Assignment 4 - Candidate Moves for Selective TSP

Authors

- Mateusz Idziejczak 155842
- Mateusz Stawicki 155900

Github

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

Problem Description

This is a variant of the Traveling Salesman Problem where:

- 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

Goal

Improve the time efficiency of steepest local search using **candidate moves**. From Assignment 3, the best local search configuration was:

- **Starting solution**: Random
- **Search strategy**: Steepest descent
- **Neighborhood**: Edges exchange (2-opt)
- **Results**: TSPA avg 73,859 in 16.2 ms, TSPB avg 48,334 in 16.3 ms

Algorithm Pseudocode

Delta Calculation

Delta represents the change in objective function after applying a move: `delta = newCost - oldCost`
If `delta < 0`, the move improves the solution.

1. Intra-Route: Edge Exchange (Reverse Segment)

Reverse the segment between positions `pos1` and `pos2`.

```
Given: [..., pos1, A, B, C, ..., Z, pos2, next, ...]
After: [..., pos1, pos2, Z, ..., C, B, A, next, ...]

Remove edges: pos1→A and pos2→next
Add edges:    pos1→pos2 and A→next

oldCost = dist(pos1, A) + dist(pos2, next)
```

```
newCost = dist(pos1, pos2) + dist(A, next)
delta = newCost - oldCost
```

Note: Only 2 edges change regardless of segment length (2-opt property).

2. Inter-Route: Node Exchange

Replace a selected node with an unselected node.

```
Given: [..., prev, oldNode, next, ...] with cost[oldNode]
After: [..., prev, newNode, next, ...] with cost[newNode]

oldCost = dist(prev, oldNode) + dist(oldNode, next) + cost[oldNode]
newCost = dist(prev, newNode) + dist(newNode, next) + cost[newNode]
delta = newCost - oldCost
```

Building Nearest Neighbors

For each node, precompute k nearest neighbors based on combined metric: `distance[i][j] + cost[j]`

```
buildNearestNeighbors(n, distance, costs, k):
    nearestNeighbors = array of size n

    For each node i from 0 to n-1:
        neighbors = []

        For each node j != i:
            combinedCost = distance[i][j] + costs[j]
            neighbors.append((combinedCost, j))

        Sort neighbors by combinedCost (ascending)
        nearestNeighbors[i] = first k nodes from sorted neighbors

    Return nearestNeighbors
```

Candidate Edge Definition

An edge `(u, v)` is a **candidate edge** if and only if:

- Node `v` is in `u`'s k nearest neighbors, **OR**
- Node `u` is in `v`'s k nearest neighbors

This symmetric definition ensures we don't miss potentially good moves.

Local Search - Steepest Descent with Candidate Moves

Key principle: Only evaluate moves that introduce at least one candidate edge. This dramatically reduces the neighborhood size from $O(n^2)$ to $O(nk)$.

Understanding Candidate Move Selection

For 2-opt (intra-route): Reversing segment between positions `i` and `j` introduces exactly 2 new edges:

1. Edge `(sol[i], sol[j])` — connects the positions that define the segment boundaries
2. Edge `(sol[i+1], sol[j+1])` — connects the nodes after the boundaries

We evaluate the move if **at least one** of these two edges is a candidate edge.

For node exchange (inter-route): Replacing `sol[pos]` with `newNode` introduces exactly 2 new edges:

1. Edge `(sol[prev], newNode)` — connects predecessor to new node
2. Edge `(newNode, sol[next])` — connects new node to successor

We evaluate the exchange if **at least one** of these two edges is a candidate edge.

