武漢理工大學

数学建模暑期培训论文

第 1 题

基于免疫遗传与改进二阶 Prim 算法的管道铺设优化模型

第 A010 组

姓名	方向
刘子川	编程
程宇	建模
祁成	写作

2025 年 5 月 30 日

摘要

供水管道路线优化对农村供水工程设计和农村供水管理体系改革意义重大。本文建立了基于图论的最小生成树模型，设计改进的二阶 Prim 算法，并引入免疫遗传算法对管道铺设模型进行优化求解。

针对问题一，建立二阶最小生成树模型，并针对该模型设计**二阶 Prim 算法**以求解管道使用总里程最小的铺设方案。本组首先将供水站间的距离关系转化为权重矩阵，先求解一级供水站与中心供水站构成的最小生成树，并在该基础上加入二级供水站，构成二阶最小生成树。求得 II 型管道铺设里程为 **403.40km**、I 型管道与 II 型管道的最小铺设总和为 **524.34km**。

针对问题二，本组在问题一的基础上，设计了**启发式算法**，将两个二级供水站升级为一级供水站，使得 II 型管道铺设里程最小。本组深度分析问题要求，发现在第一问结果基础上，断开最长的 II 型管道，将生成树分成两部分。将不含一级供水站点的生成树的一个二级供水站点升级为一级，并连接至一级生成树。求得需要将序号为 **89** 和 **125** 的两个二级供水站升级为一级供水站，II 型管道使用量为 **392.001km**，相对于问题一方案减少了 **11.403km**。

针对问题三，本组改进了 **Prim 算法**，提出了孤立点的概念，并将其作为惩罚因子嵌入免疫遗传算法，优化求解最少的须升级二级供水站数量。本组首先使用**改进的 Prim 算法**将功率约束条件转化为孤立点约束，之后对二级供水站进行 $0-1$ 编码，引入**免疫遗传算法**，优化计算使得升级的 II 级供水站数量最小的情况下，实现全面供水。求得最少需要升级 **1** 个二级供水站，其编号为 **41**，这种配置下铺设管道的总里程最少为 **548.02km**。最后对求解结果进行灵敏度分析。

本文的优点为：1. 算法基于图论经典算法，易求得全局最优解。2. 基于免疫遗传算法具有较强的鲁棒性，且兼顾了局部搜素能力和全局搜索能力。

关键词： **二阶 Prim 算法　最小生成树　免疫遗传算法　孤立点**

目录

1 问题重述

1.1 问题背景

村通自来水工程是指在现有农村居民饮水安全工程的基础上，通过扩网、改造、联通、整合和新建等措施，把符合国家水质标准的自来水引接到行政村和有条件的自然村，形成具有高保证率和统一供水标准的农村供水网络，基本形成覆盖全县农村的供水安全保障体系，实现农村供水由点到面、由小型分散供水到适度集中供水、由解决水量及常规水质到水量、水质、水压达标等方面的提升，使广大农村居民长期受益，实现我县农村饮水“提质增效升级”的目的[1]。

自来水管道铺设是搭建自来水系统的重要环节，合理的管道铺设方案可以大幅度节约成本。本问题要求在充分考虑市场因素后，研究用两种不同型号的管道铺设该村的自来水管道的方案，使得建设成本降低。由于不同类形的管道的成本不同，且在实际应用中自来水厂有功率限制，研究自来水管的铺设对于村通自来水工程有着重要意义。

1.2 问题概述

围绕相关附件和条件要求，研究两种型号的管道在各自来水厂间的铺设方案，依次提出以下问题:

问题一：设计从中心供水站 A 出发使得自来水管道的总里程最少的铺设方案，并求出该方案下 I 型管道和 II 型管道总里程数。

问题二：由于二型管道数量不足，设计自来水厂升级方案使得两个二级自来水厂升级为一级自来水厂，使得二级管道的使用量尽可能减小。

问题三：考虑自来水厂的功率限制，设计升级方案使得若干的二级自来水厂升级为一级，并求解该情况下的最小铺设总长度。

2 模型假设

(1) 忽略水管的直径、负压能力、耐腐蚀性、工艺等物理参数和单价、材料等经济效益参数，将供水站理想化为图上离散的节点，相邻两供水站之间的水管理想化为边。
(2) 所有供水站和各级管道构成图，在一个理想的平面内。
(3) 每个供水站能连接的供水站个数不受限制，保证用水在满足功率里程要求的情况下的充分供应。
(4) I、II 级供水管道除了连接供水站的区别外没有其他不同。

[1] http://www.lianjiang.gov.cn/zwgk/ljxx/content/post_521033.html.

3 符号说明

符号	说明
T_1, T_2	第一、第二生成树
V_0, V_1, V_2	中心、一级、二级供水站节点集合
v_0, v_1, v_2	中心、一级、二级供水站节点元素
$V(T_1), V(T_2)$	第一、第二生成树的节点集合
$E(T1), E(T_2)$	第一、第二生成树的边集合
$\|E(T1)\|, \|E(T_2)\|$	第一、第二生成树的边的模长
k	供水管类型
$cost_k(v_i, v_j)$	第 k 类供水管中 v_i 和 v_j 节点间的代价
$G(T_1), G(T_2)$	第一、第二生成树的图集合
L_j	v_j 到子树 T 的最短距离
Γ	升级站点的染色体
γ_i	第 i 个孤立点
M	惩罚因子
$\varphi(V_\Gamma)$	惩罚函数
$S(\Gamma_1, \Gamma_2)$	亲和力
g	迭代次数
C	抗体浓度
n	抗体长度
r	随机数种子

注：表中未说明的符号以首次出现处为准

4 问题一模型的建立与求解

4.1 问题描述与分析

问题一要求分析一级供水站与二级供水站的分布情况，并根据分析结果，设计从中心供水站出发总里程最少的管道铺设方案。由于供水必须从中心供水站出发，流入一级供水站，再流入二级供水站。故一级供水站与中心供水站必须优先联连通，再此之基础之上才能铺设二级供水网络。

故本组认为该问题本质是一个带特殊约束的最小生成树问题，即在不考虑二级供水站的情况下，要优先使得一级供水站相互连通。针对该问题，本组先量化了附件中供水站的位置与距离关系，将供水站间的距离关系转化为权重矩阵，并设计二阶 Prim 算法以求解管道使用总里程最小的铺设方案。

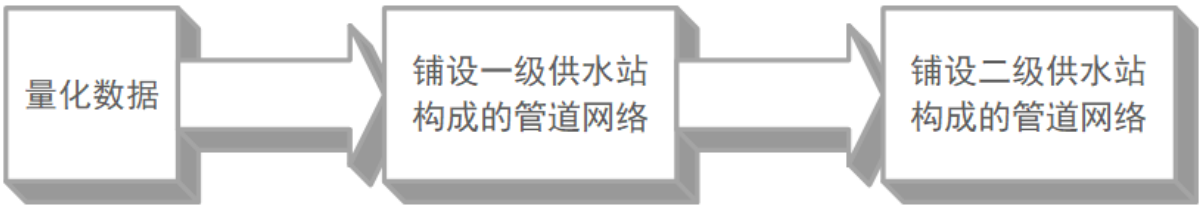


图 **1**　问题一思维流程图

4.2 模型的建立

在村村通自来水工程的连通图 $G=(V(G),E(G))$ 中，每个供水站可以视作一个节点 $v\in V$，即对于中心供水站 $v_0\in V_0$、一级供水站 $v_1(i)\in V_1(i=1,2,\cdots,12)$ 和二级供水站 $v_2(i)\in V_2(i=1,2,\cdots,168)$，有 $V(G)=V_0\cup V_1\cup V_2$。节点间的供水管道看作边 $e\in E(G)$。由中心供水站于一级供水站间可构成生成树[1]：

$$T_1=(V(T_1),E(T_1)),$$

其中 T_1 代表一级生成树，其中 $V(T_1)=V_0\cup V_1$, 即一级生成树中的节点只包含中心供水站和一级供水站。

在一级生成树生成树 T_1 的基础上引入二级站点 V_2 构成二级生成树:

$$T_2=(V(T_2),E(T_2)),$$

其中 $V(T_2)=V(T_1)\cup V_2$, 且满足 $E(T_1)\subseteq E(T_2)$，且在任意生成树 T 中，都有 $|E(T)|=|V(T)|-1$，即边数量为节点数量减一。

