Evolutionary Computation

Assignment 7

Antoni Lasik 148287
Daniel Jankowski 148257
https://github.com/JankowskiDaniel/evolutionary-computation/tree/AL/assignment7

Problem description

The task involves analyzing three columns of integers, each row corresponding to a single node. The initial two columns designate the x and y coordinates, pinpointing the nodes' locations on a plane, while the third column specifies the cost associated with each node. The objective is to meticulously choose an exact half of the total nodes (in cases where the total node count is an odd number, the count of nodes to be selected is adjusted upward to the nearest whole number) to construct a Hamiltonian cycle, which is essentially a continuous loop that passes through each member of the selected set of nodes. The criterion for this selection is that the aggregate of the complete path's length and the cumulative cost of the chosen nodes should be as low as possible.

To quantify the distances between nodes, we employ the Euclidean distance formula, and the resulting figures are rounded off to the nearest integer in a standard mathematical fashion. Moreover, as part of the distance between nodes, we take into account the cost of the destination node. This ensures that cost has a significant impact on the final results.

In this assignment we implement the Large-Scale Neighborhood Search in two variants. In both tested options, a solution after being destroyed is repaired using Greedy 2-regret cycle with a weighted sum, however, in the second experiment in additional the Steepest Local Search has been applied.

Pseudocode of implemented algorithms

```
calculate_distance_matrix(coords, costs):
        dist_matrix = [][]
        FOR i IN RANGE(len(coords)):
                FOR j IN RANGE(len(coords)):
                        dist_matrix[i][j] = round(sqrt((coords[i].x - coords[j].x)**2 +
                        (coords[i].y - coords[j].y)**2)
        RETURN dist_matrix

objective_function(solution, dist_matrix, costs):
        total_score = 0
        n = len(solution)
        FOR x in range(n):
                        total_score += dist_matrix[solution[x - 1]][solution[x]]
                        total_score += costs[solution[x]]
        RETURN total_score

generate_random_solution(n):
    RETURN random.sample(range(0, n * 2), n)
```

To destroy a solution we’ve implemented a heuristic that remove from a solution the worst subpath of 25 nodes (the worst area w.r.t. to the objective function value). To randomize the mechanism we’ve added additional parameter that controlled potential subpath acceptance (acceptance means to choose such subpath that the one to be removed from the solution). With consecutive worse subpaths, the probability of choosing them increased by 5pp, e.g. the first worse subpath had 5% to be choosen, the next 10% etc. If any was returned by the end of the iteration, the worst possible one has been selected.

```
destroy_solution(solution, dist_matrix, costs)
       num_nodes = len(solution)
       subset_length = 25
       worst_subpath_delta = objective_function(solution[:subset_length], dist_matrix,
                                                                                costs)
       selected_solution = solution[:subset_length]
       start = selected_solution[-1]
       worst_start = start
       previous_subpath_delta = worst_subpath_delta
       acceptance_probability = 0.05

       FOR start_index IN RANGE(1, n):
              start = solution[start_index - 1]
              IF start_index + subset_length > n:
                     new_solution = solution[start_index+subset_length-n:start_index]
              ELSE:
                     new_solution = solution[:start_index]+solution[start_index+
                                                                  subset_length]
              delta = (
               -dist_matrix[start][solution[start_index]]
               +dist_matrix[solution[(start_index+subset_length-2)%n]]
                           [solution[(start_index+subset_length-1)%n]]
               -costs[start]
               +costs[solution[(start_index+subset_length+1)%n]]
              )
              subpath_score = previous_subpath_delta + delta
              IF subpath_score > worst_subpath_delta:
                     worst_subpath_delta = subpath_score
                     selected_solution = solution
                     worst_start = start
                     previous_subpath_delta = subpath_score
                     IF random() < acceptance_probability:
                            RETURN selected_solution, worst_start
                     ELSE:
                            acceptance_probability += 0.05
       RETURN selected_solution, worst_start


repair_solution(solution, dist_matrix, costs, num_nodes, start_node):
       start_idx = solution.index(start_node)+1
       repaired_solution = generate_greedy_weight_regret(dist_matrix,
                                                         costs,
                                                         solution,
                                                         num_nodes,
                                                         0.5,
                                                         start_idx)
       RETURN repaired_solution
```

