Greedy Heuristics

Students

- Patrick Molina, 157419
- Chihabeddine Zitouni, 158753

Code Repository

https://github.com/patrickmolina1/Evolutionary-Computation

Problem Description

The task is to select a subset of nodes and form an optimal Hamiltonian cycle minimizing the total cost and travel distance. Each node is defined by three attributes: x-coordinate, y-coordinate, and cost. Exactly 50% of the nodes must be selected (rounded up if the total number is odd). The goal is to minimize the sum of the cycle length and the total cost of the selected nodes. Distances between nodes are computed using the Euclidean distance, rounded to the nearest integer. A distance matrix is precomputed after reading each instance and used throughout the optimization process, allowing instances to be represented solely by distance values

Methods

Four constructive heuristics are implemented and adapted to this problem:

1. **Random Solution** – Generates a feasible cycle by randomly selecting nodes.
2. **Nearest Neighbor (End Insertion)** – Adds the next node giving the best improvement when inserted only at the **end** of the path.
3. **Nearest Neighbor (Flexible Insertion)** – Adds the next node at any position (beginning, end, or inside) that best improves the objective function.
4. **Greedy Cycle** – Builds the cycle by repeatedly inserting nodes that minimize the total increase in distance and cost.

Here, "nearest" is understood as the **best improvement in the objective value**, not just geometric closeness. For each heuristic, **200 solutions** are generated from each starting node, and **200 random solutions** are also produced for comparison.

Computational experiment

CPU Specs

AMD Ryzen 7 5800H with Radeon Graphics

Cores	Threads	Clockspeed	Turbo Speed
8	16	3.2 GHz	4.4 GHz

Pseudocode

Algorithm: Random Solution
Input: instance
Output: solution

1. Get n ← number of nodes in instance
2. Calculate numToSelect ← ⌈n / 2⌉
3. Create shuffled copy of all nodes
4. Select first numToSelect nodes from shuffled list
5. Create order list with IDs of selected nodes
6. Shuffle order list randomly
7. Calculate totalDistance:
    - For each consecutive pair in order (including wrap-around):
        - Add distance from current to next node
8. Calculate totalNodeCost ← sum of costs of selected nodes
9. Calculate totalCost ← totalDistance + totalNodeCost
10. Return solution with selected nodes, order, costs

Algorithm: Nearest Neighbor End Only
Input: instance
Output: solution

1. Select random startNode from all nodes
2. Initialize selected ← [startNode]
3. Initialize order ← [startNode.id]
4. Initialize remaining ← all nodes except startNode
5. Calculate numToSelect ← ⌈n / 2⌉
6. While selected.size < numToSelect AND remaining is not empty:
   a. Get lastNode ← last node in selected
   b. Get firstNode ← first node in selected
   c. Initialize bestCandidate ← null, minIncrease ← ∞
   d. For each candidate in remaining:
      i. Calculate distToCandidate ← distance[lastNode][candidate]
      ii. Calculate distCandidateToFirst ← distance[candidate][firstNode]
      iii. Calculate distLastToFirst ← distance[lastNode][firstNode]
      iv. Calculate distanceIncrease ← distToCandidate + distCandidateToFirst − distLastToFirst
      v. Calculate objectiveIncrease ← distanceIncrease + candidate.cost
      vi. If objectiveIncrease < minIncrease:
         - Update minIncrease ← objectiveIncrease
         - Update bestCandidate ← candidate
   e. If bestCandidate ≠ null:
      i. Add bestCandidate to end of selected
      ii. Add bestCandidate.id to end of order
      iii. Remove bestCandidate from remaining
7. Calculate totalDistance (sum distances between consecutive nodes in cycle)
8. Calculate totalNodeCost ← sum of costs of selected nodes
9. Calculate totalCost ← totalDistance + totalNodeCost
10. Return solution with selected nodes, order, costs