边 e 的代价是两节点 $v_a(i_1), v_b(i_2)$ 间的欧式距离, 可表示为：

$$cost_k(v_a(i_1), v_b(i_2)) = |v_a(i_1) - v_b(i_2)|, \tag{1}$$

其中 $a, b = 0, 1, 2$，分别表示中心与一、二级供水站。$k = 1, 2$ 分别表示 I 型与 II 型管道，即有

$$k = \begin{cases} 1, a + b \leqslant 2 \\ 2, a + b > 2 \end{cases} \tag{2}$$

在此二阶最小生成树问题中，已知节点和边的关系为 $|E(T)| = |V(T)| - 1$。即决策变量可表示为生成树的边集合：

$$E(T_k) = \begin{bmatrix} (v_{11}, v_{12}) \\ (v_{21}, v_{22}) \\ ..., ... \\ (v_{|E(T_k)|,1}, v_{|E(T_k)|,2}) \end{bmatrix}, k = 1, 2, \tag{3}$$

其中，$v_{i1}, v_{i2} (i = 1, 2, \cdots, |E(T_k)|)$ 分别代表边 e_i 的端点，且满足 $E(T_1) \subseteq E(T_2)$，即生成树 T_1 优先生成并满足：

$$\sum_{i \in E(T_1)} cost_1(v_{i1}, v_{i2}) = \min \left\{ \sum_{i \in E(T_1)} cost_1(v_{i1}, v_{i2}) \right\}, \tag{4}$$

即必须确保 I 型管道铺设距离最短，即保证一级生成树 T_1 为中心供水站和一级供水站构成的最小生成树。同时，总里程是关于 E_i 的函数。在二级生成树中对 I 型与 II 型管道的铺设距离求总和，可得目标函数即最短铺设总距离为：

$$\sum_{i \in E(T_2)} cost_1(v_{i1}, v_{i2}) + \sum_{i \in E(T_2)} cost_2(v_{i1}, v_{i2}), \tag{5}$$

此问题的约束条件为问题一中的二级供水站生成树优化不能先于一级供水站的最小连通树生成，即对应总模型的数学描述为：

$$\min \left\{ \sum_{i \in E(T_2)} cost_1(v_{i1}, v_{i2}) + \sum_{i \in E(T_2)} cost_2(v_{i1}, v_{i2}) \right\}, \tag{6}$$

$$s.t.\begin{cases} E(T_1) \subseteq E(T_2) \\ \sum\limits_{i \in E(T_1)} cost_1(v_{i1}, v_{i2}) = \min \left\{ \sum\limits_{i \in E(T_1)} cost_1(v_{i1}, v_{i2}) \right\} \end{cases}$$

4.3 模型的求解

4.3.1 二阶 Prim 算法

为求取管道最小里程和最优路径，需要搜索每一层生成树边集合 $E(T_k), k = 1, 2$。针对该二阶最小生成树问题，我们设计了二阶 Prim 算法，分别实现由中心供水站到 I 级供水站、由 I 级供水站到 II 级供水站的最小生成树，即可求得最终的边集合 $E(T_2)$。

二阶 Prim 算法的流程流程图如图 2 所示：

图 **2**　二阶 **Prim** 算法流程图

4.4 实验结果及分析

以中心供水站作为根节点运行二阶 Prim 算法，首先生成由中心供水站与一级供水站构成的供水网络如图 3 所示：

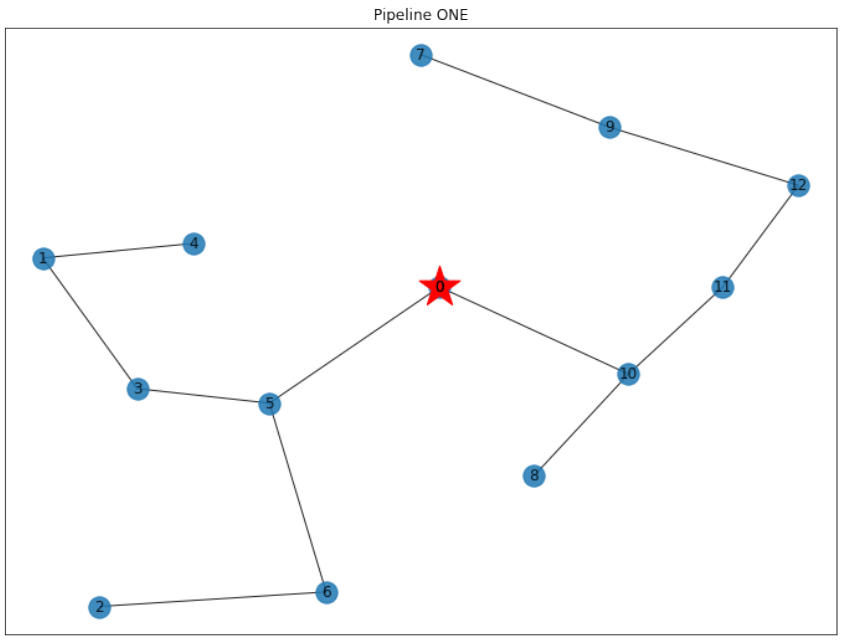


图 **3**　一阶最小生成树

图中 0 节点为中心供水站点，1 12 节点是一级供水站点，其构成的连通图即为其最小生成树。解得所需的 I 型管道的最小铺设距离为 $120.94km$。此时，将二级供水站并入节点集，在该生成树的基础上继续运行 Prim 算法，求解得到整体管道网络铺设如图 4 所示：

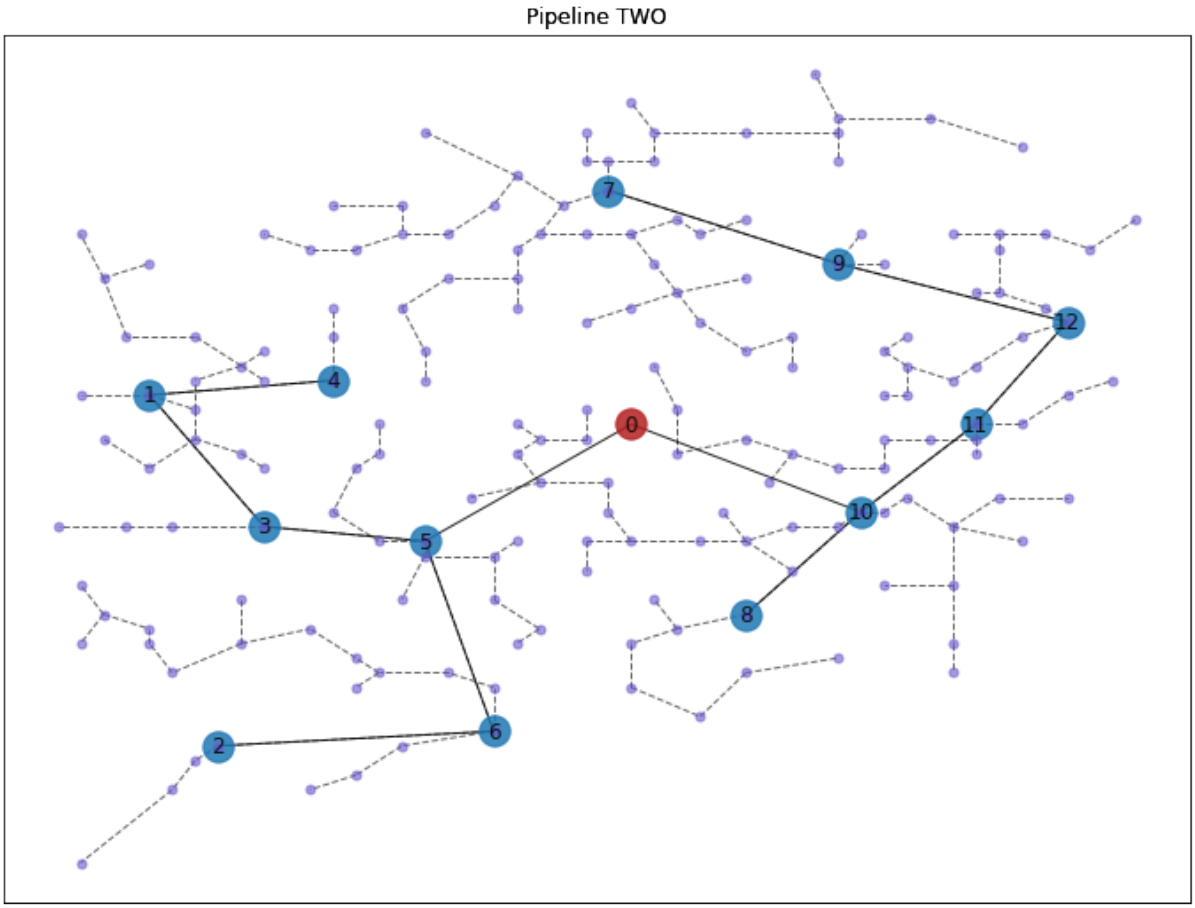


图 **4**　二阶阶最小生成树

图中的 168 个小节点即为二级供水站，解得 II 型管道铺设里程为 $403.40km$、I 型管道与 II 型管道的最小铺设总和为 $524.34km$。二阶 Prim 算法的基本原理基于经典图论算法 Prim 算法，其求解值可被证明为该模型最优解[2, 3]。

