配电网可靠性和故障软自愈研究

张荣凯、倪诗宇、张朝阳

摘要

电网是全国民生、工业基础，由发、输、变、配、用各环节构成统一整体。运行需要有较高的稳定性来保证供电质量，并且在时空拓扑结构不断变化的情况下，要能够对故障能够进行判断和处理。本文利用加权无向拓扑图数据结构，对实际电网的结构进行建模，将电网问题转变为图论问题，直接对拓扑图上信息进行计算处理。并且能够在业务流动过程中，不断并入新的电网，实现业务拓展，即图节点与边的增删改。基于指标和图拓扑对可靠性进行分析，并且给出了基于连续时空拓扑的智能化配电网软自愈控制方案。

针对问题一：

本文采用最小路和等值法的混合算法进行可靠性评估：首先给出负荷点以及配电系统的可靠性评估指标。然后使用等值法把复杂配电网结构转化成一个简单的配电网结构，在简化的基础上，将其抽象为无向图。在无向图结构上求解以电源为源点到各个负荷点的最小路，将负荷点的可靠性计算分为最小路上以及非最小路上的计算，并且把非最小路上的元件故障率等价到最小路上的点上，然后根据公式计算出该层负荷点可靠性，递归计算等值法中每一层的各个负荷点的指标,从而得到所有负荷点的指标，再进一步利用负荷点可靠性指标计算配电系统的可靠性指标。最后，对于拓扑信息进行了灵敏度分析，给出了定性分析和定量的计算方式，举出算例说明结果。

针对问题二：

配电网在任意时刻本质上都是一张拓扑图，因此我们首先将配电网抽象成具有拓扑结构的图。根据不同元件的的拓扑关系，推导出电网元件之间动作的因果关系，从而构建因果网络。然后，将因果网络中包含的因果信息保存在规则矩阵 R 中。之后根据电网检测系统获取的电网警告信息建立状态向量 T，通过将规则矩阵 R 的转置矩阵 R^T 与向量 T 进行逻辑乘，计算出故障向量信息 T^*。最后，统计出可能故障的节点，并保存在向量 F 中。我们将 T^* 与 F 按位相与，得到最终故障节点向量 F^*，F^* 中为 1 的节点就是故障节点。该模型具有极快的运算速度和较好的鲁棒性。同时，在原有模型上进行改进，提高了模型的准确性。

针对问题三：

本文首先将馈线以及开关建模为加权无向拓扑图 G 上的点和边,其中权值 $Cost_m(I,t)$ 是根据 t 时刻电流值得到的。然后本文给出了故障的定义和分析，将故障的情况对应到图上边的权值异常。根据图 G 中的边权检测来推测出故障点，实现故障判断。接着对故障点附近的开关进行断开，实现故障自动隔离，之后通过连通分支个数判断因故障隔离产生的无故障段“孤岛”的个数，通过开关闭合把“孤岛”并入连通分支，实现无故

障段负荷转移恢复供电。最后，从设备价格成本、维护成本、对配电系统可靠性的影响(代价成本) 角度，给出了“软自愈”和“硬自愈”的成本差异分析，阐明了本文算法优缺点。

关键字：配电网　故障自愈　最小路　等值法　因果网络　连通分量　BFS　马尔可夫链

一、 问题重述

1.1 问题背景

配电网是供电系统的关键组成部分，是决定供电质量的重要因素，90% 的断电都是因为配电网配置不合理导致的。配电网的可靠性是高质量供电的重要指标, 因此许多国家、企业都以提高配电网可靠性为目标进行配电网优化。连续时空拓扑信息，是一个利用发展的眼光研究配电系统业务的重要角度，需要考虑时间和空间的影响，既考虑电网运行过程中的拓展情况，也要考虑给定电网下某段时间内电网信息的变化情况。而对于电网故障，从最原始的人工排除故障恢复供电，发展到 21 世纪通过电子电气设备实现故障的自动判断、恢复。再到今天，国家电网提出“电网一张图”的概念，软件控制实现的软自愈被更多人关注它能更快速地进行线路故障的自动判断、自动隔离和负荷转移，还能做之间无法做的数据分析的工作，进行预测预知，对于提升供电质量有了更大的积极作用。

1.2 问题提出

问题一: 针对配电网，建立一种基于连续时空电网拓扑的可靠性评估模型，并且要对连续时空拓扑信息差异引起的模型可靠性差异进行说明，也就是对拓扑信息进行灵敏度分析。

问题二: 建立一种基于连续时空电网拓扑的配电网故障检测模型，并举例配电网信息差异引起的故障件检测差异。

问题三: 基于“业务内生电网拓扑信息”，研究设计电网的“软自愈”方案，实现同类线路故障的自动判断、自动隔离、负荷转移恢复供电的算法，最后估算“软自愈”与“硬自愈”方案的成本差异，并且结合中压 (10kv) 电网故障图说明。

二、 问题分析

2.1 问题一分析

要求建立可靠性评估模型，并且是依赖于连续时空电网拓扑的，已经明确了要考虑连续时空电网拓扑，即对于时间：我们考虑任意某时刻下的给定的拓扑结构中的元件信息，而空间则是考虑变化的拓扑结构。对于可靠性评估，采用指标 + 模型的评估思路。首先先通过定义评价指标，然后建立计算这些指标的方法，依据拓扑信息，量化指标，对可靠性进行一个合理评估。最后对自变量——电网拓扑信息进行灵敏度分析。

2.2 问题二分析

随着配电网的不断智能化，配电网可以通过 SCADA 系统[1]进行数据检测和控制。通过收集配电网的检测信息，可以得知配电网当前运行状态。配电网存在一定的拓扑结构，各个元件产生的动作之间具有一定的因果关系。因此，可以通过配电网的拓扑结构构建出因果关系网络，元件的动作作为因果关系网络的节点。、我们将收集到的配电网检测信息作为结果，通过因果关系，逆向推导出这些检测信息发生的原因，从而定位故障节点。

2.3 问题三分析

关于软自愈的设计方案：首先将配电网抽象成一个加权无向拓扑图，在此基础上先分析可能导致故障的情形，分析如何判断这种故障出现，进而确定故障点，实现自动判断。然后给出一个初步的开关动作将故障点周围开关关闭，使其孤岛化，实现自动隔离。之后分析是否考虑有误关情况，对于因此变为孤岛的正常节点，输出对应开关动作将其并入存在电源的连通分支，即实现了恢复供电。 本文从设备价格成本、设备维护成本以及对系统的可靠性影响 (看作一种代价成本) 估算与硬自愈的成本差异，最后还分析了除了成本之外的软自愈优点。

三、 模型假设

假设 1 配电网仅存在弱环电网或者无环拓扑结构。

假设 2 配电网有完备的数据监控采集设备，能够根据记录的时间来判断监控设备开关动作的时序。可以用于查找故障主因，便于故障的后续分析。

假设 3 配电网中包括三种类型的节点：故障节点、保护动作节点和短路器跳闸节点。

例如: 故障节点线路 *L1* 上发生错误，则保护节点 *OP1* 将动作使断路器 *CB1* 跳闸。

假设 4 此模型不讨论含有分布式电源 *DG* 的情况

四、 符号说明

表 1　符号表

变量	说明	量纲
λ	负荷点平均故障停运率	次/a
U	负荷点年平均停运时间	h/a
γ	负荷点每次故障平均停电持续时间	h/a
$SAIFI$	系统平均停电频率指标	次 (户 · a)
$SAIDI$	系统平均停电持续时间	h(户 · a)
$ASAI$	平均供电可靠率	%
$ASUI$	平均供电不可靠率	%
$AIHC$	用户平均停电时间	h
$AITC$	用户平均停电次数	次

五、 模型的建立与求解

5.1 问题一

配电网系统的可靠性就是研究向用户供电和分配电能的配电系统本身以及其对用户供电能力的可靠性。必须要建立可靠性评估模型和指标，才能以此为基础进行定量表达。配电网的可靠性评估方法大致分为：模拟法和解析法[2]。解析法模型准确、原理简单，并且便于对不同元件引起的拓扑结构差异进行灵敏度分析，所以我们以解析法中的网络法为基础，采用基于等值法和最小路法的混合算法评估模型来计算可靠性[3]，使用等值法简化配电网网络拓扑结构，使用最小路法计算负荷点或者等值点的可靠性指标，再分层递归计算出每一个负荷点的可靠性指标。并且使用 IEEE 所提出的标准指标对负荷点以及整体配电系统进行量化评价，然后根据这些指标，计算配电系统的整体指标对拓扑信息的偏导以及元件影响的个数来实现灵敏度进行定量计算和定性分析。

5.1.1 指标建立

配电网的可靠性指标分为负荷点和系统的可靠性指标。

负荷点的可靠性指标主要包括:

1) 负荷点平均故障停运率 λ(次/a), 表示某一负荷点在一个具体时间内由于元件故障引起的停电次数; 2) 负荷点年平均停电持续时间 U (h/a)；表示某负荷点在一年内的停电时长的平均值。3) 负荷点每次故障平均停电持续时间 γ; 表示负荷点每次故障会造成影响多久。

系统的可靠性指标包括: 系统平均停电频率指标 $SAIFI$; 系统平均停电持续时间 $SAIDI$; 系统平均供电可靠率 $ASAI$; 系统平均供电不可靠率 $ASUI$。这些指标从频率以及影响用户数、波及范围等不同层面给出了可靠性的评估方式。

很多辐射型网络最终都可以归结到元件的串并联形式上，针对电网的最基础的元件串并联结构，本文给出相关指标计算公式：

n 个串联可修复元件，计算故障指标采用如下公式，其中 λ_i 为某元件 i 的平均故障率;U_i 是某元件 i 的年平均停电时间; γ_i 表示某元件 i 每次故障的平均停电持续时间;μ_i 表示某元件 i 的平均修复率。计算得到 λ_s、U_s 和 γ_s 分别表示这个串联系统 s 的平均故障率、年平均停电持续时间、每次故障的平均停电时间。

$$\begin{cases} \lambda_s = \sum_{i=1}^{n} \lambda_i \\ U_s = \sum_{i=1}^{n} \lambda_i \gamma_i \\ \gamma_s = \frac{\sum_{i=1}^{n} \lambda_i \gamma_i}{\lambda_s} \end{cases} \tag{1}$$

对于 n 个并联可修复元件，计算采用如下公式, 其中 λ_i 为某元件 i 的平均故障率;U_i 是某元件 i 的年平均停电时间; γ_i 表示某元件 i 每次故障的平均停电持续时间;μ_i 表示某

元件 i 的平均修复率。计算得到 λ_s、U_s 和 γ_s 分别表示这个并联系统 s 的平均故障率、年平均停电持续时间、每次故障的平均停电时间。

$$\begin{cases} \lambda_s = \prod_{i=1}^{n} \lambda_i \frac{\sum_{i=1}^{n} \mu_i}{\prod_{i=1}^{n} \mu_i} \\ U_s = \prod_{i=1}^{n} U_i \\ \gamma_s = \frac{1}{\sum_{i=1}^{n} \mu_i} \end{cases} \tag{2}$$

对于系统的可靠性指标，可以通过依据负荷点的故障频率、故障时间以及用户数(量化波及范围、严重程度) 计算得到, 其中 λ_i 为负荷点 i 的年平均故障率;U_i 是负荷点 i 的年平均停电持续时间, N_i 代表用户数，$\sum$ 的含义是整个系统所有负荷点都纳入计算。得到系统的可靠性指标的公式:

$$\begin{cases} SAIFI = \dfrac{\sum \lambda_i N_i}{\sum N_i} \quad ; \quad SAIDI = \dfrac{\sum U_i N_i}{\sum N_i} \\ ASAI = 1 - \dfrac{\sum U_i N_i}{8760 \sum N_i} \quad ; \quad ASUI = 1 - ASAI \end{cases} \tag{3}$$

至此我们已经得到了进行可靠性评估的指标依据以及最基本的计算方法。下面就是阐明如何在一个电网图上进行可靠性计算，本文采用的最小路和等值法混合的计算方法。

5.1.2 等值法简化电网

该问题本质上是一个图论问题，更准确来说是树的问题。

实际的配电系统往往由主馈线和副馈线构成，对这种结构比较复杂的配电网，可以利用可靠性等值的方法将其等值为简单的辐射形配电网，从而简化计算。

等值原理: 按系统的馈线数对其进行分层处理，每一条馈线及该馈线所连接的各种元件（包括隔离开关、分段断路器、熔断器、分支线等）均属同一层，每一层都可以等值为一条相应的等效分支线，这样从最末层开始向上逐层等值，最后一个复杂的含有多条馈线的配电网就等值为一个简单的辐射形网络。

等值原则: 在计算等效分支线可靠性参数时，由于馈线上隔离开关和分段断路器位置的不同，网络结构的不同，可以总结出以下两条原则

1. 如果馈线上装有隔离开关或分段断路器，那么隔离开关或分段断路器后的元件发生故障所引起的等效分支线停运时间为隔离开关或分段断路器的操作时间 S, 并且隔离开关或分段断路器后的元件的检修不会引起等效分支线的停运
2. 隔离开关或分段断路器前的元件发生故障所引起的等效分支线的停运时间为故障元件的修复时间

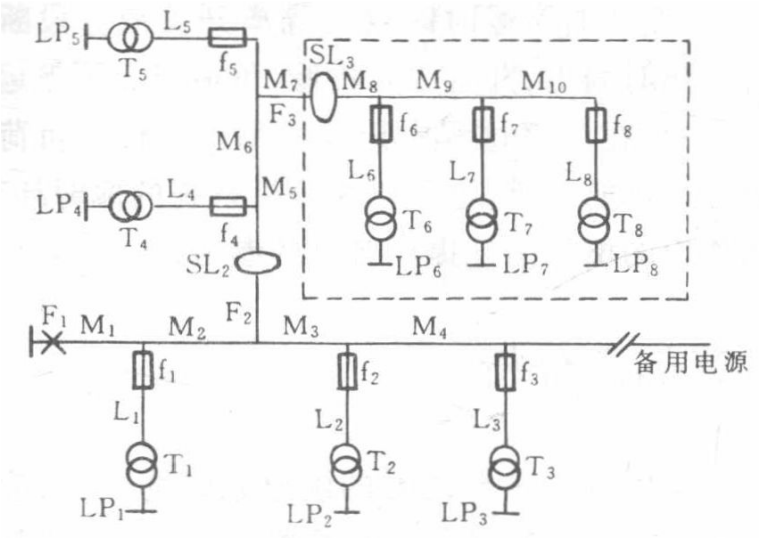

图 1　简单配电网拓扑图

等值法流程如下:

下层元件对上层元件的影响: 低层馈线及其所连接的元件发生故障不仅会影响本层负荷点的可靠性，也会影响其上层负荷点的可靠性。这样，上图图 1 中馈线 3 对馈线 2 的影响可以等值为等效分支线 EL_3 对馈线 2 的影响。等效分支线 EL_3 的可靠性参数通过上述**等值原则计算**。等值后的电网拓扑图如下图 2:

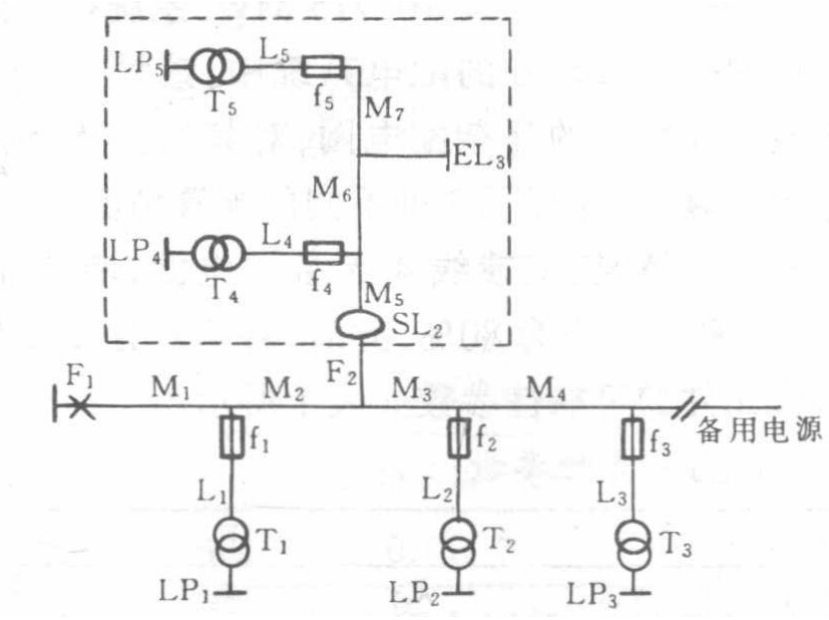

图 2　馈线 3 等值化电网拓扑图

同理可得, 上图中馈线 2 对馈线 1 的影响可以等值为等效分支线 EL_2 对馈线 2 的影响。等效分支线 EL_2 的可靠性参数通过上述**等值原则计算**。等值后的电网拓扑图如下:

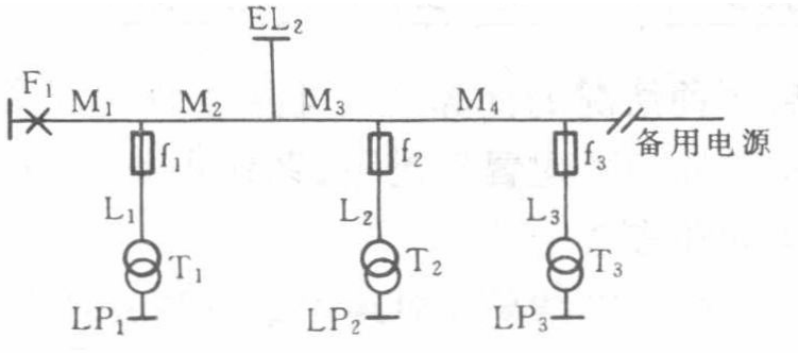

图 3　馈线 2 等值化电网拓扑图

上层元件对下层负载的影响:

上层馈线上连接的元件故障也会影响下层馈线上负荷点的可靠性。可以在下层主馈线上增加一个串联元件 SL，用以表征上层元件故障对下层负载点的影响。这样在计算下一层负荷点的可靠性指标时，仍然可以按只含有一条馈线的简单配电系统计算，从而简化了计算。

从等值法流程可知，进行等值的过程是将电网拓扑结构简化的过程，也是一个折叠的过程。考虑上层元件对下层负载的影响是将简化的电网拓扑图拆分的过程。具体流程就是构建一棵电网拓扑树 T，如下所示：

将主馈线及主馈线上的负载抽象为树的根节点，主馈线上的副馈线及副馈线上所连接的负载抽象为其孩子节点。然后遍历每一条副馈线。此时，将遍历到的副馈线作为二级主馈线。若有其他副馈线连接到该二级主馈线，将该副馈线及副馈线上的负载抽象为树的一个节点，并作为该二级主馈线对应树节点的子节点。然后遍历每一条副馈线。

此时，将遍历到的副馈线作为三级主馈线，以此类推，直到没有副馈线连接到该主馈线，也就是该主馈线及该主馈线上的负责抽象为树的叶子节点

对上述过程进行递归，将电网拓扑结构构建为一棵树。多级馈线以及负载点为 T 上的顶点，馈线与馈线、馈线与负载点存在的连接关系为 T 上的边。 这种方法不仅简化了电网的拓扑结构，同时使得上层馈线上元件的故障得以向下层馈线上的负载传递。以下图电网拓扑结构构建树的过程来辅助说明上述流程。

第一步: 将该主馈线抽象为树的根节点。本例中将馈线 1 及其上连接的元件抽象为根节点

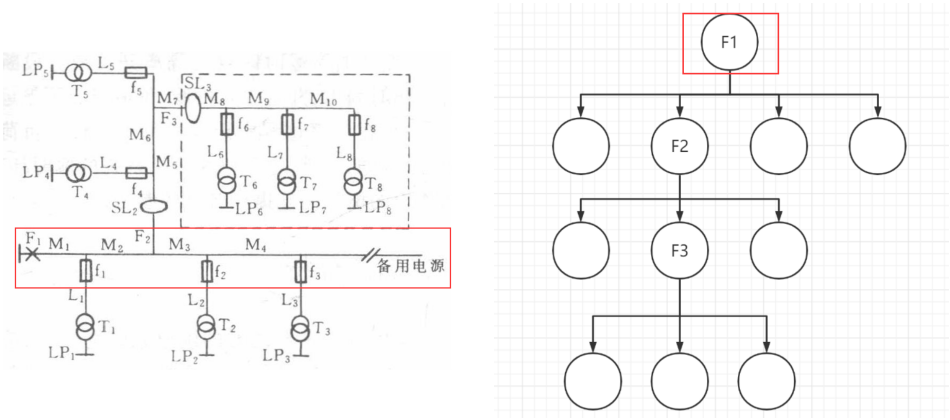

图 4　馈线 1 抽象为根节点

第二步: 搜寻连接到该主馈线上的副馈线，并将该副馈线及其上连接的元件抽象为根节点的子节点。

当遍历到第一条副馈线时，发现该没有更低层的副馈线连接到该馈线。也就是说，该副馈线及其上的元件抽象的节点是树的**叶子节点**。该副馈线会受到与其连接到同一条主馈线上的其他副馈线的影响，也就是说一个节点会受到其兄弟节点的影响，而该影响通过兄弟节点与同一个父节点的连接得到传递。我们计算出该影响的大小 $T1$，并将其作为该节点与父节点连接的边的边权。

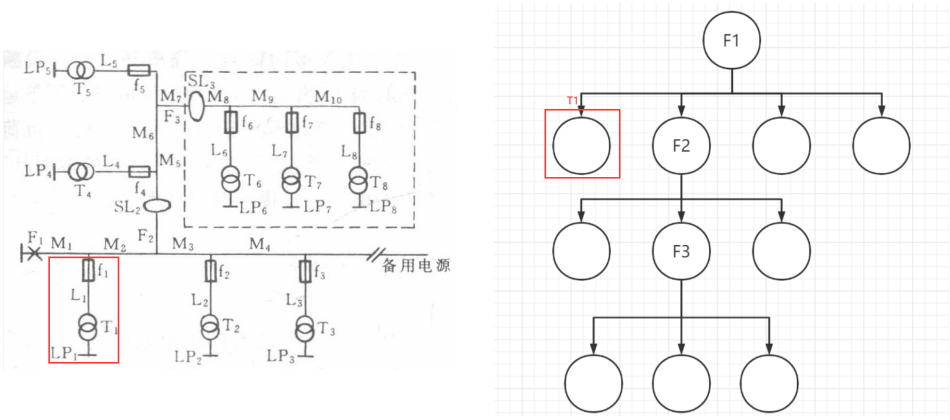

图 5　副馈线 1 抽象为子节点

同理，遍历第二条副馈线，也就是等效分支线 EL_2，计算得到边权 $T2$，该副馈线还可作为二级主馈线，因此向下继续进行，直到对任何节点都不能找到子节点。本过程是一个递归的过程。

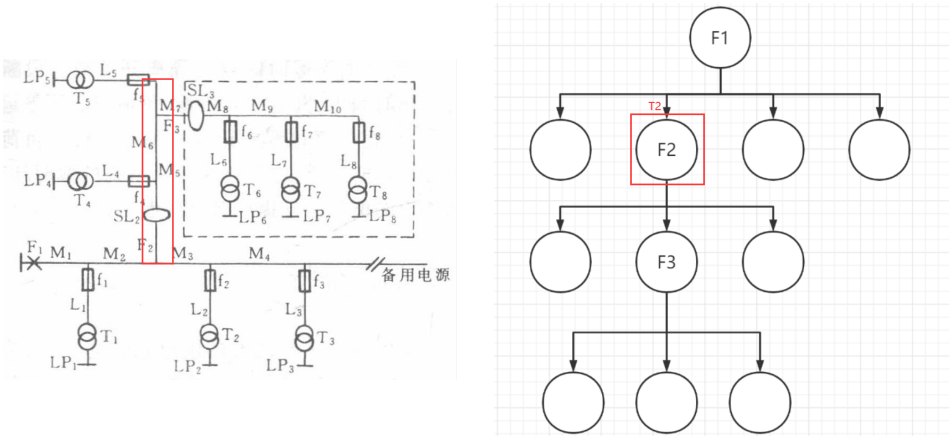

图 6　等效分支线 EL_2 抽象为子节点

将 EL_2 作为主馈线递归上述方法可以构建出下层节点，其中包括等效分支线 EL_3 的节点构建。整个树的构建过程是一个**前序遍历**的过程

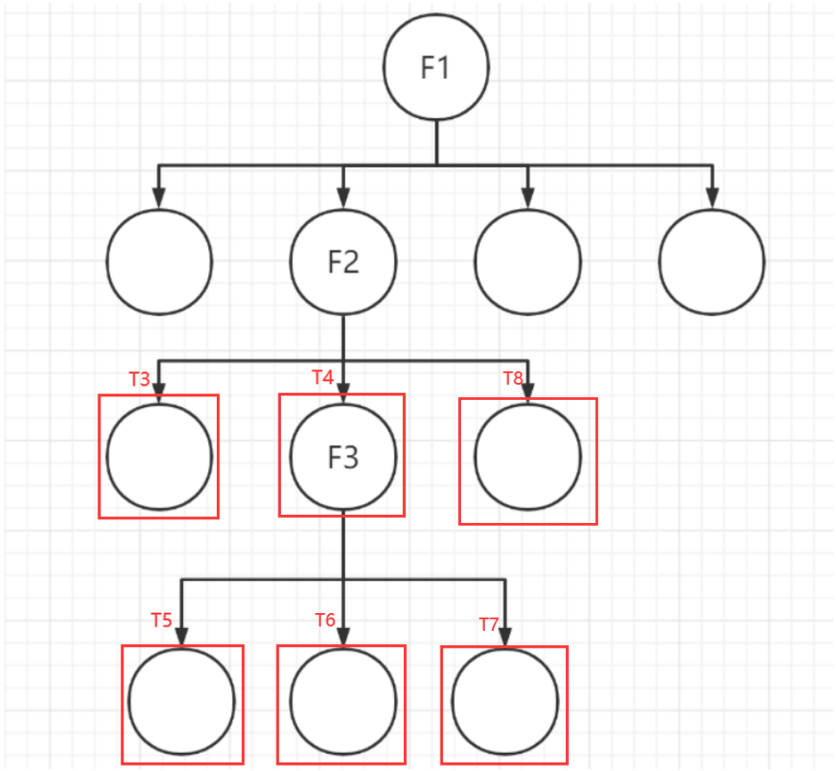

图 7　等效分支线 EL_2 的下层馈线抽象为子节点

遍历完第二条副馈线及其子节点之后，遍历第三条，第四条副馈线。

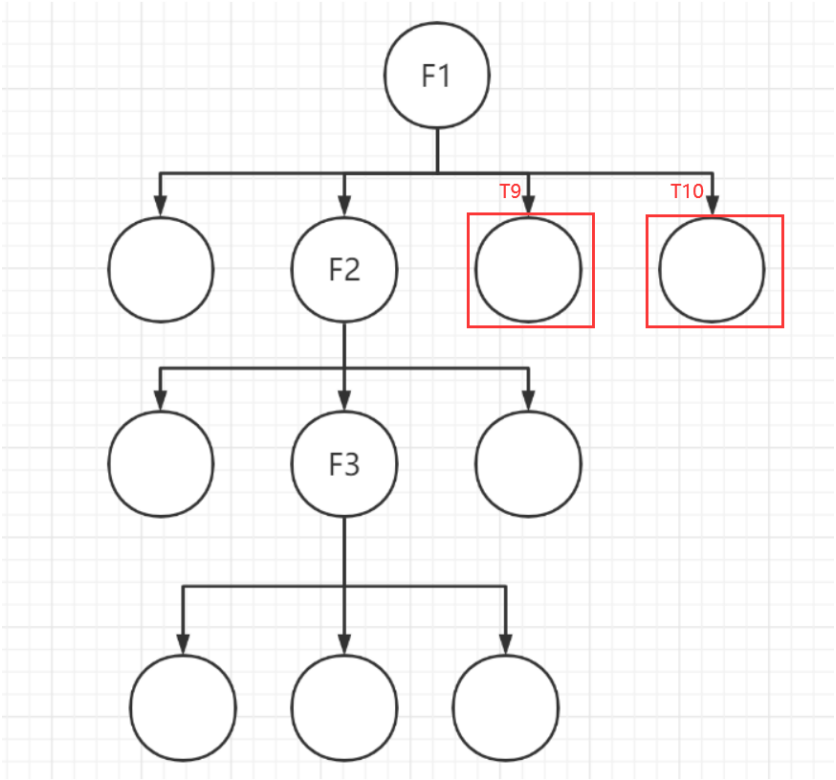

图 8　剩余馈线抽象为子节点

至此，建树完毕。所有上层节点对下层节点的影响通过边权得以传递，这些信息将在最小路算法中得到使用。

综上，等值法能够利用等效分支线，简化电网拓扑结构，使得后续的处理过程更加简单。同时我们也说明了，下层馈线上的元件会对上层馈线上负载的稳定性产生影响，也说明了上层馈线上的元件对下层馈线上的负载的影响是如何传递的。