Pseudocode

```
localSearchSteepestEdgesCandidates(initial, distance, costs, n, k):
    1. Start with initial solution
    2. Build nearest neighbors: nearestNeighbors = buildNearestNeighbors(n,
distance, costs, k)
    3. Mark nodes in solution and create position map
    4. While improvement found:
       a. bestDelta = 0
       b. For each position i in solution (intra-route):
          - For each neighbor in nearestNeighbors[sol[i]] that is selected:
            If move introduces candidate edge:
              Calculate delta for reversing segment between i and neighbor
position
              If delta < bestDelta: remember this move
          - For each neighbor in nearestNeighbors[sol[i+1]] that is selected:
            If move introduces candidate edge:
              Calculate delta for reversing segment
              If delta < bestDelta: remember this move
       c. For each position pos in solution (inter-route):
          - For each neighbor in nearestNeighbors[sol[prev]] that is NOT selected:
            Calculate delta for exchanging sol[pos] with neighbor
            If delta < bestDelta: remember this move
          - For each neighbor in nearestNeighbors[sol[next]] that is NOT selected:
            Calculate delta for exchanging sol[pos] with neighbor
            If delta < bestDelta: remember this move
       d. If bestDelta < 0:
          Apply best move and update data structures
       e. Else:
          Stop (no improvement found)
    5. Return solution
```

Key difference from baseline: Only evaluate O(nk) candidate moves per iteration instead of $O(n^2)$ all moves.

Key Results

Objective Function Values

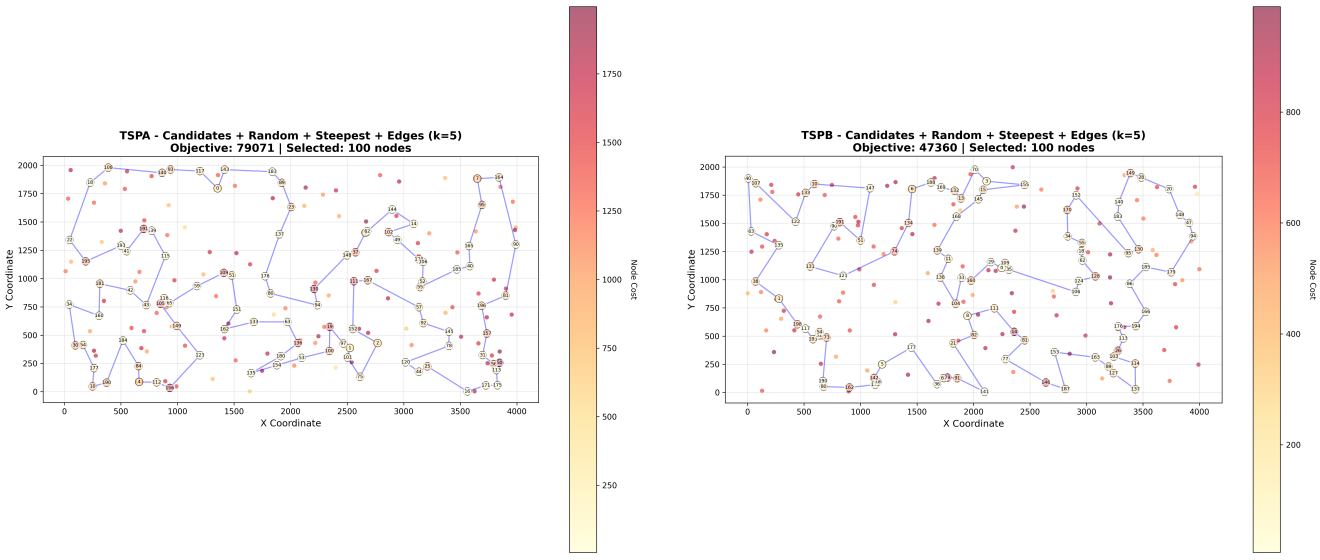
Method	TSPA	TSPB
Random	264501 (235453 - 288189)	212513 (189071 - 238254)
Nearest Neighbor (end only)	85108 (83182 - 89433)	54390 (52319 - 59030)
Nearest Neighbor (any position)	73178 (71179 - 75450)	45870 (44417 - 53438)
Greedy Cycle	72646 (71488 - 74410)	51400 (49001 - 57324)
Greedy 2-Regret	115474 (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	88236 (81008 - 99985)	62812 (54837 - 69593)
LS: Random + Greedy + Nodes	92958 (84623 - 100736)	65784 (59084 - 73710)
LS: Random + Greedy + Edges	81244 (75645 - 87637)	54153 (50134 - 60012)
LS: Greedy + Steepest + Nodes	71614 (70626 - 72950)	45414 (43826 - 50876)
LS: Greedy + Steepest + Edges	71460 (70510 - 72614)	44979 (43921 - 50629)
LS: Greedy + Greedy + Nodes	71906 (71093 - 73048)	45583 (43862 - 51165)
LS: Greedy + Greedy + Edges	71821 (70977 - 72712)	45376 (43845 - 51647)
Baseline: Random + Steepest + Edges (k=∞)	**73859 (71173 - 77472)**	**48334 (46098 - 51864)**
Candidates k=5	**84105 (79071 - 91253)**	**50012 (47360 - 53034)**
Candidates k=10	**77575 (73098 - 83326)**	**48300 (45571 - 51886)**
Candidates k=15	**75396 (71576 - 81409)**	**48537 (45790 - 53317)**
Candidates k=20	**74440 (71436 - 81577)**	**48443 (45932 - 52241)**

