Assignment 7 - Large Neighborhood Search for Selective TSP

Authors

- Mateusz Idziejczak 155842
- Mateusz Stawicki 155900

Github

> https://github.com/Luncenok/EvolutionaryComputing

Problem Description

This is the same variant of the Traveling Salesman Problem as in previous assignments:

- Select exactly 50% of nodes (rounded up if odd)
- Form a Hamiltonian cycle through selected nodes
- Minimize: total path length + sum of selected node costs
- Distances are Euclidean distances rounded to integers

Instances:

- **TSPA, TSPB** with 200 nodes, selecting 100 nodes.

Goal

Implement Large Neighborhood Search (LNS) in two versions:

1. **LNS with Local Search**: Apply local search after each destroy-repair iteration
2. **LNS without Local Search**: Only apply destroy-repair without subsequent local search

For both versions:

- Use a random starting solution with local search applied initially
- Run **20 times** per instance
- Use a **time limit equal to the average MSLS time** from Assignment 6
- Report the number of main loop iterations

Algorithm Pseudocode

Large Neighborhood Search (LNS)

LNS iteratively destroys and repairs a solution to explore larger neighborhoods.

```
LNS(timeLimit, useLocalSearch=True):
    # Initialize with random solution
    current = generateRandomSolution()
```

```
    current = localSearch(current)  # Always apply to initial

    best = current
    bestObjective = objective(current)

    startTime = now()
    iterations = 0

    while (now() - startTime) < timeLimit:
        iterations += 1

        # Destroy: remove ~30% of nodes
        partial = destroy(current)

        # Repair: rebuild using greedy heuristic
        repaired = repair(partial)

        # Optional local search
        if useLocalSearch:
            candidate = localSearch(repaired)
        else:
            candidate = repaired

        obj = objective(candidate)

        # Update best
        if obj < bestObjective:
            bestObjective = obj
            best = candidate

        # Accept if better (greedy acceptance)
        if obj < objective(current):
            current = candidate

    return best, iterations
```

Destroy Operator

The destroy operator removes approximately 30% of nodes from the current solution using weighted random selection. Nodes connected by longer edges have higher probability of removal:

```
destroy(solution, destroyFraction=0.30):
    numToRemove = solSize * destroyFraction

    # Calculate weights based on adjacent edge costs
    weights = []
    for i in range(len(solution)):
        prev = (i - 1) % len(solution)
        next = (i + 1) % len(solution)
        edgeCost = distance[solution[prev]][solution[i]] +
distance[solution[i]][solution[next]]
```

```
            weight = edgeCost + costs[solution[i]]
            weights.append(weight)

        # Weighted random selection of nodes to remove
        toRemove = weightedRandomSelect(numToRemove, weights)

        # Return remaining nodes (preserving tour order)
        return [node for i, node in enumerate(solution) if i not in toRemove]
```

Destroy rationale:

- Weighted selection targets "bad" edges (longer connections) and costly nodes
- 30% removal provides significant diversification while retaining solution structure
- Preserving tour order helps maintain good partial structures
- Randomization prevents deterministic cycling

Repair Operator

The repair operator rebuilds the solution to full size using the weighted 2-regret heuristic (the best-performing greedy heuristic from previous assignments):

```
repair(partial, selectCount):
    solution = partial

    while len(solution) < selectCount:
        bestNode = None
        bestPos = None
        bestScore = -infinity

        for each unselected node:
            # Find best and second-best insertion positions
            best1, best2 = findTwoBestInsertions(node, solution)

            regret = best2 - best1
            score = wRegret * regret - wBest * best1

            if score > bestScore:
                bestScore = score
                bestNode = node
                bestPos = bestInsertPosition

        solution.insert(bestPos, bestNode)

    return solution
```

Repair rationale:

- Weighted 2-regret achieves best results among greedy heuristics
- Balances urgency (regret) with quality (insertion cost)
- Efficiently rebuilds while maintaining good tour quality

Experimental Setup

- **Instances**: TSPA, TSPB (200 nodes, 100 selected)
- **Objective**: Minimize path length + sum of selected node costs
- **Local search**: Steepest descent with edge exchange (from Assignment 3)
- **Destroy fraction**: 30% of nodes removed
- **Repair heuristic**: Weighted 2-regret (w_regret=1.0, w_best=1.0)
- **Evaluation**:
  - Run both LNS versions **20 times** per instance
  - Use **time limit = average MSLS time** (~1090 ms)
  - Report min, max, and average objective values and running times
  - Report average number of destroy-repair iterations