5 问题二模型的建立与求解

5.1 问题描述与分析

问题二要求在考虑市场因素的条件下尽可能减少 II 型管道的使用量，即将两个 II 级供水站升级为 I 级供水站，使得 II 级管道里程数最少。本组认为可以基于第一问建立的二阶生成树模型，以需要升级的二级站点为决策变量，并将 II 型管道的总里程作为目标函数构建优化模型。

在深度分析分析生成树优化原理后，我们发现只需在问题一的结果的基础上断开最长的 II 型管道并，并将断开管道后的分割成的两棵树通过一级管道重新连接即可使得 II 级管道铺设里程减小最多，基于此我们设计了启发式算法用于搜索须升级的二级供水站。

5.2 模型的建立

分析问题可知，该模型决策变量可表示为须要升级的二级供水站，即：

$$\theta = \{v_2(i), v_2(j)\}, \tag{7}$$

其中 θ 为决策变量，$v_2(i), v_2(j)$ 分别为将被升级的二级供水站。完成升级操作后根据问题一模型即可得到一级、二级生成树 $T_1' = (V(T_1'), E(T_1)'), T_2' = (V(T_2'), E(T_2'))$ 其中生成树的边集合仍可以表示为：

$$\sum_{i \in E(T_1)} cost_1(v_{i1}, v_{i2}) = \min \left\{ \sum_{i \in E(T_1)} cost_1(v_{i1}, v_{i2}) \right\}, \tag{8}$$

$$E(T_k') = \begin{bmatrix} (v_{11}, v_{12}) \\ (v_{21}, v_{22}) \\ \dots, \dots \\ (v_{|E(T_k')|,1}, v_{|E(T_k')|,2}) \end{bmatrix}, k = 1, 2, \tag{9}$$

其仍必须满足 $E(T_1') \subseteq E(T_2')$，目标函数为 II 型管铺设总里程，即：

$$\sum_{i \in E(T_2)} cost_2(v_{i1}, v_{i2}). \tag{10}$$

结合上述决策变量、目标函数、约束条件，得到以将两个升级的一级供水站为决策

变量，以 II 级管道的总里程为目标函数，生成树 T_1 优先生成为约束条件的优化模型：

$$min \sum_{i \in E(T_2)} cost_2(v_{i1}, v_{i2})$$

$$s.t. \begin{cases} E(T_1') \subseteq E(T_2') \\ \sum_{i \in E(T_1')} cost_1(v_{i1}, v_{i2}) = min \left\{ \sum_{i \in E(T_1')} cost_1(v_{i1}, v_{i2}) \right\} \end{cases}$$

5.3 模型的求解

深度分析问题二要求可知，问题二目标仅需要减小 II 级管道的使用量，当某一个二级供水站升级为一级供水站后：

$$|V(T_1')| = |V(T_1)| + 1,$$
$$|V(T_2')| = |V(T_2)|,$$

即一级生成树的节点数加一，且二级生成树的节点数不变。且根据式 $|E(T)| = |V(T)| - 1$ 可知，I 型管道数量加，同时 II 型管道数量减一。由此本组设计启发式算法，在问题一结果的基础上，断开最长的二级管道，将原生成树分割为两个生成树。分割后的两个生成树中必有一个不包含一级供水站，将该生成树中的任意一个二级站点升级为一级站点并连接该站点与一级供水网络，即可使 II 型管道的铺设减小量最大化，其流程图如下图 5 所示

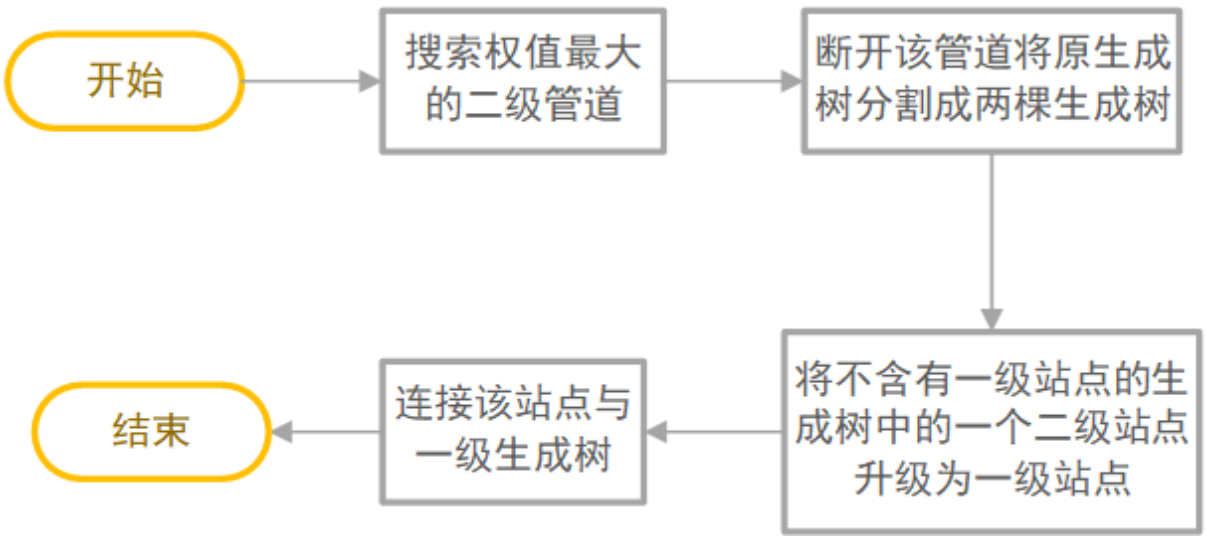


图 **5**　启发式算法

5.4 实验结果及分析

连续两次执行启发式算法后，升级后的管道网络如图 6 所示

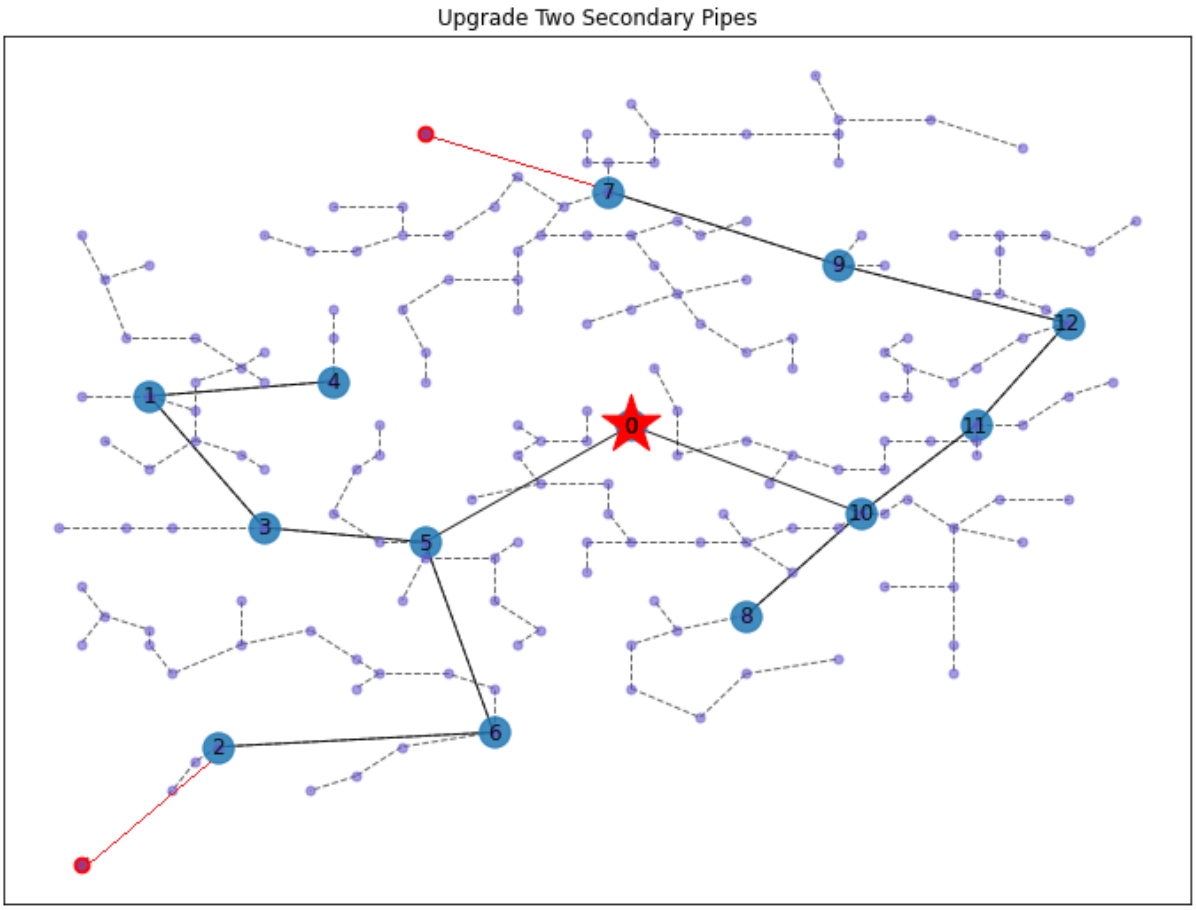


图 **6**　升级后的二阶最小生成树

图中红点表示升级为一级供水站的二级供水站，红线表示新增设的 I 型管道。其中 II 型管道使用量为 392.001km，相对于问题一方案减少了 11.403km。

