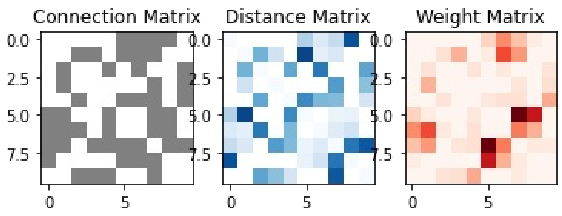
Exercise 15.1

In [6]:

```
import numpy as np
import random
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

N = 10
M = np.zeros((N,N))
k = 16


def simplify_sequence(sequence):
    result = []
    i = 0
    while i < len(sequence):
        if sequence[i] in sequence[i+1:]:          # if the current node is repeated
            i = sequence.index(sequence[i], i+1)   # find the next index of the repeated node
        else:
            result.append(sequence[i])            # if the current node is not repeated, append it to the simplii
            i += 1
    return result


for i in range(N):
    for j in range(i+1, N):
        if random.uniform(0,1) < 0.5:
            M[i,j] = 1
            M[j,i] = 1
        else:
            M[i,j] = 0
            M[j,i] = 0
plt.subplot(1,3,1)
colors = ['White','Black']
colormap = ListedColormap(colors)
plt.title("Connection Matrix")
plt.imshow(M,colormap)


D = np.zeros((N,N))
W = np.zeros((N,N))
for i in range(N):
    for j in range(i+1, N):
        if random.uniform(0,1) < 0.5:
            if M[i,j] == 1:
                r = random.uniform(1,100)
                D[i,j] = r
                D[j,i] = r
            else:
                D[i,j] = 100000000
                D[j,i] = 100000000
            W[i,j] = 1/D[i,j]
            W[j,i] = 1/D[j,i]
plt.subplot(1,3,2)
plt.title("Distance Matrix")
plt.imshow(D,vmin = 0,vmax = 100, cmap = 'Blues')
plt.subplot(1,3,3)
plt.title("Weight Matrix")
plt.imshow(D, cmap = 'Reds')


path = []
start_vertex = random.randint(0,N)
path.append(start_vertex)
for i in range (k-1):
    consecutive_vertex = path[i]
    consecutive_vertex_list = M[:, consecutive_vertex]
    index = np.where(consecutive_vertex_list == 1)
    path.append(round(random.choice(random.choice(index))))

distance = 0
for i in range(len(path)-1):
    distance += D[path[i], path[i+1]]


print('Path is ' + str(path))
print('Distance travelled = ' + str(distance))

paths = simplify_sequence(path)

simplified_distance = 0
for i in range(len(paths)-1):
    simplified_distance += D[paths[i], paths[i+1]]
```

```
print('Simplified Path is ' + str(paths))
print('Simplified Distance travelled = ' + str(simplified_distance))
```

```
Path is [3, 1, 7, 1, 7, 8, 5, 1, 3, 9, 4, 7, 3, 1, 7, 4]
Distance travelled = 493.4033227211329
Simplified Path is [3, 1, 7, 4]
Simplified Distance travelled = 78.08394945359345
```



Exercise 15.7

In [2]:

```
import numpy as np
from scipy.spatial import Delaunay
import matplotlib.pyplot as plt


def total_distance(sequence,D):
    return sum(D[sequence[i], sequence[i+1]] for i in range(len(sequence)-1))

#remove all the elements between repeated nodes
def simplify_sequence(sequence):
    result = []
    i = 0
    while i < len(sequence):
        if sequence[i] in sequence[i+1:]:           # if the current node is repeated
            i = sequence.index(sequence[i], i+1)   # find the next index of the repeated node
        else:
            result.append(sequence[i])             # if the current node is not repeated, append it to the simpli
            i += 1
    return result


def branch_decision(i,T,D,M,alpha,beta):
    possible_next = np.where(M[i] == 1)[0]                # can only move to the nodes that are connected to the c
    probability_next = np.zeros(len(possible_next))      # probability of moving to each of the possible next noc
    for j in range(len(possible_next)):
        probability_next[j] = T[i,possible_next[j]]**alpha * (1/D[i,possible_next[j]])**beta   # formula for the
    probability_next = probability_next / sum(probability_next)                              # normalize the pr
    return np.random.choice(possible_next, p=probability_next)
```

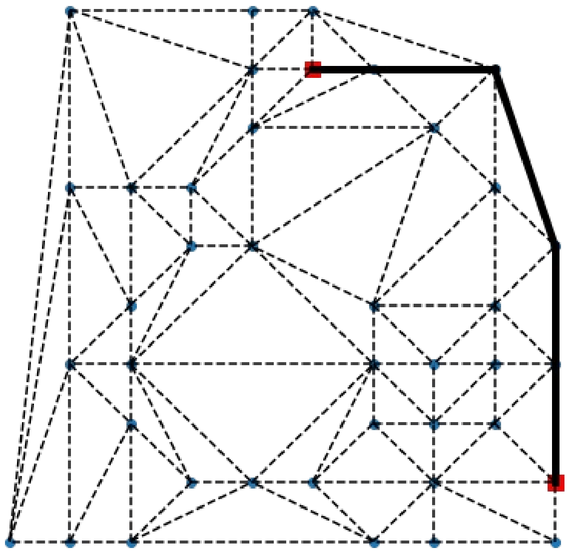
In [3]:

```
n = 20            # number of ants
alpha = 0.8       # alpha parameter
beta = 1         # beta parameter
rho = 0.5         # rho parameter
max_steps = 80   # maximum number of steps
number_of_iterations = 100 # number of iterations

N = 40  # number of nodes

# generate N points at unique random positions (x,y) with x,y in [0,N]

all_points = []             # Generate an array of all the possible points in grid
for i in range(10):
    for j in range(10):
        all_points.append([i,j])

# generate N random unique points from all_points
indices = np.random.choice(len(all_points), N, replace=False)
r = np.array([all_points[i] for i in indices])


# connect the generated points by a Delaunay triangulation
tri = Delaunay(r)

# create the corresponding adjacency matrix
M = np.zeros((N, N))
for simplex in tri.simplices:
    for i in range(len(simplex)):
        for j in range(i+1, len(simplex)):
            M[simplex[i], simplex[j]] = 1
            M[simplex[j], simplex[i]] = 1
```