Algorithm: Nearest Neighbor All Positions
Input: instance
Output: solution

1. Select random startNode from all nodes
2. Initialize selected ← [startNode]
3. Initialize order ← [startNode.id]
4. Initialize remaining ← all nodes except startNode
5. Calculate numToSelect ← ⌈n / 2⌉
6. While selected.size < numToSelect AND remaining is not empty:
   a. Initialize bestCandidateToAdd ← null, minSelectionMetric ← ∞
   b. For each candidate in remaining:
      i. Find minDistanceToTour ← minimum distance from candidate to any node in selected
      ii. Calculate selectionMetric ← minDistanceToTour + candidate.cost
      iii. If selectionMetric < minSelectionMetric:
         - Update minSelectionMetric ← selectionMetric
         - Update bestCandidateToAdd ← candidate
   c. If bestCandidateToAdd ≠ null:
      i. Initialize bestPosition ← -1, minIncrease ← ∞
      ii. For each position i in order:
         - Get prevNodeId ← order[i]
         - Get nextNodeId ← order[(i + 1) % order.size]
         - Calculate distPrevToNext ← distance[prevNodeId][nextNodeId]
         - Calculate distPrevToCandidate ← distance[prevNodeId][bestCandidateToAdd]
         - Calculate distCandidateToNext ← distance[bestCandidateToAdd][nextNodeId]
         - Calculate distanceIncrease ← distPrevToCandidate + distCandidateToNext − distPrevToNext
         - Calculate objectiveIncrease ← distanceIncrease + bestCandidateToAdd.cost
         - If objectiveIncrease < minIncrease:
            - Update minIncrease ← objectiveIncrease
            - Update bestPosition ← i + 1
      iii. Insert bestCandidateToAdd at bestPosition in selected
      iv. Insert bestCandidateToAdd.id at bestPosition in order
      v. Remove bestCandidateToAdd from remaining
   d. Else: break
7. Calculate totalDistance (sum distances between consecutive nodes in cycle)
8. Calculate totalNodeCost ← sum of costs of selected nodes
9. Calculate totalCost ← totalDistance + totalNodeCost
10. Return solution with selected nodes, order, costs

Algorithm: Greedy Cycle

Input: instance, startNode

Output: solution

```
1. Initialize selected ← [startNode]
2. Initialize order ← [startNode.id]
3. Initialize remaining ← all nodes except startNode
4. Calculate numToSelect ← ⌈n / 2⌉
5. While selected.size < numToSelect AND remaining is not empty:
   a. Initialize bestCandidate ← null, bestPosition ← -1, minIncrease ← ∞
   b. For each candidate in remaining:
      i.   For each position pos in order:
           • Get prevNodeId ← order[pos]
           • Get nextNodeId ← order[(pos + 1) % order.size]
           • Calculate distPrevToNext ← distance[prevNodeId][nextNodeId]
           • Calculate distPrevToCandidate ← distance[prevNodeId][candidate]
           • Calculate distCandidateToNext ← distance[candidate][nextNodeId]
           • Calculate distanceIncrease ← distPrevToCandidate +
             distCandidateToNext − distPrevToNext
           • Calculate objectiveIncrease ← distanceIncrease + candidate.cost
           • If objectiveIncrease < minIncrease:
                ○ Update minIncrease ← objectiveIncrease
                ○ Update bestCandidate ← candidate
                ○ Update bestPosition ← pos + 1
   c. If bestCandidate ≠ null:
      i.   Insert bestCandidate at bestPosition in selected
      ii.  Insert bestCandidate.id at bestPosition in order
      iii. Remove bestCandidate from remaining
6. Calculate totalDistance (sum distances between consecutive nodes in cycle)
7. Calculate totalNodeCost ← sum of costs of selected nodes
8. Calculate totalCost ← totalDistance + totalNodeCost
9. Return solution with selected nodes, order, costs
```

