Evolutionary Computation - Assignment 3 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 a new method with multiple variations to solve this problem created in C++.

- **Local Search**

- **Input:**
  - `cycle` : An array of a previously generated cycle
  - `costDistanceInfo` : A symmetric matrix of distances and costs between nodes
  - `swapEdges` : A boolean instructing what intra method to use
  - `isGreedy` : A boolean instructing whether the local search is greedy or steepest
  - `repetition` : An integer used as a seed for random shuffling
- **Output:**
  - An array of new `cycle` node IDs
- **Function:**

```
FUNCTION optimizeCycle(cycle, costDistanceInfo, swapEdges,
isGreedy, seed)
    IF swapEdges THEN
        intraMoves = generateAllIntraEdgeMoves(ret,
costDistanceInfo)
    ELSE
        intraMoves = generateAllIntraNodeMoves(ret,
costDistanceInfo)
    REPEAT
        possibleMoves = COMBINE generateAllInterMoves(ret,
costDistanceInfo) AND intraMoves
        SHUFFLE possibleMoves using seed (std::shuffle function)
        FOR each move IN possibleMoves DO
            increase = move.calculateFunctionDelta(moveType, ret)
// NEGATIVE VALUE = IMPROVEMENT
            IF increase < bestIncrease THEN
                bestIncrease = increase
                bestMove = move
                IF isGreedy THEN BREAK
        IF bestMove IS NOT NULL THEN bestMove.performMove(ret)
    UNTIL NO improvement
    RETURN best_cycle_found

FUNCTION calculateFunctionDelta(moveType, currentCycle)
    IF moveType == "IntraNodeChangeNode" THEN
        currentValue = getDistance(node1, node2) +
getDistance(node2, node3)
        newValue = getDistance(node1, node3) + getDistance(node3,
node2)
    ELSE IF moveType == "IntraNodeChangeEdge" THEN
        currentValue = getDistance(node1Start, node1End) +
getDistance(node2Start, node2End)
        newValue = getDistance(node1End, node2End) +
getDistance(node1Start, node2Start)
    ELSE IF moveType == "InterNode" THEN
        currValue = getNodeCost(nodeInCycle) +
```

```
getDistance(nodeInCycle, lN) + getDistance(nodeInCycle, rN)
        newValue = getNodeCost(nodeToAddId) +
getDistance(nodeToAddId, lN) + getDistance(nodeToAddId, rN)
    RETURN newValue - currentValue
```

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 = ['.\\TSPAGreedyIntraSwapEdgeskRegretWeightedStart.csv', '.\\TSPAGreedy
              '.\\TSPAGreedyIntraSwapNodeskRegretWeightedStart.csv', '.\\TSPAGreedy
              '.\\TSPASteepestIntraSwapEdgeskRegretWeightedStart.csv', '.\\TSPAStee
              '.\\TSPASteepestIntraSwapNodeskRegretWeightedStart.csv', '.\\TSPAStee
              '..\\Lab2\\TSPAKRegret.csv','..\\Lab2\\TSPAKRegretGreedyCombination.c
              '..\\Lab1\\TSPANNLast.csv', '..\\Lab1\\TSPARandom.csv',
              '.\\TSPBGreedyIntraSwapEdgeskRegretWeightedStart.csv', '.\\TSPBGreedy
              '.\\TSPBGreedyIntraSwapNodeskRegretWeightedStart.csv', '.\\TSPBGreedy
              '.\\TSPBSteepestIntraSwapEdgeskRegretWeightedStart.csv', '.\\TSPBStee
              '.\\TSPBSteepestIntraSwapNodeskRegretWeightedStart.csv', '.\\TSPBStee
              '..\\Lab2\\TSPBKRegret.csv','..\\Lab2\\TSPBKRegretGreedyCombination.c
              '..\\Lab1\\TSPBNNLast.csv', '..\\Lab1\\TSPBRandom.csv']
methods = ['Greedy LS (Edges) on 2-Regret Weighted', 'Greedy LS (Edges) on Random',
           'Steepest LS (Edges) on 2-Regret Weighted', 'Steepest LS (Edges) on Rand
           '2-regret Cycle', '2-regret Weighted Cycle', 'Greedy Cycle', 'NN on all
results = []
best_solutions = []
counter = 0
for file_path, method in zip(file_paths, methods * 2):
    df = pd.read_csv(file_path)
    costs = df.iloc[:, -1]
    minimum = costs.min()
    maximum = costs.max()
    mean = round(costs.mean(), 2)
    if counter < len(methods):
        results.append((method, 'TSPA', f"{mean} ({minimum} - {maximum})"))
    else:
        results.append((method, 'TSPB', f"{mean} ({minimum} - {maximum})"))
    if '..' not in file_path:
        min_sol = df.loc[costs.idxmin()][:-1].to_list()
        best_solutions.append(min_sol)
    counter += 1
print(len(best_solutions))
result_df = pd.DataFrame(results, columns=['Method', 'Column', 'Value'])
result_df = result_df.pivot(index='Method', columns='Column', values='Value')
result_df.columns.name = None
result_df
```

Out[3]:

Method	TSPA	TSPB
2-regret Cycle	115570.82 (105692 - 126951)	72746.68 (67809 - 78406)
2-regret Weighted Cycle	72134.84 (71108 - 73395)	50913.93 (47144 - 55700)
Greedy Cycle	73036.23 (71237 - 75002)	51852.88 (48898 - 58531)
Greedy LS (Edges) on 2-Regret Weighted	71509.42 (70571 - 72485)	50033.92 (45855 - 54814)
Greedy LS (Edges) on Random	73806.03 (70564 - 77419)	48432.73 (46113 - 52308)
Greedy LS (Nodes) on 2-Regret Weighted	71629.31 (70687 - 72707)	50211.62 (46328 - 55555)
Greedy LS (Nodes) on Random	85761.41 (78044 - 96412)	60965.73 (53878 - 69110)
NN on all vertices	73293.75 (71227 - 76036)	47444.68 (44377 - 53019)
NN on last node	85110.16 (83182 - 89433)	54385.49 (52319 - 59030)
Random	263481.34 (236302 - 293567)	213568.36 (188701 - 239495)
Steepest LS (Edges) on 2-Regret Weighted	71470.14 (70510 - 72614)	49895.7 (45867 - 54814)
Steepest LS (Edges) on Random	73868.19 (71654 - 78313)	48361.54 (45987 - 50939)
Steepest LS (Nodes) on 2-Regret Weighted	71619.69 (70626 - 72950)	50188.3 (46371 - 55385)
Steepest LS (Nodes) on Random	88258.31 (81036 - 96035)	62825.07 (56373 - 69716)

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

In [4]:

```
methods_for_times = [['Greedy LS (Edges) on Random', 'Greedy LS (Edges) on 2-Regret
                      'Steepest LS (Edges) on Random', 'Steepest LS (Edges) on 2-Re
                     ['2-regret Cycle', '2-regret Weighted Cycle'], ['Random', 'NN
times_files = ['.\\times.csv','..\\Lab2\\times.csv', '..\\Lab1\\times.csv']

results_times = []
for method_list, time_file in zip(methods_for_times, times_files):
    df_temp = pd.read_csv(time_file, header=None).iloc[:, :-1]
    counter = 0
    for column, method in zip(df_temp.columns, method_list * 2):
        min_value = df_temp[column].min()
        max_value = df_temp[column].max()
        avg_value = df_temp[column].mean()
        if counter < len(method_list):
            results_times.append((method, 'TSPA', f"{round(avg_value, 4)} ({round(m
        else:
            results_times.append((method, 'TSPB', f"{round(avg_value, 4)} ({round(m
```

```
        counter += 1

times_df = pd.DataFrame(results_times, columns=['Method', 'Column', 'Value'])
times_df = times_df.pivot(index='Method', columns='Column', values='Value')
times_df.columns.name = None
times_df
```

Out[4]:

Method	TSPA	TSPB
2-regret Cycle	27.6908 (24.8662 - 38.6567)	27.3929 (24.5655 - 35.2519)
2-regret Weighted Cycle	27.5682 (24.7153 - 35.1762)	27.4779 (24.4544 - 32.7938)
Greedy Cycle	12.5365 (11.7672 - 14.9002)	12.1852 (11.5659 - 14.7737)
Greedy LS (Edges) on 2-Regret Weighted	9.7991 (3.5297 - 23.6288)	13.8843 (4.7005 - 28.4957)
Greedy LS (Edges) on Random	383.6733 (317.032 - 525.722)	375.0126 (325.498 - 426.852)
Greedy LS (Nodes) on 2-Regret Weighted	7.7295 (2.8208 - 19.0481)	12.5055 (4.9377 - 25.9375)
Greedy LS (Nodes) on Random	398.6882 (335.15 - 556.069)	378.8708 (304.659 - 446.093)
NN on all vertices	11.9891 (11.2653 - 13.4535)	11.6631 (11.2167 - 14.0679)
NN on last node	0.365 (0.3222 - 0.8055)	0.3538 (0.3182 - 0.4824)
Random	0.0075 (0.0063 - 0.0411)	0.0073 (0.0062 - 0.032)
Steepest LS (Edges) on 2-Regret Weighted	10.6584 (4.4004 - 18.6813)	18.1456 (5.9029 - 37.1348)
Steepest LS (Edges) on Random	198.1662 (167.179 - 275.628)	190.8946 (166.271 - 224.632)
Steepest LS (Nodes) on 2-Regret Weighted	9.2906 (3.2524 - 18.6304)	15.1823 (6.2209 - 22.8452)
Steepest LS (Nodes) on Random	235.2782 (191.605 - 299.675)	233.5712 (190.613 - 303.508)

Visualization of the best path for each method

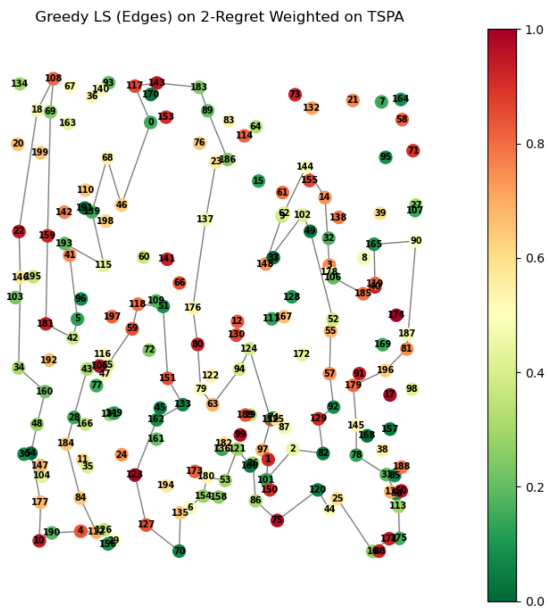
Additionally, a list of node indices is presented.

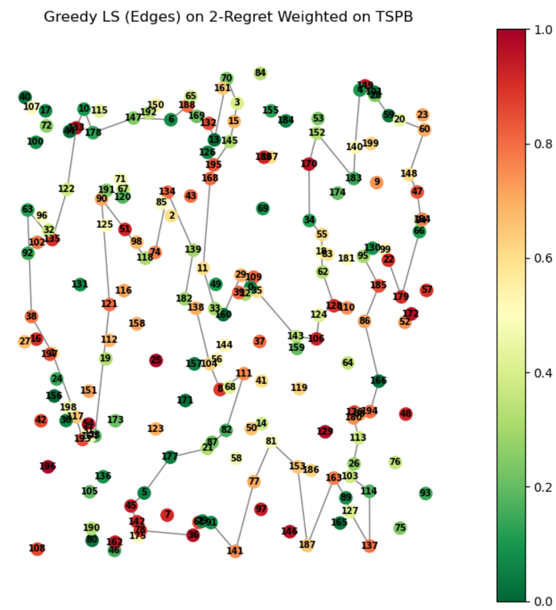
In [5]:

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

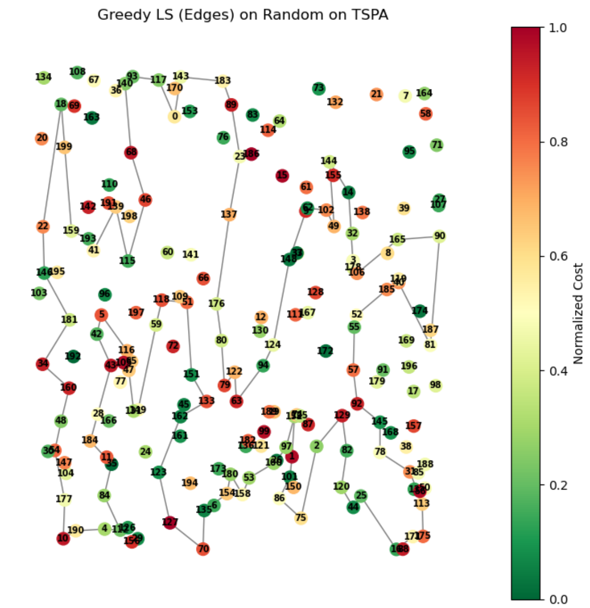
```
for count, method in enumerate(methods):
    if count == len(best_solutions) // 2:
        break
    print(method)
    print('TSPA')
    print(best_solutions[count])
    print(count, count + len(best_solutions)//2)
    print('TSPB')
    print(best_solutions[count + len(best_solutions)//2])

