Evolutionary Computation - Assignment 2 Report

- Kajetan Sulwiński 151954
- Mikołaj Marmurowicz 151956

Imports

In [1]:
```
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib.cm import ScalarMappable
from matplotlib import MatplotlibDeprecationWarning
import warnings
```

Problem instance reading and cost scaling

In [2]:
```
dfTSPA = pd.read_csv('.\\TSPA.csv', sep=';', names=['X', 'Y', 'Cost'])
dfTSPB = pd.read_csv('.\\TSPB.csv', sep=';', names=['X', 'Y', 'Cost'])

min_cost_A = dfTSPA['Cost'].min()
max_cost_A = dfTSPA['Cost'].max()
dfTSPA['Normalized_Cost'] = (dfTSPA['Cost'] - min_cost_A) / (max_cost_A - min_cost_

min_cost_B = dfTSPB['Cost'].min()
max_cost_B = dfTSPB['Cost'].max()
dfTSPB['Normalized_Cost'] = (dfTSPB['Cost'] - min_cost_B) / (max_cost_B - min_cost_
```

Problem description

Problem

We were to find a cycle that consisted of exactly 50% of the available nodes, where each node had its own cost along with x and y coordinates. The objective function was a sum of node costs and distances (Euclidean) between each traveled node.

Solution implementation

We have added two new methods to solve this problem. All of them were created in C++.

- **2-Regret solution**

- **Input:**
  - `starting_node_id` : ID of the first node added to the cycle
  - `distance_matrix` : A symmetric matrix of distances between nodes
  - `nodes_cost` : An array of nodes cost
  - `nodes_in_cycle` : An integer defining the desired number of nodes in the cycle
- **Output:**
  - An array of `nodes_in_cycle` node IDs
- **Function:**

```
func generateTwoRegretCycle():
    cycle = []
    add starting_node_id to cycle
    increases = []
    for node not in cycle:// Add second node to cycle using only
the change in function
        ease = distance_matrix[starting_node_id][node] +
nodes_cost[node]
        increases[node] = func_increase
    add arg_min(increases) to cycle
    for element_to_add_count in 2…nodes_in_cycle:
        regrets = []
        best_position_to_add = []
        for node not in cycle:
            increasesForNode = []
            for left_n in cycle:
                right_n = get_next_node_in_cycle(cycle,
from=left_n)
                increase = distance_matrix[node][leftN] +
distance_matrix[node][rightN] - distance_matrix[leftN][rightN] +
nodes_cost[node]
                add_to_list(increasesForNode, increase)
                regret = second_smallest_increase(increasesForNode)
- smallest_increase(increasesForNode)
                regrets[node] = regret
                best_position_to_add[node] =
get_position_with_smallest_increase(cycle, node)
        best_node_to_add = arg_max(regrets)
        add best_node_to_add to cycle at best_position_to_add[node]
```

- **Weighted 2-Regret**
  - **Input:**
    - `starting_node_id` : ID of the first node added to the cycle
    - `distance_matrix` : A symmetric matrix of distances between nodes
    - `nodes_cost` : An array of nodes cost
    - `nodes_in_cycle` : An integer defining the desired number of nodes in the

cycle
    - `regret_weight` : A float defining the weight of regret value in the objective function - in range [0; 1]
- **Output:**
    - An array of `nodes_in_cycle` nodes IDs
- **Function:**

```
func generateWeightedTwoRegretCycle():
    cycle = []
    add starting_node_id to cycle
    increases = []
    for node not in cycle:// Add second node to cycle using only
the change in function
        func_increase = distance_matrix[starting_node_id][node] +
nodes_cost[node]
        increases[node] = func_increase
    add arg_min(increases) to cycle
    for element_to_add_count in 2…nodes_in_cycle:
        objective_functions = []
        best_position_to_add = []
        for node not in cycle:
            increasesForNode = []
            for left_n in cycle:
                right_n = get_next_node_in_cycle(cycle,
from=left_n)
                increase = distance_matrix[node][leftN] +
distance_matrix[node][rightN] - distance_matrix[leftN][rightN] +
nodes_cost[node]
                add_to_list(increasesForNode, increase)
                regret = second_smallest_increase(increasesForNode)
- smallest_increase(increasesForNode)
                objective_functions[node] = -regret_weight * regret
+ (1 - regret_weight) * smallest_increase(increasesForNode)
                best_position_to_add[node] =
get_position_with_smallest_increase(cycle, node)
        best_node_to_add = arg_min(objective_functions)
        add best_node_to_add to cycle at best_position_to_add[node]
```

Presenting the results

Results presented as minimum, average and maximum of objective function

Presented in a table below, each method and each problem instance is shown.

In [3]:
```
file_paths = ['.\\TSPAKRegret.csv','.\\TSPAKRegretGreedyCombination.csv','..\\Lab1\
methods = ['TSPA 2-regret Cycle', 'TSPA 2-regret Weighted Cycle','TSPA Greedy Cycle
results = []
```

```
best_solutions = []

for file_path, method in zip(file_paths, methods):
    df = pd.read_csv(file_path)
    costs = df.iloc[:, -1]
    minimum = costs.min()
    maximum = costs.max()
    mean = round(costs.mean(), 2)
    results.append([minimum, mean, maximum])

    min_sol = df.loc[costs.idxmin()][:-1].to_list()
    best_solutions.append(min_sol)

result_df = pd.DataFrame(results, columns=['Minimum', 'Average', 'Maximum'], index=
result_df
```

Out[3]:

	Minimum	Average	Maximum
TSPA 2-regret Cycle	105692	115570.82	126951
TSPA 2-regret Weighted Cycle	71108	72134.84	73395
TSPA Greedy Cycle	71237	73036.23	75002
TSPA NN on all vertices	71227	73293.75	76036
TSPA NN on last node	83182	85110.16	89433
TSPA Random	236302	263481.34	293567
TSPB 2-regret Cycle	67809	72746.68	78406
TSPB 2-regret Weighted Cycle	47144	50913.93	55700
TSPB Greedy Cycle	48898	51852.88	58531
TSPB NN on all vertices	44377	47444.68	53019
TSPB NN on last node	52319	54385.49	59030
TSPB Random	188701	213568.36	239495

Aditional information regarding the running time of each method (in milliseconds).

In [4]:
```
methods_for_times = ['2-Regret', 'Weighted 2-Regret', 'Random', 'NN on last node',
times_files = ['.\\times.csv', '..\\Lab1\\times.csv']
times_df = pd.DataFrame()
for time_file in times_files:
    df_temp = pd.read_csv(time_file, header=None).iloc[:, :-1]
    times_df = pd.concat([times_df, df_temp], axis=1)
times_df.columns = methods_for_times

times_stats = []
for method in methods_for_times:
    min_value = times_df[method].min()
    max_value = times_df[method].max()
    avg_value = times_df[method].mean()
```

```
    times_stats.append([min_value, avg_value, max_value])
stats_time_df = pd.DataFrame(times_stats, columns=['Minimum', 'Average', 'Maximum']
stats_time_df
```

Out[4]:

	Minimum	Average	Maximum
2-Regret	24.5655	27.541826	38.6567
Weighted 2-Regret	24.4544	27.523027	35.1762
Random	0.0062	0.007379	0.0411
NN on last node	0.3182	0.359385	0.8055
NN on all vertices	11.2167	11.826113	14.0679
Greedy Cycle	11.5659	12.360844	14.9002