Key Results

Summary Comparison

Instance	ILS Avg	LNS+LS Avg	LNS Avg	LNS+LS vs ILS	LNS vs ILS
TSPA	69340	69689	69817	+0.50%	+0.69%
TSPB	43674	44239	44294	+1.29%	+1.42%

Iteration Count Table

Instance	LNS with LS	LNS without LS	ILS (LS runs)
TSPA	1476.9	1683.4	3544.5
TSPB	1438.95	1620.15	3566.7

Comparison with All Previous Methods

Method	TSPA	TSPB
Random	264638 (238611 – 287962)	213875 (190076 – 244960)
Nearest Neighbor (end only)	85108 (83182 – 89433)	54390 (52319 – 59030)
Nearest Neighbor (any position)	73178 (71179 – 75450)	45870 (44417 – 53438)
Greedy Cycle	72646 (71488 – 74410)	51400 (49001 – 57324)
Greedy 2-Regret	115474 (105852 – 123428)	72454 (66505 – 77072)
Greedy Weighted (2-Regret + BestDelta)	72129 (71108 – 73395)	50950 (47144 – 55700)
Nearest Neighbor Any 2-Regret	116659 (106373 – 126570)	73646 (67121 – 79013)
Nearest Neighbor Any Weighted	72401 (70010 – 75452)	47653 (44891 – 55247)
LS Random + Steepest + Nodes	88011 (81817 – 97630)	62848 (55928 – 70479)
LS Random + Greedy + Nodes	93267 (86375 – 101454)	65388 (57842 – 76707)

Method	TSPA	TSPB
LS Random + Greedy + Edges	81101 (76362 – 87763)	54088 (50858 – 59045)
LS Greedy + Steepest + Nodes	71614 (70626 – 72950)	45414 (43826 – 50876)
LS Greedy + Steepest + Edges	71460 (70510 – 72614)	44979 (43921 – 50629)
LS Greedy + Greedy + Nodes	71908 (71093 – 73048)	45584 (43917 – 51165)
LS Greedy + Greedy + Edges	71825 (70977 – 72706)	45376 (43845 – 51170)
LS Random + Steepest + Edges	73965 (71371 – 78984)	48252 (45823 – 51965)
LM Random + Steepest + Edges	74981 (72054 – 79520)	49325 (45965 – 52805)
Candidates (k=5)	84726 (78843 – 91459)	49873 (47117 – 53865)
Candidates (k=10)	77773 (72851 – 84000)	48450 (45669 – 51178)
Candidates (k=15)	75510 (72276 – 83040)	48295 (45582 – 51938)
Candidates (k=20)	74416 (71292 – 80264)	48221 (45338 – 51285)
LM Candidates (k=10)	75157 (72331 – 80832)	49219 (46145 – 52021)
LM Candidates (k=20)	74976 (72054 – 79520)	49302 (45965 – 52805)
MSLS (200 iterations)	71306 (70748 – 71959)	45741 (45356 – 46168)
ILS	69340 (69107 – 69861)	43674 (43473 – 44056)
LNS with LS	**69689 (69185 – 70194)**	**44239 (43566 – 45964)**
LNS without LS	**69817 (69496 – 70217)**	**44294 (43630 – 45602)**