6 问题三模型的建立与求解

6.1 问题三描述与分析

问题三在基础管道铺设的条件下考虑了供水站的功率限制，要求针对该限制重新制定管道铺设方案。在深入分析了模型三新增的约束条件后，本组针对其新增约束改进了二阶 Prim 算法，以求解该类带约束的生成树问题。

由于该问题的决策变量本身就为 $0-1$ 向量，很适合使用遗传算法求解。故我们引入了免疫遗传算法，搜索最佳升级方案，使得在升级二级供水站数量最少的情况下，实现对所有的供水站供水。

6.2 模型的建立

问题三中，将 168 个二级供水站转化为按顺序构成的 $0-1$ 行向量作为决策变量

$$A=\begin{bmatrix} a_1 & a_2 & ... & a_{168} \end{bmatrix}, \tag{11}$$

其中，$a_i=1(i=1,2,...,168)$ 表示将第 i 个 II 级供水站升级成 I 级供水站；$a_i=0$ 表示第 i 个 II 级供水站不升级。根据题目要求，优化目标为是需要升级的二级供水站个数，即

$$min\sum_{i=1}^{168}a_i,$$

在问题一的基础上，问题三中的功率条件限制了从 I 级供水站输送的总里程最大为 $40km$，即约束了

$$\sum_{e_i\in E(T_{2k})}|e_i|\leq 40,$$

其中 T_{2k} 是以一级站点 $v_1(k)$ 为根节点所衍生出的二阶生成树，$E(T_{2k})$ 是该分支中边的集合。结合前两问与上述决策变量、目标函数、约束条件，得到以二级供水站序列向量为决策变量，以需要升级的二级供水站个数为目标函数，以功率限制等为约束条件的优化模型：

$$\min\sum_{i=1}^{168}a_i, \tag{12}$$

$$s.t.\begin{cases} E(T_1)\subseteq E(T_2) \\ \sum_{i\in E(T_1)} cost_1(v_{i1},v_{i2}) = \min\left\{\sum_{i\in E(T_1)} cost_1(v_{i1},v_{i2})\right\} \\ \sum_{i\in E(T_2)} cost_1(v_{i1},v_{i2}) + cost_2(v_{i1},v_{i2}) = \min\left\{\sum_{i\in E(T_2)} cost_1(v_{i1},v_{i2}) + cost_2(v_{i1},v_{i2})\right\} \\ \sum_{e_i\in E(T_{2k})} |e_i| \le 40 \end{cases}$$

6.3 模型的求解

在问题一的基础上分析得出，要实现全面供水，必须消除所有无法与其他供水站相连的孤立点。本组对二级供水站进行编码，对无法满足约束条件的二级供水站附加一个与孤立点个数成正比的罚函数，定义惩罚因子为常数，并引入免疫遗传算法，通过优化需要升级的二级供水站个数最小，实现全面供水。

6.3.1 改进二阶 Prim 算法

阵对带约束的生成树问题，本组在二阶 Prim 算法的基础上加入边权重判定机制，即将某一个解纳入生成树后，若权重和超过了限制条件则退回到上一步，并将该路径的边权值修改为无穷大。在此基础上，若某个二级节点的所有边权重值都被修改为无穷大则将其定义为孤立节点。

改进 Prim 算法的流程图如下图所示：

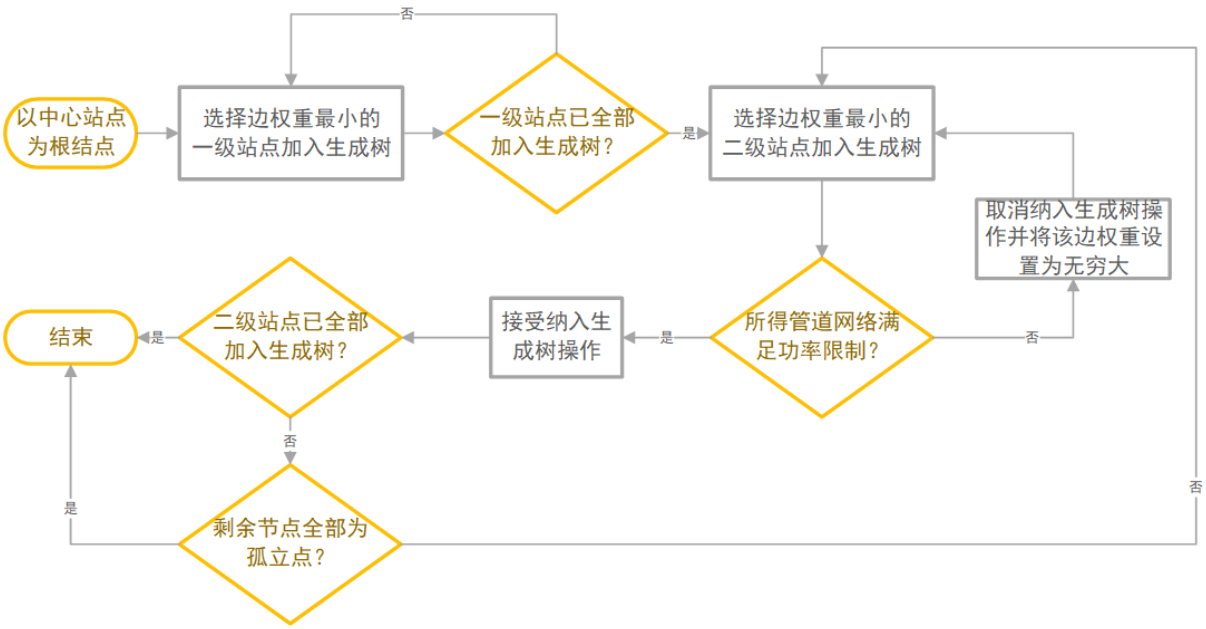


图 7　改进二阶 **Prim** 算法流程图

6.3.2 免疫遗传算法

0-1 编码 执行改进二阶 Prim 算法，即可求得所有未能被供水的节点 $\left\{v_i^{aban}\right\}$，其中 $i=1,2,\cdots,n$，n 表示孤立节点数量，即染色体可对应编码为：

$$\Gamma=[\gamma_1,\gamma_2,\cdots,\gamma_n], \tag{13}$$

其中 Γ 表示升级站点的决策变量，即染色体。$\gamma_i=\{0,1\}$ 为孤立节点 v_i^{aban} 对应的决策值，当 $\gamma_i=0$ 时表示函数值孤立节点 v_i^{aban} 不做修改；当 $\gamma_i=1$ 时表示 v_i^{aban} 对应的二级供水站被升级为一级供水站。

变异与交叉 在第 g 次迭代中，将染色体间两两配对, 对其执行断点交叉[4, 5]，对其中一组染色体

$$\begin{aligned}\Gamma_i&=[\gamma_1^i,\gamma_2^i,\cdots,\gamma_p^i,\gamma_{p+1}^i,\cdots,\gamma_q^i,\cdots,\gamma_n^i],\\ \Gamma_j&=[\gamma_1^j,\gamma_2^j,\cdots,\gamma_p^j,\gamma_{p+1}^j\cdots,\gamma_q^j,\cdots,\gamma_n^j],\end{aligned}$$

