In [1]:

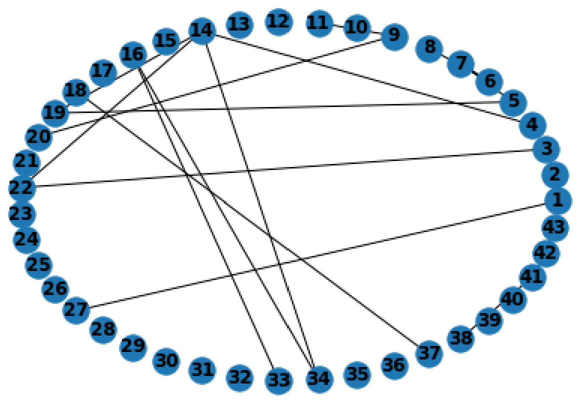
```
import os
import numpy as np
import pandas as pd
import networkx as nx

import plotly.io as pio
pio.renderers.default = "svg"
```

In [2]:

```
data_dir = "./../../history/2.4.3/"

edges = pd.read_excel(data_dir + "edges.xls").fillna("")
nodes = pd.read_excel(data_dir + "nodes.xls")

node_list = nodes["斑块编号"].values

edges_list = []
for idx, i in edges.iterrows():
    if i["途径斑块1编号"] == "" and i["途径斑块2编号"] == "":
        edges_list.append([int(i["起点斑块编号"]), int(i["终点斑块编号"])])
    elif i["途径斑块2编号"] == "":
        edges_list.append([i["起点斑块编号"], int(i["途径斑块1编号"])])
        edges_list.append([int(i["途径斑块1编号"]), i["终点斑块编号"]])
    else:
        edges_list.append([i["起点斑块编号"], int(i["途径斑块2编号"])])
        edges_list.append([int(i["途径斑块2编号"]), int(i["途径斑块1编号"])])
        edges_list.append([int(i["途径斑块1编号"]), i["终点斑块编号"]])

E_list = nodes["MESC指数"].values
```

In [3]:

```
G = nx.Graph()
G.add_nodes_from(node_list)
G.add_edges_from(edges_list)

import matplotlib.pyplot as plt
nx.draw_circular(G, with_labels=True, font_weight='bold')
```



In [4]:

```
display(edges, nodes)
```

	Number	名称参考	起点斑块编号	终点斑块编号	途径斑块1编号	途径斑块2编号	河流廊道长度（m）
0	1	苕溪	14	19			60110.484431
1	2	吴淞江	1	27			68350.600080
2	3	张家港	20	11	9.0		69176.237587
3	4	太浦河	14	33	16.0	34.0	40171.377566
4	5	望虞河	3	14	22.0		61266.648248
5	6	西苕溪	6	8			135424.869181
6	7	埭溪	19	5			33266.407168
7	8	合溪	4	14			42274.548963
8	9	浒溪	5	7			39757.282515
9	10	南溪	6	7			47819.594147
10	11	泗安溪	18	37			46180.726343
11	12	钱塘江	43	38	41.0	42.0	107788.770473
12	13	长江	2	3			122488.646029

	斑块编号	X坐标	Y坐标	MESC指数	斑块面积（m2）	斑块土地利用类型
0	1	1.461940e+06	3.420439e+06	2.3	1.038614e+07	湖塘
1	2	1.468556e+06	3.496538e+06	2.1	7.512075e+06	湿地
2	3	1.474689e+06	3.488331e+06	2.0	5.905125e+06	湿地
3	4	1.386035e+06	3.401217e+06	3.2	2.174091e+08	林地
4	5	1.400016e+06	3.349699e+06	4.5	6.239034e+08	林地
5	6	1.365930e+06	3.345708e+06	4.7	3.300772e+08	林地
6	7	1.370875e+06	3.326612e+06	4.9	4.625801e+08	林地
7	8	1.349678e+06	3.332400e+06	3.6	8.437728e+07	林地
8	9	1.466448e+06	3.466357e+06	3.2	1.767367e+07	湖塘
9	10	1.472290e+06	3.450123e+06	3.9	1.103745e+08	湖塘
10	11	1.476637e+06	3.460697e+06	3.7	5.617859e+07	湖塘
11	12	1.480318e+06	3.426347e+06	3.3	4.810208e+07	湖塘
12	13	1.484063e+06	3.419651e+06	2.6	3.463853e+07	湖塘
13	14	1.427750e+06	3.409197e+06	5.0	1.784691e+09	湖塘
14	15	1.424141e+06	3.388490e+06	2.1	5.787761e+06	湿地
15	16	1.453224e+06	3.394227e+06	2.5	4.021417e+07	湖塘
16	17	1.432485e+06	3.406053e+06	2.5	1.125133e+07	林地
17	18	1.379826e+06	3.388059e+06	2.5	1.463333e+08	林地
18	19	1.413577e+06	3.360325e+06	2.3	7.313191e+07	林地
19	20	1.459986e+06	3.472646e+06	2.2	7.100339e+06	湖塘
20	21	1.463750e+06	3.457957e+06	2.2	5.606684e+06	湖塘
21	22	1.451404e+06	3.453714e+06	2.2	7.127965e+06	湖塘
22	23	1.480171e+06	3.448672e+06	2.2	6.885154e+06	湖塘
23	24	1.472386e+06	3.445123e+06	2.6	7.649559e+06	湖塘
24	25	1.490494e+06	3.431199e+06	2.3	2.169134e+06	湖塘
25	26	1.495038e+06	3.427587e+06	2.5	6.457394e+06	湖塘
26	27	1.472176e+06	3.421495e+06	2.3	1.503599e+07	湖塘
27	28	1.495150e+06	3.421917e+06	2.8	1.487946e+07	湖塘
28	29	1.487645e+06	3.410985e+06	2.4	1.182396e+07	湖塘
29	30	1.481912e+06	3.410641e+06	2.4	9.898603e+06	湖塘
30	31	1.476249e+06	3.409886e+06	2.5	2.082133e+07	湖塘
31	32	1.464692e+06	3.408728e+06	2.2	7.888008e+06	湖塘
32	33	1.481792e+06	3.402403e+06	2.7	3.827214e+06	湖塘
33	34	1.478802e+06	3.400051e+06	2.2	2.900081e+06	湖塘
34	35	1.476388e+06	3.393173e+06	2.5	5.690284e+06	湖塘
35	36	1.460640e+06	3.390252e+06	2.4	1.103299e+07	湖塘
36	37	1.372427e+06	3.373219e+06	2.4	3.545245e+06	湖塘
37	38	1.519278e+06	3.363338e+06	2.0	3.634200e+06	林地
38	39	1.514710e+06	3.360375e+06	2.0	2.517300e+06	湿地
39	40	1.497071e+06	3.333539e+06	2.0	9.889875e+06	林地
40	41	1.507791e+06	3.353840e+06	2.1	1.341630e+07	湿地
41	42	1.503414e+06	3.338381e+06	2.0	4.480650e+06	湿地
42	43	1.499191e+06	3.330960e+06	2.1	1.650420e+07	湿地

