



---

# POZNAN UNIVERSITY OF TECHNOLOGY

---

FACULTY OF COMPUTING AND TELECOMMUNICATION  
Institute of Computing Science

## LABORATORY 4: LOCAL SEARCH WITH CANDIDATE MOVES

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 Local Search with Candidate Moves

The goal of this assignment is to enhance the time efficiency of the Steepest Local Search (LS) by restricting the search to a small subset of candidate moves instead of searching through the whole neighbourhood.

### 2.1 Candidates Definition

For each node  $i$ , a list of  $K = 10$  "nearest" candidate nodes is pre-calculated. The score is determined by the sum of the edge length between the nodes and the new node's cost:

$$\text{Score}(i, j) = \text{dist}(i, j) + \text{cost}(j)$$

During the LS iterations only moves that introduce at least one candidate edge into the solution are evaluated.

### 2.2 Inter-route Moves (Node Exchange)

A swap of a node already in the cycle  $i$  with a node outside of it  $k$  is considered a candidate if the newly introduced edge  $(prev_i, k)$  or  $(k, next_i)$  is a candidate edge, where  $prev_i$  and  $next_i$  are the predecessor and successor of  $i$  in the current cycle. The implemented strategy iterates over the candidate list of  $prev_i$  and  $next_i$  to find suitable  $k$  nodes.

### 2.3 Intra-route Moves (Edge Exchange)

An Intra-route move (reversing a subpath) is considered a candidate if one of the two newly introduced edges,  $(i, j)$  or  $(i + 1, j + 1)$ , is a candidate edge. The implementation iterates over the candidate list of  $i$  and  $i + 1$  to find feasible indices  $j$  or  $j + 1$  within the cycle.

The full procedure looks following:

```

1 procedure steepestSearch (selected, remaining, inst, candidate_list)
2   bestDelta , bestMove, n = 0, null, selected.size
3
4   // 1. Inter-route: Node exchange
5   for (i_idx=0; i_idx<n; i_idx++)
6     prev_i = selected.get((i_idx-1+n) %n)
7     next_i = selected.get((i_idx+1) %n)
8
9     // Check if candidate edge: prev_i -> k_node
10    for (k_node : candidate_list[prev_i])
11      if (remaining.contains(k_node))
12        delta = calculateDeltaInterNode(selected, i_idx, k_node, inst)
13        if delta < betDelta
14          //update bestDelta and formulate bestMove
15
16    // Check if candidate edge: k_node -> next_i
17    for (k_node : candidate_list[next_i])
18      if (remaining.contains(k_node))
19        delta = calculateDeltaInterNode(selected, i_idx, k_node, inst)
20        if delta < betDelta
21          //update bestDelta and formulate bestMove
22
23  // 2. Intra-route: Edge exchange
24  for (i_idx=0; i_idx<n; i_idx++)
25    i_node = selected.get(i_idx);
26    next_i = selected.get((i_idx+1) %n);
27
28    // Check if candidate edge: i -> j
29    for (j_node : candidate_list[i_node])
30      j_idx = selected.indexOf(j_node)
31      if j_idx in selected
32        delta = calculateDeltaIntraEdge(selected, i_idx, j_idx, inst);
33        if (delta < best_delta)
34          //update bestDelta and formulate bestMove
35
36    // Check if candidate edge: i+1 -> j+1
37    for (j_next : candidate_list[next_i])
38      j_next_idx = selected.indexOf(j_next)
39      if j_next_idx in selected
40        j_idx = (j_next_idx - 1 + n) % n; //find original j_idx
41        delta = calculateDeltaIntraEdge(selected, i_idx, j_idx, inst);
42        if (delta < best_delta)
43          //update bestDelta and formulate bestMove
44
45  if bestMove != null
46    //apply bestMove
47    return true
48  return false

