基于用电可靠性的配电网规划

摘要

配电网规划中，在满足需求各项指标的前提下减少建设成本并提高用电可靠性是建设的关键。本文通过建立基于目标优化的单供、双供树状结构配电网供电模型，求解出了能够满足题设要求的理想配电网，并对其相关数据（分叉点、可靠性和建设成本等）进行了研究。

针对于问题一，本文在基于K-means**聚类**结合**最小哈密顿图**的主线模型，同时利用**最小生成树**的优良拓扑特性，首先将主线和支线的数据进行分层处理，随即再进行逐层求解。已知的电源和负荷节点进行拓扑结构的确认，进而通过结合线路造价成本，微调在最小生成树中的分支节点，以此来构造电网中的分叉点，并将其坐标作为决策变量，构建**单目标优化模型**进行求解。在确认了网络最终的拓扑结构之后，进一步求解其建造成本及用电可靠性。模型的正确性将在问题二的测试中得以展现。

针对于问题二，面对双电源单供电网间负荷分属电源情况的判定，本文提出了一种针对问题目标的**基于几何分析的二分类**算法，结合问题一建立的单目标优化模型进行拓展，最终得到了两个网络的**最低可靠性都在96%以上，平均用电可靠性达到了97.66%，且数据的方差控制在了10^{-5}数量级。**结合现实生活也可知，在增加了电源后，通过优化配电网分配，必定可以增加用电可靠性，如此验证了模型一、二的正确性。

对于问题三、四，都是在问题二建立的配电网基础上添加联络线，使其变为双供电网。问题三需要在控制成本上限的情况下，将最低的用电可靠性得到最大的提升；而问题四则需要保证用户的最低用电可靠性在阈值之上，控制建造成本最低。本文建立了**基于容斥原理的概率模型**对用户的用电可靠性进行描述，且使用期望的方式刻画扩展后的功率及其伴随的花费。在此基础上，借助贪心算法的思想，首先对最低用电可靠性的用户进行联络线连接的考虑，这样必定能在一定限度内提升这些用户的用电可靠性，由此提升整个网络中最低用电可靠性的水平。此外，由于贪心算法带来的时间复杂度很高，本文提出了一种**启发式的贪心算法**，使用**两级最小堆**的结构，结合**负荷分块改进算法**，能够在面对大规模问题时快速求得近似最优解，将求解的时间复杂度从指数级别优化到对数级别。最后，根据两个问题不同的条件限制，本文分别设立了两个目标函数，结合问题二的两个单供配电网模型进行联络线的选择，以及模型成本、用户可靠性等信息进行求解。最终，模型经过了的多轮优化后，在负荷可靠性梯度图中，可以看到，优化后的区域用户用电可靠性差距减小、分布均衡，如此验证了该模型的有效性。

在文章的最后，本文客观地指出了模型的优缺点。此外，以上问题的所有结果数据均存放于附录中。

关键词：K-means 最小生成树 容斥原理 贪心算法 负荷分块 两级最小堆

一、问题重述

1.1 问题背景

随着城市化的推进，解决供电问题成为了当下城市建设中的一个主要难点。由于配电线路逐渐规模化和集群化，在城市化的进程中需要解决针对城市配电网线路输送紧张，同时兼顾保障供电可靠性等问题[1]。因此，电网在规划阶段需要解决的问题是在满足需求指标的前提下减少投入并增加电网可靠性，工作人员应该积极研究供电可靠性提升的措施与方法，从而更加科学的进行配电网规划设计，进而提升配电网运行的稳定性与安全性[2]。

本题以上述生活需求为背景，讨论了如下的问题：

如果一批用户变压器（下面简称用户）仅由一个电源变电站（下面简称电源）供电，称为**单供**。

这时配电网由电线和开关联接成以电源为根节点的**树状结构图**，使得每个用户所在顶点都在图中有路（电线）联接到电源根节点。一些电力用户一旦发生停电，无论停电时间长短，都会带来较大损失，降低用电满意度。

因此，定义**用户用电可靠性**：指定时间段内不因配电网故障停电或限电的概率。为了提升用户用电可靠性，可在两个电源的单供配电网之间建立联络线，并增设开关和扩充电源可供电功率，形成**双电源供电配电网**（简称双供配电网，如图 1 所示）。

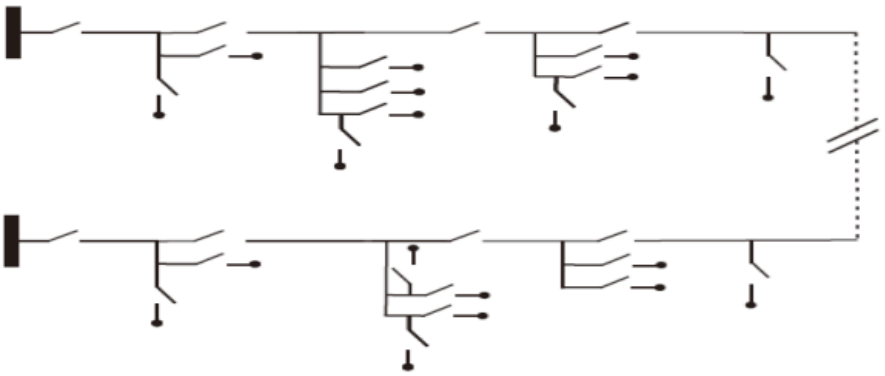

图 1　双电源供电配电网图示

此外，还需要满足以下要求：

开关设置原则：

1.配电网中开关（电源出线后开关除外）的设置必须使得网中某处发生故障时，通过开关隔离故障后，保持供电的用户需求功率之和最大化；

2.在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；

3.配电网中开关的设置包含但不限于以下情况：每个用户前端有开关，每个分岔点后端的每条支路上有开关，双供配电网的每条联络线上有开关。

配电网设施可靠性单元划分及其可靠性：

忽略电源至它的后端第一个开关部分，忽略用户至它的第一个前端开关部分，配电网设备可靠性（故障）单元由电源、用户、开关，以及仅含两个开关之间的路（下面称为故障单元路）构成。每个单元设备在指定时间段内正常运行的概率称为单元设备可靠性，它等于 1 减去该单元设备的故障率。

双供配电网用户供电调度原则：

1.满足一个用户全部需求功率，否则断开该用户；

2.首先满足各自单供配电网内用户的需求；双供电源多余功率的分配优先提高全配电网供电功率总和，然后提升全配电网最低的用电可靠性。

1.2 需要求解的问题

对题目进行分析总结，得到各个问需要解决的问题如下：

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。设计建造费用最低的单供配电网供电所有用户，给出树状配电网的分叉点坐标，并计算该配电网中每个用户的用电可靠性。

问题二：已知两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。设计建造费用最低的两个单供配电网，使得每个用户都被供电。给出树状配电网的分叉点坐标，并计算该配电网中每个用户的用电可靠性。

问题三：在第 2 题结果的基础上，通过建立两个单供配电网之间的联络线，增设开关，并扩充电源可供电功率，形成双供配电网，以提高用户的用电可靠性。假设两个电源各自能扩充可供电功率 50%，建造双供配电网总花费上限为 X，求使得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。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：在第 2 题结果的基础上，通过建立两个单供配电网之间的联络线，增设开关，并扩充电源可供电功率形成双供配电网，以提高用户的用电可靠性。假设两个电源各自能扩充可供电功率 50%，设计建造总费用最低的双供配电网，使得双供配电网中每个用户的用电可靠性不低于 Y%。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。

二、问题分析

2.1 问题一的分析

问题一要求，针对一批已知的用户及其需求和电源及设备的建造费用，设计一个费用最低的单供配电网络，设计要素包括网络的部分节点（将树状输电网中除了电源（根节点）和用户（叶节点）的节点称为“分叉点”）坐标以及各个节点之间的拓扑连接关系。同时处理这两个要素比较困难，故采取先确定连接电源与用户之间的拓扑结构，再考虑优化分叉点的坐标位置。

考虑到主线和支线上线路与开关的造价差距较大，导致主线在造价里占据更大的权重。故再次使用分层求解的思路，首先确定主线网络，然后再确定支线网络：主线求取分叉点时，考虑到负荷聚簇分布的优良位置特性，使用**聚类**方法确定中心分叉点；设计主线的拓扑结构时，则选取了**最小哈密顿路**作为最优的解。支线首先设计拓扑结构。我们发现最小生成树有良好的拓扑特性，它舍去了冗余的连线，并且最小化了网络线路的长度总和。

因此，本文在**最小生成树算法**的基础上确定网络的拓扑结构。然后处理支线的分叉点坐标。我们以分叉点坐标为决策变量，以建造费用为目标函数，建立**基于单目标优化的单供配电网模型**，并且利用 $MATLAB$ 的 $fminunc()$ 函数求解总花费的最小值，得到了最优的分叉点坐标。

图 2　求解单供网络分叉点的思路流程图

2.2 问题二的分析

问题二要求，设计建造费用最低的**两个**单供配电网，与问题一唯一不同在于提供了两个电源，需要建立两个相互独立的单供配电网。

我们基于问题一进行分析，确定每个负荷的供电电源，即对负荷集合进行二分类后，可对两个节点集合各自建立问题一中的模型进行求解。

由于假设负荷节点的分布是随机、均匀的，通过传统的聚类算法难以求解最优分类。因此，我们提出针对问题目标，即使建造费用最低的**基于几何分析的二分类方法**，结合问题一的模型完成设计要求。

2.3 问题三、四的分析

问题三和四要求：需要建立一条联络线，连接两个单供电网，以实现在不同背景下最大程度提高用户用电可靠性的最低水平。本文设定联络线的两个端点分别位于两个网络的分叉点上，将每一段供电的线路拆分成多个子路段，根据各个子路段之间的独立性，使用容斥原理对用户可靠性进行描述。同时，考虑在各种不同调度方案下的开关闭合情形，制定相应的供电调度方案。

由题意得知，扩展电源会优先提高最低可靠性的负荷的用电可靠性。因此，我们将采用**贪心算法**来构造联络线，在成本允许的范围内，优先考虑对可靠性最低的负荷搭建联络线，以此来最大程度上地提升可靠性最低负荷的用电可靠性。

模型中电源遵从**优先供给原则**，首先考虑使用本供电网中电源，如果本供电网电源无法连接，才会考虑使用另一个网络中的电源。此外，使用期望的方式计算扩展后网络的功率，并在此基础上，进一步考虑扩供带来的花费期望。

由于三、四问对于最终目标都是一致的，即最大限度提升整体的用电可靠性水平，只是在具体限制下有所不同，因此针对不同的限制，设立相应的约束目标。其余整体论断大致相同，因此采用同一套约束方案。由此，求得使**双供配电网中最低用电可靠性达到最大**的联络线和开关设计（问题三）和对**双供配电网中建设成本达到最小**的联络线和开关设计（问题四），并最大程度上保证用户的用电可靠性。

三、模型假设

为了适当地对模型进行合理简化，本文做出如下假设：

1、负荷分布在主线（路网）附近,并且成簇聚集；

2、开关设在导线的端点处，靠近分叉点或负荷；

3、联络线支持双向供电，其他供电线路仅支持单向供电；

4、在双供电网中，联络线两端联络点位于主线上。

四、符号说明

序号	符号	意义
1	B_i	树状配电网中的电源 i
2	K_i	树状配电网中电源的位置
3	G	树状配电网的邻接矩阵
4	A_i	树状配电网中第 i 个负荷节点
5	D_i	树状配电网中的第 i 个分叉点
6	$cost_{circuit}$	线路单价向量
7	$cost_{switch}$	开关单价向量
8	$length$	线路长度向量
9	num	开关数量向量
10	$load[i]$	树状配电网中节点 i 所在子树的负荷数
11	fa_i	树状配电网中节点 i 的父亲节点
12	$type(ifa_i)$	配电网中，线路 (i, fa_i) 的电线线路类型
13	$length(k)$	配电网中，线路类型为 k 的线路长度
14	$dis(i,j)$	配电网中，节点 i 到节点 j 的距离
15	$r[i]$	配电网中，节点 i 的用电可靠性
16	$C[i]$	用户用电的状态变量
17	P_i	双供电网中，网络 i 的配电功率
18	p_{k_i}	双供电网中，联络线位于 i 电源部分的端点（联络点）
19	N_i	双供配电网中，由 bat_i 供电部分的网络部分
20	$V[N_i]$	双供电网中，位于 i 电源部分的负荷集合

五、模型的建立与求解

5.1 问题一：设计建造费用最低的单供配电网络

针对于问题一，我们将设计对象分为拓扑结构与分叉点位置两部分。

拓扑结构是网络中点之间连线的相对关系，分叉点是配电网络中线路分叉的节点。电源到负荷的通路，往往不会是直来直去的，而是经过多级输送，分叉点就是电源到负荷间处在中间层级。一个给定的拓扑结构和分叉点位置的网络设计如图所示：

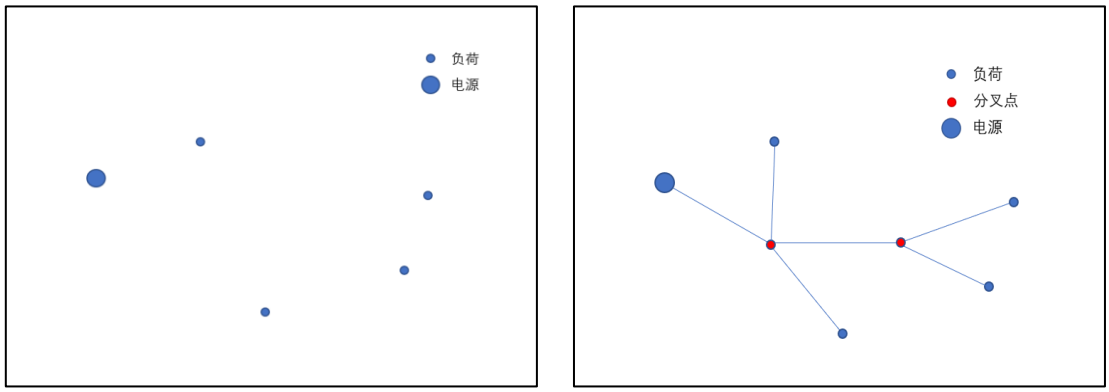

图 3 在给定的电源和负荷下对网络的构造示意

起初，我们假设负荷均匀的随机分布在一定的区域内，并由此设计了基于最小生成树的单变量最优化模型；然而，分析题目并结合生活实际，我们发现，电网的供电目标往往是一个小区，一个村庄或是一些工厂这样的群体性对象，负荷的分布具有天然成簇的优良特征。如下图所示，可以看出，图中的绿色点（负荷）的分布具有明显聚集性：

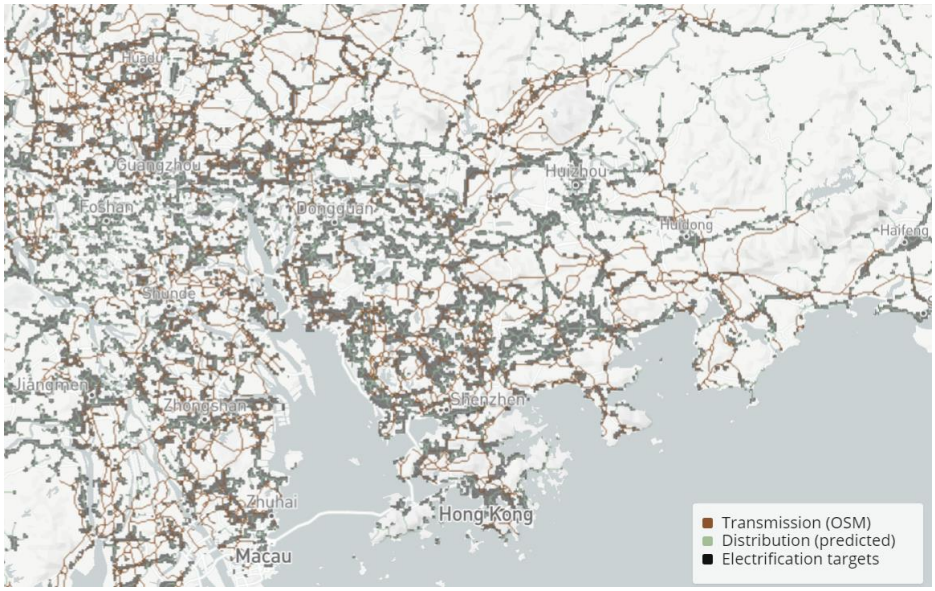

图 4 现实生活中，城市间的负荷分布示意图

直接使用最小生成树模型得到的结果似乎更具有普适性，但无法利用聚簇分布的优良特征。故针对负荷的这种分布特点，我们优化了问题的解法。

我们根据“电源——分叉点（类似于现实生活中的变压器，能控制多个用户或支路的汇点）——用户”三个层次，对应到树状结构中的三种状态：“根节点——分支节点——叶节点”，如此一来，不仅方便后续图的建立和操作，还能层次分明、逻辑清晰，且便于最终选择相应的线路进行建设，以求得最小成本耗费下的分叉点坐标和用户用电可靠性等目标结果。

