POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

LABORATORY 3: LOCAL SEARCH

Piotr Balewski, 156037
Lidia Wiśniewska, 156063

POZNAŃ 2025

1 Problem Description

The problem description remains the same as in previous labs. The problem is to select exactly 50% of the nodes (from n available) and build a Hamiltonian cycle through the selected nodes. The objective is to minimize the sum of two components:

1. The total length of the Hamiltonian cycle (calculated as the sum of Euclidean distances between consecutive nodes in the cycle).
2. The total cost of the selected nodes (the sum of the cost attributes for each selected node).

All distances are rounded to the nearest integer.

2 Implemented Algorithms

This lab implements local search (LS) algorithms, which start with a given initial solution and iteratively introduce small modifications (moves) to reduce the objective function cost until a local minimum is reached. We have explored 8 combinations based on search, neighborhood and starting types.

2.1 Search Types

Two main move selection strategies have been implemented:

2.2 Steepest Descent

In each iteration, the entire defined neighborhood is searched (all possible moves of both types: intra- and inter-route). The move guaranteeing the greatest improvement (i.e., with the most negative delta) is then executed. The algorithm terminates when no move in the neighborhood yields an improvement.

```
procedure steepestSearch(current_solution, remaining, inst, move_type)
  bestDelta, bestMove = 0, null

  // 1. Check all Intra-route moves
  for i =  0 to current_solution.size()
    for j = i + 1 to current_solution.size()
      if move_type == "node"
        delta = calculateDeltaIntraNode(i, j, current_solution, inst)
      else
        delta = calculateDeltaIntraEdge(i, j, current_solution, inst)

      if delta < bestDelta
        bestDelta = delta
        bestMove = createIntraMove(i, j, move_type)
```

```
15
16  // 2. Check all Inter-route moves
17  for i = 0 to current_solution.size()
18    for k_node in remaining
19      delta = calculateDeltaInter(i, k_node, current_solution, inst)
20      if delta < bestDelta
21        bestDelta = delta
22        bestMove = createInterMove(i, k_node)
23
24  // 3. Apply best move if found
25  if bestMove != null
26    applyMove(bestMove, current_solution, remaining)
27    return true
28
29  return false //for the general LS funciton; stop when better solution not found
```

2.3 Greedy

The neighborhood is browsed in a random order. The algorithm executes the first encountered move that improves the current solution (has a negative delta). After executing the move, the iteration ends, and a new search begins (with a new random order).

```
1  procedure greedySearch(current_solution, remaining, inst, move_type, rand)
2      //create the neighbourhood
3      inter_moves = shuffled list of indices pairs //(i_idx, k_node)
4      intra_moves = shuffled list of indices pairs //(i_idx, j_idx)
5
6      boolean checkInterFirst
7      if checkInterFirst:
8          //1. Inter-route
9          if (findApplyBest(sol, rem, inst, move_type, inter_moves, isInter=true)):
10             return true
11         //2. Intra-route (nodes or edges)
12         if (findApplyBest(sol, rem, inst, move_type, intra_moves, false)):
13             return true
14     else:
15         //1. Intra-route (nodes or edges)
16         if (findApplyBest(sol, rem, inst, move_type, intra_moves, false)):
17             return true
18         //2. Inter-route
19         if (findApplyBest(sol, rem, inst, move_type, inter_moves, true)):
20             return true
21
22     return false //no better solutions found
```

2.4 Neighborhood Types

The LS algorithm uses two types of moves, which together form the searched neighborhood.

2.5 Inter-route Moves (external)

Exchanging one node i belonging to the cycle with one node k not belonging to the cycle. The delta considers both the change in cost (removing cost(i), adding cost(k)) and the change in cycle length (removing i and inserting k in its place).

```
procedure calculateDeltaInter(i_idx, k_node, solution, inst)
  n = solution.size

  i_node = solution[i_idx]
  i_prev = solution[((i_idx - 1 + n) % n)] // handles wrap-around
  i_next = solution[((i_idx + 1) % n)]

  removed_cost = dist(i_prev, i_node) + dist(i_node, i_next) + cost(i_node)
  added_cost = dist(i_prev, k_node) + dist(k_node, i_next) + cost(k_node)

  return added_cost - removed_cost
```

2.6 Intra-route Moves (internal)

Modify the order of nodes/edges within the cycle without changing the set of selected nodes.