The greedy cycle algorithm used for repairing a solution has been adjusted to work with already provided part of the solution.

```
generate_greedy_weight_regret(dist_matrix, costs, solution, num_select, weight,
                              start_inx):
       num_nodes = dist_matrix.shape[0]
       path = solution[start_idx:]+solution[:start_idx]
       start = path[-1]
       unselected_nodes = set(range(num_nodes)) - set(solution)

       WHILE len(path) < num_select:
              score_node = None
              score_position = None
              best_score = float("-inf")
              FOR node IN unselected_nodes:
                     best_node = None
                     best_position = None
                     best_min_increase = float("inf")
                     second_best_min_increase = float("inf")
                     FOR i IN RANGE(path.index(start), len(selected_nodes)):
                            next_i = (i+1) % len(path)
                            increase = (
                             +dist_matrix[path[i]][node]
                             +dist_matrix[node][path[next_i]]
                             +costs[node]
                             -dist_matrix[path[i]][path[next_i]]
                            IF increase < second_best_min_increase:
                                   IF increase < best_min_increase:
                                          best_min_increase = increase
                                          best_node = node
                                          best_position = next_i
                                   IF second_best_min_increase == float("inf"):
                                          second_best_min_increase = increase
                            ELSE:
                                   second_best_min_increase = increase
                     regret = second_best_min_increase – best_min_increase
                     score = weight * regret – (1-weight)*best_min_increase
                     IF score > best_score:
                            best_score = score
                            score_node = best_node
                            score_position = best_position
              IF score_position == 0:
                     path.append(score_node)
              ELSE:
                     path.insert(score_position, score_node)
              unselected_nodes.remove(score_node)
       RETURN path


large_scale_no_ls(dist_matrix, costs, max_time):
       solution = generate_random_solution(100)
       solution, score = SteepestLocalSearch(solution)
       best_solution, best_score = solution, score
       start = time()
       n_epoch = 0
       WHILE time()-start < max_time:
              n_epoch += 1
              destroyed_sol, start_idx = destroy_solution(best_solution,
                                                          dist_matrix, costs)
              solution = repair_solution(destroyed_sol, dist_matrix, costs, 100,
                                         start_idx)
              IF score < best_score:
                     best_score = score
                     best_solution = solution
       runtime = time() – start
       RETURN best_solution, best_score, runtime, n_epoch
```

```
large_scale_ls(dist_matrix, costs, max_time):
        solution = generate_random_solution(100)
        solution, score = SteepestLocalSearch(solution)
        best_solution, best_score = solution, score
        start = time()
        n_epoch = 0
        WHILE time()-start < max_time:
                n_epoch += 1
                destroyed_sol, start_idx = destroy_solution(best_solution,
                                                            dist_matrix, costs)
                solution = repair_solution(destroyed_sol, dist_matrix, costs, 100,
                                            start_idx)
                solution, score = SteepestLocalSearch(solution)
                IF score < best_score:
                        best_score = score
                        best_solution = solution
        runtime = time() - start
        RETURN best_solution, best_score, runtime, n_epoch
```

Results

Method	Instance A	Instance B	Instance C	Instance D
Greedy LS, random solution, two-edges + inter route	77,014(74,663-79,803)	69,990(67,877-74,141)	50,998 (49,340-53,141)	48,068 (45,336-51,629)
Greedy LS, random solution, two-nodes + inter route	90,940(84,816-99,390)	85,570(77,908-97,299)	63,929 (58,135-70,886)	62,175 (54,310-71,108)
Greedy LS, best solution from 2-regret with weighted sum, two-edges + inter route	75,792 (74,221-79,688)	71,266 (67,384-77,120)	52350,15(48,931-55,758)	51,013 (45,212-59,478)
Greedy LS, best solution from 2-regret with weighted sum, two-nodes + inter route	75,932(74,344-79,315)	71,839 (67,384-77,565)	52,638 (49,649-56,472)	51,248(45,097-60,185)
Steepest LS, best solution from 2-regret with weighted sum, two-edges + inter route	75,728(74,091-79,220)	71,233 (67,384-77,057)	52,299 (49,098-5,5665)	50,977(45,097-59,478)
Steepest LS, best solution from 2-regret with weighted sum, two-nodes + inter route	75,880(74,280-79,220)	71,894(67,384-77,420)	52,607 (49,460-56,472)	51,247 (45,097-60,185)
Candidates LS, random solution, two-edges + inter route	81,129(76,609-86,447)	73,977(69,300-80,189)	51,588(49,120-54,801)	48,429(45,385-51,392)

Steepest LS, random solution, two-nodes + inter route	92,714(84,218-103,034)	87,666(79,356-97,895)	65,679(59,604-73,386)	64,162(54,716-75,351)
Steepest LS, random solution, two-edges + inter route	78,017 (74,874-82,619)	71,337(67,909-76,199)	51,485 (49,235-53,755)	48,225 (45,673-51,639)
Deltas from previous iteration, random solution, two-edges + inter route	78,192(75,149-82,556)	71,709(68,307-76,210)	51,940(49,347-55,591)	48,509(45,966-52,016)
Multi Start Local Search random solution, two-edges + inter route	75,447(74,773-76,051)	68,523(67,810-69,028)	49,567(49,141-50,190)	45,267(45,870-46,275)
Iterated Local Search random solution, two-edges + inter route	**73,114(72,894-73,445)**	**66,239(66,137-66,422)**	**47,259(46,805-47,686)**	**44,131(43,690-44,784)**
Large Scale Search without LS	78,788(76,120-81,680)	72,062(70,211-75,098)	51,986(50,094-53,606)	48,619(46,564-50,932)
Large Scale Search with LS	76,682(74,766-79,213)	70,725(68,804-73,873)	51,275(49,389-53,318)	48,453(46,008-50,428)