Use graph distance

In [5]:

```
def calculate_pij(G):
    return nx.adjacency_matrix(G).todense()

def calculate_distance(G):
    distance = dict(nx.all_pairs_dijkstra_path_length(G))
    distance  = pd.DataFrame.from_dict(distance)
    cols = list(distance.columns)
    distance = distance.loc[cols, cols].fillna(0)
```

```python
    one_div_one_plus_dk = 1 / (distance + 1)
    return one_div_one_plus_dk

def calculate_IIPC(G):
    pij = calculate_pij(G)
    one_div_one_plus_dk = calculate_distance(G)
    A = pij*one_div_one_plus_dk
    IIPC = np.dot(E_list.reshape(1, -1), np.dot(A, E_list.reshape(-1, 1)))
    return IIPC
```

In [6]:

```python
def calculate_ITSI(G, i, j):
    G1 = G.copy()
    G1.add_edges_from([(i, j)])
    return calculate_IIPC(G1) / calculate_IIPC(G) - 1
```

In [7]:

```python
df_rst = pd.DataFrame([], index=node_list, columns=node_list)

for i in node_list:
    for j in node_list:
        if i != j:
            val = calculate_ITSI(G, i, j)[0][0]
            df_rst.loc[i, j] = val

df_rst
```

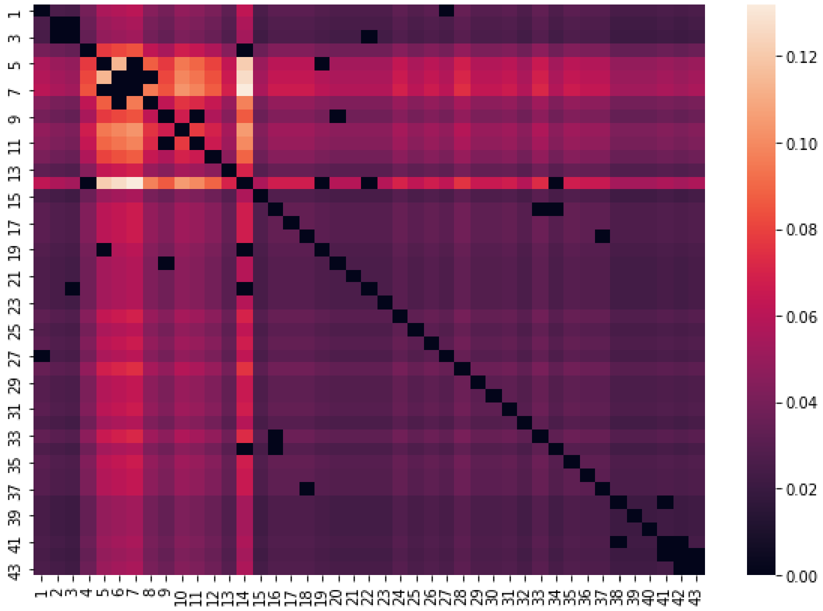
Out[7]:

	1	2	3	4	5	6	7	8	9	10	...	34	35	36
1	NaN	0.026042	0.024802	0.039683	0.055804	0.058284	0.060765	0.044643	0.039683	0.048364	...	0.027282	0.031002	0.029762
2	0.026042	NaN	0.0	0.036232	0.050952	0.053216	0.055481	0.040761	0.036232	0.044158	...	0.02491	0.028306	0.027174
3	0.024802	0.0	NaN	0.034507	0.048525	0.050682	0.052839	0.03882	0.034507	0.042055	...	0.023724	0.026959	0.02588
4	0.039683	0.036232	0.034507	NaN	0.077641	0.081091	0.084542	0.062112	0.055211	0.067289	...	0.037958	0.043134	0.041408
5	0.055804	0.050952	0.048525	0.077641	NaN	0.114035	0.0	0.087346	0.077641	0.094624	...	0.053378	0.060657	0.05823
6	0.058284	0.053216	0.050682	0.081091	0.114035	NaN	0.0	0.0	0.081091	0.09883	...	0.05575	0.063353	0.060818
7	0.060765	0.055481	0.052839	0.084542	0.0	0.0	NaN	0.09511	0.084542	0.103036	...	0.058123	0.066048	0.063406
8	0.044643	0.040761	0.03882	0.062112	0.087346	0.0	0.09511	NaN	0.062112	0.0757	...	0.042702	0.048525	0.046584
9	0.039683	0.036232	0.034507	0.055211	0.077641	0.081091	0.084542	0.062112	NaN	0.067289	...	0.037958	0.043134	0.041408
10	0.048364	0.044158	0.042055	0.067289	0.094624	0.09883	0.103036	0.0757	0.067289	NaN	...	0.046261	0.052569	0.050466
11	0.045883	0.041894	0.039899	0.063838	0.089772	0.093762	0.097752	0.071818	0.0	0.077802	...	0.043888	0.049873	0.047878
12	0.040923	0.037365	0.035585	0.056936	0.080067	0.083625	0.087184	0.064053	0.056936	0.069391	...	0.039144	0.044482	0.042702
13	0.032242	0.029439	0.028037	0.044859	0.063083	0.065887	0.06869	0.050466	0.044859	0.054672	...	0.030841	0.035046	0.033644
14	0.062005	0.056613	0.053917	0.0	0.121313	0.126705	0.132097	0.097051	0.086267	0.105138	...	0.0	0.067396	0.0647
15	0.026042	0.023777	0.022645	0.036232	0.050952	0.053216	0.055481	0.040761	0.036232	0.044158	...	0.02491	0.028306	0.027174
16	0.031002	0.028306	0.026959	0.043134	0.060657	0.063353	0.066048	0.048525	0.043134	0.052569	...	0.0	0.033698	0.03235
17	0.031002	0.028306	0.026959	0.043134	0.060657	0.063353	0.066048	0.048525	0.043134	0.052569	...	0.029654	0.033698	0.03235
18	0.031002	0.028306	0.026959	0.043134	0.060657	0.063353	0.066048	0.048525	0.043134	0.052569	...	0.029654	0.033698	0.03235
19	0.028522	0.026042	0.024802	0.039683	0.0	0.058284	0.060765	0.044643	0.039683	0.048364	...	0.027282	0.031002	0.029762
20	0.027282	0.02491	0.023724	0.037958	0.053378	0.05575	0.058123	0.042702	0.0	0.046261	...	0.026096	0.029654	0.028468
21	0.027282	0.02491	0.023724	0.037958	0.053378	0.05575	0.058123	0.042702	0.037958	0.046261	...	0.026096	0.029654	0.028468
22	0.027282	0.02491	0.0	0.037958	0.053378	0.05575	0.058123	0.042702	0.037958	0.046261	...	0.026096	0.029654	0.028468
23	0.027282	0.02491	0.023724	0.037958	0.053378	0.05575	0.058123	0.042702	0.037958	0.046261	...	0.026096	0.029654	0.028468
24	0.032242	0.029439	0.028037	0.044859	0.063083	0.065887	0.06869	0.050466	0.044859	0.054672	...	0.030841	0.035046	0.033644
25	0.028522	0.026042	0.024802	0.039683	0.055804	0.058284	0.060765	0.044643	0.039683	0.048364	...	0.027282	0.031002	0.029762
26	0.031002	0.028306	0.026959	0.043134	0.060657	0.063353	0.066048	0.048525	0.043134	0.052569	...	0.029654	0.033698	0.03235
27	0.0	0.026042	0.024802	0.039683	0.055804	0.058284	0.060765	0.044643	0.039683	0.048364	...	0.027282	0.031002	0.029762
28	0.034723	0.031703	0.030194	0.04831	0.067936	0.070955	0.073974	0.054348	0.04831	0.058877	...	0.033213	0.037742	0.036232
29	0.029762	0.027174	0.02588	0.041408	0.05823	0.060818	0.063406	0.046584	0.041408	0.050466	...	0.028468	0.03235	0.031056
30	0.029762	0.027174	0.02588	0.041408	0.05823	0.060818	0.063406	0.046584	0.041408	0.050466	...	0.028468	0.03235	0.031056
31	0.031002	0.028306	0.026959	0.043134	0.060657	0.063353	0.066048	0.048525	0.043134	0.052569	...	0.029654	0.033698	0.03235
32	0.027282	0.02491	0.023724	0.037958	0.053378	0.05575	0.058123	0.042702	0.037958	0.046261	...	0.026096	0.029654	0.028468
33	0.033483	0.030571	0.029115	0.046584	0.065509	0.068421	0.071332	0.052407	0.046584	0.056775	...	0.032027	0.036394	0.034938
34	0.027282	0.02491	0.023724	0.037958	0.053378	0.05575	0.058123	0.042702	0.037958	0.046261	...	NaN	0.029654	0.028468
35	0.031002	0.028306	0.026959	0.043134	0.060657	0.063353	0.066048	0.048525	0.043134	0.052569	...	0.029654	NaN	0.03235
36	0.029762	0.027174	0.02588	0.041408	0.05823	0.060818	0.063406	0.046584	0.041408	0.050466	...	0.028468	0.03235	NaN
37	0.029762	0.027174	0.02588	0.041408	0.05823	0.060818	0.063406	0.046584	0.041408	0.050466	...	0.028468	0.03235	0.031056
38	0.024802	0.022645	0.021567	0.034507	0.048525	0.050682	0.052839	0.03882	0.034507	0.042055	...	0.023724	0.026959	0.02588
39	0.024802	0.022645	0.021567	0.034507	0.048525	0.050682	0.052839	0.03882	0.034507	0.042055	...	0.023724	0.026959	0.02588
40	0.024802	0.022645	0.021567	0.034507	0.048525	0.050682	0.052839	0.03882	0.034507	0.042055	...	0.023724	0.026959	0.02588
41	0.026042	0.023777	0.022645	0.036232	0.050952	0.053216	0.055481	0.040761	0.036232	0.044158	...	0.02491	0.028306	0.027174
42	0.024802	0.022645	0.021567	0.034507	0.048525	0.050682	0.052839	0.03882	0.034507	0.042055	...	0.023724	0.026959	0.02588
43	0.026042	0.023777	0.022645	0.036232	0.050952	0.053216	0.055481	0.040761	0.036232	0.044158	...	0.02491	0.028306	0.027174

43 rows × 43 columns

In [8]:

```
import plotly.express as px
import seaborn as sns

fig, ax = plt.subplots(1, 1, figsize=(10, 7))
ax = sns.heatmap(df_rst.fillna(0), linewidth=0.001)
```