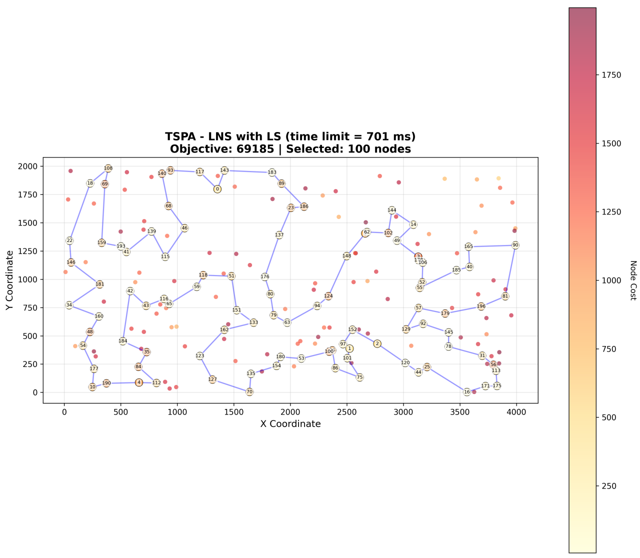
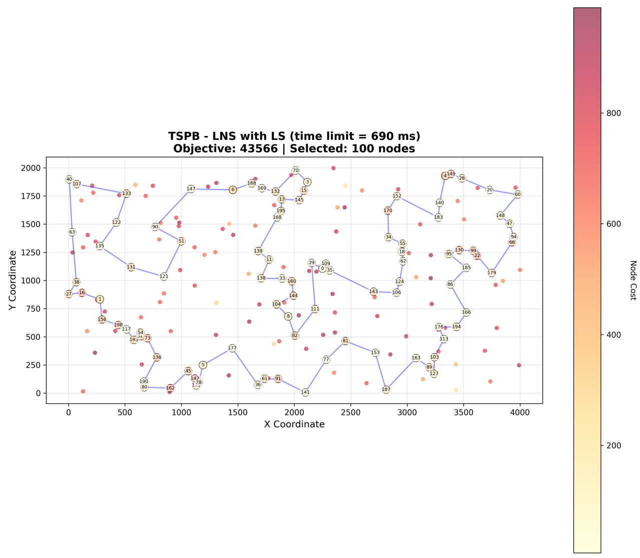
Running Times (ms)

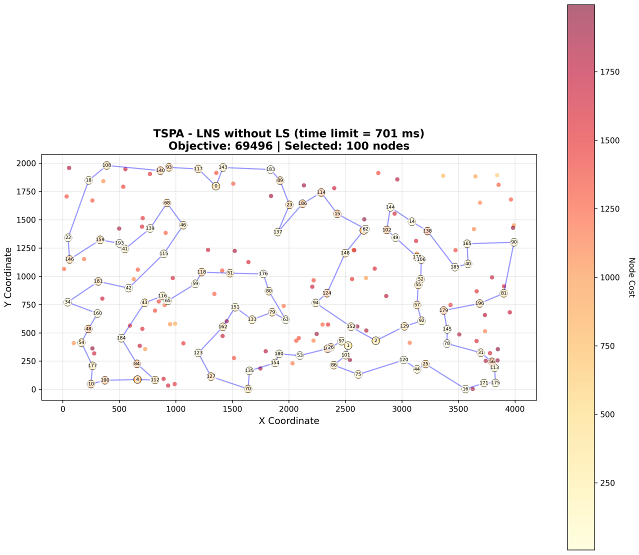
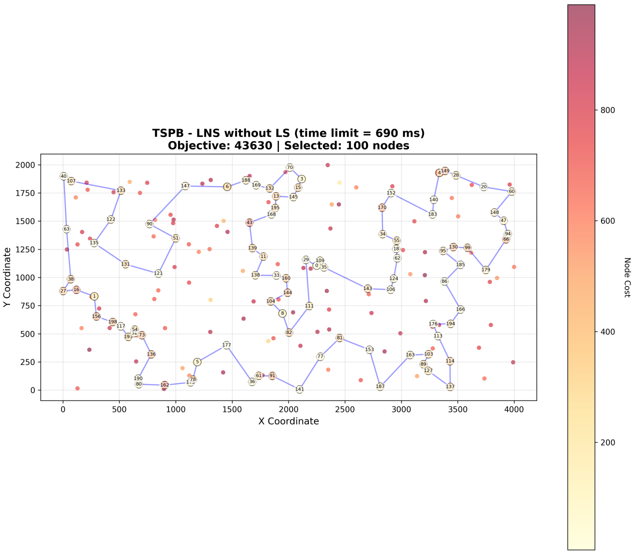
Method	TSPA	TSPB
Random	0.0001 (0.00004 – 0.003)	0.00003 (0.00 – 0.0001)
Nearest Neighbor (end only)	0.0198 (0.0149 – 0.063)	0.0204 (0.0152 – 0.062)
Nearest Neighbor (any position)	0.878 (0.691 – 2.102)	0.720 (0.692 – 0.862)
Greedy Cycle	0.683 (0.654 – 0.967)	0.681 (0.657 – 0.796)
Greedy 2-Regret	0.952 (0.920 – 1.263)	0.941 (0.920 – 1.057)
Greedy Weighted (2-Regret + BestDelta)	0.957 (0.920 – 1.302)	0.946 (0.918 – 1.052)
Nearest Neighbor Any 2-Regret	0.893 (0.852 – 1.175)	1.105 (0.844 – 21.87)
Nearest Neighbor Any Weighted	0.905 (0.857 – 1.405)	0.894 (0.852 – 1.287)
LS Random + Steepest + Nodes	6.297 (4.872 – 66.19)	5.700 (4.538 – 7.177)
LS Random + Greedy + Nodes	4.173 (2.613 – 7.520)	3.813 (2.502 – 7.189)

