复杂通信网络的修复策略与鲁棒性研究

摘要

随着科学技术的迅速发展，通信技术已成为国家竞争和社会进步的关键环节。日常生活中通信网络的可靠性保证对于信息的收集获取、消息的传递和日常生活的有序进行起着至关重要的作用。故本文对复杂通信网络的修复策略与鲁棒性进行研究，给出节点严重毁坏时备选节点的确定与连接方式，以保证在最短路径下恢复网络连通，同时给出高连通性网络设计方案，本文工作如下：

问题一要求给出网络的最短路径连接方案。对此本文首先基于Great-circle公式计算各城市节点间的球面距离，然后将问题转化为最小生成树问题，利用Prim算法求解。同时使用 Python 绘制网络的连接示意图，进行可视化展示。其中最短路径的计算结果为25343.943km。

问题二要求给出指定节点故障后，备份节点的位置、数目及连接方式使网络恢复连通。对此本文首先分析故障节点的边数，并将其分为边数为1，边数为2，边数大于2，三类情况进行讨论。当边数大于2时，该问题本质上为Steiner tree的构建问题，本文以最短路径为目标，讨论设置不同备选节点数量时的路径长度，利用实码加速遗传算法结合“先粗后精”搜索策略进行求解。最终计算结果显示上海、武汉、北京的备选节点数目均为1，其连接方式与问题一保持一致，且位置坐标分别为(121.46,31.24)，(114.77,30.76)，(115.79,39.21)。

问题三要求给出9个城市节点被同时损坏时，备份节点的数目、位置与连接方式，同时给出衡量网络连通性的指标。对此本文以问题二为基础，分析故障节点的位置及连接节点信息，发现该故障节点均集中于东经111.29~117.23，北纬28.23~32.99这一区域内，且相互间存在较强关联。对此本文仍以启发式搜索算法求解备份节点数目为1～15时网络的最短路径长度。结果表明，当节点数目设置为7时网络路径最短，为2055.71km。针对网络的连通性评价，本文借鉴复杂网络的自然连通度指标进行描述。该指标具有区别网络结构分辨率高、可解释性好、严格单调等优点。

问题四要求在问题一构建网络的基础上设计“高可靠、短路径”的通信网，并模拟测试网络 10%节点随机故障时其连通性变化。对此本文利用禁忌搜索与遗传算法相结合的策略寻求网络的最小支配集，然后以最小支配集中的顶点为关键节点，在其间增加连边，以提高网络的连通性。同时在不同关键节点数目下，探究增加的连边数目与自然连通度间的关系，并模拟仿真 10%的节点遭遇故障时网络的连通度变化。实验表明当关键节点选择数目为25，边的添加数目为60时，网络的鲁棒性高且路径相对较短。

关键词：复杂通信网络，鲁棒性，最短路径修复，自然连通度

1 问题重述

互联网、物联网、社交网、公路网、物流网、电力网、水利网、通信网……我们生活这一个被众多网络包围的世界中，大型复杂的网络研究有着实际的价值和意义[1-4]。网络的可靠性保证在日常生活种有着至关重要的作用，因此如何保证网络的高可靠同时较好控制网络的路径长度，以及如何对故障网络进行快速修复是本文的研究重点。

基于以上问题，本文回答下列问题：

（1）给定数个城市地理位置坐标，寻求最短路径网络，给出网络的构建方案；

（2）若部分节点故障，要求给出备份节点的数目及位置坐标，同时给出节点连接方案，以使得网络连通。

（3）若多个密集网络节点故障，如何设置备份节点数目和位置使网络连通，同时给出节点连接方案，并提出衡量网络的连通性指标。

（4）网络的连通性与最短路径间存在天然的互斥矛盾，如何在问题（1）的基础上构建网络，在保证网络连通性较高的前提下，使得路径尽可能的短。同时模拟仿真测试网络 10%的随机故障后，网络的连通性。

2 模型假设

1. 假设地球是一个标准的球体，其半径为 6371.393km，在计算各城市间的最短球面距离时忽略海拔等地势信息的影响。
2. 附件所给城市个节点位置信息准确无误。
3. 地理坐标与投影坐标间的相互转化参考国际公式，本论文无其它特殊标准要求。
4. 启发式搜索策略，每次求解结果相差不大，忽略其细微差别。

3 符号及变量说明

符号	含义
θ	经纬度
δ	弧度
L	两点间的球面距离
V	城市节点
E	节点连边
$\left\langle V_i, V_j \right\rangle$	城市 i, j 间的连边
d_{ij}	城市 i, j 间的距离
f	网络路径最短目标函数
b_i	故障节点 i

l_{ij}	与故障节点 i 相连的节点 j
S	网络整体连通度
$\overline{\lambda}$	网络自然连通度

4 问题分析

4.1 问题一分析

问题一要求给出 139 个城市节点的最短连通方案，此问题本质上为图论中的最小生成树问题。对此应首先确定各个节点间的球面距离，即最短劣弧长度，对词可以利用球面距离公式：Great-circle 求解。然后利用各节点间的距离矩阵，调用 Prim 算法构建最小生成树，给出网络的连接方案。同时利用 python 实现网络结构的可视化。

4.2 问题二分析

问题一要求给出指定节点故障后，网络的修复方案，其中备份节点的数目和位置可以任意指定。该问题与一般的最小生成树问题有较大的区别，即其可以引入新的节点降低网络的连通路径，而此类问题即对应 Steiner 树的构建模型。对此本文以最短路径为目标函数，首先分析故障节点的地理位置信息与其连通节点数目等情况，通过指定不同的备份节点数目求解网络最短路径长度，观察其变化趋势，同时确定最优连接方案。此外 Steiner 树的构建为 NP 难问题，对此我们可以考虑采用启发式搜索，如 GA 算法，同时结合“先粗后细”的搜索策略进行求解。

4.3 问题三分析

问题三的前半小问同问题二类似，只不过在问题二的基础上增加了更多的故障节点数目，且在地理位置上更加集中，对此本文仍以最短路径为目标函数，利用启发式算法求解，不过却别第二问的是在备选节点数目的选择上将尝试更多节点数目。而对于网络连通性指标的确定，可以参考图论中的点连通度、边连通度的概念，对于复杂网络的连通性即鲁棒性的衡量，这里选择自然连通度进行刻画。

4.4 问题四分析

问题四指出网络的连通性与网络的长度两者不可兼得，其间存在“跷跷板”现象，因此如何在增加网络连通性的同时尽可能的减小网络的路径长度，这是研究的重点。对此本文以问题一的网络结构为基础，参考图论中的最小支配集的概念，通过寻找最小支配集而确定关键节点，同时在关键节点间添加连边，以增加网络的冗余度，提高网络的鲁棒性和连通性。由于关键节点数目较多且最小支配集的求解为 NP 难问题，对此本文可以考虑采用禁忌搜索与遗传相结合的策略求解。在确定关键节点后，我们在其间添加任意指定数目的连边以提高网络的连通性。最后给出设计网络在 10%节点故障失效时，不同关键节点下网络的自然连通度随连边数目的变化曲线和仿真结果，以确定最优的关键节点数目和连边数目。

5 模型的建立与求解

通讯网络的可靠性设计与可靠性研究，以及复杂网络的修复策略和鲁棒性，在实际的日常生活以及战争中有着广泛的研究价值和重要的现实意义，其对于信息的传输、数据的交换、通信，以及社会的发展至关重要的作用。其中网络的鲁棒性与网络的总长度这两个指标间存在一定的互斥关系，而如何在保证通信网络线路较短的情况下，构建高可靠、鲁棒性强的网络则是本文研究的重点，其本质上是在保证网络具有较高连通度的情况下，寻找最短路径网络的构建方法。该问题与图论中的最小生成树，连通度，最小支配集以及运筹学等知识密切相关。而问题的求解又涉及各种算法如 Prim 算法，启发式算法的应用，因此具有较高的综合性。

5.1 最短路径通信网的构建

问题一要求寻找路径最短的 139 个节点的连通图，对此本论文首先根据附件一中所提供的各城市地理位置信息即经纬度，将其转化为弧度，然后利用 Great-circle 公式计算各个城市节点间的最短球面距离，并利用 python 绘制空间地理位置信息地图，以实现城市间空间位置的可视化。

5.1.1 各城市间最短球面距离的计算

附件一所提供的各个城市地理位置信息为经纬度，对此我们首先需要将角度转化为弧度，然后基于 Great-circle 公式[5]计算最短球面距离。

Step1. 经纬度转弧度

附件 1 所给经纬度信息其实际上是空间球面坐标，而在计算球面距离时首先我们需要完成角度转弧度的操作：

$$\delta=\theta\bullet\pi/180 \tag{1}$$

上式中，θ 为经纬度；

δ 为弧度。

表 5.1　经纬度转弧度

city	湛江	北海	凭祥	澳门	香港	深圳	南宁	肇庆	广州	汕头	个旧	畹町
longtitude_radian	1.9261	1.9045	1.8635	1.9816	1.9926	1.9907	1.8914	1.9628	1.9768	2.0365	1.8005	1.7115
latitude_radian	0.3712	0.3749	0.3855	0.3875	0.3896	0.3934	0.3983	0.4023	0.4037	0.4075	0.4077	0.4203
city	柳州	厦门	韶关	昆明	龙岩	桂林	大理	郴州	赣州	福州	都匀	三明
longtitude_radian	1.9099	2.0611	1.9827	1.7951	2.0424	1.9234	1.7493	1.9724	2.0059	2.0822	1.8766	2.0532
latitude_radian	0.4246	0.4273	0.433	0.4341	0.4376	0.4405	0.4466	0.4498	0.4508	0.455	0.4583	0.4583
city	攀枝花	六盘水	贵阳	衡阳	怀化	萍乡	遵义	西昌	温州	长沙	鹰潭	椒江
longtitude_radian	1.7753	0.4641	1.861	1.9647	0.4812	1.9872	1.8663	1.7848	2.1066	1.9712	2.0433	2.1195
latitude_radian	0.4639	0.4641	0.4651	0.4693	0.4812	0.4821	0.484	0.4868	0.4885	0.4927	0.4932	0.5004
city	南昌	宜宾	常德	金华	日喀则	景德镇	岳阳	重庆	拉萨	九江	黄山	宁波
longtitude_radian	2.0221	1.8263	1.9495	2.0883	0.5109	0.5109	1.9745	1.8596	1.5902	2.0246	2.0654	2.1227
latitude_radian	0.5006	0.5018	0.5067	0.5075	0.1509	0.1509	0.5124	0.5159	0.5173	0.5185	0.5187	0.5212
city	黄石	杭州	沙市	安庆	武汉	成都	宜昌	上海	芜湖	绵阳	无锡	合肥
longtitude_radian	2.0078	2.0981	1.9593	2.0431	1.9949	1.8164	1.9424	2.1201	2.067	1.827	2.0998	2.046
latitude_radian	0.5271	0.528	0.529	0.5328	0.5339	0.5349	0.5356	0.5451	0.5472	0.5493	0.5496	0.5554
city	襄樊	南京	信阳	扬州	十堰	蚌埠	南阳	汉中	徐州	西安	宝鸡	天水
longtitude_radian	1.9569	2.0735	1.9912	2.0841	1.9338	2.0488	1.964	1.8679	2.0469	1.9014	1.8717	1.8453
latitude_radian	0.5587	0.5596	0.5611	0.5653	0.5695	0.5746	0.5758	0.5772	0.5971	0.5993	0.5997	0.6035
city	连云港	洛阳	郑州	开封	运城	济宁	兰州	青岛	临汾	安阳	长治	格尔木
longtitude_radian	0.6039	1.9626	1.983	0.6074	1.9375	2.0349	1.8122	2.101	1.9464	1.9965	1.9743	0.6355
latitude_radian	0.6039	0.6042	0.6065	0.6074	0.6114	0.618	0.6294	0.6295	0.6299	0.6301	0.6318	0.6355
city	延安	西宁	邯郸	济南	潍坊	和田	荣成	德令哈	烟台	太原	青铜峡	石家庄
longtitude_radian	1.911	1.7764	1.9991	2.0441	2.0797	1.3949	2.1379	0.6522	2.1197	1.9644	0.6636	1.9986
latitude_radian	0.6386	0.6391	0.6393	0.6397	0.6407	0.6477	0.6487	0.6522	0.6538	0.661	0.6636	0.6639
city	榆林	银川	保定	大连	张掖	天津	喀什	唐山	北京	秦皇岛	丹东	大同
longtitude_radian	1.9153	1.8541	2.0152	2.1225	1.7532	2.0455	1.3263	2.0626	2.0317	0.6971	2.1705	1.9775
latitude_radian	0.6683	0.6718	0.6784	0.6791	0.6795	0.6822	0.6889	0.6917	0.6964	0.6971	0.6981	0.6995
city	敦煌	玉门	包头	营口	张家口	呼和浩特	承德	锦州	鞍山	阿克苏	沈阳	通化
longtitude_radian	1.6521	1.6938	1.9171	2.1335	0.7116	1.9504	2.0588	2.1141	2.1466	0.7186	2.1546	2.1981
latitude_radian	0.7006	0.7032	0.7097	0.7098	0.7116	0.7128	0.7147	0.7173	0.7175	0.7775	0.728	0.7283
city	哈密	图们	四平	二连浩特	乌鲁木齐	长春	锡林浩特	牡丹江	鸡西	白城	哈尔滨	塔城
longtitude_radian	1.6321	2.2661	2.1703	1.9539	1.5293	2.1872	2.0262	0.7775	2.2859	2.144	0.7994	1.4483
latitude_radian	0.7472	0.75	0.7535	0.7617	0.7648	0.7648	0.7667	0.7775	0.7906	0.7962	0.7944	0.8159
city	佳木斯	齐齐哈尔	同江	阿勒泰	海拉尔	满洲里	黑河					
longtitude_radian	0.8168	2.1628	2.3127	0.8351	0.8589	0.8657	2.2258					
latitude_radian	0.8168	0.8264	0.8315	0.8351	0.8589	0.8657	0.877					