5.1.1 模型的建立——单供配电网模型

（1）单供配电网最小花费的层次求解模型

本题的优化目标是设计建造费用最低的配电线路。

确定配电网络的因素由上文所述包括分叉点位置与拓扑结构；而影响参数包括线路的造价$cost_{circuit}$,开关的造价 $cost_{switch}$，根据题目给出的数据，在网络不同位置，开关与线路具有不同的价格。这导致求解的复杂性的增加。而通过观察，已知

$$cost_{circuit} = (325.7 \quad 188.6 \quad 239.4) \tag{1}$$

$$cost_{switch} = (2.6 \quad 58.6) \tag{2}$$

可以看出，主线上线路和开关的造价均高于支线。故根据层次分析、分层求解的思想，不妨将单价视为模型的权重，按权重高低，分主线、支线两部分，先处理主线、后处理支线进行问题的求解。

这与上文中提到的聚簇处理的思想紧密结合，本文针对主线提出了**基于 K-means 聚类与最小哈密顿路的主线模型**，同时针对支线网络中负荷分布不规律的情况，引入**基于最小生成树的配电网拓扑设计**，提出了**基于单目标优化的单供配电网模型**，二者分层有机结合，形成了本题中所使用的**单供配电网最小花费的层次求解模型**。

（2）基于 K-means 聚类与最小哈密顿路的主线模型

首先解决主线的最优化设计问题。设计主线就要设计主线的分叉点和拓扑结构。设图中负荷属于集合A，$A_i \in A$；主线部分共有n_M 个分叉点，构成集合K,按照接入网络的次序，有第i个分叉点满足

$$K_i \in K, 1 \leq i \leq n_M \tag{3}$$

则主线L_M的长度为：

$$length(L_M) = \sum_{i=1}^{n-1} K_i K_{i+1} \tag{4}$$

按照开关设计原则，电源附近需要设置开关S_0，分叉点K_i前设置开关S_{ki}，则主线部分开关的数量为：

$$size(S_M) = 1 + size(K) \tag{5}$$

需要的总花费为：

$$cust_M = length(L_M) \cdot cost_{circuit}(1) + size(S_M) \cdot cost_{switch}(1) \tag{6}$$

局部的最优化目标为：

$$\min\left(\text{length}(L_M) \cdot cost_{circuit}(1) + size(S_M) \cdot cost_{switch}(1)\right) \tag{7}$$

观察上式可知，分叉点的数量的增多不利于总花费的下降。再结合成簇分布的优良特性，通过对负荷进行聚簇的方法，进行分叉点的构造。

首先随机选取k个中心点，定义损失函数，通过迭代，使得聚类结果对应的损失函数最小。其中，损失函数可以定义为各个样本距离所属簇中心点的误差平方和，即：

$$J(c,\mu) = \sum_{i=1}^{M} ||x_i - \mu_{c_i}||^2 \tag{8}$$

这就是著名的 K-Means 算法。

通过该算法，我们将数据点分为k组，定义第i组的中心点为C_i,

$$C_i \in C, 1 \le i \le k \tag{9}$$

由于取这些中心点为分叉点，故集合C与集合K有相同的元素,即：

$$C = K \tag{10}$$

然后要确定拓扑关系。以题目要求，即给K中元素确定一个偏序关系，保证优化目标得以完成。

由于分叉点数量已经由k确定，故仅需令路径长度最小。

该问题可以抽象为从电源出发的，寻找一条遍历所有点的路径的问题，即最小哈密顿路问题。

（3）最小生成树的优良拓扑特性

首先，设计建造费用最低的配电线路应尽量减少多余的连线，使得电源与每个负荷之间的通路唯一，这要求配电线路具有树形结构。如此，不仅满足了题目中的开关设置原则 2，并且也能寻找出成本最低的拓扑方案。

其次，由聚簇后，负荷失去了成簇的分布特性，使用最小生成树可以得到解决问题的一种普适的方案。

最后，主线上的分叉点设为聚簇的中心节点（以下统称为中心分叉点），故选取以中心分叉点为根节点的辐射式网络易于实现局部最优。

（4）基于最小生成树的配电网拓扑设计

首先，对中心分叉点和负荷节点求最小生成树。此时负荷节点可能是树的分支节点，根据生活常识，用户往往是处于配电线路的末端，作为树状网络的叶子节点。因此，我们通过构造分叉点加入网络，微整网络拓扑，得到基于最小生成树的配电网拓扑设计。

分叉点的构造：

1、对最小生成树中原先作为分支节点的负荷节点 A_i，在其附近构造一个分叉点 A_i'；

2、删除节点 A_i 的临界线路：

$$\{A_i \mathrm{D}_j \mid j = 1, 2, \ldots\} \tag{11}$$

3、增加线路 $A_i\ A_i'$ 以及分叉点 A_i' 的邻接线路：

$$\{A_i' \mathrm{D}_j \mid j = 1, 2, \ldots\} \tag{12}$$

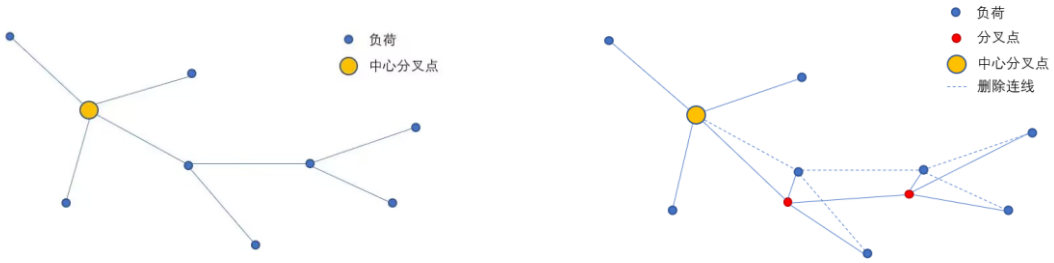

图 5　最小生成树示意图[左]　　分叉点的构造示意图[右]

最终，我们得到拓扑结构和原先最小生成树相似的树状局部配电网设计，其中分叉点作为根节点、分叉点作为分支节点、负荷作为叶子节点。

设树状配电网的邻接矩阵为 G，中心分叉点为C_j，分叉点集合为

$$\mathrm{D} = \{d_i \mid i = 1, 2, \ldots, n_b\} \tag{13}$$

叶子集合为：

$$L = \{l_i \mid i = 1, 2, \ldots, n_l\} \tag{14}$$

（5）开关的铺设以及线路的选型

在拓扑结构确定之后，我们考虑开关的铺设以及线路的选型。

根据开关设置原则 3“每个用户前端有开关，每个分叉点后端的每条支路上有开关”，基于成本考虑，满足最低的设计要求，即每条线路增设一个开关。

关于线型的选择，由附录第 4 点，主线可承载 2 倍的电源额定功率，支线 A 可承载至多两个负荷，支线 B 可以承载 3 个及以上的负荷。由于主线和支线 B 在该问题背景下无法区分，所以，根据经验，我们假设与电源直接相连的线路使用主线，其他情况使用支线 A、B。

设节点 i 的所在子树的负荷数为 $load[i]$，由 G 从C_j出发的深度优先搜索可以求得每个节点 i 的孩子节点集合，将其设为

$$son_i = \{s_{ij} \mid j = 1,2,\ldots,n_i\} \tag{15}$$

设其父亲节点为 fa_i，由递推公式：

$$load[i] = \begin{cases} 1, & i \in L \\ \sum_{s_{ij} \in son_i} load[s_{ij}], & i \in B \end{cases} \tag{16}$$

可以求得 $load[i]$，此时可以确定所有线路的选型。

对于节点 i，我们考察线路 (i, fa_i) ,即 ifa_i，设其线型为 $type(ifa_i)$，有：

$$type(ifa_i) = \begin{cases} 1\text{（主线）}, & fa_i = bat \\ 2\text{（支线 A）}, & load[i] \leq 2 \\ 3\text{（支线 B）}, & load[i] > 2 \end{cases} \tag{17}$$

（6）分叉点的选址以及线路建造费用的计算

设线路单价向量为 $cost_{circuit}$，开关单价向量为 $cost_{switch}$，线路长度向量为$length$，开关数量向量为 num。

为了确定 $length$，我们引入决策变量——分叉点坐标，设为 $\{(x_i, y_i) \mid i \in B\}$，同时，根据聚类后“负荷均匀、随机地分布于一定区域，并且中心分叉点位于区域中心”构造测试数据集。

设负荷坐标为$\{(x_i, y_i) \mid i \in L\}$，中心分叉点坐标为$\{(x_i, y_i) \mid i \in C_j\}$。

使用欧氏距离公式，可以计算 i, j 两节点之间的距离：

$$dis(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{18}$$

由此可以确定：

$$length(k) = \sum_{G(i,j)=1 \text{ 且 } type(i,j)=k} dis(i,j),\ k = 1,2,3 \tag{19}$$

而开关数量向量的确定，容易知道每个负荷前有一个负荷前开关，每个分叉点（包括电源）后有一个主线开关，推得 $num = (n_l, n_b)$。

因此，可得线路建造费用总和为：

$$cost_{sum} = cost_{circuit} * length + cost_{switch} * num \quad (20)$$

（7）用户可靠性的计算

设节点 i 的用电可靠性为 $r[i]$，已知其父亲节点为 fa_i，有如下式子：

$$r[i] = r|fa_i| * (1 - 0.5\%) * (1 - 0.2\%) * \left(1 - 0.002 * dis(i, fa_i)\right) \quad (21)$$

类似于 $load[i]$，上式可通过深度优先搜索的过程中递推计算。最终，可以得到：$\{r[i] \mid i \in L\}$ 为用户用电可靠性。

（8）基于单目标优化的单供配电网模型

要求建造费用最低的局部配电网供电所有用户，基于最小生成树算法，我们设计了以中心分叉点为根节点的辐射式树状结构，一方面，满足了供电给所有用户的需求，另一方面，这个优良的拓扑结构也为了减少成本上的支出打下了基础。

之后，我们通过引入决策变量——分叉点的坐标 $\{(x_i, y_i) \mid i \in B\}$，量化描述线路造价，来建立基于单目标优化的单供配电网模型。

目标函数：最小化线路建造费用，即 $min(cost_{sum})$，由于开关部分的花费在拓扑结构确定后也随即确定，所以式子进一步可简化为：

$$min(cost_{circuit} * length) \quad (22)$$

约束条件：

1、配电网是以电源为根节点的树状结构；

2、开关设置原则以及安培限流；

3、线路安培限流；

以上约束通过构造性算法得以满足，因此最后得到的实际上是无约束模型。

综上所述，建立单目标优化模型：

$$min(cost_{circuit} * length)$$

$$\begin{cases} load[i] = \begin{cases} 1, & i \in L \\ \sum_{s_{ij} \in son_i} load[s_{ij}], & i \in B \end{cases} \\ type(if a_i) = \begin{cases} 1（主线）, & fa_i = bat \\ 2（支线\ A）, & load[i] \le 2 \\ 3（支线\ B）, & load[i] > 2 \end{cases} \\ dis(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \\ length(k) = \sum_{G(i,j)=1 \text{ 且 } type(i,j)=k} dis(i,j), k = 1,2,3 \\ r[i] = r|fa_i| * (1 - 0.5\%) * (1 - 0.2\%) * (1 - 0.002 * dis(i, fa_i)) \end{cases} \tag{23}$$

5.1.2 模型的求解

问题一建立的基于单目标优化的单供配电网模型求解的算法步骤如下：

Step1：测试数据集提取

通过提取测试数据集，生成的负荷数据存储于'load_demo.xlsx'文件中。

序号	x 坐标（km）	y 坐标（km）	需求量（兆瓦）
1	-25000.4	-13115.2	315
2	-25948	-13394.1	1000
3	-26066.5	-13418.3	500
...	...	...	...

表 1 部分负荷数据集示意

Step2：利用 K-MEANS 算法求取负荷的聚类中心点

a）读取'load_demo.xlsx'文件，生成点集；

b）随机选取 k 个点作为中心点；

c）定义损失函数：$J(c,\mu) = min \sum_{i=1}^{M} ||x_i - \mu_{c_i}||^2$；

d）重复如下过程直到收敛：

d.1 对于每一个样本点 point，将其分配到距离最近的中心 group；

d.2 对于每一个类中心 k，重新计算该类的中心。

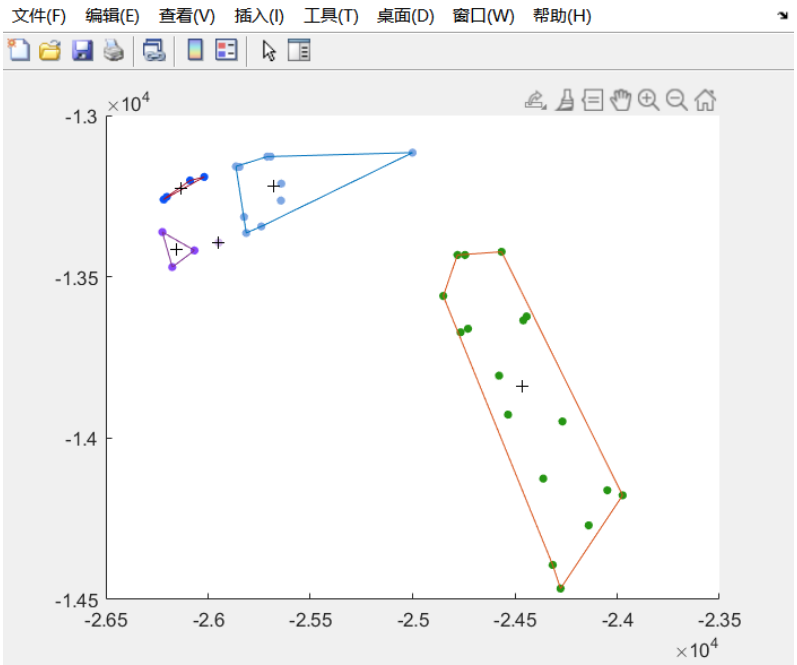

图 6　聚类情况示意图

Step3：利用状态压缩动态规划算法求最小哈密顿路

a）读取’center.xlsx ’文件，将得到的中心分叉点使用二进制形式遍历其所有状态；

b）针对每一个状态，利用位运算判断是否点在路径中；

c）进行状态的转移，状态转移方程如下：

$$dp[S][i] = \min(dp[S][i], dp[S1 \ll i][j] + weight[j][i])$$

Step4：利用 Kruskal 算法求最小生成树

a）读取’load_demo.xlsx ’文件，将二维坐标点之间的距离转化为带权边集 E，将边集 E 中的边按边权从小到大排序；

b）初始化并查集，依次遍历边集 E 中的边 e，判断 e 的端点是否连通，若不连通，则将 e 在最小生成树的邻接矩阵 G_{mst} 中标记，并且并查集中合并两端点，否则跳过；

c）返回 G_{mst}。

Step5：构造分叉点

a）预处理 G_{mst} 中所有节点的度数 deg；

b）对 G_{mst}，从 1 号电源节点出发进行深度优先搜索（后面简称 dfs），对于 $deg(i) > 1$ 的节点构造其邻接的分叉点（构造过程详见模型的建立）；

c）返回包含构造分叉点的配电网拓扑的邻接矩阵 G。

Step6：确定线路选型

a）对 G 进行 dfs，根据公式（6）计算 $load$ 数组；

b）根据 dfs 得到 $load$ 数组以及公式（7）确定线路选型，并在 G 中赋予相应权重 $type$。

Step7：利用 MATLAB 求解器求无约束函数最值，计算最低建造费用的网络

a）预处理邻接矩阵 G 获取对应边集 E_G；

b）对决策变量分叉点的坐标进行编码为：

$$X = [x_1, x_2, \ldots, x_n]$$

根据公式（8）-（12）定义目标函数，其中遍历边集 E_G 以达到遍历图的效果，优化算法效率；

c)通过优化函数 $fminunc()$ 求解最值 $fval_{best}$ 以及最值点 x_{best}，$fval_{best}$ 为模型求解的最低建造费用；

d）对 x_{best} 解编码获得分叉点的坐标，写入文件'bifurcation_coordinates_t1.xlsx'中。

Step6：代入最值点，求用户用电可靠性

将 step7 中获取分叉点的坐标与负荷电源坐标整合为配电网拓扑 G 的二维坐标信息 p_{xy}，对 G 进行 dfs 的过程中根据公式(13)计算用户用电可靠性为 r，写入文件'reliability.xlsx'。

Step7：通过 GraphPlot 的相关 API 将最优配电网可视化

5.2 问题二：设计建造费用最低的两个单供配电网络

根据问题一奠定的基础，本问中再增设一个电源后，需讨论求解新增电源后配电网的分叉点坐标位置确认，以及用电可靠性的大小。问题二中重要的是如何在双电源的情况下分配用户的供电区域，进而讨论其分叉点坐标以及用电可靠性。

5.2.1 模型的建立

（1）**传统聚类方法的尝试**

将问题转化为二分类问题后，首先想到应用传统的聚类/分类方法，如 k 均值聚类、层次凝聚聚类法以及均值漂移聚类等，但这些方法多少存在以下问题：

1、不适合发现非凸形状的簇，我们假设节点是随机均匀分布的，并不是一个个具有凸形状的簇；

2、无法确定最终分类的数量，根据问题需要要求将负荷点集合二分类，像均值漂移无法确定分类数量；

3、分类的适应度函数和问题目标函数关联较小，我们的目标是最小化线路建造费用，而传统聚类方法往往是基于距离、密度进行的。

（2）**针对问题目标的基于几何分析的二分类算法**

在尝试传统方法后，我们针对问题优化目标提出一种基于几何分析的二分类算法：

1、设 K_i 代表电源的位置，并作两个电源之间的连线

$$l = K_1K_2 \tag{24}$$

设 l 的中点为 $O(x_O, y_O)$，令

$$k = \frac{y_{K_1} - y_{K_2}}{x_{K_1} - x_{K_2}} \tag{25}$$

用直线方程一般表达式描述 l 可得

$$l: f(x,y) = y - y_O - k(x - x_O) = 0 \tag{26}$$

2、作 l_1、l_2 垂直于 l，且关于点 O 对称，设 l_1、l_2 和 l 的交点分别为 T_1、T_2 令

$$t = T_1O = T_2O \tag{27}$$

根据几何关系可得：

$$\begin{cases} x_{T_{1,2}} = x_O \pm \frac{t}{\sqrt{k^2+1}} \\ y_{T_{1,2}} = y_O \pm \frac{kt}{\sqrt{k^2+1}} \end{cases} \tag{28}$$