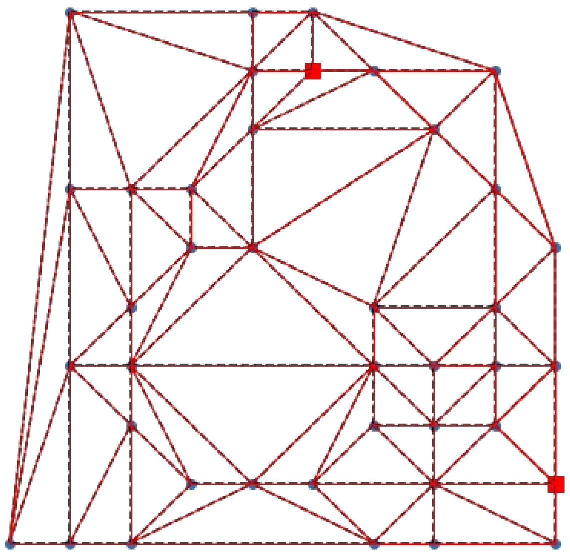
```python
# define the Euclidean distance all points in r
D = np.zeros((N, N))
for i in range(N):
    for j in range(i+1,N):
        D[i,j] = np.linalg.norm(r[i]-r[j])
        D[j,i] = D[i,j]

# make the distance inf for points that are not connected
for i in range(N):
    for j in range(i+1,N):
        if M[i,j] == 0:
            D[i,j] = np.inf
            D[j,i] = np.inf

T = M   # initialize the Pheromone matrix, initially equal to the adjacency matrix

distance_bird = 0
while distance_bird < 7:              # we don't want the starting and the target node to be too close
    # decide a starting node s
    s = np.random.randint(0, N)
    # decide a target node t
    t = np.random.randint(0, N)
    distance_bird = np.sqrt((r[t,0] - r[s,0])**2 + (r[t,1] - r[s,1])**2)

shortestDistance = []
shortestPath = []
for iteration in range(number_of_iterations):
    # initialize the paths of the ants
    T = T*(1-rho)
    paths = []
    for i in range(n):
        paths.append([s])
        for step in range(max_steps):
            next_node = branch_decision(paths[i][-1], T, D, M, alpha, beta)
            paths[i].append(next_node)
            if next_node == t:
                break

    # check if the target node is in the path of any of the ants
    shortestDist = np.inf
    for i in range(n):
        if t in paths[i]:
            paths[i] = simplify_sequence(paths[i])
            path_distance = total_distance(paths[i],D)
            if path_distance<shortestDist:
                shortestDist = path_distance
                shortPath = paths[i]
            for j in range(len(paths[i])-1):
                T[paths[i][j],paths[i][j+1]] += 1/path_distance
                T[paths[i][j+1],paths[i][j]] += 1/path_distance

    shortestDistance.append(shortestDist)
    shortestPath.append(shortPath)

    if iteration % 40 == 0:
        plt.figure(figsize=(7,7))
        plt.plot(r[:,0], r[:,1], 'o')
        plt.plot(r[s,0], r[s,1], 's', color='red', markersize=10)
        plt.plot(r[t,0], r[t,1], 's', color='red', markersize=10)
        for i in range(N):
            for j in range(i+1,N):
                if M[i,j] == 1:
                    plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'k--')

        p = shortestPath[iteration]
        for q in range(len(p)-1):
            i = p[q]
            j = p[q+1]
            plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'black',linewidth=5)

        plt.title('Iteration ' + str(iteration+1)+' - Shortest Path')
        plt.axis('off')
        plt.show()

    if iteration % 40 == 0:
        plt.figure(figsize=(7,7))
        plt.plot(r[:,0], r[:,1], 'o')
        plt.plot(r[s,0], r[s,1], 's', color='red', markersize=10)
        plt.plot(r[t,0], r[t,1], 's', color='red', markersize=10)
        for i in range(N):
            for j in range(i+1,N):
                if M[i,j] == 1:
                    plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'k--')

        for i in range(N):
            for j in range(i+1,N):
                plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'r',linewidth=T[i,j]*2)
        plt.title('Iteration ' + str(iteration+1) + ' - Pheromone Matrix')
```

```
        plt.axis('off')
        plt.show()

plt.plot(np.linspace(0,number_of_iterations-1,number_of_iterations),shortestDistance)
plt.xlabel("No. of Iterations")
plt.ylabel("Shortest Path Distance")
```