Intra-Node (node exchange)

Swapping the positions of two nodes i and j in the cycle.

```
procedure calculateDeltaIntraNode(i, j, solution, inst)
    if i > j: swap(i, j)
    i_node, j_node = solution[i], solution[j]

    if nodes adjacent: //3 edges change
      if j == i + 1:
        i_prev = solution[i - 1 + n) % n]
        j_next = solution[(j + 1) % n]

        removed_cost = dist(i_prev, i_node) + dist(j_node, j_next) + dist(i_node,
    j_node)
        added_cost = dist(i_prev, j_node) + dist(i_node, j_next) + dist(j_node,
    i_node)

      else if (i == 0 && j == solution.size()-1):
        j_prev = solution[n-2]
        i_next = solution[1]

        removed_cost = dist(j_prev, j_node) + dist(i_node, i_next) + dist(j_node,
    i_node)
        added_cost = dist(j_prev, i_node) + dist(j_node, i_next) + dist(i_node,
    j_node)

```

```
    return added_cost - removed_cost

    // Standard case (4 edges)
    i_prev, j_prev = solution[(i - 1 + n) % n], solution[(j - 1 + n) % n]
    i_next, j_next = solution[(i + 1) % n], solution[(j + 1) % n]
    removed_cost = dist(i_prev, i_node) + dist(i_node, i_next) +
                   dist(k_prev, k_node) + dist(k_node, k_next)
    added_cost = dist(i_prev, k_node) + dist(k_node, i_next) +
                 dist(k_prev, i_node) + dist(i_node, k_next)

    return added_cost - removed_cost
```

Intra-Edge (edge exchange)

Removing two edges from the cycle, $(i, i+1)$ and $(j, j+1)$, and reconnecting the cycle with edges (i, j) and $(i+1, j+1)$. This requires reversing the order of all nodes on the path between $i+1$ and j.

```
procedure calculateDeltaIntraEdge(i, j, solution, inst)
    if i > j then swap(i, j)

    i_node, i_next = solution[i], solution[i+1]
    j_node, j_next = solution[j],  solution[(j+1) % size]

    if adjecent nodes then return 0

    removed_cost = dist(i_node, i_next) + dist(j_node, j_next)
    added_cost = dist(i_node, j_node) + dist(i_next, j_next)

  return added_cost - removed_cost
```

2.7 Starting Solutions

Two methods for generating initial solutions for the LS algorithm were tested:

Random Solution

Generated by randomly shuffling all n instance nodes and selecting the first $(n+1)/2$ nodes.

Greedy Solution

Using the best construction heuristic from previous labs. Based on our results we chose the regretNearestAny algorithm with regret weight = 0.5.

3 Experimental Results

3.1 TSPA: steepestNodesNNregret

Results: min: 69801 max: 75440 avg: 72010
Best path: [108, 69, 18, 159, 22, 146, 181, 34, 160, 48, 54, 177, 184, 84, 4, 112, 35, 131, 149, 65, 116, 43, 42, 5, 41, 193, 139, 68, 46, 115, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 154, 180, 53, 121, 100, 26, 86, 75, 101, 1, 97, 152, 2, 120, 44, 25, 16, 171, 175, 113, 56, 31, 78, 145, 179, 92, 129, 57, 55, 52, 185, 40, 196, 81, 90, 165, 106, 178, 14, 49, 102, 144, 62, 9, 148, 124, 94, 63, 79, 80, 176, 137, 23, 89, 183, 143, 0, 117, 93, 140]

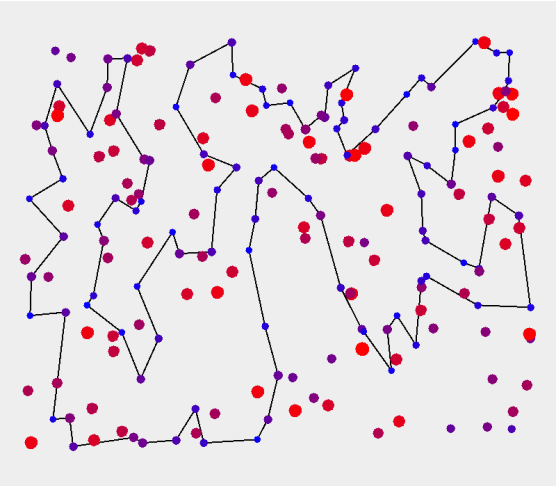

Figure 1: Best solution for steepestNodesNNregret, TSPA.

3.2 TSPA: steepestNodesRandom

Results: min: 80805 max: 95207 avg: 88109
Best path: [154, 53, 86, 75, 101, 1, 2, 120, 44, 25, 16, 171, 175, 113, 196, 81, 90, 165, 40, 185, 106, 178, 52, 55, 57, 92, 145, 31, 78, 129, 49, 14, 144, 68, 159, 22, 146, 193, 139, 198, 46, 115, 5, 42, 43, 131, 149, 123, 162, 94, 124, 148, 186, 23, 137, 151, 127, 112, 4, 10, 177, 54, 184, 84, 35, 65, 116, 41, 18, 108, 93, 117, 0, 143, 183, 89, 15, 62, 9, 167, 152, 97, 121, 63, 122, 79, 133, 80, 176, 51, 118, 59, 181, 34, 160, 70, 26, 100, 180, 135]

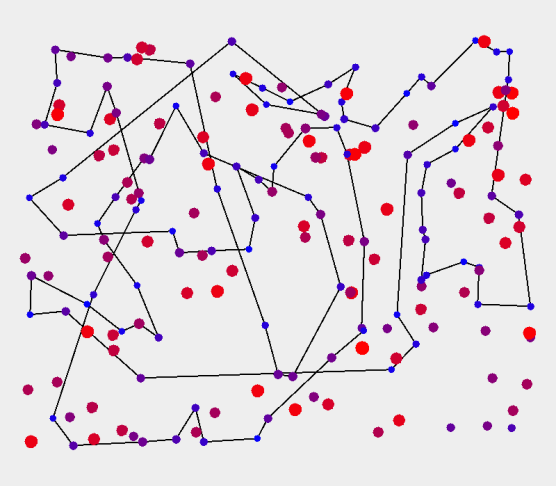

Figure 2: Best solution for steepestNodesRandom, TSPA.