同理，可以通过直线方程的一般表达式描述 l_1、l_2：

$$\begin{cases} l_1: f_1(x,y) = y - y_{T_1} + \frac{1}{k}(x - x_{T_1}) = 0 \\ l_2: f_2(x,y) = y - y_{T_2} + \frac{1}{k}(x - x_{T_2}) = 0 \end{cases} \tag{29}$$

l_1、l_2 将平面点集划分为 M_1、M_2、M_3 三个部分，其中：

$$\begin{cases} M_1 = \{(x_i, y_i) \mid f_1(x_i, y_i) > 0\} \\ M_2 = \{(x_i, y_i) \mid f_2(x_i, y_i) < 0\} \\ M_3 = \{(x_i, y_i) \mid f_1(x_i, y_i) * f_2(x_i, y_i) < 0\} \end{cases} \tag{30}$$

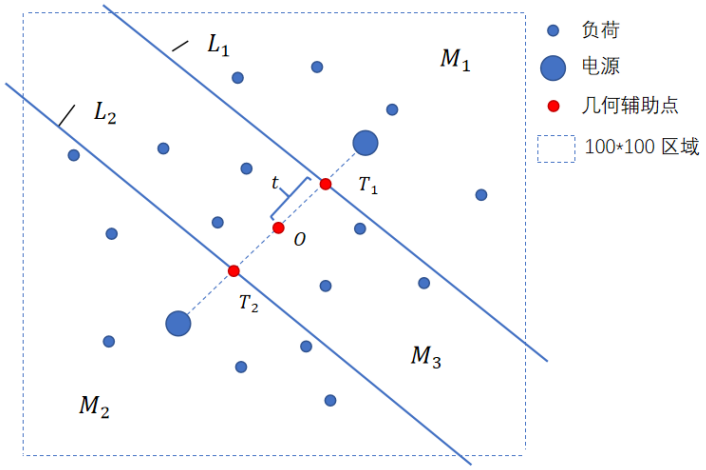

图 7　二分类算法构造示意图

3、选择一个较小的值 α，平移 l_1、l_2，即通过改变 t，使得 $|M_3| = \alpha$，这里可以利用二分法确定 t 的取值；

4、将 M_1 和 K_1 分类至集合 S_1 中，M_2 和 K_2 分类至集合 S_2 中，用一个 0-1 状态向量 s 描述 M_3 中点的分类情况，其中‘0’代表属于集合 S_1，‘1’代表属于集合 S_2。

设 M_3 中属于 S_1 的点集为 s_1，属于 S_2 的点集为 s_2，由此可以简洁地描述分类情况：

$$S_i = M_i \cup S_i, \quad i = 1,2 \tag{31}$$

5、枚举状态向量 s；

6、对集合 S_1、S_2 各自建立问题一中的单供配电网模型，各自求得其对应的最低耗费 $min_{S1}(cost_{sum})$ 以及 $min_{S2}(cost_{sum})$。

设：

$$cost = min_{S1}(cost_{sum}) + (min_{S2}(cost_{sum}) \tag{32}$$

令 $cost_{best}$ 为最优的 $cost$，S_{best} 为最优 $cost$ 所对应的 S_1；

7、更新 $cost_{best}$ 以及 S_{best}；

8、若枚举还未结束，回到算法的步骤 5，否则结束。

（3）**单供配电网模型的拓展**

基于前面提出的基于几何分析的二分类算法，结合问题一的单供配电网模型，建立得到问题二的模型：

$$min(min_{S1}(cost_{sum}) + (min_{S2}(cost_{sum}))$$

$$\begin{cases} k = \dfrac{y_{K_1} - y_{K_2}}{x_{K_1} - x_{K_2}} \\ \begin{cases} l: f(x,y) = y - y_O - k(x - x_O) = 0 \\ l_1: f_1(x,y) = y - y_{T_1} + \dfrac{1}{k}(x - x_{T_1}) = 0 \\ l_2: f_2(x,y) = y - y_{T_2} + \dfrac{1}{k}(x - x_{T_2}) = 0 \end{cases} \\ \begin{cases} M_1 = \{(x_i, y_i) \mid f_1(x_i, y_i) > 0\} \\ M_2 = \{(x_i, y_i) \mid f_2(x_i, y_i) < 0\} \\ M_3 = \{(x_i, y_i) \mid f_1(x_i, y_i) * f_2(x_i, y_i) < 0\} \end{cases} \\ S_i = M_i \cup S_i, \quad i = 1,2 \\ load[i] = \begin{cases} 1, & i \in L \\ \sum_{s_{ij} \in son_i} load[s_{ij}], & i \in B \end{cases} \\ type(if a_i) = \begin{cases} 1（主线）, & fa_i = bat \\ 2（支线 A）, & load[i] \le 2 \\ 3（支线 B）, & load[i] > 2 \end{cases} \\ dis(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \\ length(k) = \sum_{G(i,j)=1 \text{ 且 } type(i,j)=k} dis(i,j), k = 1,2,3 \\ r[i] = r|fa_i| * (1 - 0.5\%) * (1 - 0.2\%) * (1 - 0.002 * dis(i, fa_i)) \end{cases} \tag{33}$$

5.2.2 模型的求解

问题二提出的基于几何分析的二分类算法的求解步骤如下：

Step1：测试数据集提取

针对前海站和粤海梦站两个数据集，可推测出节点的坐标。

节点编号	x 坐标（m）	y 坐标（m）
前海 25	-25531.816406	-13508.734375
粤海 36	-24602.492188	-11826.172852

Step2：实现基于几何分析的二分类算法

a)根据公式(16)-(22)构建基本几何关系,得到平面点集 $\{P_i \mid i = 1,2,3\}$;

b）选取参数 α 的值为 10，通过**二分法**确定 t 的值，具体过程：

①设左右边界取值为：$L = 0$, $R = 1e2$;

②设 $mid = (L + R)/2$，通过 $check$ 函数判断，当 $t = mid$ 时,$|P_3|$ 与 α 的关系：

$$\begin{cases} \text{若 } |P_3| > \alpha \text{，则调整 } R = mid; \\ \text{若 } |P_3| < \alpha \text{，则调整 } L = mid; \\ \text{若 } |P_3| = \alpha \text{，则 } t = mid \text{ 为所求;} \end{cases}$$

③若 $|P_3| = \alpha$，则算法结束；否则回到②。

c）设置状态向量描述 s，枚举 s 值，通过公式（23）确定集合 S_1、S_2;

d）将问题一中的单供配电网模型封装为函数 SSDN_model(全称 Single Supply and Distribution Network model)，将 S_1、S_2 分别代入 SSDN_model 进行求解最值，更新最低耗费 $cost_{best}$ 以及相关的状态信息 $info_{best}$（分叉点坐标、用户用电可靠性等等），直至完成 s 值的枚举；

e）将最优状态 $info_{best}$ 写入文件’result_t2.xlsx’。

Step3：通过 GraphPlot 的相关 API 将拥有两个电源的最优配电网可视化

5.2.3 结果分析

基于单目标优化的单供配电网模型求解后，得到的配电网如下图所示：

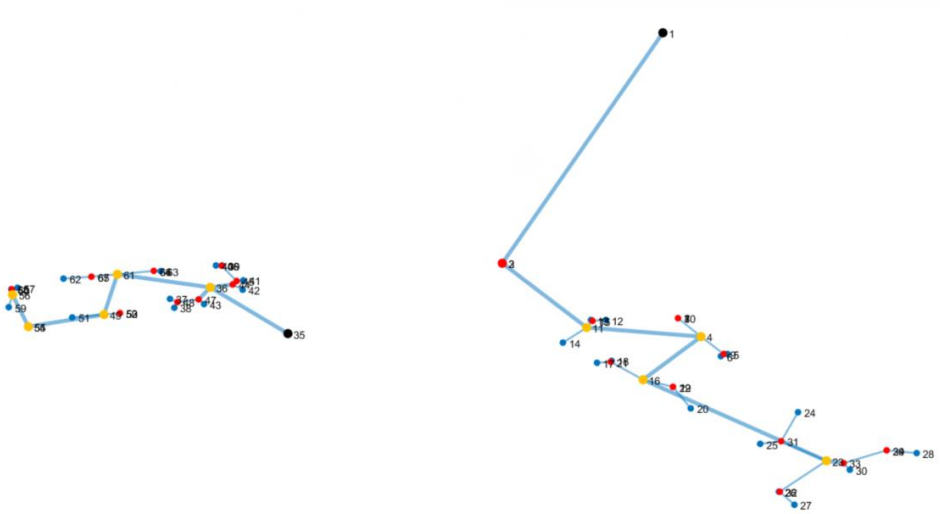

图 8　两个单供电网的分布

（蓝色为负荷，黄色为中心分叉点，红色为分叉点，黑色为电源）

根据求解结果，可以建立出关于用电可靠性的折线统计图，如下图所展示：

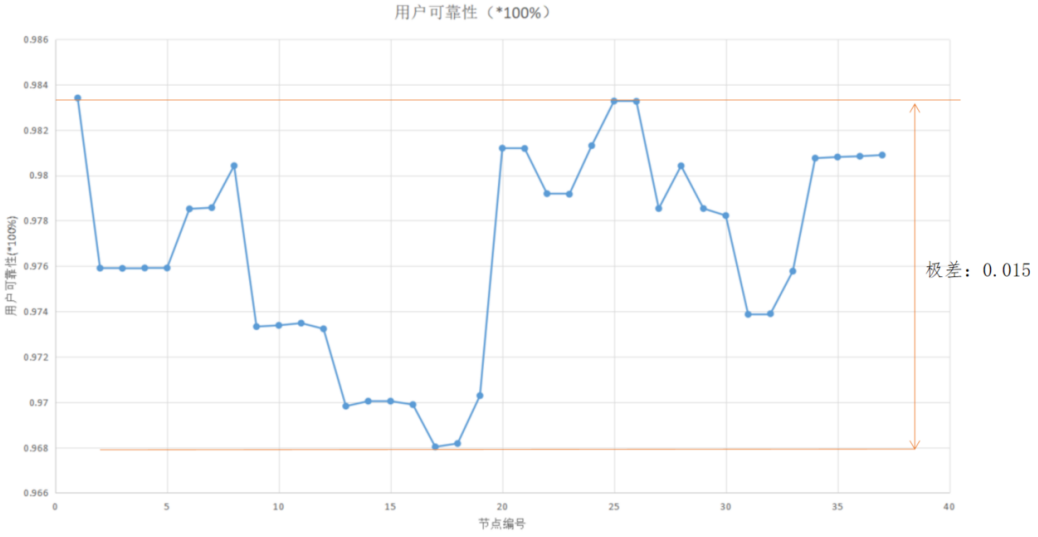

图 9　用电可靠性的折线统计图

可以看出，所有负荷的用电可靠性的大小在 96.8%以上，且进一步求得方差约为 0.00002，可以看出模型求解的精确性。

同时，根据用户可靠性分布趋势绘制分布图，如下所示：

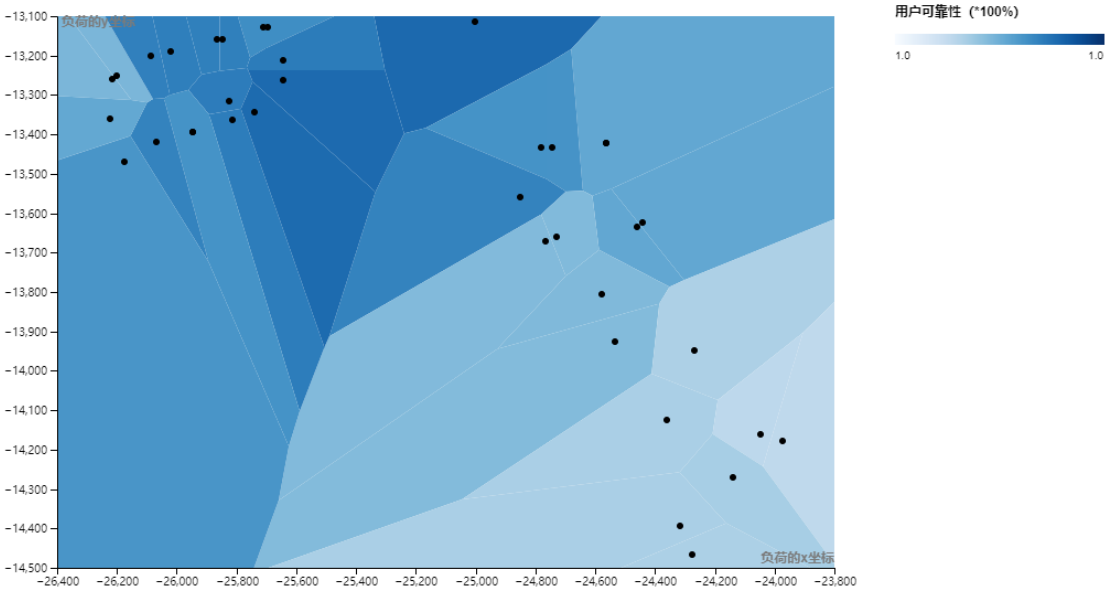

图 10　问题二求解的用电可靠性分布示意图

通过分布示意图，可以看到，离电源越远的地方可靠性相对较低，这也为启发后续问题作了铺垫。

5.3 问题三、四：设计建造总花费上限为 X，网中最低的用电可靠性达到最大的双供配电网络、设计网中最低的用电可靠性下限为 Y%，布线成本最小的双供配电网络

问题三是在问题二的基础上添加联络线产生的。在原有两个单供供电网中添加联络线，并保证功率约束与建造总花费合理的约束的基础上，对网中最低的用电可靠性这一目标函数求取最大值，最终得到用电可靠性最大的双供配电网络。

问题四与问题三类似，都是在问题二的基础上添加联络线产生的。在原有两个单供供电网中添加联络线，并保证功率约束与用户用电最低可靠性在要求的阈值之上，对网中最低建造成本这一目标函数求取最小值，最终得到建造费用最低的双供配电网络。

由于问题三与问题四背景相似，仅在约束条件和目标求解问题上有所不同。因此，本文基于同样的假设建立求解模型，仅在目标函数上有所区别。

5.3.1 模型的建立--双供配电网模型

在（1）（2）问中，以建造费用为唯一优化目标设计的单供网络，具有树状结构，且仅支持单向供电，否则会导致更高的成本。

而双供配电网络，是在前者基础上增加联络线而来，目的是控制成本的情

况下提高网络中用户用电的可靠性。在（3）（4）问中，需要对不同条件约束下的双供网络进行求解，以便最大程度提升在各自网络下的用户用电可靠性。

（1）双供配电网的网络拓扑图

本文假定两个单供网络的地位是等价的，联络线支持双向供电，考虑由此带来工程难度的增加，定义联络线单位造价为主线的 2 倍，且每个方向均需设置开关。

$$cost_{link} = 2cost_{circuit}(1) \tag{34}$$

设联络线的端点为联络点，联络点属于两个单供配电网的分叉点；两个单供网络分叉点集合分别为K_1, K_2，联络线$L_{cont} = (u, v)$，则有$u \in K_1$且$v \in K_2$，即：

$$(u, v) \in \{L_{cont}\}: u \in K_1, v \in K_2 \tag{35}$$

根据前面假设“每个电源只有一条主线出线”，可以进一步抽象出双供配电网络主干部分的拓扑图：首先，每个单供网络仅保留电源、主线以及主线上的分叉点；其次，选择分叉点设置联络点，并在联络点之间建立联络线。

由此，得到双供配电网的网络拓扑图如下：

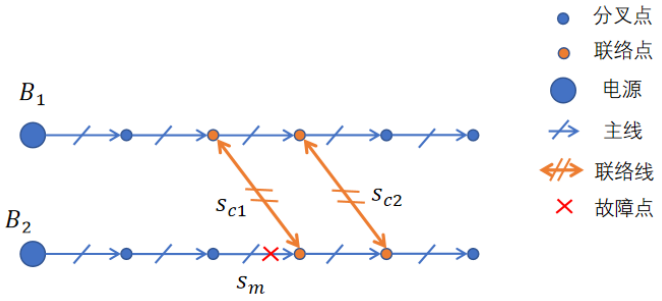

图 11　双供配电网的网络拓扑图

在使用联络线进行扩展供电时，不应出现支线为主线的上一级线路，这要求**联络点必须设置在主线上**，因此仅保留主线上的分叉点。正常情况下，联络线上联络开关s_{c1}，s_{c2}断开。一旦发生主线某部分故障，如上图所示，线路发生故障。此时可行调度方案为（假设电源B_1可扩供所有故障节点）s_m断开，s_{c1}闭合，由电源B_1通过左侧联络线为故障点后所有的用户供电。

而配电网的**支线部分发生故障**，并没有相应的调度方案。这符合我们的一般认知，双供调度是为解决区域内一定规模的用户用电故障问题。对于影响较小的故障，问题排查容易，恢复时间快，调度带来效益不高！