Visualization of the best path for each method

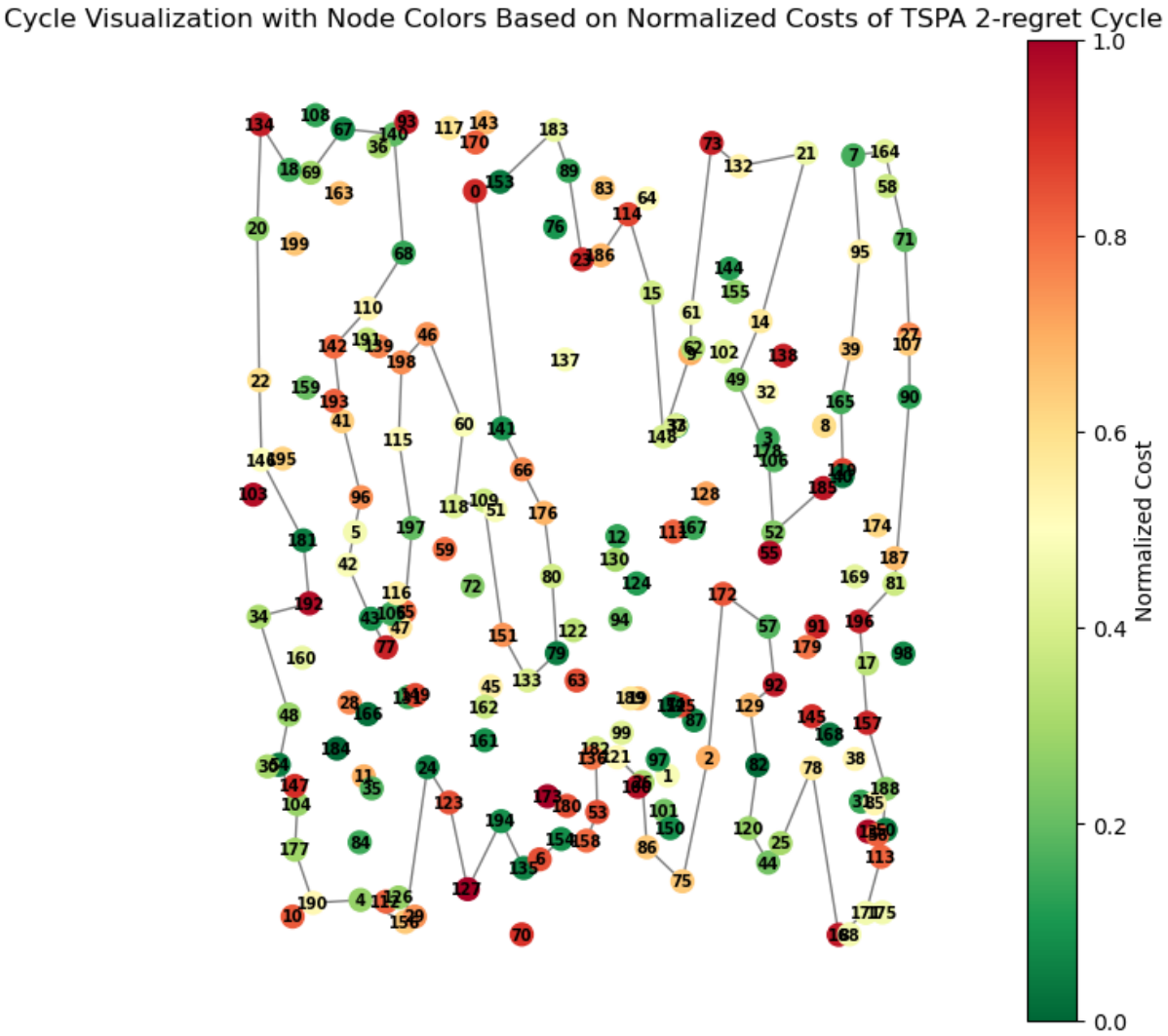
Additionally, a list of node indices is presented.

In [5]:

```
warnings.filterwarnings("ignore", category=MatplotlibDeprecationWarning)
cmap = plt.cm.get_cmap('RdYlGn_r')

for count, method in enumerate(methods):
    print(method)
    print(best_solutions[count])

    G = nx.Graph()
    positions = {}

    if count < 6:
        for idx in best_solutions[count]:
            G.add_node(idx, size=dfTSPA.loc[idx, 'Normalized_Cost'])
            positions[idx] = (dfTSPA.loc[idx, 'X'], dfTSPA.loc[idx, 'Y'])

        for idx in [i for i in range(0,200) if i not in best_solutions[count]]:
            G.add_node(idx, size=dfTSPA.loc[idx, 'Normalized_Cost'])
            positions[idx] = (dfTSPA.loc[idx, 'X'], dfTSPA.loc[idx, 'Y'])

        for i in range(len(best_solutions[count]) - 1):
            G.add_edge(best_solutions[count][i], best_solutions[count][i + 1])
        G.add_edge(best_solutions[count][-1], best_solutions[count][0])


        normalized_costs = dfTSPA['Normalized_Cost']
        norm = mcolors.Normalize(vmin=normalized_costs.min(), vmax=normalized_costs
        node_colors = [cmap(norm(dfTSPA.loc[idx, 'Normalized_Cost'])) for idx in ra

    else:
        for idx in best_solutions[count]:
            G.add_node(idx, size=dfTSPB.loc[idx, 'Normalized_Cost'])
            positions[idx] = (dfTSPB.loc[idx, 'X'], dfTSPB.loc[idx, 'Y'])
```

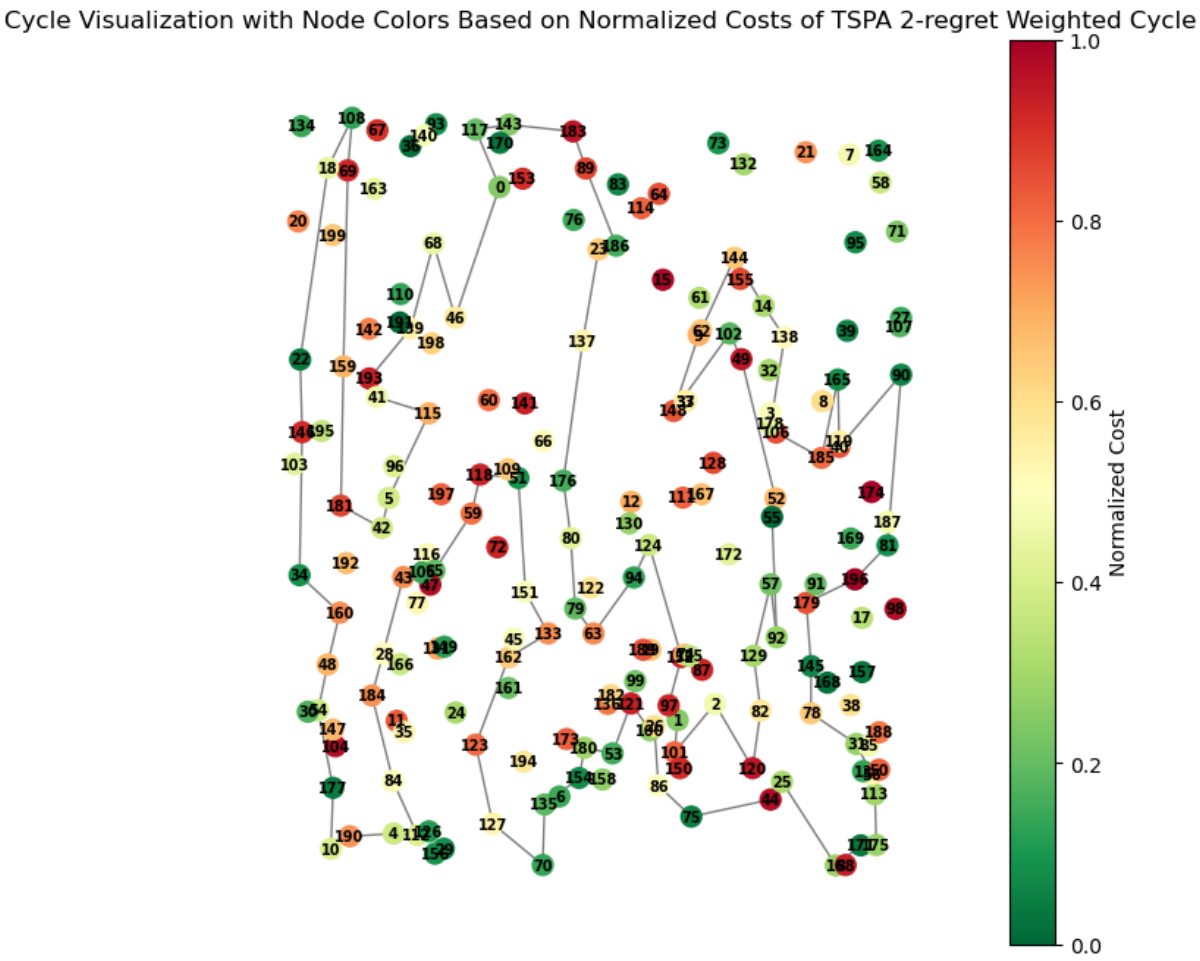
```
        for idx in [i for i in range(0,200) if i not in best_solutions[count]]:
            G.add_node(idx, size=dfTSPB.loc[idx, 'Normalized_Cost'])
            positions[idx] = (dfTSPB.loc[idx, 'X'], dfTSPB.loc[idx, 'Y'])

        for i in range(len(best_solutions[count]) - 1):
            G.add_edge(best_solutions[count][i], best_solutions[count][i + 1])
        G.add_edge(best_solutions[count][-1], best_solutions[count][0])

        normalized_costs = dfTSPB['Normalized_Cost']
        norm = mcolors.Normalize(vmin=normalized_costs.min(), vmax=normalized_costs
        node_colors = [cmap(norm(dfTSPB.loc[idx, 'Normalized_Cost'])) for idx in ra

    fig, ax = plt.subplots(figsize=(8, 8))
    nx.draw(G, pos=positions, with_labels=True, node_color=node_colors, node_size=1

    sm = ScalarMappable(cmap=cmap, norm=norm)
    sm.set_array([])

    cbar = plt.colorbar(sm, ax=ax)
    cbar.set_label('Normalized Cost')

    plt.title(f"Cycle Visualization with Node Colors Based on Normalized Costs of {
    plt.show()
```

```
TSPA 2-regret Cycle
[52, 178, 49, 14, 21, 132, 73, 61, 9, 148, 15, 114, 186, 23, 89, 183, 153, 0, 141, 6
6, 176, 80, 79, 133, 151, 109, 118, 60, 46, 198, 115, 197, 65, 77, 43, 42, 96, 41, 1
42, 110, 68, 140, 67, 69, 18, 134, 20, 22, 146, 181, 192, 34, 48, 54, 104, 177, 190,
4, 156, 24, 123, 127, 194, 135, 6, 154, 158, 53, 182, 121, 26, 86, 75, 2, 172, 57, 9
2, 129, 82, 120, 44, 25, 78, 16, 171, 113, 188, 157, 196, 81, 90, 27, 71, 164, 7, 95
, 39, 165, 119, 185]
```