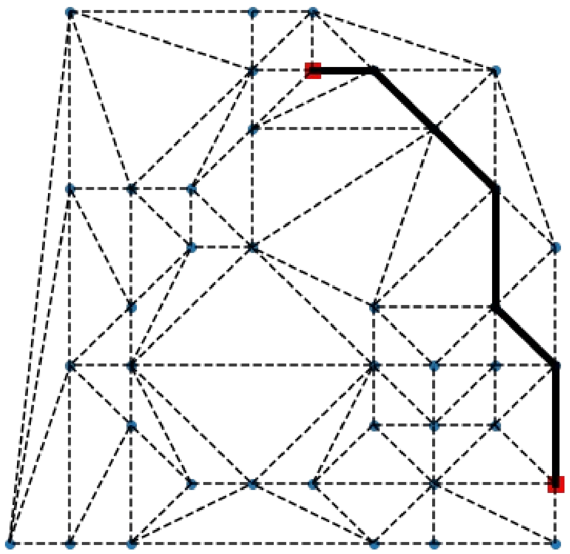
Iteration 1 - Shortest Path



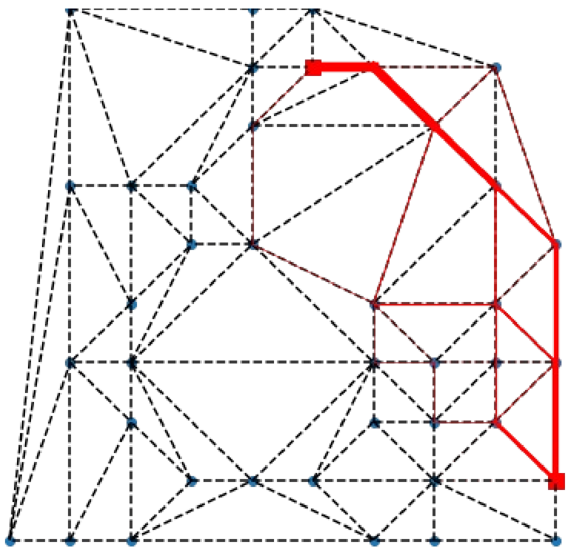
Iteration 1 - Pheromone Matrix



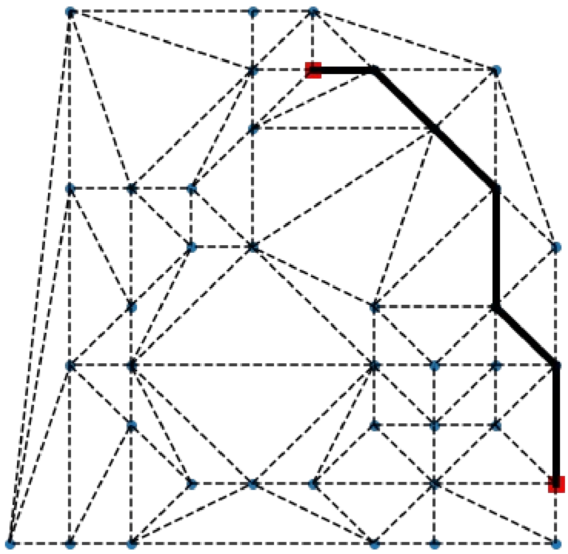
Iteration 41 - Shortest Path



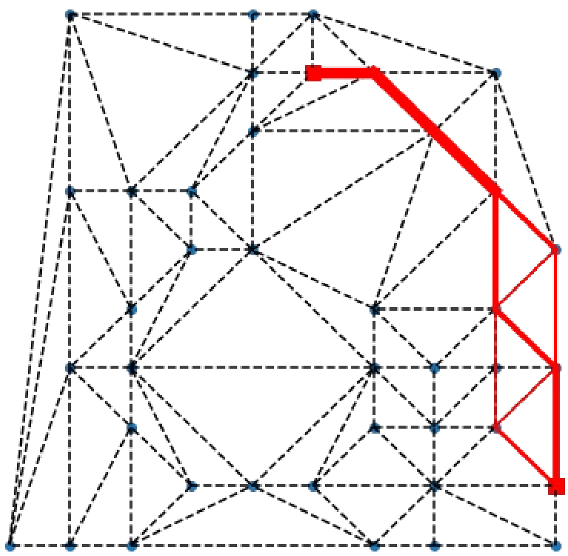
Iteration 41 - Pheromone Matrix



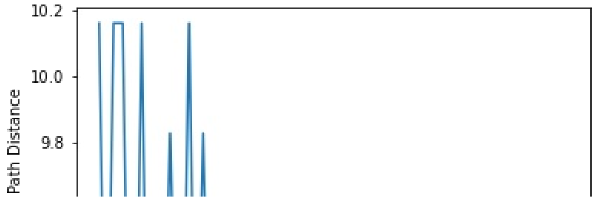
Iteration 81 - Shortest Path

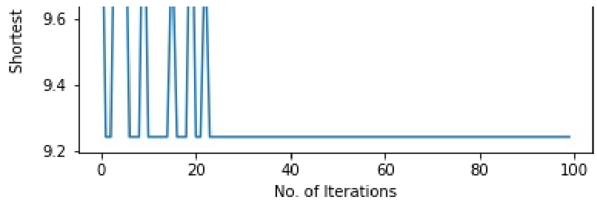


Iteration 81 - Pheromone Matrix



Out[3]: Text(0, 0.5, 'Shortest Path Distance')





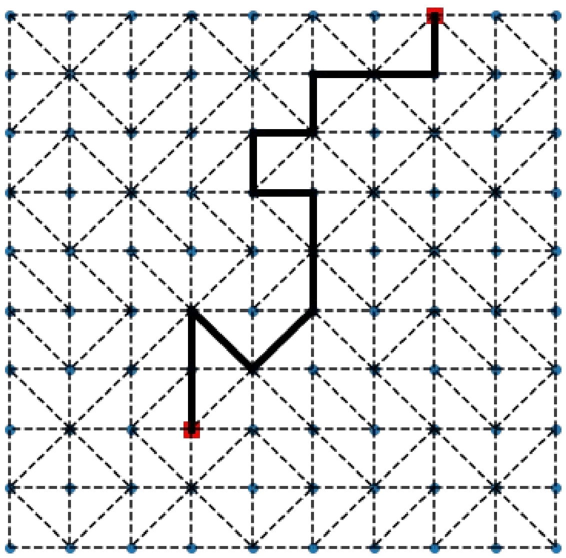
Exercise 15.8

In [4]:

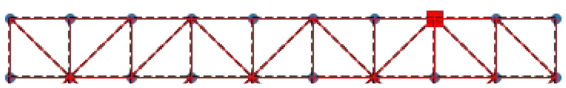
```
n = 20              # number of ants
alpha = 0.8         # alpha parameter
beta = 1          # beta parameter
rho = 0.5          # rho parameter
max_steps = 80   # maximum number of steps
number_of_iterations = 100 # number of iterations

N = 100  # number of nodes

# generate N points at unique random positions (x,y) with x,y in [0,N]

all_points = []              # Generate an array of all the possible points in grid
for i in range(10):
    for j in range(10):
        all_points.append([i,j])

# generate N random unique points from all_points
indices = np.random.choice(len(all_points), N, replace=False)
r = np.array([all_points[i] for i in indices])


# connect the generated points by a Delaunay triangulation
tri = Delaunay(r)

# create the corresponding adjacency matrix
M = np.zeros((N, N))
for simplex in tri.simplices:
    for i in range(len(simplex)):
        for j in range(i+1, len(simplex)):
            M[simplex[i], simplex[j]] = 1
            M[simplex[j], simplex[i]] = 1


# define the Euclidean distance all points in r
D = np.zeros((N, N))
for i in range(N):
    for j in range(i+1,N):
        D[i,j] = np.linalg.norm(r[i]-r[j])
        D[j,i] = D[i,j]

# make the distance inf for points that are not connected
for i in range(N):
    for j in range(i+1,N):
        if M[i,j] == 0:
            D[i,j] = np.inf
            D[j,i] = np.inf

T = M   # initialize the Pheromone matrix, initially equal to the adjacency matrix

distance_bird = 0
while distance_bird < 7:              # we don't want the starting and the target node to be too close
    # decide a starting node s
    s = np.random.randint(0, N)
    # decide a target node t
    t = np.random.randint(0, N)
    distance_bird = np.sqrt((r[t,0] - r[s,0])**2 + (r[t,1] - r[s,1])**2)

shortestDistance = []
shortestPath = []
for iteration in range(number_of_iterations):
    # initialize the paths of the ants
    T = T*(1-rho)
    paths = []
    for i in range(n):
        paths.append([s])
        for step in range(max_steps):
            next_node = branch_decision(paths[i][-1], T, D, M, alpha, beta)
            paths[i].append(next_node)
            if next_node == t:
                break
```