（2）用电可靠性精确概率模型

为论述概率模型的思想，本部分以下图所描述的情况为例进行论述，为简化示意图，省略了线路开关。

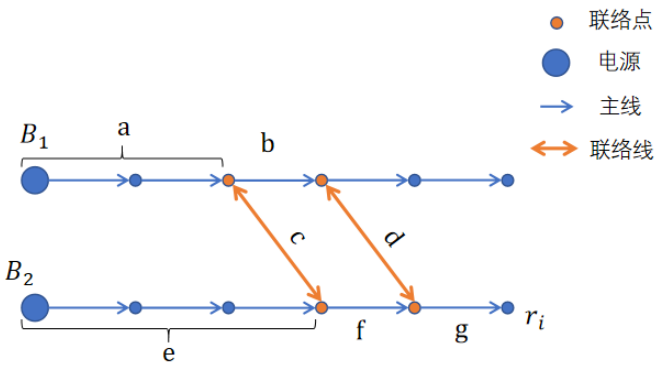

图 12　双供配电网的线路可靠性示意图，省略开关

不同于单供网络，双供网络由于多条联络线，存在多个扩展供电方案。设供电路径$P=(p_1,p_2,..,p_n)$，其中p_i为子路径。用字母$a-g$表示子路径的可靠性（为了方便表达有时可指代路径本身），如字母a表示从电源出发一直到首个联络点（不包含该联络点）之间路径的可靠性。假设目前需求解最右下角节点i的可靠性r_i，观察到存在(a,c,f,g)和(a,b,d,g)两条来自电源B_1扩展供电路径，以及原先单供网络电源B_2的供电路径(e,f,g)，容易得到每条路径的可靠性：

$$r_1=r(a,c,f,g)=acfg$$
$$r_2=r(a,b,d,g)=abdg$$
$$r_3=r(e,f,g)=efg$$

由于路径相互重叠，它们之间的可靠性不是相互独立的，如：

$$r\big((a,c,f,g)\cap(a,b,d,g)\big)\neq r(a,c,f,g)\times r(a,b,d,g) \tag{36}$$

但可以用两条路径包含的所有子路径可靠性之积表达，因为子路径之间是相互独立的。如：

$$r\big((a,c,f,g)\cap(a,b,d,g)\big)=\prod_{p\in(a,c,f,g)\cup(a,b,d,g)}r(p)=acfgbd \tag{37}$$

进一步，可通过容斥原理计算存在(a,c,f,g)和(a,b,d,g)两条供电线路时的节点的可靠性r_{12}：

$$\begin{aligned}r_{12}&=r\big((a,c,f,g)\cup(a,b,d,g)\big)\\&=r(a,c,f,g)+r(a,b,d,g)-r\big((a,c,f,g)\cap(a,b,d,g)\big)\\&=ag\times(bd+cf-bdcf)\end{aligned}$$

同理可得：

$$r_{13}=g\times(abd+ef-abdef)$$

$$r_{23} = fg \times (ac + e - ace)$$

进而可根据 3 元容斥原理计算在 3 条供电路径下的r_i：

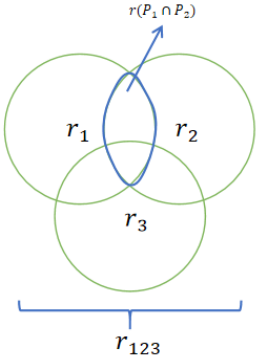

图 13　韦恩图视角下的三元容斥原理

$$r_i = r_{123} = acfg + abdg + efg - acfgbd - acfge - abdgef + abcdefg \quad (38)$$

以上讨论，可进一步拓展至n条供电路径的情况。即，设对于目标供电节点i，存在供电路径集合$P = \{P_1, P_2, \dots, P_n\}$，现求节点$i$的用电可靠性$r_i$。相应地，应用$n$元容斥原理可得：

$$r_i = r(P_1 \cup P_2 \cup \dots \cup P_n) = \sum r(P_i) - \sum r(P_i \cap P_j) + \sum r(P_i \cap P_j \cap P_k) - \cdots + (-1)^{n-1} r(P_1 \cap P_2 \cap \dots \cap P_n) \quad (39)$$

设 $a_i \in \{1,2,\dots,n\}$且$\forall\, i \neq j$，有$a_i \neq a_j$，则$m(m < n)$条供电路径 $\{P_{a_1}, P_{a_2}, \dots, P_{a_m}\}$，同时可靠的概率描述为：

$$r(P_{a_1} \cap P_{a_2} \cap \dots \cap P_{a_m}) = \prod_{p \in \{P_{a_i} | i=1,2,\dots,m\}} r(p) \quad (40)$$

其中 $p \in \{P_{\{a_i\}} | i = 1,2,\dots,m\}$为最小子路径，相互独立。

（3）基于具体调度方案的用电可靠性刻画

根据双供配电网用户供电调度原则第（1）点——满足一个用户全部需求功率，否则断开该用户——以及配电网规划的实际过程，并不是设置联络线后就为**所有故障后可恢复用户**均准备供电调度方案（如在（1）部分最后的调度方案），这是因为一方面受限于电源的可扩展供电量和扩供成本，另一方面有些用户本身用电可靠性较高，失电对于他们是小概率事件，而本问中优化目标为可控成本下提高最低的用电可靠性，因此这些用户并不是首要的优化目标，甚至不是优化目标。

因此，设节点i备有调度方案的供电路径集合$P' = \{P'_1, P'_2, \dots, P'_m\}, |P'| = m$，且满足约束$P' \subseteq P$，则节点$i$的用电可靠性可以刻画为：

$$r_i = r(P'_1 \cup P'_2 \cup \dots \cup P'_m) \tag{41}$$

（4）联络线的设置约束

首先，本文认为联络线的设置至少满足：

〈1〉每个联络点仅和一条联络线相关；

〈2〉联络线之间不存在相互重叠的情况；

否则，设想$|K_1| \times |K_2|$条联络线的情况，如下图所示，此时双供网络之间的联系甚至强于自身网络，联络线的作用已经不仅仅是联络，而是构建了一个新网络。即使是在成本允许的情况下这也不能被允许！

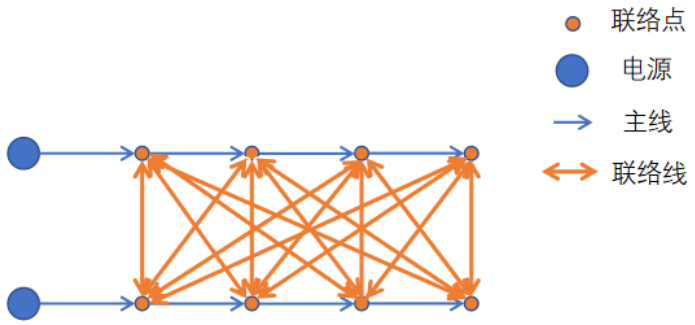

图 14 任意两点间均有联络线的情况示意

对于第〈2〉点，城市中配电电缆往往铺设在地下空间，线路重叠意味着施工深度的增加，与其他建设管道冲突可能性的提升。因此，规划时应当保证联络线不重叠。

而要满足以上约束，需要保证在所有联络线构成的集合$\{L_{cont}\}$中，对于任意两条不同联络线(u_i, v_i)和(u_j, v_j)，$i \neq j$，有$u_i \neq u_j$，$v_i \neq v_j$，即：

$$\{L_{cont}\}: (u_i, v_i) \neq (u_j, v_j) \Longrightarrow u_i \neq u_j,\ v_i \neq v_j \tag{42}$$

设B_1所在网络存在两个联络点K_{1i}和K_{1j}，它们通过联络线相关的、位于B_2所在网络的联络点分别为$K_{2i'}$和$K_{2j'}$，**为表达简洁抽象为下图**。

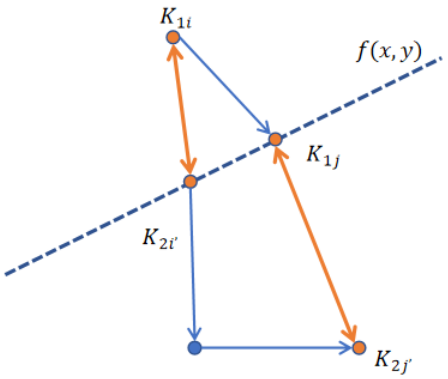

图 15　网络中的联络线几何关系示意

因此联络线为$K_{1i}K_{2i'}$和$K_{1j}K_{2j'}$。设$K_{1j}K_{2i'}$所在直线方程为

$$l_{K_{1j}K_{2i'}}: f(x,y)=0 \tag{43}$$

$$f(x,y)=\frac{y-y_{K_{2i'}}}{y_{K_{1j}}-y_{K_{2i'}}}-\frac{x-x_{K_{2i'}}}{X_{K_{1j}}-x_{K_{2i'}}}=0 \tag{44}$$

为避免联络线重叠，点K_{1i}和$K_{2j'}$应在直线$K_{1j}K_{2i'}$的两侧，存在约束：

$$f\left(x_{K_{1i}},y_{K_{1i}}\right)\times f\left(x_{K_{2j'}},y_{K_{2j'}}\right)<0 \tag{45}$$

（5）调度方案的初步讨论

通过第（4）点的讨论，对整个双供系统的全貌有了更清楚的认识。下面通过一个例子引出调度规则的定义。

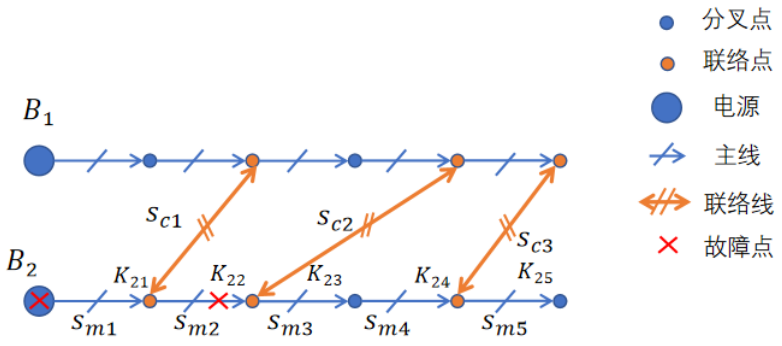

图 16　调度规则示意图

非常不幸地，B_2电源供电网络发生多处故障，连电源本身也需要故障修复，整个单供网络瘫痪！现通过调度，采取B_2电源扩展供电的方式恢复B_1网络。一个可行的调度方案为：

断开S_{m1},S_{m2} → 隔离K_{21}和故障点

闭合$S_{c1} \rightarrow K_{21}$恢复

闭合$S_{c2} \rightarrow K_{22}, K_{23}, K_{24}, K_{25}$恢复

观察到，断开本网络开关S_m，主要为了隔离用户与故障点。因此，当线路故障，断开线路的控制开关；电源故障，断开主线上第一个控制开关；负荷故障，断开负荷前开关。

而闭合联络开关，是为了恢复联络点以及联络点下级线路的供电。调度方案的最后一步可以变为：

闭合$S_{c3} \rightarrow K_{24}, K_{25}$恢复

这样，K_{22}, K_{23}的子网络无论是否存在相应的调度方案，都无法恢复，因为没有物理线路的支持！而且这样调度会使得处于上行电路的联络线使用频率变低。因此，**选择故障点后第一个正常联络点**，由于联络点只与一条联络线相关，实际上选择了一条联络线。闭合联络开关，恢复供电。进而得出，**一个具体调度方案是和所使用联络线绑定的**。

回顾第（2）点中的精确用户可靠性概率模型，对于一个用电节点，当存在多条可靠供电路径时如何选择，以上调度规则给出选择优先级。

（6）调度方案的数学表达

在第（5）点的讨论中发现，调度方案实际上是所有开关（包括联络开关$\{S_c\}$和常闭开关$\{S\}$）的状态集合。但为了后面求解算法表达的简便性，本文选择从用户的角度出发定义调度方案。

第（5）点提及，对于一个网络而言，调度方案和联络线相互绑定，也就是两者一一对应。但是，仅仅说明使用哪条联络线并不能具体描述出整个调度方案，因为调度不一定扩供以联络点为根的整个子树（受扩供功率、成本的约束），换而言之，处于支线的开关状态还未给出。

为此，定义调度方案扩供的负荷集合为$\{W_i'\}$，因此，调度方案D_i定义为：

$$D_i: \left(L_{cont_i}, \{W_i'\}\right) \tag{46}$$

为了表达的简洁性，扩充定义：使用本网络电源正常供电的调度方案为D_0。

根据$\{W_i'\}$可以依据以下原则推出常闭开关状态集合$\{S\}$：若S_i所在线路连接的子树中对于$\forall u$均有$u \notin \{W_i\}$，则断开S_i，否则将其闭合。

举个例子，闭合S_c使用相应联络线进行扩供，但负荷节点A，B，C不在调度集合$\{W_i\}$内，则自下而上地**绿色曲线包裹的区域内**的开关都将断开，若K_{i+1}连接所有负荷也不在$\{W_i\}$内，则断开$S_{m(i+1)}$。

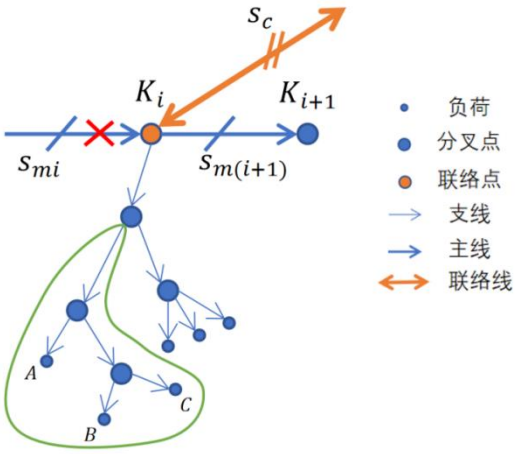

图 17　区域调度举例示意图

（7）电源供电功率的约束

所有调度方案中，必定存在一个用电功率最大的方案。设调度方案集合为$\{D_i\}$，对应扩展用电功率为P_{D_i}，最大扩展用电功率为

$$P_{max} = max(P_{D_i}) \tag{47}$$

而每个网络中负荷需求之和为W_y，由于电源扩展功率的上限为50%，因此电源供电功率约束为

$$P_{D_i} + W_y \le P_{max} + W_y \le 1.2W_y \times 150\% \tag{48}$$

（8）电源功率的期望

在考察实际电源功率P_y，发现在不同调度方案中扩展电源功率并不是一定会产生，而是故障发生后，采取该调度方案后产生的。因此本文通过期望的方式刻画电源功率，并在此基础上考察扩供的花费期望。

定义调度方案D_i的使用概率为$P(D = D_i)$，电源功率的期望描述为：

$$E(P_y) = \sum P_{D_i} \times P(D = D_i) \tag{49}$$

实际上，这里定义的$P(D = D_i)$并不容易得到，而前文一直是从负荷视角描述调度方案，**负荷的可靠性增量是由调度带来的！**已知负荷在单供配电网络中的用电可靠性为r_{0i}，在给定联络线和调度方案下的可靠性为r_i'，显然$r_i' > r_{0i}$，设可靠性增量为$\Delta r = r_i' - r_{0i}$，有

$$\sum P_{D_i,y} \times P(D = D_i, y) = \sum r_{0i,y} \times W_{i,y} + \sum \Delta r_{3-y} \times W_{i,3-y} \tag{50}$$

$$E(P_y) = \sum r_{0i,y} \times W_{i,y} + \sum \Delta r_{3-y} \times W_{i,3-y} \quad (51)$$

进一步，电源扩展功率的期望可以表达为

$$E(P_y') = \begin{cases} E(P_y) - 1.2W_y, E(P) \geq 1.2W_y \\ 0 \qquad\qquad\quad , E(P) < 1.2W_y \end{cases} \quad (52)$$