Runtimes

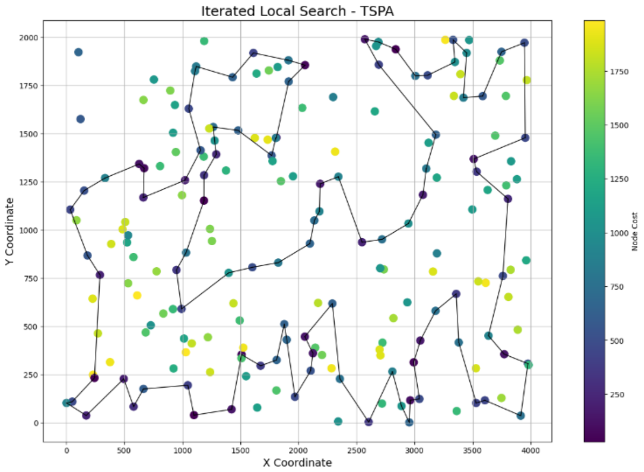
Method	Instance A	Instance B	Instance C	Instance D
Greedy LS, random solution, two-edges + inter route	1.56(1.06-2.63)	1.95(1.19-3.48)	1.25(0.77-2.28)	1.18(0.72-1.99)
Greedy LS, random solution, two-nodes + inter route	1.68(1.03-2.98)	1.95(0.81-6.66)	1.38(0.79-2.21)	1.37(0.77-2.36)
Greedy LS, best solution from 2-regret with weighted sum, two-edges + inter route	0.67(0.51-0.97)	0.7(0.5-1.18)	0.66(0.5-0.93)	0.65(0.51-0.89)
Greedy LS, best solution from 2-regret with weighted sum, two-nodes + inter route	0.63(0.46-0.89)	0.69(0.53-1.15)	0.68(0.49-1.24)	0.67(0.54-1.18)
Steepest LS, best solution from 2-regret with weighted sum, two-edges + inter route	0.85(0.55-1.53)	0.95(0.53-1.78)	0.94(0.57-1.6)	1(0.58-1.38)

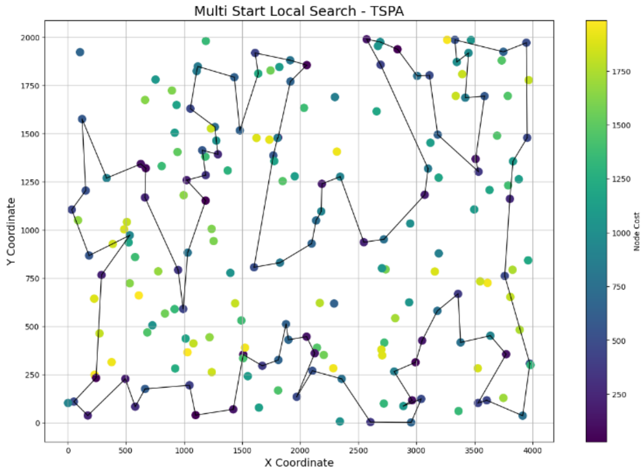
Steepest LS, best solution from 2-regret with weighted sum, two-nodes + inter route	0.88(0.58-1.71)	0.83(0.53-1.57)	0.89(0.5-1.58)	1.03(0.67-1.5)
Candidate LS, random solution, two-edges + inter route	4.43(3.95-6.41)	4.52(3.99-5.70)	4.53(3.83-6.44)	4.58(4.06-5.88)
Steepest LS, random solution, two-nodes + inter route	6.82(5.46-8.96)	6.63(4.89-10.51)	6.8(5.41-9.2)	0.69(0.5-1.18)
Steepest LS, random solution, two-edges + inter route	5.46(4.47-7.46)	5.64(4.51-7.16)	5.41(4.72-6.54)	5.64(4.76-6.88)
Deltas from previous iteration, random solution, two-edges + inter route	1.34(1.06-2.13)	1.80(0.80-2.31)	1.82 (1.05-2.51)	1.88(1.23-2.44)
Multi Start Local Search random solution, two-edges + inter route**	379.31(338.99-403.86)	308.32(294.82-326.69)	301.18(290.45-320.35)	334.86(320.82-346.04)
Iterated Local Search random solution, two-edges + inter route**	379.40(379.32-379.60)	308.42(308.32-308.70)	301.29(301.20-301.44)	334.98(334.86-335.23)
Large Scale Search without LS	379.36(379.31-379.46)	308.38(308.33-308.45)	301.23(301.18-301.30)	334.91(334.88-334.96)
Large Scale Search with LS	379.50(379.32-379.77)	308.48(308.34-308.83)	301.35(301.19-301.78)	335.03(334.88-335.28)

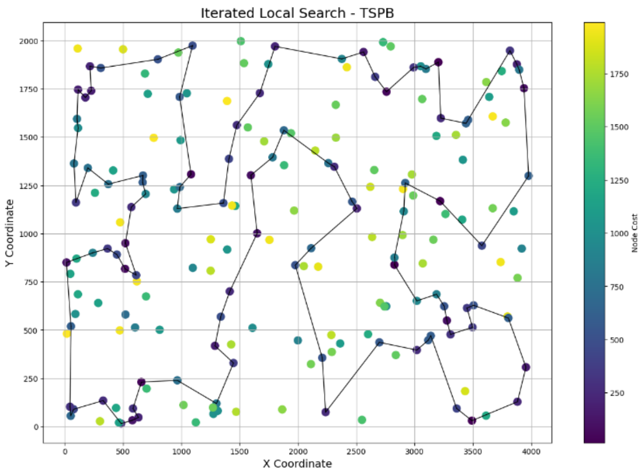
**Runtimes are different in comparison to the previous report, because all experiments has been parallelized and rerun on another machine.