    fig, axs = plt.subplots(1, 2, figsize=(14, 7))

    for count, sol in enumerate([best_solutions[count], best_solutions[count + len(
        if count == 0:
            df_temp = dfTSPA
            ax = axs[0]
            instance = 'TSPA'
        else:
            df_temp = dfTSPB
            ax = axs[1]
            instance = 'TSPB'

        G = nx.Graph()
        positions = {}

        for idx in sol:
            G.add_node(idx, size=df_temp.loc[idx, 'Normalized_Cost'])
            positions[idx] = (df_temp.loc[idx, 'X'], df_temp.loc[idx, 'Y'])

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

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


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

        nx.draw(G, pos=positions, with_labels=True, node_color=node_colors, node_si
            font_size=7, edge_color='gray', linewidths=1, font_weight='bold', ax=ax

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

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

        ax.set_title(f" {method} on {instance}")

    plt.tight_layout()
    plt.show()
```

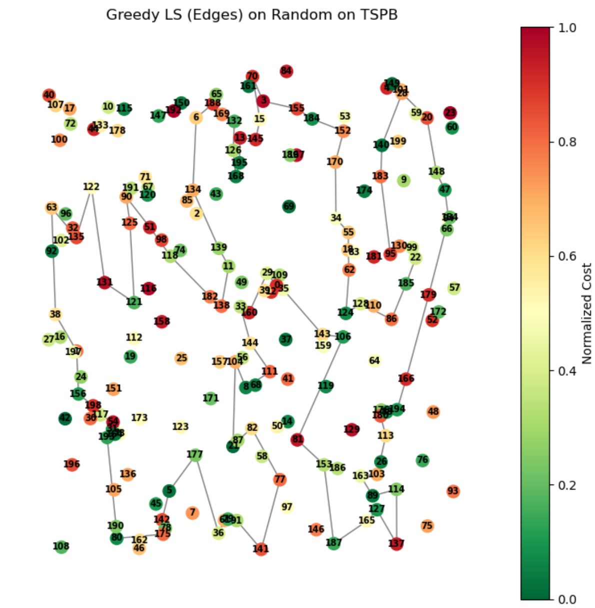
Greedy LS (Edges) on 2-Regret Weighted
TSPA
[183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 124, 152, 97, 1, 101, 2, 82, 129, 92, 5
7, 55, 52, 49, 102, 148, 9, 62, 144, 14, 3, 178, 106, 185, 40, 165, 90, 81, 196, 179
, 145, 78, 31, 56, 113, 175, 171, 16, 25, 44, 120, 75, 86, 26, 100, 121, 53, 180, 15
4, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 65, 116, 43, 184, 84, 112, 4, 190,