3.3 TSPA: steepestEdgesNNregret

Results: min: 69788 max: 72702 avg: 70900

Best path: [78, 145, 179, 196, 81, 90, 165, 119, 40, 185, 55, 52, 106, 178, 49, 14, 144, 102, 62, 9, 148, 137, 23, 186, 89, 183, 143, 117, 0, 93, 140, 68, 46, 115, 139, 41, 193, 159, 69, 108, 18, 22, 146, 195, 181, 34, 160, 48, 54, 177, 10, 190, 4, 112, 84, 35, 184, 42, 43, 116, 65, 59, 118, 51, 176, 80, 94, 63, 79, 133, 151, 162, 123, 127, 70, 135, 154, 180, 53, 86, 100, 26, 97, 152, 1, 101, 75, 2, 129, 57, 92, 120, 44, 25, 16, 171, 175, 113, 56, 31]

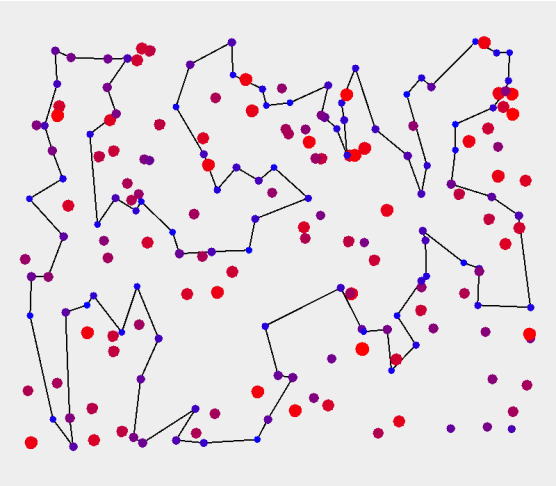

Figure 3: Best solution for steepestEdgesNNregret, TSPA.

3.4 TSPA: steepestEdgesRandom

Results: min: 72746 max: 79436 avg: 75098

Best path: [35, 184, 160, 42, 115, 59, 118, 65, 116, 43, 131, 149, 24, 127, 123, 135, 70, 154, 180, 158, 53, 86, 101, 97, 152, 1, 75, 2, 120, 44, 25, 129, 92, 145, 78, 16, 171, 175, 113, 31, 81, 90, 165, 119, 40, 185, 179, 57, 55, 52, 106, 178, 14, 49, 102, 144, 9, 62, 148, 124, 94, 122, 63, 79, 80, 133, 151, 162, 51, 176, 137, 23, 186, 114, 89, 183, 143, 0, 117, 108, 18, 69, 68, 46, 139, 41, 193, 159, 22, 146, 181, 34, 48, 54, 177, 10, 190, 4, 112, 84]

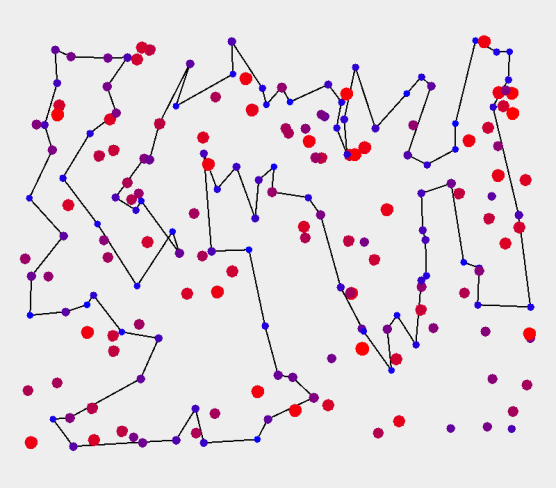

FIGURE 4: Best solution for steepestEdgesRandom, TSPA.

3.5 TSPA: greedyNodesNNregret

Results: min: 69801 max: 75440 avg: 72010

Best path: [108, 69, 18, 159, 22, 146, 181, 34, 160, 48, 54, 177, 184, 84, 4, 112, 35, 131, 149, 65, 116, 43, 42, 5, 41, 193, 139, 68, 46, 115, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 154, 180, 53, 121, 100, 26, 86, 75, 101, 1, 97, 152, 2, 120, 44, 25, 16, 171, 175, 113, 56, 31, 78, 145, 179, 92, 129, 57, 55, 52, 185, 40, 196, 81, 90, 165, 106, 178, 14, 49, 102, 144, 62, 9, 148, 124, 94, 63, 79, 80, 176, 137, 23, 89, 183, 143, 0, 117, 93, 140]

