp_reliability = tmp_reliability * 0.998; %
开关
                                tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                                tmp_len = 0;
                            end
                        end

                        % 算上联络线本身的可靠性
                        contact_line_re0 = contact_line_re0 * 0.998 * (1 -
0.002 * contact_line_length);
                        % 算上联络线到一级分叉点的可靠性
                        contact_line_re0      =      contact_line_re0      *
tmp_reliability;



                        % 第二联络线在本树的端点 到 每个一级分叉点 的距离
```

```
                        tmp1 = 2*m+1;
                        tmp2 = i2;
                        tmp_reliability = 1;
                        % line2
                        tmp_len = 0;
                        for o = min(tmp1, tmp2):max(tmp1, tmp2)-1
                            tmp_len = tmp_len + 0.5 * length0(ceil(o / 2) );
                            if mod(o, 2) == 0
                                tmp_reliability = tmp_reliability * 0.998; %
开关
                                tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                                tmp_len = 0;
                            elseif o == max(tmp1, tmp2)-1 && mod(o, 2) == 1
                                tmp_reliability = tmp_reliability * 0.998; %
开关
                                tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                                tmp_len = 0;
                            end
                        end

                        % 算上联络线本身的可靠性
                        contact_line2_re0 = contact_line2_re0 * 0.998 * (1 -
0.002 * contact_line2_length);
                        % 算上联络线到一级分叉点的可靠性
                        contact_line2_re0     =     contact_line2_re0     *
tmp_reliability;

                        % line3
                        tmp_len = 0;
                        contact_line3_re0 = contact_line_3_re0;
                        for o = tmp1 : size(nearest_dot0, 1)-1
                            tmp_len = tmp_len + 0.5 * length0(ceil(o / 2) );
                            if mod(o, 2) == 0
                                contact_line3_re0   =   contact_line3_re0   *
0.998; % 开关
                                contact_line3_re0 = contact_line3_re0 * (1 -
0.002 * tmp_len);
                                tmp_len = 0;
                            end
                        end
```

```
                    % 负荷可靠性计算
                    tmp_re = 0;
                    for e = 1:size(info0{2}, 2)
                        tmp_contact_re = 1 - contact_line3_re0;
                        if i >= 2 * (size(info0{2}, 2) - e) + 1 % 故障发生在联络线 1 之前，可靠性要计算联络线 1
                            tmp_contact_re = tmp_contact_re * (1 - contact_line_re0);
                        end
                        if i2 >= 2 * (size(info0{2}, 2) - e) + 1 % 故障发生在联络线 2 之前，可靠性要计算联络线 2
                            tmp_contact_re = tmp_contact_re * (1 - contact_line2_re0);
                        end
                        tmp_re = tmp_re + decision_mat0(m, n, e) * possibility0(m, e) * (1 - tmp_contact_re );
                    end
                    tmp_re = tmp_re + main_line_re0;
                    reliability0(counter) = sub_line_re0 * tmp_re;
                    counter = counter + 1;
                end
            end


            clear reliability1; counter = 1;
            for m = 1:size(info1{2}, 2)
                for n = 1:size(info1{2}{m}, 1)
                    % 本树的主线可靠性和支线可靠性
                    [~, sub_line_re1, main_line_re1] = reliability_fun00(0, info1{2}{m}(n, :), info1{2}, info1{3}, info1{4}, info1{5}, info1{6}, nd1, si1, po1);

                    % 联络线在另一树的端点的可靠性
                    if i == 1
                        contact_line_re1 = 0.995;
                    elseif mod(i, 2) == 1
                        contact_line_re1 = reliability_fun00(1, nearest_dot0(i, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6}, nd0, si0, po0);
                    else
                        contact_line_re1 = reliability_fun00(1, nearest_dot0(i+1, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6}, nd0, si0, po0);
                        contact_line_re1 = contact_line_re1 / (1 - 0.002
```