（9）双供配电网的建造成本

双供配电网系统的总花费由单供网络各自的建造花费$cost_y$，联络线的建造花费$cost_{cont}$，以及扩展功率花费$cost_{pow}$三部分组成，即

$$cost_{sum} = \sum cost_y + cost_{cont} + cost_{pow} \quad (53)$$

第一部分由第（2）问求出，第二部分和其他类型线路的总造价计算方式相同：

$$cost_{cont} = cost_{line} + cost_{switch} \quad (54)$$

定义距离函数$d:(i,j) \to R$，有

$$d(i,j) = \left((x_i - x_j)^2 + (y_i - y_j)^2\right)^{\frac{1}{2}} \quad (55)$$

对于所有联络线$L_{cont}:(u,v)$，有

$$cost_{line} = \sum_{L_{cont}} d(u,v) \times cost_{link} \quad (56)$$

$$cost_{switch} = 2|L_{cont}| \quad (57)$$

对于第三部分，根据第（6）点中的讨论，扩展功率花费为

$$cost_{pow} = \sum_y E(P_y') \times cost_{p,y} \quad (58)$$

（10）双供配电网模型

根据问题条件构建目标函数和相应约束条件，

问题三：总花费上限为X，通过建造联络线，和设计合适的调度方案使得最低的用电可靠性最大，即

$$max_{\{L_{cont}\},\{D_i\}}(min(r_i)) \quad (59)$$

$$s.t.\, cost_{sum} \leq X \quad (60)$$

问题四：以最低的总费用建造联络线，和设计合适的调度方案使得最低的用电可靠性不低于$Y\%$，即

$$min_{\{L_{cont}\},\{D_i\}}(cost_{sum}) \quad (61)$$

$$s.t.\, min(r_i) \geq Y\% \quad (62)$$

其他相同约束如下：

$$
s.t.\begin{cases}
cost_{link}=2cost_{circuit}(1)\\
(u,v)\in\{L_{cont}\}:u\in K_1,v\in K_2\\
r(P_1\cup P_2\cup\ldots\cup P_n)=\sum r(P_i)-\sum r(P_i\cap P_j)+\ldots+(-1)^{n-1}r(P_1\cap P_2\cap\ldots\cap P_n)\\
r_i=r(P_1'\cup P_2'\cup\ldots\cup P_m')\\
\{L_{cont}\}:(u_i,v_i)\neq(u_j,v_j)\Longrightarrow u_i\neq u_j\ ,\quad v_i\neq v_j\\
f(x,y)=\frac{y-y_{K_{2i'}}}{y_{K_{1j}}-y_{K_{2i'}}}-\frac{x-x_{K_{2i'}}}{X_{K_{1j}}-x_{K_{2i'}}}=0\\
f(x_{K_{1i}},y_{K_{1i}})\times f(x_{K_{2j'}},y_{K_{2j'}})<0\\
P_{D_i}+W_y\leq 1.2W_y\times 150\%\\
E(P_y)=\sum r_{0i,y}\times W_{i,y}+\sum\Delta r_{3-y}\times W_{i,3-y}\\
E(P_y')=\begin{cases}E(P_y)-1.2W_y, & E(P)\geq 1.2W_y\\ 0, & E(P)<1.2W_y\end{cases}\\
cost_{sum}=\sum cost_y+cost_{cont}+cost_{pow}\\
cost_{cont}=cost_{line}+cost_{switch}\\
d(i,j)=((x_i-x_j)^2+(y_i-y_j)^2)^{\frac{1}{2}}\\
cost_{line}=\sum_{L_{cont}}d(u,v)\times cost_{link}\\
cost_{switch}=2|L_{cont}|\\
cost_{pow}=\sum_y E(P_y')\times cost_{p,y}
\end{cases}
\tag{63}
$$

5.3.2 模型的求解——双供配电网模型

模型的建立部分详细讨论了决策变量包括联络线$\{L_{cont}\}$的设置以及调度方案$\{D_i\}$和优化目标之间的关系，下面本文将给出如何求解决策变量的思路及方法。

（1）联络线的决策空间

模型建立第（4）点，在讨论联络线约束时设想了一种“全链接”的极端情况，这实际上是联络线在无约束情况下的决策空间的一个解。定义函数$l:K_1\times K_2\to L$，该决策空间的数学表达为$L=\{l(u,v)|u\in K_1,v\in K_2\}$，即$L_{cont}\in L$，其

大小为$|L| = 2^{|K_1|\times|K_2|}$.

联络线的设置集合满足$\{L_{cont}\} \subset L$，进一步，考虑联络线的设置约束<1>，这个约束排除了类似“全链接”的情况，使得$|\{L_{cont}\}| \le min(|K_1|,|K_2|)$，约束<1>的决策空间设为$L_1$，它可以通过组合方法求解。

设有i条联络线，满足$i \in [1, min(|K_1|,|K_2|)]$，因此在$K_1$，$K_2$中分别选择$i$个联络点，共$C^i_{|K_1|} \times C^i_{|K_2|}$种情况，$i$对联络点之间相互匹配共$i!$种情况，因此

$$|L_1| = \sum_{i=1}^{min(|K_1|,|K_2|)} C^i_{|K_1|} \times C^i_{|K_2|} \times i! \qquad (64)$$

这里取$|K| = |K_1| = |K_2|$，研究$|L_1|$与$|K|$之间关系，发现$|L_1|$随$|K|$增加呈指数趋势增加，$|K| = 13$时，$|L_1| \approx 6.7e7$；$|K| = 14$时，$|L_1| \approx 2.8e8$.

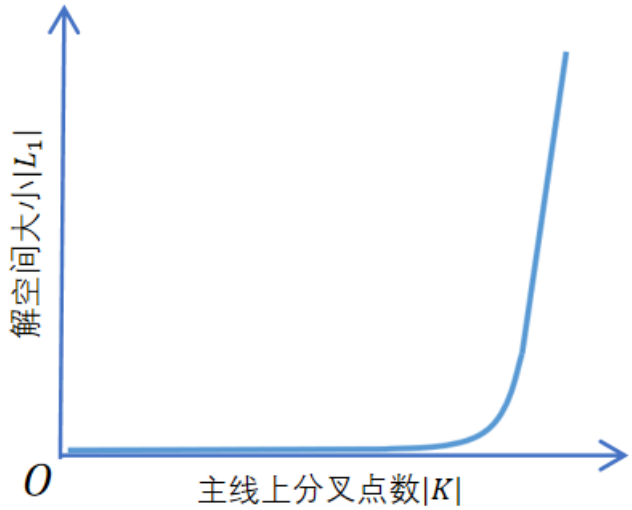

图 18　解空间大小随着主线上分叉点数的变化函数趋势图

更进一步，考虑约束<1><2>的下决策空间L_2，由于约束<2>是对分叉点具体坐标关系的限制，该限制强弱难以度量，本文考虑从L_1出发，通过约束<2>排除部分决策得到L_2. 而对于决策空间L_1，在$|K| \le 10$时，可以以遍历该空间的方式获取联络线的设置集合$\{L_{cont}\}$，在此基础上求模型最优解，但是当$|K| > 10$时，遍历的时间成本大大增加！

（2）基于启发式准则的贪心算法

既然决策空间至少是$O(2^{|K|})$，无法短时间内遍历，同时，单供网络的几何特征具有不确定性。本文提出基于启发式准则的贪心算法，在面对大规模问题快速求得近似最优解。

A. 算法步骤：

①初始化$\{L_{cont}\}=\emptyset$；

②对于$\forall(u,v)\in L$，计算$d_{uv}=d(u,v)$，定义矩阵$D=[d_{uv}]_{|K_1|\times|K_2|}$；

对于联络点u，定义u可扩供的负荷数为n_u.

③对于$\forall(u,v)\in L$，计算$n_{uv}=n_u+n_v$，定义矩阵$N=[n_{uv}]_{|K_1|\times|K_2|}$，以及$n''_{uv}=|n_u-n_v|$，定义矩阵$N'=[n'_{uv}]_{|K_1|\times|K_2|}$

④求取$i^*,j^*=arg\ min\ f(\lambda,D_{ij},N_{ij})$，**其中$\boldsymbol{f(\lambda,D_{ij},N_{ij})}$为启发式准则**，λ为协调变量；

⑤若对于$\forall(u,v)\in\{L_{cont}\}$，$(u,v)$和$(i^*,j^*)$均遵循约束<2>，则$\{L_{cont}\}=\{L_{cont}\}\cup(i^*,j^*)$；

⑥$D_{i^*j^*}=\infty$，$N_{i^*j^*}=\infty$；

⑦若$D=[\infty]_{|K_1|\times|K_2|}$，$N=[\infty]_{|K_1|\times|K_2|}$，算法结束，返回联络线设置集合$\{L_{cont}\}$；否则转到④.

B. 启发式准则：

【1】最优解具有在一定成本下尽可能提升尽可能多的负荷用电可靠性的特征，$\frac{N_{ij}}{D_{ij}}$尽可能大，即：

$$max\left(\frac{N_{ij}}{D_{ij}}-c\right)$$

$$\Leftrightarrow max(N_{ij}-c\times D_{ij})$$

$$\Leftrightarrow min(-N_{ij}+c\times D_{ij})$$

$$\Leftrightarrow min\left(D_{ij}-\frac{1}{c}N_{ij}\right)$$

令$\lambda=\frac{1}{c}$，则$max\left(\frac{N_{ij}}{D_{ij}}-c\right)\Leftrightarrow min(D_{ij}-\lambda N_{ij})$.

【2】最优解中，对于联络线(u,v)相关的联络点u和v具有一定的对等性，表现为u和v可扩供的负荷数之差$|n_u-n_v|$较小，即：$min(N'_{ij})$。

因此，定义启发式准则为：

$$f(\lambda, D_{ij}, N_{ij}) = D_{ij} - \lambda_1 N_{ij} + \lambda_2 N'_{ij} \quad (65)$$

其中，λ_1和λ_2为协调变量，用于协调D，N以及N'之间的量纲关系以及权重分配。

（3）负荷分块改进算法

在抽象双供系统网络拓扑的过程中，本文将主线上分叉点K_i的支线所连接的子树部分省略（下图绿色曲线包裹部分），原因是对于任意负荷点A_i，它和主线上分叉点K_i的用电可靠性之比为一个固定常数c_i，不会因增设联络线而改变，即$r_{A_i} = c_i \times r_{K_i}$. 因此研究时只关注$K_i$，或者说将绿色曲线包裹区域内的负荷视为一个点，用该点（这里是K_i）代表该区域内的负荷点，这是一种分块的思想方法。

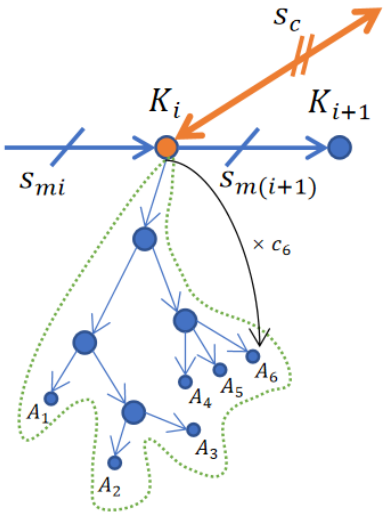

图 19　可以将绿色区域内的负荷等效为一个负荷点

考虑负荷分块，是因为在贪心方法中可能出现下图情形导致无法增加更多的联络线。拓扑图加入坐标信息，两个单供网络的**主线往相反方向延伸**。运行（2）中算法，步骤⑤第一个加入$\{L_{cont}\}$的联络线为(K_1, K_2)，其中$K_1, K_2 = arg\,min(D_{ij} - \lambda_1 N_{ij} + \lambda_2 N'_{ij})$，之后产生的联络线$(u, v)$和$(K_1, k_2)$均不能满足约束<2>，因为$(K_1, K_2)$恰好将两个单供网络剩余分叉点分割为 2 个集合。设为$S = \{u \in \{K_{B_1}\}/\{K_1\}\}$以及$T = \{v \in \{K_{B_2}\}/\{K_2\}\}$，如图，将不会有新联络线$(K_i, K_j)$被应用！

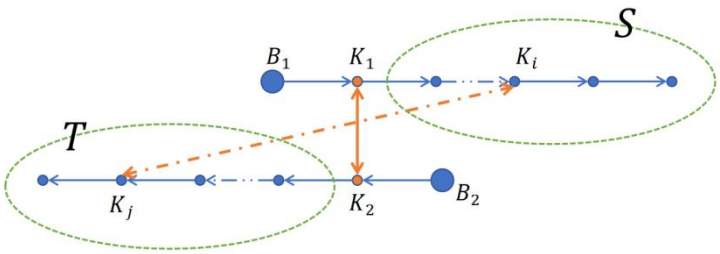

图 20　负荷分块示意图

这实际上归因于贪心算法并未考虑约束<2>带来的影响，因此，本文提出以下**负荷分块改进算法**：

①对主线上连续k个分叉点K_j，K_{j+1}，…，K_{j+k}，构造一个虚拟点K_i'，其中$x_{K_i'}=\frac{1}{k}\sum_j^k x_{K_j}$，$y_{K_i'}=\frac{1}{k}\sum_j^k y_{K_j}$，设该双供系统虚拟点的集合为$\{K_{i',y}\}$，$y\in\{1,2\}$代表两个单供网络，$\{K_{i',y}\}$和电源$B_1$，$B_2$构成一个虚拟双供系统；

②求该虚拟系统联络线设置集合$\{L_{cont}\}^*$，满足$|\{L_{cont}\}^*|=max(|\{L_{cont}\}|)$；

③对于$\forall(K_i',K_j')\in\{L_{cont}\}^*$，对$K_i'$对应的真实分叉点集合$\{K_i,K_{i+1},\dots,K_{i+k}\}$以及$K_j'$对应的分叉点集合$\{K_j,K_{j+1},\dots,K_{j+k}\}$应用基于启发式准则的贪心算法。

该算法保证了联络线的数量，因为$\forall(K_i',K_j')$之间满足约束<2>，对应真实分叉点集合至少存在一对点满足约束<2>，最终联络线设置集合满足$|\{L_{cont}\}|\geq|\{L_{cont}\}^*|$.

对上述例子应用该算法，取$k=2$：

①构造虚拟点K_i'：

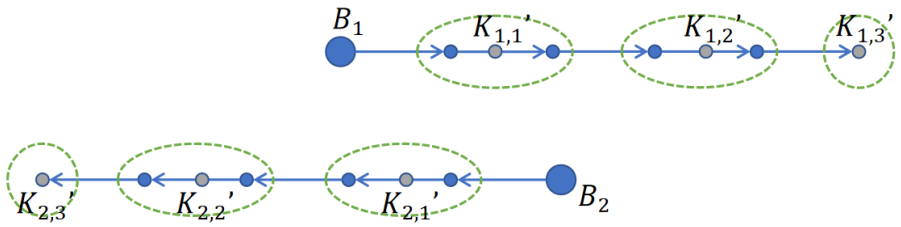

图 21　构造虚拟点K_i'

②求虚拟系统联络线设置集合$\{L_{cont}\}^*$：

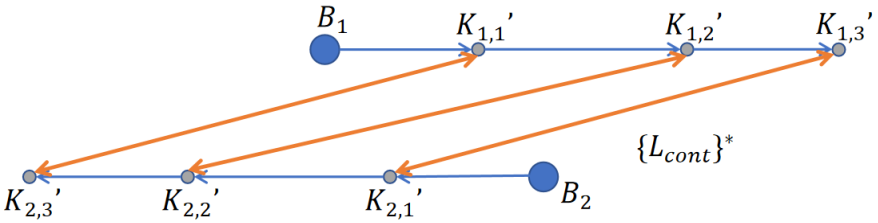

图 22　联络线集合示意图

③对K_i'和K_j'对应的真实分叉点集合应用贪心算法：

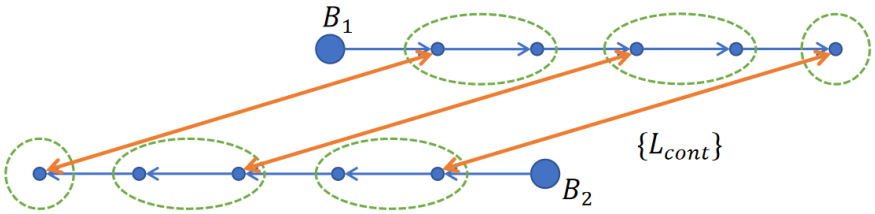

图 23　使用贪心算法示意图

（4）两级最小堆求解调度方案$\{D_i\}$

在联络线设置集合$\{L_{cont}\}$确定后，我们考虑求解调度方案$\{D_i\}$. 问题三要求求解$\{D_i\}$使得最低的用电可靠性最大，问题四要求使得可靠性不低于$Y\%$，最终的调度方案$\{D_i\}$是在约束条件下逐步优化而来，而两问中每一步优化可以均是优化用电可靠性最低的负荷。动态维护最小值，容易想到堆结构，针对问题特点，本文提出基于两级最小堆的方案：

①对主线上每个分叉点K_i维护最小堆$heap[K_i]$，存储K_i支线所连接子树中所有负荷点，堆以负荷用电可靠性为键建立，访问可获取负荷相关编号等相关信息，通过结构体重载运算符实现；

②将所有分叉点最小堆$heap[K_i]$的堆顶元素$heap[K_i].top$加入到全局最小堆$heap_{global}$中，并增加堆号标识K_i；

③单步更新方案：获取$heap_{global}.top$，弹出，访问获取对应堆号标识K_i，推知$heap_{global}.top == heap[K_i].top$，弹出，为负荷增加扩供线路，优化其用电可靠性，更新节点为$node_{new}$，将$node_{new}$加入$heap[K_i]$，将$heap[K_i]$加入$heap_{global}$，同时以差量方式更新电源功率：

$$E(P_y) = E(P_y) - E(node_{old}) + E(node_{new}) \tag{66}$$

根据公式（52）（58）进一步更新$cost_{pow}$.

④算法结束条件：调度方案饱和或者下一步优化调度不满足电源供电功率$1.2W_y \times 150\%$的约束.