```

### 3 Experimental Results

#### 3.1 TSPA: steepestCandidate

**Results:** min: 73814 max: 85046 avg: 77435

**Best path:** [31, 56, 113, 175, 171, 16, 78, 44, 120, 2, 75, 86, 101, 1, 152, 97, 100, 121, 124, 94, 63, 79, 80, 176, 66, 51, 118, 59, 162, 151, 133, 180, 53, 158, 154, 135, 70, 127, 123, 112, 4, 84, 190, 10, 177, 54, 34, 160, 184, 42, 5, 43, 116, 65, 197, 115, 46, 198, 139, 41, 193, 159, 22, 18, 69, 108, 140, 93, 117, 0, 143, 183, 89, 23, 137, 186, 114, 15, 148, 9, 62, 49, 178, 14, 144, 21, 164, 90, 165, 119, 40, 185, 106, 52, 55, 57, 92, 145, 196, 81]

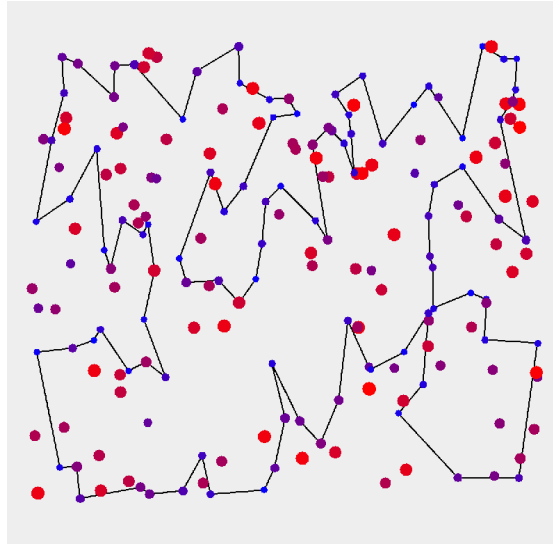


FIGURE 1: Best solution for steepestCandidate, TSPA.

#### 3.2 TSPB: steepestCandidate

**Results:** min: 45856 max: 51365 avg: 48464

**Best path:** [103, 127, 89, 163, 153, 81, 77, 141, 61, 36, 5, 78, 175, 142, 45, 80, 190, 136, 73, 54, 31, 193, 117, 198, 1, 16, 27, 38, 102, 63, 135, 122, 131, 121, 25, 177, 21, 8, 104, 144, 160, 33, 138, 11, 139, 134, 74, 118, 98, 51, 191, 90, 133, 10, 147, 6, 188, 169, 132, 161, 70, 3, 15, 145, 13, 195, 168, 29, 0, 109, 35, 106, 124, 62, 18, 55, 34, 170, 152, 183, 140, 28, 20, 60, 148, 47, 94, 66, 172, 179, 22, 99, 130, 95, 185, 86, 166, 194, 176, 113]

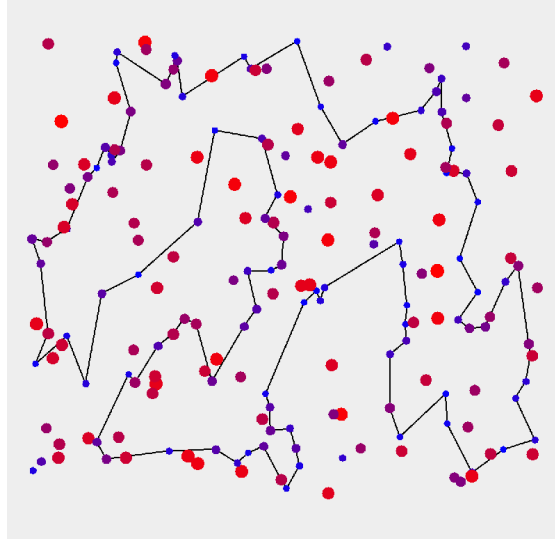


FIGURE 2: Best solution for steepestCandidate, TSPB.

TABLE 1: Results summary.