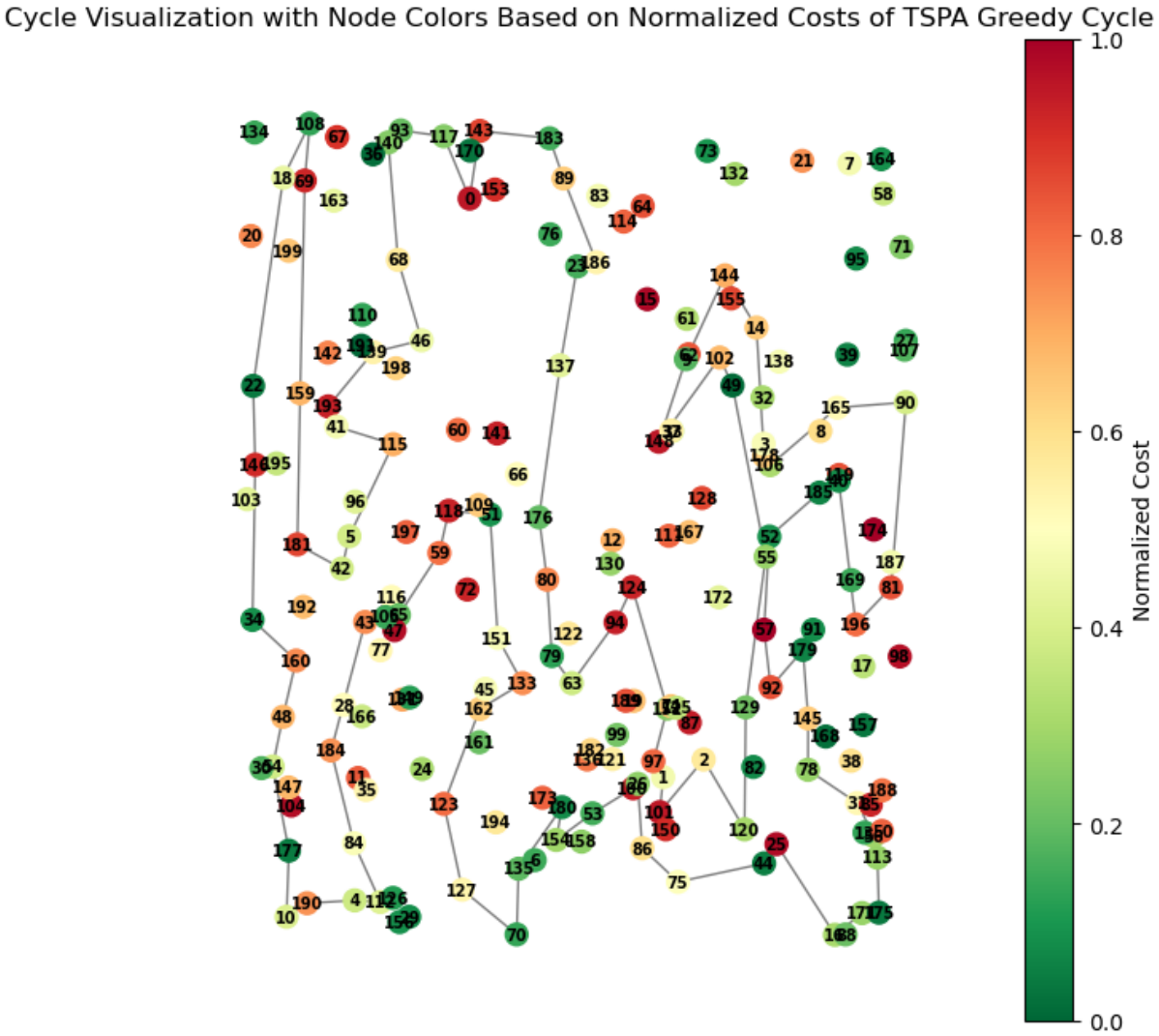
TSPA 2-regret Weighted Cycle

[143, 117, 0, 46, 68, 139, 193, 41, 115, 5, 42, 181, 159, 69, 108, 18, 22, 146, 34, 160, 48, 54, 177, 10, 190, 4, 112, 84, 184, 43, 116, 65, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 154, 180, 53, 121, 100, 26, 86, 75, 44, 25, 16, 171, 175, 113, 56, 31, 78, 145, 179, 196, 81, 90, 40, 165, 185, 106, 178, 138, 14, 144, 62, 9, 148, 102, 49, 52, 55, 92, 57, 129, 82, 120, 2, 101, 1, 97, 152, 124, 94, 63, 79, 80, 176, 137, 23, 186, 89, 183]