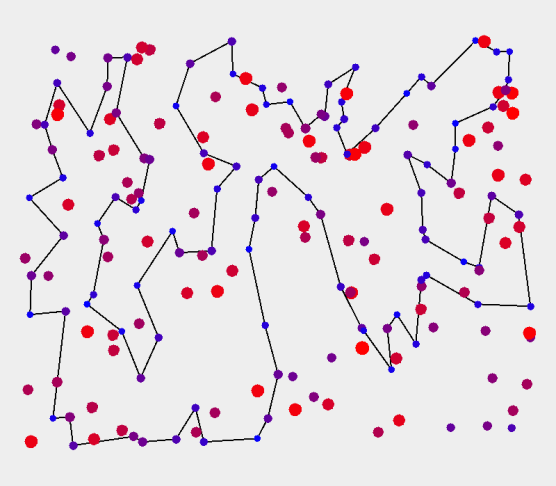

FIGURE 5: Best solution for greedyNodesNNregret, TSPA.

3.6 TSPA: greedyNodesRandom

Results: min: 79377 max: 95075 avg: 86057

Best path: [144, 102, 49, 14, 178, 106, 185, 40, 196, 179, 92, 129, 2, 152, 55, 52, 165, 90, 81, 145, 78, 120, 75, 86, 70, 112, 4, 84, 35, 131, 59, 118, 51, 79, 63, 180, 158, 53, 100, 121, 154, 135, 127, 123, 149, 65, 41, 193, 139, 46, 0, 143, 117, 108, 18, 22, 159, 5, 116, 43, 184, 190, 10, 177, 30, 54, 48, 160, 34, 181, 42, 115, 23, 186, 62, 9, 148, 167, 57, 31, 113, 175, 171, 16, 25, 44, 101, 1, 97, 26, 94, 122, 133, 162, 151, 80, 176, 137, 183, 89]

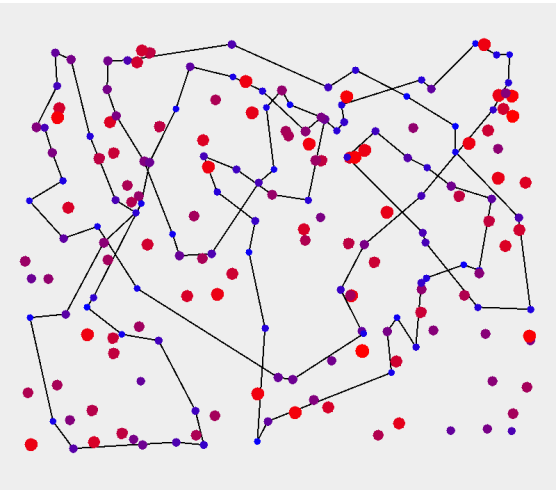

FIGURE 6: Best solution for greedyNodesRandom, TSPA.

3.7 TSPA: greedyEdgesNNregret

Results: min: 69817 max: 72750 avg: 71125

Best path: [193, 159, 69, 108, 18, 22, 146, 181, 34, 48, 54, 177, 10, 190, 4, 112, 84, 184, 160, 42, 43, 116, 65, 59, 118, 51, 176, 80, 79, 133, 151, 162, 123, 127, 70, 135, 154, 180, 63, 94, 53, 100, 26, 86, 75, 101, 1, 97, 152, 2, 129, 57, 92, 82, 120, 44, 25, 16, 171, 175, 113, 56, 31, 78, 145, 179, 196, 81, 90, 165, 40, 185, 55, 52, 106, 178, 49, 14, 144, 102, 62, 9, 148, 15, 114, 186, 137, 23, 89, 183, 143, 0, 117, 93, 140, 68, 46, 139, 115, 41]

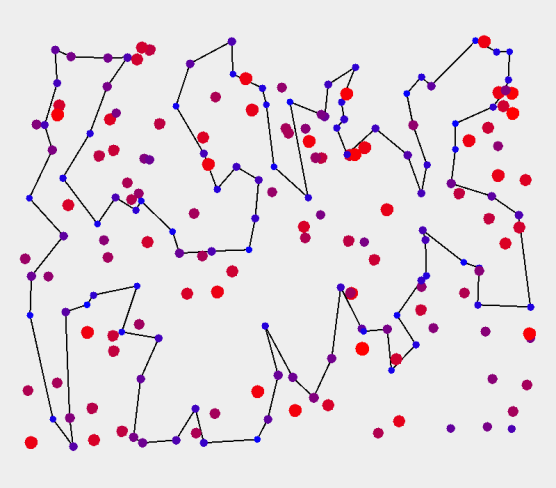

FIGURE 7: Best solution for greedyEdgesNNregret, TSPA.