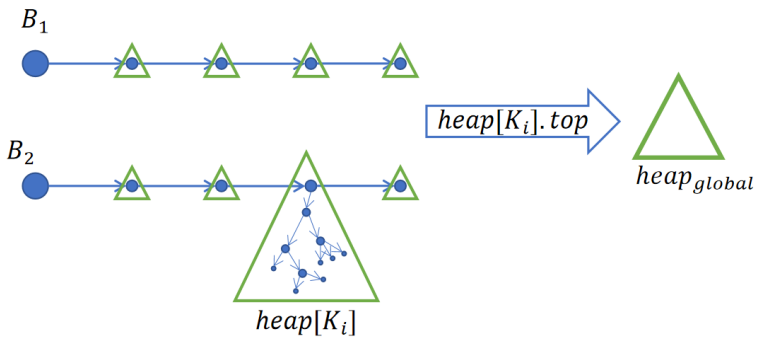

图 24　两级最小堆原理示意图

该方案的现实意义在于对整个双供系统进行层次分级，设立不同层次的中心点收集局部信息，通过中心点将信息逐层向上汇聚，最终获得全局信息的摘要，以进行全局规划。

该方案对最小值单步更新没有明显影响，时间复杂度仍为$O(log(\sum|K_i|))$，实际上堆中的每棵子树均满足根节点键最大的性质，或者说每棵子树均是一个子堆，而该方案将一个堆拆为多个子堆，并没有改变堆的结构，自然对操作的时间复杂性没有影响。

（5）问题三的求解

约束条件转化为

$$cost_{cont} + cost_{pow} = cost_{sum} - \sum cost_y \le X - \sum cost_y \tag{67}$$

设联络线建设花费与扩供花费比例为$\frac{\gamma}{1-\gamma}$，可得

$$\begin{cases} cost_{cont} \le \gamma \times (X - \sum cost_y) \\ cost_{pow} \le (1-\gamma) \times (X - \sum cost_y) \end{cases} \tag{68}$$

以$\Delta\gamma = 0.02$单步长遍历γ，应用负荷改进后的贪心算法求解联络线设置集合$\{L_{cont}\}$，对每次更新$\{L_{cont}\} = \{L_{cont}\} \cup (i^*, j^*)$，同时更新$cost_{cont} = cost_{cont} + cost(i^*, j^*)$，增加算法结束条件：

$$cost_{cont} + cost(i^*, j^*) > \gamma \times \left(X - \sum cost_y\right) \tag{69}$$

应用两级最小堆求解调度方案$\{D_i\}$，每步优化同时更新：

$$cost_{pow} = cost_{pow} + (\Delta r \times W_i) \times cost_{p,y} \quad (70)$$

增加算法结束条件为：

$$cost_{pow} + \Delta cost > (1-\gamma) \times (X - \sum cost_y) \quad (71)$$

最终结果取：

$$max_{\gamma}(maxmin(r_i)) \quad (72)$$

（6）问题四的求解

应用负荷改进后的贪心算法，对每次更新得到的$\{L_{cont}\}$，应用两级最小堆求解调度方案$\{D_i\}$，增加算法结束条件为单步更新的优化目标$node$满足

$$r(node) \geq Y\% \quad (73)$$

最终结果取：

$$min_{\{L_{cont}\},\{D_i\}}(cost_{cont} + cost_{pow}) \quad (74)$$

5.3.3 结果分析

（1）联络线设置集合$\{L_{cont}\}$

在第 2 题结果的基础上，取分块大小$k = 3$运行负荷改进后的贪心算法，算法依次建立联络线(2,8)，(4,9)，(5,10)，如下图：

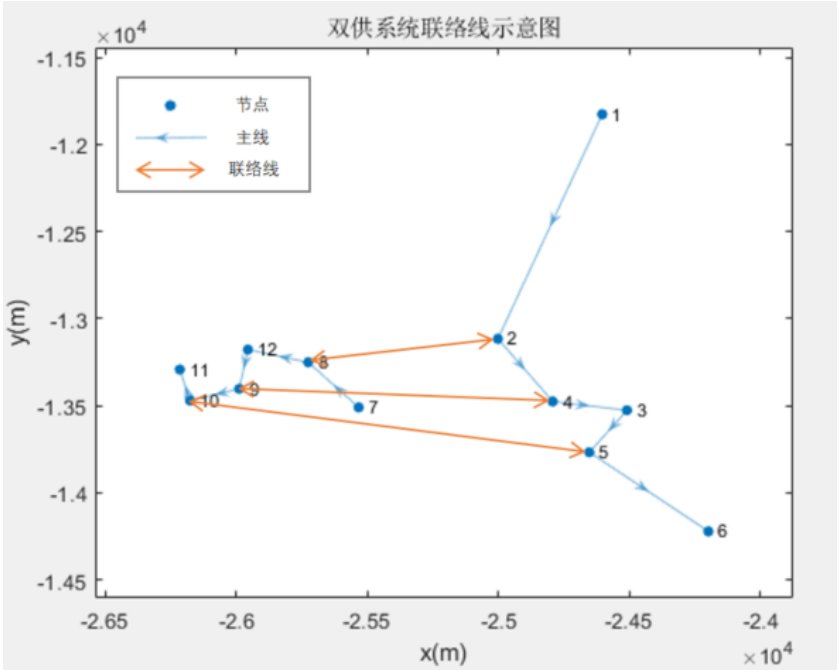

图 25　三四问结果示意图

可以看到，联络线设置集合$\{L_{cont}\}=\{(2,8),(4,9),(5,10)\}$将两个单供网络紧紧联系，同时满足联络线设置的约束。算法运行过程中联络线设置集合$\{L_{cont}\}$以及相应建设花费$cost_{cont}$变化如下表，可以观察到建设花费增量δ在不断增加，符合贪心原则的结果。

联络线设置集合$\{L_{cont}\}$	建设花费$cost_{cont}$（千元）	增量δ（千元）
$\{\varnothing\}$	0	0
$\{(2,8)\}$	593.117404	593.117404
$\{(2,8),(4,9)\}$	1487.211164	894.09376
$\{(2,8),(4,9),(5,10)\}$	2611.857373	1124.646209

记上表中建设花费一列为$cost_{cont}=[594;1488;2612]$，问题三对于给定$X$，可令$\gamma\times\left(X-\sum cost_y\right)=cost_{cont}(i),i=0,1,2,3$，进行下面调度方案$\{D_i\}$的求解.

（2）**调度方案**$\{D_i\}$

运行两级最小堆方案，进行多轮优化，结果如下表。在弱化约束条件的基础上，每轮优化的目标负荷所在分块由于局部性集中在主线的末端，而每次优化带来的可靠性增量在$0.1\%-0.5\%$内，最低可靠性在$96\%-97\%$的水平。

优化轮数(t)	优化负荷所在分块(Ki)	优化前的可靠性(×100%)	优化后的可靠性(×100%)	可靠性增量δ(×100%)
1	6	0.968026	0.969213	0.001187
2	6	0.968174	0.969361	0.001187
3	6	0.969213	0.971051	0.001838
4	6	0.969361	0.971199	0.001838
5	6	0.969821	0.97101	0.001189
6	6	0.969889	0.971078	0.001189
7	6	0.970041	0.97123	0.001189
8	6	0.970042	0.971231	0.001189
9	6	0.970285	0.971475	0.00119
10	6	0.97101	0.97285	0.00184
11	6	0.971051	0.975321	0.00427
12	6	0.971078	0.972919	0.001841
13	6	0.971199	0.975469	0.00427
14	6	0.97123	0.973071	0.001841
15	6	0.971231	0.973072	0.001841
16	6	0.971475	0.973316	0.001841
17	6	0.97285	0.977128	0.004278

18	6	0.972919	0.977197	0.004278
19	6	0.973071	0.97735	0.004279
20	6	0.973072	0.977351	0.004279
21	5	0.973231	0.974424	0.001193
22	6	0.973316	0.977596	0.00428
23	5	0.973329	0.974522	0.001193
24	5	0.973383	0.974576	0.001193
25	5	0.973481	0.974675	0.001194
26	5	0.974424	0.976271	0.001847
27	5	0.974522	0.976369	0.001847
28	5	0.974576	0.976424	0.001848
29	5	0.974675	0.976522	0.001847

根据上表，绘制优化后的最低可靠性折线图，可以看到，用户用电可靠性在每一轮优化之后都有得到提升，且从原先的 96.8%经过了 29 轮次的优化后最终提升至了 97.7%。

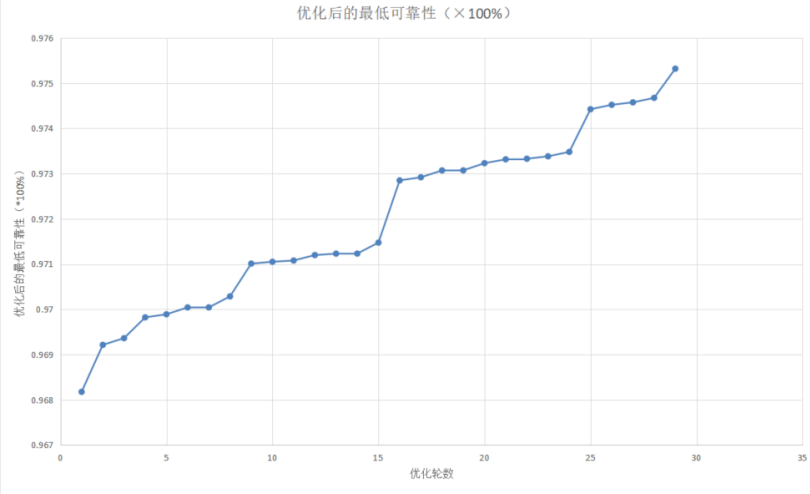

图 26　最低可靠性的提升折线

绘制优化前后的用电可靠性分布梯度图（左图为优化前，右图为优化后），可以看到，在右下角区域的用户可靠性得到了提升，使得整体用电可靠性趋于均匀。

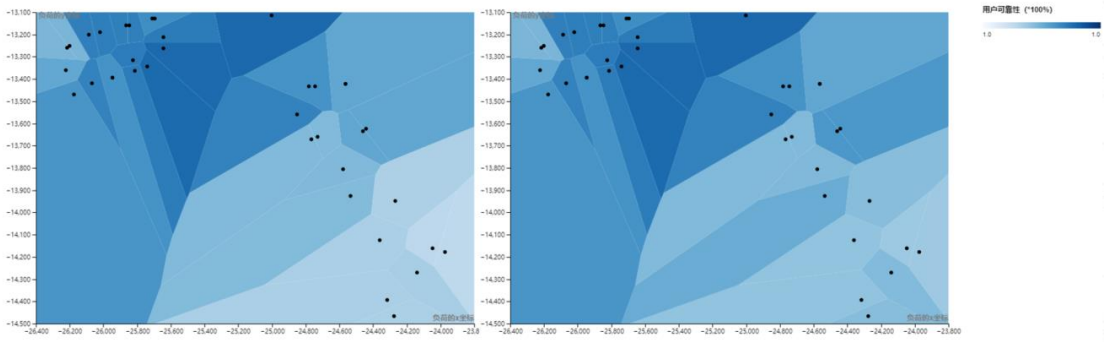

图 27　优化前后的可靠性梯度分布图

六、模型的评价与优化

6.1 模型的优点

1、本文基于 K-means 聚类方法，结合最小生成树的优良拓扑性质，别出心裁地对分叉点进行了分层次的求解方法，能够良好地契合问题目标，进行建模求解。

2、在一定的假设条件下，本文精确地刻画了生活中的用电可靠性，符合实际情况。

6.2 模型的缺点

1、由于专业知识的不足以及模型假设条件导致的简化，本文建立的模型无法完全模拟生活中配电网建造时所要考虑的各种情况。

2、问题三、四中使用期望的方法来刻画电网功率，在一定程度上存在数据误差。

6.3 模型的优化及推广

1、模型可深入结合现实生活，对限制条件进行更为精细的描述，并且对功率的描述进行进一步优化后，可以良好地模拟现实生活中电网建造时候的拓扑情况，且获得的可靠性亦可作为一定的价值参考。

2、模型同样可以推广于生活中的管道建设、路径规划等问题，具有较好的普适性。

七、参考文献

[1]王聃，孙哲军，张敬思，周刚.基于供电可靠性的配电网规划[J].设计应用（电力），2021（12）：59-63.

[2]张宗东，杨立.基于供电可靠性的配电网规划实践浅谈[J].教育论坛，2017（6）：339.

[3]王聃，陆渊超，张敬思，周刚.基于改进最小生成树的配网线路优化[J].设计应用（工业控制），2022（1）：46-49.

八、附录

在附录部分，展示模型求解时所需要的代码及求解的部分数据表格。

负荷聚类后所属的簇	负荷的 x 坐标	负荷的 y 坐标	用户可靠性（*100%）
A	-25000.44727	-13115.21582	0.983408301
A	-24442.59961	-13622.91016	0.975906742
A	-24458.83594	-13635.16699	0.975897031
A	-24565.08594	-13422.53516	0.975913222
A	-24565.08594	-13422.53516	0.975913222
A	-24743.37695	-13432.51563	0.978519156
A	-24780.7793	-13432.51563	0.978572792
A	-24850.33789	-13559.50391	0.980422872
A	-24765.7207	-13672.08789	0.973328585
A	-24729.53711	-13660.99219	0.973383162
A	-24577.45898	-13806.47949	0.973481077
A	-24533.52344	-13927.25098	0.973230863
A	-24267.62891	-13948.49316	0.969820557
A	-24361.44336	-14125.56152	0.97004088
A	-24315.20898	-14393.13184	0.970041665
A	-24276.59766	-14466.30566	0.969888626
A	-23973.41602	-14177.28027	0.968026428
A	-24048.05859	-14161.86035	0.968174014
A	-24139.21094	-14270.57422	0.970285105
B	-25824.13867	-13314.61426	0.981194498
B	-25813.4082	-13364.19629	0.981181418
B	-25694.75195	-13127.44824	0.979190342
B	-25710.1875	-13127.59863	0.979166769
B	-25642.39648	-13211.29981	0.981307357
B	-25643.90625	-13263.61231	0.983273162
B	-25739.65625	-13343.95898	0.983260573
B	-25947.96289	-13394.10938	0.978535604
B	-26066.48828	-13418.3125	0.980417516
B	-25947.96289	-13394.10938	0.978535604
B	-26175.04688	-13469.98438	0.978224147
B	-26201.95508	-13252.4541	0.973868139
B	-26217.38477	-13260.64258	0.973897028
B	-26223.77539	-13361.33301	0.975773222
B	-26088.04883	-13201.42383	0.980755181
B	-25847.06055	-13159.47363	0.980806502
B	-25864.07422	-13158.0127	0.98084
B	-26018.85547	-13190.59961	0.980892574

表 1 问题二 K-means 聚类后的用户可靠性

check_25.m

```
clear,clc;
data=importdata('Ç°25.xlsx');

ID = data.data;
data.textdata(1,:) = [];
name = data.textdata(:,4);
type = data.textdata(:,5);
content = data.textdata(:,6);
front = data.textdata(:,7);
back = data.textdata(:,8);
XY = data.textdata(:,9);
size_all = length(ID);

xy = cell(size_all,1);
line = [];
swt = [];
load = [];
load_c = [];
for i = 1:size_all
    tuple = strsplit(XY{i},'/');
    s = length(tuple);
    for j = 1:(s-1)
        p = strsplit(tuple{j});
        px = str2num(p{1});
        py = str2num(p{2});

        xy{i} = [xy{i};px,py];
    end
end

for i = 1:size_all
    if strcmp(type{i}, 'µ¼Ïß')
        line = [line; ID(i)];
        plot(xy{i}(:,1),xy{i}(:,2),'color',[1 0 1]);
        hold on;
        elseif strcmp(type{i},'¿ª¹Ø')
            swt = [swt;ID(i) ];
            plot(xy{i}(1),xy{i}(2),'+','Markersize',10,'color',[1
0.5 1]);

        elseif strcmp(type{i},'±äÑ¹Æ÷')
```

```
            plot(xy{i}(1),xy{i}(2),'.','Markersize',15,'color',[1
0.7 1]);
            load = [load;xy{i}(1),xy{i}(2)];
            c = str2num(content{i}(1:6));
            load_c = [load_c;c]

    end

end
plot(-25531.816406,-13508.734375,'o','Markersize',20,'color',[0
0 0]);
xlswrite('load_demo.xlsx',[load,load_c],'qh');


%indexOf(ID,102)£»


function index = indexOf(ID,id)
index = find(ID==id);
end
```

check_36.m

```
clear,clc;
data=importdata('ÔÁ36.xlsx');

ID = data.data;
data.textdata(1,:) = [];
name = data.textdata(:,4);
type = data.textdata(:,5);
content = data.textdata(:,6);
front = data.textdata(:,7);
back = data.textdata(:,8);
XY = data.textdata(:,9);
size_all = length(ID);

xy = cell(size_all,1);
line = [];
swt = [];
load = [];
load_c = [];
```

```
for i = 1:size_all
    tuple = strsplit(XY{i},'/');
    s = length(tuple);
    for j = 1:(s-1)
        p = strsplit(tuple{j});
        px = str2num(p{1});
        py = str2num(p{2});

        xy{i} = [xy{i};px,py];
    end
end
content
for i = 1:size_all
    if strcmp(type{i}, 'µ¼Ïß')
        line = [line; ID(i)];
        plot(xy{i}(:,1),xy{i}(:,2),'color',[0 1 1],'lineStyle','-
-');
        hold on;
        elseif strcmp(type{i},'¿ª¹Ø')
            swt = [swt;ID(i) ];

plot(xy{i}(1),xy{i}(2),'+','Markersize',10,'color',[0.5 1 1]);

        elseif strcmp(type{i},'±äÑ¹Æ÷')

plot(xy{i}(1),xy{i}(2),'.','Markersize',15,'color',[0.8 1 1]);
            load = [load;xy{i}(1),xy{i}(2)];
            c = str2num(content{i}(1:6));
            load_c = [load_c;c]



    end

end
plot(-24602.492188,-11826.172852,'o','Markersize',20,'color',[0
0 0]);

xlswrite('load_demo.xlsx',[load,load_c],'yh');
%indexOf(ID,102)£»


function index = indexOf(ID,id)
index = find(ID==id);
```

```
end
```