10, 177, 54, 48, 160, 34, 146, 22, 18, 108, 69, 159, 181, 42, 5, 41, 193, 115, 139,
68, 46, 0, 117, 143]
0 8
TSPB
[59, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 128, 124, 106, 143, 35, 109, 0,
29, 160, 33, 11, 168, 195, 145, 15, 3, 70, 13, 132, 169, 188, 6, 147, 178, 10, 133,
122, 135, 63, 38, 1, 117, 193, 31, 54, 73, 121, 90, 51, 118, 74, 134, 139, 182, 138,
104, 8, 111, 82, 21, 177, 5, 45, 142, 175, 78, 36, 61, 91, 141, 77, 81, 153, 187, 16
3, 89, 127, 137, 114, 103, 113, 180, 176, 194, 166, 86, 185, 95, 130, 99, 22, 179, 6
6, 94, 47, 148, 60, 20]





Greedy LS (Edges) on Random
TSPA
[140, 93, 117, 0, 143, 183, 89, 23, 137, 176, 80, 79, 122, 63, 94, 124, 148, 9, 62,
102, 49, 144, 14, 178, 106, 165, 90, 81, 40, 185, 52, 55, 57, 92, 145, 78, 31, 113,
175, 171, 16, 25, 44, 120, 82, 129, 2, 75, 86, 101, 1, 152, 97, 26, 100, 53, 158, 18
0, 154, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 149, 131, 47, 65, 116, 5, 42,
43, 184, 35, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 181, 146, 22, 18, 159, 193,
41, 139, 115, 46, 68]
1 9
TSPB
[89, 114, 137, 127, 165, 187, 153, 81, 106, 143, 35, 109, 0, 29, 160, 33, 144, 111,
8, 104, 21, 87, 82, 77, 141, 91, 61, 36, 177, 5, 142, 78, 175, 80, 190, 193, 31, 54,
117, 198, 156, 1, 38, 63, 135, 122, 131, 121, 125, 90, 51, 118, 182, 138, 11, 139, 1
34, 6, 188, 169, 132, 126, 168, 195, 13, 145, 15, 70, 3, 155, 184, 152, 170, 34, 55,
18, 62, 124, 128, 86, 185, 22, 99, 130, 95, 183, 140, 28, 20, 148, 47, 94, 179, 166,
194, 176, 180, 113, 103, 163]





Greedy LS (Nodes) on 2-Regret Weighted

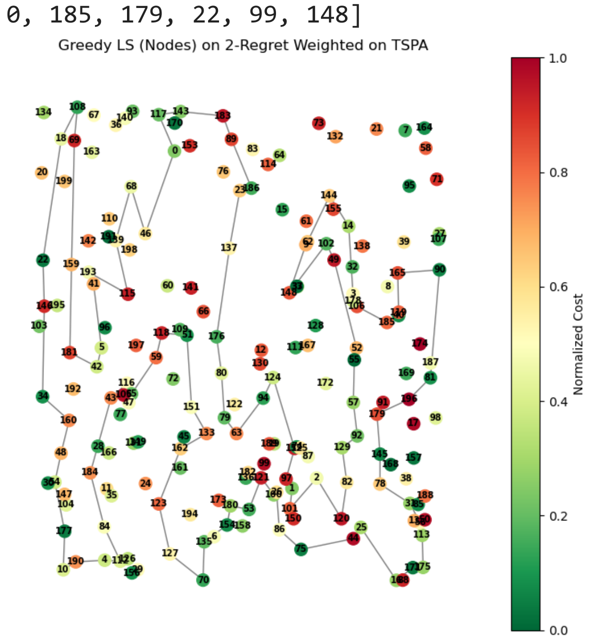