Step2. Great-circle 公式计算最短球面距离

这里假设地球为一个标准的球体，设球心为 O, 地球半径为 r，如图 5.1 所示：

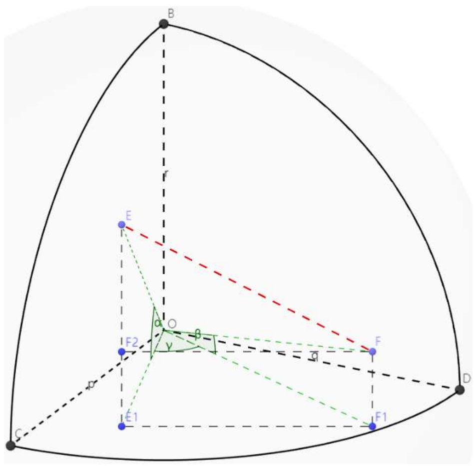


图 5.1　球面两点距离计算示意图

图 5.1 中线段 OE, OF, EF 构成一个等腰三角形，其中 OE=OF=r，这里定义 $\angle EOF$ 弧度为 δ ，且定义 E,F 两点的最短球面距离为 L，则有：

$$L = r \bullet \delta \tag{2}$$

根据余弦定理可得：

$$\cos\delta = \frac{OE^2 + OF^2 - EF^2}{2 \bullet OE \bullet OF} \tag{3}$$

同时，将 OE=OF=r 代入式（2）有：

$$\cos\delta = 1 - \frac{EF^2}{2 \bullet r^2} \tag{4}$$

$$\delta = \arccos\left(1 - \frac{EF^2}{2 \bullet r^2}\right) \tag{5}$$

最后即有：

$$L = r \bullet \arccos\left(1 - \frac{EF^2}{2 \bullet r^2}\right) \tag{6}$$

此外，图 1 中 B 点为北极点，C，D 两点均位于赤道上，且 OB，OC，OD 构造空间直角坐标系，OCD 即赤道面。这里，过 E 点做垂线垂直于面 OCD 交于 E1 点，过 F 点做垂线垂直于面 OCD 交于 F1 点，过 F 点做垂线垂直于线 EE1 于 F2 点。并定义 $\angle EOE_1$ 弧度为 α，$\angle FOF_1$ 弧度为 β，$\angle E_1OF_1$ 弧度为 γ。故 α 即为 E 点的纬度弧度，β 即为 F 点的纬度弧度，γ 即为 F 点和 E 点的经度之差所对应的弧度。同时根据勾股定理有：

$$EF^2 = EF_2^{\ 2} + FF_2^{\ 2} \tag{7}$$

且：

$$EF_2 = EE_1 - FF_1 \tag{8}$$

$$EF_2 = EE_1 - FF_1 \tag{9}$$

$$FF_2 = E_1F_1 \tag{10}$$

故：

$$EF^2 = \left(EE_1 - FF_1\right)^2 + E_1F_1^2 \tag{11}$$

又 $\angle OE_1E, \angle OF_1F$ 为直角，则：

$$\begin{aligned} EE_1 &= OE \bullet \sin\alpha = r \bullet \sin\alpha \\ FF_1 &= OF \bullet \sin\beta = r \bullet \sin\beta \end{aligned} \tag{12}$$

由余项定理有：

$$E_1F_1^{\ 2} = OE_1^{\ 2} + OF_1^{\ 2} - 2OE_1 \bullet OF_1 \bullet \cos\gamma \tag{13}$$

且：

$$
\begin{aligned}
OE_1 &= OE \bullet \cos\alpha = r \bullet \cos\alpha \\
OF_1 &= OF \bullet \cos\beta = r \bullet \cos\beta
\end{aligned}
\tag{14}
$$

因此有：

$$
\begin{aligned}
E_1F_1^2 &= r^2 \bullet \cos^2\alpha + r^2 \bullet \cos^2\beta - 2r^2 \bullet \cos\alpha \bullet \cos\beta \bullet \cos\gamma \\
&= r^2\left(\cos^2\alpha + \cos^2\beta - 2\cos\alpha \bullet \cos\beta \bullet \cos\gamma\right)
\end{aligned}
\tag{15}
$$

将公式（12）、（15）代入（11）整理得：

$$
\begin{aligned}
EF^2 &= r^2\left(\sin^2\alpha + \cos^2\alpha\right) + r^2\left(\sin\beta^2 + \cos^2\beta\right) - 2r^2 \bullet \sin\alpha \bullet \sin\beta - 2r^2\cos\alpha\cos\beta\cos\gamma \\
&= 2r^2\left(1 - \sin\alpha \bullet \sin\beta - \cos\alpha \bullet \cos\beta \bullet \cos\gamma\right)
\end{aligned}
\tag{16}
$$

最后将式（12）代入式（6）即得到计算球面间两点距离 Great-circle 公式：

$$
L = r \bullet \arccos\left(\sin\alpha\sin\beta + \cos\alpha\cos\beta\cos\gamma\right) \tag{17}
$$

故各城市间的球面距离如图 5.2 所示（完整结果见附件）：

	湛江	北海	凭祥	澳门	香港	深圳	南宁	肇庆	广州	汕头	个旧	畹町	柳州	厦门	韶关	昆明	龙岩	桂林	大理	郴州	赣州
湛江	0.0	130.5112	382.0044	344.3661	410.3383	407.0082	267.91	293.1691	363.2026	690.0074	776.1729	1299.631	3								
北海	130.5112	0.0	251.96	463.1866	529.3209	522.7308	167.8412	385.4988	463.7358	804.3783	647.2756	1169.8359									
凭祥	382.0044	251.96	0.0	697.3509	762.2182	751.5428	183.3768	593.905	676.1296	1025.89	396.2737	917.8821	368								
澳门	344.3661	463.1866	697.3509	0.0	66.195	65.4895	535.5383	145.6807	107.3346	346.4043	1071.7835	1595.8073									
香港	410.3383	529.3209	762.2182	66.195	0.0	26.9509	598.1225	193.312	129.711	281.5805	1133.9615	1657.2442	53								
深圳	407.0082	522.7308	751.5428	65.4895	26.9509	0.0	584.6034	173.5527	105.0079	283.0067	1119.6044	1642.0441									
南宁	267.91	167.8412	183.3768	535.5383	598.1225	584.6034	0.0	419.6273	501.8032	852.0113	536.2898	1060.8344									
肇庆	293.1691	385.4988	593.905	145.6807	193.312	173.5527	419.6273	0.0	82.3167	432.5964	950.9736	1471.5807	3								
广州	363.2026	463.7358	676.1296	107.3346	129.711	105.0079	501.8032	82.3167	0.0	350.2987	1032.0753	1551.6786									
汕头	690.0074	804.3783	1025.89	346.4043	281.5805	283.0067	852.0113	432.5964	350.2987	0.0	1379.7599	1896.089									
个旧	776.1729	647.2756	396.2737	1071.7835	1133.9615	1119.6044	536.2898	950.9736	1032.0753	1379.7599	0.0	525									
畹町	1299.631	1169.8359	917.8821	1595.8073	1657.2442	1642.0441	1060.8344	1471.5807	1551.6786	1896.0893	525.										
柳州	353.3759	318.5114	368.6954	482.0326	533.0807	512.5719	199.6646	339.7829	412.0751	745.3403	646.7157	115									
厦门	868.4856	976.753	1186.011	529.1888	466.5855	464.0882	1006.9204	594.5775	513.7895	190.6126	1522.0046	20									
韶关	514.6195	588.9473	759.4433	290.3011	282.9207	256.7376	576.1828	227.4406	189.9862	352.3143	1071.7841	15									
昆明	866.1259	743.4425	505.3345	1129.1887	1187.468	1170.0255	605.8774	997.0648	1074.7841	1413.2638	170.										

