Evolutionary Computation

Assignment 5

Antoni Lasik 148287

Daniel Jankowski 148257

<https://github.com/JankowskiDaniel/evolutionary-computation/tree/AL/assignment5>

**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 report we implement the steepest version of a local search with deltas from previous iteration. While the results should be almost the same, the purpose of the modification is to reduce the runtime of a classical algorithm version.

**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**

The methods for checking the applicability of a given move. The methods returns the following values:

* -1 if a move is not applicable and should be removed from LM
* 0 if a move is not applicable now, but shouldn’t be removed from LM
* 1 if a move is applicable and will be accepted

**is\_intra\_move\_applicable(solution, move):**

# for the intra moves changes affects only nodes that are inside a given solution, # therefore first we check if all nodes in edges, that a move introduced, are # present in the current solution

**FOR** edge **IN** move.added\_edges:

**IF** edge.source\_node **NOT IN** solution **OR**

edge.dest\_node **NOT IN** solution:

**RETURN -1**

# check if all edges that a move removed are present in a solution

all\_edges\_match = True

**FOR** edge **IN** move.removed\_edges:

reversed\_edge = edge[-1] # reverse the edge

**IF** edge **NOT IN** solution **AND** reversed\_edge **NOT IN** solution:

**RETURN -1**

**IF** edge **NOT** **IN** solution **AND** reversed\_edge **IN** solution:

all\_edges\_match = False

**RETURN 1 IF** all\_edges\_match **ELSE 0**

**is\_inter\_move\_applicable(solution, move):**

# added edges by a move have shape:(old\_node\_1, NEW\_NODE),(NEW\_NODE, old\_node\_2)

# therefore first we check if all old nodes are present in the current solution

**IF** move.added\_edges[0].source\_node **NOT IN** solution **OR**

move.added\_edges[1].dest\_node **NOT** **IN** solution:

**RETURN -1**

# if the node that will be inserted is not in the list of currently unselected # nodes, a move can’t be applied

**IF** move.added\_edges[0].dest\_node **NOT IN** unselected\_nodes:

**RETURN -1**

# check if all edges that a move removed are present in a solution

all\_edges\_match = True

**FOR** edge **IN** move.removed\_edges:

reversed\_edge = edge[-1] # reverse the edge

**IF** edge **NOT IN** solution **AND** reversed\_edge **NOT IN** solution:

**RETURN -1**

**IF** edge **NOT** **IN** solution **AND** reversed\_edge **IN** solution:

all\_edges\_match = False

**RETURN 1 IF** all\_edges\_match **ELSE 0**

To control moves that has been already discovered and evaluated the heap has been implemented using built-in python package. The heap stores information about discovered moves and their delta score, always sorted in a descending order by the score. We have used two basic methods:

* heap.add\_move() – add a new move to the heap
* heap.heappop() – take the first element from the heap (with the best score)

**two\_edges\_exchange(solution, heap, dist\_matrix, nodes\_to\_check):**

n = len(solution)

nodes\_indices = [solution.index(i) **FOR** i **IN** nodes\_to\_check]

**FOR** i **IN** nodes\_indices:

**FOR** j **IN** RANGE(n):

x, y = i, j

**IF** y < x:

x, y = y, x

**IF** abs(x-y) **>=** 2:

score\_delta = (

-dist\_matrix[solution[x]][solution[x+1]]

-dist\_matrix[solution[y]][solution[(y+1)%n]]

+dist\_matrix[solution[x]][solution[y]]

+dist\_matrix[solution[x+1]][solution[(y+1)%n]]

**IF** score\_delta **<** 0:

added\_edges = ((solution[x], solution[y]),

(solution[x+1], solution[(y+1)%n]))

removed\_edges = ((solution[x], solution[x+1]),

(solution[y], solution[(y+1)%n]))

# Add a move to the LM

heap.add\_move((score\_delta, (added\_edges, removed\_edges)))

**inter\_route\_exchange(solution, unselected, heap, dist\_matrix, nodes\_to\_check):**

n\_unselected = len(unselected)

index\_pairs = [(solution.index(i), j) **FOR** i **IN** nodes\_to\_check **FOR** j **IN**