执行交叉操作，交换其中随机的一段基因片段 $\{\gamma_p,\gamma_{p+1},\cdots,\gamma_q\}$ 得

$$\begin{aligned}\Gamma_i&=[\gamma_1^i,\gamma_2^i,\cdots,\gamma_p^j,\gamma_{p+1}^j,\cdots,\gamma_q^j,\cdots,\gamma_n^i],\\ \Gamma_j&=[\gamma_1^j,\gamma_2^j,\cdots,\gamma_p^i,\gamma_{p+1}^i,\cdots,\gamma_q^i,\cdots,\gamma_n^j],\end{aligned}$$

其中 p,q 为随机数，且有 $0\leqslant p<q\leqslant n$。且生成的任意染色体 Γ_i，都存在 φ 的概率执行变异操作，此时选择其中随机位置的基因 γ_r^i 执行变异操作，使得

$$\gamma_r^i=\overline{\gamma_r^i},$$

其中 r 为随机数，且有 $0\leqslant r\leqslant n$。

适应度计算 将决策变量 Γ 所表征的升级操作执行后[6]，即可得到升级后点集 V_Γ，将其带入改进后的二阶 Prim 算法可求得生成孤立点数量 $\varphi(V_\Gamma)$。若 $\varphi(V_\Gamma)>0$ 则说明任然存在孤立节点，即没有实现完全供水。基于此我们将 $\varphi(V_\Gamma)$ 作为惩罚因子代换功率约束条件，即决策变量 Γ 的抗原适应度为：

$$F(\Gamma_i)=\sum_{j=1}^{n}\gamma_j+M\cdot\varphi(V_{\Gamma i}), \tag{14}$$

其中 $\sum_{i=1}^{n}\gamma_i$ 表示升级的二级站点数量和。M 为较大的正数，与 $\varphi(V_\Gamma)$ 构成罚函数。同时计算抗体亲和力：

$$S(\Gamma_1,\Gamma_2)=(\frac{k(\Gamma_1,\Gamma_2)}{n}>T), \tag{15}$$

$k(\Gamma_1,\Gamma_2)$ 表示染色体相同的位数，n 为抗体长度。T 为可设定的阈值，即当相同的位数的比例大于 T 时，$S(\Gamma_1,\Gamma_2)=1$；否则 $S(\Gamma_1,\Gamma_2)=0$。抗体与抗体之间的亲和力反映了抗体之间的相似程度。定义抗体浓度 $C(\Gamma_i)$ 表示群体中与 Γ_i 相似抗体所占的比例, 即可计算如下：

$$C(\Gamma_i)=\frac{\sum_{j=1,j\neq i}^{N}k(\Gamma_i,\Gamma_j)}{N}, \tag{16}$$

即综合适应度可表示为：

$$\min P(\Gamma_i)=\lambda\frac{F(\Gamma_i)}{\sum_{i=1}^{N}F(\Gamma_i)}+(1-\lambda)\frac{C(\Gamma_i)}{\sum_{i=1}^{N}C(\Gamma_i)}, \tag{17}$$

精英库保留策略　先将染色体 $\{\Gamma_i\}$ 以抗原适应度 $F(\Gamma_i)$ 大小降序排列，并优先保留其前 α 个染色体进入下一代，由此防止目标函数值最优的个体被舍去。之后，将剩余染色体以综合适应度 $P(\Gamma_i)$ 排序，并将前 $N-\alpha$ 个染色体放入下一代。

免疫遗传算法流程图如图 8 所示：

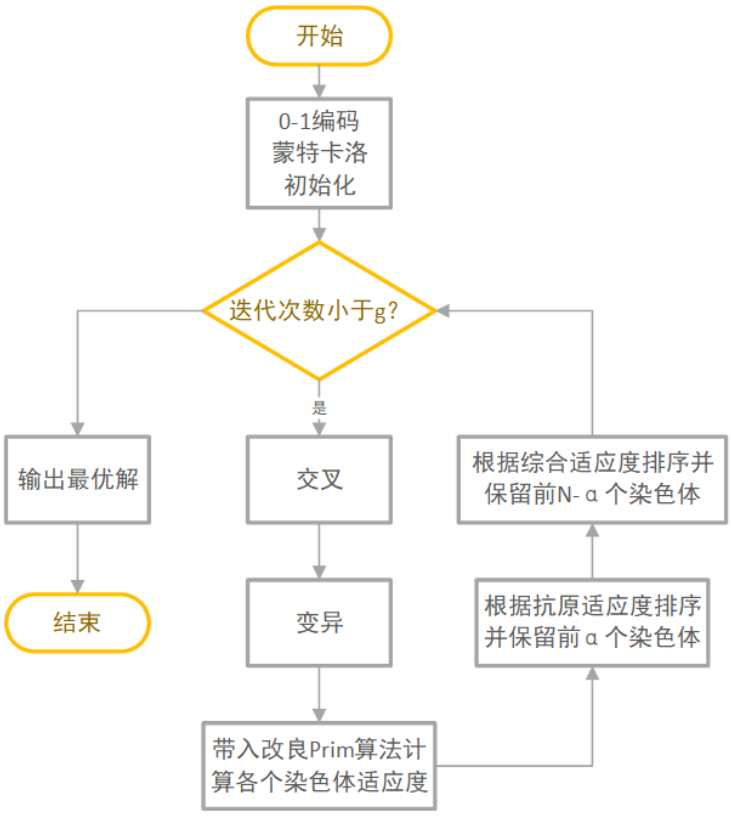


图 **8**　免疫遗传算法流程图

6.4 实验结果及分析

利用改进 Prim 算法求出最大总里程为 40km 时候的孤立点如图9所示，其中紫色方块表述孤立点：

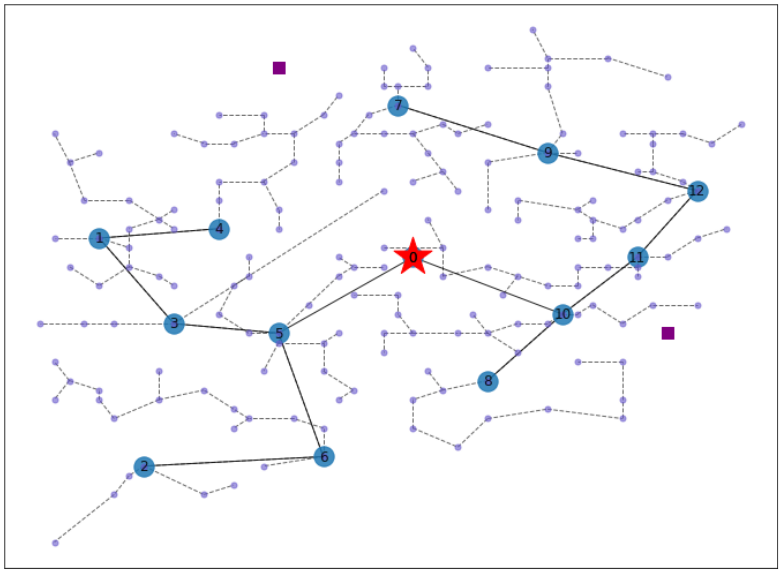


图 **9**　改进 **Prime** 算法求解孤立点

每个一级管道对应的二极管道树枝总里程如表所示：

I 级管序号	树枝总里程	I 级管序号	树枝总里程
中心管	0	7	39.953km
1	38.958km	8	36.067km
2	17.289km	9	37.092km
3	28.798km	10	38.022km
4	37.527km	11	37.097km
5	39.263km	12	38.381km
6	37.728km		

将二级管道生成的染色体 Γ 带入免疫遗传算法中，求解出最小升级管道数为一个，其 x 序号为 41，再次利用改进 Prim 算法求出 I 级、II 级管道最短所需要的历程之和为 548.02km, 这种配置下铺设管道的总里程数最少。其管道分布图如下：