图 5.2 各城市间的距离（截取部分）

5.1.2 各城市间最小连通图的确定

问题一要求我们给出 139 个城市间的通信线路连接方案使其总长度最短。该问题本质上即为图论中的最短路径问题也即最小生成树（minimum spanning tree，MST）[6]问题。而稠密图的最小生成树的构建最早是由罗伯特•普里姆于 1957 年发明，即 Prim 算法[7]。20 世纪 70 年代，优先队列提出后其很快被用于寻找稀疏图中的最小生成树问题中。1984 年，迈克尔•弗里德曼和罗伯特•塔扬发明了斐波那契堆，Prim 算法所需要的运行时间在理论上由 $E\log E$ 提升到至 $E+V\log V$。此后又陆续有学者对其进行改进，如 Kruskal 算法[8][9]。

本问题中可以将各个城市视为图中的节点 V，此时我们需遍历所有节点同时使得路径最短，其中各个节点间边的权值即为 5.1.1 所求各个节点间的距离。这里记城市 i 与城市 j 间的距离为 d_{ij}，城市 i 与城市 j 间的边为 $\left\langle V_i, V_j \right\rangle$，最终生成的无向图记为 G。

（1）路径最短问题即最小生成树问题的定义：

在一给定的无向图 $G=(V,E)$ 中，$\left\langle V_i, V_j \right\rangle$ 代表连接顶点 V_i 与顶点 V_j 的边（即 $\left\langle V_i, V_j \right\rangle \in E$，而 $w(V_i, V_j)$ 代表此边的权重，若存在 T 为 E 的子集（即 $T \subseteq E$）且 (V,T) 为树，使得：

$$\min w(T)=\sum_{(V_i,V_j)} w(V_i,V_j)$$

则此T即为G的最小生成树。从上述定义可以看出最小生成树是指各边权值之和最小。求解 MST 问题的一般算法可以简单描述为：针对图G，从空树T开始向集合 T 中逐条选择并加入 n-1 条安全边$\left\langle V_i,V_j\right\rangle$（当边$\left\langle V_i,V_j\right\rangle$加入$T$时必须保证$T\cup\left\{\left\langle V_i,V_j\right\rangle\right\}$仍是 MST 的子集，故称这样的边为安全边），其对应的算法也主要包括 Prim 算法和 Kruskal 算法。

（2）Prim 算法介绍：

Step1. 以某一个点开始，寻找当前该点可以访问的所有边；

Step2. 在已经寻找的边中挑选最小边，其这条边必须至少有一个点还未被访问，将还没有访问的点加入至集合中，并记录添加的边；

Step3. 寻找当前集合可以访问的所有边，重复 Step2 的过程，直到没有新的点加入；

Step4. 此时由所有边构成的树即为最小生成树。

（3）Kruskal 算法介绍：

Step1. 假设一个图有m个节点，n条边。首先，把m个节点看成m个独立的生成树，并且把n条边按照从小到大的数据进行排列。

Step2. 在n条边中，依次取出其中的每一条边，如果发现边的两个节点分别位于两棵树上，则把两棵树合并成为一颗树；如果树的两个节点位于同一棵树上，那么忽略这条边，继续运行。

Step3. 直到所有的边都遍历结束，如果所有的生成树可以合并成一棵生成树，那么其即为最小生成树，反之则没有最小生成树。

对比上述两种算法可以看出，Prim 算法是以点为对象，挑选与点相连的最短边来构成最小生成树。而 Kruskal 算法是则以边为对象，不断地加入新的不构成环路的最短边来构成最小生成树。

（4）最小生成树的构建与最短连通路径求解

本文采用 Prim 算法构建最小生成树，其网络连接方案如表 5.2 所示：

表 5.2　最短路径连接方案

vertex	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
vertex_edge	0	1	6	2	12	17	28	38	42	33	29	50	54	27	19	14
edge_length	0	130.5112	167.8412	183.3768	199.6646	127.6055	259.8646	232.5096	143.5849	127.0148	113.0137	134.9181	102.0951	151.1674	132.0403	122.1277
vertex	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
vertex_edge	60	66	64	62	52	48	45	36	41	34	46	51	59	65	56	61
edge_length	166.6092	115.5624	141.9161	173.4836	174.6167	83.1796	107.3494	115.3413	124.2557	112.8245	122.922	136.7838	144.3592	123.2415	125.1129	86.3649
vertex	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
vertex_edge	63	58	55	49	39	47	35	32	68	77	87	20	73	74	75	76
edge_length	68.1142	131.2592	113.8805	138.2201	140.9092	142.4725	133.4771	104.7112	143.8119	147.5644	145.8946	175.3213	181.4096	107.957	63.2699	139.1317
vertex	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
vertex_edge	80	82	81	86	95	98	104	101	103	105	114	112	107	113	110	93
edge_length	126.5806	144.1956	114.5768	60.4481	156.8169	123.9447	140.6595	112.7327	103.4661	126.1434	147.9676	161.1186	154.9286	156.0127	162.1402	172.8899
vertex	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
vertex_edge	88	79	92	90	99	115	111	116	118	106	122	125	72	8	7	5
edge_length	182.0503	130.3518	181.5944	97.4669	161.8478	182.6609	104.8554	79.8065	76.9641	169.2112	178.3621	106.5234	183.1962	189.9862	82.3167	105.0079
vertex	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
vertex_edge	4	3	84	96	69	70	71	67	119	16	13	23	21	9	78	85
edge_length	26.9509	65.4895	190.1427	190.3271	204.2367	156.0891	140.5763	144.8864	206.5398	226.2326	126.3953	146.1943	167.0197	190.6126	238.3289	193.8949
vertex	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
vertex_edge	130	133	129	127	128	132	121	134	100	57	53	37	43	30	26	22
edge_length	239.9744	263.6266	209.3931	279.9282	134.4842	174.1811	176.5063	189.9462	282.2163	282.9428	108.1257	218.3381	206.1951	206.8502	123.7094	98.6497
vertex	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
vertex_edge	25	31	24	18	15	10	11	94	97	123	91	83	109	108	120	126
edge_length	179.0683	251.8401	155.1498	185.1057	221.297	170.8369	275.9479	295.5993	53.8812	311.7997	321.0775	241.0363	325.8171	203.6289	311.977	333.8891
vertex	128	129	130	131	132	133	134	135	136	137	138					
vertex_edge	136	137	138	124	135	131	117	102	89	44	40					
edge_length	372.0292	176.1819	416.9291	489.4958	449.9474	407.8083	657.484	408.3485	431.8159	832.6237	218.8567					

网络的最短连通路径为 25343.943km。

5.1.3 最小连通图的可视化展示

这里根据上述 5.1.1 及 5.1.2 所得结果对最小连通图进行可视化展示，如下图所示：

图 5.3　最短路径通信网络图

从图 5.3 可以明显看出该网络的每一个节点均为割点，网络的鲁棒性很差。此外，如南昌市、北京市、沈阳市等城市其节点的度很大，即其与多个城市直接相连，且上述城市均为交通枢纽。这也侧面反映了该城市地理位置的重要性，在实际的生活中，该节点有着十分重要的战略意义和现实价值，应该重点防范和保护。

5.2 备份节点的确定与网络的修复

问题二要求我们确定最优的备份节点数目与位置坐标，以期在某些节点严重损毁时以最快恢复通信网。这里本文仍以网络间路径最短为目标函数 f，寻求最优候选节点的数目与位置，使得网络恢复连通所需路径最小。因此，首先确定与故障节点 b_i 直接相连的城市节点 l_j，同时根据 l_j 所构成的集合 L 确定最小覆盖圆 S_i。然后利用遗传算法在最小覆盖圆的区域内搜索最优节点 a_{ik} 的数目与位置坐标使得目标函数 f 取得极小值，即恢复网络连通所需的路径最短。该问题本质上是通讯网络中的 Steiner 树[10][11]的构建问题，为避免暴力穷举的计算量，这里我们采用启发式搜索策略，即遗传算法（GA）[12]。同时为进一步减少计算量，以有效缩短计算时间，在使用遗传算法搜索最优解时，我们采用“先粗后精”的策略，即先粗分网格确定备份节点的数量和大致区域，然后在此基础上进行精确收缩确定最终的位置坐标。

5.2.1 最短路径各种情况的分类讨论

根据题目已知北京、武汉、上海这三座城市节点严重故障，故我们以问题一中最小连通网为基础确定此三座城市的连接节点，如下：

表 5.3　故障节点其连接节点

北京	保定、天津、张家口
武汉	信阳、黄石
上海	无锡

从表 5.3 可以看到，北京、武汉、上海这三座城市节点间并未存在连通边，故这三座城市的备份节点间也将不必存在连通边。因此要使得恢复网络的总路径最短则只需要使其备份节点距离各自的连接节点路径最短。这里我们对上述三种情况进行讨论：

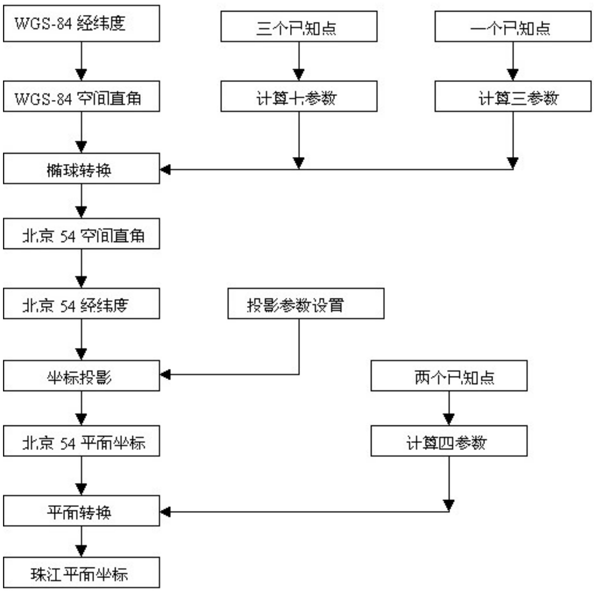
情况一：与故障节点直接相连的节点数目仅为 1，即该故障节点为顶点。此时备份节点可以直接选择故障节点的复制点，即备份节点与故障节点地理位置保持相同或进行较小的偏移。

情况二：与故障节点直接相连的节点数目为 2。此时若要满足路径最短原则，则根据两点间直线最短定理，我们可在相连节点的弧线上任意选择一点作为备份节点。

请况三：与故障节点直接相连的节点数目为 3，对此首先求解最小覆盖圆，然后在此圆中确定备份节点数目与位置使其路径最短。该问题不同于一般的最小生成树问题，而是可以额外引入新的节点即“虚设”节点使其距离进一步降低，该问题即为 Steiner tree 的构建问题，对此采用遗传算法求解。

5.2.2 地理坐标系统与投影坐标系统的相互转化

题目所给的各个城市节点的位置信息为经纬度即地理坐标或称 WGS84 坐标，而要确定两点间的连线以及 Steiner 树的构建均需要平面投影坐标即地方坐标。因此我们需要对这两者进行简单的转化[13]，其转化流程如下图所示：