最小路算法中，会将叶节点到根节点的所有边权进行求和，来计算叶节点的可靠度。**因为边权是上层节点对下层节点的影响大小的值，因此本文在求和的过程中就计算了上层节点对下层节点的影响**。

5.1.3 最小路法

配电网通常采用多闭环的设计思路，而运行过程中采用开环的辐射状结构。其次复杂配电系统等值之后，都可以简化为简单的辐射性网络。对于可靠性评估，我们首先计算不同元件的可靠度对各个负载点的影响，进而估计出整个系统的可靠度。**所以，需要对元件进行分类，采用最小路法，把元件分成最小路元件以及非最小路元件，非最小路对负荷点的影响可以折算到最小路上**。

基本思路：

将配电网抽象成图论上的无向图，具体图化方式为: 将负荷母线以及变电站、变压器看作节点，线路、分段开关看作边。配电网可以抽象为图 $\mathbf{G}=(\mathbf{V},\mathbf{E})$ 表示，自然得出

图的邻接矩阵，电源看作最短路的源点，化成单源无向图。对每一负荷点，然后求其到电源最短路。之后，根据实际的电网，把非最小路的元件故障影响折算到最小路。对每一个负荷点的可靠性的计算仅需考虑，最小路上的原件与节点之间的计算，即可以得到相应负载得到可靠性。

下面选用简单示例图 9、10 来说明最小路法的计算过程图 9 为原始简单电网拓扑图，图 10 为抽象到图论意义上的电网拓扑图。以此来说明最小路法的计算过程。

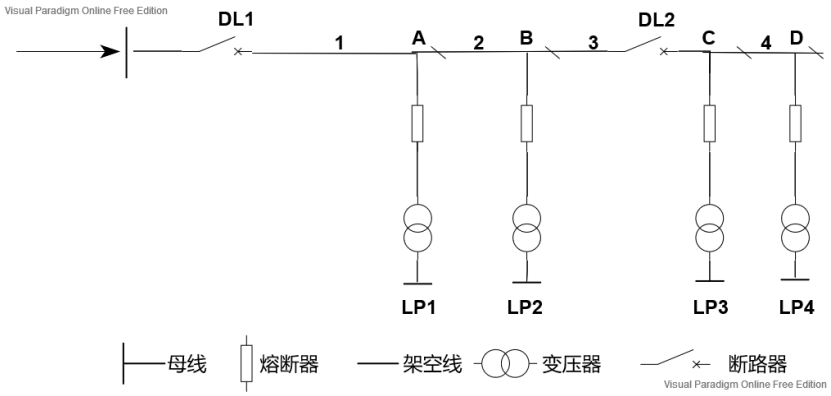

图 9　原始简单电网拓扑图

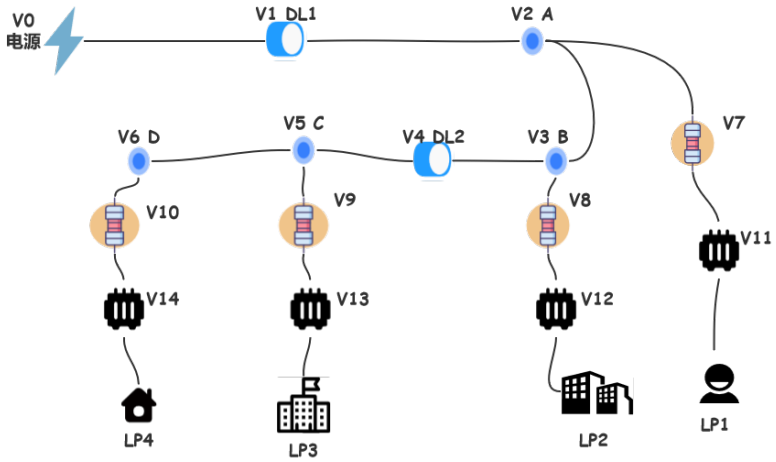

图 10　抽象简化电网图

首先求取每个负荷点的最小路，那么一主馈线上的节点可以分成，非最小路上的节点和最小路上的节点。以图四为例，对于负荷点 LP_2, 其最小路为DL_1, A, B, LP_2

(1) 对于最小路上的节点，采用如下的处理方法。

考虑系统无备用电源的情况下，最小路上的每一个元件的故障或者检修都会引起负荷点的停运。应计算最小路上元件的故障率与检修率之和，以及总停运时间之和。

(2) 对于非最小路上的元件，如果发生故障也会对最小路上的负荷点产生影响，并采用如下方法将非最小路上的影响折算到最小路的节点上：

① 如果在分支线上安装了隔离装置（如隔离开关和分段断路器，熔断器等）。当被隔离的支路发生故障时，会引起其他支路上负荷点停运，并且停运的时间由隔离装置作用时间和备用电源恢复时间共同决定，即其余支路负荷点的停运时间为 max$\{S,T\}$，其中 S 为隔离装置操作时间，T 为备用电源的倒闸时间。并且，分支线上安装了隔离装置情况下，被隔离的支路检修不会导致其余之路停运。结合图 4 为例说明：当支路 a 中存在故障时，负荷点 LP_1 的停运时间为元件修复时间，负荷点 LP_2 的停运时间为 max$\{S,T\}$。如果支路 a 检修，负荷点 2 不会停运。而支路 b 上存在故障或检修，必然导致负荷点 2 停运。

② 如果在主馈线上装有隔离开关或分段断路器，那么隔离开关或者分段断路器后的元件发生故障，隔离开关前的负荷点停运时间影响为隔离开关或者分段断路器的动作时间 S，即当隔离装置后出现故障，隔离装置前的负载也会停运，但是当隔离装置起作用后，隔离装置之前的负载会恢复运作。

③ 当支路不存在隔离装置时，一条支路故障，对其余支路停运时间的影响就是元件修复的时间。

例如，当计算负荷点 2 的停运率和停运时间时，考虑分支线 a 故障对其的影响，可以将 max$\{S,T\}$ 折算到节点 A 上。同理，馈线 3、4 和分支线 c、d 的影响折算到节点 B 上。那么，非最小路上的元件对负载的影响，就转换成了 A、B 的等效可靠性指标，由于 A、B 在从源点到负载 2 的最小路上，则可以依据之前在最小路上的处理方法进行计算。

本文给出的具体计算过程如 5.1.5 举例说明所示。

5.1.4 灵敏度分析

配电网的可靠性水平与连续时空电网拓扑息息相关。而电网拓扑本质上是取决于电网的布线以及元件位置参数、性质参数，进而导致拓扑结构的位置不同。因此，配电网的可靠性灵敏度分析是要对元件、设备的参数进行偏分，从而反应设备参数变化引起的时空拓扑变化对可靠性的影响程度。使用参数[5]用户平均停电时间 $AIHC$ 以及用户平均停电次数 $AITC$ 来计算灵敏度，公式如下：

用户平均停电时间对元件的故障率 λ 的灵敏度:

$$\frac{\partial AIHC}{\partial \lambda_i} = \frac{\sum_{j\in i} r_{ij} N_j}{\sum_{j=1}^{n} N_j} \tag{4}$$

用户平均停电时间对元件的故障时间 r 的灵敏度:

$$\frac{\partial AIHC}{\partial r_i} = \frac{\sum_{j\in i} \lambda_{ij} N_j}{\sum_{j=1}^{n} N_j} \tag{5}$$

用户平均停电次数对元件的故障率 λ 的灵敏度:

$$\frac{\partial AITC}{\partial \lambda_i} = \frac{\sum_{j \in i} N_j}{\sum_{j=1}^{n} N_j} \tag{6}$$

5.1.5 举例说明

以图 9、10 为例进行可靠性计算，其已经是等值之后的最简电网拓扑结构，对其使用 Dijstra 算法得到每个负荷点或者到电源 $V0$ 的最短路长度以及路径如下表所示:

表 2　最短路

顶点	最短跳数	路径
LP1	5	V0 → V1 → V2 → V7 → V11 → LP1
LP2	6	V0 → V1 → V2 → V3 → V8 → V12 → LP2
LP3	8	V0 → V1 → V2 → V3 → V4 → V5 → V9 → V13 → LP3
LP4	9	V0 → V1 → V2 → V3 → V4 → V5 → V6 → V10 → V14 → LP4

表 3 给出了各个元器件的故障率、平均修复时间以及平均动作时间 (用于非最小路等价)，用于计算负荷点的故障指标。

表 3　元件信息

设备名称	故障率/(次 · 台 $^{-1}$)	平均故障持续时间/h	平均动作时间
断路器	0.25	3	0.01
隔离开关	0.25	2.5	0.01
熔断器	0.2	2	0.01
变压器	0.35	4	0.01

将非最小路等价到最小路之后，只需要考虑最小路上的元件信息，沿最小路从电源到每个负荷点，利用元件本身的故障信息、非最小路上等价过来的故障信息以及可靠性计算公式 (2)(3) 计算负荷点的指标，得到最终的负荷点可靠性指标如表 4 所示。

表 4　负荷点可靠性指标

负荷点	λ	U	γ	N(给定用户数)
LP1	1.05	0.80h	1.3125	300
LP2	1.05	0.80h	1.3125	500
LP3	3.30	2.55h	1.2941	150
LP4	3.30	2.55h	1.2941	10

之后，根据表 4 所计算出来的负荷点可靠性信息以及用户数 N, 再对系统的可靠性进行评估, 得到结果如表 5。

表 5　系统可靠性指标

$ASAI$%	$ASUI$ %	$SAIDI$(h(户 · a))	$SAIFI$(次 (户 · a))
0.999695	0.000305	2.675000	0.841667

最后，基于不同的时空拓扑结构进行灵敏度的定性分析和定量计算，如图 11 的拓扑结构，配电网只有一条主干线路，多条分支线路，没有分段开关，所以可以对 8 台配电变压器进行灵敏度定性分析，基于拓扑结构，给出定性分析如表 6。除此之外，我们可以根据公式 (4)(5)(6) 对于不同元件进行偏导，得到定量计算结果。

表 6　灵敏度分析

影响排序	影响负荷点数量	负荷点编号	解释
1	8	a、b、c、d、f	没有熔断器、断路器或其他保护装置
2	1	g	有断路器, 对其他元件影响弱
3	1	e	有熔断器, 对其他元件影响极微弱
4	1	h	有熔断器和断路器, 只影响本身

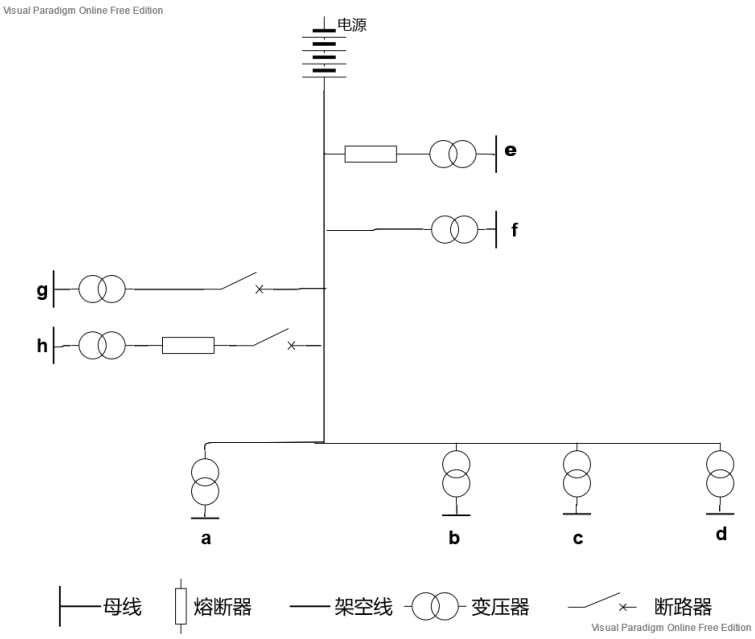

图 11　基于不同元件拓扑的电网图

5.2 问题二

针对本问题，我们采用构建因果网络模型的方法来解决，该模型是图形化的逻辑推理模型，可以用事件节点和带方向的箭头表示。因果网络中事件节点可分为状态节点，征兆节点，假设节点 3 类节点，各类节点间存在着相应的因果关系，因果网络通过节点之间的因果关系建模后进行逻辑推理来解决实际问题。该模型的最显著优点为运算速度快, 鲁棒性强。

5.2.1 配电网的数据采集及监控系统

SCADA 系统是以计算机为基础的 DCS 与电力自动化监控系统。在电力系统中，SCADA 系统应用最为广泛，发展最为成熟，可以对现场的运行设备进程监视和控制，能够实现数据采集、测量、参数调剂以及各类信号报警等功能。

5.2.2 配电网重合闸与永久性故障

自动重合闸装置是将因故障跳开后的断路器按需要自动投入的一种制动装置，电力系统在运行过程中，绝大多数故障都是瞬时性的，永久性故障不到 10%，因此在继电动作切除短路故障后，绝大多数情况下短路处的绝缘可以恢复，因此经过重合闸就可以解决大多数的故障。然而对于永久性故障，对于快速做出故障诊断，尽快隔离故障区，恢复非故障区的供电是有必要的。当故障发生且不能通过重合闸进行解决时，应当采取相应的快速精准的定位故障区域，避免大规模停电，挽回经济损失。因此，配电网的故障检测评价指标可以分成以下 3 种:

1. 实时性：对于网络错误的诊断具有一定的实时性要求。
2. 通用性：当配电网的结构发生变化的时候，算法依旧有效。
3. 容错性能：当系统面临信息缺失时和畸变时，就面临较大的准确性挑战，这就要求算法对如输入信息有一个良好的容错性。