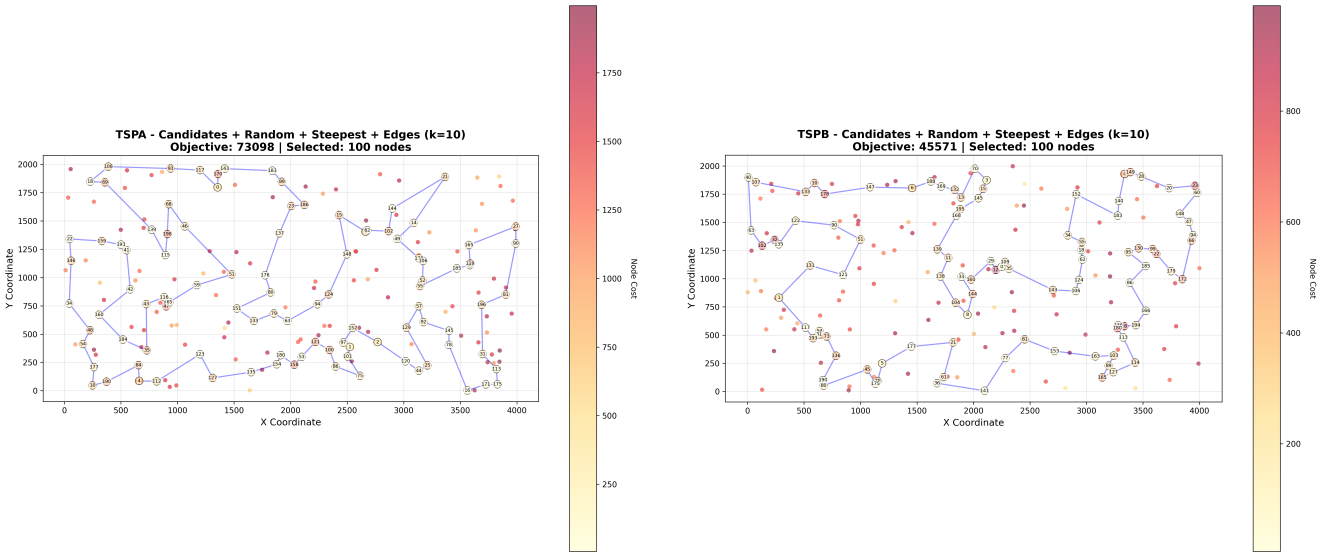
Running Times (ms)