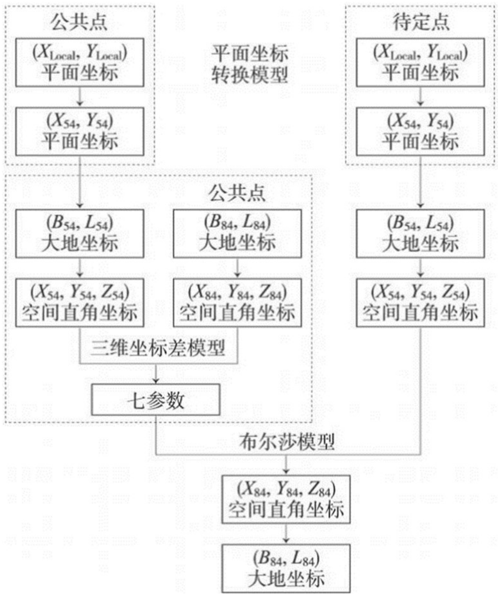


图 5.4 地理坐标转投影坐标流程图　　图 5.5 投影坐标转地理坐标流程图

根据图 5.4 可得各城市的投影坐标如下表所示（截取部分，完整文件件附件）：

表 5.4　各城市地理坐标与投影坐标对应表（截取部分）

city	longtitude	latitude	x	y	city	longtitude	latitude	x	y
湛江	110.36	21.27	12285219	2424102.476	萍乡	113.86	27.62	12674837.22	3201148.41
北海	109.12	21.48	12147182.84	2449206.359	遵义	106.93	27.73	11903393.15	3214975.417
凭祥	106.77	22.09	11885582.03	2522334.244	西昌	102.26	27.89	11383531.13	3235112.372
澳门	113.54	22.2	12639214.98	2535554.62	温州	120.7	27.99	13436262.54	3247713.077
香港	114.17	22.32	12709346.26	2549988.668	长沙	112.94	28.23	12572423.29	3278002.596
深圳	114.06	22.54	12697101.12	2576483.399	鹰潭	117.07	28.26	13032172.79	3281793.564
南宁	108.37	22.82	12063693.22	2610265.148	椒江	121.44	28.67	13518638.96	3333711.103
肇庆	112.46	23.05	12518989.93	2638066.413	南昌	115.86	28.68	12897476.2	3334979.91
广州	113.26	23.13	12608045.53	2647747.527	宜宾	104.64	28.75	11648471.52	3343864.954
汕头	116.68	23.35	12988758.19	2674400.452	常德	111.7	29.03	12434387.12	3379464.917
个旧	103.16	23.36	11483718.67	2675612.995	金华	119.65	29.08	13319377.07	3385832.188
畹町	98.06	24.08	10915989.27	2763159.492	日喀则	88.89	29.27	9895189.537	3410056.074
柳州	109.43	24.33	12181691.88	2793671.916	景德镇	117.18	29.27	13044417.93	3410056.074
厦门	118.09	24.48	13145718.67	2812008.241	岳阳	113.13	29.36	12593573.99	3421546.248
韶关	113.6	24.81	12645894.15	2852425.388	重庆	106.55	29.56	11861091.74	3447116.405
昆明	102.85	24.87	11449209.63	2859785.481	拉萨	91.11	29.64	10142318.81	3457358.628
龙岩	117.02	25.07	13026606.81	2884344.999	九江	116	29.71	12913060.93	3466327.249
桂林	110.2	25.24	12267407.89	2905252.13	黄山	118.34	29.72	13173548.54	3467608.99
大理	100.23	25.59	11157552.56	2948388.666	宁波	121.62	29.86	13538676.47	3485566.813
郴州	113.01	25.77	12580215.65	2970622.233	黄石	115.04	30.2	12806194.22	3529283.971
赣州	114.93	25.83	12793949.08	2978040.905	杭州	120.21	30.25	13381715.99	3535725.66
福州	119.3	26.07	13280415.25	3007753.329	沙市	112.26	30.31	12496726.04	3543460.015
都匀	107.52	26.26	11969071.65	3031318.856	安庆	117.06	30.53	13031059.59	3571859.95
三明	117.64	26.26	13095624.9	3031318.856	武汉	114.3	30.59	12723817.8	3579616.53
攀枝花	101.72	26.58	11323418.6	3071095.583	成都	104.07	30.65	11585019.41	3587377.915
六盘水	104.83	26.59	11669622.22	3072340.389	宜昌	111.29	30.69	12388746.13	3592554.846
贵阳	106.63	26.65	11869997.3	3079811.514	上海	121.47	31.23	13521978.55	3662655.183
衡阳	112.57	26.89	12531235.08	3109735.412	芜湖	118.43	31.35	13183567.29	3678287.213
怀化	110	27.57	12245143.99	3194868.001	绵阳	104.68	31.47	11652924.3	3693939.201

5.2.3 不同情况下备份节点的确定

（1）情况一：上海备份节点的确定

对于上海，该节点仅与唯一节点相连，对此我们可将备份节点设为上海节点的备份或其周边附近，即其位置坐标为东经121.47±0.01，北纬31.23±0.01　。

（2）情况二：武汉备份节点的确定

武汉与信阳和黄石两座城市相连。我们根据两点间直线最短确定备份节点的地理位置。由表 5.4 可得，信阳、黄石这两座城市对应的地理位置坐标分别为：x1=(12700440.7,3783016.604),x2=(12806194.22,3529283.971)。根据两点间直线最短原则，备份节点可以处于信阳、黄石两点连线的线段任意处，但考虑到所设备份节点应尽可能的靠近武汉 x3=(12723817.8,3579616.53)，故这里选择武汉与两点连线的垂足处建立备份节点，其计算公式为：

$$\begin{cases} \dfrac{x-x1}{y-y1}=\dfrac{x-x2}{y-y2} \\ \dfrac{x-x3}{y-y3}*\dfrac{x-x1}{y-y1}=-1 \end{cases} \tag{18}$$

求解得备份节点的投影坐标为(12776050.08,3601554.09)，其对应的地理位置坐标为东经 114.77，北纬 30.76。

（3）情况三：北京备份节点的确定

对于北京此节点，其一共有三个节点与之相连。为使引入的备份节点使其相互连通且路径最短，我们引入 Steiner 最小树模型，即以将 Steiner 最小树问题转化为组合优化问题，并利用“先粗后细”加启发式遗传算法搜索策略求解最优候选点[14]。

Step1.最小覆盖圆的确定。根据保定、天津、张家口这三座城市的位置坐标我们可以确定其最小覆盖圆区域，算法如下：

① 将所有点随机排布；

② 初始随意确定两点，设为P_1,P_2，以P_1P_2为直径创建初始圆，记为C_2（其中C_i表示包含前i个点的最小圆）；

③ 按顺序依次加点，设当前点为P_i：若P_i在当前圆C_{i-1}内，则$C_i=C_{i-1}$，否则转至④；

④ 以P_1P_i为直径得到C_i；

⑤ 确定不在C_i中的一点$P_j(j<i)$，同时直接以P_1P_j为直径得到C_j；

⑥ 确定不在C_j中的一点$P_k(k<j<i)$，那么P_i,P_j,P_k 一定在更新的圆的边界上，因为三点确定一个圆，故P_i,P_j,P_k构成了新的圆，一定能覆盖前i个点。

在实际情况中为简化计算我们也可以直接采用矩形区域代替最小圆覆盖搜索最优解，即（（min_longtitude,min_lagtitude）,(max_longtitude,max_lagtitude)）=（(114.89,38.87)，(117.2,40.77)）。

Step2.优化目标函数的确定。

$$\min f(a)=\sum_{i=0}^{m}\min\left(a_i-l_j\right)+T_a,\qquad 1\le j\le k,$$
$$s.t.\ 114.89\le longtitude\left(a_i\right)\le 117.2, 38.87\le lagtitude\left(a_i\right)\le 40.77 \tag{19}$$

上式中，a_i即为备份节点位置坐标，其总数目为m；

l_j即为连接节点的位置坐标，其总数目为k；

T_a为各备份节点间的最短路径，这里可根据问题一中的最小生成树求解。

Step3.“先粗后细”启发式遗传算法搜索最优解

遗传算法（GA）由 John Holland 根据生物进化论和遗传学的思想提出的一种全局优化算法，其利用遗传算子（选择，交叉，变异）促进解集合类似生物种群在自然界中的自然选择、优胜劣汰、不断进化并最终达到收敛状态。其在解决复杂的全局优化问题，尤其是 NP 难问题，如 TSP 旅行商问题，背包问题等均有较好的应用。

本文选择实码加速遗传算法求解目标函数（19），其算法流程如下：

①优化变量的实数编码；

②父代群体的初始化；

③计算父代群体的适应度评价；

④进行选择操作，产生第一个子代群体；

⑤对父代的种群进行杂交操作；

⑥进行变异操作；

⑦演化迭代；

⑧将第一、二次产生的优秀个体变化区间作为下次迭代时的优化变量区间，

算法转入步骤①，缩小优秀个体的变化区间，实现加速演化。

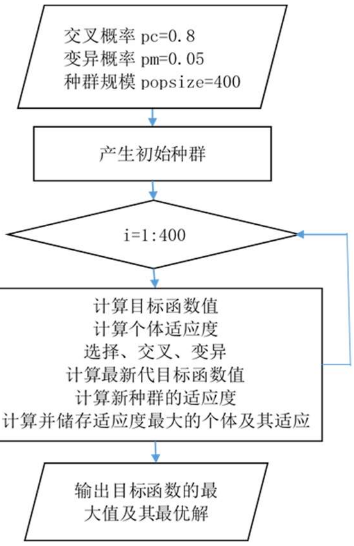


图 5.6　实码加速遗传算法

其中实码加速遗传算法（RAGA）的参数为：总群规模 $n=400$，交叉概率 $p_c=0.8$，变异概率 $p_m=0.05$。

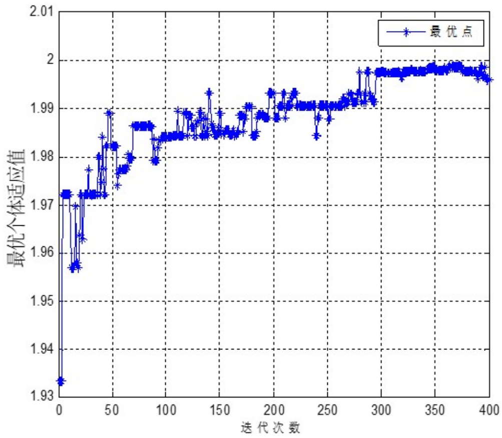


图 5.7　备份点数目为 3 时迭代曲线图

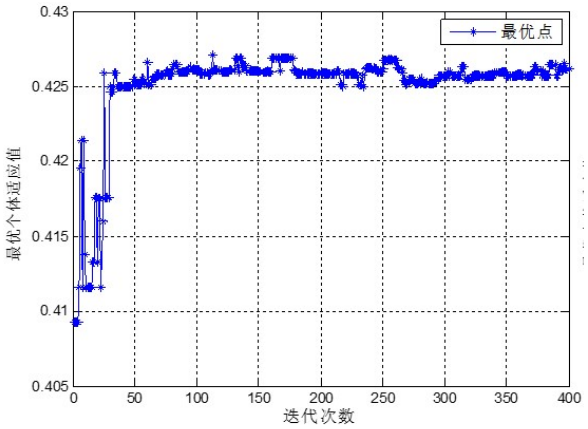


图 5.8　备份点数目为 1 时迭代曲线

从上图可以看出当迭代次数达到 200 左右时，其训练已经收敛。

若当备份节点数目为 0 时该问题即转化为对于给定城市节点，就最小生成树问题，根据 Prim 算法即可求解，这里我们将对此情况不进行讨论。本论文对给定备份节点数目为[1, 10]区间内求解最短路径，实验结果如下图所示：

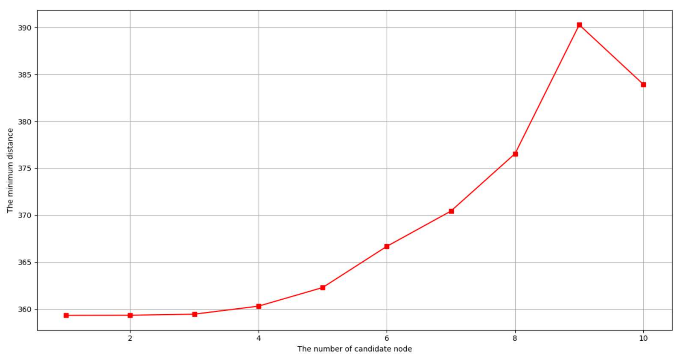


图 5.9　网络最短路径随备份点数目增加折线图

从上图可以看出当故障节点边数较少时，随着备份节点数目的增加其网络连通的最小距离也将增加，且随着备份节点数目增加至 5 个以上时其距离将急剧增加，这表明当网路中存在较多冗余节点时，其备份节点间的通讯路径将会迅速增加，而占据主导地位，这将使网络的整体路径变长，从而带来来不必要的成本和复杂度。故这里我们选择北京的备份节点数目为 1，其位置坐标为东经 115.187，北纬 39.212。