5.2.3 因果网络数学模型

在因果网络中，使用 P_i 来表示动作节点 i，各个节点之间的因果关系由一条自原因指向结果的有向线段表示。

如下图一个简单的因果网络模型：自节点 P_3 指向节点 P_4 的有向线段，表示 P_3 为 P_4 的原因，P_4 为 P_3 的结果。

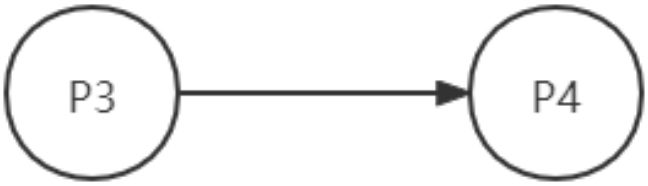

图 12　因果节点示例图

由于一个因果网络中存在较多的因果关系，所以使用规则矩阵 R 来表示整个网络的因果关系。其中 $R[i,j]=1$ 表示 P_j 是 P_i 发生的原因,P_i 是 P_j 发生的结果。若 P_iP_j 无此因果关系, 则 $R[i,j]=0$。即：

$$R[i,j]=\begin{cases}1, & P_j \Rightarrow P_i \\ 0, & else\end{cases} \tag{7}$$

如下图中含有四个节点 P_1,P_2,P_3,P_4 的简单因果网络，通过对各节点之间的因果关系分析，建立的规则矩阵 R 可以表示为：

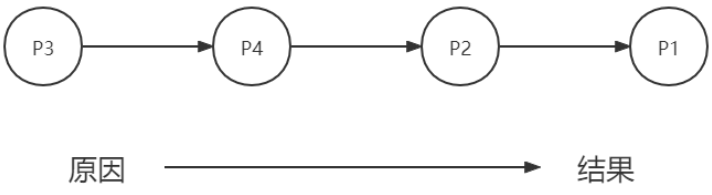

图 13　原因导致结果图

$$R = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} P_1 & P_2 & P_3 & P_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

规则矩阵 R 的转置 R^T 表示为：

$$R^T = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} P_1 & P_2 & P_3 & P_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

R^T 表示从果到因的推导，若 $R^T[i,j]=1$，则表示 P_j 是 P_i 导致的结果,P_i 是 P_j 发生的原因。如上式中第二行中 $R^T[2,1]=1, R^T[2,2]=1$ 表明 P_2 是 P_1 和 P_2 发生的原因。

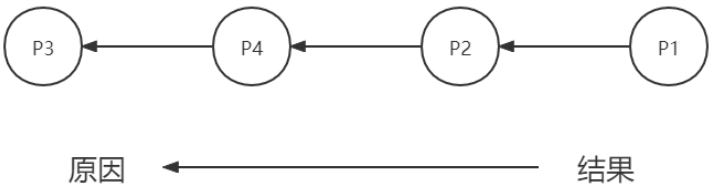

图 14　结果追溯原因图

由此引出本算法的**核心思想: 由已经发生的结果，也就是获得到的警告信息，逆向推导出这些警告信息产生的原因，以定位故障点**。

5.2.4 因果网络算法流程

上文已经提到，本算法的核心思想是利用已经发生的结果，来逆向推导导致这些结果发生的可能原因。也就是说，若电网发生故障，则会导致对应的结果，结果就是各种警告信息。我们可以根据这些警告信息推测出故障发生的可能原因。同时，结合可能发生故障的节点，就能够迅速推断出故障发生的具体位置。整体流程入下图所示：

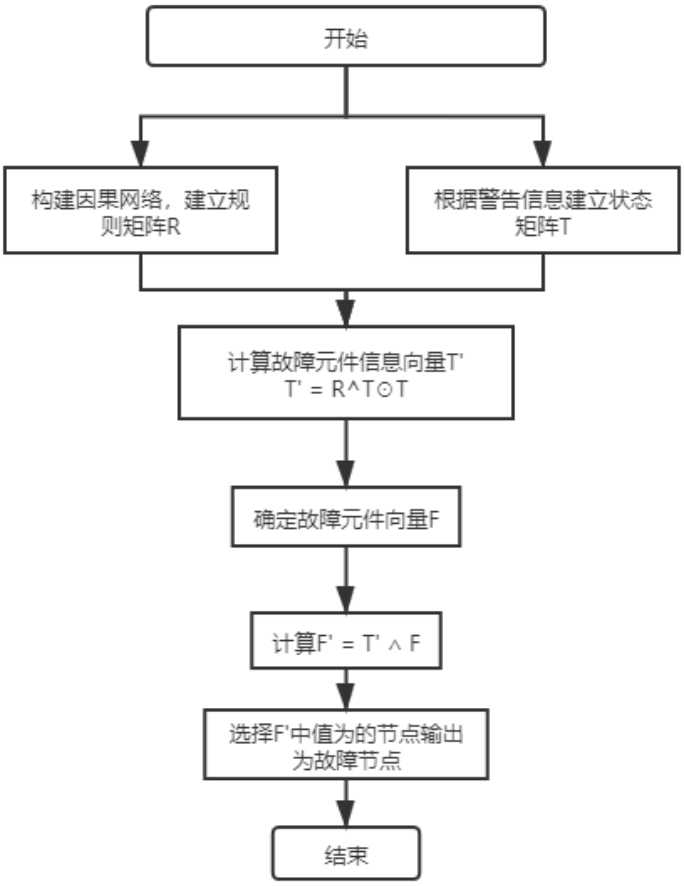

图 15　因果网络算法流程

其中 ⊙ 为逻辑乘运算，⋀ 为逻辑与运算，下文会进行详细介绍。

5.2.5 建立配电网因果模型

在配电网中，可以使用因果关系网络来表示设备保护动作[12]，分段断路器跳闸/拒动和元件（将线路也看作元件）故障之间的因果关系。将这三类信息分别抽象成故障元件节点、保护动作节点、断路器跳闸节点和断路器拒动节点这四种事件节点，且四种节点的因果关系如下所示。

因果关系说明:

1. 元件故障时会触发主保护，导致保护动作发生
2. 触发保护动作，会导致断路器进行跳闸
3. 若断路器拒动，则触发后备保护，导致后备断路器跳闸

以如下简单配电网络为例，说明配电网的故障诊断流程。

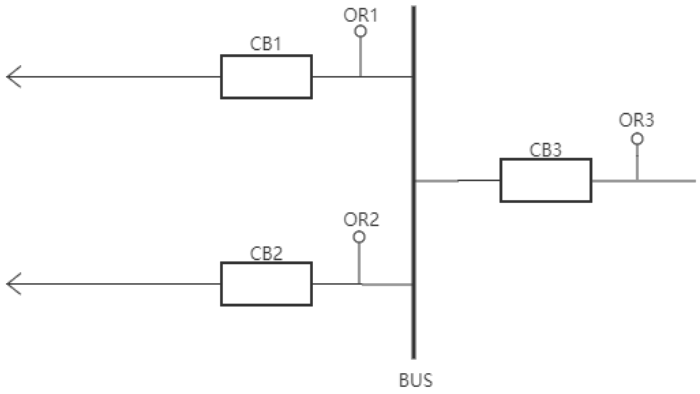

图 16　简单配电网络

该配电网由母线 BUS、支线 L_1, L_2、三段式电流保护器 OR 和断路器 CB 组成。其中 CB_3 为支线 L_1 和 L_2 的后备保护。当 CB_1 或 CB_2 拒动时，后备保护 CB_3 跳闸。电网的监控系统将收集电网中的信息，如三段式电流保护的动作信息，断路器的跳闸动作信息等，用于故障判断的依据，也就是因果网络中的各个结果。

假设线路 L_1 上发生故障，则保护 OR_1 动作触发 CB_1 跳闸。若 CB_1 拒动，则会触发后备保护 OR_3 跳闸，将故障区域隔离。

在上图的示例中存在 3 个可能的故障节点，3 个保护节点。由于 CB1 和 CB2 可能跳闸和拒动，CB3 可能跳闸，因此存在 5 个断路器动作节点。所以，整个配电网络中存在 $3+3+5=11$ 个事件节点

将所有的 11 个事件节点含义列出，如下表所示：

表 7　简单配电网络模型节点含义表

节点	含义	节点	含义
P_1	线路 L_1 故障	P_7	线路 L_2 故障
P_2	保护 OR_1 动作	P_8	保护 OR_2 动作
P_3	断路器 CB_1 跳闸	P_9	断路器 CB_2 跳闸
P_4	OR_1 动作 CB_1 拒动	P_{10}	OR_2 动作 CB_2 拒动
P_5	保护 OR_3 动作	P_{11}	母线 BUS 故障
P_6	断路器 CB_3 跳闸		

根据电网中元件的拓扑关系，我们可以推导出不同元件动作之间的因果关系。根据

这些节点及其之间的因果关系，可以得到整个配电网的简单因果网络，如下图所示：

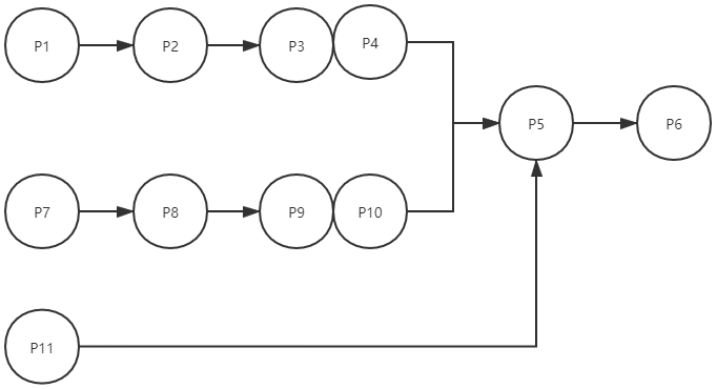

图 17　因果关系网络

之后，依据配电网抽象出的因果关系网络，建立规则矩阵 R。

5.2.6 建立规则矩阵

上文已经对规则矩阵进行了介绍，这个矩阵的作用就是存储不同事件之间的因果关系。若 P_j 是 P_i 的原因，则 $R(i,j)=1$。若 P_j 和 P_i 之间没有因果关系，则 $R(i,j)=0$

由此，可以根据因果关系网络，建立网络的规则矩阵 R。具体数值如下所示：

$$
R = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \\ P_8 \\ P_9 \\ P_{10} \\ P_{11} \end{array}
\begin{array}{c}
\begin{array}{ccccccccccc} P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 & P_{10} & P_{11} \end{array} \\
\left(\begin{array}{ccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array} \right)
\end{array}
$$

5.2.7 建立警告信息向量

警告信息向量 T：反映了故障发生时，所有保护动作和断路器的动作信息。当某一结点 P_i 发生动作时，系统应收到对应的警告信息，则令 $T(i)=1$，否则令 $T(i)=0$。

$$T(i)=\begin{cases}1, & P_i\text{为真}\\ 0, & \text{反之}\end{cases} \tag{8}$$

5.2.8 建立可能故障元件向量

可能故障元件向量 F：向量 F 反映所有可能出现故障的节点，用来缩小故障检测范围。当 P_i 属于可能故障的节点时，$F(i)=1$，否则 $F(i)=0$。

$$F(i)=\begin{cases}1, & P_i\text{属于故障区域节点}\\ 0, & \text{反之}\end{cases} \tag{9}$$

5.2.9 计算故障元件信息向量 T^* 和故障结果向量 F^*

$\odot$ 为逻辑乘运算, 运算举例如下：

$$\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\odot\begin{vmatrix}0\\ 1\end{vmatrix}=\begin{bmatrix}(1\text{ and }0)\text{ or }(1\text{ and }1)\\ (0\text{ and }0)\text{ or }(1\text{ and }1)\end{bmatrix}=\begin{bmatrix}1\\ 1\end{bmatrix}$$

$\bigwedge$ 为逻辑与运算，运算举例如下：

$$\begin{bmatrix}1 & 1\\ 0 & 1\\ 0 & 0\end{bmatrix}\bigwedge\begin{vmatrix}1\\ 0\\ 0\end{vmatrix}=\begin{bmatrix}(1\text{ and }1)\\ (0\text{ and }1)\\ (0\text{ and }0)\end{bmatrix}=\begin{bmatrix}1\\ 0\\ 0\end{bmatrix}$$

在规则矩阵 R 中 $R[i,j]=1$，表明 P_j 是 P_i 的因，P_i 是 P_j 的果，对 R 进行转置，得到矩阵 R^T。在矩阵 R^T 中，$R[i,j]$ 表示 P_j 是 P_i 的果，P_i 是 P_j 的因。因此，在 R^T 中，第 i 行中所有为 1 的位置，表示若 P_i 动作能够产生的所有结果。

上文已经叙述过警告信息向量 T，当 P_i 发生时，$T(i)=1$。可以将 T 与 R^T 的每一行进行逻辑乘运算，得到故障元件信息向量 T^*。原理如下：

例如 R^T 的第一行为 $R^T(1)=[1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0]$, 状态向量 $T=[0\,1\,0\,1\,1\,1\,0\,0\,0\,0\,0]^T$。逻辑乘结果为 $[(0\,and\,1)\,or\,([1\,and\,1])\,or\,([0\,and\,0])\,or\,([1\,and\,0])\cdots\cdots]=1$。这代表着若 P_1 动作会产生的结果中确有发生的，因此 P_1 可能发生。也就是说 P_1 可能是导致警告信息产生的原因 (或许还有其他原因会导致相同的结果发生)，可能是发生错误的地方。本算法的核心思想**通过结果的发生推导出可能原因**在逻辑乘运算中得以体现。

对整个 R^T 来说，若 $R^T(i)\odot T=1$，则 P_i 为可能故障点。

通过 R^T 与 T 进行逻辑乘，得到故障元件信息向量 $T^* = [11011100011]$，即得到了所有警告信息产生的可能原因。将 T' 与之前定义的可能故障元件向量 F 进行对应位置的与运算，筛选错误原因，即可得到电网的故障结果向量 $F^* = [10000000001]$。F^* 中为 1 的位置对应节点即为故障节点, **线路 L_1 和母线 BUS 可能发生故障**。

5.2.10 算法举例

为了验证该故障检测算法的准确性，我们根据一个实际配电网作为算例进行建模。简单分析发现，该系统包含有两台变压器 $T1$、$T2$，两条母线 BUS_1、BUS_2, 以及 5 条配电支路 $L_1 - L_5$。并且还包含线路保护 OR, 母线保护 BR, 变压器保护 TR, 断路器 CB。其中 CB_1 是 CB_2、CB_3 的后备保护，当 CB_2、CB_3 拒动时，CB_1 跳闸。CB_4、CB_5 是 CB_6、CB_7、CB_8 的后备保护。配电网图如下所示