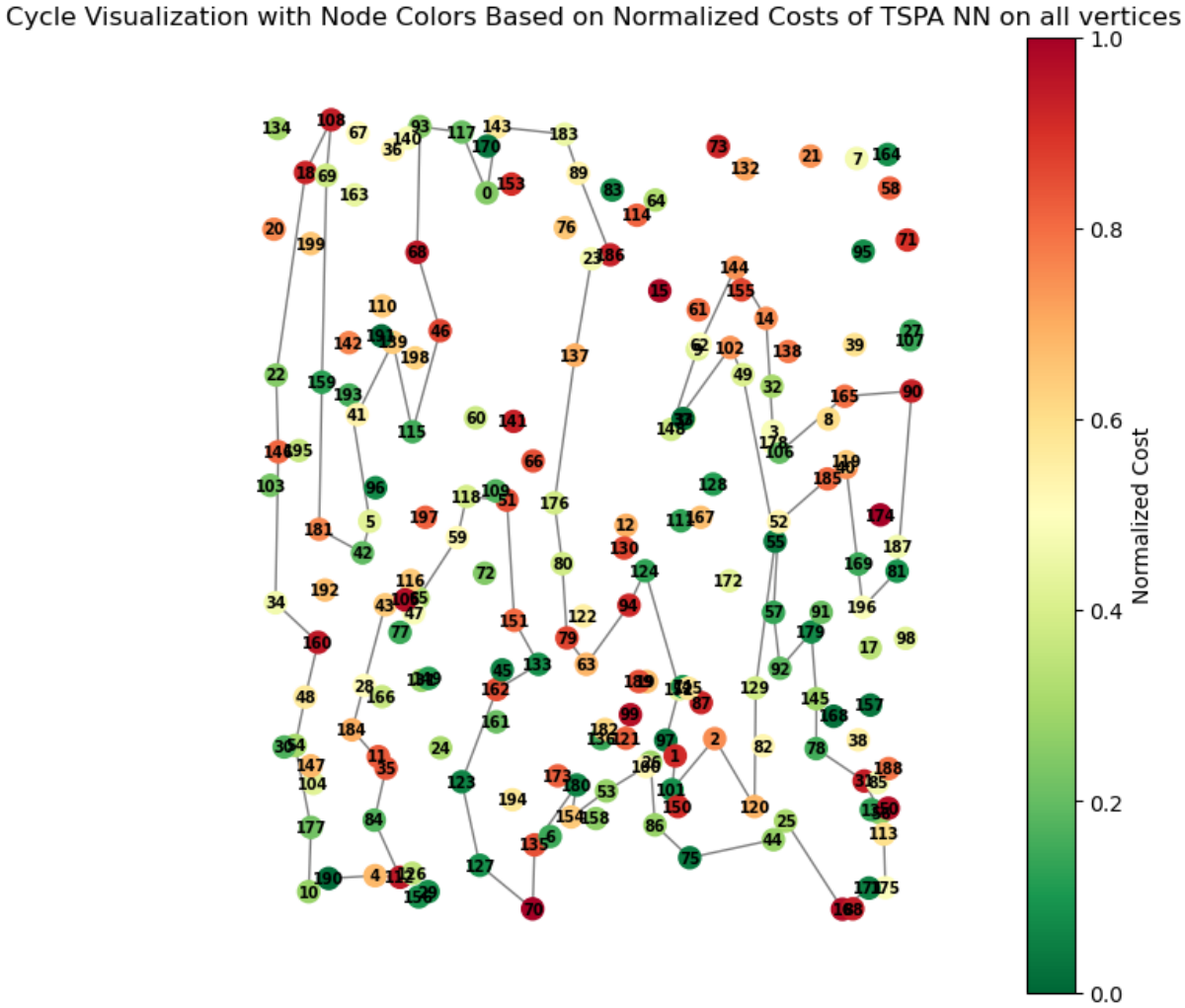


TSPA Greedy Cycle

[117, 93, 140, 68, 46, 139, 193, 41, 115, 5, 42, 181, 159, 69, 108, 18, 22, 146, 34, 160, 48, 54, 177, 10, 190, 4, 112, 84, 184, 43, 116, 65, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 180, 154, 53, 100, 26, 86, 75, 44, 25, 16, 171, 175, 113, 56, 31, 78, 145, 179, 92, 57, 52, 185, 119, 40, 196, 81, 90, 165, 106, 178, 14, 144, 62, 9, 148, 102, 49, 55, 129, 120, 2, 101, 1, 97, 152, 124, 94, 63, 79, 80, 176, 137, 23, 186, 89, 183, 143, 0]

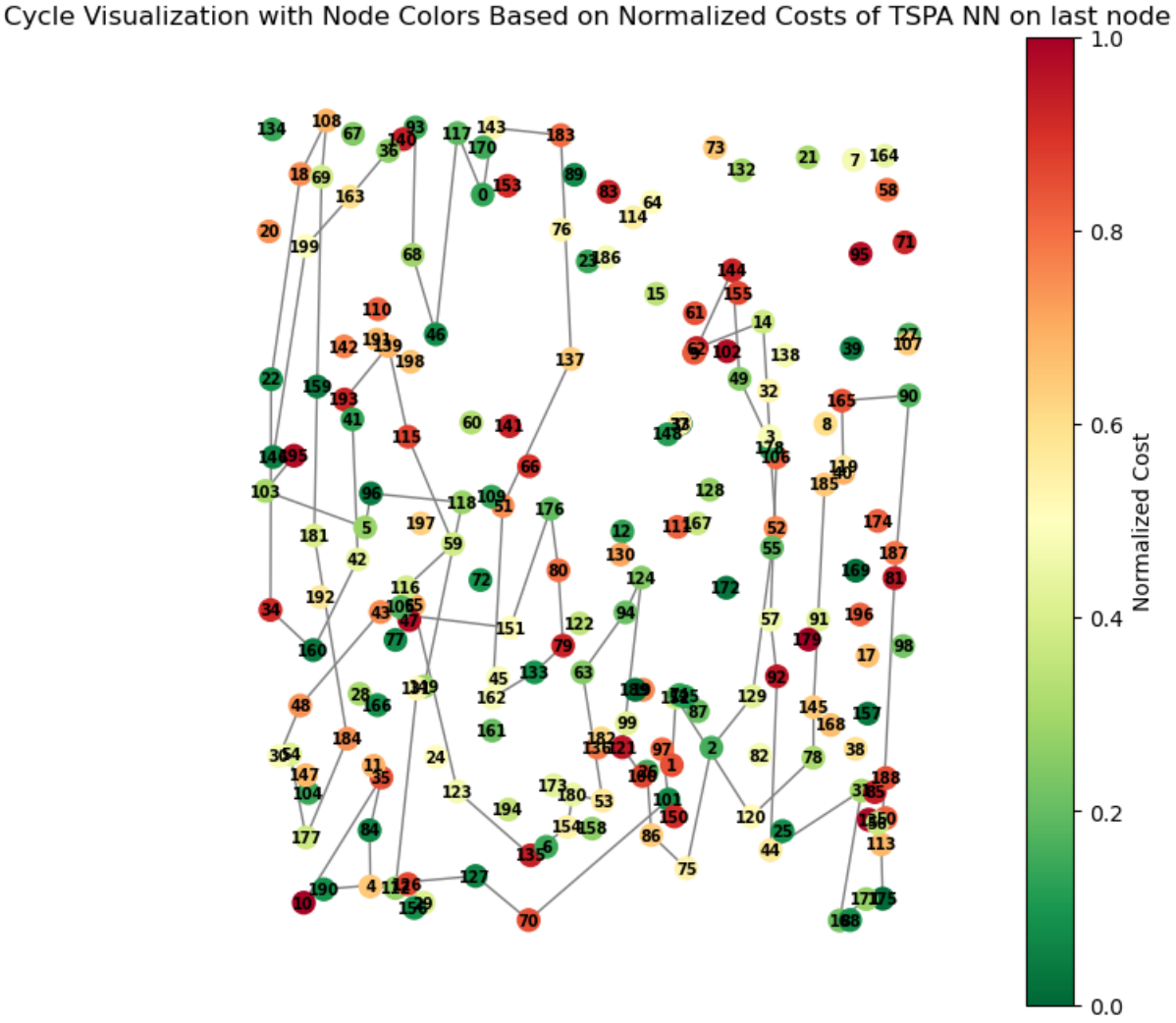


TSPA NN on all vertices
[93, 117, 0, 143, 183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 124, 152, 97, 1, 101, 2, 120, 129, 55, 49, 102, 148, 9, 62, 144, 14, 178, 106, 165, 90, 81, 196, 40, 119, 185, 52, 57, 92, 179, 145, 78, 31, 56, 113, 175, 171, 16, 25, 44, 75, 86, 26, 100, 53, 154, 180, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 65, 116, 43, 184, 35, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 146, 22, 18, 108, 69, 159, 181, 42, 5, 193, 41, 139, 115, 46, 68]



