Supporting Information for

Identification of the Pollution Mechanisms and Remediation Strategies for Abandoned Wells in the Karst Areas of Northern China

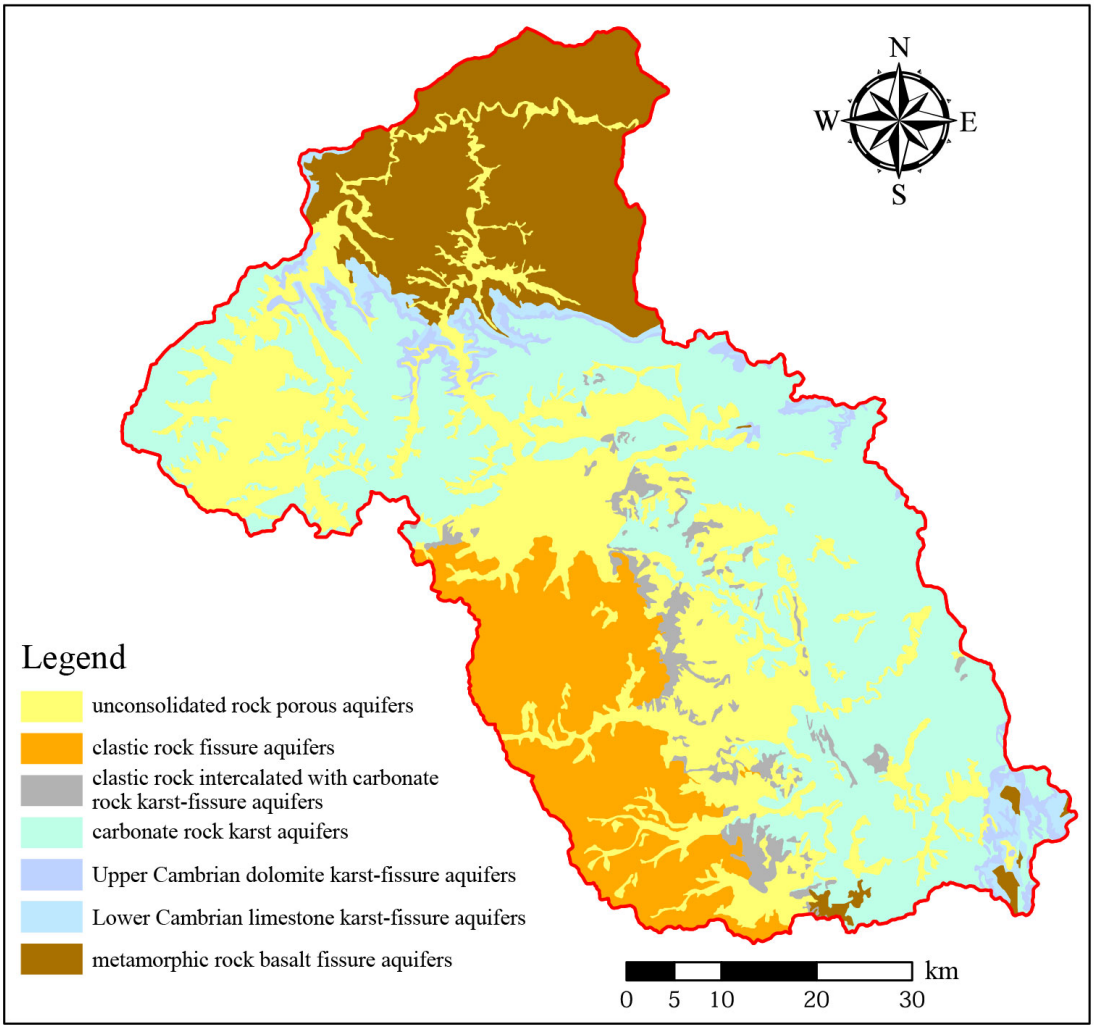

Figure S1. Hydrogeological map of Yangquan City

Table S1. Abandoned well pollution risk comprehensive score

No.	Well no.	Well depth	Comprehensive score
1	YX01	787	4.2514
2	YX02	153	3.7496
3	YX03	738.5	3.5091
4	YX04	86.5	3.4617
5	YX05	400	2.8627
6	YX06	146	2.6652
7	YX07	152.6	3.9155
8	YX08	138	3.9155

9	YX09	98.6	2.7872
10	YX10	126.66	2.9531
11	YX11	66.64	2.7872
12	YX12	157.42	2.9531
13	YX13	150	2.9531
14	YX14	77.13	2.9531
15	YX15	745.5	4.4715
16	YX16	531.6	3.0828
17	YX17	563	4.0452
18	YX18	596.23	4.0452
19	YX19	591	4.0452
20	YX20	200	3.8793
21	YX21	685	3.5091
22	YX22	537.6	3.0828
23	YX23	655.69	4.1836
24	YX24	800	4.4715
25	YX25	700	3.5091
26	YX26	150	2.7872
27	YX27	300	2.6968
28	YX28	100	0.3862
29	YX29	100	0.5741
30	YX30	80	0.3862
31	YX31	80	0.3862
32	YX32	80	0.3862
33	YX33	120	0.5741
34	YX34	100	0.3862
35	YX35	100	0.3862
36	YX36	100	0.3862
37	YX37	100	0.5741
38	YX38	170	0.3862
39	YX39	90	0.5741
40	YX40	636.76	3.289
41	YX41	80	2.4993
42	YX42	80	2.4993
43	YX43	80	2.4993
44	YX44	150	0.3862
45	YX45	100	0.3862
46	YX46	65	0.3862
47	YX47	60	0.3862
48	YX48	100	0.3862
49	YX49	90	0.3862
50	YX50	130	0.5741
51	YX51	100	0.5741