3.8 TSPA: greedyEdgesRandom

Results: min: 72794 max: 78607 avg: 74771

Best path: [94, 124, 167, 9, 148, 62, 144, 102, 14, 49, 3, 106, 178, 52, 55, 57, 92, 179, 196, 185, 40, 165, 90, 81, 157, 31, 56, 113, 175, 171, 16, 145, 78, 25, 44, 120, 82, 129, 2, 1, 152, 97, 101, 75, 86, 53, 158, 154, 180, 135, 70, 123, 127, 112, 84, 4, 35, 184, 190, 10, 177, 30, 54, 48, 160, 34, 181, 146, 22, 108, 18, 159, 193, 41, 42, 43, 65, 116, 115, 139, 68, 46, 93, 117, 143, 0, 183, 89, 137, 176, 51, 118, 59, 162, 151, 133, 80, 79, 122, 63]

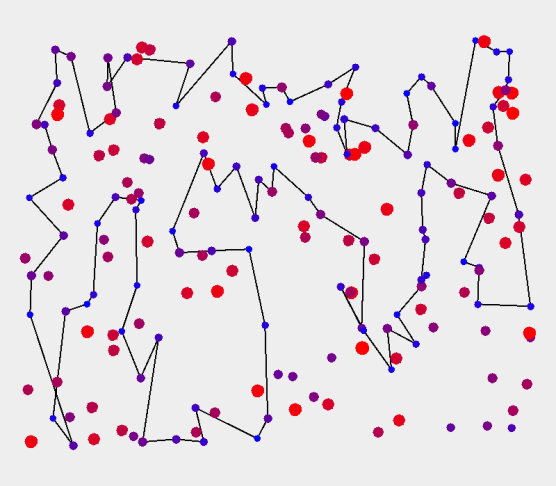

Figure 8: Best solution for greedyEdgesRandom, TSPA.

3.9 TSPB: steepestNodesNNregret

Results: min: 44488 max: 54391 avg: 47137

Best path: [131, 121, 51, 90, 147, 6, 188, 169, 132, 13, 195, 168, 145, 15, 70, 3, 155, 184, 152, 170, 34, 55, 18, 62, 124, 106, 128, 95, 130, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 57, 172, 179, 185, 86, 166, 194, 176, 113, 103, 127, 89, 163, 187, 153, 81, 77, 141, 91, 36, 61, 21, 82, 8, 104, 33, 160, 0, 35, 109, 29, 11, 138, 182, 25, 177, 5, 78, 175, 45, 80, 190, 136, 73, 54, 31, 193, 117, 198, 156, 1, 16, 27, 38, 135, 63, 100, 40, 107, 122]

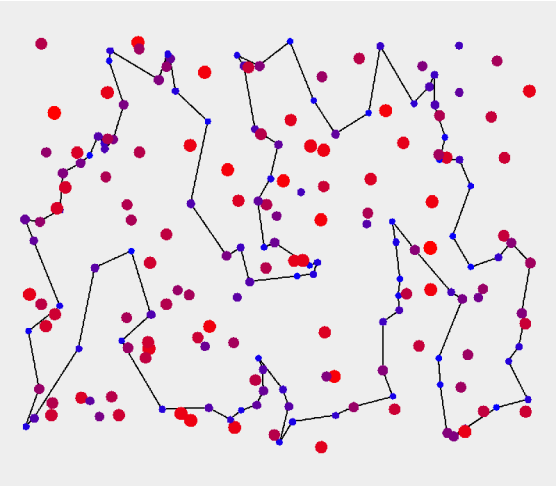

Figure 9: Best solution for steepestNodesNNregret, TSPB.

3.10 TSPB: steepestNodesRandom

Results: min: 55302 max: 70406 avg: 62972

Best path: [143, 111, 144, 104, 11, 139, 168, 195, 70, 3, 15, 145, 13, 132, 169, 188, 6, 147, 10, 40, 107, 133, 191, 51, 125, 90, 122, 63, 135, 190, 80, 162, 175, 78, 142, 177, 5, 45, 136, 193, 31, 54, 73, 187, 113, 194, 166, 22, 99, 130, 95, 185, 86, 124, 62, 18, 55, 34, 33, 138, 25, 117, 198, 27, 38, 1, 121, 8, 82, 21, 61, 36, 91, 141, 77, 81, 179, 94, 47, 148, 20, 28, 140, 183, 152, 155, 29, 160, 0, 109, 35, 153, 163, 103, 89, 127, 137, 114, 176, 106]

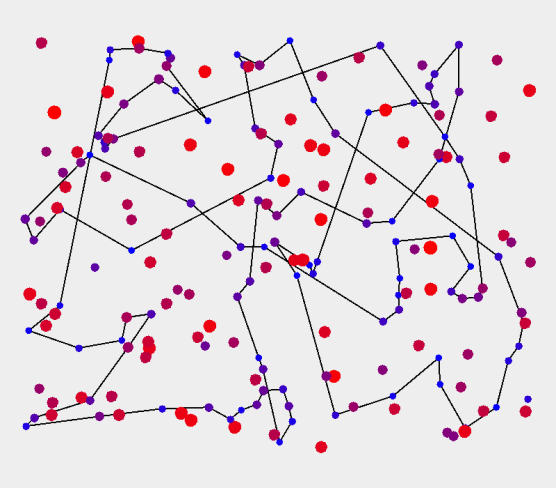

Figure 10: Best solution for steepestNodesRandom, TSPB.