Method	TSPA	TSPB
LS Random + Greedy + Edges	3.375 (2.184 – 8.683)	3.275 (2.243 – 5.501)
LS Greedy + Steepest + Nodes	1.188 (0.993 – 3.055)	0.937 (0.803 – 1.608)
LS Greedy + Steepest + Edges	1.139 (1.000 – 1.856)	0.915 (0.808 – 1.441)
LS Greedy + Greedy + Nodes	2.840 (2.182 – 4.170)	2.556 (1.909 – 4.489)
LS Greedy + Greedy + Edges	2.877 (2.169 – 5.164)	2.628 (1.967 – 4.344)
LS Random + Steepest + Edges	3.356 (2.852 – 3.959)	3.515 (2.990 – 5.310)
LM Random + Steepest + Edges	3.858 (2.957 – 5.073)	3.949 (3.062 – 10.59)
Candidates (k=5)	8.697 (7.449 – 9.838)	9.417 (8.245 – 10.69)
Candidates (k=10)	10.03 (8.757 – 11.80)	11.05 (9.389 – 49.22)
Candidates (k=15)	12.03 (9.876 – 50.62)	12.01 (10.41 – 13.79)
Candidates (k=20)	12.55 (11.18 – 14.29)	13.30 (11.60 – 16.14)
LM Candidates (k=10)	4.316 (3.698 – 5.795)	4.124 (3.478 – 4.605)
LM Candidates (k=20)	15.04 (12.35 – 43.77)	14.85 (12.92 – 17.07)
MSLS (200 iterations)	701.38 (665.26 – 788.26)	690.25 (672.52 – 749.00)
ILS	701.46 (701.38 – 701.57)	690.37 (690.25 – 690.60)
LNS with LS	**701.59 (701.38 – 701.84)**	**690.52 (690.32 – 690.67)**
LNS without LS	**701.54 (701.39 – 701.76)**	**690.40 (690.26 – 690.66)**

Visualizations

Best solutions found by LNS visualized on both instances:

LNS with LS - TSPA	LNS with LS - TSPB
	

LNS without LS - TSPA	LNS without LS - TSPB
	

Analysis and Conclusions

LNS vs ILS Comparison

Key findings:

- **ILS outperforms LNS** on both instances with the same time budget
- TSPA: ILS achieves 69340 avg vs LNS+LS's 69689 (+0.50% worse)
- TSPB: ILS achieves 43674 avg vs LNS+LS's 44239 (+1.29% worse)
- ILS performs ~2.4× more local search runs than LNS iterations (3545 vs 1477 for TSPA)

LNS characteristics:

- LNS's large neighborhood (30% destruction) provides more diversification
- Each LNS iteration is more expensive due to greedy repair ($O(n^2)$ per iteration)
- LNS may be better for escaping very deep local optima

Effect of Local Search in LNS

Metric	LNS with LS	LNS without LS	Difference
TSPA Avg Obj	69689	69817	+0.18%
TSPB Avg Obj	44239	44294	+0.12%
TSPA Iterations	1477	1683	+14.0% more
TSPB Iterations	1439	1620	+12.6% more

Observations:

- **LNS with LS is slightly better** in solution quality
- LNS without LS completes ~12-14% more iterations
- The quality gain from local search outweighs the iteration count loss
- The greedy repair alone produces solutions close to local optima, but LS provides final refinement