**RANGE**(n\_uselected)]

**FOR** i, j **IN** index\_pairs:

selected\_node = solution[i]

new\_node = unselected[j]

new\_solution = solution.copy()

new\_solution[i] = new\_node

prev\_node\_index = (i-1)% len(solution)

next\_node\_index = (i+1)% len(solution)

removed\_edges = ((solution[prev\_node\_index], selected\_node),

(selected\_node, solution [next\_node\_index]))

added\_edges = ((solution[prev\_node\_index], new\_node),

(new\_node, solution[next\_node\_index]))

score\_delta = (

-dist\_matrix[solution[prev\_node\_index]][selected\_node]

-dist\_matrix[selected\_node][solution[next\_node\_index]]

+dist\_matrix[solution[prev\_node\_index]][new\_node]

+dist\_matrix[new\_node][solution[next\_node\_index]]

-selected\_node.cost

+new\_node.cost

**IF** score\_delta < 0:

heap.add\_move((score\_delta, (added\_edges, removed\_edges)))

**apply\_edge\_move(solution, added\_edges):**

# indices of first nodes of edges that are exchanged

x = solution.index(added\_edges[0][0])

y = solution.index(added\_edges[0][1])

**IF** y < x:

X, y = y, x

# update class parameter that holds current solution

new\_solution = solution[:x + 1] + solution[x+1:y+1][::-1] + solution[y+1:]

# return new solution and all nodes that has been affected by a move, and should

# be checked in the next iteration

**RETURN** new\_solution, added\_edges[0] + added\_edges[1]

**apply\_inter\_move(solution, added\_edges, removed\_edges):**

new\_node = added\_edges[0][1]

index\_old\_node = solution.index(removed\_edges[0][1])

# update changed node in the solution

solution[index\_old\_node] = new\_node

self.unselected\_nodes.remove(new\_node)

self.unselected\_nodes.append(removed\_edges[0][1])

# return all nodes that has been affected by a move, and should be checked in the

# next iteration

**RETURN** solution, (added\_edges[0] + added\_edges[1])

**run\_algorithm(start\_solution, dist\_matrix, costs, moves)**

current\_score = objective\_function(start\_solution, dist\_matrix, costs)

LM = []

solution = start\_solution

# Call both moves methods to add the whole neighborhood in the first iteration to

# the LM

inter\_route\_exchange(solution, unselected, LM, dist\_matrix)

two\_edges\_exchange(solution, LM, dist\_matrix)

progress = True

**WHILE**(progress):

progress = False

moves\_to\_restore = []

**WHILE**(LM):

delta\_score, (move\_type, added\_edges, removed\_edges) = LM.heappop()

# call move applicability w.r.t. to its type

**IF** is\_move\_applicable(solution, added\_edges, removed\_edges) == 1:

# a move is applicable and will be accepted:

new\_solution, affected\_nodes = apply\_move(solution,added\_edges)

solution = new\_solution

current\_score += delta\_score

progress = True

**BREAK**

**ELIF** is\_move\_applicable(solution, added\_edges, removed\_edges) == 0:

moves\_to\_restore.append((delta\_score, (move\_type, added\_edges,

removed\_edges)))

**ELSE**:

# a move is not applicable and it’s okay to keep it deleted

**CONTINUE**

**FOR** move **IN** moves\_to\_restore:

LM.add\_move(move)

**IF** progress:

# performs new moves, and add them to the LM

inter\_route\_exchange(solution, LM, dist\_matrix, affected\_nodes)

two\_edges\_exchange(solution, LM, dist\_matrix, affected\_nodes)

**RETURN** solution, current\_score

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Instance A | Instance B | Instance C | Instance D |
| Random solution | 264,028(237,941-288,302) | 266,665(243,288-295,269) | 214,929(192,705-241,451) | 219,678(191,218-242,515) |
| Nearest Neighbor | 87,679(84,471-95,013) | 79,282(77,448-82,631) | 58,290(56,304-63,697) | 54,290.68(50,335-59,846) |
| Greedy Cycle | 76,711(75,136-80,025) | 70,464(67,896-76,096) | 55,859(53,020-58,499) | 54,931(50,288-60,208) |
| 2-regret GC | 116,772(106,734-124,404) | 116,871(104,997-125,925) | 68,444(63,247-72,558) | 68,585(62,852-74,184) |
| 2-regret with weighted sum | 76,980(74,708-82,990) | 73,067(67,490-80,001) | 53,795(50,158-58,173) | 52,930(46,549-62,321) |
| 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) |