最终备份节点在图中的分布如下：

图 5.10　备份节点的确定与网络的修复图

其中上海备份节点的位置为东经 121.46，北纬 31.24；武汉备份节点的选择为东经 114.77，北纬 30.76,；北京备份节点的选择为东经 115.79，北纬 39.21。其图的连接方式与问题一保持一致。

对于满足最短路径的网络修复，其流程图如下所示：

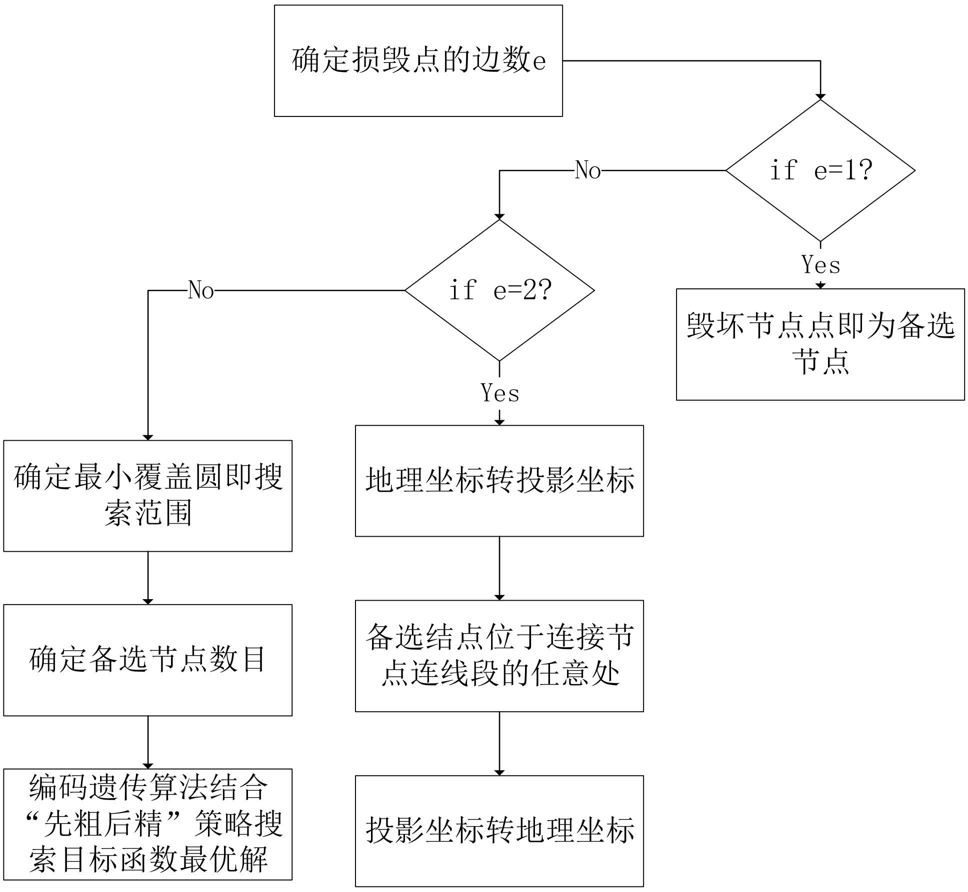


图 5.11　备份节点的确定与网络的修复流程图

由图 5.11 所示，对于备份节点的选择与网络的修复，首先我们判断其故障节点的变数目：若边数目为 1，则备份节点的选择可考虑故障节点的周边位置；若边数目为 2，则首先需要完成地理坐标与空间坐标的相互转化，此外，备份节点建议选择由其连接节点的线段与故障节点的垂足确定（垂足落在线段内，如垂足在线段外，则备份节点为线段上的任一点）；若边的数目超过 2 时，我们将在一定的区间内逐一计算最短路径所需的节点数目和位置坐标，而其坐标与数目的确定由实码加速遗传算法与“先粗后精”搜索策略求得。一般情况下当连接节点数目较少时其备份节点也将倾向于较少数目。

5.3 网络的修复与连通性能指标

问题三要求在 9 个城市节点同时故障的情况下，恢复通信网络且使得其路径最短。按照问题二的基本方法，这里我们首先对这 9 个城市节点的连通节点进行分析，并根据实码加速遗传算法求解节点数目在[1,15]区间内最短路径与最小生成树。对于网络连通性指标，本文借鉴图论中的点连通度和边连通度的概念，领用复杂网络的自然连通度刻画网络的连通性。

5.3.1 故障节点及其连通节点分析

根据题目所述，我们对 9 个故障城市节点其位置及连通节点信息进行分析，如表 5.4 所示：

表 5.4　故障节点其连接节点

武汉	信阳、黄石	信阳	南阳、武汉

黄石	武汉、九江	南昌	九江
岳阳	常德、长沙、沙市	九江	黄石、南昌、景德镇、安庆
沙市	岳阳、宜昌	安庆	九江、合肥
宜昌	沙市、襄樊		

分析上表发现，其故障节点及其相关节点均集中在东经111.29~117.23，北纬28.23~32.99这一区域内，较为集中。且对于武汉、黄石、沙市、南昌这些城市其连接节点也均完全损毁，而像岳阳、信阳、九江、安庆这些城市其均还与外部城市，如襄樊、南阳、景德镇、合肥城市节点有连接。因此该问题一共涉及武汉，黄石，岳阳，沙市，宜昌，信阳，南昌，九江，安庆，襄樊，南阳，景德镇，合肥共12所城市。

5.3.2 备份节点的确定与网络的恢复

依据 5.3.1 对故障节点的位置信息及问题二所述方法，此问题应在东经111.29~117.23，北纬28.23~32.99区域内利用实码加速遗传算法求解备份节点的数目和位置，使其连通，且路径最短。这里我们设置备份节点的数目为1～15，其最小距离随节点数目变化的曲线如下所示：

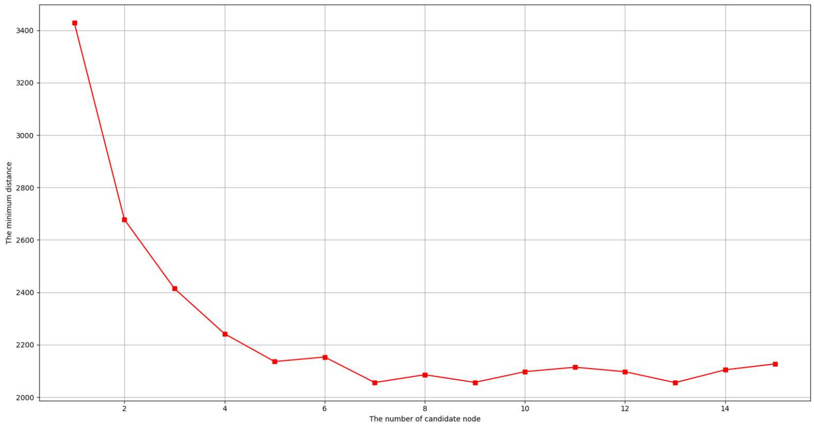


图 5.12　备份节点数目与最短路径曲线图

表 5.5　备份节点数目与最短路径表

备份节点数目	1	2	3	4	5	6	7	8
最短路径	3429.03	2677.83	2414.48	2241.90	2135.75	2153.02	2055.71	2085.46
备份节点数目	9	10	11	12	13	14	15	
最短路径	2056.30	2097.07	2144.09	2097.02	2055.40	2104.26	2126.73	

从上图可以看出随着节点数目的增加其最短路径迅速减少，当备份节点数目为7时最短路径达到最小为2055.71km，此后随着节点数目的继续增加最短路径长度开始出现波动，即增加节点数目将不会对路径的减少带来更大的收益。考虑到网络的复杂程度及成本，这里我们选择节点数目为7，其位置信息如表5.6所示：

表 5.6　备份节点位置信息

备选节点	1	2	3	4	5	6	7
西经	116.10	113.78	112.60	114.91	112.24	116.28	113.13
北纬	29.52	32.13	31.85	30.18	30.42	30.28	29.71

其连接方案如下：

（1）　备份节点的连接方式：(1,6)，(6,4)，(4,7)，(7,5)，(5,3)，(3,2)，其备份节点构成的最小生成树距离为 793.49km。

（2）　各个节点与备份节点的连接方式为：（长沙，7），（南阳，3），（南昌，1），（常德，7），（景德镇，1），（岳阳，7），（九江，1），（黄石，4），（沙市，5），（安庆，6），（武汉，4），（宜昌，5），（合肥，6），（襄樊，3），（信阳，2）。

图 5.12　备份节点数目与最短路径曲线图

图 5.12 即为备选节点及连接方案示意图。该子网络的总长度为 2055.71km，与原始网络的长度相比其距离减小了 14.34%，因此其在结构上更加优化，距离更短，若备份节点与故障城市节点间不要求保持连通，则网络的路径长度还将大幅降低。

5.3.3 自然连通度刻画网络连通性

无论是公路网、通信网还是社交网等，对于这类复杂网络在数学上均可以描述为图 $G=(V,E)$，其中 $V=\{v_1,v_2,...,v_N\}$ 为节点集合，$E=\{e_i,e_2,...,e_w\}\subseteq V\times V$ 为边集合。同时记 $N=|V|$ 为节点数量，$W=|E|$ 为边数量。我们可以利用简单邻接矩阵 $A(G)=(a_{ij})_{N\times N}$ 表示简单的无权图 G，若 v_i,v_j 节点间连通，则 $a_{ij}=1$，否则为 0。

网络的抗毁坏和连通性显然由网络中冗余边的数量决定，若两节点间冗余边或可替代途径数目越多则网络的连通性和鲁棒性将越强，也即网络系统的可靠性

将越高。故可以通过统计网络中所有节点v_i, v_j间不同边长度k的数目$n_{ij}{}^k$反映网络整体的连通度：

$$S = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=0}^{\infty} n_{ij}{}^k \tag{20}$$

上式（20）对于复杂网络将很难计算，其时间成本将较大，因此在实际中网络中闭途径的数目来衡量网络的冗余度（记简单图途径为$w = v_0 v_1 \ldots v_k$，若$v_0 = v_k$则称w为闭途径）。

这里令$n_i{}^k$表示起点和终点均为v_i，长度为k的闭途径数目，则式（20）将转化为：

$$S = \sum_{i=1}^{N}\sum_{k=0}^{\infty} n_{ij}{}^k = \sum_{k=0}^{\infty}\sum_{i=1}^{N} n_i{}^k = \sum_{k=0}^{\infty} n_k$$