TSPA

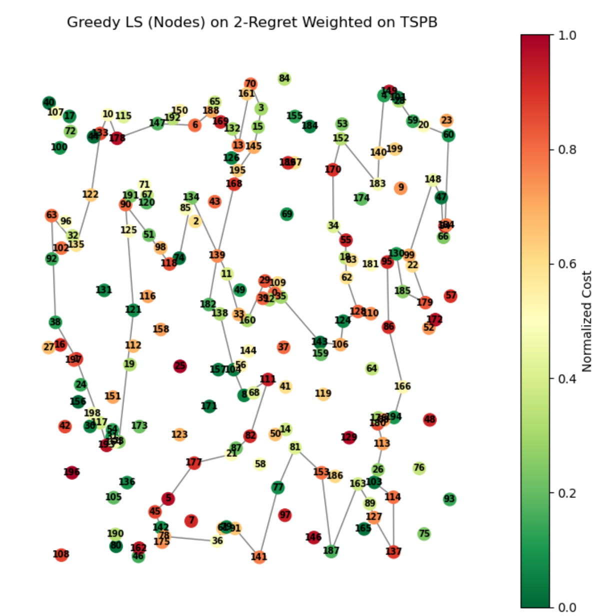
[143, 117, 0, 46, 68, 139, 115, 193, 41, 5, 42, 181, 159, 69, 108, 18, 22, 146, 34, 160, 48, 54, 177, 10, 190, 4, 112, 84, 184, 43, 116, 65, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 154, 180, 53, 121, 100, 26, 86, 75, 44, 25, 16, 171, 175, 113, 56, 31, 78, 145, 179, 196, 81, 90, 165, 40, 185, 106, 178, 3, 14, 144, 62, 9, 148, 102, 49, 52, 55, 57, 92, 129, 82, 120, 2, 101, 1, 97, 152, 124, 94, 63, 79, 80, 176, 137, 23, 186, 89, 183]

2 10

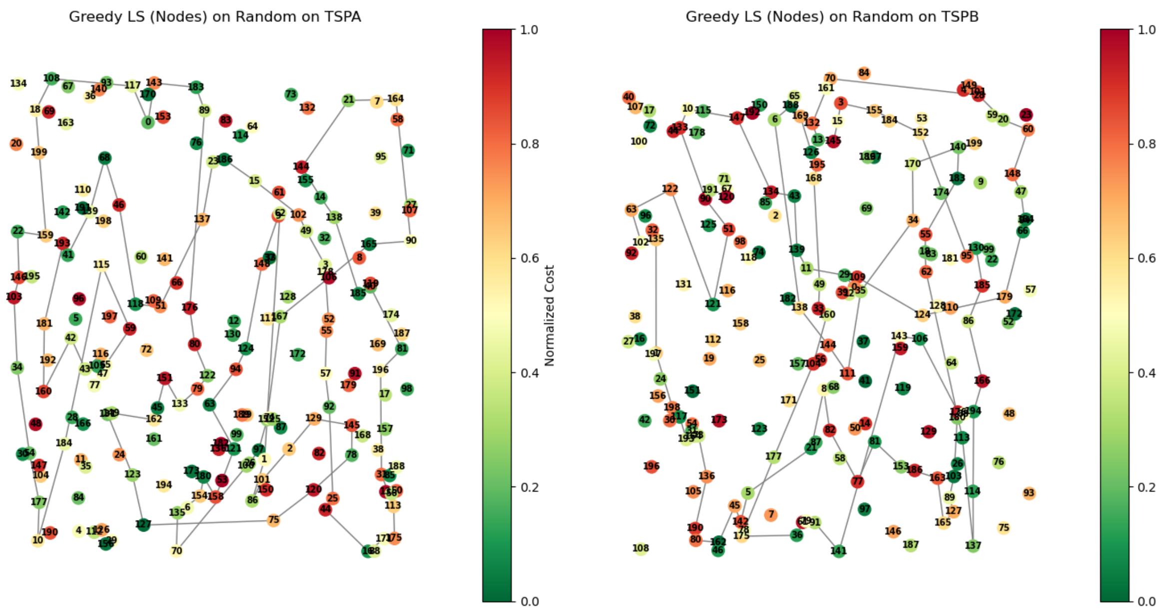
TSPB

[47, 66, 94, 60, 20, 59, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 128, 124, 106, 143, 35, 109, 0, 29, 160, 33, 11, 134, 74, 118, 51, 90, 121, 73, 54, 31, 193, 117, 1, 38, 63, 135, 122, 133, 10, 178, 147, 6, 188, 169, 132, 13, 70, 3, 15, 145, 195, 168, 139, 182, 138, 104, 8, 111, 82, 21, 177, 5, 45, 142, 175, 78, 36, 61, 91, 141, 77, 81, 153, 187, 163, 89, 127, 137, 114, 103, 113, 180, 176, 194, 166, 86, 95, 130, 185, 179, 22, 99, 148]

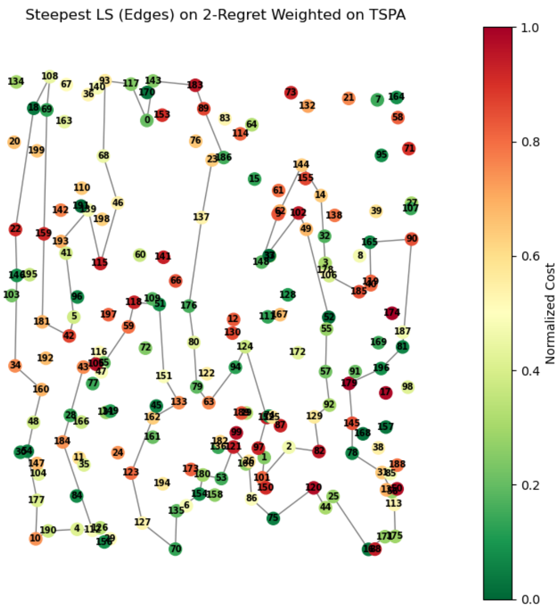


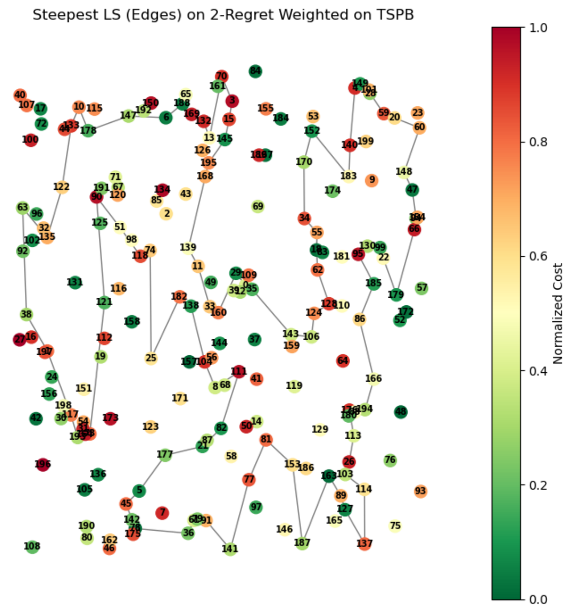


Greedy LS (Nodes) on Random
TSPA
[34, 54, 177, 10, 184, 115, 59, 65, 116, 43, 42, 160, 181, 193, 41, 139, 68, 46, 118
, 51, 137, 23, 186, 15, 102, 49, 3, 178, 52, 55, 57, 92, 25, 44, 16, 171, 175, 113,
31, 196, 81, 40, 185, 138, 14, 144, 21, 164, 90, 165, 106, 167, 152, 97, 26, 100, 70
, 135, 154, 180, 158, 53, 121, 63, 94, 124, 148, 9, 62, 1, 86, 101, 2, 129, 145, 78,
120, 75, 127, 123, 131, 149, 162, 151, 133, 79, 122, 80, 176, 89, 183, 143, 0, 117,