constructive_algorithm.m

```
% 构造性算法实现分叉点构造、线路选型
% 输入：G——最小生成树的邻接矩阵
% (P_x0, P_y0)——负荷坐标
% 输出：G_2——配电网拓扑的邻接矩阵
% (P_i,P_j)——所有节点的坐标
function [G_2, P_i, P_j] = constructive_algorithm(G, P_x0, P_y0)
    global G_1  % 最小生成树的邻接矩阵
    global G_2  % 配电网拓扑的邻接矩阵
    global P_i  % 所有节点的坐标
    global P_j
    global n1   % 负荷数+1
    global n2   % 分叉点数+1
    global cnt  % 计数器，为分叉点编号
    global deg  % 点度数
    global pair % 分叉点与负荷的映射关系
    global Load % 子树的负荷数

    G_1 = G;
    deg = zeros(1, n1);
    n2 = cnt_bran_node();       % 计数分叉点

    % 节点总数为(n1+n2-1)
    G_2 = zeros(n1 + n2 - 1);
    P_i = zeros(1, n1 + n2 - 1);
    P_j = zeros(1, n1 + n2 - 1);
    P_i(1 : n1) = P_x0(1 : n1);
    P_j(1 : n1) = P_y0(1 : n1);

    cnt = n1;
    pair = zeros(1, n1);

    for i = 1 : n1
        for j = i + 1 : n1
            G_2(i, j) = G_1(i, j);
        end
    end
    G_2 = G_2 + G_2';
```

```
    % 构造分叉点
    construct_cross_node();
    Load = zeros(1, n1 + n2 - 1);
    % 通过负载数确定线的类型
    judge_line_type();
    % 假设与电源直接相连的为主线
%    set_main_line();

%     % 画图
%     figure
%     G_n = graph(G_2);
%     col = zeros(n1 + n2 - 1, 3);
%     for i = 1 : n1
%         col(i, :) = [0 0.4470 0.7410];
%     end
%     for i = n1 + 1 : n1 + n2 -1
%         col(i, :) = [1 0 0];
%     end
%     h = plot(G_n, 'NodeColor', col);  % 返回一个 Graphplot 对象
%     % 设置节点位置
%     h.XData = P_i;
%     h.YData = P_j;
%     % 设置节点属性
%
%     % h.NodeCData = col;
%     % 设置线型
%     h.LineWidth = 1.5 * G_n.Edges.Weight;
end

% 计数度数大于 1 的点
function cnt = cnt_bran_node()
    global G_1
    global n1
    global deg
    for i = 1 : n1
        deg(i) = sum(G_1(i, :));
    end
    bran_node = find(deg(:) > 1);
    cnt = size(bran_node, 1);
end

function [] = construct_cross_node()
    dfs(1, 0);
```

```
end

function [] = judge_line_type()
    dfs1(1, 0);
end

function [] = dfs(u, fa)
    global G_1
    global G_2
    global n1
    global cnt
    global P_i
    global P_j
    global deg
    global pair

    if deg(u) > 1 && u ~= 1
        % 构造一个新分叉点
        cnt = cnt + 1;
        P_i(cnt) = P_i(u) + rand * 5;
        P_j(cnt) = P_j(u) + rand * 5;
        G_2(cnt, u) = 1;
        G_2(u, cnt) = 1;
        pair(u) = cnt;
        for v = 1 : n1
            if G_1(u, v) == 1
                if pair(v) ~= 0
                    G_2(u, pair(v)) = 0;
                    G_2(pair(v), u) = 0;
                    G_2(cnt, pair(v)) = 1;
                    G_2(pair(v), cnt) = 1;
                else
                    G_2(u, v) = 0;
                    G_2(v, u) = 0;
                    G_2(cnt, v) = 1;
                    G_2(v, cnt) = 1;
                end
            end
        end
    end

    for v = 1 : n1
        if G_1(u, v) == 1 && v ~= fa
            dfs(v, u);
```

```
        end
    end
end


function [] = dfs1(u, fa)
    global Load
    global G_2
    global n1
    global n2
    if u <= n1 && u ~= 1
        Load(u) = 1;
    end
    for v = 1 : n1 + n2 - 1
        if G_2(u, v) >= 1 && v ~= fa
            dfs1(v, u);
            if Load(v) > 2
                G_2(u, v) = 2;
                G_2(v, u) = 2;
            end
            Load(u) = Load(u) + Load(v);
        end
    end
end


function set_main_line()
    global G_2
    global n1
    global n2
    for v = 1 : n1 + n2 - 1
        if G_2(1, v) >= 1
            G_2(1, v) = 3;
            G_2(v, 1) = 3;
        end
    end
end
```

create_data.m

```
clear,clc;
center_num = 5;   %簇的数量
point_each = 10;  %每个簇点的数目
r = 100;           %簇的最大半径
```

```
center_point = floor( r * rand(center_num, 2) );  % 簇的中心点
point_delta = 10 * 2 * ( rand(point_each * center_num, 2) - ones(point_each * center_num, 2) );     % 点相对中心点的偏移量
point = [];
for i = 1:5
    for j = 1:10
        point    =    [point;(center_point(i,:)    +point_delta(j+(i-1)*5,:))];
    end
end

xlswrite('test_data_create.xlsx',point);
```

draw_pic_t1.m

```
clear all
close all

load('t1_best.mat');

G_bool = logical(G_3);
G_int = int32(G_bool);
G_col_cum = sum(G_int);
node_leaf = find(G_col_cum <= 1);
K1 = 1;
node_blue = [K1, node_leaf];
node = [1 : size(G_col_cum, 2)];
node_red = setdiff(node, node_blue);

figure
G_n = graph(G_3);
node_col = zeros(size(G_col_cum, 2), 3);
for i = 1 : size(node_blue, 2)
    node_col(node_blue(i), :) = [0 0.4470 0.7410];
end
for i = 1 : size(node_red, 2)
    node_col(node_red(i), :) = [1 0 0];
end
node_col(i, :) = [1 0 0];
node_size = zeros(1, size(G_col_cum, 2));
node_size(:) = 6;
node_size(K1) = 12;
h = plot(G_n, 'NodeColor', node_col, 'MarkerSize', node_size);
% h = plot(G_n, 'NodeColor', node_col);
% h = plot(G_n);
```

```
h.XData = P_x;
h.YData = P_y;
h.LineWidth = 1.5 * G_n.Edges.Weight;
```

get_MST.m

```
% 获取最小生成树
% 输入：(P_x0,P_y0)——所有节点（包括电源）的坐标
% 输出：G——最小生成树的邻接矩阵
function G = get_MST(P_x0, P_y0)
    global fa           % 并查集
    global W            % 欧几里得距离
    global P_ij         % W 对应的线段端点
    global G            % 最小生成树的邻接矩阵
    global e            % 最小生成树的边集
    global n1           % 负荷数+1

    n1 = size(P_x0, 2);
    e = zeros(2, n1 - 1);
    G = zeros(n1);
    P_ij = zeros(2, n1 * (n1 - 1) / 2);
    W = zeros(1, n1 * (n1 - 1) / 2);
    fa = zeros(1, n1);
    for i = 1 : n1
        fa(i) = i;
    end

    cnt0 = 1;
    for i = 1 : n1
        for j = i + 1 : n1
            P_ij(1, cnt0) = i;
            P_ij(2, cnt0) = j;
            W(cnt0) = sqrt((P_x0(i) - P_x0(j))^2 + (P_y0(i) -
P_y0(j))^2);
            cnt0 = cnt0 + 1;
        end
    end
    % kruskal 求解最小生成树
    get_mst();
    G = G + G';

%     % 画图
%     figure
```

```
%     G_2 = graph(G);
%     % G_2 = graph(e(1, :), e(2, :));
%     h = plot(G_2);
%     h.XData = P_x0;
%     h.YData = P_y0;
end

function [] = get_mst()
    global G
    global P_ij
    global W
    global fa
    global n1
    global e
    [W, iw] = sort(W);
    P_ij(:, :) = P_ij(:, iw);
    cnt = 0;
    for i = 1 : n1 * (n1 - 1) / 2
        u = P_ij(1, i);
        v = P_ij(2, i);
        fx = Find(u);
        fy = Find(v);
        if fx ~= fy
            fa(fx) = fy;
            G(u, v) = 1;
            cnt = cnt + 1;
            e(1, cnt) = u;
            e(2, cnt) = v;
            if cnt >= n1 - 1
                break;
            end
        end
    end
end

function res = Find(x)
    global fa
    if x == fa(x)
        res = x;
    else
        fa(x) = Find(fa(x));
        res = fa(x);
    end
end
```

Hamilton.m

```
% Single supply and distribution network model
% 输入: (point , k)——聚类点, 聚类次数
% 输出: G_1——配电网络拓扑图
% (P_x, P_y)——网络中节点坐标
function [shortest,G] = Hamilton(center)
    size = length(center);
    center
    %dp[1 << 20][27]; // %dp[i][j]表示经过了 i 对应二进制数的点 到第 j-1 个点的最短距离
    %1 号点出发到某个点的 hamilton 路径的最小值
    dist = zeros(size,size);
    for i = 1:size
        for j = 1:size
            dist(i,j) = norm(center(i,:)-center(j,:));

        end
    end
    dist
    dp = Inf*ones(bitshift(1, size),size);

    road = cell(bitshift(1, size),size);
    road{1+1,1} = [1];
    road{1+1,1};
    dp(1+1,0+1) = 0;
    for i = 1:(bitshift(1, size)-1)
        for j = 0:(size-1)
            bitshift(i, -1*j);
            bitand(bitshift(i, -1*j),1);
            if(bitand(bitshift(i, -1*j),1 ) ) %此时的 j 点是已走点
                for k = 0:(size-1) % 上一次经过的点对应的二进制数是 i ^ 1 << j
                    if( bitxor(i, bitand( bitshift( bitshift(1,j),-1*k) ,1) ) )  %在上一次经过的点中 k 必须是经过了的  现在是求把 j 点加入的最短路径

if(dp(i+1,j+1)>dp(bitxor(i ,bitshift(1,j) )+1,k+1) + dist(k+1,j+1))
                            road{i+1,j+1}                           = road{bitxor(i ,bitshift(1,j))+1,k+1};
                            road{i+1,j+1} = [road{i+1,j+1},j+1];
                        end
                        dp(i+1,j+1)     =     min(     dp(i+1,j+1), dp(bitxor(i ,bitshift(1,j) )+1,k+1) + dist(k+1,j+1));
```

```
                    end
                end
            end
        end
    end
    dp
    for i = 1:bitshift(1, size)
        road{i,:}
    end
    result = min(dp(bitshift(1,size)- 1 +1, :));
    result_index = find(dp(bitshift(1,size)- 1 +1, :)==result);
    shortest = road{bitshift(1,size)- 1+1,result_index}
    x = center(:,1);
    y = center(:,2);

    A = zeros(size,size);
    for i = 1:size-1
        A(shortest(i),shortest(i+1))                                    =
dist(shortest(i),shortest(i+1));
        %A(shortest(i+1),shortest(i))                                   =
dist(shortest(i+1),shortest(i));
    end

    G = digraph(A);
end


function count = find_one(bin)
    count = 0;

    while (bin ~=0 )
        if (bitand(bin,1) == 1)

            count = count+1;

        end
        bin = bitshift(bin,-1);
    end
end
```

k_means.m

```
% Single supply and distribution network model
% 输入: (point , k)——聚类点, 聚类次数
% 输出: G_1——配电网络拓扑图
% (P_x, P_y)——网络中节点坐标
function [center,group] = k_means(point , k)

    count = 100000;  % 定义最大循环次数
    [N,~] = size(point);
    center = point(1:k,:);  % 令前 k 个点为初始的聚类中心
    group=cell(k);
    distance_square = zeros(N,k);
    while count~=0
        for i = 1:k
            distance_square(:,i)        =        sum((point        -
repmat(center(i,:),N,1)).^2,2);
            group{i} = [];
        end  % 计算到每个点到各个聚类中心的距离

        for i = 1:N
            minposition                                                =
find(distance_square(i,:)==min(distance_square(i,:)));
            group{minposition} =  [group{minposition};point(i,:)];
        end  % 建立第一次分类后的分类点集

        for i = 1:k
            center_New(i,:) = mean(group{i},1);
        end  % 计算新的聚类中心

        if sym(sum((center_New - center).^2)) == 0
            break
        else
            center = center_New;
        end  % 如果中心未改变则跳出循环

        count = count-1;
    end
end
```

k_means_2.m

```
clear,clc;

k = 2;  % 输入聚类组数
%data = importdata('test_data_create.xlsx');
```

```
data_ = importdata('load_demo.xlsx');
data = data_.qh;
data = [data;data_.yh];
[center,group_] = k_means(data,2);
center
group = cell(k,1)
for i = 1:k
    group{i,1} = group_{i,1}
end
group{i,1}(:,1)
hold on

for i = 1:k

plot(group{i,1}(:,1),group{i,1}(:,2),'.','Markersize',15,'color',[r
and rand rand]);

    kn = boundary(group{i,1}(:,1),group{i,1}(:,2),0.1);

    if isempty(kn)
        plot(group{i,1}(:,1),group{i,1}(:,2));
    else
        plot(group{i,1}(kn,1),group{i,1}(kn,2));
    end
    plot(center(:,1),center(:,2),'k+');
end  % 绘图
xlswrite('part.xlsx',center,'center');

for i = 1:k
    xlswrite('part.xlsx',group{i,1},['group',num2str(i)]);
end
```

k_means_5.m

```
clear,clc;

k = 5;  % 输入聚类组数
data_ = importdata('part.xlsx');
dataA = data_.group1
dataB = data_.group2

pointA = dataA;   %各坐标
pointB = dataB;   %各坐标
[center1,groupA] = k_means(pointA,k);
[center2,groupB] = k_means(pointB,k);
```

```
hold on
for i = 1:k

plot(groupA{i,1}(:,1),groupA{i,1}(:,2),'.','Markersize',15,'color',
[rand rand rand]);

    kn = boundary(groupA{i,1}(:,1),groupA{i,1}(:,2),0.1);

    if isempty(kn)
        plot(groupA{i,1}(:,1),groupA{i,1}(:,2));
    else
        plot(groupA{i,1}(kn,1),groupA{i,1}(kn,2));
    end
    plot(center1(:,1),center1(:,2),'k+');
end  % 绘图
xlswrite('centerA.xlsx',center1,'center');

for i = 1:k
    xlswrite('centerA.xlsx',groupA{i,1},['groupA',num2str(i)]);
end



for i = 1:k

plot(groupB{i,1}(:,1),groupB{i,1}(:,2),'.','Markersize',15,'color',
[rand rand rand]);

    kn = boundary(groupB{i,1}(:,1),groupB{i,1}(:,2),0.1);

    if isempty(kn)
        plot(groupB{i,1}(:,1),groupB{i,1}(:,2));
    else
        plot(groupB{i,1}(kn,1),groupB{i,1}(kn,2));
    end
    plot(center2(:,1),center2(:,2),'k+');
end  % 绘图
xlswrite('centerB.xlsx',center2,'center');

for i = 1:k
    xlswrite('centerB.xlsx',groupB{i,1},['groupB',num2str(i)]);
end
```

plot_road.m

```
source1 = [-24602.492188,-11826.172852];
source2 = [-25531.816406,-13508.734375];
data1 = importdata('centerA.xlsx');
data2 = importdata('centerB.xlsx');

center1 = data1.center
center2 = data2.center



center1 = [source1;center1];
center2 = [source2;center2];
xlswrite('road.xlsx',center1,'center1');
xlswrite('road.xlsx',center2,'center2');
%center = [0,0;0,1;0,2];
size1 = length(center1);
size2 = length(center2);



[shortest1,G1] = Hamilton(center1);
[shortest2,G2] = Hamilton(center2);
xlswrite('road.xlsx',shortest1,'shortest1');
xlswrite('road.xlsx',shortest2,'shortest2');

x1 = center1(:,1);
y1 = center1(:,2);
plot(G1,'XData',x1,'YData',y1);
hold on;
x2 = center2(:,1);
y2 = center2(:,2);
plot(G1,'XData',x2,'YData',y2);
hold on;
%for i = 1:k
 %   xlswrite('center.xlsx',group{i},['group',num2str(i)]);
%end
```