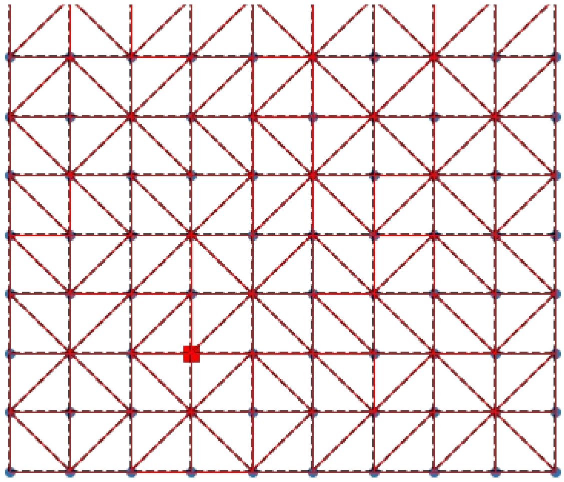
```python
    # check if the target node is in the path of any of the ants
    shortestDist = np.inf
    for i in range(n):
        if t in paths[i]:
            paths[i] = simplify_sequence(paths[i])
            path_distance = total_distance(paths[i],D)
            if path_distance<shortestDist:
                shortestDist = path_distance
                shortPath = paths[i]
            for j in range(len(paths[i])-1):
                T[paths[i][j],paths[i][j+1]] += 1/path_distance
                T[paths[i][j+1],paths[i][j]] += 1/path_distance

    shortestDistance.append(shortestDist)
    shortestPath.append(shortPath)

    if iteration % 40 == 0:
        plt.figure(figsize=(7,7))
        plt.plot(r[:,0], r[:,1], 'o')
        plt.plot(r[s,0], r[s,1], 's', color='red', markersize=10)
        plt.plot(r[t,0], r[t,1], 's', color='red', markersize=10)
        for i in range(N):
            for j in range(i+1,N):
                if M[i,j] == 1:
                    plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'k--')

        p = shortestPath[iteration]
        for q in range(len(p)-1):
            i = p[q]
            j = p[q+1]
            plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'black',linewidth=5)

        plt.title('Iteration ' + str(iteration+1)+' - Shortest Path')
        plt.axis('off')
        plt.show()

    if iteration % 40 == 0:
        plt.figure(figsize=(7,7))
        plt.plot(r[:,0], r[:,1], 'o')
        plt.plot(r[s,0], r[s,1], 's', color='red', markersize=10)
        plt.plot(r[t,0], r[t,1], 's', color='red', markersize=10)
        for i in range(N):
            for j in range(i+1,N):
                if M[i,j] == 1:
                    plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'k--')

        for i in range(N):
            for j in range(i+1,N):
                plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'r',linewidth=T[i,j]*2)
        plt.title('Iteration ' + str(iteration+1) + ' - Pheromone Matrix')
        plt.axis('off')
        plt.show()

plt.plot(np.linspace(0,number_of_iterations-1,number_of_iterations),shortestDistance)
plt.xlabel("No. of Iterations")
plt.ylabel("Shortest Path Distance")
```