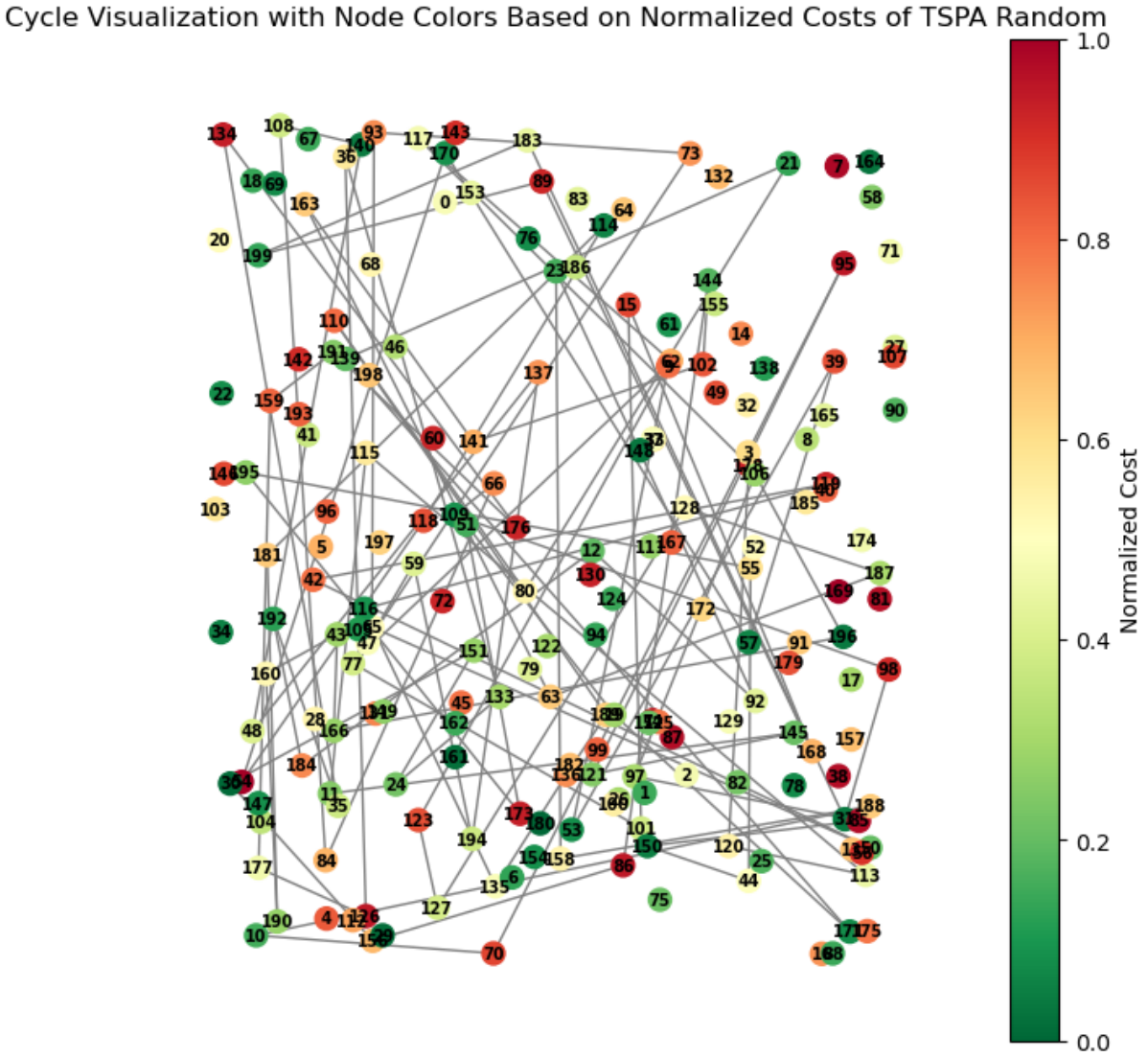
TSPA NN on last node

[124, 94, 63, 53, 180, 154, 135, 123, 65, 116, 59, 115, 139, 193, 41, 42, 160, 34, 22, 18, 108, 69, 159, 181, 184, 177, 54, 30, 48, 43, 151, 176, 80, 79, 133, 162, 51, 137, 183, 143, 0, 117, 46, 68, 93, 140, 36, 163, 199, 146, 195, 103, 5, 96, 118, 149, 131, 112, 4, 84, 35, 10, 190, 127, 70, 101, 97, 1, 152, 120, 78, 145, 185, 40, 165, 90, 81, 113, 175, 171, 16, 31, 44, 92, 57, 106, 49, 144, 62, 14, 178, 52, 55, 129, 2, 75, 86, 26, 100, 121]



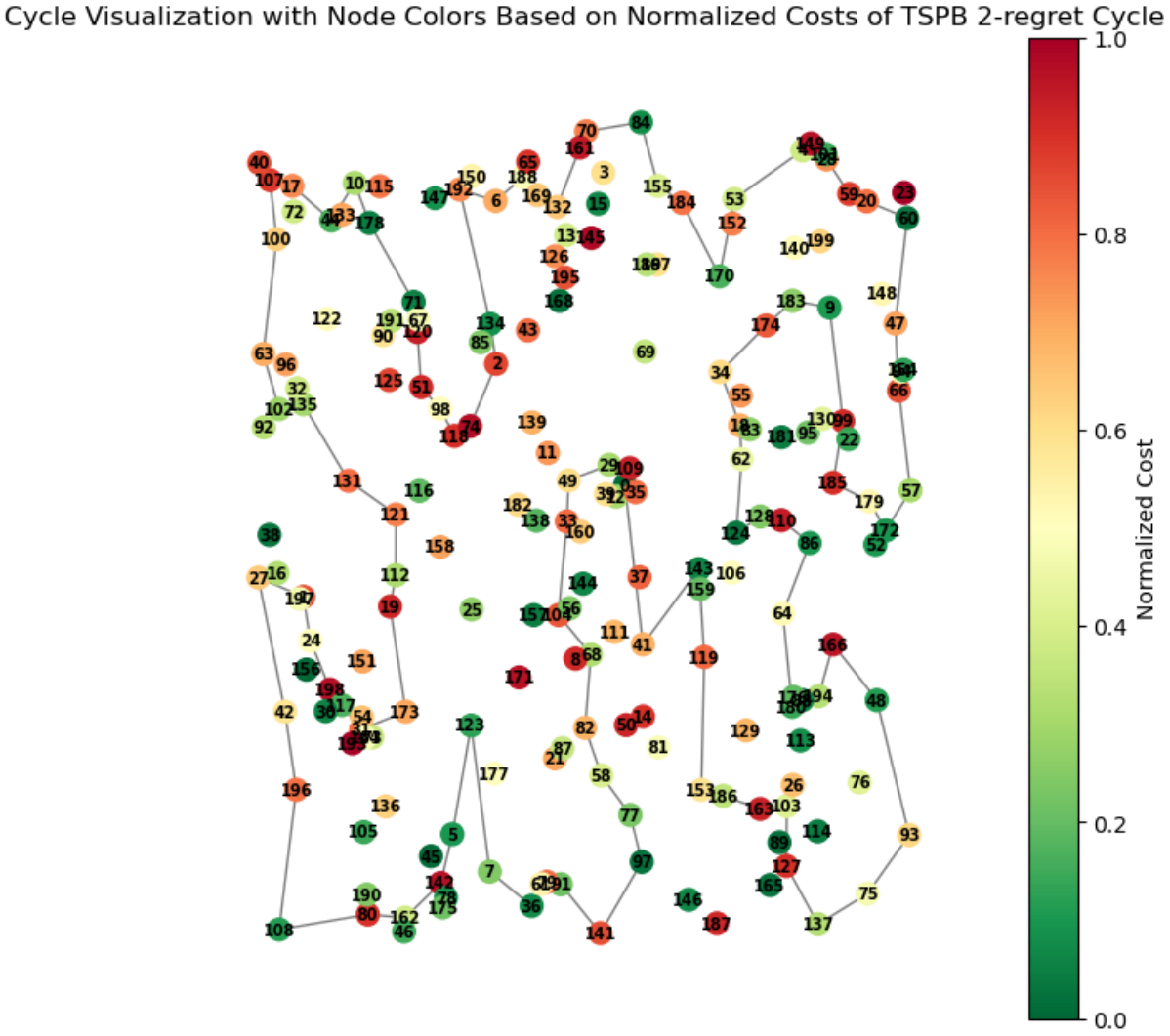
TSPA Random

[121, 145, 11, 115, 113, 120, 178, 117, 62, 77, 101, 15, 168, 89, 199, 183, 31, 98, 109, 66, 84, 108, 140, 48, 137, 194, 59, 47, 73, 93, 65, 50, 42, 119, 116, 135, 136, 189, 110, 80, 21, 139, 171, 19, 51, 173, 195, 55, 44, 150, 85, 97, 106, 196, 166, 43, 160, 190, 181, 114, 118, 54, 170, 57, 39, 53, 9, 123, 127, 128, 187, 63, 163, 141, 102, 144, 86, 156, 30, 151, 24, 133, 36, 126, 177, 159, 191, 176, 134, 35, 192, 184, 12, 92, 23, 158, 188, 10, 70, 95]