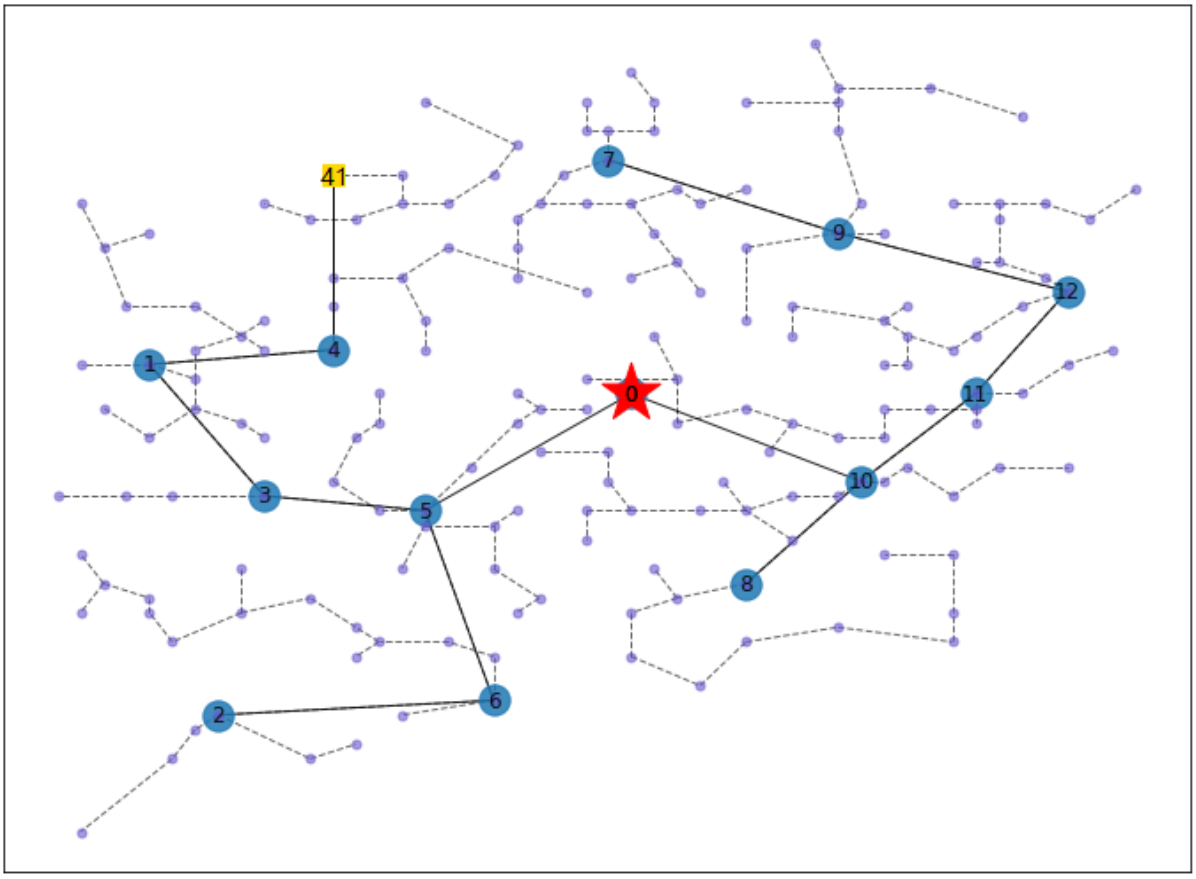


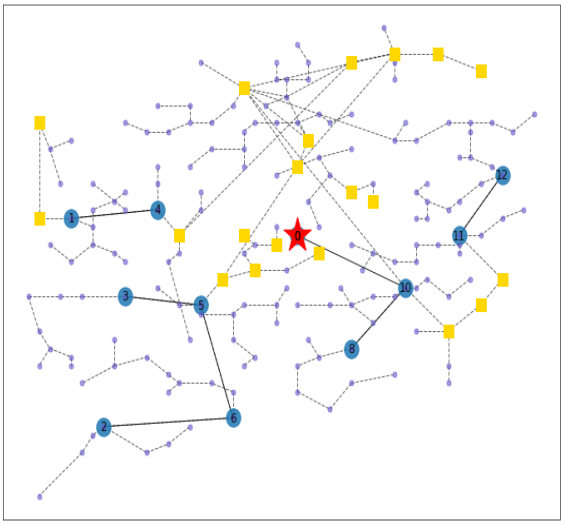
图 **10**　升级后的二级管道分布图

每个一级管道对应的二极管道树枝总里程如表所示：

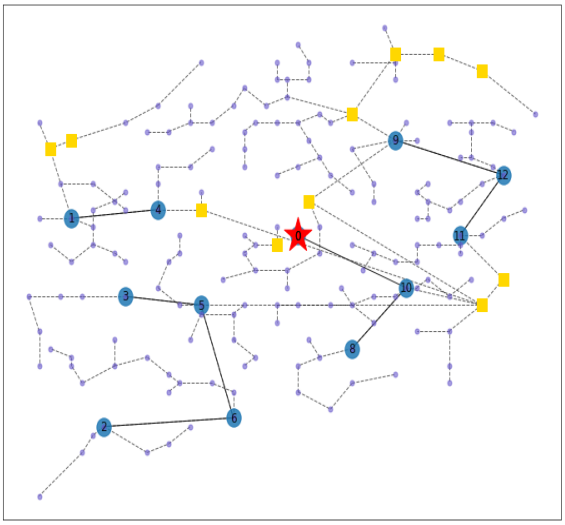
I 级管序号	树枝总里程	I 级管序号	树枝总里程
中心管	0	7	39.953km
1	38.957km	8	36.067km
2	17.289km	9	38.331km
3	9.0km	10	38.022km
4	37.527km	11	37.097km
5	39.263km	12	38.381km
6	37.728km	41	32.634km

7 灵敏度分析

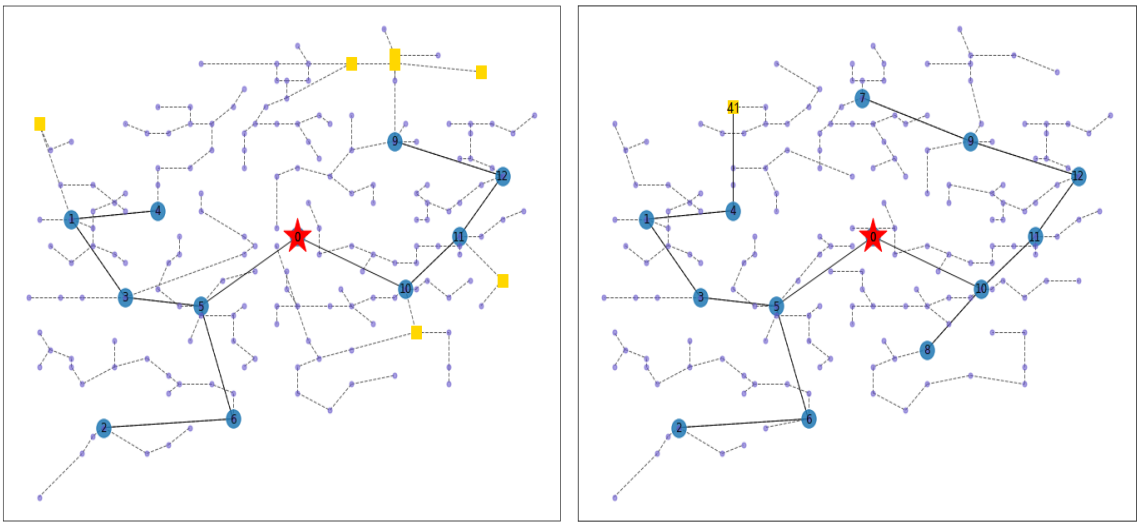
由于题目中铺设供水代价最大 40 公里，只能升级一个管道。若降低供水最大长度，使用遍历算法的时间复杂度为 $O(2^n)$，n 为二级管道个数。而采用免疫遗传算法可以大大降低时间复杂度，搜索升级的最优管道个数。至此，将二级管道铺设最大代价取值为 25、30、35、40km 时，得到结果分别如图 11 所示：