108, 18, 159, 22, 146, 103]
3 11
TSPB
[103, 180, 176, 106, 143, 141, 91, 61, 36, 175, 78, 177, 104, 160, 33, 49, 168, 195,
126, 169, 188, 6, 138, 144, 111, 35, 34, 170, 140, 183, 55, 18, 62, 113, 137, 114, 1
94, 166, 86, 185, 130, 95, 152, 155, 3, 15, 145, 13, 132, 70, 28, 20, 60, 148, 47, 9
4, 179, 124, 109, 0, 29, 11, 139, 43, 134, 147, 10, 133, 90, 51, 121, 122, 63, 102,
135, 1, 198, 117, 193, 54, 31, 164, 73, 136, 190, 80, 162, 45, 142, 5, 21, 8, 82, 77
, 81, 153, 163, 165, 127, 89]

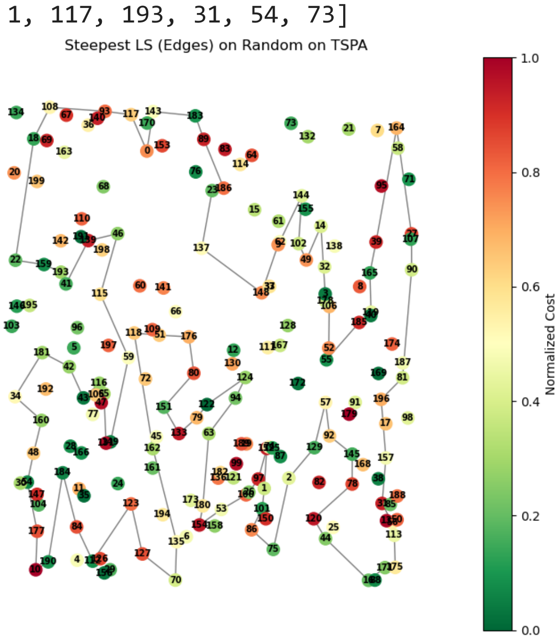


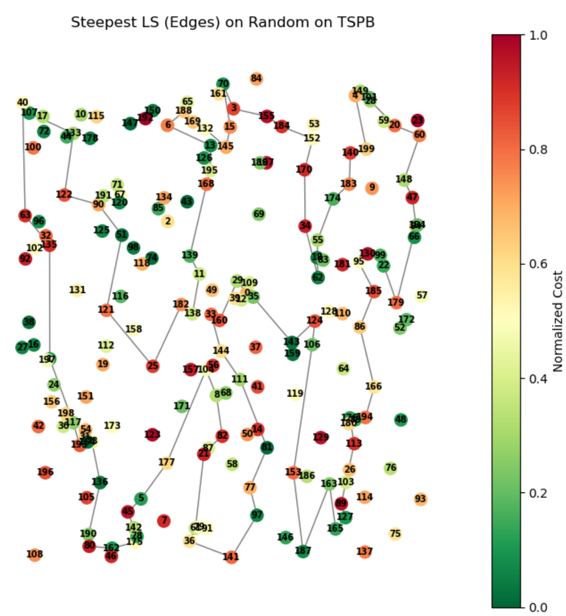
Steepest LS (Edges) on 2-Regret Weighted
TSPA
[143, 0, 117, 93, 68, 46, 115, 139, 193, 41, 5, 42, 181, 159, 69, 108, 18, 22, 146,
34, 160, 48, 54, 177, 10, 190, 4, 112, 184, 43, 116, 65, 59, 118, 51, 151, 133, 162,
123, 127, 70, 135, 154, 180, 53, 121, 100, 26, 86, 75, 120, 44, 25, 16, 171, 175, 11
3, 56, 31, 78, 145, 179, 196, 81, 90, 165, 40, 185, 106, 178, 3, 14, 144, 62, 9, 148
, 102, 49, 52, 55, 57, 92, 129, 82, 2, 101, 1, 97, 152, 124, 94, 63, 79, 80, 176, 13
7, 23, 186, 89, 183]
4 12
TSPB
[47, 94, 66, 179, 22, 99, 130, 95, 185, 86, 166, 194, 176, 113, 26, 103, 114, 137, 1
27, 89, 163, 187, 153, 81, 77, 141, 91, 61, 36, 78, 175, 142, 45, 5, 177, 21, 82, 11
1, 8, 104, 138, 182, 25, 74, 118, 51, 90, 121, 73, 54, 31, 193, 117, 1, 38, 63, 135,
122, 133, 10, 178, 147, 6, 188, 169, 132, 13, 70, 3, 15, 145, 195, 168, 139, 11, 33,
160, 29, 0, 109, 35, 143, 106, 124, 128, 62, 18, 55, 34, 170, 152, 183, 140, 4, 149,
28, 59, 20, 60, 148]



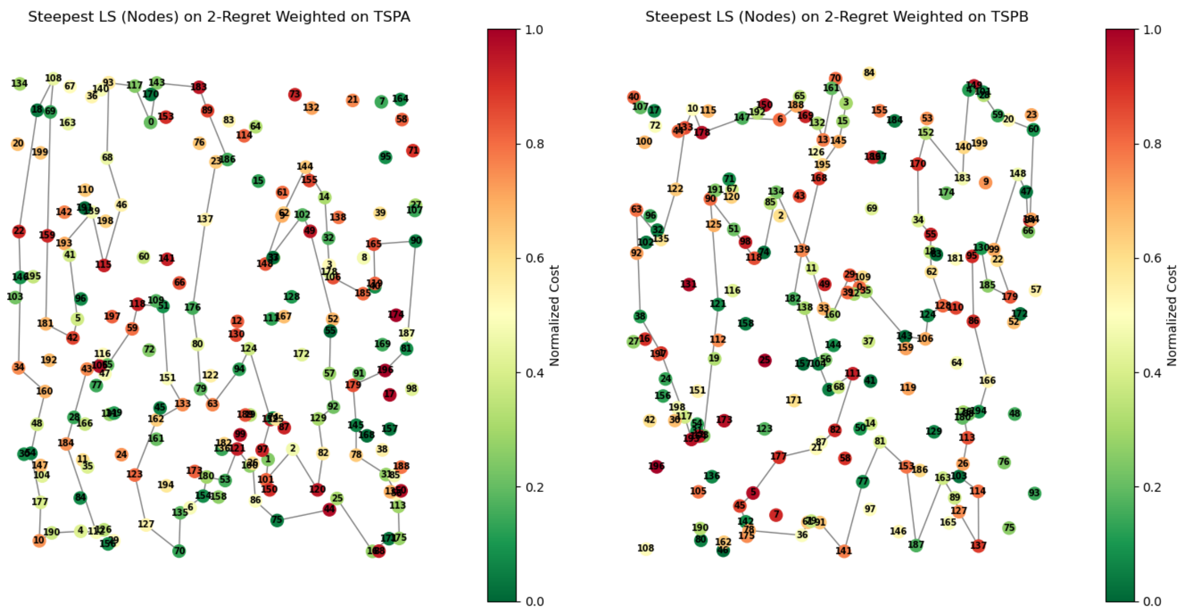