TSPB 2-regret Cycle

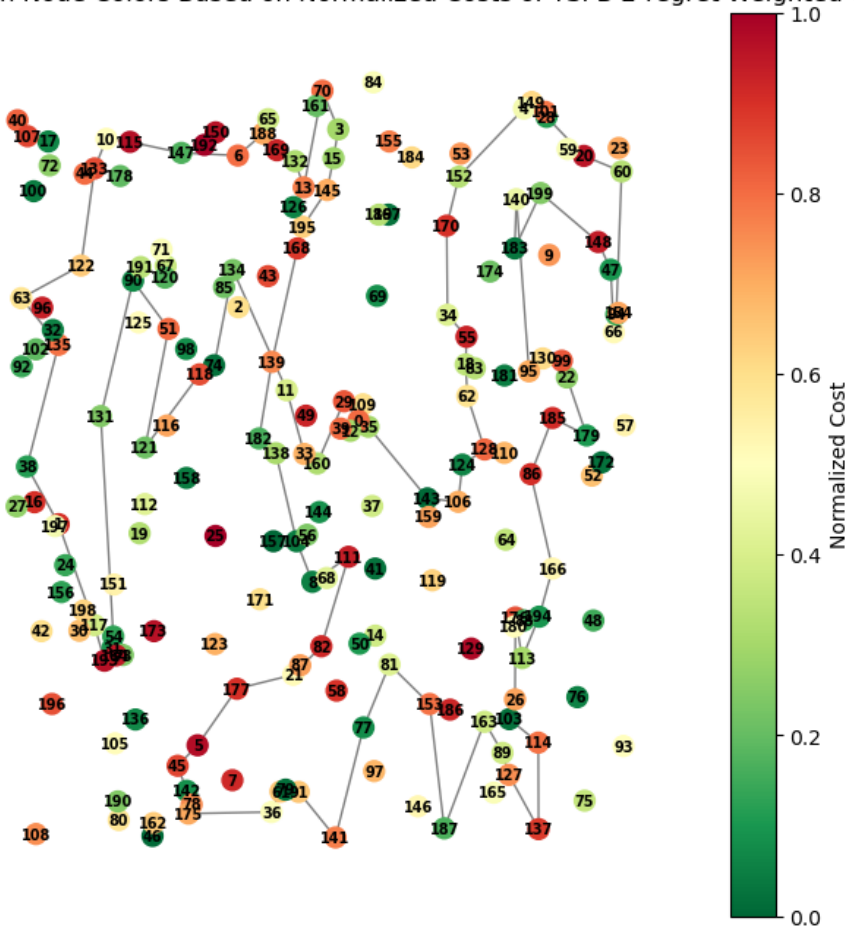
```
[124, 128, 110, 86, 64, 180, 194, 166, 48, 93, 75, 137, 127, 103, 163, 186, 153, 119
, 143, 41, 0, 29, 49, 33, 104, 68, 82, 58, 77, 97, 141, 91, 79, 36, 7, 123, 5, 142,
162, 80, 108, 196, 42, 27, 1, 24, 198, 117, 31, 173, 19, 112, 121, 131, 135, 102, 63
, 100, 107, 17, 44, 10, 178, 71, 120, 51, 98, 118, 74, 2, 134, 192, 6, 188, 169, 132
, 70, 84, 155, 184, 170, 53, 4, 28, 59, 20, 60, 47, 66, 57, 172, 179, 185, 99, 9, 18
3, 174, 34, 18, 62]
```



TSPB 2-regret Weighted Cycle

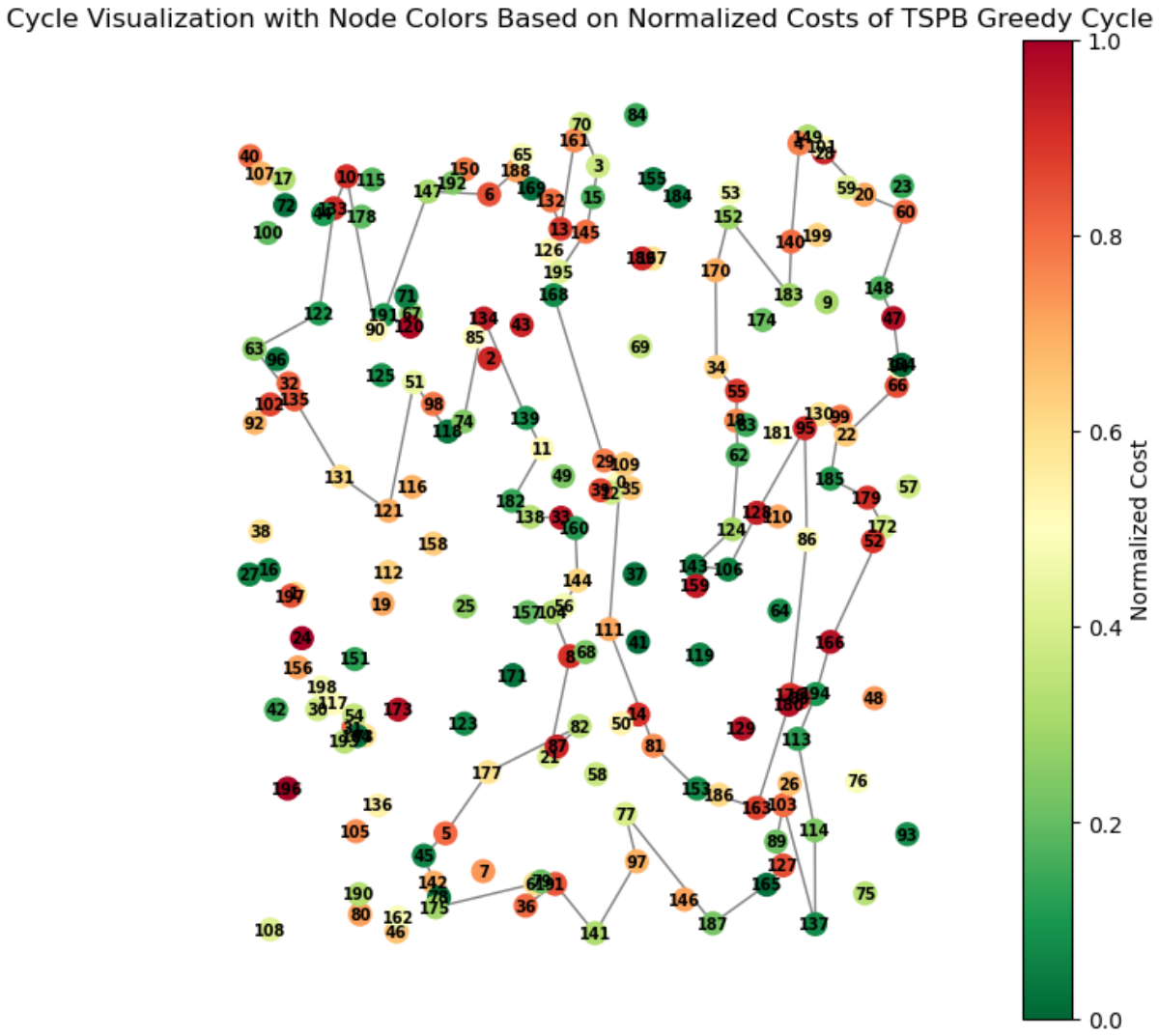
[183, 199, 148, 47, 66, 94, 60, 20, 28, 149, 4, 152, 170, 34, 55, 18, 62, 128, 124, 106, 143, 35, 109, 0, 29, 160, 33, 11, 134, 74, 118, 121, 51, 90, 131, 54, 31, 193, 117, 1, 38, 135, 63, 122, 133, 10, 115, 147, 6, 188, 169, 132, 13, 70, 3, 15, 145, 195, 168, 139, 182, 138, 104, 8, 111, 82, 21, 177, 5, 45, 142, 78, 175, 36, 61, 91, 141, 77, 81, 153, 187, 163, 89, 127, 137, 114, 103, 26, 176, 113, 194, 166, 86, 185, 179, 22, 99, 130, 95, 140]