(a) 最大 25km，升级 20 个水管



(b) 最大 30km，升级 11 个水管



(c) 最大 35km，升级 7 个水管　　(d) 最大 40km，升级 1 个水管

图 **11**　不同二级管道铺设最大代价

其免疫遗传算法的算法收敛图如图所示：

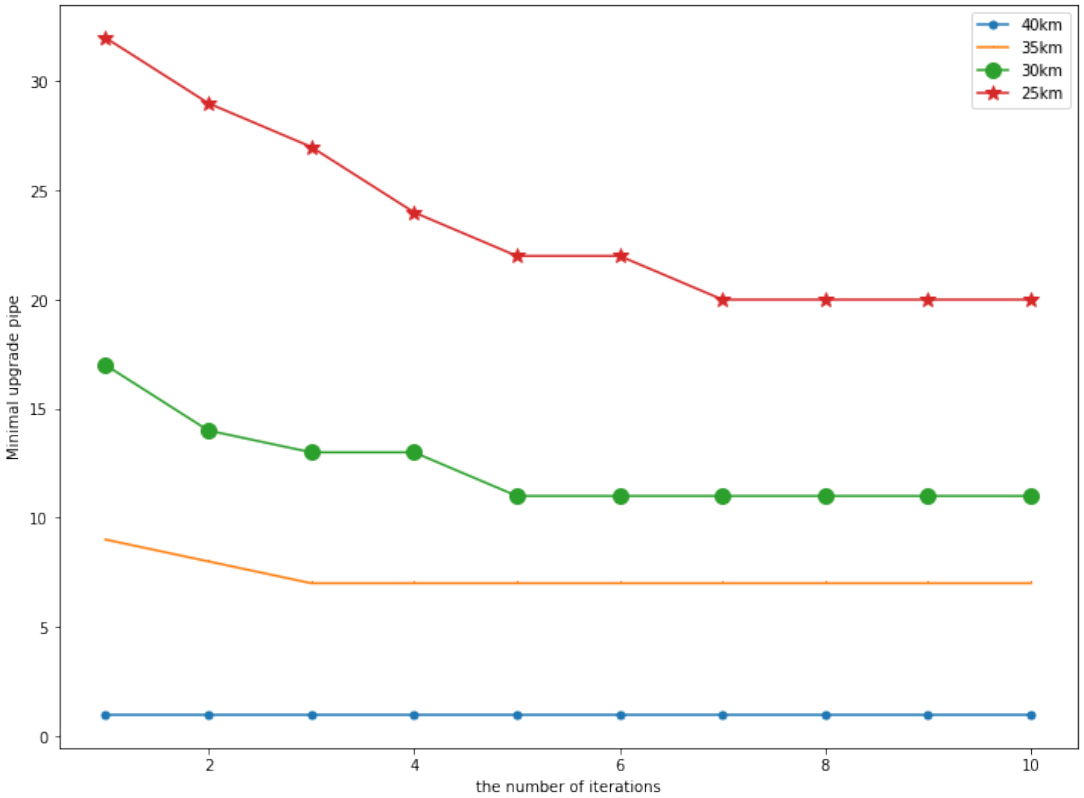


图 **12**　各个最大里程的免疫遗传算法收敛图

由算法收敛图可知算法收敛速度较快，当代价取较小时早熟问题解决的较好，全局搜素能力较强，算法复杂度远低于遍历求解。

8 模型的评价

8.1 模型的优点

(1) 问题一与问题二算法原理基于经典图论算法 Prim 算法，所得解可被证明为全局最优解。

(2) 基于免疫遗传算法进行优化计算，使得算法具有很强的鲁棒性，且兼顾了局部搜素能力和全局搜索能力。

8.2 模型的缺点

免疫遗传算法寻优时间较长，时间复杂度有待改善。

参考文献

[1] 卢开澄, 卢华明. 图论及其应用 [M]. 清华大学出版社有限公司, 2004.

[2] Jiang B, Zhang L. Research on minimum spanning tree based on prim algorithm[J]. Computer Engineering and Design, 2009, 13.

[3] Graham R L, Hell P. On the history of the minimum spanning tree problem[J]. Annals of the History of Computing, 1985, 7(1): 43-57.

[4] 丁建立, 陈增强, 袁著祉. 遗传算法与蚂蚁算法的融合 [J]. 计算机研究与发展, 2003, 40(9): 1351-1356.

[5] 贺毅朝, 刘坤起, 张翠军, 等. 求解背包问题的贪心遗传算法及其应用 [J]. 计算机工程与设计, 2007, 28(11): 2655-2657.

[6] 王磊, 潘进, 焦李成. 免疫算法 [D]. , 2000.

附录 A 问题一、二代码及其可视化

Graph 类实现最小生成树算法

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import copy
import networkx as nx
from tqdm.notebook import tqdm

class Graph(object):
def __init__(self, Matrix, add_edge=None, add_node=None):
self.Matrix = Matrix
self.nodenum = len(self.Matrix)
self.edgenum = self.get_edgenum()
self._weight_ = np.zeros((self.nodenum, self.nodenum))
self.add_edge = add_edge
self.add_node = add_node

def get_edgenum(self):
count = 0
for i in range(self.nodenum):
for j in range(i):
if self.Matrix[i][j] > 0 and self.Matrix[i][j] < 9999:
count += 1
return count

def plot_matrix(self, pos=None, figsize=(15,15), title="Pipeline ONE"):
plt.figure(figsize=(12,9))
self._get_edge()
G_nx = nx.Graph()
G_nx2 = nx.Graph()
if self.add_edge!=None:
for i in range(self.nodenum):
for j in range(self.nodenum):
if self._weight_[i, j]!=0 and i<13 and j<13:
G_nx.add_edge(i, j)
if self._weight_[i, j]!=0 and i>0 and j>0:
G_nx2.add_edge(i, j)
else:
for i in range(self.nodenum):
for j in range(self.nodenum):
if self._weight_[i, j]!= 0:
G_nx.add_edge(i, j)
```

```
        if self.add_edge!=None:
        nx.draw_networkx(G_nx, pos[:len(self.add_edge)+1], alpha=0.85)
        nx.draw_networkx(G_nx2,pos,alpha=0.6,with_labels=False,node_color='slateblue',
        node_shape=".", node_size=100, style='dashed')
        else:
        nx.draw_networkx(G_nx, pos, alpha=0.85)
        GG = nx.Graph()
        GG.add_node(0)
        nx.draw_networkx(GG, {0:pos[0]}, node_color='r',node_shape='*', node_size=1200)
        plt.title(title)
        plt.show() # display

        def _get_edge(self):
        edge = self.prim()
        if self.add_node!=None:
        for i in edge:
        for id_j, j in enumerate(self.add_node):
        if i[0]==j:
        i[0] = 13+id_j
        if i[1]==j:
        i[1] = 13+id_j
        for k in edge:
        self._weight_[k[0],k[1]] = self.Matrix[k[0],k[1]]
        if self.add_node!=None:
        for i in edge:
        for id_j, j in enumerate(self.add_node):
        if i[0]==13+id_j:
        i[0] = j
        if i[1]==13+id_j:
        i[1] = j
        return self._weight_

        def prim(self, first_node = 0):
        # 存储已选顶点，初始化时可随机选择一个起点
        select = [first_node]
        # 存储未选顶点
        candidate = list(range(0, self.nodenum))
        candidate.remove(first_node)
        if self.add_edge!=None:
        node = []
        select.remove(first_node)
        for i in self.add_edge:
        if i[0] not in node:
        node.append(i[0])
        if i[1] not in node:
        node.append(i[1])
        for i in node:
```

```
        select.append(i)
        if i in candidate:
        candidate.remove(i)
        # 存储每次搜索到的最小生成树的边
        edge = []+self.add_edge if self.add_edge!=None else []

        def min_edge(select, candidate, graph):
        min_weight = np.inf
        v, u = 0, 0
        for i in select:
        for j in candidate:
        if min_weight > graph[i][j]:
        min_weight = graph[i][j]
        v, u = i, j
        return v, u

        num = len(self.add_edge)+1 if self.add_edge!=None else 1
        for i in range(num, self.nodenum):
        v, u = min_edge(select, candidate, self.Matrix)
        edge.append([v, u])
        select.append(u)
        candidate.remove(u)
        if self.add_node!=None:
        for i in edge:
        for id_j, j in enumerate(self.add_node):
        if i[0]==13+id_j:
        i[0] = j
        if i[1]==13+id_j:
        i[1] = j
        return edge
```

问题一代码实现及可视化

```
        def distance(x1,y1,x2,y2):
        return np.sqrt((x1-x2)**2+(y1-y2)**2)

        def fix(x):
        if x.startswith('A'):
        return 0
        return 1 if x.startswith('V') else 2

        def get_xy(i,j=0, add_node = None):
        pos = [] # 元组中的两个数字是第i（从0开始计数）个点的坐标
        if add_node==None:
        for k in range(j, i):
```

```
        pos.append((data['X坐标'].loc[k], data['Y坐标'].loc[k]))
        else:
        for k in range(j, i):
        pos.append((data['X坐标'].loc[k], data['Y坐标'].loc[k]))
        for k in add_node:
        pos.append((data['X坐标'].loc[k], data['Y坐标'].loc[k]))
        return pos

        weight_array = np.zeros((181,181))
        data = pd.read_excel('/content/drive/My Drive/competitions/CMCM/demo1/data.xlsx')
        data['类型'] = data['类型'].apply(lambda x:fix(x))

        # 初始化权重矩阵
        for i in tqdm(range(181)):
        for j in range(181):
        point_i = data[data['序号']==i]
        point_j = data[data['序号']==j]
        weight_array[i][j] = distance(point_i['X坐标'].values,
        point_i['Y坐标'].values,
        point_j['X坐标'].values,
        point_j['Y坐标'].values)
        if (i==0 and j>12) or (j==0 and i>12):
        weight_array[i][j]=0
        weight_array[weight_array==0] = 10000

        weight_array_A = weight_array[:13,:13]
        G_A = Graph(weight_array_A)
        pos_A = get_xy(G_A.nodenum)
        edge_A = G_A.prim(first_node=0)
        G_A.plot_matrix(pos_A)