Use Euclid distance

In [9]:
```
def calculate_pij(G):
    return nx.adjacency_matrix(G).todense()

def calculate_distance(nodes):
    m = nodes.shape[0]
    distance = np.zeros((m, m))
    for i in range(m):
        for j in range(i+1, m):
            cord1, cord2 = nodes.loc[i, ["X坐标", "Y坐标"]].values, nodes.loc[j, ["X坐标", "Y坐标"]].values
            dist = ((cord2 - cord1)**2).sum() ** 0.5
            distance[i, j] = dist
            distance[j, i] = dist
    one_div_one_plus_dk = 1 / (distance + 1)
    return one_div_one_plus_dk

def calculate_IIPC(G, one_div_one_plus_dk):
    pij = calculate_pij(G)
    A = pij*one_div_one_plus_dk
    IIPC = np.dot(E_list.reshape(1, -1), np.dot(A, E_list.reshape(-1, 1)))
    return IIPC[0, 0]
```

In [10]:
```
def calculate_ITSI(G, i, j, one_div_one_plus_dk):
    G1 = G.copy()
    G1.add_edges_from([(i, j)])
    return calculate_IIPC(G1, one_div_one_plus_dk) / calculate_IIPC(G, one_div_one_plus_dk) - 1
```

In [11]:
```
df_rst = pd.DataFrame([], index=node_list, columns=node_list)
one_div_one_plus_dk = calculate_distance(nodes)

for i in node_list:
    for j in node_list:
        if i != j:
            val = calculate_ITSI(G, i, j, one_div_one_plus_dk)
            df_rst.loc[i, j] = val

df_rst
```

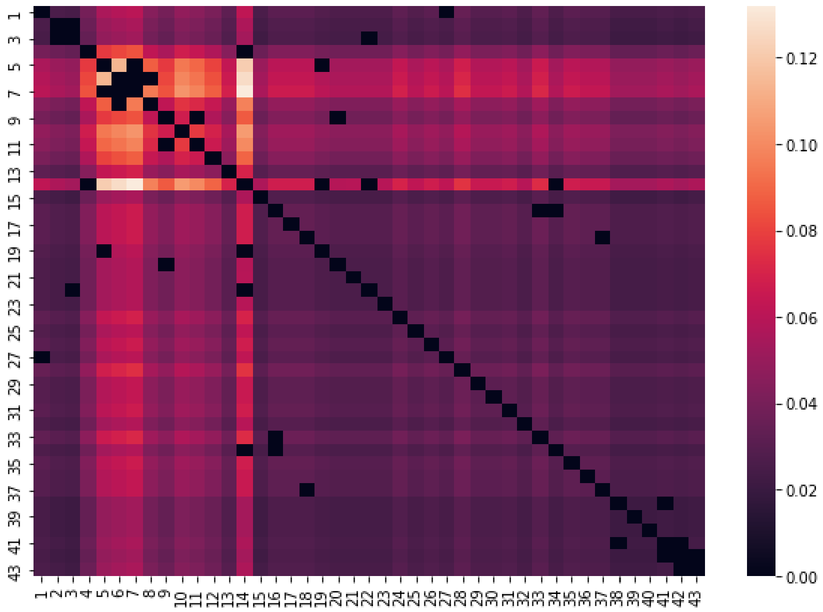
Out[11]:

	1	2	3	4	5	6	7	8	9	10	...	34	35	36
1	NaN	0.026052	0.024813	0.039693	0.055814	0.058293	0.060773	0.044651	0.0397	0.048384	...	0.027306	0.031021	0.029779
2	0.026052	NaN	0.0	0.036231	0.050946	0.05321	0.055473	0.040757	0.036237	0.044164	...	0.024924	0.028315	0.027181
3	0.024813	0.0	NaN	0.034508	0.048524	0.050679	0.052835	0.038819	0.034514	0.042064	...	0.023739	0.026969	0.025889
4	0.039693	0.036231	0.034508	NaN	0.077622	0.08107	0.084519	0.062098	0.055211	0.067289	...	0.037975	0.043141	0.041414
5	0.055814	0.050946	0.048524	0.077622	NaN	0.113997	0.0	0.087318	0.077635	0.094618	...	0.053398	0.060663	0.058234
6	0.058293	0.05321	0.050679	0.08107	0.113997	NaN	0.0	0.0	0.081084	0.098822	...	0.05577	0.063358	0.060821
7	0.060773	0.055473	0.052835	0.084519	0.0	0.0	NaN	0.095077	0.084533	0.103026	...	0.058142	0.066053	0.063408
8	0.044651	0.040757	0.038819	0.062098	0.087318	0.0	0.095077	NaN	0.062109	0.075695	...	0.042719	0.048531	0.046587
9	0.0397	0.036237	0.034514	0.055211	0.077635	0.081084	0.084533	0.062109	NaN	0.067301	...	0.037981	0.043149	0.041421
10	0.048384	0.044164	0.042064	0.067289	0.094618	0.098822	0.103026	0.075695	0.067301	NaN	...	0.04629	0.052588	0.050482
11	0.0459	0.041897	0.039905	0.063835	0.089761	0.093749	0.097736	0.071809	0.0	0.077813	...	0.043914	0.049888	0.047891
12	0.040945	0.037374	0.035597	0.056943	0.08007	0.083628	0.087185	0.064057	0.056953	0.069412	...	0.039173	0.044502	0.04272
13	0.032266	0.029452	0.028052	0.044873	0.063098	0.065902	0.068705	0.050479	0.044881	0.054699	...	0.030869	0.035069	0.033665
14	0.062019	0.05661	0.053918	0.0	0.121282	0.12667	0.132058	0.097026	0.086266	0.105138	...	0.0	0.067407	0.064708
15	0.026055	0.023782	0.022651	0.036235	0.050951	0.053215	0.055479	0.040761	0.036241	0.044169	...	0.024927	0.028318	0.027184
16	0.031018	0.028313	0.026967	0.043138	0.060659	0.063354	0.066048	0.048527	0.043146	0.052584	...	0.0	0.033713	0.032363
17	0.03102	0.028314	0.026968	0.04314	0.060661	0.063356	0.066051	0.048529	0.043148	0.052586	...	0.029677	0.033715	0.032365
18	0.031013	0.028308	0.026962	0.04313	0.060647	0.063342	0.066036	0.048518	0.043138	0.052574	...	0.02967	0.033707	0.032357
19	0.028533	0.026045	0.024806	0.039682	0.0	0.058277	0.060756	0.044639	0.039688	0.048371	...	0.027298	0.031012	0.02977
20	0.027296	0.024916	0.023731	0.037962	0.05338	0.055752	0.058123	0.042704	0.0	0.046274	...	0.026115	0.029668	0.02848
21	0.027301	0.02492	0.023735	0.037968	0.053388	0.05576	0.058132	0.042711	0.037974	0.046282	...	0.026119	0.029672	0.028484
22	0.027297	0.024916	0.0	0.037962	0.05338	0.055752	0.058123	0.042705	0.037969	0.046275	...	0.026115	0.029668	0.02848
23	0.027302	0.024921	0.023736	0.03797	0.053391	0.055763	0.058135	0.042713	0.037976	0.046284	...	0.02612	0.029674	0.028486
24	0.032264	0.02945	0.02805	0.044871	0.063095	0.065898	0.068701	0.050476	0.044879	0.054696	...	0.030868	0.035067	0.033663
25	0.028543	0.026054	0.024815	0.039695	0.055817	0.058297	0.060777	0.044654	0.039702	0.048387	...	0.027307	0.031023	0.029781
26	0.031023	0.028317	0.026971	0.043145	0.060668	0.063363	0.066058	0.048534	0.043152	0.052592	...	0.02968	0.033718	0.032368
27	0.0	0.026055	0.024816	0.039697	0.05582	0.0583	0.060779	0.044656	0.039704	0.048389	...	0.027309	0.031024	0.029782
28	0.034743	0.031713	0.030205	0.048318	0.067942	0.07096	0.073978	0.054354	0.048326	0.058898	...	0.033239	0.037761	0.036249
29	0.029785	0.027187	0.025894	0.041422	0.058246	0.060834	0.063421	0.046597	0.04143	0.050493	...	0.028496	0.032372	0.031076
30	0.029787	0.02719	0.025897	0.041426	0.058251	0.060839	0.063427	0.046601	0.041433	0.050497	...	0.028498	0.032375	0.031079
31	0.031026	0.02832	0.026974	0.043149	0.060674	0.063369	0.066065	0.048539	0.043156	0.052597	...	0.029683	0.033722	0.032372
32	0.027302	0.024921	0.023736	0.037969	0.05339	0.055762	0.058134	0.042713	0.037976	0.046283	...	0.02612	0.029674	0.028486
33	0.033506	0.030584	0.029129	0.046597	0.065523	0.068434	0.071344	0.052418	0.046605	0.056801	...	0.032056	0.036417	0.034959
34	0.027306	0.024924	0.023739	0.037975	0.053398	0.05577	0.058142	0.042719	0.037981	0.04629	...	NaN	0.029678	0.02849
35	0.031021	0.028315	0.026969	0.043141	0.060663	0.063358	0.066053	0.048531	0.043149	0.052588	...	0.029678	NaN	0.032366
36	0.029779	0.027181	0.025889	0.041414	0.058234	0.060821	0.063408	0.046587	0.041421	0.050482	...	0.02849	0.032366	NaN
37	0.029772	0.027176	0.025883	0.041405	0.058221	0.060808	0.063394	0.046577	0.041412	0.050471	...	0.028483	0.032359	0.031063
38	0.024813	0.022649	0.021572	0.034509	0.048524	0.05068	0.052835	0.038819	0.034514	0.042065	...	0.023739	0.026969	0.025889
39	0.024814	0.02265	0.021573	0.03451	0.048525	0.050681	0.052837	0.038821	0.034515	0.042066	...	0.02374	0.02697	0.02589
40	0.024815	0.022651	0.021574	0.034511	0.048527	0.050683	0.052839	0.038822	0.034517	0.042068	...	0.023741	0.026971	0.025891
41	0.026053	0.023781	0.02265	0.036233	0.050949	0.053212	0.055475	0.040759	0.036239	0.044167	...	0.024925	0.028317	0.027183
42	0.024813	0.022649	0.021572	0.034508	0.048524	0.05068	0.052835	0.038819	0.034514	0.042065	...	0.023739	0.026969	0.025889
43	0.026054	0.023782	0.022651	0.036235	0.050951	0.053215	0.055478	0.040761	0.036241	0.044169	...	0.024927	0.028318	0.027184

43 rows × 43 columns

In [12]:

```python
import plotly.express as px
import seaborn as sns

fig, ax = plt.subplots(1, 1, figsize=(10, 7))
ax = sns.heatmap(df_rst.fillna(0), linewidth=0.001)
```



Topology

In [13]:

```python
def plot_network_graph(G):

    import plotly.graph_objects as go

    edge_x = []
    edge_y = []
    for edge in G.edges():

        x0, y0 = nodes.loc[nodes["斑块编号"] == edge[0], ["X坐标", "Y坐标"]].values[0]
        x1, y1 = nodes.loc[nodes["斑块编号"] == edge[1], ["X坐标", "Y坐标"]].values[0]

        edge_x.append(x0)
        edge_x.append(x1)
        edge_x.append(None)
        edge_y.append(y0)
        edge_y.append(y1)
        edge_y.append(None)

    node_x = []
    node_y = []
    node_t = []
    for node in G.nodes():
        x, y = nodes.loc[nodes["斑块编号"] == node, ["X坐标", "Y坐标"]].values[0]
        node_x.append(x)
        node_y.append(y)
        node_t.append(node)

    edge_trace = go.Scatter(
        x=edge_x, y=edge_y,
        line=dict(width=0.5, color='#888'),
        hoverinfo='none',
        mode='lines')


    node_trace = go.Scatter(
        x=node_x, y=node_y,
        text=node_t,
        mode='markers+text',
        hoverinfo='text',
        textposition="middle center",
        textfont=dict(
            size=10,
            color="White"
        ),
        marker=dict(
            showscale=False,
            colorscale='Viridis',
            reversescale=True,
            color="#5D69B1",
            size=20,
            # colorbar=dict(
            #     thickness=15,
            #     title='Node Connections',
            #     xanchor='left',
            #     titleside='right'
            # ),
            line_width=2))

    fig = go.Figure(data=[edge_trace, node_trace],
                    layout=go.Layout(
                        title='<br>Lanscape Network Visualization',
                        titlefont_size=25,
```

```python
                                showlegend=False,
                                width=1000,
                                height=600,
                                hovermode='closest',
                                margin=dict(b=20,l=5,r=5,t=40),
                                xaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
                                yaxis=dict(showgrid=False, zeroline=False, showticklabels=False))
                            )
    return fig
```