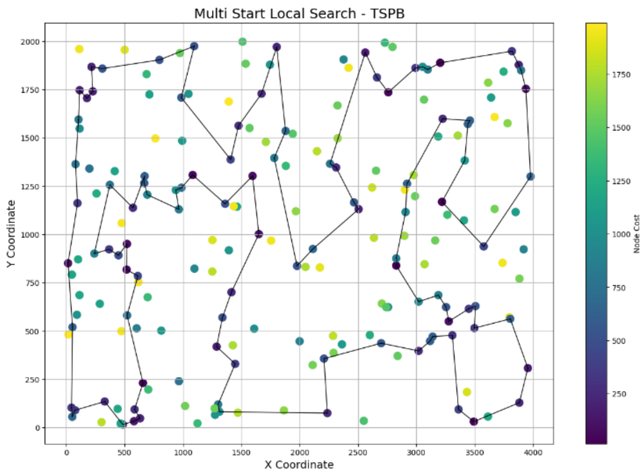
Iterations

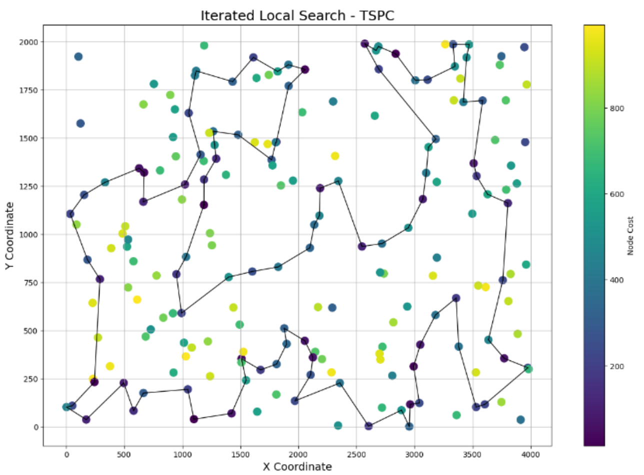
Iterated Local Search random solution, two-edges + inter route	1923(1878-1986)	1534(1478-1579)	1517(1448-1556)	1668(1520-1798)
Large Scale Search without LS	3499(2726-5522)	2794(2248-4449)	2849(2365-4436)	3393(2712-5659)
Large Scale Search with LS	1184(771-1987)	1171(519-2411)	1039(504-1803)	1142(509-2294)