```
* 0.5 * length0(i/2) );
                        end
                        if contact_line_re1 == -1
                            disp("ERROR"); % 报错
                        end

                        if i2 == 1
                            contact_line2_re1 = 0.995;
                        elseif mod(i2, 2) == 1
                            contact_line2_re1     =     reliability_fun00(1,
nearest_dot0(i2, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6}, nd0,
si0, po0);
                        else
                            contact_line2_re1     =     reliability_fun00(1,
nearest_dot0(i2+1, :), info0{2}, info0{3}, info0{4}, info0{5}, info0{6},
nd0, si0, po0);
                            contact_line2_re1  =  contact_line2_re1  /  (1  -
0.002 * 0.5 * length0(i2/2) );
                        end
                        if contact_line2_re1 == -1
                            disp("ERROR"); % 报错
                        end


                        % 联络线在本树的端点 到 每个一级分叉点 的距离
                        tmp1 = 2*m+1;
                        tmp2 = j;
                        tmp_reliability = 1;
                        % line1
                        tmp_len = 0;
                        for o = min(tmp1, tmp2):max(tmp1, tmp2)-1
                            tmp_len = tmp_len + 0.5 * length1(ceil(o / 2) );
                            if mod(o, 2) == 0
                                tmp_reliability = tmp_reliability * 0.998; %
开关
                                tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                                tmp_len = 0;
                            elseif o == max(tmp1, tmp2)-1 && mod(o, 2) == 1
                                tmp_reliability = tmp_reliability * 0.998; %
开关
                                tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                                tmp_len = 0;
```

```
                        end
                    end
                    % 算上联络线本身的可靠性
                    contact_line_re1 = contact_line_re1 * 0.998 * (1 -
0.002 * contact_line_length);
                    % 算上联络线到一级分叉点的可靠性
                    contact_line_re1      =      contact_line_re1      *
tmp_reliability;


                    % 联络线在本树的端点 到 每个一级分叉点 的距离
                    tmp1 = 2*m+1;
                    tmp2 = j2;
                    tmp_reliability = 1;
                    % line2
                    tmp_len = 0;
                    for o = min(tmp1, tmp2):max(tmp1, tmp2)-1
                        tmp_len = tmp_len + 0.5 * length1(ceil(o / 2) );
                        if mod(o, 2) == 0
                            tmp_reliability = tmp_reliability * 0.998; %
开关
                            tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                            tmp_len = 0;
                        elseif o == max(tmp1, tmp2)-1 && mod(o, 2) == 1
                            tmp_reliability = tmp_reliability * 0.998; %
开关
                            tmp_reliability  =  tmp_reliability  *  (1  -
0.002 * tmp_len);
                            tmp_len = 0;
                        end
                    end
                    % 算上联络线本身的可靠性
                    contact_line2_re1 = contact_line2_re1 * 0.998 * (1 -
0.002 * contact_line2_length);
                    % 算上联络线到一级分叉点的可靠性
                    contact_line2_re1      =      contact_line2_re1      *
tmp_reliability;


                    % line3
                    tmp_len = 0;
                    contact_line3_re1 = contact_line_3_re1;
                    for o = tmp1 : size(nearest_dot0, 1)-1
```

```
                        tmp_len = tmp_len + 0.5 * length0(ceil(o / 2) );
                        if mod(o, 2) == 0
                            contact_line3_re1   =   contact_line3_re1   *
0.998; % 开关
                            contact_line3_re1 = contact_line3_re1 * (1 -
0.002 * tmp_len);
                            tmp_len = 0;
                        end
                    end