Iteration 1 - Shortest Path

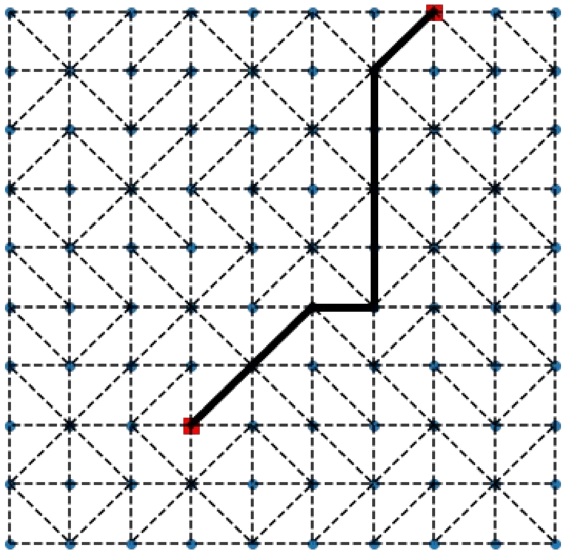


Iteration 1 - Pheromone Matrix

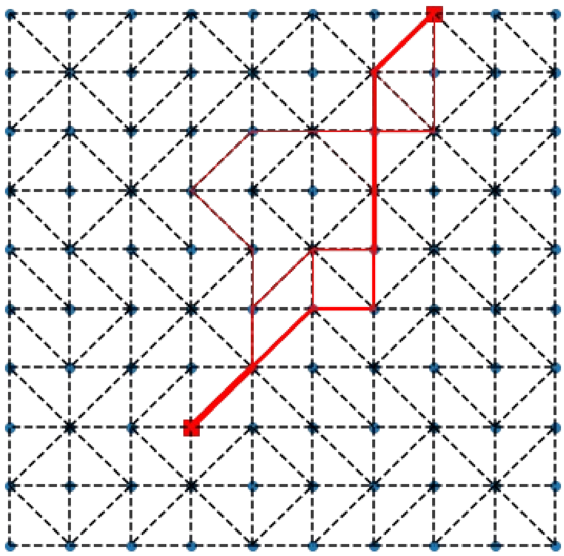




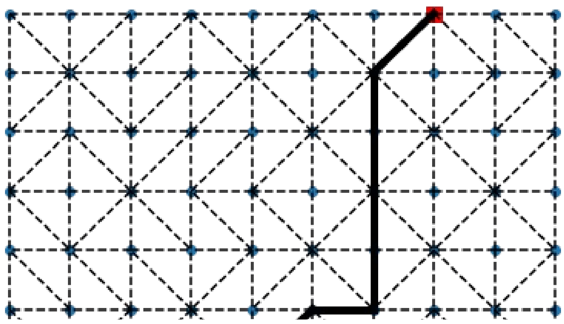
Iteration 41 - Shortest Path

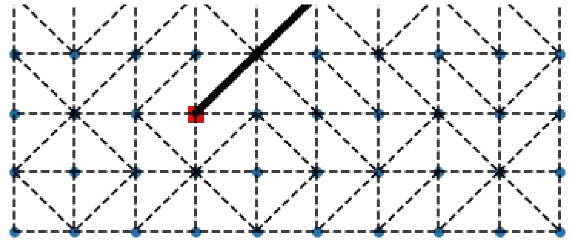


Iteration 41 - Pheromone Matrix

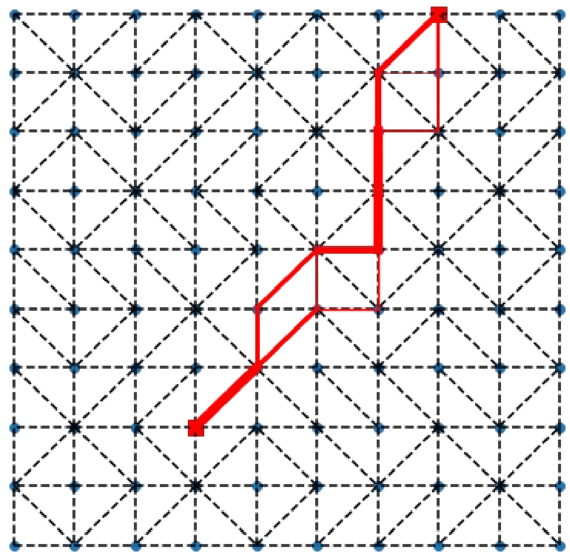


Iteration 81 - Shortest Path

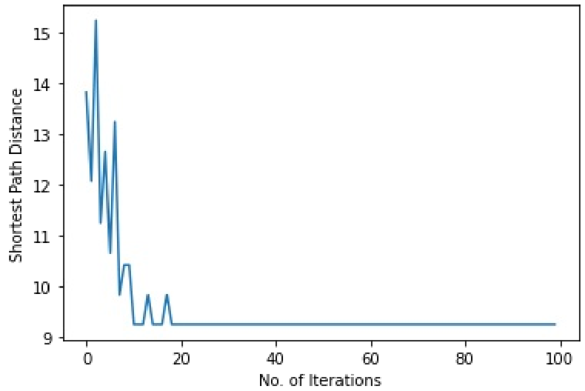




Iteration 81 - Pheromone Matrix



Out[4]: Text(0, 0.5, 'Shortest Path Distance')

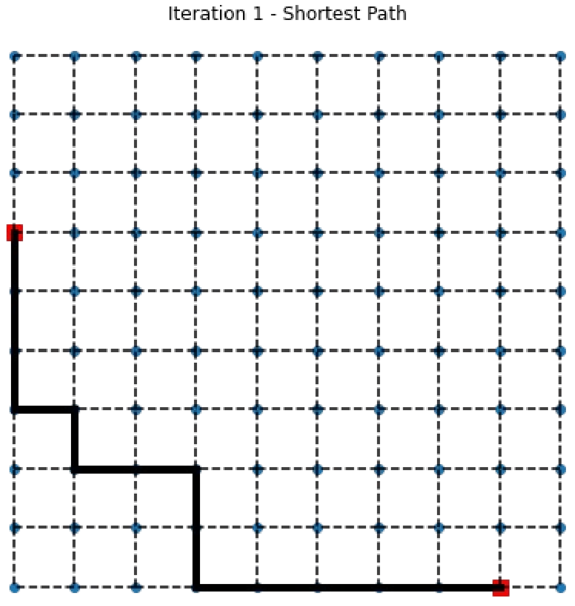


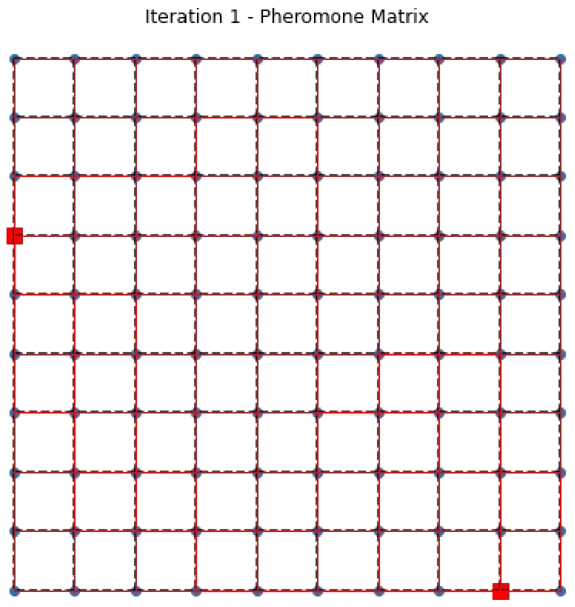
Exercise 15.9

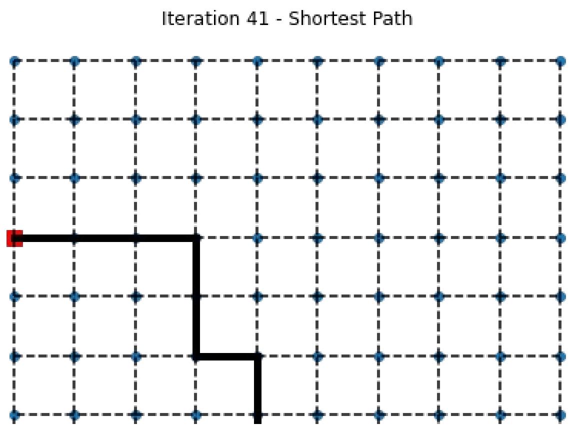
In [5]:
```
from scipy.spatial import Voronoi

n = 20               # number of ants
alpha = 0.8          # alpha parameter
beta = 1           # beta parameter
rho = 0.5           # rho parameter
max_steps = 80   # maximum number of steps
number_of_iterations = 300 # number of iterations

N = 100  # number of nodes

# generate N points at unique random positions (x,y) with x,y in [0,N]

all_points = []             # Generate an array of all the possible points in grid
for i in range(10):
    for j in range(10):
        all_points.append([i,j])

# generate N random unique points from all_points
indices = np.random.choice(len(all_points), N, replace=False)
r = np.array([all_points[i] for i in indices])
```