3.11 TSPB: steepestEdgesNNregret

Results: min: 44355 max: 51997 avg: 46416

Best path: [133, 10, 147, 6, 188, 169, 132, 13, 168, 195, 145, 15, 70, 3, 155, 184, 152, 170, 34, 55, 18, 62, 124, 106, 128, 95, 130, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 57, 172, 179, 185, 86, 166, 194, 176, 113, 103, 127, 89, 163, 187, 153, 81, 77, 141, 91, 61, 36, 21, 82, 8, 104, 111, 0, 35, 109, 29, 160, 33, 11, 139, 138, 182, 25, 177, 5, 78, 175, 45, 80, 190, 136, 73, 54, 31, 193, 117, 198, 1, 131, 121, 51, 90, 122, 135, 63, 100, 40, 107]

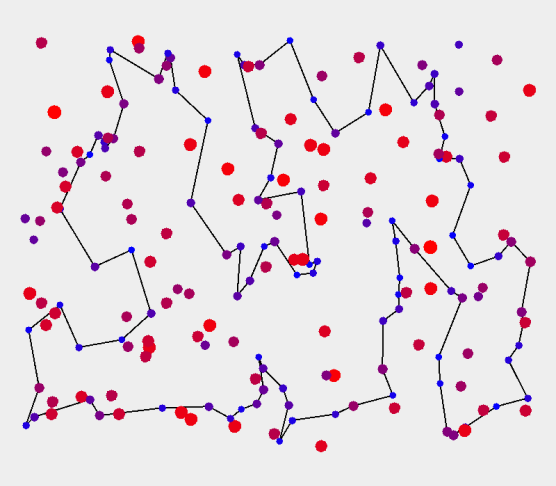

Figure 11: Best solution for steepestEdgesNNregret, TSPB.

3.12 TSPB: steepestEdgesRandom

Results: min: 46671 max: 52022 avg: 49275

Best path: [139, 195, 168, 13, 145, 15, 3, 70, 132, 188, 169, 6, 147, 71, 51, 125, 121, 131, 90, 191, 178, 10, 133, 122, 100, 107, 40, 63, 102, 135, 38, 1, 117, 54, 73, 31, 164, 193, 80, 190, 45, 142, 175, 78, 5, 177, 25, 182, 138, 104, 8, 111, 82, 21, 61, 36, 91, 141, 97, 77, 81, 153, 163, 89, 127, 103, 113, 176, 194, 166, 86, 185, 130, 95, 99, 179, 172, 94, 66, 47, 148, 60, 20, 28, 140, 183, 34, 18, 55, 62, 106, 124, 143, 35, 109, 29, 0, 160, 33, 11]

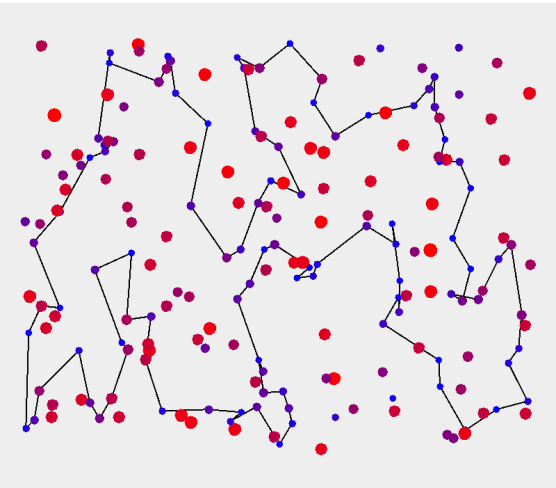

Figure 12: Best solution for steepestEdgesRandom, TSPB.

3.13 TSPB: greedyNodesNNregret

Results: min: 44456 max: 52768 avg: 47114

Best path: [122, 133, 107, 40, 63, 135, 38, 27, 16, 1, 156, 198, 117, 193, 31, 54, 73, 136, 190, 80, 45, 175, 78, 5, 177, 25, 182, 138, 11, 29, 109, 35, 0, 160, 33, 104, 8, 82, 21, 61, 36, 91, 141, 77, 81, 153, 187, 163, 89, 127, 103, 113, 176, 194, 166, 86, 185, 179, 172, 57, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 130, 95, 128, 106, 124, 62, 18, 55, 34, 170, 152, 184, 155, 3, 70, 15, 145, 168, 195, 13, 132, 169, 188, 6, 147, 90, 51, 121, 131]

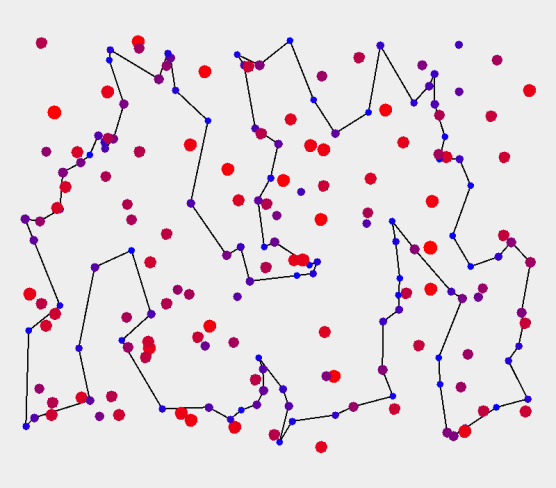

FIGURE 13: Best solution for greedyNodesNNregret, TSPB.

