Plot

July 19, 2020

```
In [ ]: import networkx as nx
        import matplotlib.pyplot as plot
        import math
        from itertools import permutations


        # This function takes
        # 1. a list of 2-tuples which represent the coordinates of the given points
        # 2. and a cycle to be visualized.
        def plot_cycle(coordinates, cycle):
            # Compute the x and y coordinates in the order according to the cycle
            x_coordinates = [coordinates[i][0] for i in cycle]
            y_coordinates = [coordinates[i][1] for i in cycle]

            # Add the first vertex of the cycle (to close the cycle)
            x_coordinates.append(coordinates[cycle[0]][0])
            y_coordinates.append(coordinates[cycle[0]][1])

            plot.plot(x_coordinates, y_coordinates, 'xb-', )
            plot.show()
```

```
In [ ]: # This function computes the weight of the given cycle.
        def cycle_length(g, cycle):
            # Checking that the number of vertices in the graph equals the number of vertices in
            assert len(cycle) == g.number_of_nodes()
            # Write your code here.
            return sum(g[cycle[i]][cycle[i + 1]]['weight'] for i in range(len(cycle) - 1)) + g[c
```

```
In [ ]: # This function computes the distance between two points.
        def dist(x1, y1, x2, y2):
            return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
```

```
In [ ]: # This function receives a list of 2-tuples representing the points' coordinates,
        # and returns the corresponding graph.
        def get_graph(coordinates):
            g = nx.Graph()
            n = len(coordinates)
            for i in range(n):
```

```
            for j in range(i + 1):
                g.add_edge(i, j, weight=dist(coordinates[i][0], coordinates[i][1], coordinat
        return g
```

In [ ]:
```
# This function iterates through all permutations and returns an optimal cycle.
# You can implement any other algorithm here and visualize it.
#
# Note that in the previous questions you were asked to compute the length of an optimal
# while this function returns an optimal cycle itself (so that we could plot it later).
def all_permutations(g):
    # n is the number of vertices.
    n = g.number_of_nodes()
    opt = float('inf')

    # Iterate through all permutations of n vertices
    for p in permutations(range(n)):
        if cycle_length(g, p) < opt:
            opt = cycle_length(g, p)
            opt_perm = p

    return opt_perm
```

In [ ]:
```
# Example 1
# Consider the following 3 points.
coordinates = [
    (166, 282),
    (43, 79),
    (285, 44)
    ]
# Create a corresponding graph.
g = get_graph(coordinates)
# Compute an optimal Hamiltonian path using some algorithm (e.g., the all_permutations a
cycle = all_permutations(g)
# Plot the resulting cycle
plot_cycle(coordinates, cycle)
```

In [ ]:
```
# Example 2
# Consider the following 5 points.
coordinates = [
    (284, 87),
    (183, 254),
    (113, 185),
    (159, 38),
    (271, 257)
    ]
# Create a corresponding graph.
g = get_graph(coordinates)
# Compute an optimal Hamiltonian path using some algorithm (e.g., the all_permutations a
```

```
    cycle = all_permutations(g)
    # Plot the resulting cycle
    plot_cycle(coordinates, cycle)
```

In [ ]:
```
# Example 3
# Consider the following 7 points.
coordinates = [
    (231, 72),
    (68, 9),
    (11, 90),
    (237, 116),
    (168, 112),
    (141, 69),
    (17, 18)
    ]
# Create a corresponding graph.
g = get_graph(coordinates)
# Compute an optimal Hamiltonian path using some algorithm (e.g., the all_permutations a
cycle = all_permutations(g)
# Plot the resulting cycle
plot_cycle(coordinates, cycle)
```