```
Steepest LS (Edges) on Random
TSPA
[44, 16, 171, 175, 113, 56, 31, 157, 196, 81, 90, 27, 164, 39, 165, 119, 40, 185, 55
, 52, 106, 178, 3, 14, 49, 102, 144, 62, 9, 148, 137, 23, 186, 89, 183, 143, 0, 117,
93, 108, 18, 22, 159, 193, 41, 139, 46, 115, 59, 149, 131, 65, 116, 43, 42, 181, 34,
160, 48, 54, 177, 10, 190, 184, 84, 112, 123, 127, 70, 135, 162, 118, 51, 176, 80, 1
51, 133, 79, 122, 124, 94, 63, 180, 154, 53, 100, 26, 97, 152, 1, 101, 86, 75, 2, 12
9, 57, 92, 145, 78, 120]
5 13
TSPB
[136, 190, 80, 162, 175, 78, 142, 45, 5, 177, 104, 8, 82, 87, 21, 61, 36, 141, 97, 7
7, 81, 111, 144, 33, 160, 29, 0, 109, 35, 143, 124, 106, 153, 187, 163, 165, 127, 89
, 103, 113, 176, 194, 166, 86, 185, 95, 130, 99, 179, 94, 47, 148, 60, 20, 28, 149,
4, 199, 140, 183, 174, 55, 18, 62, 34, 170, 152, 184, 155, 3, 70, 15, 145, 132, 169,
188, 6, 13, 195, 168, 139, 11, 138, 182, 25, 121, 51, 90, 122, 133, 107, 40, 63, 135
, 1, 117, 193, 31, 54, 73]
```

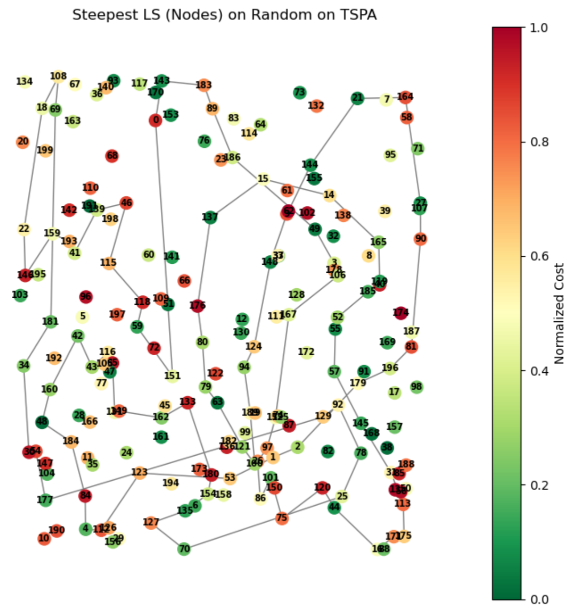


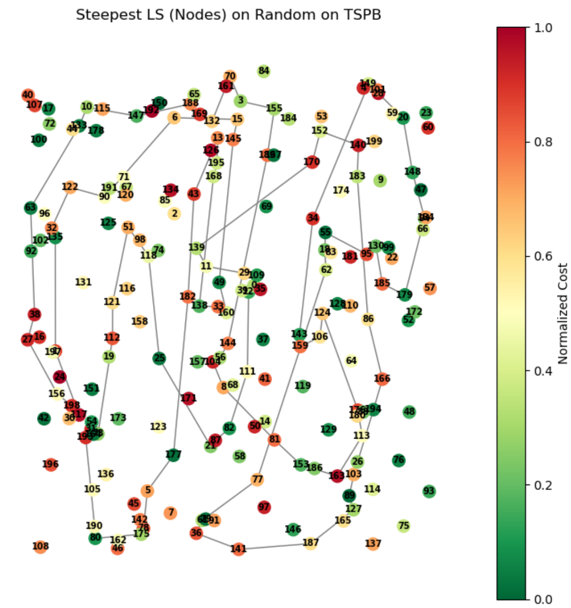


Steepest LS (Nodes) on 2-Regret Weighted
TSPA
[143, 0, 117, 93, 68, 46, 115, 139, 193, 41, 5, 42, 181, 159, 69, 108, 18, 22, 146,
34, 160, 48, 54, 177, 10, 190, 4, 112, 184, 43, 116, 65, 59, 118, 51, 151, 133, 162,
123, 127, 70, 135, 154, 180, 53, 121, 100, 26, 86, 75, 44, 25, 16, 171, 175, 113, 56
, 31, 78, 145, 179, 196, 81, 90, 165, 40, 185, 106, 178, 3, 14, 144, 62, 9, 148, 102
, 49, 52, 55, 57, 92, 129, 82, 120, 2, 101, 1, 97, 152, 124, 94, 63, 79, 80, 176, 13
7, 23, 186, 89, 183]
6 14
TSPB
[47, 66, 94, 60, 20, 59, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 128, 124, 1
06, 143, 35, 109, 0, 29, 160, 33, 11, 134, 74, 118, 51, 90, 121, 73, 54, 31, 193, 11
7, 1, 38, 63, 135, 122, 133, 10, 178, 147, 6, 188, 169, 132, 13, 70, 3, 15, 145, 195
, 168, 139, 182, 138, 104, 8, 111, 82, 21, 177, 5, 45, 142, 175, 78, 36, 61, 91, 141
, 77, 81, 153, 187, 163, 89, 127, 137, 114, 103, 26, 113, 176, 194, 166, 86, 95, 130
, 185, 179, 22, 99, 148]



Steepest LS (Nodes) on Random