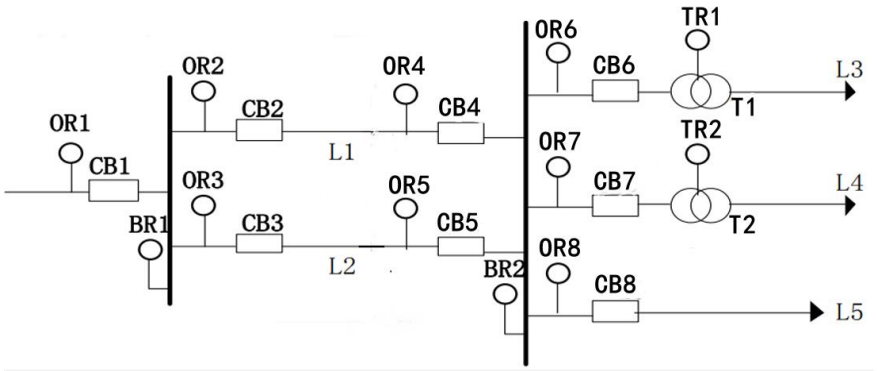

图 18　配电网验证算例

我们假设两处故障分别发生在线路 L_2 和 L_3 上，此时，保护信号 OR_3 , OR_5 , OR_6 动作，CB_3 拒动而导致保护信号 OR_1 动作，后备保护 CB_1 跳闸。断路器 CB_5 可靠动作从而隔离故障。断路器 CB_6 拒动而导致信号保护 OR_4 动作，后备保护 CB_4 跳闸从而隔离故障。故障时各设备保护动作如下:

保护动作: OR_1 , OR_3 , OR_4 , OR_5 , OR_6

断路器跳闸: CB_1 , CB_4 , CB_5

断路器拒动: CB_3 , CB_6

根据配电网保护配置和配电网拓扑结构可知，系统中可能故障的元件有 9 处，可抽象成 9 个故障元件节点。同理，当各元件发生故障时，可能动作的保护和断路器共抽象成 25 个节点。所以本例共 $9 + 25 = 34$ 个节点。建立的配电网因果网络如下图所示:

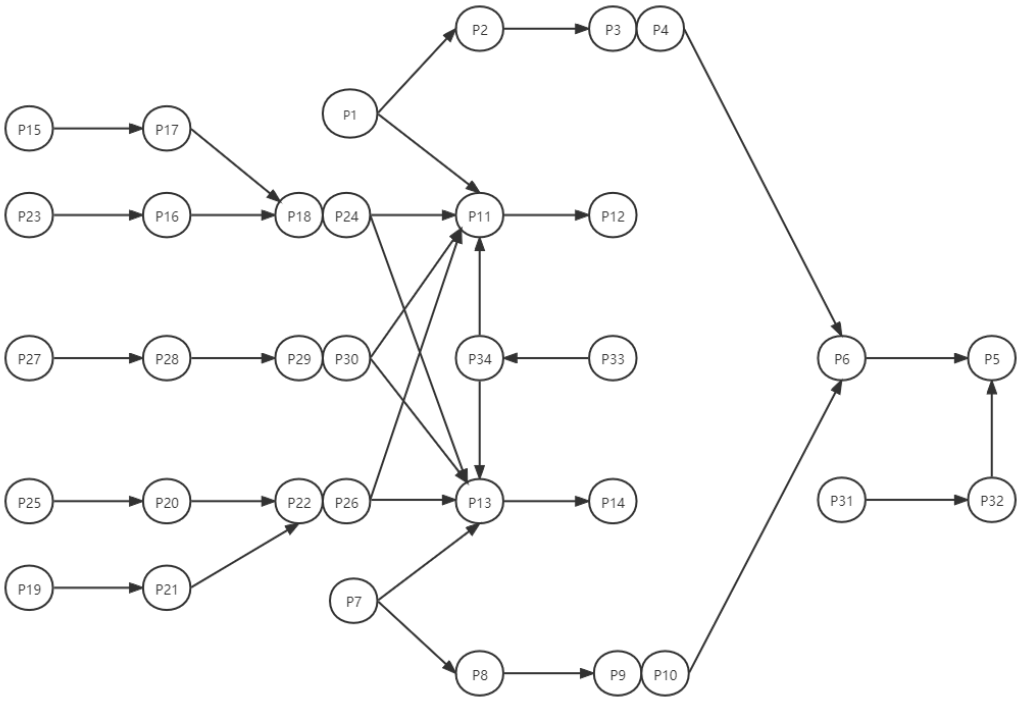

图 19　对应因果网络图

各节点含义如下表所示：

节点	含义	节点	含义	节点	含义
P_1	线路L_1 故障	P_{13}	保护OR_5 动作	P_{25}	线路L_4 故障
P_2	保护OR_2 动作	P_{14}	断路器CB_5 跳闸	P_{26}	OR_7 动作CB_7拒动
P_3	断路器CB_2 跳闸	P_{15}	变压器T_1 故障	P_{27}	线路L_5 故障
P_4	OR_2 动作CB_2 拒动	P_{16}	保护OR_6 动作	P_{28}	保护OR_8 动作
P_5	保护OR_1 动作	P_{17}	保护TR_1 动作	P_{29}	断路器CB_8 跳闸
P_6	断路器CB_1跳闸	P_{18}	断路器CB_6 跳闸	P_{30}	OR_8 动作CB_8 拒动
P_7	线路L_2 故障	P_{19}	变压器T_2 故障	P_{31}	母线BUS_1 故障
P_8	保护OR_3 动作	P_{20}	保护OR_7 动作	P_{32}	保护BR_1 动作
P_9	断路器CB_3跳闸	P_{21}	保护TR_2 动作	P_{33}	母线BUS_2 故障
P_{10}	OR_3 动作CB_3 拒动	P_{22}	断路器CB_7 跳闸	P_{34}	保护BR_2 动作
P_{11}	保护OR_4动作	P_{23}	线路L_3 故障		
P_{12}	断路器CB_4跳闸	P_{24}	OR_6动作CB_6 拒动		

建立规则矩阵 R，网络中共有 34 个节点，R 是一个主对角线均为 1 的 34× 34 阶的方阵，除主对角线外其它非零元素见下表所示。

表 8　规则矩阵 R 中非 0 元素

元素	元素	元素	元素
$R(2,1)$	$R(6,10)$	$R(22,21)$	$R(29,28)$
$R(11,1)$	$R(12,11)$	$R(16,23)$	$R(11,30)$
$R(3,2)$	$R(14,13)$	$R(11,24)$	$R(13,30)$
$R(6,4)$	$R(17,15)$	$R(13,24)$	$R(32,31)$
$R(5,6)$	$R(18,16)$	$R(20,25)$	$R(5,32)$
$R(8,7)$	$R(18,17)$	$R(11,26)$	$R(34,33)$
$R(13,7)$	$R(21,19)$	$R(13,26)$	$R(11,34)$
$R(9,8)$	$R(22,20)$	$R(28,27)$	$R(13,34)$

从提供的保护动作以及断路器的跳闸信息可得到状态向量 T。T 为 34 x 1 的列向量，其中的非 0 元素如下表所示:

表 9　状态向量 T 中非零元素

元素	元素
$T(5)$	$T(6)$
$T(8)$	$T(12)$
$T(11)$	$T(14)$
$T(13)$	$T(10)$
$T(16)$	$T(24)$

故障元件向量 T 中的所有非 0 元素节点如下表所示通过公式 $T^* = R^T \bigodot T$ 计算故障元件信息向量 T^*

表 10　故障设备向量 T 中非零元素

F_i	节点	含义	F_i	节点	含义
$F(1)$	P_1	线路 L_1 故障	$F(25)$	P_{25}	线路 L_4 故障
$F(7)$	P_7	线路 L_2 故障	$F(27)$	P_{27}	线路 L_5 故障
$F(15)$	P_{15}	变压器 T_1 故障	$F(31)$	P_{31}	母线 BUS_1 故障
$F(19)$	P_{19}	变压器 T_2 故障	$F(33)$	P_{33}	母线 BUS_2 故障
$F(23)$	P_{23}	线路 L_3 故障			

最后计算 $F^* = T^* \bigwedge F$, 得到的结果 F^* 中为 1 的节点即为故障节点。利用 $Python$ 计算得到最终 F^* 中 $F^*(1) = 1$, $F^*(7) = 1$, $F^*(23) = 1$。也就是说节点 P_1 , 节点 P_7 , 节点 P_{23} 对应的节点为故障节点。也就是 L_1 , L_2 , L_3 发生故障。

但实际只有 L_2 和 L_3 发生故障。我们认为 L_1 也可能发生故障是因为，保护 OR_4 动作，模型认为可能该动作可能是 L_1 故障导致的。

5.2.11 模型优缺点总结

本模型的核心在于: 若一个事件发生能够导致若干结果，当这些结果中任意一个发生时，我们就认为该事件发生了。

优点：本模型会在定位故障节点的基础上，额外定位一些节点。这样的好处是模型具有一定的鲁棒性，在一些元件失效 (比如警告信息的部分缺失) 时也可以定位到故障节点。

缺点：模型关于故障节点的定位范围太大，给故障修复带来一定的额外工作。

改进：一个简单的修改将会提高模型定位故障点的精度，降低模型的鲁棒性，我们将在模型改进中提出。

5.2.12 模型改进

若能够获取完整，正确的警告信息，推荐使用本改进后的模型。若警告信息可能出现丢失，错误，推荐使用原模型。

核心思想：若一个事件发生能够导致若干结果 (除自身以外)，当**所有**这些结果均发生时，我们才认为该事件发生了。

我们有充分的理由证明该思想的正确性，这里我们使用反正法来证明。前提条件

为：一个事件的发生会导致若干结果的发生。当这些结果中个别发生，如果此时我们认为该事件发生，则该事件的发生又会导致所有结果的发生。因此，前后矛盾，我们证明了该思想的正确性。

修改过程十分简单，我们只需修改故障元件信息向量 T^* 的计算方式即可。

原模型:$T^* = R^T \bigodot T$

现模型:

$$T^*(i) = \begin{cases} 1, & T(i) = 1 \quad or \quad R^T(i) \bigodot T = \sum_{j \neq i} R^T(i,j) - 1 \\ 0, & 反之 \end{cases} \tag{10}$$

$T(i) = 1$ 表明已经得到 P_i 发生的信息。$R^T(i) \bigodot T = \sum_{j \neq i} R^T(i,j) - 1$ 表明我们并未直接获得 P_i 发生的消息，但是若 P_i 发生会导致的结果全部发生，可以判断 P_i 也发生。

利用该模型对上述算法举例进行计算，得到 $F^*(7) = 1$, $F^*(23) = 1$, 即 L_2 , L_3 发生故障, 与实际情况相符，完全正确。

优点：该模型对故障节点的定位十分精准，不会额外定位多余的故障节点。

缺点：对向量 T 的要求较高，需要保证警告信息向量中的信息完整且正确，否则可能无法完全定位所有故障节点。

5.3 问题三

基于动态模式以及时空角度的配电网拓扑结构，自然需要软自愈方案来实现配电网的故障方案解决，并且提前判断故障征兆，从而实现稳定供电；此外，通过软自愈，能够通过实时检测，优化负荷分配，实现负荷转移, 相比硬自愈还能够减少硬件设备的使用，降低成本的同时还能够减少对电网可靠性的负面影响。

5.3.1 建立抽象拓扑结构

依据实际的配电网结构，遵循以下原则将其抽象为拓扑图：

将配电网中的联络开关看作无向边 e，从而得到边集 E；依据开关的开闭状态将 E 分为 $E1$ 以及 $E2$ 两个子边集，其中处于断开状态的开关包含在边集 $E1$ 中，处于闭合状态的开关包含在边集 $E2$ 中。并且，依据开关的闭合与断开动作，使得两个集合的元素可以动态转化。当 $E1$ 中断开的开关闭合时，此开关所对应的边将从 $E1$ 中删除，并加入到 $E2$ 中，反之亦然。

为每一条边 e 赋予一个权值 $Cost_m(I,t)$，表示 t 时刻某边 e_m 的电流值为 I。从而在进行故障判断、故障预知以及负荷调整时，实现基于连续时空拓扑信息 + 电流的判断。

将配电网的开关之间的馈线看作图中的顶点，所有的顶点组成顶点集 V，最后构建得到了一个**加权无向拓扑电网图** $G = (V, E)$。

对于题目三中所给的多分段三联络架空网供电事故场景 (如图 17)，使用上述构建无向图的方法来进行处理，把开关 K_i 看作边 e_i，变电站不变，视为图中一个特殊源点，开关之间的馈线 F_j 抽象成图 G 的顶点 V_j，最后得到的结果如图 18 所示。

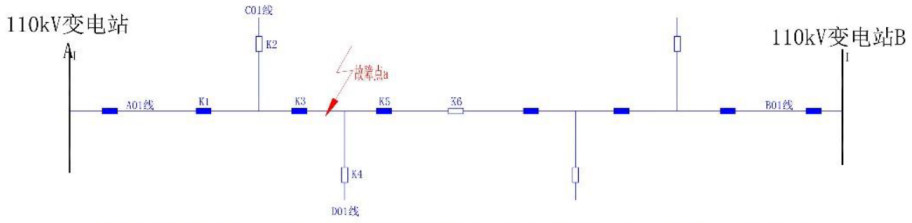

图 20　题目例图

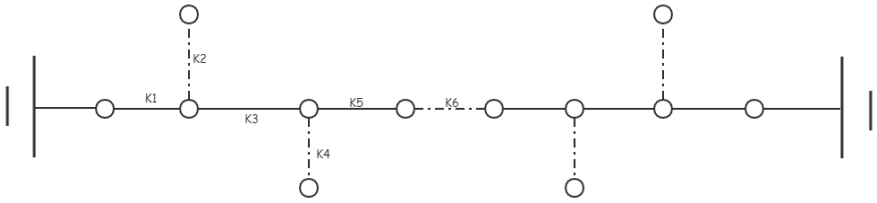

图 21　抽象化题目样例图

5.3.2 故障判断

本文在上述构建的无向拓扑图的基础上，进行故障的判断。首先根据中压配电网的常规电流值给出故障电流阈值 I_{bound}、故障电流增量阈值 I_{Change} 以及优化电流阈值 $I_{PreBound}$。如果超过 I_{bound} 或者偏导超过 I_{Change} 则说明该开关附近电流过大或者瞬间电流增量巨大，可能已经发生了线路故障，造成了电流异常; 如果超过 $I_{PreBound}$ 则说明附近线路负载过大，需要进行负载平衡优化。根据中压配电网特性设置[6] $I_{bound} = 800A$, $I_{Change} = 8000A$, $I_{PreBound} = 600A$。

接着因为电流对应于图上的 $Cost_m(I,t)$，而 $Cost_m(I,t)$ 与时间和电流相关，则我们基于此对故障给出定义:

断路: $Cost_m(I,t) \to 0$

短路: $\frac{\partial Cost_m(I,t)}{\partial t} > I_{Change} \vee Cost_m(I,t_i) > I_{bound}$

他们的含义分别是如果电流趋于 0，则认为断路停电、损失负荷。如果电流值瞬间增大或超过开关跳闸的阀值，代表有短路情况存在。而两种情况都认为是故障情况。

从而可以根据阈值判断出异常的开关，对应于拓扑图上就是边出现了故障，边权异常。而开关不会对电流产生影响，那么如果一旦出现故障馈线，闭合的开关两侧的电流

必然时一样的，开关一侧电流异常另一侧也必然异常，即馈线的故障通过开关具有传递性，那么对应于拓扑图上则有如下结论:

加权无向图G中的边e_i的权值实时统一，即不会出现一条边在某时刻权值不同的情况

基于以上结论，在电网图中易得出以下结论:

若馈线存在故障 $\Longleftrightarrow$ 馈线所直连的所有开关电流值都异常

然后根据图 G 异常电流的边，确定故障点。确定故障点的方法如下: 若点 v_i 的所有属于边集 E_2 的邻接边 e 都是故障的，那可以推断出该点故障。对应于电网图即: 一条馈线所连接的所有开关电流都异常，那么该馈线出现故障。至此即确定出了故障点，定位到了故障馈线，故障定位示例如图?:

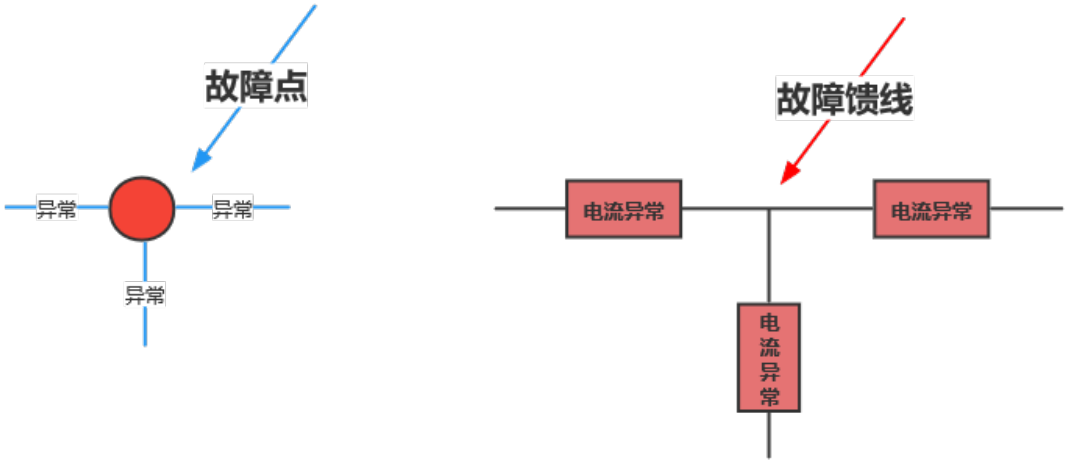

图 22　故障判断示意图

5.3.3 故障隔离

当配电网中，开关上的电流出现异常值后，则判定该开关附近存在出现故障的线路。为了隔离有故障的线路，防止因短路等故障造成的损失过大，应及时将故障线路隔离。因此，应将出现异常电流的开关全部断开。基于上述，对应到配电网的图像拓扑图中，只需要把所有的电流值异常的边 e_i 都从包含所有闭合开关的边集 $E1$ 删除，并重新加入到包含所有断开开关的边集 $E2$ 中，即可实现故障点的隔离操作。

5.3.4 恢复供电

因为故障导致的开关动作会导致除故障点所在的线路外，无故障的线路也有可能断电，这就导致了配电网中的孤岛产生，如下图所示:

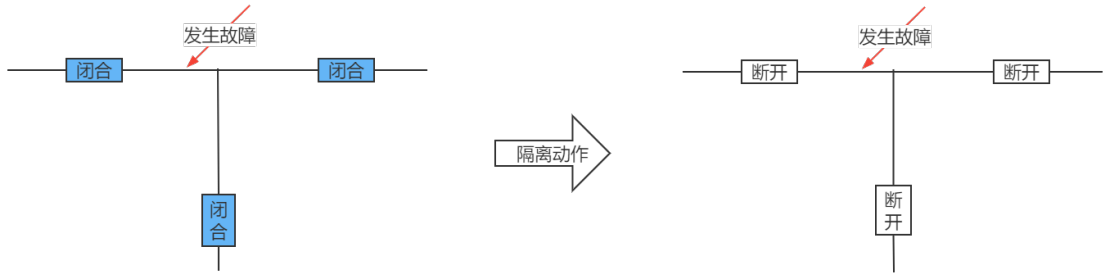

图 23　故障隔离示意图

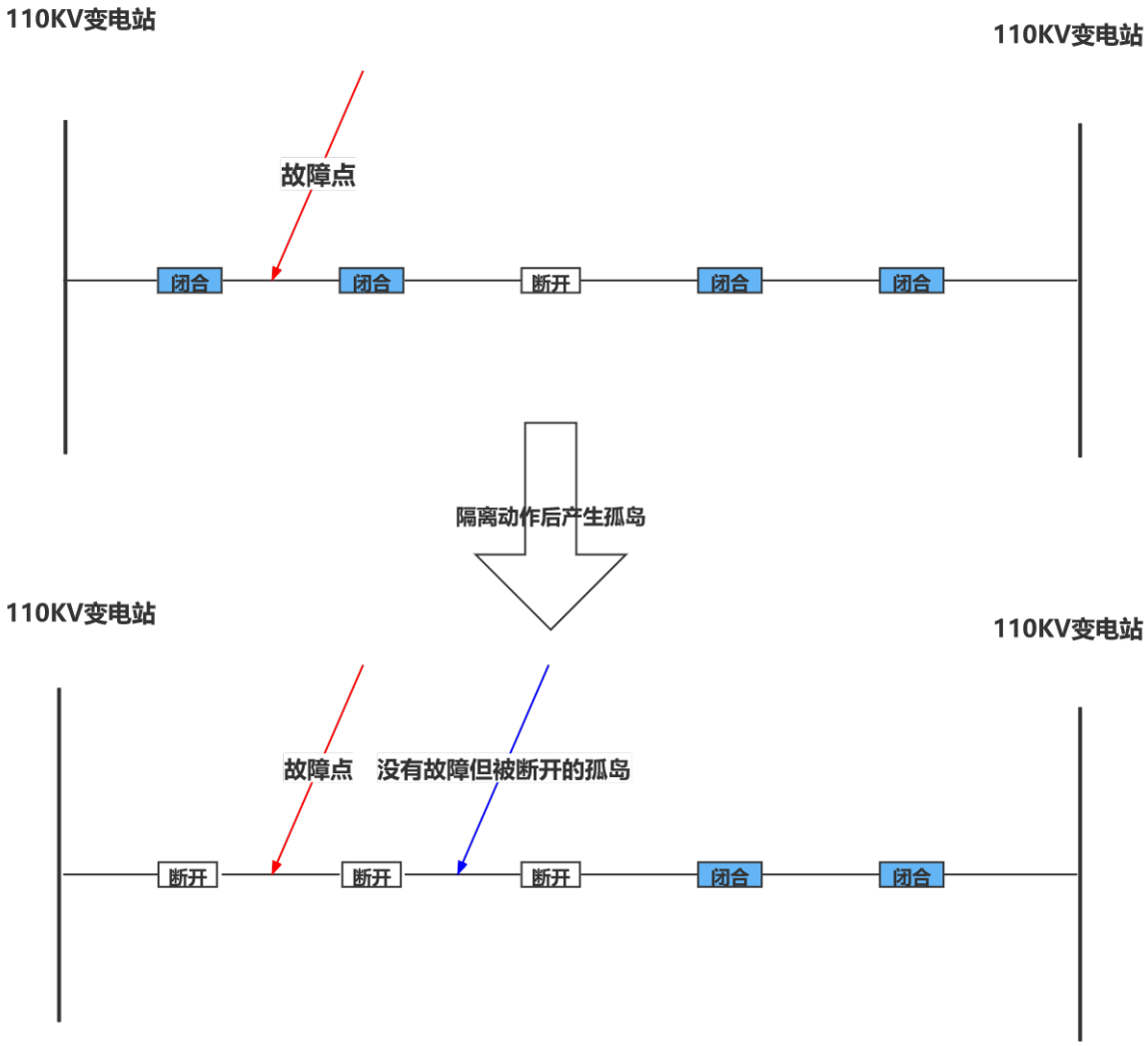

图 24　产生孤岛

依据电网拓扑图来快速判断配电网中是否存在孤岛的方法是:

$$\text{连通分支数目} > \text{变电站个数} + \text{故障点个数} \Longleftrightarrow \text{存在额外无故障段孤岛}$$

由于在电路拓扑图中线路由节点表示，恢复供电就是要将属于孤岛的节点并入无故障的电网，从而使得断电的孤岛恢复供电。通过使用 BFS 算法求解连通分支，将每次

从变电站节点通过 BFS 遍历到的所有点都标记一个相同的标号，拥有相同的标号的节点属于同一连通分支，从而得出所有的连通分支; 那么，我们可以得到所有没有得到供电的无故障线路（即孤岛），遍历孤岛节点的边，选择未闭合的但闭合之后能其够并入正常供电的连通分支的开关 (边)，也就实现了恢复供电! 题目所给的例图的解决方案如图 26 所示:

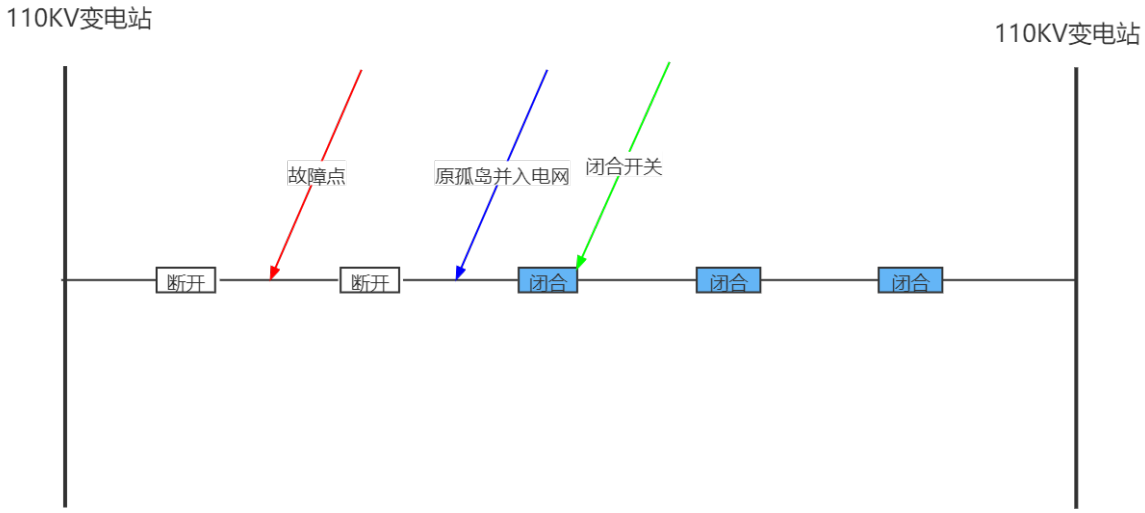

图 25　消除孤岛

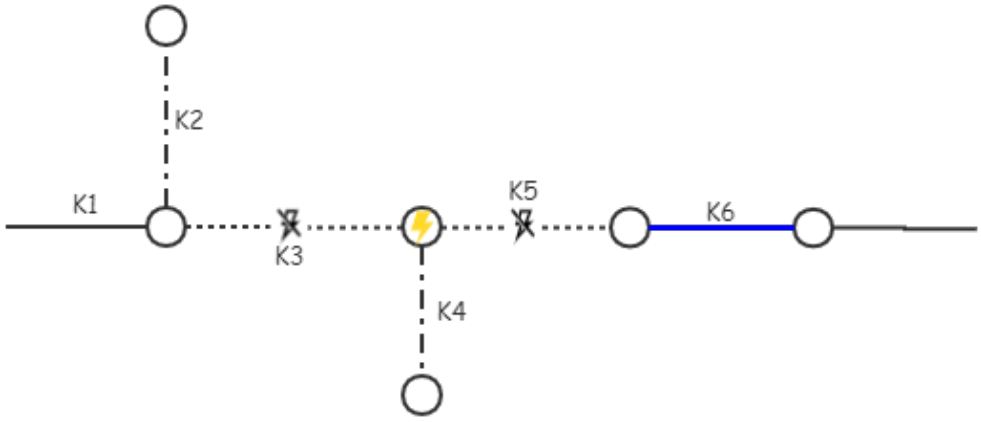

图 26　题目样例的解决方案图

5.3.5 故障预测

在电网运行过程中，开关上经过的电流连续变化，但是可以通过采取电流值分段的方式，将电流值离散化并应用马尔科夫链预测方法进行预测。并考虑到电网中实际的运行状况，在分段的过程中，可以采取不均匀分段的方法。例如：将一个 0-600A 的电流

变化域分段，分为 0-300A、300-450A、450-600A 和 600A 以上，共四段并记为状态 1-4。当电流值超过 600A 时，我们可以认为该开关的电流是处于故障预警状态的。　　在预测过程中，可以每 δt 时刻，对开关的电流值进行采样，在连续时空的条件下，可以基于前 n 个采样点的值，进行下一步预测。构建马尔科夫链转移矩阵 P，记 p_{ij} 是由状态 i 到状态 j 的转移概率，而这概率可以基于连续时空拓扑结构的统计数据来估计。　　举例运算，依据一实际情况的电流统计 (单位 A)，如下：

$$\left\{\begin{array}{cccccccccc} 610 & 480 & 420 & 280 & 620 & 465 & 290 & 285 & 310 & 460 \\ 380 & 260 & 340 & 410 & 630 & 650 & 480 & 460 & 295 & 250 \\ 266 & 488 & 523 & 420 & 235 & 365 & 325 & 385 & 650 & 630 \\ 360 & 420 & 320 & 460 & 290 & 280 & 310 & 610 & 590 & 290 \end{array}\right\}$$