3.14 TSPB: greedyNodesRandom

Results: min: 53632 max: 68594 avg: 60670

Best path: [22, 179, 66, 94, 47, 149, 4, 140, 183, 55, 18, 62, 194, 113, 180, 176, 153, 77, 141, 175, 80, 190, 136, 73, 198, 156, 27, 38, 102, 63, 135, 131, 125, 51, 121, 54, 31, 193, 117, 1, 122, 133, 10, 90, 191, 147, 188, 169, 132, 168, 11, 33, 160, 12, 0, 35, 111, 81, 163, 89, 127, 103, 166, 86, 148, 60, 20, 28, 109, 29, 144, 104, 8, 82, 87, 21, 91, 61, 36, 177, 25, 182, 138, 139, 195, 13, 145, 15, 70, 3, 155, 152, 34, 124, 106, 128, 95, 130, 185, 99]

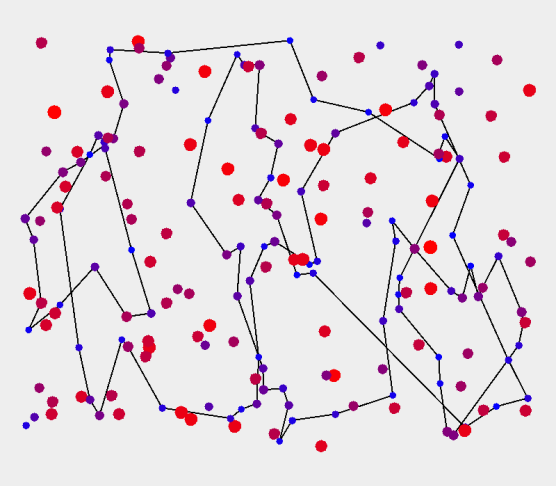

Figure 14: Best solution for greedyNodesRandom, TSPB.

3.15 TSPB: greedyEdgesNNregret

Results: min: 44355 max: 51997 avg: 46450

Best path: [122, 135, 63, 100, 40, 107, 133, 10, 147, 6, 188, 169, 132, 13, 168, 195, 145, 15, 70, 3, 155, 184, 152, 170, 34, 55, 18, 62, 124, 106, 128, 95, 130, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 57, 172, 179, 185, 86, 166, 194, 176, 113, 103, 127, 89, 163, 187, 153, 81, 77, 141, 91, 61, 36, 21, 82, 8, 104, 111, 0, 35, 109, 29, 160, 33, 11, 139, 138, 182, 25, 177, 5, 78, 175, 45, 80, 190, 136, 73, 54, 31, 193, 117, 198, 1, 131, 121, 51, 90]

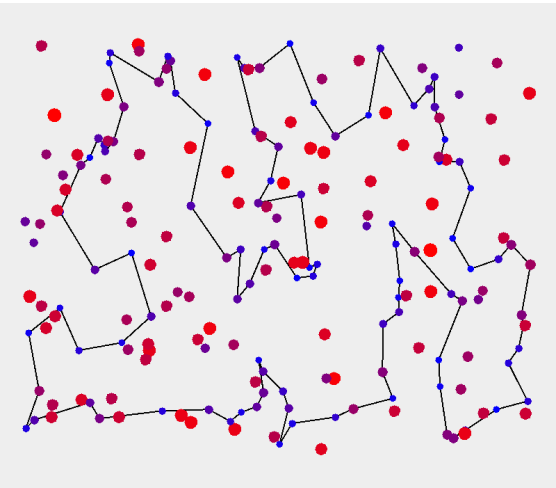

Figure 15: Best solution for greedyEdgesNNregret, TSPB.

3.16 TSPB: greedyEdgesRandom

Results: min: 46673 max: 52786 avg: 49354

Best path: [47, 94, 179, 185, 130, 95, 86, 166, 194, 176, 180, 113, 114, 127, 165, 89, 103, 163, 153, 106, 143, 124, 62, 18, 55, 34, 109, 35, 0, 29, 33, 160, 11, 139, 182, 138, 104, 8, 56, 144, 111, 81, 77, 82, 21, 141, 91, 61, 36, 5, 78, 175, 45, 190, 80, 136, 73, 164, 54, 31, 193, 117, 198, 156, 1, 38, 131, 121, 51, 90, 122, 135, 63, 107, 40, 133, 10, 147, 6, 134, 188, 169, 132, 168, 195, 13, 70, 3, 15, 145, 189, 155, 184, 152, 170, 183, 140, 28, 20, 148]

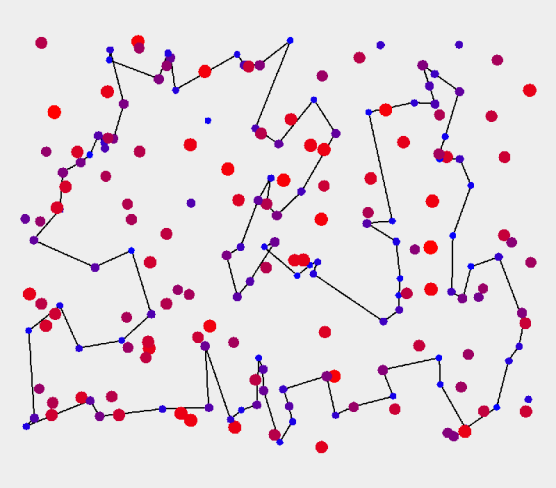

FIGURE 16: Best solution for greedyEdgesRandom, TSPB.