上式中，n_k为网络中所有长度为k的闭途径的数目。若S越大，则网络的冗余性将越高，网络连通度越高。此外，由于网络途径允许节点和边重复，这意味着闭途径的长度可以为任意长度，因此$S \to \infty$。为克服这一问题，一般对n_k进行加权处理，即有：

$$S' = \sum_{k=0}^{\infty} \frac{n_k}{k!} \tag{21}$$

从式（21）可以看出，越长的途径被重复计算的次数越多且对网络抗毁性贡献越小。记$n_k = \sum_{i=1}^{N} \lambda_i^k$，将其代入（21）有：

$$S' = \sum_{k=0}^{\infty} \frac{n_k}{k!} = \sum_{k=0}^{\infty} \frac{\sum_{i=1}^{N} \lambda_i^k}{k!} = \sum_{i=1}^{N}\sum_{k=0}^{\infty} \frac{\lambda_i^k}{k!} = \sum_{i=1}^{N} e^{\lambda_i} \tag{22}$$

当N较大时，S的值将很大，故对式（22）求对数有：

$$\bar{\lambda} = \ln\left(\frac{S'}{N}\right) = \ln\left(\frac{1}{N}\sum_{i=1}^{N} e^{\lambda_i}\right) \tag{23}$$

从式(23)可以看出$\bar{\lambda}$是所有特征根关于自然指数和自然对数的特殊平均值，故将其称为图G的自然连通度[15-17]，其中λ_i为图G邻接矩阵$A(G)$的特征根。

此外，自然连通度关于边的增、删是严格单调的，即自然连通度对于网络连通性的分辨率较高，其能很好的反映网络细微差别。此外，其他抗毁性和连通性指标相比，自然连通度能够准确、清晰地刻画出复杂网络抗毁性的演化过程，而且自然连通度从复杂网络的内部结构属性出发，通过对网络中不同长度闭途径数目进行加权求和反映网络中替代途径的冗余性，其数学形式简洁且具有明确的物理意义，可解释性强。因此本论文奖使用网络的自然连通度反映刻画网络的连通性和鲁棒性。

5.4 备份节点的确定与网络的修复

由问题四即第三问分析知，网络的连通即鲁棒性两者间存在“跷跷板”现象，

即当追求网络的最短路径时其冗余程度必然较少，鲁棒性也必然降低。但是若追求较高的连通性则网络的冗余结构也将相当复杂，此时无论对于时间还是财力均是不划算甚至不可接受的。因此如何在网络的连通性和路径长度将找到一个最佳的平衡这是第四问研究的重点。一个朴素的想法是对于网络中的关键节点进行冗余设计，而对于其他简单节点其连通性可以保持相对较低。对此我们根据图论中的最小支配集，利用贪心算法寻找问题一中设计网络的最小支配集，然后对最小支配解中的顶点进行冗余备份，同时添加连边提高网络的鲁棒性，最后进行模拟仿真实验验证网络的性能。

5.4.1 网络最小支配集的确定

最小支配集[18]是指对于图$G=\left(V,E\right)$，从V中取尽量少的点组成一个集合，使得V中剩余的点都与取出来的点有边相连。即，设$V^{'}$是图的一个支配集，则对于图中的任意顶点u，其要么属于集合$V^{'}$，要么与$V^{'}$中的顶点相邻。此外，在$V^{'}$中除去任何元素后$V^{'}$将不再是支配集，则称此支配集$V^{'}$为最小支配集，也即G的所有支配集中顶点个数最少的支配集。最小支配集中的顶点个数称为支配数，通过确定最小支配集即可发现重要节点。

对于最小支配集的确定其为 NP 难问题[19]，对此我们采用禁忌搜索与遗传算法相结合的 TSGA 算法[20][21]寻求最优解。TSGA 相比与传统的 GA 算法其有效克服了 GA 算法局部搜索能力较差，收敛慢且易早熟的问题，以更好更快的获得全局最优解，其算法流程如下：

Step1. 初始化：个体编码为 0,1 组成长度为 n 的基因库，种群规模为 N，初始种群为$\left\{S_1{}^0,S_2{}^0,...,S_n{}^0\right\}$，对每个个体校正为合法个体；

Step2. 适应度函数：取适应度函数为$F\left(S\right)=\dfrac{1000}{\sum_{i=1}^{n}x_i+n\left|C\left(S\right)\right|}$；

Step3. 选择算子：根据轮盘选择算子；

Step4. 交叉算子：单点交叉，概率为p_c，若子代优于父代则替换，否则以概率$p=\exp\left(\dfrac{f\left(S_p\right)-f\left(S_c\right)}{L}\right)$替代父代，并将子代校正为可行解；

Step5. 禁忌搜索：对每个个体进行禁忌搜索，对第j代种群中的第i个个体$S_i{}^j$进行禁忌搜索，并更新禁忌表，最优解以及当前个体；

其参数设置为，$N=50,p_c=0.6,L=n$，迭代次数$Tsgen=15$，禁忌长度为$Tlength=0.2n$，其流程图如下：

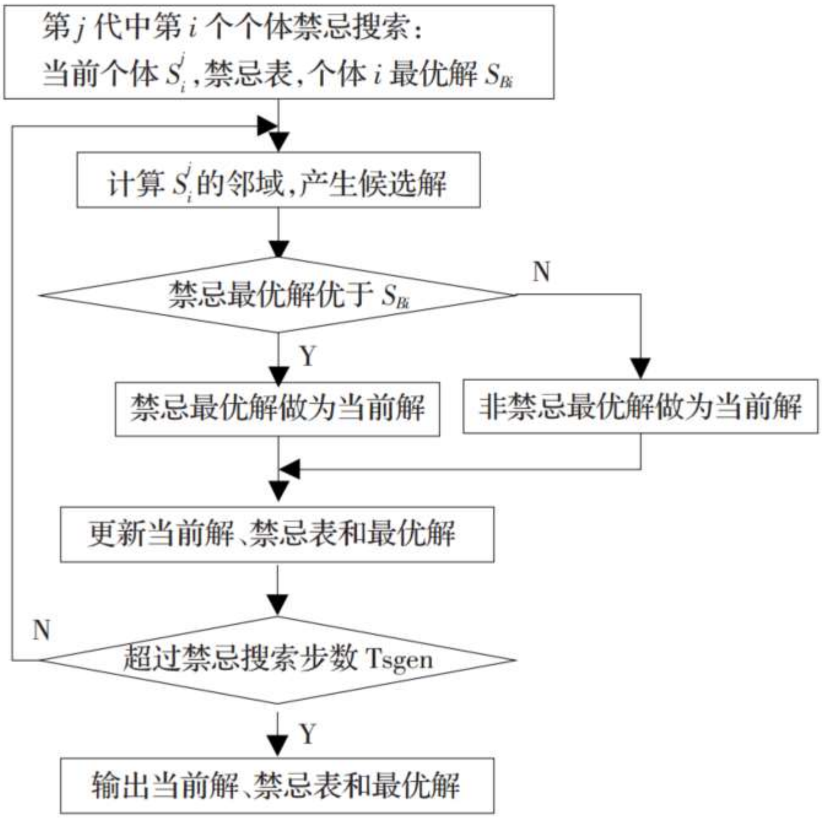


图 5.13 TSGA 流程图

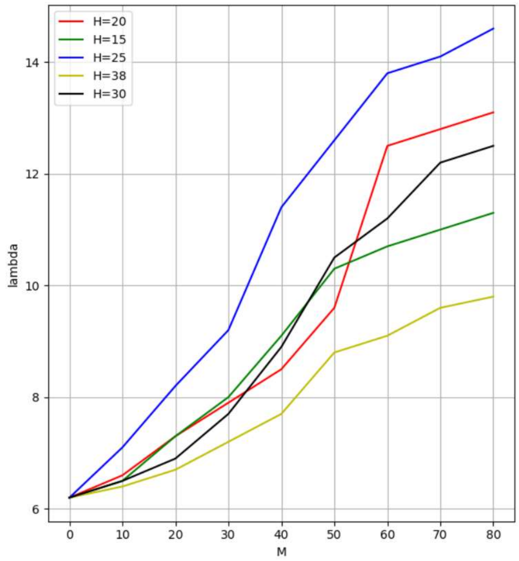
TSGA 搜索最小支配集为{'九江 '，'齐齐哈尔'，'南宁'，'岳阳'，'襄樊'，'南阳'，'景德镇'，'合肥'，'无锡'，'杭州'，'徐州'，'韶关'，'洛阳'，'运城'，'临汾'，'石家庄'，'北京'，'唐山'，'呼和浩特'，'烟台'，'鞍山'，'四平'，'广州'，'深圳'，'宝鸡'，'龙岩'，'兰州'，'哈尔滨'，'牡丹江'，'宜宾'，'贵阳'，'攀枝花'，'德令哈'}共 33 个节点。对此我们对其增加边，以提高网络的连通性。

5.4.2 添加连边提高网络连通性

提高网络的连通性以增加网络的连通性一般做法是对网络中的关键节点进行冗余备份，同时增加网络的变得数量，以优化网络结构增加网络的连通性[22]。而当网络遭到破坏时其可以有效的保证网络的连通，提供可靠的通信保障。然而关键节点和边的选择也并非随意，无限制的确定，网络边数与节点数目的增加必将导致网络结构的复杂，其网络路径长度也将必然增加，对此我们选择对最小支配集中的节点进行冗余备份同时增加其连边，具体做法如下：

假设选择关键节点数为H，添加边数为M 。这里我们对关键节点P，即支配集顶点进行冗余设计，即在其附近位置进行布置备份节点。平时正常通讯期间备份节点不接入网络，而当主节点遭到大量严重损坏而影响到网络的正常连通时可启动备份节点。同时，我们对距离较近（关键节点间相互连通，或其间最短通路不超过r个节点，这里r取 3）的关键节点P以概率p_a添加连边，而对于距离远的关键节点则以$1-p_a$添加节点，这里p_a取 0.75。

网络的鲁棒性由自然连通度$\overline{\lambda}$确定，下图给出了选择不同关键节点数目下网络连边的数量与自然连通度的曲线，以及网络 10%节点故障时，其网络连通度的仿真曲线。



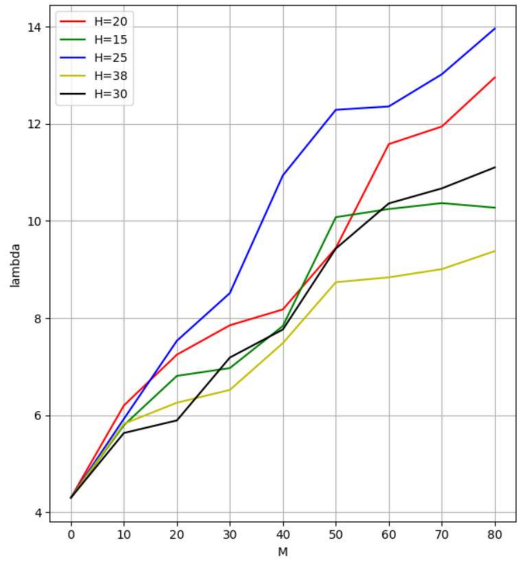


图 5.13　网络连通度变化曲线　　5.14　随机破坏 10%节点网络连通度变化曲线

从上图 5.13 可以看出随着边数目的增加网络连通度将快速增加然后趋于平缓，因此当增加连边数目为 40-50 时能获得最大的网络连通性能提升，同时也保证了较小的路径长度。同时关键节点数目的选择对网络连通性的影响较大，在网络的增加边数固定时，随着关键节点数目的增加其网络连通性性将迅速增加然后缓慢减少，呈偏态扭斜的单峰曲线。当关键节点数目取 25 时其效果最优，从图 5.14 可以看出当随机删除 10%节点后，当连边数目较少时其对网络的连通性影响较大，而当连边数目增加时网络的连通性将快速增加，其后区域平稳缓慢上升，这说明了本论文方案对网络的鲁棒性和连通性有一定的效果，且当节点数目选择为 25，边的添加数目为 60 时，网络的鲁棒性高且路径相对较短。此外，在实际情况下配合备份节点的补充，网络的连通性将进一步提高，路径也将会有所缩短。