In [14]:

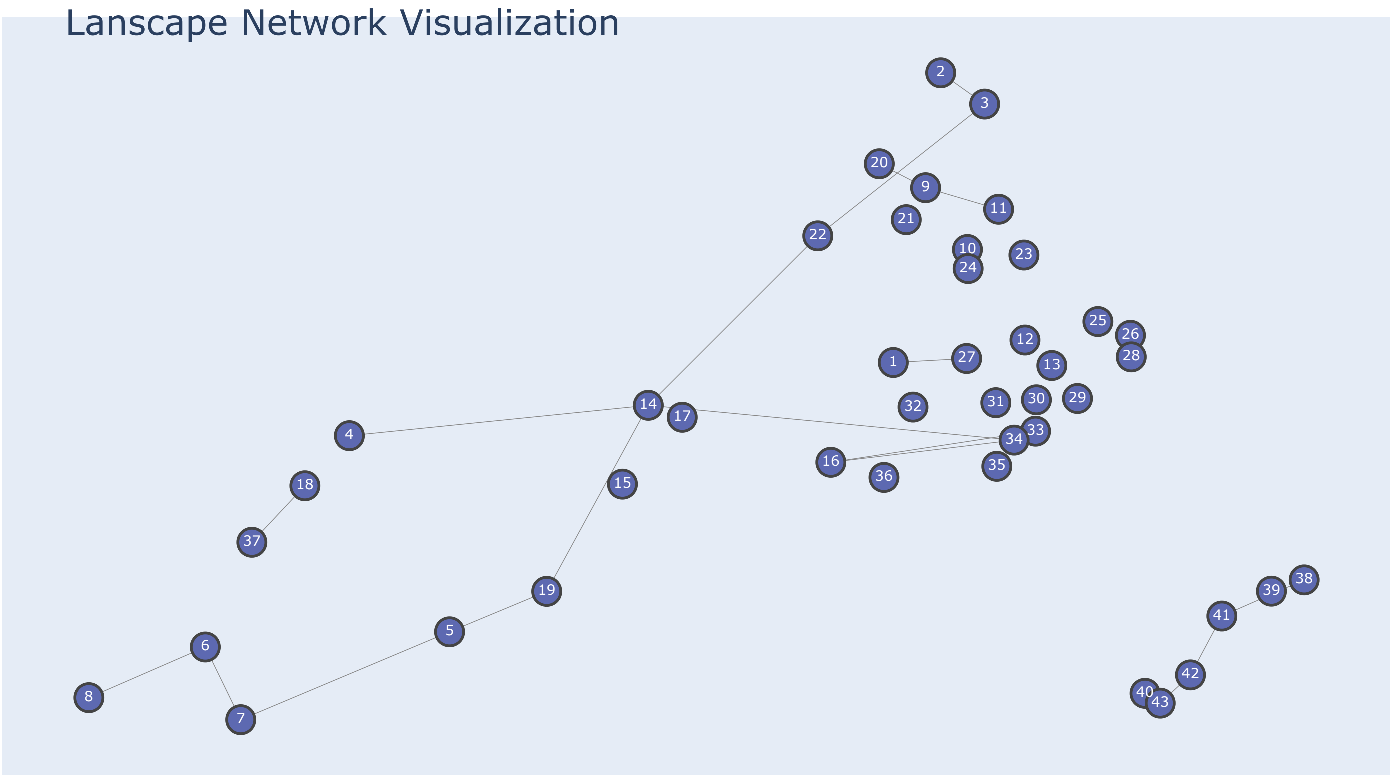
```python
fig = plot_network_graph(G)
```

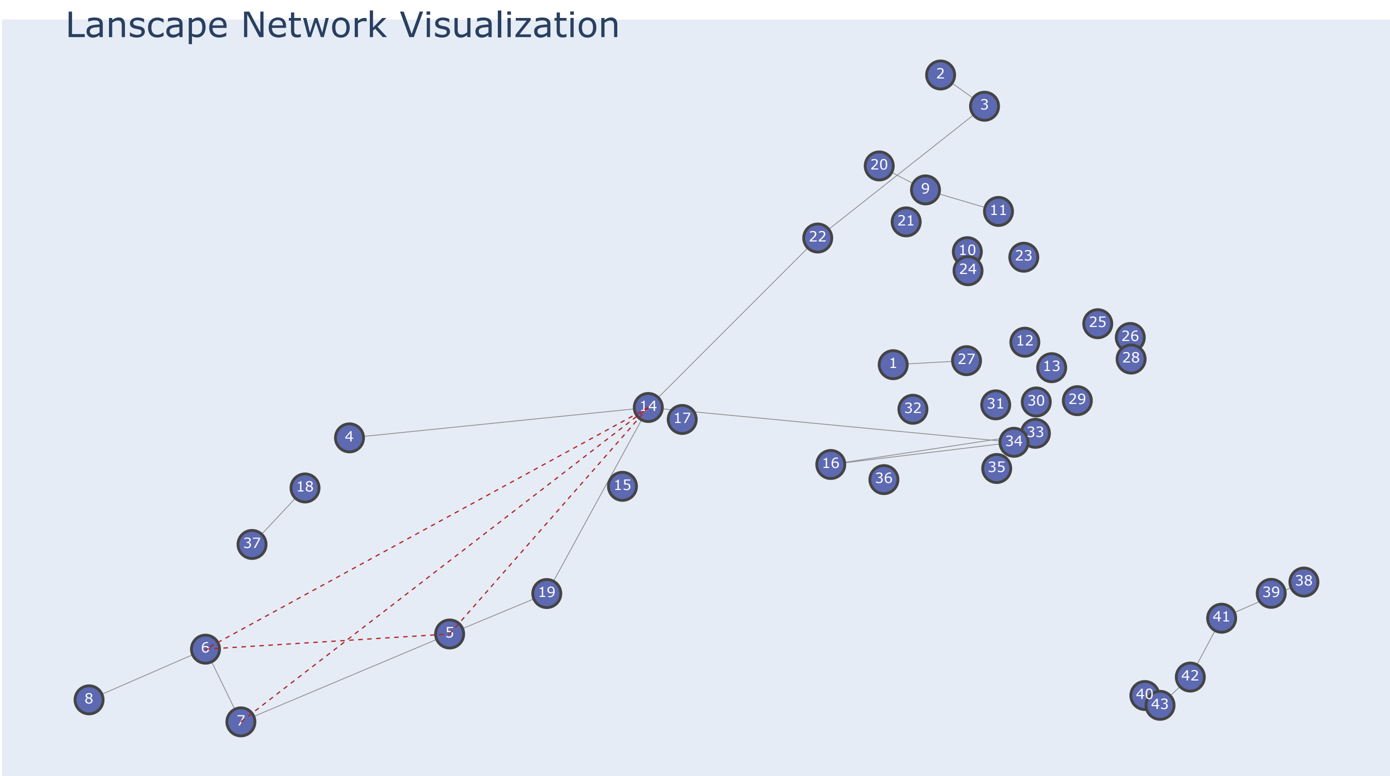
In [15]:

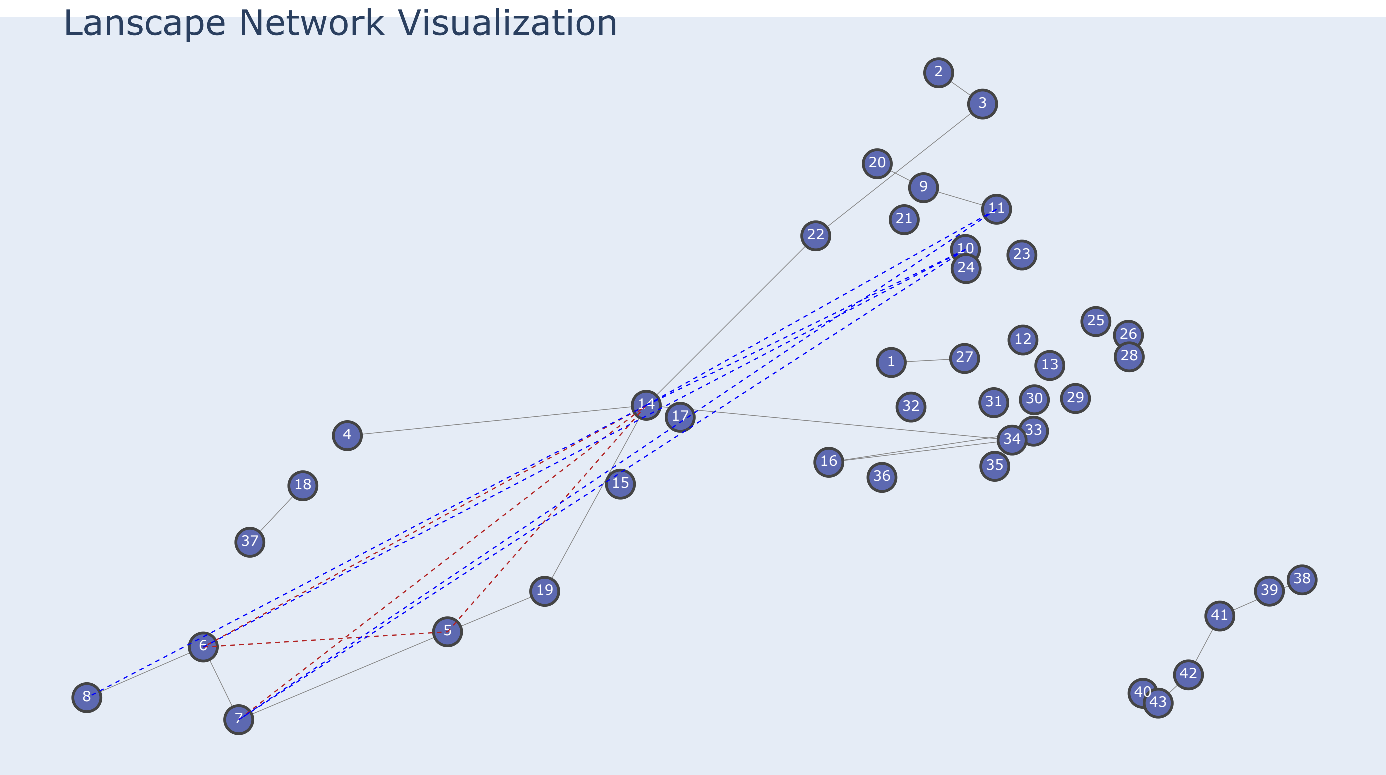
```python
def convert_matrix_to_rank(df_rst, head=0, end=20):
    df_rk = []
    for i in df_rst.index:
        for j in df_rst.columns:
            if j > i:
                df_rk.append([i, j, df_rst.loc[i, j]])

    df_rk = pd.DataFrame(df_rk, columns=["start", "end", "value"])
    df_rk.sort_values("value", ascending=False, inplace=True)
    df_rk = df_rk.reset_index(drop=True).reset_index(drop=False).rename({"index":"rank"}, axis=1)
    return df_rk.head(end).tail(end-head)

def gen_new_graph(df_rk):

    edges_list_new = []
    for idx, i in df_rk.iterrows():
        edges_list_new.append([int(i["start"]), int(i["end"])])

    G_new = nx.Graph()
    G_new.add_edges_from(edges_list_new)
    return G_new


def add_graph(fig, G_new, color='firebrick'):
    import plotly.graph_objects as go

    edge_x = []
    edge_y = []
    for edge in G_new.edges():

        x0, y0 = nodes.loc[nodes["斑块编号"] == edge[0], ["X坐标", "Y坐标"]].values[0]
        x1, y1 = nodes.loc[nodes["斑块编号"] == edge[1], ["X坐标", "Y坐标"]].values[0]

        edge_x.append(x0)
        edge_x.append(x1)
        edge_x.append(None)
        edge_y.append(y0)
        edge_y.append(y1)
        edge_y.append(None)

    edge_trace = go.Scatter(
        x=edge_x, y=edge_y,
        line=dict(width=0.8, color=color, dash='dot'),
        hoverinfo='none',
        mode='lines')

    fig.add_trace(
        edge_trace
    )
    return fig
```