Results

Method	Instance A	Instance B
Random Solution	264419.52 (230885 – 300429)	213914.41 (191803 - 235581)
Nearest Neighbor End Only	103674.65 (90323 - 117672)	69828.78 (62606 - 77415)
Nearest Neighbor All Positions	72334.99 (71515 - 73823)	48994.88 (47295 – 51030)
Greedy Cycle	72598.91 (71488 - 74410)	51532.22 (49001 – 57324)

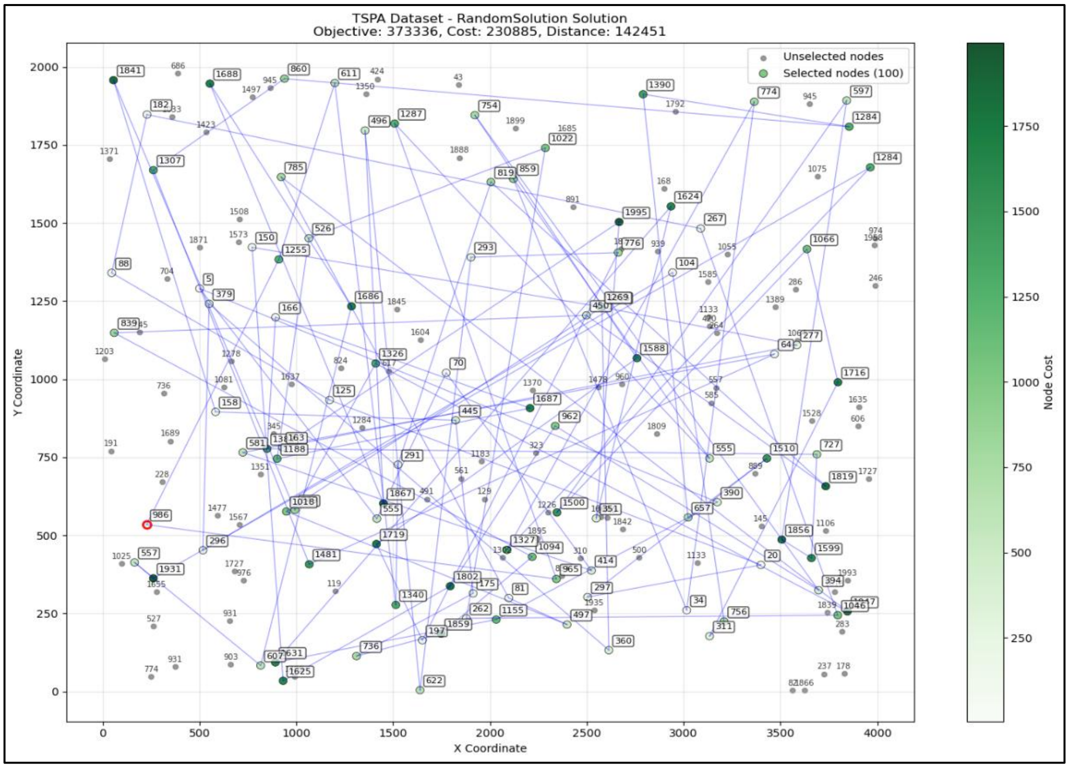
Running Times(ms)

Method	Instance A	Instance B
Random Solution	0.06 (0 - 3)	0.07 (0 - 4)
Nearest Neighbor End Only	0.25 (0 - 3)	0.30 (0 - 3)
Nearest Neighbor All Positions	1.78 (0 - 13)	2.08 (0 - 13)
Greedy Cycle	2.32 (1 - 22)	2.37 (1 - 23)