图 5.15　备份节点数 25 连边数 30 网络结构　图 5.16　备份节点数 25 连边数 50 网络结构

图 5.15，图 5.16 为备份节点数为 25 时，网络连边数分别增加 30,50 时网络的结构。

6 模型的评价与推广

6.1 模型的评价

6.1.1 问题一模型评价

针对问题一，本文首先通过经纬弧度转换公式，将经纬度转换为弧度，此后利用 Great-circle 公式计算各个城市节点间的球面距离，即劣弧距离。同时引入图论中的最小生成树模型，利用 Prim 算法求解。该方法时间复杂度低且思想简单易实现。最后我们使用 Python smopy 类库实现了大型复杂网络的可视化，简洁直观，易分析。但是对于节点的连接方案，本论文没有进行详细的分析，同时对于各个节点的重要程度需要进一步讨论。

6.1.1 问题二模型评价

针对问题二本论文在第一问最小生成树连接方案的基础上进行展开，即首先确定损坏节点的连接节点，然后对于不同的连接点数目进行分类讨论，以简化问题求解。而对于连接节点数目较多的情况，其问题本质上是 Steiner 树的构建问题，该问题目前并没有很好的解决方案，本论文首先完成地理坐标与投影坐标的相互转换，然后通过构建最小连接路径的目标函数，同时使用“先粗后精”的搜索策略减少时间复杂度，并利用实码加速遗传算法这类启发式算法求解问题，以避免传统暴力穷举的时间成本。

6.1.3 问题三模型评价

问题三为问题二的推广，其本质仍为 Steiner 树的构建。针对问题三，对此本文首先分析故障节点的位置信息和相邻点。同时本论文还给出了不同备份节点数目与最短路径间的变化曲线，这对结点数目的确定有一定的参考价值。针对网络连通性的判别指标，本文借鉴复杂网络鲁棒性的衡量准则，引入网络的自然连通度。相较于其它指标，自然连通图对于图形结构的细微差异有较高的分辨率，且计算量相对较小，可解释性强。然而对于问题二、问题三种网络备选节点的设计与构建本文仅考虑连通路径最短，且进行了较为简单的仿真实验，者仍需进一步的实验研究。

6.1.3 问题三模型评价

针对问题四，本文以前三问中备选节点的选择策略以及网络的连通性指标为基础，通过引入图论中的最小支配集模型确定网络的关键节点，对其进行冗余备份，同时在关键节点上增加连边以提高网络的连通性和鲁棒性，通过仿真模拟实验寻找最优的关键节点数以及连边数目。对于大型复杂网络的设计和鲁棒性的研究有着一定的参考价值。然而由于最小支配集的求解为 NP 难问题，传统方法求解复杂，对此本文选择使用禁忌搜索与遗传算法相结合的策略确定最优解，以避免传统 GA 早熟、不收敛的情况。在实际工程中为简化计算也可将最小支配集顶点替换为度较大的端点，且拥有相似的效果。本论为对与连边的添加只在关键节点处随机进行，既没有考虑关键节点外的网络连通性，同时也没有考虑网络的最短路径，因此该问题仍需进一步研究。

6.2 模型的推广

本论文研究问题本质上是给出大型复杂网络的设计、修复以及鲁棒性的最优方案，其为图论中的经典且火热的研究方向。在实际工程应用还是军事战争领域均有较大的价值，如大型复杂的公路、水路网系统，电力运输，仓储选择，通信网络，复杂、高可靠传感网络等。该问题拓展面较宽且研究内容复杂多样，对于较多下游问题均具有很强的辐射作用。此外，该问题同社交网络的研究也具有部分交叉，目前深度学习“红的发紫”，是否可以通过最新的 Graphic Neural Network

与 Reinforcement Learning 结合，解决此类问题仍需要进行不的研究和探索。

7 参考文献

[1] 方锦清，汪小帆，刘曾荣．略论复杂性问题和非线性复杂网络系统的研究 [J]．科技导报，2004,22(2):9-12,64.
[2] 谭跃进，吕欣，吴俊，等．复杂网络抗毁性研究若干问题的思考[J]．系统工程理论与实践，2008,28（Suppl）:116-120.
[3] 吴俊，谭跃进．复杂网络抗毁性测度研究[J]．系统工程学报，2005,20(2):128-131.
[4] 谭跃进，吴俊，邓宏钟．复杂网络抗毁性研究综述[J]．系统工程，2006,24(11):1-5.
[5] Porcu E, Bevilacqua M, Genton M G. Spatio-temporal covariance and cross-covariance functions of the great circle distance on a sphere[J]. Journal of the American Statistical Association, 2016, 111(514): 888-898.
[6] 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.
[7] Spira P M, Pan A. On finding and updating spanning trees and shortest paths[J]. SIAM Journal on Computing, 1975, 4(3): 375-380.
[8] 鲍捷, 陆林, 吉中会. 基于最小生成树 Kruskal 算法的皖北地区旅游交通优化与线路组织[J]. 人文地理, 2010, 25(3): 144-148.
[9] 江波, 张黎. 基于 Prim 算法的最小生成树优化研究[J]. 计算机工程与设计, 2009, 30(13): 3244-3247.
[10] Hwang F K, Richards D S. Steiner tree problems[J]. Networks, 1992, 22(1): 55-89.
[11] Garey M R, Johnson D S. The rectilinear Steiner tree problem is NP-complete[J]. SIAM Journal on Applied Mathematics, 1977, 32(4): 826-834.
[12] 胡大伟, 陈诚. 遗传算法 (GA) 和禁忌搜索算法 (TS) 在配送中心选址和路线问题中的应用[D]. , 2007.
[13] 刘健, 刘高峰. 高斯-克吕格投影下的坐标变换算法研究[D]. , 2005.
[14] 郑健体, 吉国力, 吴瑞意. 基于遗传算法的通讯网络最佳 Steiner 树构造[J]. 厦门大学学报 (自然科学版), 2008 (3): 5.
[15] 王班, 马润年, 王刚. 基于自然连通度的复杂网络抗毁性 研究[J]. 计算机仿真, 2015, 32(8): 315-318.
[16] 吴俊, 谭索怡, 谭跃进, 等. 基于自然连通度的复杂网络抗毁性分析[D]. , 2014.
[17] Yi-Lun S. Local natural connectivity in complex networks[J]. Chinese Physics Letters, 2011, 28(6): 068903.
[18] Guha S, Khuller S. Approximation algorithms for connected dominating sets[J]. Algorithmica, 1998, 20(4): 374-387.
[19] 彭伟, 卢锡城. 一个新的分布式最小连通支配集近似算法[D]. , 2001.
[20] 廖飞雄, 马良. 禁忌遗传算法求解最小支配集[D]. , 2007.
[21] Jiang Y S, Chen W M. Task scheduling for grid computing systems using a genetic algorithm[J]. The Journal of Supercomputing, 2015, 71(4): 1357-1377.
[22] 田田, 吴俊, 谭跃进. 基于自然连通度的复杂网络抗毁性仿真优化研究[J]. 复杂系统与复杂性科学, 2013, 10(2): 88-93.

