```
import numpy as np

num_rows = 4
num_cols = 4
num_states = num_rows * num_cols  # 4x4 grid
num_actions = 4  # Up, Down, Left, Right
num_episodes = 500
alpha = 0.6  # Learning rate
gamma = 0.9  # Discount factor

Q = np.zeros((num_states, num_actions), dtype=float)

rewards = np.random.randint(1, 10, num_states)

# Define the goal state and obstacles
goal_state = num_states - 1
obstacles = [5, 7, 10]

# Update rewards for goal state and obstacles
rewards[goal_state] = 10
for obstacle in obstacles:
    rewards[obstacle] = -10

# List to store the path taken during each episode
episode_paths = []

def q_learning(num_states, num_actions, num_episodes, alpha, gamma):
    for episode in range(num_episodes):
        state = np.random.randint(0, num_states)
        episode_path = [state]  # Initialize path for this episode

        while True:
            action = np.argmax(Q[state, :]) if np.random.rand() < 0.9 else np.random.randint(0, num_actions)

            next_state = (state + action) % num_states

            reward = rewards[next_state]

            Q[state, action] = (1 - alpha) * Q[state, action] + alpha * (reward + gamma * np.max(Q[next_state, :]))

            state = next_state
            episode_path.append(state)  # Add the next state to the episode path

            if state == num_states - 1:
                episode_paths.append(episode_path)  # Store the episode path
                break

    return Q

learned_Q_values = q_learning(num_states, num_actions, num_episodes, alpha, gamma)

print("Learned Q-values:")
print(learned_Q_values)
```

```
Learned Q-values:
[[78.88996516 80.78980777 76.8897647  82.1       ]
 [80.78998346 76.88971639 82.1        83.1       ]
 [46.13399819 82.1        77.7816     62.53471715]
 [82.09995945 83.09994519 70.09872244 89.        ]
 [82.42707128 70.09264948 89.         67.39914786]
 [-6.         89.         56.61585753 51.59793908]
 [89.         67.39999998 86.         90.        ]
 [50.6130068  86.         54.          0.        ]
 [70.93463195 90.         42.6        76.8951732 ]
 [90.         71.         85.         85.        ]
 [70.86278347 85.         84.99942957 90.        ]
 [84.99999946 84.99989418 90.         76.70226534]
 [84.99999866 90.         76.6599301  79.17371568]
 [90.         76.711      79.20099524 78.89      ]
 [76.66936199 79.13216411 78.87556821 80.79      ]
 [16.9416      0.         47.934      76.88999471]]
```

```
# Extract the shortest path from the recorded episode paths
if episode_paths:
    shortest_path = min(episode_paths, key=len)
    print("Shortest path to goal state while learning:")
    print(shortest_path)

    # Display the states visited along the shortest path
    print("States visited along the shortest path:")
    for state in shortest_path:
        row = state // num_cols
        col = state % num_cols
        print(f"State: ({row}, {col})")
else:
    print("No paths recorded during learning.")
```

```
Shortest path to goal state while learning:
[15, 15]
States visited along the shortest path:
State: (3, 3)
State: (3, 3)
```