Table 1: Results summary.

TSPA

Method	Average (Avg)	Min	Max
Lab 1: Simple Heuristics			
Random Solution	262995	237322	291386
Nearest End	85109	83182	89433
Nearest Any	73179	71179	75450
Greedy Cycle	72646	71488	74410
Lab 2: Regret Heuristics			
regretNN (w=1.0)	117138	108151	124921
regretGC (w=1.0)	115579	105692	126951
NN Regret (w=0.5)	72401	70010	75452
GC Regret (w=0.5)	72130	71108	73395
Lab 3: LS (Start: Greedy)			
G: Steepest+Node	72010	69801	75440
G: Steepest+Edge	70900	69788	72702
G: Greedy+Node	72010	69801	75440
G: Greedy+Edge	71125	69817	72750
Lab 3: LS (Start: Random)			
R: Steepest+Node	88109	80805	95207
R: Steepest+Edge	75098	72746	79436
R: Greedy+Node	86057	79377	95075
R: Greedy+Edge	74771	72794	78607

TSPB

Method	Average (Avg)	Min	Max
Lab 1: Simple Heuristics			
Random Solution	212627	185591	240630
Nearest End	54390	52319	59030
Nearest Any	45870	44417	53438
Greedy Cycle	51401	49001	57324
Lab 2: Regret Heuristics			
regretNN (w=1.0)	74444	69933	80278
regretGC (w=1.0)	72740	67809	78406
NN Regret (w=0.5)	47664	44891	55247
GC Regret (w=0.5)	50897	47144	55700
Lab 3: LS (Start: Greedy)			
G: Steepest+Node	47137	44488	54391
G: Steepest+Edge	46416	44355	51997
G: Greedy+Node	47114	44456	52768
G: Greedy+Edge	46450	44355	51997
Lab 3: LS (Start: Random)			
R: Steepest+Node	62972	55302	70406
R: Steepest+Edge	49275	46671	52022
R: Greedy+Node	60670	53632	68594
R: Greedy+Edge	49354	46673	52786

Table 2: Execution times (in milliseconds) summary.

TSPA

Method	Average (Avg)	Min	Max
Lab 1 & 2 (Construction only)			
NN Regret (w=0.5)	53,77	42,92	127,09
Random	0,06	0,02	1,87
Lab 3: LS (Start: Greedy)			
G: Steepest+Node	2,26	0,46	60,03
G: Steepest+Edge	3,01	0,40	24,64
G: Greedy+Node	4,26	0,91	55,56
G: Greedy+Edge	8,13	0,87	46,97
Lab 3: LS (Start: Random)			
R: Steepest+Node	75,49	55,15	352,30
R: Steepest+Edge	55,12	44,83	115,60
R: Greedy+Node	270,87	214,67	808,90
R: Greedy+Edge	252,01	208,78	736,58

TSPB

Method	Average (Avg)	Min	Max
Lab 1 & 2 (Construction only)			
NN Regret (w=0.5)	60,60	42,03	277,95
Random	0,06	0,03	1,66
Lab 3: LS (Start: Greedy)			
G: Steepest+Node	5,03	1,33	35,92
G: Steepest+Edge	4,79	1,05	17,07
G: Greedy+Node	9,47	2,70	65,39
G: Greedy+Edge	10,99	2,32	40,73
Lab 3: LS (Start: Random)			
R: Steepest+Node	85,70	55,29	287,95
R: Steepest+Edge	65,75	45,59	187,54
R: Greedy+Node	302,43	199,49	671,79
R: Greedy+Edge	285,46	206,53	530,24

4 Conclusions

Based on the experimental results and data analysis presented in this report, the following key conclusions can be drawn:

- The use of an initial solution generated by the regret heuristic resulted in substantially

better performance compared to solutions initialized randomly. This confirms the critical dependency of Local Search (LS) algorithms on the quality of the starting solution.

- The Edge-based intra-move neighborhood led to slighlty better solutions than the Node-based one.
- The Greedy Search algorithm executed out to be unexpectedly slow, particularly when initialized with a random solution. However, this anomaly may be due to an implementation overhead, specifically the creation and shuffling of the complete neighborhood list in every LS iteration, rather than generating moves after picking the order of neighbourhood.

Solution Checker: All best solutions obtained were verified using the provided solution checker.
Source Code: https://github.com/PBalewski/EvolutionaryComputation/tree/main/lab3