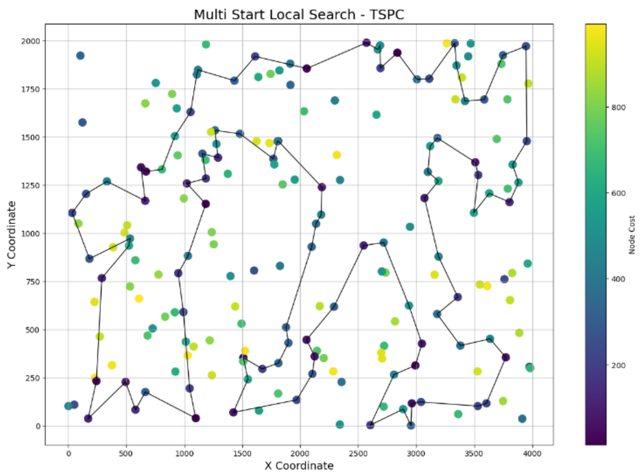

Iterated Local Search - TSPA
X Coordinate
Y Coordinate
Node Cost

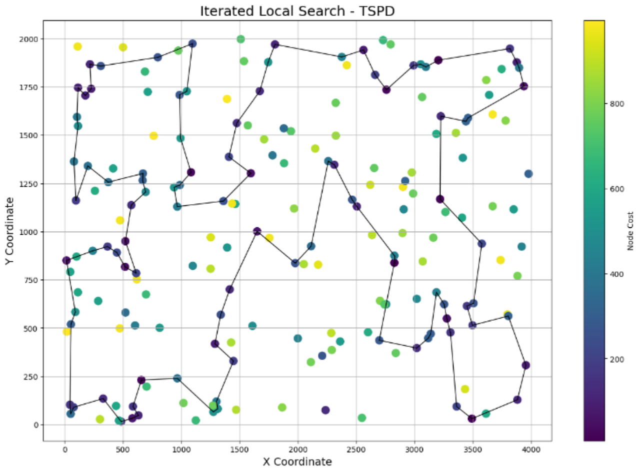

Multi Start Local Search - TSPA
X Coordinate
Y Coordinate
Node Cost

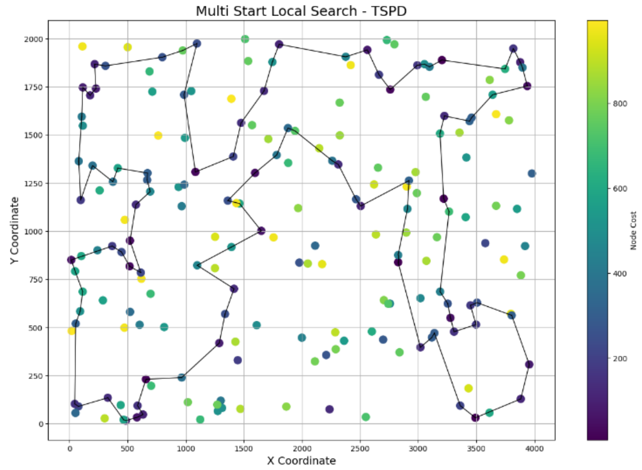

Iterated Local Search - TSPB
X Coordinate
Y Coordinate
Node Cost

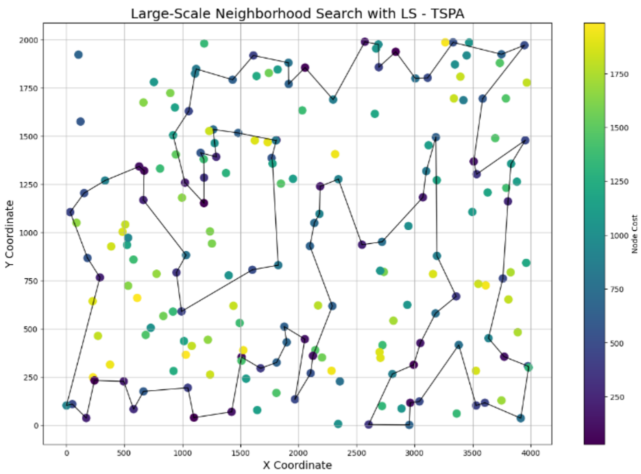

Multi Start Local Search - TSPB
X Coordinate
Y Coordinate
Node Cost

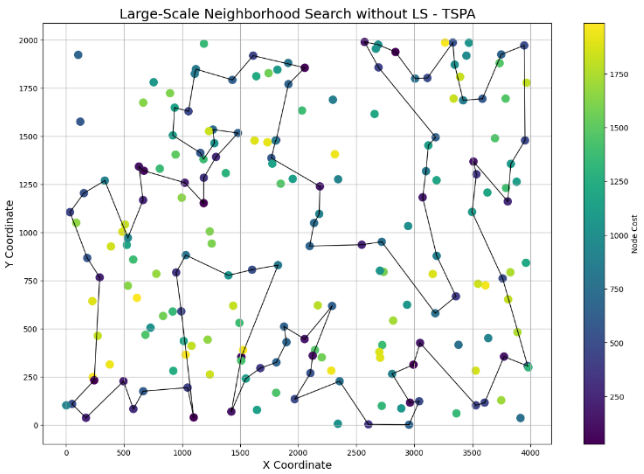

Iterated Local Search - TSPC
X Coordinate
Y Coordinate
Node Cost

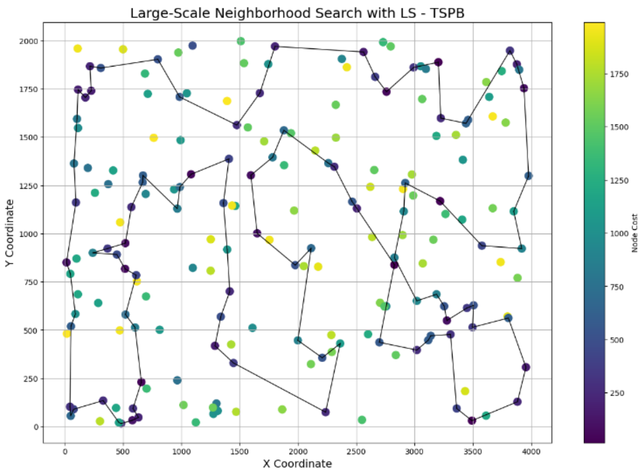

Multi Start Local Search - TSPC
X Coordinate
Y Coordinate
Node Cost

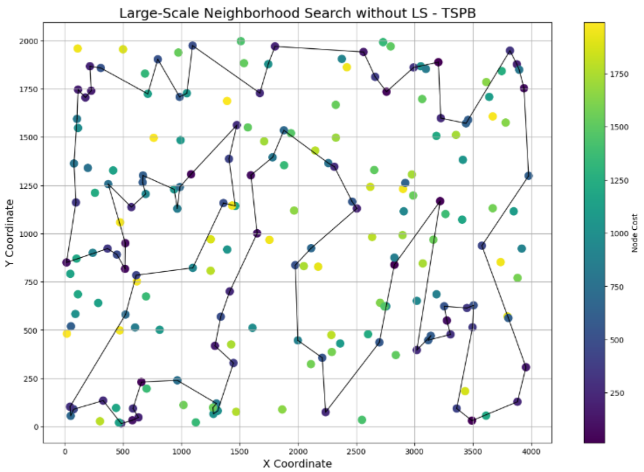

Iterated Local Search - TSPD
Y Coordinate
X Coordinate
Node Cost

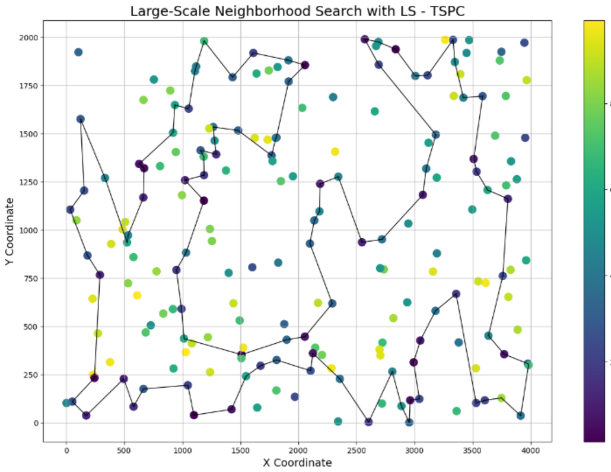

Multi Start Local Search - TSPD
Y Coordinate
X Coordinate
Node Cost

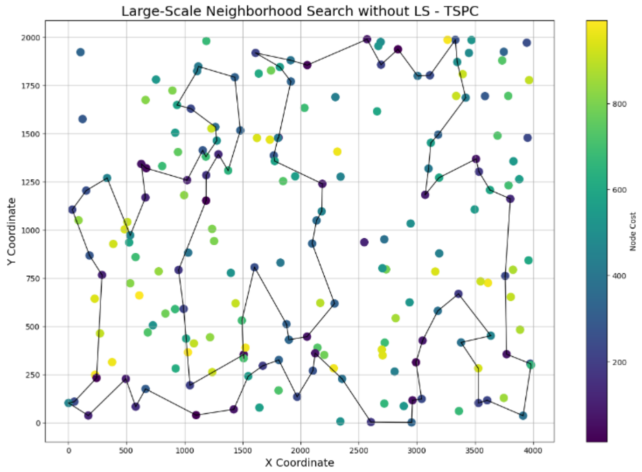

Large-Scale Neighborhood Search with LS - TSPA
Y Coordinate
X Coordinate
Node Cost

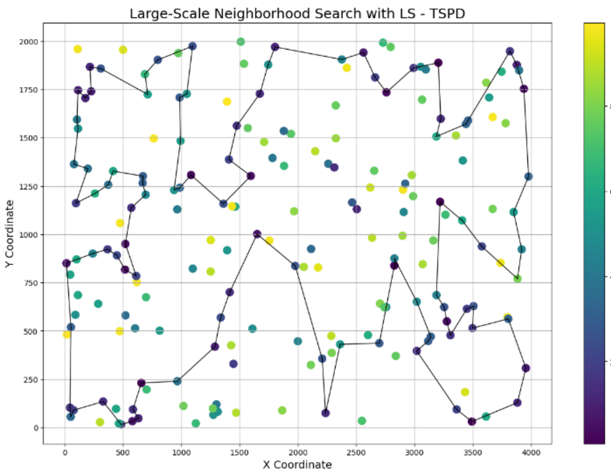

Large-Scale Neighborhood Search without LS - TSPA
Y Coordinate
X Coordinate
Node Cost

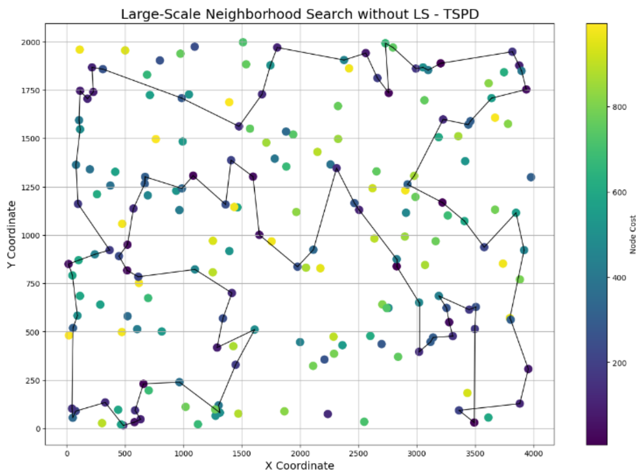

Large-Scale Neighborhood Search with LS - TSPB
Y Coordinate
X Coordinate
Node Cost


Large-Scale Neighborhood Search without LS - TSPB
Y Coordinate
X Coordinate
Node Cost









MSLS:

A:

[178, 19, 0, 149, 50, 91, 121, 114, 4, 43, 77, 192, 199, 41, 1, 137, 177, 174, 75, 189, 109, 119, 130, 92, 48, 152, 11, 160, 106, 26, 8, 124, 80, 14, 111, 94, 12, 89, 73, 31, 95, 169, 112, 72, 190, 98, 156, 6, 66, 51, 135, 101, 167, 45, 186, 127, 88, 153, 161, 76, 21, 194, 79, 87, 141, 144, 154, 133, 171, 81, 180, 32, 62, 108, 15, 117, 53, 22, 195, 55, 36, 128, 132, 113, 74, 163, 61, 183, 71, 20, 64, 181, 185, 96, 27, 147, 59, 143, 159, 164]

B:

[139, 193, 119, 59, 71, 166, 158, 162, 150, 44, 117, 196, 192, 142, 130, 141, 148, 140, 174, 51, 70, 91, 156, 67, 114, 72, 58, 89, 129, 64, 159, 147, 181, 170, 189, 132, 185, 73, 136, 33, 29, 172, 95, 135, 198, 190, 19, 145, 157, 80, 153, 4, 55, 88, 36, 25, 134, 154, 112, 50, 99, 102, 37, 165, 137, 57, 0, 169, 66, 26, 92, 122, 143, 127, 24, 121, 131, 103, 38, 101, 31, 179, 197, 183, 34, 5, 182, 2, 113, 69, 115, 82, 63, 8, 16, 18, 52, 12, 107, 97]

C:

[79, 194, 21, 171, 108, 15, 117, 53, 22, 195, 55, 36, 132, 128, 164, 178, 159, 143, 59, 147, 96, 185, 25, 181, 64, 20, 71, 61, 113, 163, 74, 138, 155, 62, 32, 180, 81, 154, 141, 6, 172, 156, 66, 98, 190, 72, 94, 12, 73, 31, 111, 14, 80, 124, 123, 8, 110, 139, 169, 95, 112, 5, 51, 196, 135, 134, 119, 109, 130, 92, 48, 11, 152, 1, 177, 41, 137, 199, 174, 75, 189, 126, 101, 167, 175, 114, 4, 77, 43, 19, 69, 0, 149, 50, 121, 91, 153, 88, 127, 186]