**Results**

*\*the structure of values in the table: avg(min-max)*

*\*\*in bold there are the best minimal solutions founded by a given method*

**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) |

***\*****the format is: avg(min-max)*

***\*\*****all runtimes are provided in seconds.*

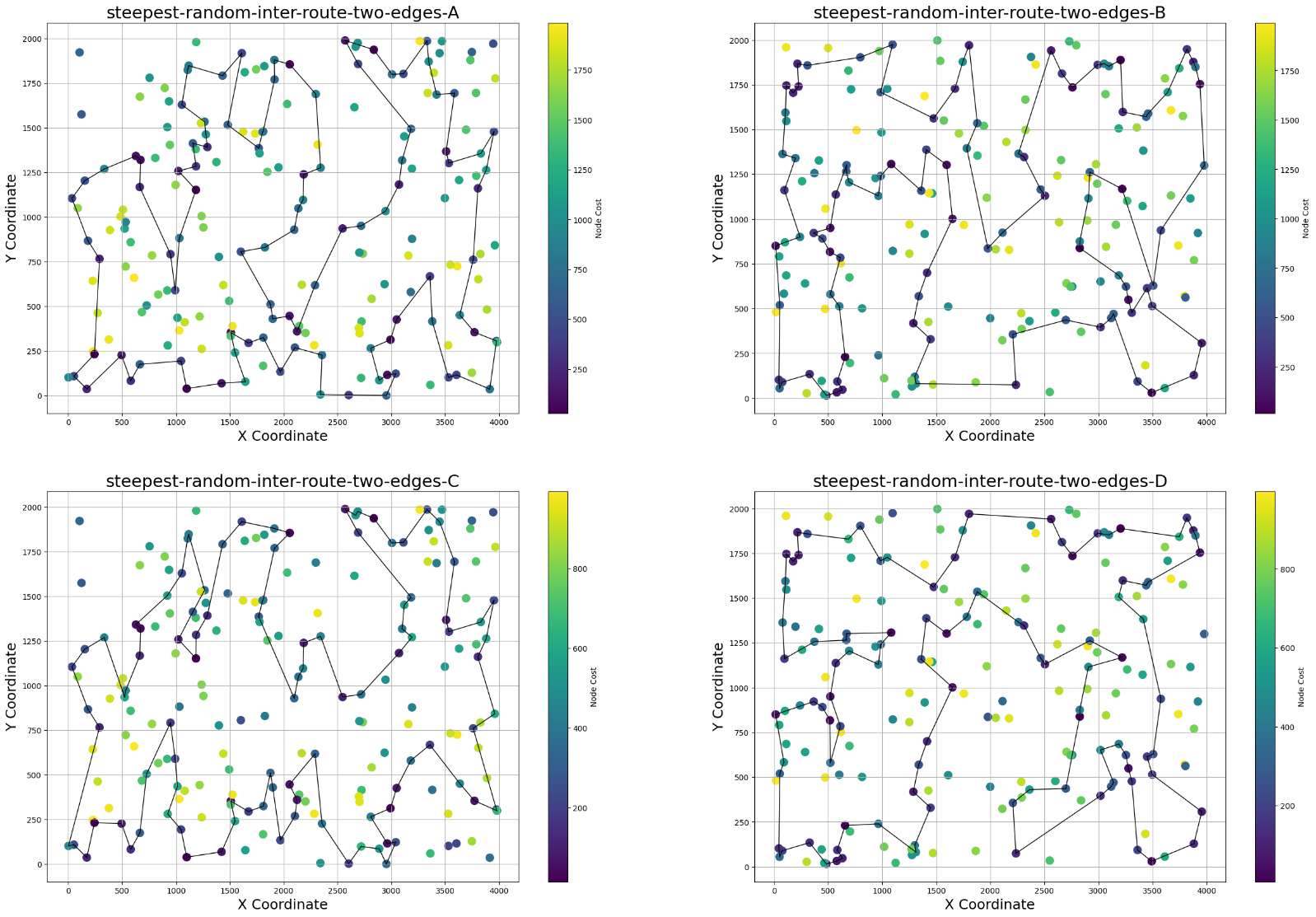
Below we’ve written down best paths found for the Candidate Local Search that were missing in the previous report.