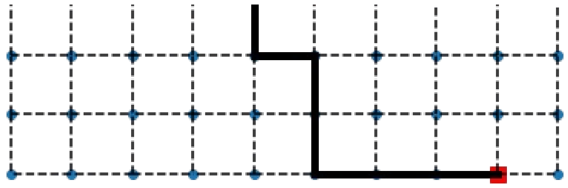
```python
# connect the generated points by a Delaunay triangulation
tri = Voronoi(r)

# create the corresponding adjacency matrix
M = np.zeros((N, N))
for simplex in tri.ridge_points:
    for i in range(len(simplex)):
        for j in range(i+1, len(simplex)):
            M[simplex[i], simplex[j]] = 1
            M[simplex[j], simplex[i]] = 1


# define the Euclidean distance all points in r
D = np.zeros((N, N))
for i in range(N):
    for j in range(i+1,N):
        D[i,j] = np.linalg.norm(r[i]-r[j])
        D[j,i] = D[i,j]

# make the distance inf for points that are not connected
for i in range(N):
    for j in range(i+1,N):
        if M[i,j] == 0:
            D[i,j] = np.inf
            D[j,i] = np.inf

T = M   # initialize the Pheromone matrix, initially equal to the adjacency matrix

distance_bird = 0
while distance_bird < 7:               # we don't want the starting and the target node to be too close
    # decide a starting node s
    s = np.random.randint(0, N)
    # decide a target node t
    t = np.random.randint(0, N)
    distance_bird = np.sqrt((r[t,0] - r[s,0])**2 + (r[t,1] - r[s,1])**2)

shortestDistance = []
shortestPath = []
for iteration in range(number_of_iterations):
    # initialize the paths of the ants
    T = T*(1-rho)
    paths = []
    for i in range(n):
        paths.append([s])
        for step in range(max_steps):
            next_node = branch_decision(paths[i][-1], T, D, M, alpha, beta)
            paths[i].append(next_node)
            if next_node == t:
                break

    # check if the target node is in the path of any of the ants
    shortestDist = np.inf
    for i in range(n):
        if t in paths[i]:
            paths[i] = simplify_sequence(paths[i])
            path_distance = total_distance(paths[i],D)
            if path_distance<shortestDist:
                shortestDist = path_distance
                shortPath = paths[i]
            for j in range(len(paths[i])-1):
                T[paths[i][j],paths[i][j+1]] += 1/path_distance
                T[paths[i][j+1],paths[i][j]] += 1/path_distance

    shortestDistance.append(shortestDist)
    shortestPath.append(shortPath)

    if iteration % 40 == 0:
        plt.figure(figsize=(7,7))
        plt.plot(r[:,0], r[:,1], 'o')
        plt.plot(r[s,0], r[s,1], 's', color='red', markersize=10)
        plt.plot(r[t,0], r[t,1], 's', color='red', markersize=10)
        for i in range(N):
            for j in range(i+1,N):
                if M[i,j] == 1:
                    plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'k--')

        p = shortestPath[iteration]
        for q in range(len(p)-1):
            i = p[q]
            j = p[q+1]
            plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'black',linewidth=5)

        plt.title('Iteration ' + str(iteration+1)+' - Shortest Path')
        plt.axis('off')
        plt.show()

    if iteration % 40 == 0:
        plt.figure(figsize=(7,7))
```

```
        plt.plot(r[:,0], r[:,1], 'o')
        plt.plot(r[s,0], r[s,1], 's', color='red', markersize=10)
        plt.plot(r[t,0], r[t,1], 's', color='red', markersize=10)
        for i in range(N):
            for j in range(i+1,N):
                if M[i,j] == 1:
                    plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'k--')

        for i in range(N):
            for j in range(i+1,N):
                plt.plot([r[i,0], r[j,0]], [r[i,1], r[j,1]],'r',linewidth=T[i,j]*2)
        plt.title('Iteration ' + str(iteration+1) + ' - Pheromone Matrix')
        plt.axis('off')
        plt.show()

plt.plot(np.linspace(0,number_of_iterations-1,number_of_iterations),shortestDistance)
plt.xlabel("No. of Iterations")
plt.ylabel("Shortest Path Distance")
```