D:

[79, 136, 61, 73, 185, 132, 12, 189, 170, 100, 181, 147, 159, 64, 129, 89, 58, 72, 114, 85, 166, 28, 59, 119, 193, 71, 44, 196, 117, 150, 162, 67, 45, 78, 3, 156, 91, 51, 174, 188, 140, 148, 141, 130, 142, 53, 82, 63, 8, 84, 14, 16, 65, 52, 18, 29, 33, 6, 19, 190, 198, 135, 57, 0, 169, 66, 34, 183, 197, 31, 101, 38, 103, 131, 24, 127, 121, 179, 143, 122, 92, 116, 99, 146, 137, 37, 165, 123, 154, 134, 25, 36, 194, 88, 55, 4, 153, 80, 157, 145]

ILS:

A:

[32, 180, 81, 154, 144, 141, 87, 79, 194, 21, 171, 108, 15, 117, 22, 55, 36, 132, 128, 145, 76, 161, 153, 88, 127, 186, 45, 167, 101, 99, 135, 51, 112, 66, 6, 156, 98, 190, 72, 12, 94, 89, 73, 31, 111, 14, 80, 95, 169, 8, 26, 92, 48, 106, 160, 11, 152, 130, 119, 109, 189, 75, 1, 177, 41, 137, 174, 199, 192, 175, 114, 4, 77, 43, 121, 91, 50, 149, 0, 19, 178, 164, 159, 143, 59, 147, 116, 27, 96, 185, 64, 20, 71, 61, 163, 74, 113, 195, 53, 62]

B:

[70, 51, 174, 140, 148, 141, 130, 142, 53, 69, 115, 82, 63, 8, 16, 18, 29, 33, 19, 190, 198, 135, 95, 172, 182, 2, 5, 34, 183, 197, 31, 101, 38, 103, 131, 24, 127, 121, 179, 143, 122, 92, 26, 66, 169, 0, 57, 99, 50, 112, 154, 134, 25, 36, 165, 37, 137, 88, 55, 4, 153, 80, 157, 145, 79, 136, 73, 185, 132, 52, 139, 107, 12, 189, 170, 181, 147, 159, 64, 129, 89, 58, 171, 72, 114, 85, 166, 59, 119, 193, 71, 44, 196, 117, 150, 162, 158, 67, 156, 91]

C:

[112, 5, 51, 135, 99, 101, 167, 45, 186, 127, 88, 153, 161, 76, 145, 128, 132, 36, 55, 22, 117, 15, 108, 171, 21, 194, 79, 87, 141, 144, 102, 154, 81, 180, 32, 62, 53, 195, 113, 74, 163, 61, 71, 20, 64, 185, 96, 27, 116, 147, 59, 143, 159, 164, 178, 19, 69, 0, 149, 50, 121, 91, 114, 4, 77, 43, 192, 199, 137, 41, 177, 1, 75, 189, 109, 119, 130, 152, 11, 106, 48, 92, 26, 8, 110, 169, 95, 31, 73, 89, 42, 94, 12, 72, 190, 98, 156, 172, 6, 66]

D:

[91, 70, 51, 174, 140, 148, 141, 130, 142, 53, 32, 113, 69, 115, 82, 63, 8, 16, 172, 95, 19, 190, 198, 135, 169, 66, 128, 5, 34, 183, 197, 92, 122, 143, 179, 31, 101, 38, 103, 131, 121, 127, 24, 50, 99, 137, 37, 165, 154, 134, 25, 36, 88, 55, 4, 153, 80, 157, 145, 79, 136, 61, 73, 185, 132, 18, 52, 139, 97, 107, 12, 109, 189, 47, 170, 181, 147, 159, 64, 129, 89, 58, 171, 72, 114, 85, 166, 59, 119, 193, 71, 44, 196, 117, 150, 162, 158, 67, 3, 156]

LSNS without LS:

A:

[50, 121, 91, 114, 175, 4, 77, 43, 192, 199, 41, 1, 177, 174, 75, 189, 152, 11, 48, 106, 26, 139, 169, 95, 8, 124, 80, 14, 111, 31, 73, 12, 94, 72, 190, 98, 156, 6, 66, 112, 5, 51, 135, 134, 119, 109, 101, 167, 153, 88, 127, 186, 21, 194, 79, 87, 141, 154, 81, 180, 32, 62, 93, 155, 53, 18, 15, 108, 171, 117, 22, 55, 195, 74, 163, 113, 181, 61, 71, 20, 64, 185, 96, 27, 147, 59, 143, 159, 164, 178, 128, 132, 36, 145, 76, 161, 0, 19, 69, 149]

B:

[69, 53, 142, 130, 141, 148, 140, 174, 51, 70, 91, 192, 196, 84, 52, 65, 132, 185, 12, 107, 139, 97, 59, 119, 193, 71, 166, 44, 117, 150, 162, 67, 114, 72, 58, 89, 129, 64, 159, 147, 87, 181, 189, 47, 170, 73, 61, 136, 145, 157, 80, 153, 88, 137, 37, 165, 36, 25, 134, 154, 112, 50, 131, 103, 38, 101, 31, 121, 24, 127, 122, 143, 179, 183, 197, 34, 99, 66, 128, 5, 2, 182, 163, 172, 95, 135, 198, 190, 19, 33, 29, 18, 16, 8, 63, 82, 115, 113, 32, 184]