附录

附录 1 Python 代码

```
from math import radians, cos, sin, asin, sqrt
import pandas as pd
import numpy as np
from random import uniform
import csv
import codecs
from pylab import *
import matplotlib.pyplot as plt
import smopy
import pyproj
from random import sample
from collections import Counter
from itertools import combinations

from mpl_toolkits.mplot3d import Axes3D


def haversine(lon1, lat1, lon2, lat2):  # 经度 1，纬度 1，经度 2，纬度
2 （十进制度数）
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # 将十进制度数转化为弧度
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # haversine 公式
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) **
2
    c = 2 * asin(sqrt(a))
    r = 6371.393  # 地球平均半径，单位为公里
    return round(c * r, 4)


def distance(city_list, longtitude, lagtitude):
    dis = []
    count_num = len(city_list)
    for i in range(count_num):
```

```
        dis_temp = []
        # dis_temp.append(str(city_list[i]))
        for j in range(count_num):
            dis_temp.append(
                haversine(float(longtitude[i]),  float(lagtitude[i]),
float(longtitude[j]), float(lagtitude[j])))
        dis.append(dis_temp)
    return dis


def data_write_csv(file_name, datas):  # file_name 为写入 CSV 文件的路
径，datas 为要写入数据列表
    file_csv = codecs.open(file_name, 'w+', 'utf-8')  # 追加
    writer  =  csv.writer(file_csv,  delimiter='  ',  quotechar='  ',
quoting=csv.QUOTE_MINIMAL)
    for data in datas:
        writer.writerow(data)


def longtitude2rad(longtitude):
    rad = []
    for i in longtitude:
        temp = round(float(i) * np.pi / 180, 4)
        rad.append(temp)
    return rad


# path = r'../data.csv'
# location_raw = pd.read_csv(path, encoding='utf-8', engine='python')
# location_raw.to_csv(r'../temp')
city = []
east_longtitude = []
north_latitude = []
with open(r'../temp', encoding='utf-8') as data:
    line = data.readlines()
    for i in range(1, len(line)):
        temp = line[i].split(',')
        city.append(temp[0])
        east_longtitude.append(temp[1])
        north_latitude.append(temp[2].replace('\n', ''))
dis_city = distance(city, east_longtitude, north_latitude)
dis_city_num = dis_city
city_name = [' '] + city
dis_city = [city_name] + dis_city
```

```
longtitude_rad = longtitude2rad(east_longtitude)
latitude_rad = longtitude2rad(north_latitude)
dataframe_radian = pd.DataFrame({'city': city, 'longtitude_radian': longtitude_rad, 'latitude_radian': latitude_rad})


# dataframe_radian.to_csv(r'../radian.csv', encoding='utf-8')
# data_write_csv(r'../distance.csv', dis_city)


def prim(dis_list):
    # Display the Chinese word
    rcParams['font.sans-serif'] = ['SimHei']
    rcParams['axes.unicode_minus'] = False

    INFINITY = 6553500  # 代表无穷大
    vexs = array(dis_list)

    lengthVex = len(vexs)  # 邻接矩阵大小
    adjvex = zeros(lengthVex)  # 连通分量，初始只有第一个顶点，当全部元素为 1 后，说明连通分量已经包含所有顶点
    adjvex[0] = 1
    lowCost = vexs[0, :]  # 记录与连通分量连接的顶点的最小权值，初始化为与第一个顶点连接的顶点权值
    lowCost[0] = 0
    count = 0
    point = []
    edge = []
    while (count < len(dis_list)):
        I = (argsort(lowCost))[count]
        point.append(I)
        # print("Vertex  [", count, "]:", I)
        adjvex[I] = lowCost[I]
        # print("Edge [", count, "]:", adjvex[I])
        edge.append(adjvex[I])
        lowCost[I] = 0
        lowCost = array(list(map(lambda x, y: x if x < y else y, lowCost, vexs[I, :])))
        count = count + 1

    # print("The length of the minimum cost spanning tree is: ", sum(adjvex))
    return point, edge, sum(adjvex)
```

```
# p, e, sum_edge = prim(dis_city_num)


def primToMST(adj, startPoint="1", num=139):
    indexdict = {i + 1: str(i + 1) for i in range(num)}
    citydict = {str(i + 1): i + 1 for i in range(num)}
    vnew = [startPoint]
    edge = []
    sum_init = 0
    # vnew
    edg = []  # element is cell, eg. (u,v)
    while len(vnew) < len(citydict):
        imin = (-1, float('Inf'))
        centerCity = ""
        for city in vnew:
            cur = citydict[city] - 1
            ws = adj[cur]
            for (i, w) in enumerate(ws):
                if indexdict[i + 1] not in vnew and 0 < w and w <
imin[1]:
                    imin = (i + 1, w)
                    centerCity = city
        vnew.append(indexdict[imin[0]])  # add the city with minimum
weight
        edge.append((centerCity, indexdict[imin[0]]))
        sum_init += imin[1]
    return sum_init, vnew, edge


sum_total, vnew, edges = primToMST(dis_city_num)
start_point = []
end_point = []
for v, u in edges:
    # print("(" + str(int(v)-1) + "," + str(int(u)-1) + ")")
    start_point.append(int(v) - 1)
    end_point.append(int(u) - 1)



# pd.DataFrame({'vertex_edge': p, 'edge_length': e}).to_csv(
#     r'../graph_1.csv', encoding='utf-8')
```

```
def show_plt(north_latitude, east_longtitude, starts, ends):
    num = []
    total = starts + ends
    for i in total:
        num.append(total.count(i))
    north_latitude_int = []
    east_longtitude_int = []
    for i in range(len(north_latitude)):
        north_latitude_int.append(float(north_latitude[i]))
        east_longtitude_int.append(float(east_longtitude[i]))
    hz = smopy.Map(
        (int(min(north_latitude_int) - 0.5),
int(min(east_longtitude_int) - 0.5), int(max(north_latitude_int) +
0.5),
         int(max(east_longtitude_int) + 0.5)), z=10)
    # hz.save_png(r'../map.png')
    x_list = []
    y_list = []
    for i in range(len(north_latitude)):
        x, y = hz.to_pixels(north_latitude_int[i],
east_longtitude_int[i])
        x_list.append(x)
        y_list.append(y)
    ax = hz.show_mpl(figsize=(8, 6))
    ax.plot(x_list, y_list, 'or', ms=2, mew=2)
    x = []
    y = []
    for i in range(len(starts)):
        x.append([x_list[int(starts[i])], x_list[int(ends[i])]])
        y.append([y_list[int(starts[i])], y_list[int(ends[i])]])
    for j in range(len(x)):
        plt.plot(x[j], y[j])
    plt.show()
    # ax.save(r'../map.png')


total_point = start_point + end_point
point_num = Counter(total_point).most_common(33)
key_city = [city[i[0]] for i in point_num]
print(key_city)
key_point = [i[0] for i in point_num if i[1] > 2]
add_edge = [sample(key_point, 2) for _ in range(30)]
start_point_add = start_point + [i[0] for i in add_edge]
end_point_add = end_point + [i[1] for i in add_edge]
```

```
#    show_plt(north_latitude,    east_longtitude,    start_point_add,
end_point_add)

# show_plt(north_latitude, east_longtitude, start_point, end_point)
beijing = []
wuhan = []
shanghai = []
for i in range(len(start_point)):
    if start_point[i] == 104:
        beijing.append(end_point[i])
    if end_point[i] == 104:
        beijing.append(start_point[i])
    if start_point[i] == 52:
        wuhan.append(end_point[i])
    if end_point[i] == 52:
        wuhan.append(start_point[i])
    if start_point[i] == 55:
        shanghai.append(end_point[i])
    if end_point[i] == 55:
        shanghai.append(start_point[i])


# print(beijing)
# print(wuhan)
# print(shanghai)


def long_lat2x_y(a1, a2, reverse=False):
    p1 = pyproj.Proj(init="epsg:4326")  # 定义数据地理坐标系
    p2 = pyproj.Proj(init="epsg:3857")  # 定义转换投影坐标系
    if reverse == False:
        # a1=lons, a2=lats
        x1, y1 = p1(a1, a2)  # 转换经纬度到投影坐标系统
        x2, y2 = pyproj.transform(p1, p2, x1, y1, radians=True)
    if reverse == True:
        # a1=x, a2=y, x1=lons, y2=lats
        # x1, y1 = p2(a1, a2)  # 转换经纬度到投影坐标系统
        # x2, y2 = pyproj.transform(p2, p1, x1, y1, radians=True)
        x2, y2 = p2(a1, a2, inverse=True)  # 反向转换
    return x2, y2


north_latitude_int = []
east_longtitude_int = []
```

```
for i in range(len(north_latitude)):
    north_latitude_int.append(float(north_latitude[i]))
    east_longtitude_int.append(float(east_longtitude[i]))

x,       y       =       long_lat2x_y(np.array(east_longtitude_int),
np.array(north_latitude_int))
x_y_dataframe = pd.DataFrame(
    {'city':  city,  'longtitude':  east_longtitude_int,  'latitude':
north_latitude_int, 'x': x, 'y': y})
# x_y_dataframe.to_csv(r'../x_y.csv', encoding='utf-8')

x,   y   =   long_lat2x_y(np.array(12776050.08),   np.array(3601554.09),
reverse=True)


# print(x, y)


def cell(long_min, long_max, lat_min, lat_max):
    interval_long = round((long_max - long_min) / 100, 4)
    interval_lat = round((lat_max - lat_min) / 100, 4)
    grid_cell = []
    for i in range(100):
        long_temp = long_min + i * interval_long
        for j in range(100):
            lat_temp = lat_min + j * interval_lat
            grid_cell.append([round(long_temp,   7),   round(lat_temp,
7)])
    return grid_cell


def dis_tree(grid_cell, att_list):
    num = len(att_list)
    dis_matrix = []
    for i in range(num):
        dis_each = []
        for j in range(num):

dis_each.append(haversine(float(grid_cell[att_list[i]][0]),
float(grid_cell[att_list[i]][1]),

float(grid_cell[att_list[j]][0]), float(grid_cell[att_list[j]][1])))
        dis_matrix.append(dis_each)
    p_1, p_2, length_path = prim(dis_matrix)
```

```
    return length_path


def alternative(grid_cell, link_long, link_lat, num_attitude):
    link_num = len(link_long)
    num_total = len(grid_cell)
    idx = [i for i in range(num_total)]
    combins = [sample(idx, num_attitude) for _ in range(100000)]
    # combins = [c for c in combinations(range(num_total),
num_attitude)]
    min_dis = 10000000000
    best_tree = []
    for i in combins:
        tree_dis_temp = dis_tree(grid_cell, i)
        temp_edge = []
        dist_att = 0
        for k in range(link_num):
            dis_temp = []
            for j in i:
                dis_temp.append(haversine(float(link_long[k]),
float(link_lat[k]), grid_cell[j][0], grid_cell[j][1]))
            temp_edge.append(i[dis_temp.index(min(dis_temp))])
            dist_att += min(dis_temp)
        if dist_att + tree_dis_temp < min_dis:
            min_dis = dist_att + tree_dis_temp
            best_edge = temp_edge
            best_tree = i
    return min_dis, best_edge, best_tree


def itt(min_long, max_long, min_lat, max_lat, long_list, lat_list):
    grid_cell = cell(min_long, max_long, min_lat, max_lat)
    min_length_ittera = []
    best_way_ittera = []
    tree_cons_itera = []
    for i in range(1, 16):
        min_length, best_way, tree_cons = alternative(grid_cell,
long_list, lat_list, i)
        min_length_ittera.append(min_length)
        best_way_ittera.append(best_way)
        tree_cons_itera.append(tree_cons)
        print(min_length, best_way, tree_cons)
    print(min_length_ittera)
    print(best_way_ittera)
```

```
    print(tree_cons_itera)


# grid_cell = cell(114.89, 117.2, 38.87, 40.77)
# print(grid_cell[3918][0])
# print(grid_cell[3918][1])
# itt(114.89, 117.2, 38.87, 40.77, [115.46, 117.2, 114.89], [38.87,
39.09, 40.77])

def plot_iter(dis_best):
    att_num = [i + 1 for i in range(len(dis_best))]
    plt.plot(att_num, dis_best, 's-', color='r')
    plt.grid(True)
    plt.xlabel('The number of candidate node')
    plt.ylabel('The minimum distance')
    plt.show()


# dis_best = [359.32849999999996, 359.3414, 359.45550000000003,
360.3084, 362.28870000000006, 366.6741,
#                370.43539999999996, 376.556, 390.27610000000004,
383.9271]
# plot_iter(dis_best)

# north_latitude[104] = str(39.212)
# east_longtitude[104] = str(115.787)
# north_latitude[52] = str(30.76)
# east_longtitude[52] = str(114.77)
# north_latitude[55] = str(31.24)
# east_longtitude[55] = str(121.46)
#
# show_plt(north_latitude, east_longtitude, start_point, end_point)
city_index = []
for i in ['武汉', '黄石', '岳阳', '沙市', '宜昌', '信阳', '南昌', '九
江 ', '安庆']:
    city_index.append(city.index(i))
city_link_node = []

for k in city_index:
    city_temp = []
    for i in range(len(start_point)):
        if start_point[i] == k:
            city_temp.append(end_point[i])
        if end_point[i] == k:
```

```
            city_temp.append(start_point[i])
    city_link_node.append(city_temp)

city_name = []
for i in city_link_node:
    city_name_temp = []
    for j in i:
        city_name_temp.append(city[j])
    city_name.append(city_name_temp)

city_idx = []
for i in city_link_node:
    city_idx += i
city_idx = list(set(city_idx))
long_zoo = []
lat_zoo = []

for i in city_idx:
    long_zoo.append(east_longtitude[i])
    lat_zoo.append(north_latitude[i])
max_long = max(long_zoo)
min_long = min(long_zoo)
max_lat = max(lat_zoo)
min_lat = min(lat_zoo)
# print(max_long, min_long, max_lat, min_lat)

# itt(111.29, 117.23, 28.23, 32.99, long_zoo, lat_zoo)

dis_itt = [3429.0277000000006, 2677.832, 2414.48, 2241.898, 2135.7499,
2153.0175, 2055.7088000000003, 2085.4559,
           2056.2977, 2097.0714, 2114.0908, 2097.015, 2055.4033,
2104.2586, 2126.7276]
# plot_iter(dis_itt)
grid_cell = cell(111.29, 117.23, 28.23, 32.99)
att_point = [8127, 4282, 2276, 6141, 1646, 8443, 3131]
long_att = []
lad_att = []
for i in att_point:
    long_att.append(grid_cell[i][0])
    lad_att.append(grid_cell[i][1])
# print(long_att)
# print(lad_att)

dis = distance(att_point, long_att, lad_att)
```

```
sum_dis, vnew, edges = primToMST(dis, '1', num=7)
# print("weight sum: " + str(sum_dis))
# print("vertex:")
# [print(city) for city in vnew]
# print("edge: ")
# [print("(" + str(v) + "," + str(u) + ")") for (v, u) in edges]


long_sub = long_zoo + long_att
lat_sub = lat_zoo + lad_att
start_point = [12, 17, 13, 17, 16, 14, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11]
end_point = [17, 15, 18, 16, 14, 13, 18, 14, 12, 18, 12, 18, 12, 15,
16, 17, 15, 16, 17, 14, 13]
# show_plt(lat_sub, long_sub, start_point, end_point)
```