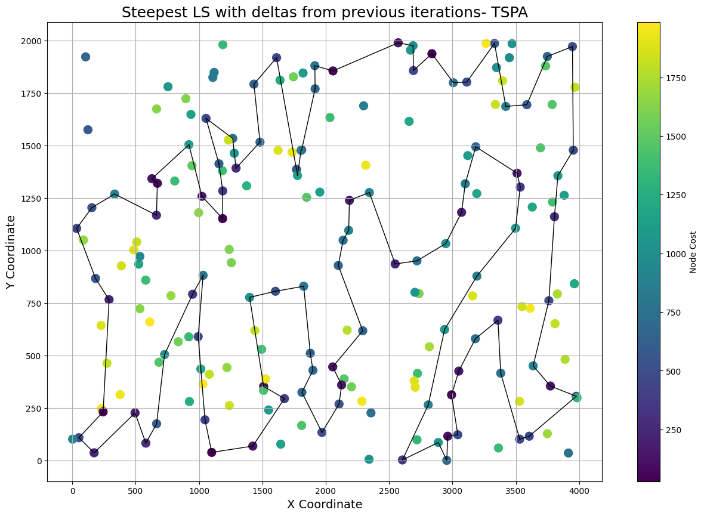
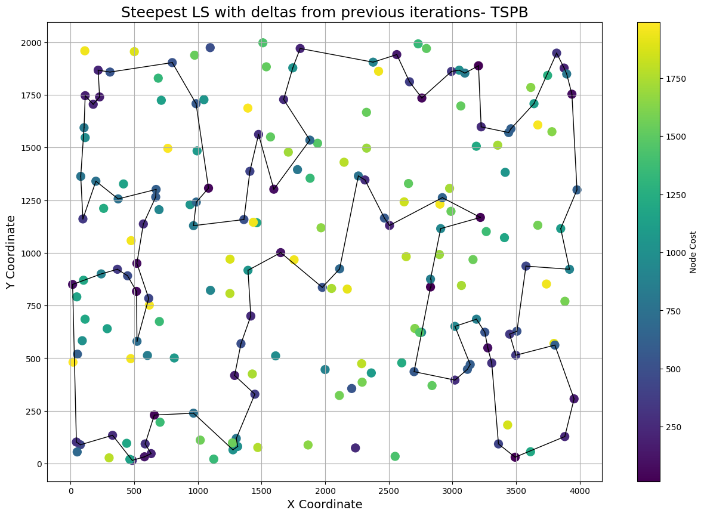
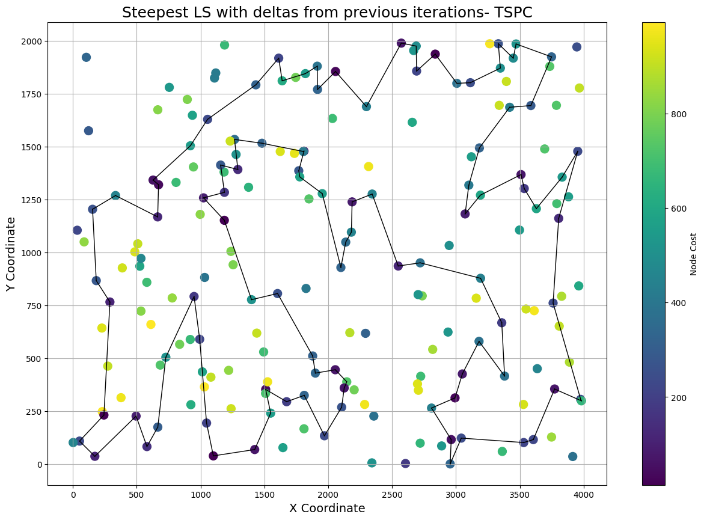
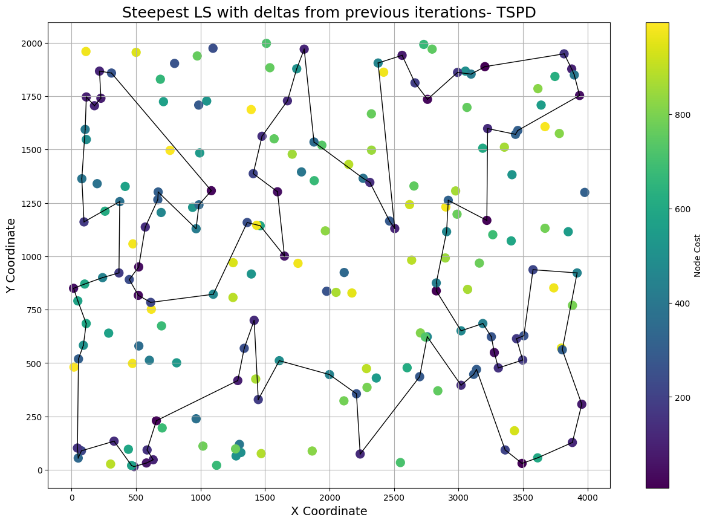
**TSPA**: [43, 121, 50, 149, 0, 19, 178, 164, 37, 159, 143, 59, 96, 147, 27, 116, 185, 64, 20, 71, 163, 74, 113, 132, 128, 36, 55, 195, 22, 18, 53, 93, 62, 180, 81, 154, 144, 87, 141, 24, 170, 21, 79, 194, 171, 108, 15, 117, 28, 76, 91, 114, 4, 175, 153, 88, 127, 186, 45, 6, 156, 172, 66, 98, 190, 72, 94, 111, 14, 80, 8, 169, 95, 31, 73, 112, 51, 196, 135, 101, 167, 126, 109, 119, 26, 92, 48, 106, 198, 160, 11, 152, 130, 189, 75, 1, 177, 41, 199, 77]