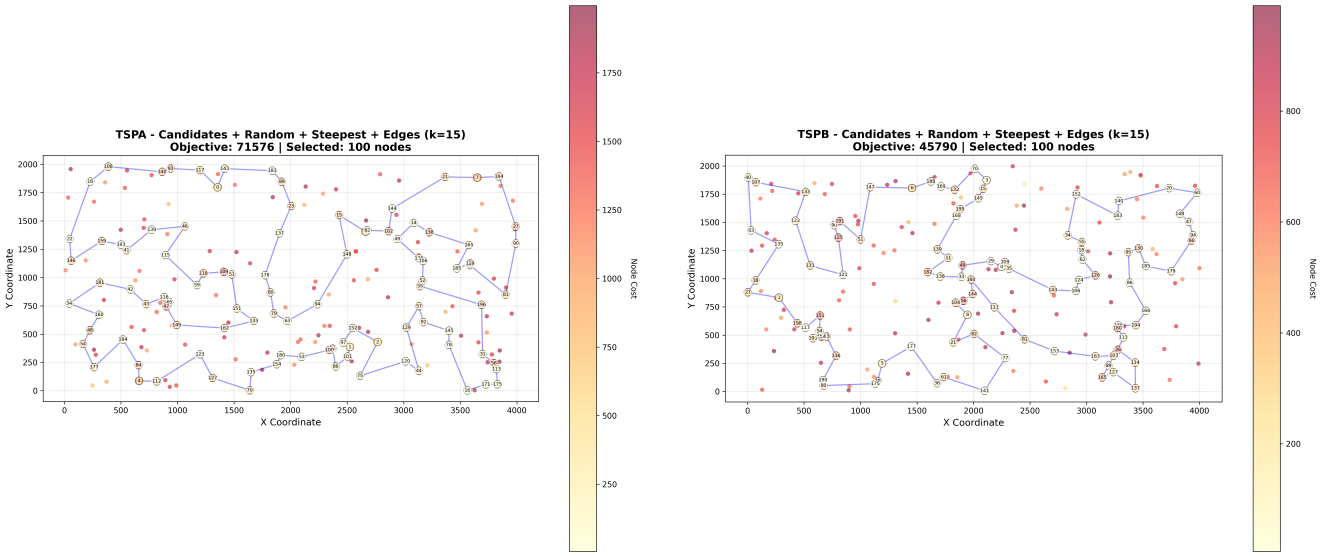
Method	TSPA	TSPB
Random	0.0040 (0.0038 - 0.0089)	0.0030 (0.0024 - 0.0129)
Nearest Neighbor (end only)	0.0389 (0.0350 - 0.0456)	0.0395 (0.0354 - 0.0449)
Nearest Neighbor (any position)	1.473 (1.461 - 1.564)	1.487 (1.469 - 1.688)
Greedy Cycle	2.620 (2.610 - 2.660)	2.617 (2.606 - 2.689)