TSPA				TSPB			
Method	Average (Avg)	Min	Max	Method	Average (Avg)	Min	Max
<i>Lab 1: Simple Heuristics</i>				<i>Lab 1: Simple Heuristics</i>			
Random Solution	262995	237322	291386	Random Solution	212627	185591	240630
Nearest End	85109	83182	89433	Nearest End	54390	52319	59030
Nearest Any	73179	71179	75450	Nearest Any	45870	44417	53438
Greedy Cycle	72646	71488	74410	Greedy Cycle	51401	49001	57324
<i>Lab 2: Regret Heuristics</i>				<i>Lab 2: Regret Heuristics</i>			
regretNN (w=1.0)	117138	108151	124921	regretNN (w=1.0)	74444	69933	80278
regretGC (w=1.0)	115579	105692	126951	regretGC (w=1.0)	72740	67809	78406
NN Regret (w=0.5)	72401	70010	75452	NN Regret (w=0.5)	47664	44891	55247
GC Regret (w=0.5)	72130	71108	73395	GC Regret (w=0.5)	50897	47144	55700
<i>Lab 3: LS (Start: Greedy)</i>				<i>Lab 3: LS (Start: Greedy)</i>			
G: Steepest+Node	72010	69801	75440	G: Steepest+Node	47137	44488	54391
G: Steepest+Edge	70900	69788	72702	G: Steepest+Edge	46416	44355	51997
G: Greedy+Node	72010	69801	75440	G: Greedy+Node	47114	44456	52768
G: Greedy+Edge	71125	69817	72750	G: Greedy+Edge	46450	44355	51997
<i>Lab 3 &amp; 4: LS (Start: Random)</i>				<i>Lab 3 &amp; 4: LS (Start: Random)</i>			
R: Steepest+Node	88109	80805	95207	R: Steepest+Node	62972	55302	70406
R: Steepest+Edge	75098	72746	79436	R: Steepest+Edge	49275	46671	52022
R: Greedy+Node	86057	79377	95075	R: Greedy+Node	60670	53632	68594
R: Greedy+Edge	74771	72794	78607	R: Greedy+Edge	49354	46673	52786
R: Steepest+Candidate	77435	73814	85046	R: Steepest+Candidate	48464	45856	51365

TABLE 2: Execution times (in milliseconds) summary.

TSPA				TSPB			
Method	Average (Avg)	Min	Max	Method	Average (Avg)	Min	Max
<i>Lab 3 &amp; 4 (Construction only)</i>				<i>Lab 3 &amp; 4 (Construction only)</i>			
NN Regret (w=0.5)	53,77	42,92	127,09	NN Regret (w=0.5)	60,60	42,03	277,95
Random	0,05	0,01	2,56	Random	0,20	0,04	6,44
Candidate list	19,03	19,03	19,03	Candidate list	37,24	37,24	37,24
<i>Lab 3: LS (Start: Greedy)</i>				<i>Lab 3: LS (Start: Greedy)</i>			
G: Steepest+Node	2,26	0,46	60,03	G: Steepest+Node	5,03	1,33	35,92
G: Steepest+Edge	3,01	0,40	24,64	G: Steepest+Edge	4,79	1,05	17,07
G: Greedy+Node	4,26	0,91	55,56	G: Greedy+Node	9,47	2,70	65,39
G: Greedy+Edge	8,13	0,87	46,97	G: Greedy+Edge	10,99	2,32	40,73
<i>Lab 3 &amp; 4: LS (Start: Random)</i>				<i>Lab 3: LS (Start: Random)</i>			
R: Steepest+Node	75,49	55,15	352,30	R: Steepest+Node	85,70	55,29	287,95
R: Steepest+Edge	55,12	44,83	115,60	R: Steepest+Edge	65,75	45,59	187,54
R: Greedy+Node	270,87	214,67	808,90	R: Greedy+Node	302,43	199,49	671,79
R: Greedy+Edge	252,01	208,78	736,58	R: Greedy+Edge	285,46	206,53	530,24
R: Steepest+Candidate	19,70	16,22	88,30	R: Steepest+Candidate	47,16	25,77	130,19

## 4 Conclusions

The introduction of Candidate Moves (using  $K = 10$ ) successfully achieved the goal of improving time efficiency compared to the full neighborhood baseline, confirming that iterating over a restricted, highly relevant neighborhood is essential for speedup. The drastic speedup caused however a minimal loss in solution quality. For TSPA, the average cost increased slightly (from 75098 to 77435). This small trade-off is often acceptable in practice, as the solution quality remains high while computation time is reduced by an order of magnitude.

---

**Solution Checker:** All best solutions obtained were verified using the provided solution checker.

**Source Code:** <https://github.com/PBalewski/EvolutionaryComputation/tree/main/lab3>