**TSPB**: [153, 80, 157, 145, 79, 136, 61, 73, 33, 29, 18, 132, 81, 185, 189, 170, 181, 147, 159, 64, 129, 89, 58, 85, 114, 67, 91, 70, 51, 174, 188, 140, 148, 141, 130, 142, 192, 196, 117, 150, 162, 44, 71, 119, 59, 97, 107, 12, 52, 16, 8, 63, 82, 115, 40, 2, 133, 75, 182, 163, 172, 95, 190, 198, 135, 169, 66, 128, 167, 5, 34, 183, 197, 92, 122, 143, 179, 127, 24, 121, 31, 101, 38, 103, 131, 152, 50, 43, 99, 57, 146, 137, 37, 165, 154, 25, 36, 88, 55, 4]

**TSPC**: [108, 15, 53, 62, 32, 180, 81, 154, 144, 141, 87, 79, 194, 21, 88, 127, 186, 45, 24, 6, 156, 172, 66, 98, 190, 72, 94, 12, 73, 31, 80, 124, 8, 169, 95, 112, 51, 196, 135, 99, 101, 167, 60, 126, 109, 119, 92, 48, 106, 160, 11, 152, 1, 41, 137, 177, 75, 189, 174, 199, 192, 4, 114, 77, 43, 50, 121, 91, 161, 76, 145, 36, 132, 128, 40, 0, 149, 69, 19, 178, 164, 143, 59, 147, 27, 96, 185, 64, 20, 71, 61, 113, 163, 74, 138, 195, 55, 22, 117, 171]

**TSPD**: [171, 129, 64, 147, 159, 89, 58, 72, 114, 158, 162, 150, 44, 117, 196, 192, 67, 3, 156, 91, 70, 51, 174, 188, 140, 148, 141, 130, 142, 53, 82, 63, 8, 16, 172, 95, 163, 182, 5, 128, 66, 34, 183, 197, 179, 143, 122, 127, 24, 121, 31, 101, 38, 103, 131, 50, 94, 112, 23, 154, 134, 25, 36, 194, 123, 165, 37, 102, 99, 137, 88, 55, 153, 157, 80, 57, 0, 135, 198, 190, 19, 6, 33, 29, 18, 73, 136, 185, 132, 65, 52, 12, 107, 139, 193, 119, 59, 71, 166, 85]