52	YX52	86	0.5741
53	YX53	45	0.5741
54	YX54	45	0.3862
55	YX55	45	0.3862
56	YX56	40	0.3862
57	YX57	50	0.3862
58	YX58	24	0.5741
59	YX59	45	0.5741
60	YX60	30	0.3862
61	YX61	40	0.3862
62	YX62	30	0.454
63	YX63	30	0.454
64	YX64	50	0.5741
65	YX65	50	0.5741
66	YX66	35	0.5741
67	YX67	30	0.5741
68	YX68	20	0.5741
69	YX69	42	0.5741
70	YX70	43	0.5741
71	YX71	45	0.5741
72	YX72	50	0.5741
73	YX73	225	2.629
74	YX74	760	3.5091
75	YX75	700	3.5091
76	YX76	80	0.5741
77	YX77	120	0.5741
78	YX78	610	3.2212
79	YX79	502.6	3.1506
80	YX80	600	4.4715
81	PD01	336.11	1.6503
82	PD02	540	1.6503
83	PD03	450	1.6503
84	PD04	495.06	2.7949
85	PD05	48.96	2.5671
86	PD06	166.52	2.5671
87	PD07	851.83	3.289
88	PD08	156.99	1.5206
89	PD09	150.11	1.5206
90	PD10	275	1.6503
91	PD11	355.15	1.6503
92	PD12	215	1.6503
93	PD13	306	1.6503
94	PD14	568.5	1.6503

95	PD15	251	1.6503
96	PD16	85.16	1.5206
97	PD17	150.14	1.5206
98	PD18	321	1.6503
99	PD19	230	1.6503
100	PD20	350	1.6503
101	PD21	280	1.6503
102	PD22	250.01	1.6503
103	PD23	452	1.6503
104	PD24	270.52	1.6503
105	KQ01	500	2.7949
106	KQ02	490.15	3.7573
107	KQ03	480.5	2.7949
108	KQ04	480.17	2.7949
109	KQ05	440.36	2.7949
110	KQ06	450.2	2.7949
111	KQ07	450.05	2.7949
112	KQ08	450.71	2.7949
113	KQ09	480.29	2.7949
114	KQ10	490.38	2.7949
115	KQ11	480.11	2.7949
116	KQ12	490.59	2.7949
117	KQ13	490.04	2.7949
118	KQ14	490.59	2.7949
119	KQ15	486.82	2.7949
120	KQ16	760	4.1836
121	KQ17	756	4.1836
122	KQ18	260	2.6968
123	CQ01	200	3.6592
124	CQ02	200	2.629
125	CQ03	574.8	2.7949
126	CQ04	397	2.7949
127	CQ05	500	2.7949
128	CQ06	489	2.7949
129	CQ07	506	2.7949
130	CQ08	420	3.0828
131	CQ09	580	3.0828
132	CQ10	400	4.0452
133	JQ01	500	3.8251
134	JQ02	650.98	4.1836
135	JQ03	500	2.8627
136	JQ04	800	3.5091
137	JQ05	40	0.6419

In Table S1, it is observed that certain abandoned wells exhibit identical comprehensive scores. This phenomenon can be attributed to the relative spatial clustering of these wells. Additionally, the environmental factors contributing to pollution risk, including the conditions of nearby pollution sources and the inherent characteristics of the abandoned wells, are homogenous.

Taking the criterion layer as an example, the Python code for calculating the pairwise comparison matrix weights is as follows:

```
import numpy as np
from numpy.linalg import eig
def ahp(matrix):
    # Step 1: Eigenvalue and Eigenvector Calculation
    eigenvalue, eigenvector = np.linalg.eig(matrix)
    max_eigenvalue = np.max(np.real(eigenvalue))
    # Step 2: Consistency Index (CI) Calculation
    n = len(matrix)
    CI = (max_eigenvalue - n) / (n - 1)
    # Step 3: Consistency Ratio (CR) Calculation
    # Random Index (RI) is assumed based on matrix size
    RI_values = {1: 0, 2: 0, 3: 0.58, 4: 0.89, 5: 1.12, 6: 1.26, 7: 1.36, 8: 1.41, 9: 1.46}
    RI = RI_values.get(n, 1.46)    # default to 1.46 for n > 9
    CR = CI / RI
    # Step 4: Normalized Eigenvector as Weights
    weight = np.real(eigenvector[:, np.argmax(np.real(eigenvalue))])
    weight_normalized = weight / np.sum(weight)
    return max_eigenvalue, CI, CR, weight_normalized
# Example usage
matrix = np.array([[1, 1/3,1/3,1/4,1/5],
                   [3,1,1,1/2,1/3],
                   [3,1,1,1/2,1/3],
                   [4,2,2,1,2/3],
```

```
                                [5,3,3,3/2,1]])
max_eigenvalue, CI, CR, weight_normalized = ahp(matrix)
print("Max Eigenvalue:", "{:.4f}".format(max_eigenvalue))
print("Consistency Index (CI):", "{:.4f}".format(CI))
print("Consistency Ratio (CR):", "{:.4f}".format(CR))
print("Normalized Weights:", ["{:.4f}".format(w) for w in weight_normalized])
```