SSDN_model.m

```
% Single supply and distribution network model
% 输入: (P_x0, P_y0)——负荷以及电源坐标, 电源为 1 号节点
% 输出: G_1——配电网络拓扑图
% (P_x, P_y)——网络中节点坐标
function [G_3, P_x, P_y, fval, r] = SSDN_model(P_x0, P_y0)
    % global 将变量放入全局工作区, 其他部分可同样通过 global 获取该变量
```

```
    global P_x      % 所有节点坐标
    global P_y
    global G_3      % 配电网拓扑的邻接矩阵
    global e_3      % 配电网拓扑的边集
    global cnt_e
    global n        % 节点总数
    global n1       % 负荷数+1
    global cost_line    % 线路单价
    global cost_switch  % 开关单价
    global r            % 用户用电可靠性


    G = get_MST(P_x0, P_y0);
    [G_3, P_x, P_y] = constructive_algorithm(G, P_x0, P_y0);    % 增加构造算法构造的分叉点

    n = size(P_x, 2);

    % 预处理获取边集，优化算法效率
    e_3 = zeros(n * (n - 1) / 2, 3);
    cnt_e = 0;
    for i = 1 : n
        for j = i + 1 : n
            if G_3(i, j) >= 1
                cnt_e = cnt_e + 1;
                e_3(cnt_e, 1) = i;
                e_3(cnt_e, 2) = j;
                e_3(cnt_e, 3) = G_3(i, j);
            end
        end
    end
%     cnt_e

    cost_line = [0 188.6 239.4 325.7];
    cost_switch = [2.6 56.8];


    % 求解器
    % 无约束极值问题
    n;
    n1;
    if(n-n1~=0)
        [x, fval] = fminunc(@f1, rand(2 * (n - n1), 1) - 0.5);
    end
```

```
    fval = f1(zeros(2 * (n - n1), 1));
    fval = fval + cost_switch * [n1 - 1; n - n1];
    % fval,x
    for i = n1 + 1 : n
        P_x(i) = P_x(i) + x(2 * (i - n1) - 1);
        P_y(i) = P_y(i) + x(2 * (i - n1));
    end

    r = zeros(n, 1);
    clac_power_reliability();

%     % 可视化
%     figure
%     G_n = graph(G_3);
%     col = zeros(n, 3);
%     for i = 1 : n1
%         col(i, :) = [0 0.4470 0.7410];
%     end
%     for i = n1 + 1 : n
%         col(i, :) = [1 0 0];
%     end
%     h = plot(G_n, 'NodeColor', col);
%     h.XData = P_x;
%     h.YData = P_y;
%     h.LineWidth = 1.5 * G_n.Edges.Weight;
end

% function cost = f1(x)
%     global n
%     global n1
%     delta_x = zeros(n - n1);
%     delta_y = zeros(n - n1);
%     for i = 1 : n - n1
%         delta_x(i) = x(2 * i - 1);
%         delta_y(i) = x(2 * i);
%     end
%     cost = f(delta_x, delta_y);
% end

% 评价函数
function cost = f1(x)
   global P_x
    global P_y
```

```
    global e_3
    global cnt_e
    global n
    global n1
    global cost_line
    cost = 0;
    delta_x = zeros(n - n1);
    delta_y = zeros(n - n1);
    for i = 1 : n - n1
        delta_x(i) = x(2 * i - 1);
        delta_y(i) = x(2 * i);
    end
    for i = n1 + 1 : n
        P_x(i) = P_x(i) + delta_x(i - n1);
        P_y(i) = P_y(i) + delta_y(i - n1);
    end
%     for i = 1 : n
%         for j = i + 1 : n
%             if G_3(i, j) >= 1
%                    dis_ij = sqrt((P_x(i) - P_x(j))^2 + (P_y(i) -
P_y(j))^2);
%                cost = cost + dis_ij * cost_line(G_3(i, j) + 1);
%             end
%         end
%     end
    for i = 1 : cnt_e
        dis_ij  =  sqrt((P_x(e_3(i,  1))  -  P_x(e_3(i,  2)))^2  +
(P_y(e_3(i, 1)) - P_y(e_3(i, 2)))^2);
        cost = cost + dis_ij * cost_line(e_3(i, 3) + 1);
    end
    for i = n1 + 1 : n
        P_x(i) = P_x(i) - delta_x(i - n1);
        P_y(i) = P_y(i) - delta_y(i - n1);
    end
end

% 计算用户用电可靠性
function [] = clac_power_reliability()
    global r
    global G_3
    global n
    r(1) = 1 - 0.005;
    for v = 1 : n
        if G_3(1, v) >= 1
```

```
            dfs2(v, 1);
        end
    end
end

function [] = dfs2(u, fa)
    global r
    global G_3
    global n
    global P_x
    global P_y
    dis_ufa = sqrt((P_x(u) - P_x(fa))^2 + (P_y(u) - P_y(fa))^2);
    r(u) = r(fa) * (1 - 0.005) * (1 - 0.002) * (1 - 0.002 * dis_ufa);
    for v = 1 : n
        if G_3(u, v) >= 1 && v ~= fa
            dfs2(v, u);
        end
    end
end
```

main.cpp

```
#include <bits/stdc++.h>
#define MAXPATH 10
using namespace std;
const int N = 55;
// 线路以及开关单位造价
const double cost_circuit1 = 325.7;
const double cost_link = 2 * cost_circuit1;
const double cost_switch2 = 56.8;
// 故障单元可靠性
const double r_node = (1 - 0.005);
const double r_switch = (1 - 0.002);
const double errate_line = 0.002;

int lamb[2];

struct load {
   double x, y;
   double r, r0;
   double c;   // r(loadj)/r(Ki)=c
   int lev;   // 优化级数
   int K;     // 所属分叉点
   bool operator < (const load &a) const {
      return r > a.r; // 最低用电可靠性优先
```

```
    }
};

struct node {
    int num;       // 节点编号
    double x, y;   // 节点坐标
    int nu;        // 节点支线连接子树中负荷总数

    // more
    double r[N / 2];   // 考虑不同数量供电路径的可靠性
    int cnt_path;      // 可供电路径数量
    vector<load> Load; // 支线连接子树中的负荷
    priority_queue<load> pq;   // 第一级最小堆
} Node[N];
/*
节点编号：1 号网络为 1-n，2 号网络为(n+1)-(n+m)
电源编号为 1 和(n+1)
*/

priority_queue<load> gpq;  // 第二级最小堆，也称全局最小堆

struct line {
    int u, v;      // 线关联两端点
    int Nuv[2];    // Nuv[0]=nu+nv,Nuv[1]=|nu-nv|
    double len;    // 线路长度
    double cost;   // 建造花费
    double fval;   // 启发式函数值

    bool operator < (const line& a) const {
        return fval > a.fval;  // min f()
    }
};

struct circ {
    int u, v;
    double r;
    bool operator < (const circ& a) const {
        return u < a.u;
    }
};

struct path {  // 供电路径
    set<circ> subpath; // 子路径集合
} Path[MAXPATH];
```

```
int cnt;

vector<node> Net[2];   // 两个单供网络主线上的分叉点集合
int n, m;

vector<int> dnet[N];   // 双供网络邻接表
map<pair<int, int>, bool> cont;

void read_graph();
void read_rdata();

// 启发式贪心算法
void heuristic_greedy_method(vector<node> net[2]);
// 负荷分块改进算法
void load_partition_improved_method(vector<node> net[2], int k);
// 基于 n 元容斥定理计算主线上分叉点的用电可靠性
void calc_reliability();
// 两级最小堆方案优化最低用电可靠性
void Two_Level_Heap_optimization();

double dis(node &a, node &b);      // 两点之间距离函数
bool judge_cross(line &a, line &b); // 判断线是否相交
vector<node> load_partition(vector<node> net, int k);  // 负荷分块
void dfs(int u, int fa, int flag); // dfs 所有供电路径

int main() {
   freopen("data.txt", "r", stdin);

   // 协调变量设定
   lamb[0] = lamb[1] = 0;

   // 数据输入
   read_graph();
   // 应用贪心算法
   heuristic_greedy_method(Net);
   // 应用负荷分块改进后的算法
   load_partition_improved_method(Net, 2);

   //for (int i = 0; i < n + m; i++) {
   // cout << "(" << i << ")";
   // for (int j : dnet[i]) {
   //    cout << j << " ";
   // }  cout << '\n';
   //}
```

```
    // 计算主线分叉点可靠性
    calc_reliability();
    read_rdata();
    // 两级堆方案优化
    Two_Level_Heap_optimization();

    for (int i = 2; i <= n + m; i++) {
        if (i == (n + 1)) continue;
        for (int j = 0; j < Node[i].Load.size(); j++) {
            //cout << Node[i].Load[j].x << '\n';
            cout << Node[i].Load[j].y << '\n';
            //cout << Node[i].Load[j].r << '\n';
        }
    }
    return 0;
}

void read_graph() {
    cin >> n >> m;
    //cout << n << ' ' << m << '\n';
    double x, y;
    for (int i = 1; i <= n; i++) {
        cin >> x >> y;
        Node[i].num = i;
        Node[i].x = x; Node[i].y = y;
    }
    for (int i = 1; i <= m; i++) {
        cin >> x >> y;
        Node[n + i].num = n + i;
        Node[n + i].x = x; Node[n + i].y = y;
    }
    int k;
    vector<int> tmp;
    for (int i = 1; i <= n; i++) {
        cin >> k;
        tmp.push_back(k);
        if (k != 1) { Net[0].push_back(Node[k]); }
    }
    for (int i = 0; i < n - 1; i++) {
        dnet[tmp[i + 1]].push_back(tmp[i]);
    }  // 构建双供网络邻接表
    tmp.clear();
    for (int i = 1; i <= m; i++) {
```

```
      cin >> k;
      tmp.push_back(n + k);
      if (k != 1) { Net[1].push_back(Node[n + k]); }
   }
   for (int i = 0; i < m - 1; i++) {
      dnet[tmp[i + 1]].push_back(tmp[i]);
   }
}

void read_rdata() {
   string line;
   double r;
   getchar(); // '\n'
   cout << '\n';
   for (int i = 2; i <= n + m; i++) {
      if (i == (n + 1)) continue;
      getline(cin, line);
      stringstream sstream(line);
      while (sstream >> r) {
         //cout << r << ' ';
         load newone;
         newone.r = r;
         newone.r0 = r;
         newone.c = newone.r0 / Node[i].r[1];
         newone.K = i;
         newone.lev = 1;
         Node[i].Load.push_back(newone);
         Node[i].pq.push(newone);
      }  //cout << '\n';
   }
   double x, y;
   for (int i = 2; i <= n + m; i++) {
      if (i == (n + 1)) continue;
      for (int j = 0; j < Node[i].Load.size(); j++) {
         cin >> x >> y;
         Node[i].Load[j].x = x;
         Node[i].Load[j].y = y;
      }
   }
}

void heuristic_greedy_method(vector<node> net[2]) {
   vector<line> L_cont;   // 联络线设置集合
   double cost_cont = 0;   // 当前联络线花费
```

```
   priority_queue<line> pq;

   for (node &i : net[0]) {
      for (node &j : net[1]) {
         line new_one;
         new_one.u = i.num; new_one.v = j.num;
         new_one.len = dis(i, j) / 1000;    // (km)
         new_one.cost = new_one.len * cost_link + 2 * cost_switch2;
         new_one.Nuv[0] = i.nu + j.nu;
         new_one.Nuv[1] = abs(i.nu - j.nu);
           new_one.fval = new_one.len - lamb[0] * new_one.Nuv[0] +
lamb[1] * new_one.Nuv[1];
          //printf("(%d,%d):{length:%lf,cost:%lf,Nuv:(%d,%d),fval:%l
f}\n", \
              //  new_one.u, new_one.v, new_one.len, new_one.cost,
new_one.Nuv[0], new_one.Nuv[1], new_one.fval);
         pq.push(new_one);
      }
   }

   while (!pq.empty()) {
      line new_cont = pq.top();
      pq.pop();
      bool flag = true;
      for (line &i : L_cont) {
         if (!judge_cross(i, new_cont)) {
            flag = false;  // 不满足约束<2>
            break;
         }
      }
      if (flag) {
         L_cont.push_back(new_cont); // {L_cont} ∪ {new_cont}
         cost_cont += new_cont.cost;
         //printf("{%lf,%lf}\n", cost_cont, new_cont.cost);
      }
      //else { break; }
   }
   //return L_cont;

   cout << "{L_cont}={";
   int L = L_cont.size();
   for (int i = 0; i < L; i++) {
       cout << "(" << L_cont[i].u << "," << L_cont[i].v << ")" <<
",}"[i == L - 1];
```

```
        dnet[L_cont[i].u].push_back(L_cont[i].v);
        dnet[L_cont[i].v].push_back(L_cont[i].u);
        cont[{L_cont[i].u, L_cont[i].v}] = true;
        cont[{L_cont[i].v, L_cont[i].u}] = true;
    }
    cout << '\n';
}

void load_partition_improved_method(vector<node> net[2], int k) {
    // k 为分块大小
    vector<node> vir_net[2];
    vir_net[0] = load_partition(net[0], k);
    vir_net[1] = load_partition(net[1], k);
    int n = vir_net[0].size(), m = vir_net[1].size();
    if (n < m) {
        swap(vir_net[0], vir_net[1]);
        swap(n, m);
    }  // n >= m
    // 将主线末端虚拟点进一步合并，保证最终 n=m
    for (int i = 0; i < n - m; i++) {
        vir_net[0][m - 1].x += vir_net[0][n - i - 1].x;
        vir_net[0][m - 1].y += vir_net[0][n - i - 1].y;
    }
    vir_net[0][m - 1].x /= (n - m + 1);
    vir_net[0][m - 1].y /= (n - m + 1); // 将 (n1-n2+1) 个点合并

    /*
    todo:|{L_cont}*|=max(|{L_cont}|)
    but O(n!)
    */
}

void calc_reliability() {
    for (int i = 2; i <= n + m; i++) {
        if (i == n + 1) continue;
        cnt = 0;
        dfs(i, 0, 0);  // Path[0-(cnt-1)]

        //cout << "[" << i << "]" << '\n';
        //for (int j = 0; j < cnt; j++) {
        //  for (circ k : Path[j].subpath) {
        //      cout << "(" << k.u << "," << k.v << ")";
        //  }cout << '\n';
        //}
```

```
      //break;
      //cout << "r:";
      Node[i].cnt_path = cnt;
      for (int j = 1; j <= cnt; j++) {   // 考虑前 j 条供电路径
         // 邻接表的建立次序保证第一条为正常供电路径
         double r = 0;
         if (j == 1) {
            r = 1;
            for (circ k : Path[0].subpath) {
               r *= k.r;
            }
            r *= r_node;   // 考虑电源可靠性
         }
         else {
            // 容斥原理
            for (int k = 1; k < (1 << j); k++) {   // 0/1 状态压缩，共 j 位代表 j 条路径
               set<circ> uset; // 路径求并
               int op = -1;   // 奇数项为正
               for (int t = 0; t < j; t++) {
                  if ((1 << t) & k) {
                                 set_union(Path[t].subpath.begin(), Path[t].subpath.end(), \
                           uset.begin(), uset.end(), inserter(uset, uset.begin()));
                     op = -op;
                  }
               }
               double r_item = 1;
               for (circ cir : uset) {
                  r_item *= cir.r;   // r(P1∩P2∩...∩Pn)
               }
               r_item *= r_node;
               r += op * r_item;
            }
         }
         Node[i].r[j] = r;
         //cout << r << ' ';
      }
      //cout << '\n';
   }
}

void Two_Level_Heap_optimization() {
```

```
    // 构建全局最小堆
    for (int i = 2; i <= n + m; i++) {
        if (i == (n + 1)) continue;
        load local_min = Node[i].pq.top();
        gpq.push(local_min);
    }

    for (int t = 1; t <= 1000; t++) {  // 第 t 轮优化
        // 单步优化
        load gm = gpq.top(); gpq.pop();
        Node[gm.K].pq.pop();
        gm.lev++;
        // 方案饱和，无法继续优化
        if (gm.lev > Node[gm.K].cnt_path) {
            cout << gm.r << '\n';
            cout << "Have reached saturation\n";
            break;
        }
        //cout << gm.r << '\n';
        //cout << gm.K << "  " << gm.r << "  ";
        gm.r = gm.c * Node[gm.K].r[gm.lev];
        //cout << gm.r << '\n';

        //printf("%lf×%lf\n", gm.c, Node[gm.K].r[gm.lev]);
        Node[gm.K].pq.push(gm);
        load lm = Node[gm.K].pq.top();
        gpq.push(lm);
    }
}

stack<circ> curr_path0;
set<circ> curr_path;

void dfs(int u, int fa, int flag) { // flag: 保证只经过 1 条联络线
    // 若搜索到电源则发现一条供电路径
    if (u == 1 || u == n + 1) {
        Path[cnt++].subpath = curr_path;
        //return;
    }
    circ cir;
    for (int v : dnet[u]) {
        if (v != fa) {
            double d = dis(Node[u], Node[v]) / 1000;   // (km)
            //cout << "d:" << d << '\n';
```

```
            cir.u = u; cir.v = v;
            cir.r = (1 - d * errate_line) * r_switch;
            if (cont[{u, v}]) {
               if (flag) continue;
               flag++;
            }
            curr_path0.push(cir);
            curr_path.insert(cir);
            //cout << "push (" << u << "," << v << ")\n";
            dfs(v, u, flag);
            if (cont[{u, v}]) { flag--; }
         }
      }
      // 节点访问结束，弹栈
      if (!curr_path0.empty()) {
         cir = curr_path0.top(); curr_path0.pop();
         //cout << "pop (" << cir.u << "," << cir.v << ")\n";
         curr_path.erase(cir);
      }
   }

   vector<node> load_partition(vector<node> net, int k) {
      vector<node> vir_net; vir_net.clear();
      int n = net.size();
      for (int i = 0; i < n; i += k) {
         node new_node;
         double sumx, sumy;
         sumx = sumy = 0;
         int re = min(k, n - i);
         for (int j = 0; j < re; j++) {
            sumx += net[i].x;
            sumy += net[i].y;
         }
         new_node.x = sumx / re;
         new_node.y = sumy / re;
         vir_net.push_back(new_node);
      }
      return vir_net;
   }

   double dis(node& a, node& b) {
       return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y -
   b.y));
   }
```

```
bool judge_cross(line& a, line& b) {
   node A, B, C, D;
   A = Node[a.u]; B = Node[b.u];
   C = Node[a.v]; D = Node[b.v];
   // 点 A 和 D 应在直线 BC 两侧
   // 需保证 BC 不和坐标轴平行
   double fa, fb;
   fa = (A.y - C.y) / (B.y - C.y) - (A.x - C.x) / (B.x - C.x);
   fb = (D.y - C.y) / (B.y - C.y) - (D.x - C.x) / (B.x - C.x);
   return (fa * fb < 0);
}
```