Cycle Visualization with Node Colors Based on Normalized Costs of TSPB 2-regret Weighted Cycle

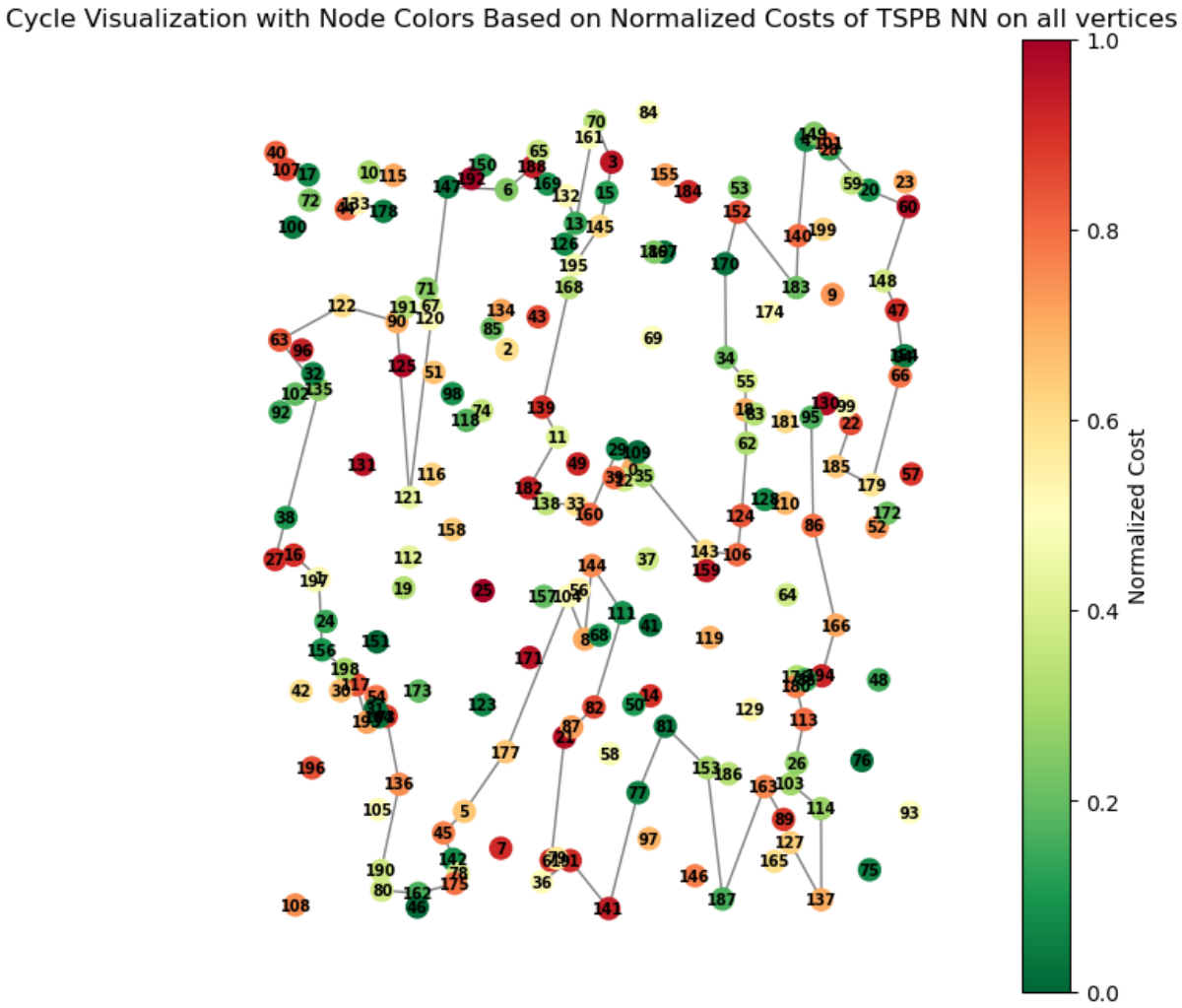


TSPB Greedy Cycle

[118, 74, 134, 139, 11, 182, 138, 33, 160, 144, 56, 104, 8, 21, 87, 82, 177, 5, 45, 142, 78, 175, 61, 36, 91, 141, 97, 77, 187, 165, 127, 89, 103, 137, 114, 113, 194, 166, 172, 179, 185, 99, 130, 22, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 124, 143, 106, 128, 95, 86, 176, 180, 163, 153, 81, 111, 0, 35, 109, 29, 168, 195, 145, 15, 3, 70, 161, 13, 132, 169, 188, 6, 147, 191, 90, 10, 133, 122, 63, 135, 131, 121, 51]