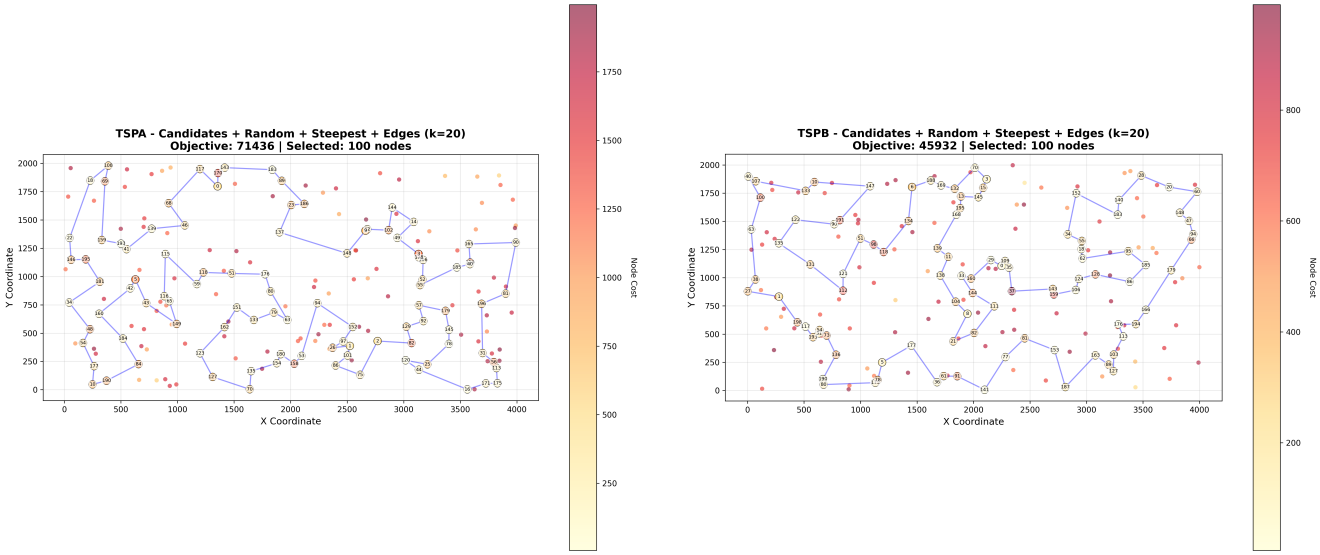
Method	TSPA	TSPB
Greedy 2-Regret	2.657 (2.647 - 2.683)	2.659 (2.647 - 2.751)
Greedy Weighted (2-Regret + BestDelta)	2.657 (2.629 - 2.752)	2.722 (2.650 - 4.617)
Nearest Neighbor Any 2-Regret	1.539 (1.455 - 1.645)	1.547 (1.474 - 2.058)
Nearest Neighbor Any Weighted	1.677 (1.482 - 2.045)	1.558 (1.441 - 1.668)
LS: Random + Steepest + Nodes	24.999 (20.194 - 34.326)	24.745 (20.042 - 30.587)
LS: Random + Steepest + Edges	16.197 (14.352 - 18.603)	16.308 (13.887 - 19.573)
LS: Random + Greedy + Nodes	3.602 (2.019 - 6.316)	3.379 (1.823 - 6.855)
LS: Random + Greedy + Edges	2.566 (1.731 - 3.992)	2.548 (1.700 - 4.449)
LS: Greedy + Steepest + Nodes	3.590 (2.965 - 4.587)	2.465 (1.949 - 5.387)
LS: Greedy + Steepest + Edges	3.524 (3.008 - 4.208)	2.436 (1.937 - 5.074)
LS: Greedy + Greedy + Nodes	3.881 (3.436 - 5.131)	2.719 (2.267 - 3.884)
LS: Greedy + Greedy + Edges	3.854 (3.405 - 4.924)	2.666 (2.193 - 3.790)
Baseline: Random + Steepest + Edges (k=∞)	**16.197 (14.352 - 18.603)**	**16.308 (13.887 - 19.573)**
Candidates k=5	**4.175 (3.613 - 4.815)**	**4.489 (3.933 - 5.157)**
Candidates k=10	**6.155 (4.975 - 15.335)**	**6.470 (5.782 - 9.737)**
Candidates k=15	**8.006 (6.910 - 10.009)**	**8.415 (7.353 - 9.458)**
Candidates k=20	**9.958 (8.944 - 12.005)**	**10.894 (9.212 - 25.982)**

Visualizations









Conclusions

Candidate moves reduce running time by restricting the neighborhood from $O(n^2)$ to $O(nk)$ moves per iteration.

Speed improvement: Candidate moves are 1.5-3.9× faster than baseline steepest descent, depending on k.

Quality trade-off: Solution quality is slightly worse (0.8-13.9% depending on k). Smaller k gives better speedup but worse quality.

Recommendation: k=10 provides a good balance (2.5× faster, ~5% quality loss for TSPA, no loss for TSPB).