                    % 负荷可靠性计算
                    tmp_re = 0;
                    for e = 1:size(info1{2}, 2)
                        tmp_contact_re = 1 - contact_line3_re1;
                        if j >= 2 * (size(info1{2}, 2) - e) + 1 % 故障发
生在联络线 1 之前，可靠性要计算联络线 1
                            tmp_contact_re   =   tmp_contact_re   *   (1   -
contact_line_re1);
                        end
                        if j2 >= 2 * (size(info1{2}, 2) - e) + 1 % 故障发
生在联络线 2 之前，可靠性要计算联络线 2
                            tmp_contact_re   =   tmp_contact_re   *   (1   -
contact_line2_re1);
                        end
                        tmp_re  =  tmp_re  +  decision_mat1(m,  n,  e)  *
possibility1(m, e) * (1 - tmp_contact_re );
                    end
                    tmp_re = tmp_re + main_line_re1;
                    reliability1(counter) = sub_line_re1 * tmp_re;
                    counter = counter + 1;
                end
            end
            sum_reliability(main_counter)    =    sum(reliability0)    +
sum(reliability1);
            min_reliability(main_counter)    =    min(min(reliability0),
min(reliability1) );
            len1(main_counter)      =      dist(nearest_dot0(i,      :),
nearest_dot1(j, :)' );
            len2(main_counter)      =      dist(nearest_dot0(i2,      :),
nearest_dot1(j2, :)' );
        end
    end
end
```

```
end



if target == 3 % 第 3 问最低可靠性最大的方案
    if sum(min_reliability) == 0
        clear len1; clear len2;
        disp("费用不够建设三条联络线");
        % 只连两条联络线
        disp("正在尝试只连两条联络线");
        [val, min_reliability, len1, len2, cost, contact, h, avg] =
reliability_fun_2(supply, info0, info1, money, request_min_reliability,
target, color, invoke_time+1);
        return;
    end
    while true
        [val, max_index] = max(min_reliability); % 最低可靠性最大
        avg = sum_reliability(max_index) / 35;

        max_index1 = ceil(max_index / (size(nearest_dot1, 1) *
size(nearest_dot0, 1) ) );
        max_index2 = mod(max_index, size(nearest_dot1, 1) *
size(nearest_dot0, 1) );
        % 得到方案对应的 index
        col1 = mod(max_index1, size(nearest_dot1, 1) );
        if col1 == 0
            col1 = size(nearest_dot1, 1);
        end
        row1 = ceil(max_index1 / size(nearest_dot1, 1) );

        col2 = mod(max_index2, size(nearest_dot1, 1) );
        if col2 == 0
            col2 = size(nearest_dot1, 1);
        end
        row2 = ceil(max_index2 / size(nearest_dot1, 1) );
        if row1 ~= row2 || col1 ~= col2
            break
        else
            min_reliability(max_index) = 0;
        end
    end
    fprintf("三条联络线的情况下最大的最低可靠性是%f\n", val);
    fprintf("此方案用户平均可靠性是%f\n", avg);
    contact = [row1, col1; row2, col2; size(nearest_dot0, 1),
```

```
size(nearest_dot1, 1)];

    len2 = len2(max_index); % 第二条联络线长度
    len1 = len1(max_index);
    cost = 3 * 56.8 + 325.7 * (len1 + len2 + len3); % 联络线花费

    hold on; % 画图
    line([nearest_dot0(row1,       1),       nearest_dot1(col1,       1)],
[nearest_dot0(row1,   2),   nearest_dot1(col1,   2)],   'Color',   color,
'LineWidth', 4);
    hold on;
    line([nearest_dot0(row2,       1),       nearest_dot1(col2,       1)],
[nearest_dot0(row2,   2),   nearest_dot1(col2,   2)],   'Color',   color,
'LineWidth', 4);
    hold on;
    line([nearest_dot0(size(nearest_dot0,             1),             1),
nearest_dot1(size(nearest_dot1, 1), 1)], ...
        [nearest_dot0(size(nearest_dot0,             1),             2),
nearest_dot1(size(nearest_dot1, 1), 2)], ...
        'Color', color, 'LineWidth', 4);
    figure;
    h = topology_plot2(info0{3}, info0{4}, info0{5}, info1{3}, info1{4},
info1{5}, contact);