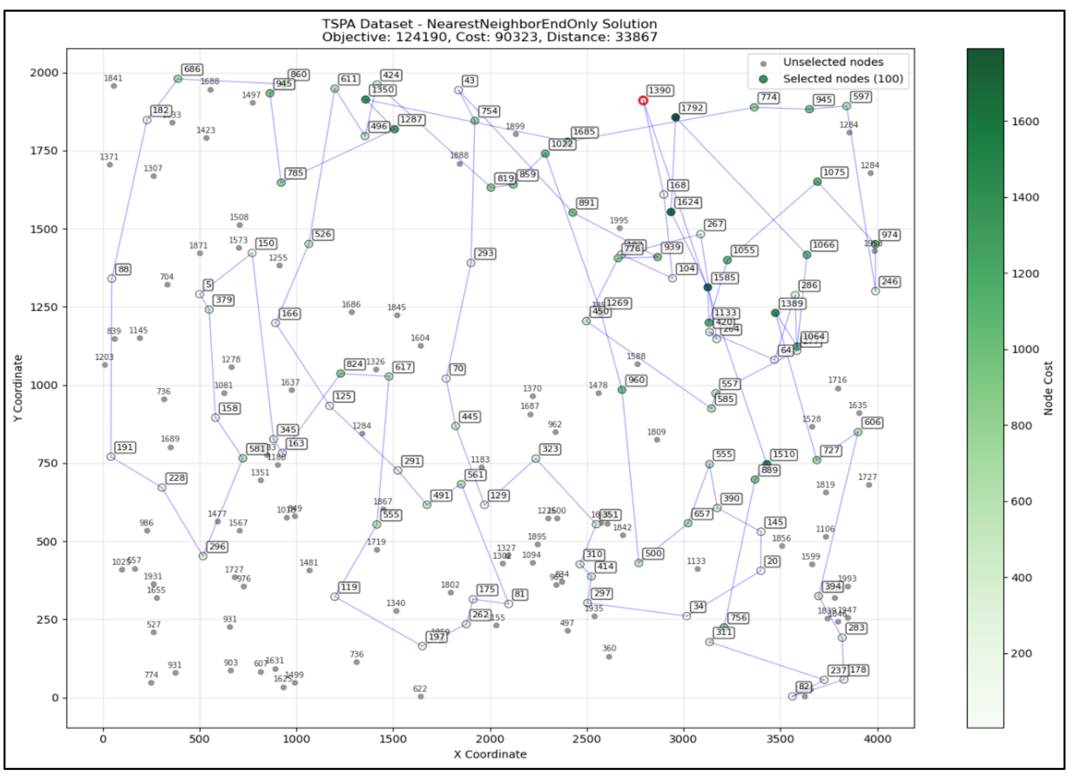
Best Solution

Instance A

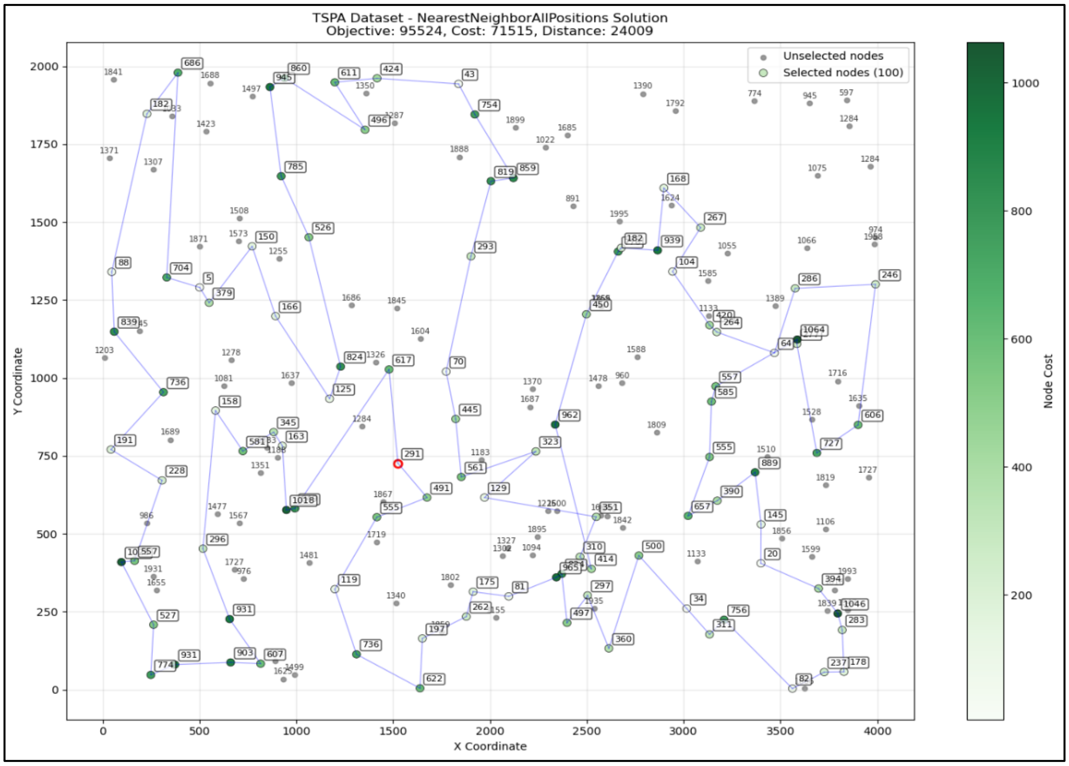