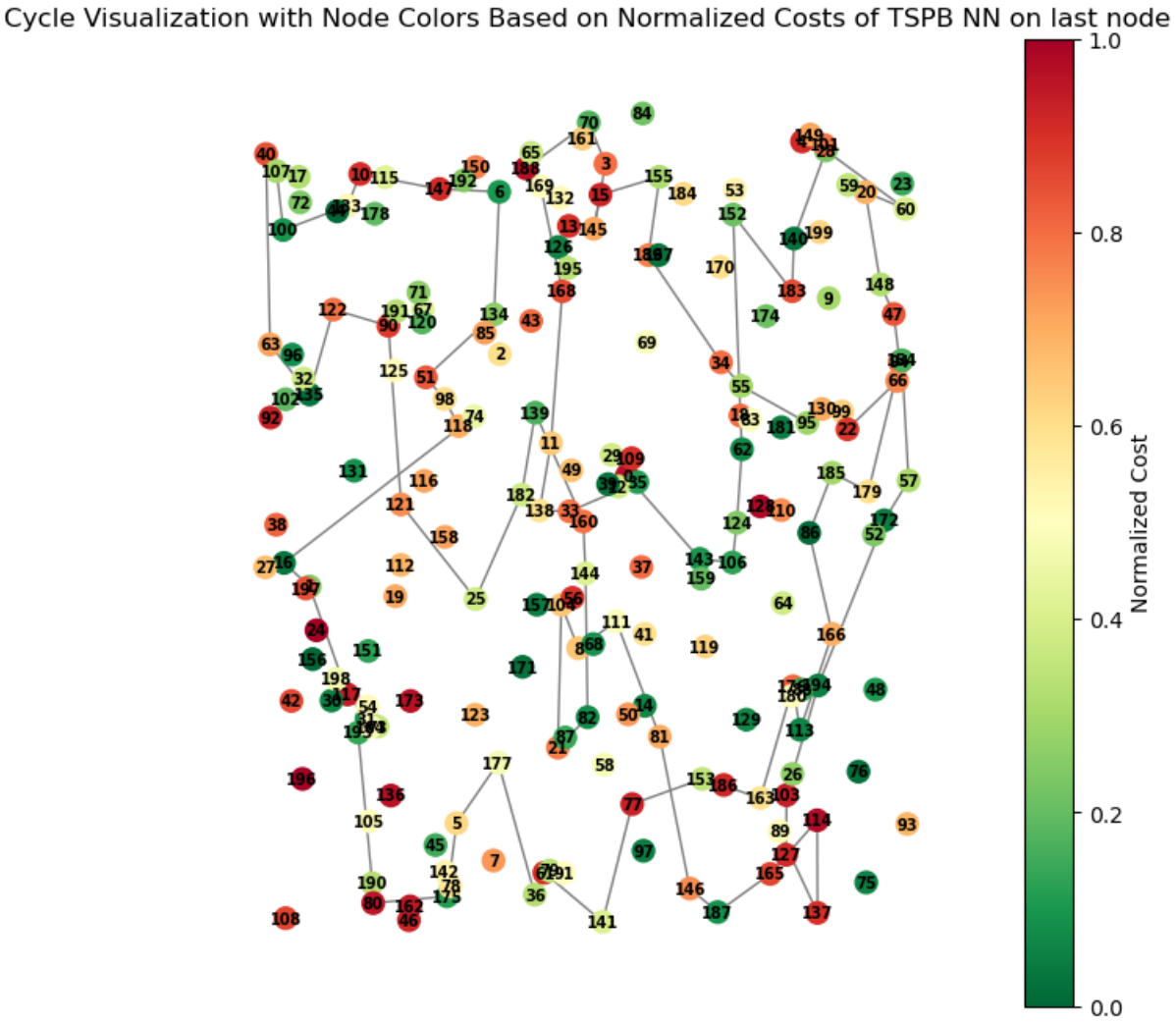


TSPB NN on all vertices
[147, 6, 188, 169, 132, 13, 70, 3, 15, 145, 195, 168, 139, 11, 182, 138, 33, 160, 29
, 0, 109, 35, 143, 106, 124, 62, 18, 55, 34, 170, 152, 183, 140, 4, 149, 28, 20, 60,
148, 47, 94, 66, 179, 185, 22, 99, 130, 95, 86, 166, 194, 176, 180, 113, 103, 114, 1
37, 127, 89, 163, 187, 153, 81, 77, 141, 91, 36, 61, 21, 82, 111, 144, 8, 104, 177,
5, 45, 142, 78, 175, 162, 80, 190, 136, 73, 54, 31, 193, 117, 198, 156, 1, 16, 27, 3
8, 135, 63, 122, 90, 121]



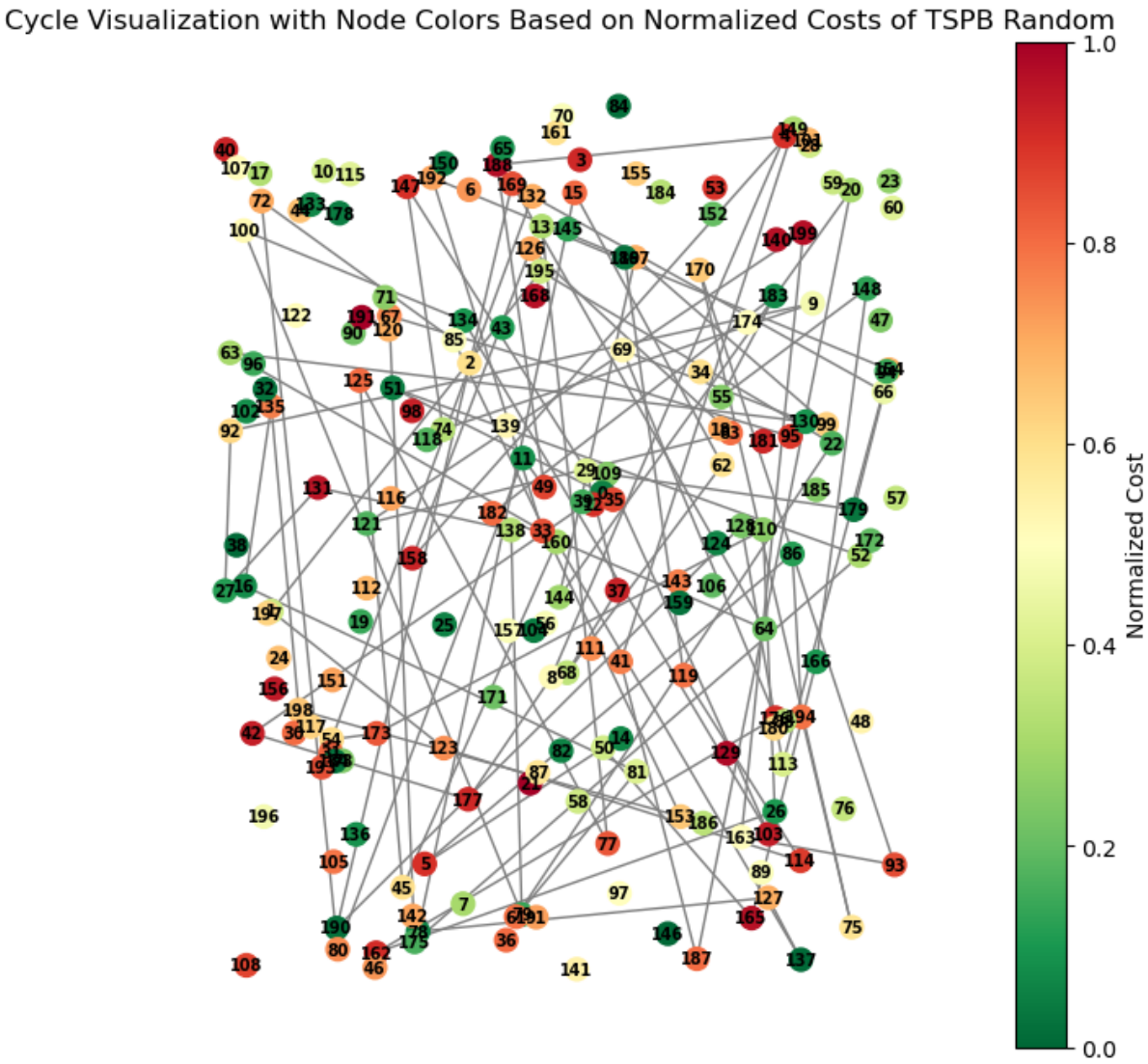
TSPB NN on last node

[16, 1, 117, 31, 54, 193, 190, 80, 175, 5, 177, 36, 61, 141, 77, 153, 163, 176, 113, 166, 86, 185, 179, 94, 47, 148, 20, 60, 28, 140, 183, 152, 18, 62, 124, 106, 143, 0, 29, 109, 35, 33, 138, 11, 168, 169, 188, 70, 3, 145, 15, 155, 189, 34, 55, 95, 130, 99, 22, 66, 154, 57, 172, 194, 103, 127, 89, 137, 114, 165, 187, 146, 81, 111, 8, 104, 21, 82, 144, 160, 139, 182, 25, 121, 90, 122, 135, 63, 40, 107, 100, 133, 10, 147, 6, 134, 51, 98, 118, 74]



TSPB Random

[179, 109, 103, 134, 100, 79, 138, 131, 27, 92, 9, 74, 147, 29, 37, 68, 62, 15, 78, 127, 137, 160, 45, 125, 77, 13, 154, 113, 128, 190, 169, 64, 182, 183, 55, 148, 166, 165, 50, 5, 86, 194, 75, 170, 93, 163, 199, 121, 83, 126, 158, 149, 41, 187, 20, 144, 167, 99, 95, 67, 175, 52, 51, 0, 42, 177, 8, 4, 188, 49, 11, 80, 72, 85, 153, 198, 135, 16, 81, 61, 22, 195, 1, 123, 114, 119, 189, 192, 33, 63, 130, 69, 176, 162, 26, 110, 173, 54, 132, 66]



Additional Information

Solution checker

We have checked all of the best solutions via the solution checker provided.

Source code link

The source code is available in a repository here under the Lab2 folder.

Conclusions

The algorithms explored in this assignment reveal important insights into solving the Traveling Salesman Problem (TSP). The basic 2-regret greedy algorithm showed limited effectiveness on its own, but when combined with other strategies—specifically the greedy cycle method enhanced by a weighted 2-regret approach—it produced results that

surpassed all previously implemented methods in TSPA instance and achieved relatively good results in TSPB. This indicates that integrating different heuristics can lead to significantly better solutions.

Moreover, the weighted 2-regret heuristic improved the performance of the greedy cycle method across various metrics. While it outperformed other methods in the TSPA problem instance, it did not manage to exceed the results of the nearest neighbor (NN) on all vertices method in the TSPB instance. This suggests that the effectiveness of this enhancement can vary depending on the specific characteristics of the problem at hand. The NN on all vertices method remains a viable option, especially in situations with high computational demands and tight time constraints, as it consistently provides decent results in a shorter time frame compared to the greedy cycle with 2-regret heuristic, which typically takes about twice as long to execute.

Examining the generated cycles indicates that these new methods sometimes produce paths with crossing edges. This finding suggests that, despite the weighted 2-regret approach yielding the best results in TSPA, the resulting cycles may not be optimal in terms of tour length. Therefore, there is still room for improvement.

Authors

- Kajetan Sulwiński 151954
- Mikołaj Marmurowicz 151956