**TSPA**: [22, 53, 62, 108, 15, 117, 171, 81, 154, 21, 157, 194, 79, 87, 144, 141, 6, 156, 66, 98, 190, 72, 94, 73, 31, 111, 14, 80, 124, 8, 26, 92, 48, 106, 11, 152, 130, 119, 109, 189, 75, 1, 177, 41, 137, 199, 174, 126, 134, 139, 169, 95, 112, 51, 135, 99, 101, 167, 45, 186, 127, 88, 153, 175, 114, 4, 77, 43, 50, 121, 91, 161, 76, 145, 0, 149, 19, 178, 164, 128, 36, 132, 37, 159, 143, 59, 147, 27, 96, 185, 64, 20, 71, 61, 113, 74, 163, 155, 195, 55]

**TSPB**: [53, 32, 113, 69, 115, 82, 63, 8, 14, 16, 172, 95, 19, 190, 198, 135, 57, 99, 0, 169, 66, 5, 34, 183, 197, 26, 92, 122, 143, 179, 31, 101, 38, 103, 131, 121, 127, 24, 50, 152, 94, 112, 154, 134, 25, 36, 123, 165, 37, 137, 88, 55, 4, 153, 80, 157, 145, 79, 136, 61, 73, 33, 18, 185, 132, 52, 139, 107, 12, 189, 181, 147, 159, 64, 129, 89, 58, 72, 114, 85, 166, 59, 119, 71, 44, 196, 192, 117, 150, 162, 158, 67, 91, 51, 174, 140, 148, 141, 130, 142]

**TSPC**: [24, 6, 156, 66, 98, 190, 72, 12, 94, 89, 42, 111, 31, 73, 112, 51, 135, 196, 95, 169, 110, 124, 80, 8, 26, 106, 198, 48, 11, 152, 1, 41, 177, 174, 75, 189, 109, 130, 119, 134, 101, 167, 45, 186, 127, 88, 153, 170, 157, 21, 79, 194, 171, 108, 117, 53, 22, 195, 55, 145, 76, 91, 121, 114, 2, 4, 77, 43, 50, 149, 0, 69, 19, 178, 164, 128, 132, 37, 159, 143, 59, 147, 27, 96, 185, 64, 71, 61, 113, 74, 163, 155, 62, 81, 154, 133, 102, 144, 87, 141]

**TSPD**: [57, 99, 137, 37, 165, 154, 134, 25, 36, 88, 55, 153, 80, 157, 145, 79, 135, 198, 190, 19, 33, 136, 61, 73, 185, 132, 18, 16, 65, 52, 84, 196, 117, 150, 44, 71, 119, 59, 139, 107, 12, 147, 159, 64, 129, 89, 58, 72, 114, 166, 162, 158, 126, 67, 45, 78, 3, 156, 91, 70, 51, 174, 188, 140, 148, 141, 130, 142, 82, 63, 8, 115, 40, 163, 182, 2, 5, 128, 34, 183, 197, 31, 101, 42, 38, 103, 131, 152, 50, 24, 127, 121, 179, 143, 122, 92, 26, 66, 169, 0]

**Conclusions**

Results obtained by the modified version of a Steepest Local Search, where we store deltas from previous iterations, are very similar to the classical version of the algorithm. It was expected, since the main idea for finding a best solution didn’t changed. The size of the neighborhood was the same, however, the way how we explore it was different. Instead of evaluating each move, we were computing delta score only for new moves, and considering only those, that provided improvement with respect to the current solution.

The purpose of this change was to reduce the runtime in comparison to the classical version of the Steepest Local Search. As the provided results showed, the implementation with deltas from previous iteration was approximately 3-4x faster than the original approach. The most positive impact on runtime was primarily due to different way of browsing the neighborhood. We haven’t anymore iterating in each iteration through all moves that are possible, but instead we were only checking solutions that were new. It was achieved, by creating new moves only for parts of the solution, that has been modified in the previous iteration. For example, if in the previous iteration, a move has introduced two new edges to the solution, in the next epoch we were only iterating trough all moves, that contains these edges (or their neighbor edges), since all other moves has been already evaluated and potentially stored in the LM previously.