将经过开关的电流分成四段，记录所观测到的状态如下：

$$\left\{\begin{array}{cccccccccc} 4 & 3 & 2 & 1 & 4 & 3 & 1 & 1 & 2 & 3 \\ 2 & 1 & 2 & 3 & 4 & 4 & 3 & 3 & 1 & 1 \\ 1 & 3 & 3 & 2 & 1 & 2 & 2 & 2 & 4 & 4 \\ 2 & 3 & 2 & 3 & 1 & 1 & 2 & 4 & 3 & 1 \end{array}\right\}$$

记 n_{ij} 是由状态 i 到状态 j 的转移次数，第 i 行之和 $n_i = \sum_{j=1}^{4} n_{ij}$, 是系统从状态 i 转移到其他状态的次数。状态转移概率 p_{ij} 的估计值 $\hat{p}_{ij} = \frac{n_{ij}}{n_i}$, 计算得状态转移矩阵得估计为：

$$\hat{P} = \left[\begin{array}{cccc} 2/5 & 2/5 & 1/10 & 1/10 \\ 3/11 & 2/11 & 4/11 & 2/11 \\ 4/11 & 4/11 & 2/11 & 1/11 \\ 0 & 1/7 & 4/7 & 2/7 \end{array}\right]$$

上例中，依据前 40 个采样点的数据，得到得预测结果为：若当前处于状态分别 1、2 和 3 时，下一个状态为 4（故障预警状态）的概率分别为 $\frac{1}{10}\square\frac{2}{11}\square\frac{2}{11}$。我们为在不同状态下向状态 4 转移的概率各自设置不同的可接受阈值，分别为 $\alpha\square\beta\square\gamma$, 且可接受阈值可以根据实际需要进行调整。当从当前非故障状态，向故障状态转移得概率超出可接受阈值时，发出故障预警。例如：设当前状态为 i，故障状态为 j，且转移概率的可接受阈值为 λ，$\hat{p_{ij}} \geq \lambda$，则输出故障预警。

对配电网在连续拓扑时空中得预测效果进行主观分析：当电网中存在固件受损时，对相关节点进行采样时，出现故障预警状态的样本会增大，则向故障状态转移的估计值也会增大，直到超出可接受阈值，发出故障预警会发出故障预警，进行提前检修，避免因为故障的发生而造成断电损失。

综上，通过利用马尔可夫链进行预测的方式，对未来电网中运行状态进行预测，并进行适当得故障预警，提醒及时检修。

5.3.6 成本差异估计

”硬自愈”是依靠自动化系统、采用传感器[7]等设备来实时检测数据，然后在此数据上使用集中式或者分布式专用电子设备装置进行故障位置判断、故障隔离、负荷移动的开关动作逻辑判断，然后发出指令进行开关动作，完成故障“自愈”。通过自动化系统，来实现故障判断进一步控制线路开关完成故障自愈。

软自愈是在自动化系统的基础上采集信息，通过软件算法完成线路故障的判断、隔离、负荷转移，然后再发出开关动作指令，实现故障“自愈”。鉴于使用软件计算，能够依靠更大的算力来提升速度、节约硬件成本还能够实现线路优化、负载均衡、提前预测等功能。

成本差异可以分为设备价格成本、设备维护代价、系统可靠性影响，具体体现如下:

1. 设备价格成本

   硬自愈需要更多的成本去铺设基础设施，像自动化开关、测控装置[9]、主站策略配置[10]、可编程控制器、变频调速设备、接触器[11] 等其他专用电子设备装置。例如:基于故障指示器 (几十元人民币) 终端的运行监视方式需要把故障指示器终端安装在架空线、电力电缆上或安装在箱式变电站、环网柜、电缆分支箱等电力设备中，通过检测故障电流指示故障所在的出线、分支和区段，并通过配置的通信模块将故障信息上传至主站。

   而软自愈则在收到监测数据之后就能够依靠软件算法进行计算，不需要像硬自愈那样那么多的专门化电子设备来进行逻辑计算，仅仅几台电脑就能够运行出结果，就能进行故障判断，从而发出指令，从而减少硬件控制的成本。

2. 设备维护代价

   硬自愈需要日常维护硬件设施的完整程度，需要付出比较大的人力物力成本检修设备，才能保证硬件系统的正常运行，而软自愈除了必要的信息收集的传感器等硬件之外，维护成本主要在软件系统的运营以及软件产品的维护上。

3. 系统可靠性影响

   对于配电系统而言，增加硬件固件例如断路器等硬自愈元件，虽然能够实现故障的判断隔离，但是硬件自身也存在着潜在的故障可能，所以对于整个配电系统而言相当于增加了大量的可故障元件，根据本文问题一的模型，必然会导致某些负荷点以及配电网的可靠性降低，相当于另一种代价成本，即：**硬自愈实现自愈是以降低系统可靠性为代价的**。 而软自愈相比而言增加的硬件是少的，更多依靠软件算法来进行判断，对于系统可靠性的降低影响是微乎其微的。

除此之外软自愈相比于硬自愈还具有很多优点，包括但不限于如下列举:

1. 速度更快

   相比硬件自愈的算力，软件算力更为强大，依靠强大的算力可以更快计算出故障点，而且可以在平时针对电网数据进行预测、分析，给出提前故障征兆等其他有用信息。
2. 拓展性更强

   针对软自愈软件可以开发对应的用户配套软件，让用户实时看到自己所关心的电网信息。除此之外，如果线路拓展等，对应于硬自愈则需要布置新的硬自愈设备，而软自愈则只需增加软件中的仿真节点。
3. 独立性高

   软自愈实现了软硬分离。假如硬自愈系统自身出现了故障，可能是自愈设备也可能是电网线路故障，都是硬件故障，判断故障原因必然会增加复杂度。而软自愈则实现了软件和硬件的独立存在，如果说自愈系统故障，那则只需检查软件系统，而如果说信息采集不到，则是信息采集硬件设备的问题。即他们独立存在，互不干扰。
4. 更大的社会效益

   鉴于其适配用户的软件，可以提升用户日常生活中对于用电的满意度以及与电力系统的友好交流。而更短的停电时间会尽可能的降低因故障停电产生的经济损失，提高企业的经济效益，对社会的发展具有重要意义。

5.3.7 算法的优缺点

优点如下:

1. 算法速度快，复杂度低 $O(|V|+|E|)$，简洁高效。
2. 能够通过软自愈实现一套完整的故障判断、故障隔离、负荷转移、恢复供电。
3. 能够根据电网中运行情况提前判断故障征兆。

缺点如下:

1. 没有更多考虑电网使用中的峰谷、峰顶进行更细致的划分。
2. 马尔科夫链进行预测时，考虑的是过去一段时间的状态，如果瞬间变化很大，不能进行很有效的提前预知。

六、 模型评价与改进

6.1 模型的综合评价

问题一:

针对本问题，本文的指标清晰，对于可靠性评估目标进行了具体的量化分析; 相比于传统的评价模型，本文基于基本的电网结构体系，进行了具体的指标建模，以及图论计算，并非生搬硬套泛化的评价模型。所有问题都基于了配电网拓扑信息，并且可以通过等值法简化配电网，简化了问题，便于统一模型。从 PDO(Power Grid Operation Object) 角度出发，关注一段时间之内的配电网可靠性评估。既考虑了单个负荷点，即某负载的可靠性，也考虑了整个配电系统的可靠性，其中系统里面考虑用户数这一因素，相当于把影响面纳入了模型计算，考虑全面合理。 但也存在部分缺点: 例如对灵敏度分析不够精确等。

问题二:

针对本问题，我们提出的因果检测模型毫无疑问具有合理性，因为配电网的拓扑结构会使元件中的动作产生固定的因果关系。一旦构建出配电网的因果网络，后续计算将完全是矩阵计算，运算速度很快，能够在故障发生的第一时间检测出故障点。在本问题中，我们提出了鲁棒性强，但是检测范围较广的故障检测模型，这个模型可以在配电网故障检测系统信息不准确时，一定程度上检测出所有的故障点。同时，我们还提出了准确度高的故障检测模型，这个模型能够在配电网故障检测系统信息准确时，精准定位所有故障点，效果较好。但是，我们提出的因果检测模型具有一定的局限性，因果网络的初始化较为复杂。

问题三:

此模型能根据开关的电流值，判断电流值是否超出阈值，从而进一步判断电网中是否发生跳闸故障。并且根据电网中的实时电流数据，实时预测故障征兆，以提前预知故障的发生。从多个方面但也存在部分缺点：例如在电网故障判断的过程中，依赖于各个开关的电流输入，以判断开关是否跳闸，对于开关据动等情况没有充分的考虑。对于恢复供电时，没有细致划分对于负载最优的重新分配方案。

6.2 模型的改进与思考

问题一:

可以对具体不同元件等效到最小路的方法再进行细化区分，如具体等效到哪个最小路原点等。可以建立一个三维模型，单个平面代表某时刻下给定的电网拓扑结构，然后 Z 轴代表时间，求出故障点和持续时间的体积，利用一个体积比来进行评价。

问题二:

本模型对于故障点的诊断还有提升空间。例如，若事件 1 会导致事件 2 和事件 3 的

发生，事件 4 也会导致事件 2 和事件 3 的发生。此时，事件 2 和事件 3 同时发生，则模型会认为事件 1 和事件 4 均发生，而不具有分辨能力。但实际情况中，两个事件发生导致结果完全相同的情况并不多，因此本模型仍具有较好的实用性。

问题三:

在故障判断中，应对电流激增和骤降做不同的故障分类。/, 在故障隔离中，或许可以改进先隔离后恢复的方法，直接一次性的开关动作完成故障隔离，不需要恢复，减少影响。/, 在故障预测中，除了故障情况外，电流往往具有周期性，因为每天用电高峰期是同周期的。所以除了使用马尔科夫链，还可以使用时间序列进行一个基于时间的预测，从而得到与时间相关的电流变化情况。

参考文献

[1] 文向南. 基于因果网络的配电网故障诊断研究 [D]. 青岛大学,2017.

[2] 赵华, 王主丁, 谢开贵, 李文沅. 中压配电网可靠性评估方法的比较研究 [J]. 电网技术,2013,37(11):3295-3302.

[3] 别朝红, 王秀丽, 王锡凡. 复杂配电系统的可靠性评估 [J]. 西安交通大学学报,2000(08):9-13.

[4] Billintoo R, Wang P. Reliability-network-equivalent approach to distribution-system-reliability evalation [J].IEEE Proc,1998,145(2):149 153.

[5] 高亚静, 林琳, 刘建鹏. 油田配电网的可靠性与灵敏度分析 [J]. 电力科学与工程,2013,29(12):36-40.

[6] http://www.abcbxw.com/news/8139.html 黔西南州人民政府网—兴义供电局 2018 年第一批 10kv 线路

[7] https://shupeidian.bjx.com.cn/html/20150130/586522-2.shtml 智能电网实现“秒级自愈”

[8] http://www.chinasmartgrid.com.cn/news/20211117/640353.shtml 广州建成全国最大规模的自愈配电网

[9] https://shupeidian.bjx.com.cn/html/20190408/973471.shtml 江苏电网首次实现双线路故障自愈：新一代配电主站系统建设已经全面铺开

[10] https://shupeidian.bjx.com.cn/html/20180711/911892.shtml 国网北京电力成功试点配网故障自愈功能

[11] 李天友. 智能配电网自愈功能及其效益评价模型研究 [D]. 华北电力大学,2012.

[12] 张勇, 张岩, 文福拴, 董明, 孙维真, 黄远平. 基于时序因果网络的电力系统故障诊断 [J]. 电力系统自动化,2013,37(09):47-53.

七、附录

问题一: 计算 C++ 程序:

```
#include<bits/stdc++.h>
#include<vector>
#define MAXN  20
using namespace std;

struct Node{
    int type;
    float rate_b;//损坏率
    float time_fix;//原件每次修复时间
    float time_op;//隔离断路器动作时间
    float RATE;//总损坏率
    float TIME;//总的停运时间
    float RATE_n;//总非最小路损坏率
    float TIME_n;//总的非最小路停运时间
    Node(int t = 3,float rb = 0,float tf = 0,float to = 0){
        type = t;
        rate_b = rb;
        time_fix = tf;
        time_op = to;
        RATE = 0;
        TIME = 0;
    }
};

struct Edge{
    int u;
    int v;
    int nxt;
}e[2*MAXN];
int tot;
int head[MAXN];
int vis[MAXN];
```

```
vector<Node> units;//用于存放各种元件
vector<Node> nodes;//运用存放实际的点
Node u0(0, 0.25,3,0.01);//断路器
Node u1(1, 0.2, 2,0.01);//熔断器
Node u2(2, 0.35, 4);//变压器
Node u3(0,0,0);//连接节点

void init(){


    tot = 0;

    memset(head,0,sizeof(head));
    memset(vis,0,sizeof(vis));

}

void addEdge(int u,int v){
    tot++;
    e[tot].u = u;
    e[tot].v = v;
    e[tot].nxt = head[u];
    head[u] = tot;
}


void addNodes(int num,vector<int> u){
    for(int i=1;i<=num;i++)
    {
        nodes.push_back(u[i]);
    }
}

void shortestPath( ){
```

```
    memset(vis,0,sizeof(vis));
    queue<int> q;
    q.push(0);
    vis[0] = 1;

    while(q.size()){
        int u = q.front();
        q.pop();
        for(int i = head[u];i;i = e[i].nxt){
            int v = e[i].v;
            if(vis[v])
                continue;
            q.push(v);
            vis[v]  = 1;
            nodes[v].RATE = nodes[u].RATE + nodes[u].rate_b;
            nodes[v].TIME = nodes[u].TIME + nodes[u].rate_b*nodes[u].
        }
    }
}

void precondition(int s){

    memset(vis,0,sizeof(vis));
    queue<int> q;
    q.push(s);
    vis[s] = 1;
    while(q.size()){
        int u = q.front();

        q.pop();
        for(int i = head[u];i;i = e[i].nxt){
            int v = e[i].v;
            if(vis[v]) continue;

            nodes[v].RATE_n = nodes[u].RATE
```

```
                + nodes[u].rate_b;
                nodes[v].TIME_n = nodes[v].RATE_n
                * nodes[v].time_op;
                q.push(v);
                vis[v] = 1;
                if(nodes[v].type == 3){
                    nodes[v].RATE += nodes[u].RATE_n;
                    nodes[v].TIME += nodes[u].TIME_n;
                    break;
                }
            }
        }
}


int main(){
    freopen("a.in","r",stdin);
    freopen("a.out","w",stdout);
    int n;
    cin >> n;
    addNodes();
    init();
    for(int i=0;i < n;i++){
            int u,v;
            cin >>u>>v;
            addEdge(u,v);
            addEdge(v,u);
    }
     for(int i  = 0;i < 4;i++){
            int t;
            cin >> t;
            precondition(t);
     }
     shortestPath();
     int N[4] = {10,150,500,300};
```

```
        float SAIFI = 0;
        float SAIDI = 0;
        float ASAI = 0;
        float ASUI = 0;
        for(int i=0;i < 4;i++){
            SAIFI += nodes[15 + i].RATE*N[i];
            SAIDI += nodes[15 + i].TIME*N[i];
        }
        SAIFI = SAIFI /(N[0] + N[1] + N[2] + N[3]);
        SAIDI = SAIDI /(N[0] + N[1] + N[2] + N[3]);
        ASAI = 1 - SAIDI/8760;
        ASUI = 1 - ASAI;
        cout <<"LP1 n " << N[3] << endl;
        cout <<"LP2 n " << N[2] << endl;
        cout <<"LP3 n " << N[1] << endl;
        cout <<"LP4 n " << N[0] << endl;
        cout << "SAIFI :" << SAIFI <<endl;
        cout << "SAIDI :" << SAIDI <<endl;
        cout << "ASAI :" << ASAI <<endl;
        cout << "ASUI :" << ASUI <<endl;

        cout << "LP4 rate:"<<nodes[15].RATE<<" LP4 time:"
        << nodes[15].TIME<<endl;
        cout <<"LP3 rate:"<< nodes[16].RATE<<" LP3 time:"
         << nodes[16].TIME<<endl;
        cout <<"LP2 rate:"<< nodes[17].RATE<<" LP2 time:"
        << nodes[17].TIME<<endl;
        cout <<"LP1 rate:" <<nodes[18].RATE<<" LP1 time:"
        << nodes[18].TIME<<endl;
        return 0;
    }
```