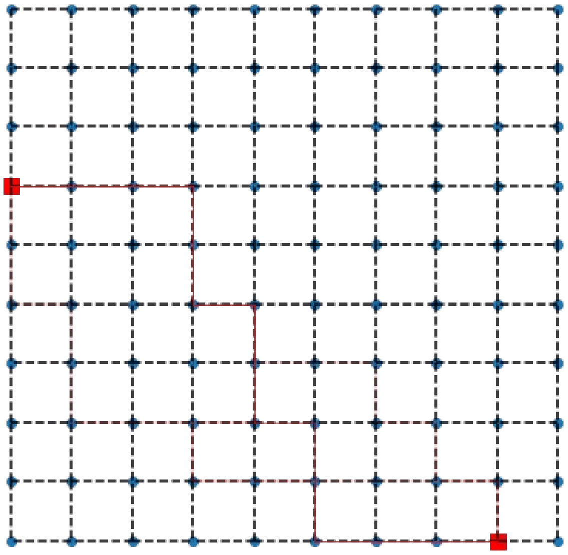




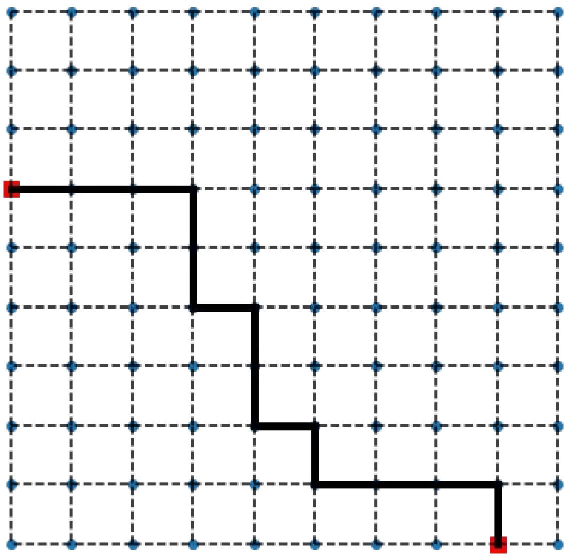




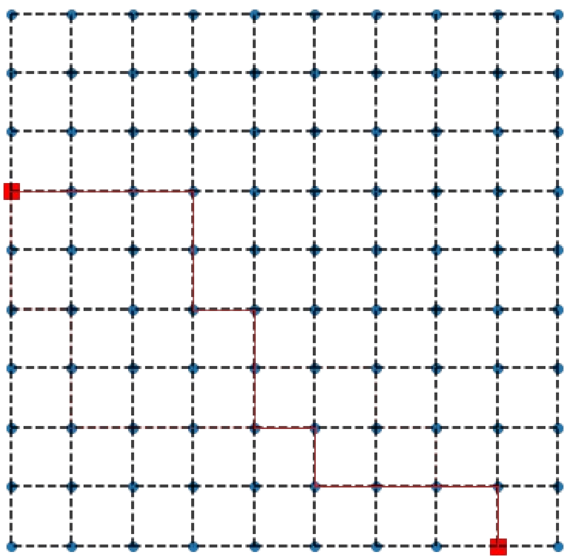
Iteration 41 - Pheromone Matrix



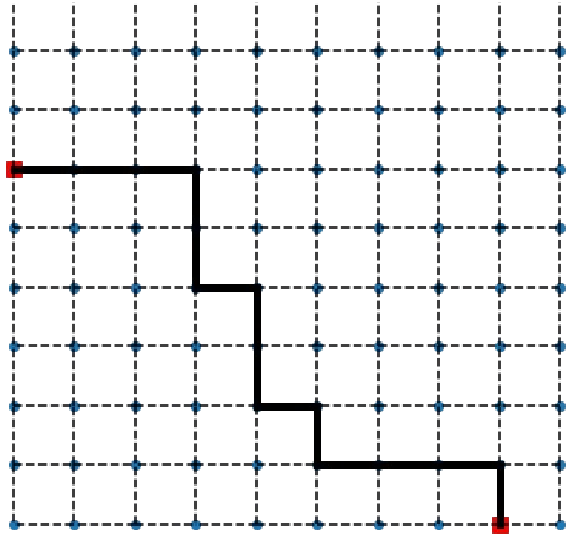
Iteration 81 - Shortest Path



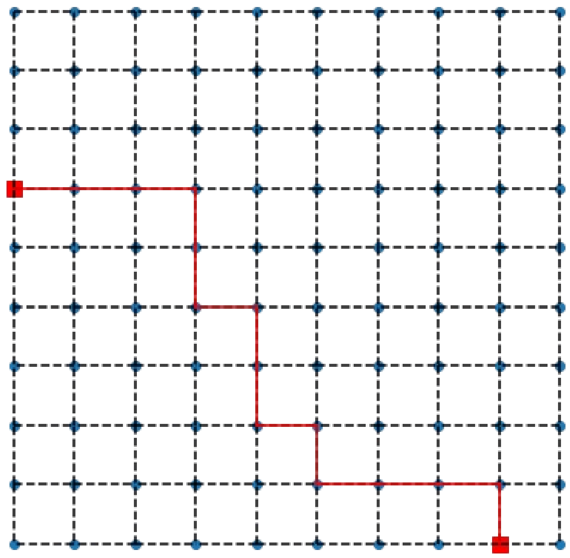
Iteration 81 - Pheromone Matrix



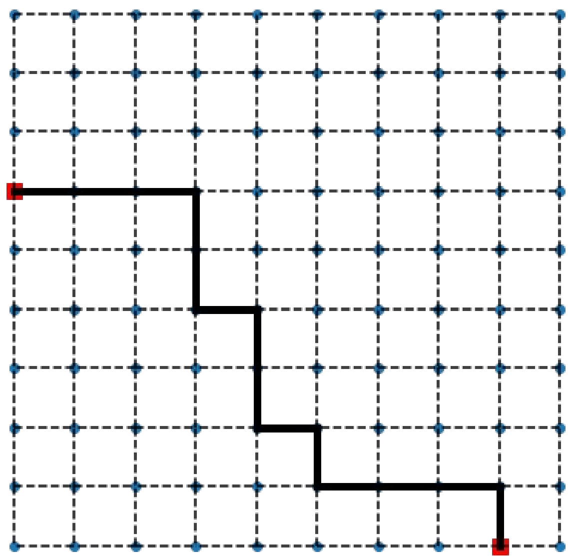
Iteration 121 - Shortest Path



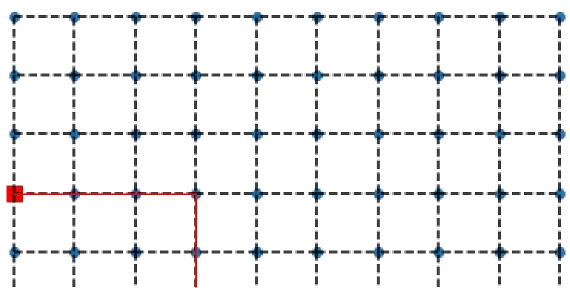
Iteration 121 - Pheromone Matrix

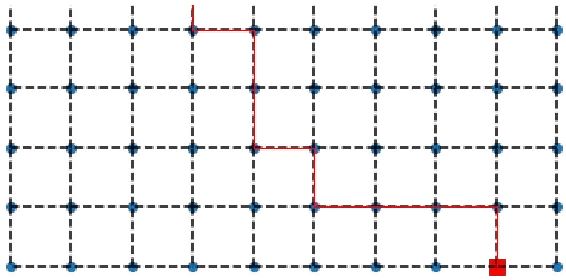


Iteration 161 - Shortest Path

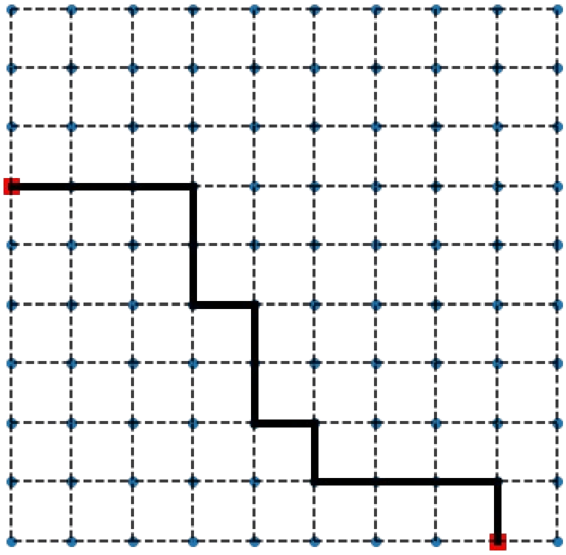