        G = Graph(weight_array, edge_A)
        print('节点数据为%d，边数为%d\n'%(G.nodenum, G.edgenum))
        pos = get_xy(G.nodenum)
        edge = G.prim()
        G.plot_matrix(pos, title="Pipeline TWO")

        sum = 0
        for p in edge:
        i,j=p[0],p[1]
        sum = sum+weight_array[i][j]
        sum
```

问题二代码实现及可视化

```
_max = (0,0,0)
_max2 = (0,0,0)
for ed in edge:
i,j = ed[0],ed[1]
if _max[0] < weight_array[i][j] and not (i<13 and j<13):
_max = (weight_array[i][j], i, j)
for ed in edge:
i,j = ed[0],ed[1]
if i!=126 and j!=125:
if _max2[0] < weight_array[i][j] and not (i<13 and j<13):
_max2 = (weight_array[i][j], i, j)

_max, _max2

_weight_ = G._get_edge()
plt.figure(figsize=(12,9))
G_nx = nx.Graph()
G_nx2 = nx.Graph()
G_nx3 = nx.Graph()
G_nx4 = nx.Graph()
GG = nx.Graph()
for i in range(G.nodenum):
for j in range(G.nodenum):
if _weight_[i, j]!=0 and i<13 and j<13:
G_nx.add_edge(i, j)
if _weight_[i, j]!=0 and i>0 and j>0:
G_nx2.add_edge(i, j)
# G_nx3.add_edge(126, 125)
# G_nx4.add_edge(88, 89)
G_nx3.add_node(125)
G_nx4.add_node(89)

nx.draw_networkx(G_nx3, {125:pos[125],},node_color='r',
node_size=300, node_shape='.',with_labels=False, style='dashed')
nx.draw_networkx(G_nx4, {89:pos[89],},node_color='r',
node_size=300, node_shape='.',with_labels=False, style='dashed')

nx.draw_networkx(G_nx, pos[:len(G.add_edge)+1], alpha=0.85)
nx.draw_networkx(G_nx2,pos,alpha=0.6,with_labels=False,node_color='slateblue',
node_shape=".", node_size=100, style='dashed')
GG = nx.Graph()
GG.add_node(0)
nx.draw_networkx(GG, {0:pos[0]}, node_color='r',node_shape='*', node_size=1200)

plt.title("Upgrade Two Secondary Pipes")
plt.show()
```

附录 B 问题三代码及其可视化

改进 prim 算法

```
def prim_pro(Matrix, add_edge, first_node=0, cost=40):
nodenum = len(Matrix)

select = [first_node]
candidate = list(range(0, nodenum))
candidate.remove(first_node)
node = []
for i in add_edge:
if i[0] not in node:
node.append(i[0])
if i[1] not in node:
node.append(i[1])
node_A = {}
for i in node:
node_A[i] = [0,[i]] # 存储形式是[dis, [v,u1,u2...]]
for i in node:
select.append(i)
if i in candidate:
candidate.remove(i)
# 存储每次搜索到的最小生成树的边
edge = []+add_edge
loss = []
def min_edge(select, candidate, graph):
min_weight = np.inf
v, u = 0, 0
for i in select:
for j in candidate:
if min_weight > graph[i][j] and (i,j) not in loss:
min_weight = graph[i][j]
v, u = i, j
for x in node:
if v in node_A[x][1]:
if node_A[x][0]+min_weight<=cost:
node_A[x][0] = node_A[x][0]+min_weight
node_A[x][1].append(u)
else:
loss.append((v,u))
return False, v, u
return True, v, u

num = len(add_edge)+1
test = 0
```

```
        while test<num*(nodenum-num):
        flag, v, u = min_edge(select, candidate, Matrix)
        if flag:
        edge.append([v, u])
        select.append(u)
        candidate.remove(u)
        else:
        test = test+1
        return edge, candidate, node_A
```

免疫遗传算法及其可视化

```
        COST = 40
        e, c, node_A = prim_pro(weight_array, edge_A, cost=COST)
        _size = len(c)
        _size#, c

        _weight_ = np.zeros((181,181))
        for k in e:
        _weight_[k[0],k[1]] = weight_array[k[0],k[1]]
        plt.figure(figsize=(12,9))
        G_nx = nx.Graph()
        G_nx2 = nx.Graph()
        G_nx3 = nx.Graph()

        for i in range(181):
        for j in range(181):
        if _weight_[i, j]!=0 and i<13 and j<13:
        G_nx.add_edge(i, j)
        if _weight_[i, j]!=0 and i>0 and j>0:
        G_nx2.add_edge(i, j)
        nx.draw_networkx(G_nx, pos[:len(edge_A)+1], alpha=0.85)
        nx.draw_networkx(G_nx2,pos,alpha=0.6,with_labels=False,node_color='slateblue',
        node_shape=".", node_size=100, style='dashed')
        GG = nx.Graph()
        GG.add_node(0)
        nx.draw_networkx(GG, {0:pos[0]}, node_color='r',node_shape='*', node_size=1200
        ,label='Central pipeline')
        p = {}
        for i in c:
        G_nx3.add_node(i)
        p[i] = pos[i]
        nx.draw_networkx(G_nx3, p, node_color='purple', node_size=120,with_labels=False,
        node_shape=",",label='Isolated point')
```

```
plt.show() # display

def fun(node_B, learning = 100):
node_B_index = []
for i in range(_size):
if node_B[i]:
node_B_index.append(i+13)
node_A_index = list(range(0, 13))+node_B_index
weight_array_A = np.zeros((len(node_A_index),len(node_A_index)))
for id_i,i in enumerate(node_A_index):
for id_j,j in enumerate(node_A_index):
weight_array_A[id_i][id_j] = weight_array[i][j]
G_A = Graph(weight_array_A)
edge_AA = G_A.prim(first_node=0)
e, c, node_A = prim_pro(weight_array, edge_AA, cost=COST)
return np.sum(node_B)+learning*len(c)

def mutate(node_B1, mu = 0.05):
l = len(node_B1)
for i in range(l):
p = np.random.random()
if p < mu:
node_B1[i] = 0 if node_B1[i]==1 else 0
return node_B1

def cross(node_B1,node_B2):
le = round(len(node_B1)/3)
temp = node_B1[:le].copy()
node_B1[:le] = node_B2[:le]
node_B2[:le] = temp
return node_B1,node_B2

def select(geti):
sss = []
for i in tqdm(geti):
sss.append(fun(i))
idx = [i for i,v in sorted(enumerate(sss), key=lambda x:x[1])]
geti2 = []
for i in idx:
geti2.append(geti[i])
le = round(len(geti)*3/4)
# le = len(geti)-1
g = []
for i in range(le):
g.append(geti2[i])
return g, sss, idx
```

```
add_node = [89]
node_A_index = list(range(0, 13))+add_node
weight_array_A = np.zeros((len(node_A_index),len(node_A_index)))
for id_i,i in enumerate(node_A_index):
for id_j,j in enumerate(node_A_index):
weight_array_A[id_i][id_j] = weight_array[i][j]
G_A = Graph(weight_array_A,add_node=add_node)
edge_AA = G_A.prim(first_node=0)

pos_A = get_xy(13,add_node=add_node)
G_A.plot_matrix(pos_A)
e, c, node_A = prim_pro(weight_array, edge_AA, cost=COST)

_weight_ = np.zeros((181,181))
for k in e:
_weight_[k[0],k[1]] = weight_array[k[0],k[1]]
plt.figure(figsize=(12,9))
G_nx = nx.Graph()
G_nx2 = nx.Graph()
G_nx3 = nx.Graph()
pos_A = get_xy(13,add_node=add_node)
for i in range(181):
for j in range(181):
if _weight_[i, j]!=0 and i<13 and j<13:
G_nx.add_edge(i, j)
if _weight_[i, j]!=0 and i>0 and j>0:
G_nx2.add_edge(i, j)
nx.draw_networkx(G_nx, pos_A, alpha=0.85)
nx.draw_networkx(G_nx2,pos,alpha=0.6,with_labels=False,node_color='slateblue',
node_shape=".", node_size=100, style='dashed')
GG = nx.Graph()
GG.add_node(0)
nx.draw_networkx(GG, {0:pos[0]}, node_color='r',node_shape='*', node_size=1200
,label='Central pipeline')
p = {}
for i in add_node:
G_nx3.add_node(i)
p[i] = pos[i]
nx.draw_networkx(G_nx3, p, node_color='gold', node_size=150,with_labels=False,
node_shape=",",label='Isolated point')

plt.show() # display
```