elseif target == 4 % 第 4 问费用最低的方案
    len = len1 + len2;
    while true
        [min_len, min_index] = min(len); % 第二条联络线最短的方案
        if min_len == 20000
            % 如果连两条线怎么样都达不到可靠性的要求
            disp("三联络线无法达到可靠性要求");
            cost = -1; val = -1; contact = -1; h = -1;
            return;
        elseif min_reliability(min_index) < request_min_reliability % 如果
不满足最低可靠性的要求
            len(min_index) = 20000; % 则舍去这个方案
        else
            % 得到方案对应的 index
            min_index1   =   ceil(min_index   /   (size(nearest_dot1,   1)   *
size(nearest_dot0, 1) ) );
            min_index2   =   mod(min_index,   size(nearest_dot1,   1)   *
size(nearest_dot0, 1) );
            col1 = mod(min_index1, size(nearest_dot1, 1) );
```

```
            if col1 == 0
                col1 = size(nearest_dot1, 1);
            end
            row1 = ceil(min_index1 / size(nearest_dot1, 1) );
            col2 = mod(min_index2, size(nearest_dot1, 1) );
            if col2 == 0
                col2 = size(nearest_dot1, 1);
            end
            row2 = ceil(min_index2 / size(nearest_dot1, 1) );

            if row1 == row2 && col1 == col2
                len(min_index) = 20000;
                continue;
            end

            cost = 3 * 56.8 + 325.7 * (min_len + len3); % 总花费
            val = min_reliability(min_index); % 本方案最低可靠性
            avg = sum_reliability(min_index) / 35;
            disp("三条联络线的情况下满足了可靠性要求");
            fprintf("最低费用为%f\n", cost);
            contact = [row1, col1; row2, col2; size(nearest_dot0, 1),
size(nearest_dot1, 1)];
            hold on; % 画图
            line([nearest_dot0(row1,    1),    nearest_dot1(col1,    1)],
[nearest_dot0(row1,   2),   nearest_dot1(col1,   2)],   'Color',   color,
'LineWidth', 4);
            hold on; % 画图
            line([nearest_dot0(row2,    1),    nearest_dot1(col2,    1)],
[nearest_dot0(row2,   2),   nearest_dot1(col2,   2)],   'Color',   color,
'LineWidth', 4);
            hold on;
            line([nearest_dot0(size(nearest_dot0,          1),          1),
nearest_dot1(size(nearest_dot1, 1), 1)], ...
                [nearest_dot0(size(nearest_dot0,          1),          2),
nearest_dot1(size(nearest_dot1, 1), 2)], ...
                'Color', color, 'LineWidth', 4);
            figure;
            h  =  topology_plot2(info0{3},  info0{4},  info0{5},  info1{3},
info1{4}, info1{5}, contact);
            return;
        end
    end
end
return;
```

程序二十六：背包问题求解程序

package_problem2.m

```
function [max_val, choice, f] = package_problem2(capacity, v, w)

% n 件物品（重量为 w，价值为 v），背包容量为 m 的 01 背包问题
num_of_load = length(v);
f = zeros(1, capacity); % f(i, j) j 体积下前 i 个物品的最大价值
counter = 1;

for i = 1:num_of_load
    for j = 1:capacity
        if i == 1 && j >= v(i)
            f(i, j) = w(i);
            if j == capacity
                choice(counter) = i;
                counter = counter + 1;
            end
        elseif i == 1 && j < v(i)
            f(i, j) = 0;
        elseif j < v(i)
            f(i, j) = f(i-1, j);
        elseif j == v(i)
            f(i, j) = max(f(i-1, j), w(i) );
            if w(i) > f(i-1, j) && j == capacity
                choice(counter) = i;
                counter = counter + 1;
            end
        else
            f(i, j) = max(f(i-1, j), f(i-1, j-v(i) ) + w(i) );
            if f(i-1, j-v(i) ) + w(i) > f(i-1, j) && j == capacity
                choice(counter) = i;
                counter = counter + 1;
            end
        end
    end
end
max_val = f(num_of_load, capacity)
```