Iteration 161 - Pheromone Matrix

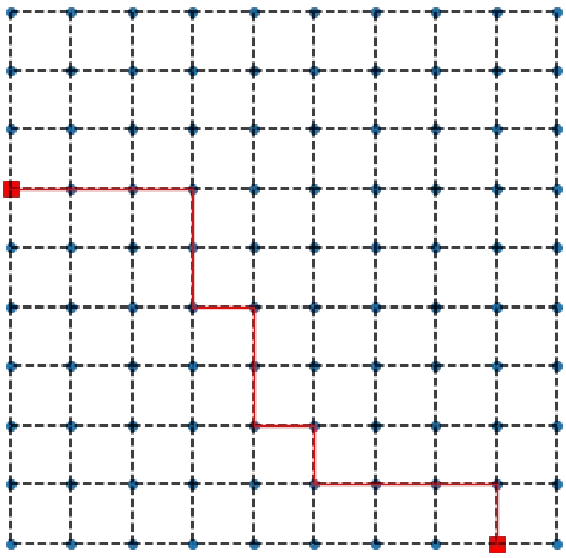




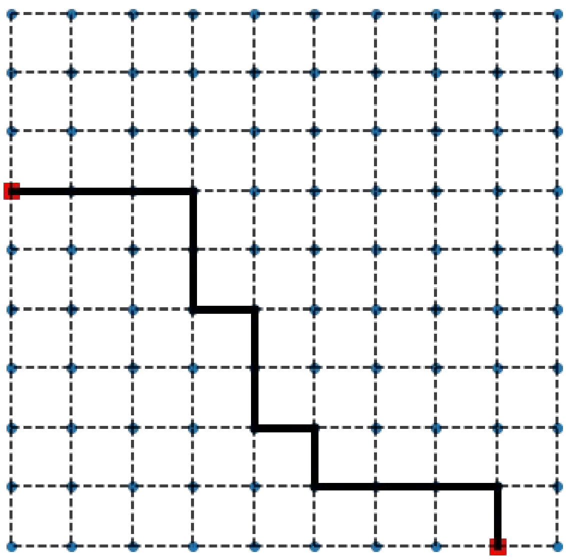
Iteration 201 - Shortest Path



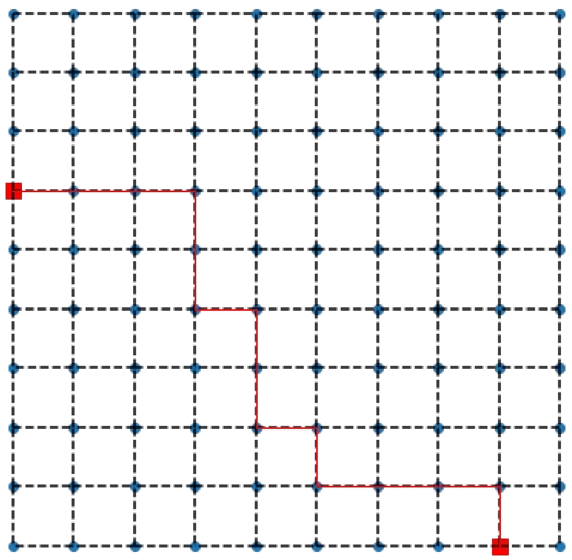
Iteration 201 - Pheromone Matrix



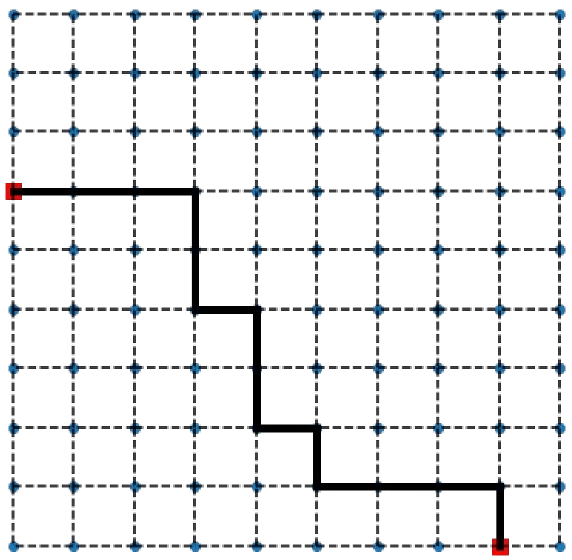
Iteration 241 - Shortest Path



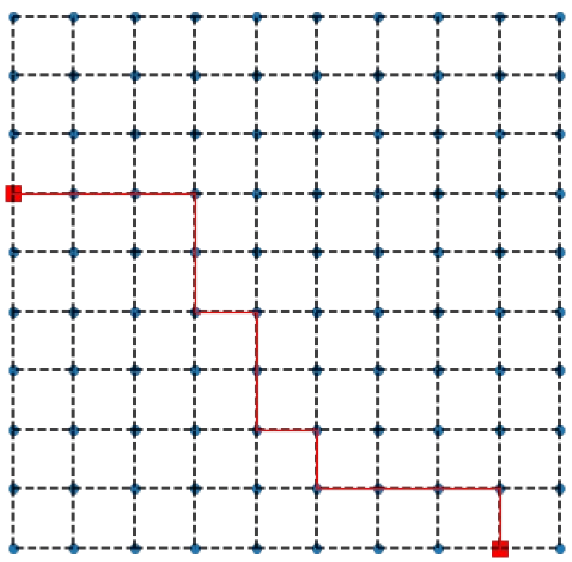
Iteration 241 - Pheromone Matrix



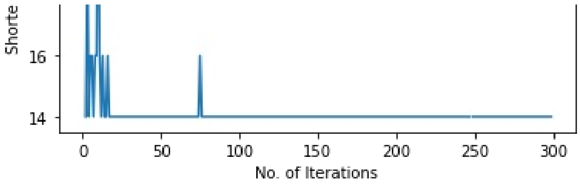
Iteration 281 - Shortest Path



Iteration 281 - Pheromone Matrix



```
Out[5]: Text(0, 0.5, 'Shortest Path Distance')
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js