C:

[92, 119, 109, 189, 75, 1, 177, 41, 199, 192, 4, 77, 43, 50, 149, 69, 19, 178, 159, 143, 59, 147, 116, 27, 96, 185, 64, 20, 71, 61, 181, 113, 163, 74, 195, 53, 18, 15, 108, 62, 93, 32, 180, 81, 171, 28, 117, 22, 55, 132, 128, 40, 164, 0, 115, 49, 76, 91, 121, 114, 175, 153, 88, 127, 186, 157, 21, 194, 79, 87, 102, 154, 144, 141, 6, 66, 98, 190, 72, 94, 12, 73, 112, 5, 51, 135, 196, 95, 169, 110, 8, 26, 48, 106, 198, 160, 11, 152, 16, 130]

D:

[180, 80, 157, 145, 79, 136, 61, 73, 185, 189, 147, 159, 64, 129, 89, 58, 171, 72, 114, 162, 158, 67, 45, 3, 156, 91, 70, 51, 174, 140, 148, 141, 130, 142, 53, 113, 69, 115, 40, 82, 63, 8, 84, 196, 117, 150, 44, 71, 119, 59, 107, 12, 52, 132, 18, 16, 172, 95, 190, 198, 135, 169, 66, 26, 34, 183, 197, 179, 143, 122, 92, 127, 24, 121, 101, 31, 38, 103, 131, 93, 152, 94, 50, 43, 99, 57, 62, 137, 37, 165, 123, 154, 134, 25, 36, 88, 55, 4, 153, 160]

LSNS with LS:

A:

[119, 109, 189, 75, 174, 199, 41, 177, 1, 130, 152, 11, 160, 198, 106, 48, 92, 26, 8, 124, 80, 169, 95, 31, 14, 111, 94, 72, 190, 98, 66, 156, 6, 24, 141, 87, 144, 154, 81, 180, 32, 62, 155, 195, 55, 22, 18, 53, 117, 15, 108, 171, 194, 79, 21, 157, 161, 76, 128, 132, 36, 113, 74, 163, 61, 71, 20, 64, 185, 116, 27, 147, 96, 59, 143, 159, 164, 178, 19, 0, 149, 50, 121, 91, 114, 43, 77, 4, 175, 153, 88, 127, 186, 45, 167, 101, 135, 51, 112, 134]

B:

[169, 0, 57, 99, 50, 152, 94, 112, 154, 25, 36, 165, 37, 137, 88, 153, 80, 157, 145, 136, 73, 185, 189, 181, 147, 159, 64, 129, 89, 58, 72, 114, 67, 45, 3, 156, 91, 70, 51, 174, 188, 140, 148, 141, 130, 142, 21, 192, 196, 117, 150, 158, 162, 44, 71, 119, 59, 139, 107, 12, 132, 52, 14, 8, 63, 82, 115, 2, 133, 182, 163, 95, 172, 16, 18, 29, 33, 19, 190, 198, 135, 66, 128, 5, 34, 183, 197, 179, 31, 101, 38, 103, 131, 121, 24, 127, 143, 122, 92, 26]

C:

[96, 27, 147, 59, 143, 159, 164, 178, 19, 69, 149, 50, 77, 4, 192, 199, 174, 137, 41, 177, 1, 75, 189, 109, 119, 152, 11, 162, 160, 198, 106, 48, 92, 26, 8, 110, 169, 95, 31, 73, 12, 94, 72, 190, 98, 156, 6, 66, 112, 51, 135, 101, 167, 45, 186, 127, 88, 153, 175, 114, 121, 0, 40, 128, 132, 36, 55, 195, 22, 18, 53, 117, 15, 108, 171, 21, 194, 79, 87, 141, 144, 154, 81, 131, 180, 32, 62, 93, 155, 163, 74, 113, 181, 25, 61, 183, 71, 20, 64, 185]

D:
[187, 181, 170, 47, 189, 109, 97, 107, 12, 52, 18, 132, 185, 73, 61, 136, 79, 145, 157, 80, 153, 88, 137, 146, 37, 165, 36, 25, 154, 112, 94, 152, 125, 50, 43, 99, 92, 122, 143, 179, 127, 24, 121, 131, 103, 38, 101, 31, 34, 183, 197, 26, 169, 66, 128, 5, 133, 2, 182, 172, 16, 8, 63, 82, 53, 142, 130, 141, 148, 140, 174, 51, 70, 91, 156, 67, 158, 162, 150, 117, 196, 44, 71, 193, 119, 59, 28, 166, 110, 114, 85, 72, 171, 58, 89, 129, 64, 159, 147, 87]

Conclusions

The results obtained by both variants are slightly worse in comparison to the algorithms implemented in the previous assignment. The key aspect of it might be that the greedy algorithm wasn't that valuable that the Local Search. The confirmation of it is directly seen in the objective function values of two methods, where the version with applied Local Search to the solution repaired by the greedy heuristic obtained better results. However, still both methods were able to find quite good routes without crossings in the path.

According to the number of iterations in the main loop, there is no surprise that the methods without Local Search had more epochs per run. It's due to the fact that the algorithm didn't spend time on the Local Search computations, and basically performing only greedy heuristics paths was significantly cheaper in the case of the computation complexity.