TSPA
[34, 181, 69, 108, 18, 22, 146, 159, 193, 41, 139, 46, 115, 118, 59, 151, 51, 0, 143
, 183, 89, 186, 49, 3, 178, 106, 167, 152, 97, 26, 86, 101, 75, 120, 44, 16, 171, 17
5, 113, 31, 145, 57, 55, 52, 185, 40, 165, 14, 15, 137, 176, 80, 79, 63, 121, 100, 1
89, 94, 124, 148, 9, 62, 144, 21, 164, 27, 90, 81, 196, 179, 2, 1, 53, 123, 112, 4,
84, 184, 48, 160, 42, 43, 116, 65, 131, 149, 162, 133, 180, 154, 135, 127, 70, 25, 7
8, 92, 129, 177, 54, 30]
7 15
TSPB
[25, 21, 87, 82, 111, 12, 0, 35, 109, 29, 11, 139, 170, 152, 140, 183, 86, 166, 194,
103, 89, 127, 187, 141, 36, 61, 77, 62, 18, 55, 95, 130, 185, 179, 94, 148, 20, 28,
149, 34, 143, 159, 106, 124, 176, 113, 163, 153, 81, 8, 104, 56, 144, 189, 155, 3, 7
0, 132, 169, 188, 147, 10, 133, 63, 38, 27, 156, 198, 117, 1, 135, 32, 122, 90, 6, 1
5, 145, 49, 160, 33, 138, 168, 195, 13, 43, 182, 177, 5, 78, 175, 80, 190, 193, 31,
54, 73, 112, 121, 51, 118]





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 Lab3 folder.

Conclusions

The local search algorithm was easily able to improve given cycles, no matter the previous method. But the method of creating the previous cycle matters, as the better the previous cycle to faster the Local Search method is going to converge and possibly the better the created solution is going to be.

Greedy algorithm tends to run longer than the steepest variation, as it takes more steps in obtaining the final solution and takes longer time to converge. Steepest version of the algorithm runs faster, but does not necessarily recommend better solutions to the greedy version.

While both intra methods are rather comparable, we notice a slight advantage of the Edge method, not only does it provide better results in all of the cases, it also tends to run much faster on previously generated random solutions. Interestingly, when the starting solution is already good, the Edge method takes longer time to converge compared to the Node method.

Authors

- Kajetan Sulwiński 151954
- Mikołaj Marmurowicz 151956