问题二: 故障检测函数

```
    function [F1] = TopoErrorDetect(R,T,F)
    %Inputs:
```

```
    %R: 规则矩阵
    %T: 警告信息向量
    %F: 可能故障元件向量
        %Outputs:
    %F1: 检测结果
T1 = logical(R' * T');%依据输入信息, 逆推可能各章的元件
F1 = logical(F' .* T1);%依据实际, 筛选故障元件
end
```

```
原版故障检测函数
import numpy as np

n = 34
R = np.zeros((n , n))
T = np.zeros((1 , n))
F = np.zeros((1 , n))
result_T = np.zeros((1 , n))

for i in range(R.shape[0]):
    R[i][i] = 1

#读取 T 向量的信息
with open(r"D:\vscode-new\nsy\demo\inputT2.txt" , "r") as f:
    file = f.readlines()
    for line in file:
        position = line.strip("\n")
        position = int(position)
        print("T_position:%d" % position)
        T[0][position - 1] = 1

#读取 F 向量的信息
with open(r"D:\vscode-new\nsy\demo\inputF2.txt" , "r") as f:
    file = f.readlines()
    for line in file:
        position = line.strip("\n")
```

```
            position = int(position)
            print("T_position:%d" % position)
            F[0][position - 1] = 1

    # 读取规则矩阵 R 的信息
    with open(r"D:\vscode-new\nsy\demo\inputR2.txt", "r") as f:
        file = f.readlines()
        for line in file:
            position = line.strip("\n")
            position = position.split(",")
            position_x = int(position[0])
            position_y = int(position[1])
            print("(%d,%d)" % (position_x, position_y))
            R[position_x-1][position_y-1] = 1

    R_T = R.T
    print("T:%s" % T)
    for i in range(R_T.shape[0]):
        print(R_T[i])
        # num_not_zero = np.sum(R_T[i])
        # if T[0][i] == 1:
        #     result_T[0][i] = 1
        # elif num_not_zero > 1 and np.sum(np.multiply(R_T[i], T)) == nu
        if np.sum(np.multiply(R_T[i], T)) > 0:
            result_T[0][i] = 1
        else:
            pass
    print("result_T:%s" % result_T)
    print("F:%s"%F)
    print("res:%s" % np.multiply(result_T, F))
```

```
    改进的故障检测函数
    import numpy as np

    n = 34
```

```
R = np.zeros((n , n))
T = np.zeros((1 , n))
F = np.zeros((1 , n))
result_T = np.zeros((1 , n))

for i in range(R.shape[0]):
    R[i][i] = 1

#读取 T 向量的信息
with open(r"D:\vscode-new\nsy\demo\inputT2.txt" , "r") as f:
    file = f.readlines()
    for line in file:
        position = line.strip("\n")
        position = int(position)
        print("T_position:%d" % position)
        T[0][position - 1] = 1

#读取 F 向量的信息
with open(r"D:\vscode-new\nsy\demo\inputF2.txt" , "r") as f:
    file = f.readlines()
    for line in file:
        position = line.strip("\n")
        position = int(position)
        print("T_position:%d" % position)
        F[0][position - 1] = 1

# 读取规则矩阵 R 的信息
with open(r"D:\vscode-new\nsy\demo\inputR2.txt" , "r") as f:
    file = f.readlines()
    for line in file:
        position = line.strip("\n")
        position = position.split(",")
        position_x = int(position[0])
        position_y = int(position[1])
        print("(%d,%d)" % (position_x , position_y))
```

```
            R[position_x-1][position_y-1] = 1

    R_T = R.T
    print("T:%s" % T)
    for i in range(R_T.shape[0]):
        print(R_T[i])
        num_not_zero = np.sum(R_T[i])
        if T[0][i] == 1:
            result_T[0][i] = 1
        elif num_not_zero > 1 and np.sum(np.multiply(R_T[i] , T)) == num
        # if np.sum(np.multiply(R_T[i] , T)) > 0:
            result_T[0][i] = 1
        else:
            pass
    print("result_T:%s" % result_T)
    print("F:%s"%F)
    print("res:%s" % np.multiply(result_T , F))
```

```
#include<bits/stdc++.h>
#include<algorithm>
#include"queue"

using namespace std;
struct Edge{
    int u,v,w,nxt;
    bool open;//open代表开关是否打开
};
class elecNet{
public:
    Edge * Edges;
    int* head;
    int tot;
    int iBound;//电流上界，超出界限则认为发生故障，下一步将进行跳闸.
    int iPreBound;
    vector<int> lable;
```

```
int s1 ;
int s2;
int n;
int m;
vector<int> brokenNode;
vector<int> vis;
vector<int> alwaysClose;
vector<int> overPre;
vector<int> overPre_1;
vector<int> overPre_2;
vector<int > removeNodes;
void findRemoveNode();
void remove();
elecNet(int n0,int m0,int sa,int sb) {
     n = n0;
     m = m0;
    vis = vector<int>(n+1,0);
    Edges = new Edge[2*m+2];
    head  =  new int [n+1];
    lable = vector<int>(n+1,0);
    lable[sa] = 1;
    lable[sb] = 2;
    s1 = sa;
    s2 = sb;
    tot = 0;
    memset(head , sizeof(head), 0);
    memset(Edges, sizeof(Edges),0);
    for(int i = 0;i <=2*m+1;i++ )
        Edges[i].nxt = 0;
    for(int i = 0;i <= n;i++)
        head[i] = 0;
    iBound = 10;
    iPreBound = 8;
}
```

```
    void addEdge(int u,int v,bool open);//初始化
    void addAlwaysClose(vector<int> a);
    void inputEdge(vector<int>&);//每一个时刻重新输入电流
    void connect();
    void bfs(int s,int flag);
};
void elecNet:: addEdge(int u,int v,bool open){
    tot++;
    Edges[tot].u = u;
    Edges[tot].v = v;
   // Edges[tot].w = w;
    Edges[tot].open = open;
    Edges[tot].nxt = head[u];
    head[u] = tot;
}

void elecNet::inputEdge(vector<int> & a) {
    vector<int> preBroken;
    overPre_1 =  vector<int>() ;
    overPre_2 =  vector<int>() ;
    brokenNode = vector<int>();
    for(int j = 1;j<= m;j++){

        int i = j*2;
        Edges[i].w = a[j];
        Edges[i-1].w = a[j];
        if(a[j] > iBound) {//电流
            Edges[i].open = false;//关闭电闸
            Edges[i-1].open = false;//关闭电闸
            cout << "switch " << j <<": close"<<endl;
            if (find(preBroken.begin(), preBroken.end(),
            Edges[i].u) != preBroken.end())
                brokenNode.emplace_back((Edges[i].u));
            else preBroken.emplace_back(Edges[i].u);
            if (find(preBroken.begin(), preBroken.end(),
```

```
            Edges[i].v) != preBroken.end())
                brokenNode.emplace_back((Edges[i].v));
            else preBroken.emplace_back(Edges[i].v);
        }
        else if(Edges[i].w > iPreBound){
            if(!Edges[i].open)continue;
           if(find(overPre_1.begin(),overPre_1.end(),
           Edges[i].u) != preBroken.end())
               overPre_2.emplace_back(Edges[i].u);
           else overPre_1.emplace_back(Edges[i].u);
            if(find(overPre_1.begin(),overPre_1.end(),
            Edges[i].v) != preBroken.end())
                overPre_2.emplace_back(Edges[i].v);
            else overPre_1.emplace_back(Edges[i].v);
            for(int k = 0;k < overPre_1.size();k++){
                if(find(overPre_2.begin(),overPre_2.end(),
                overPre_1[k]) == overPre_2.end())
                    overPre.emplace_back(overPre_1[k]);
            }
        }
    }
    for(int i = 0;i < brokenNode.size();i++)
    {  cout <<"broken node : "<<brokenNode[i]<<endl;
        alwaysClose.emplace_back(brokenNode[i]);
    }
    return ;
}

void elecNet::connect() {


   for(int i = 0;i <= lable.size();i++)
        lable[i]  = 0;
    lable[s1] = 1;
    lable[s2] = 2;
```

```
for(int i = 0; i< vis.size();i++)
    vis[i] = 0;
bfs(s1, 1);
bfs(s2, 2);
list<int> island;
for(int i=1;i <= n; i++){
    if(!lable[i]) {
        if (find(brokenNode.begin(), brokenNode.end(), i)
         == brokenNode.end())
            island.push_back(i);
    }

}
bool flag = true;
while(flag){
    flag = false;
    for(auto it = island.begin();it != island.end(); it++){
        int u = *it;
        if(find(alwaysClose.begin(),alwaysClose.end(),u)
        != alwaysClose.end()) continue;
        for(int i = head[u];i;i = Edges[i].nxt){
            int v = Edges[i].v;
            if(lable[v]){
                lable[u] = lable[v];//并入电网
                Edges[(i+1)/2*2].open = true;//打开电闸
                Edges[(i+1)/2*2-1].open = true;
                cout << "switch " << (i+1)/2 <<": open"
                <<endl;
                it = island.erase(it);//从孤岛中删除
                flag = true;
                break;
            }

        }
    }
```

```
    }
    return ;
}

void elecNet::bfs(int s,int flag) {
    queue<int> q;
    lable[s] = flag;
    q.push(s);
    vis[s] = 1;
    while(!q.empty()){
        int u = q.front();
        q.pop();
        for(int i = head[u];i;i = Edges[i].nxt){
            int v = Edges[i].v;
            if(vis[v])continue;
            if(lable[v] == 0 && Edges[i].open){
                lable[v] = flag;
                q.push(v);
                vis[v] = 1;
            }
        }
    }
    return ;
}

void elecNet::addAlwaysClose(vector<int> a) {
    alwaysClose = vector<int>(a);
}

void elecNet::findRemoveNode() {
    for(int i = 0;i < overPre.size();i++){
        queue<int > q;
        vector<int> vis(n,0);
        q.push(overPre[i]);
        vis[overPre[i]];
```

```
        while(!q.empty()){
            int u = q.front();
            q.pop();
            int cnt = 0;
            for(int e = head[u];e;e = Edges[e].nxt){
                int v = Edges[e].v;
                if(find(overPre_2.begin(),overPre_2.end(),v)
                 != overPre_2.end()) continue;
                if(!Edges[e].open) continue;
                cnt++;
                q.push(v);
            }
            if(cnt == 0){
                removeNodes.emplace_back(u);
                break;
            }

        }

    }
}

void elecNet::remove() {
    for(int i = 0;i < removeNodes.size();i++){
        int u = removeNodes[i];
        int u1 = u;
        int eClose = 0;
        for(int e = head[u]; e; e = Edges[e].nxt){
            if(Edges[e].open) {
                u1 = Edges[e].v;
                eClose = e;
                continue;
            }
        }
        for(int e = head[u];e;e = Edges[e].nxt){
```

```
            if(Edges[e].open) continue;
            int v = Edges[e].v;
            if(lable[v] == lable[u] || lable[v] == 0)
             continue;
            //并到另一个树上
            int j = (eClose+1)/2*2;
            Edges[j].open = false;
            Edges[j-1].open = false;
            cout <<"cut node "<< u << " from "<<lable[u]
            <<" to "<<lable[v]<<endl;
            cout <<"switch "<<j/2 << ": close"<<endl;
            int k = (e+1)/2*2;
            Edges[k].open = true;
            Edges[k-1].open = true;
            cout <<"switch " <<k/2 <<": open"<<endl;
            lable[u] = lable[v];
            break;
        }

    }
}


int main() {
     freopen("a.in","r",stdin);
    freopen("a.out","w",stdout);
    int n0,m0,s1,s2;
    cin >>n0>>m0>>s1>>s2;
    elecNet en(n0,m0,s1,s2);
    int t;
    cin >>t;
    vector<int> close;
    for(int i = 0; i < t;i ++){
        int tmp;
        cin >> tmp;
```

```
            close.emplace_back(tmp);
        }
        en.addAlwaysClose(close);
        for(int i = 1;i <= m0;i++){
            int u,v;
            bool open;
            cin >> u>>v>>open;
            en.addEdge(u,v,open);
            en.addEdge(v,u,open);
        }

        vector<int> a;
        a.emplace_back(0);
        for(int i = 1;i <= m0;i++){
            int w;
            cin >> w;
           a.emplace_back(w);
         //   a.emplace_back(w);
        }
        en.inputEdge(a);
        en.connect();
        en.findRemoveNode();
        en.remove();
        return 0;
}
//input:
//14 13 1 2
//4 5 7 11 14
//1 3 1
//3 4 1
//4 5 0
//4 6 1
//6 8 1
//6 7 0
//8 9 0
```

```
//9  10  1
//10  12  1
//10  11  0
//12  14  0
//12  13  1
//2  13  1
//1
//2
//0
//11
//11
//0
//0
//2
//2
//0
//0
//2
//3
```