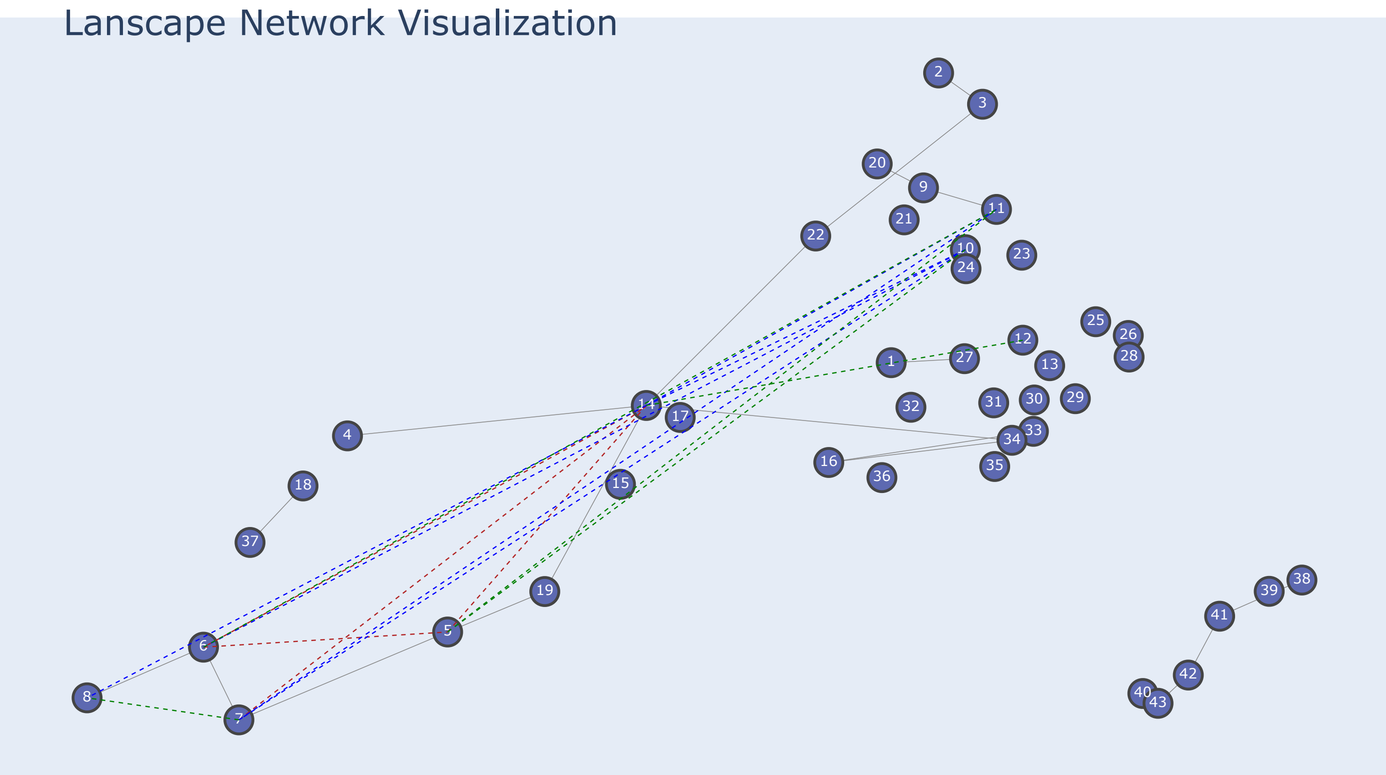
In [26]:

```python
fig = plot_network_graph(G)
fig.show(renderer="svg")
fig.write_image("./../../results/2.4.3/landscape_origin.png")

df_rk = convert_matrix_to_rank(df_rst, head=0, end=4)
G1 = gen_new_graph(df_rk)
fig = add_graph(fig, G1, color='firebrick')
fig.show(renderer="svg")
fig.write_image("./../../results/2.4.3/landscape_origin_00_04.png")

df_rk = convert_matrix_to_rank(df_rst, head=4, end=10)
G2 = gen_new_graph(df_rk)
fig = add_graph(fig, G2, color='blue')
fig.show(renderer="svg")
fig.write_image("./../../results/2.4.3/landscape_origin_04_10.png")


df_rk = convert_matrix_to_rank(df_rst, head=10, end=15)
G3 = gen_new_graph(df_rk)
fig = add_graph(fig, G3, color='green')
fig.show(renderer="svg")
fig.write_image("./../../results/2.4.3/landscape_origin_10_15.png")

# df_rk = convert_matrix_to_rank(df_rst, head=10, end=20)
```


Lanscape Network Visualization
2
3
20
9
11
21
22
10
23
24
25
26
12
1
27
28
13
32
31
30
29
14
17
33
34
4
16
36
35
15
18
37
38
39
41
19
5
6
42
40
43
8
7


Lanscape Network Visualization
2
3
20
9
11
21
22
10
23
24
25
26
12
1
27
28
13
32
31
30
29
14
17
33
34
4
16
36
35
15
18
37
38
39
41
19
5
6
42
40
43
8
7





Save File

In [23]:
```
df_rk = convert_matrix_to_rank(df_rst, head=0, end=df_rst.shape[0]*df_rst.shape[0])
df_rk.to_csv("./../../results/2.4.3/result_rank.csv", index=False)
```