Random Solution



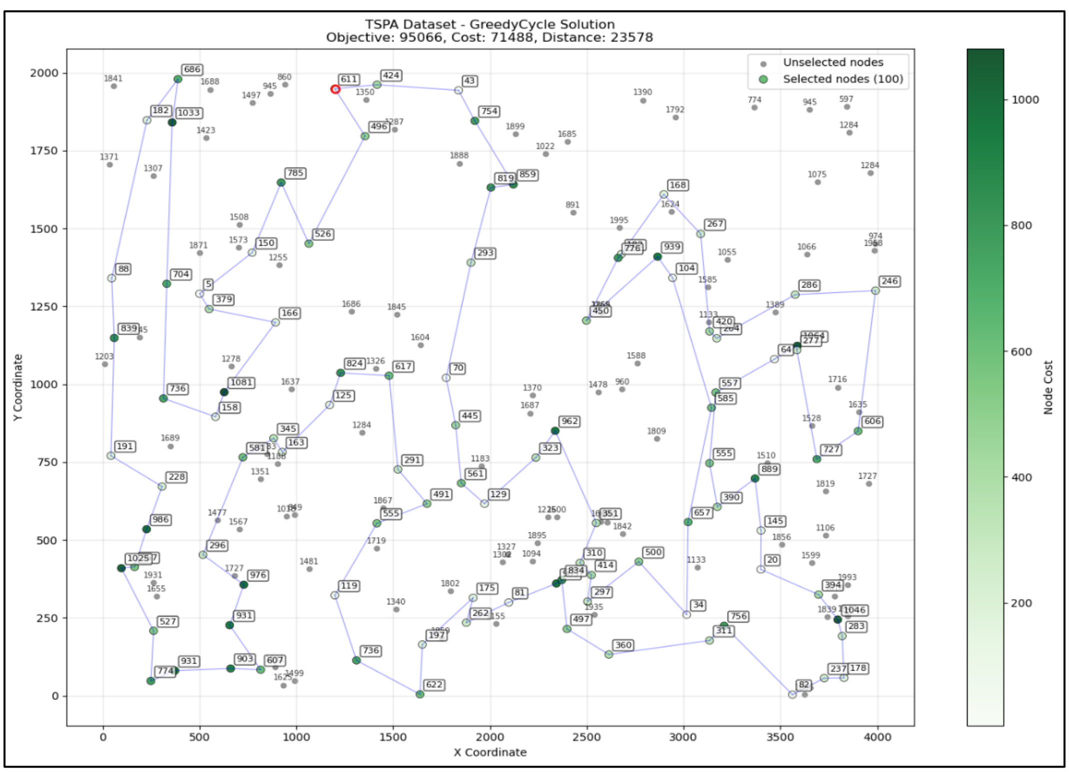
Nearest Neighbor End Only



Nearest Neighbor All Positions

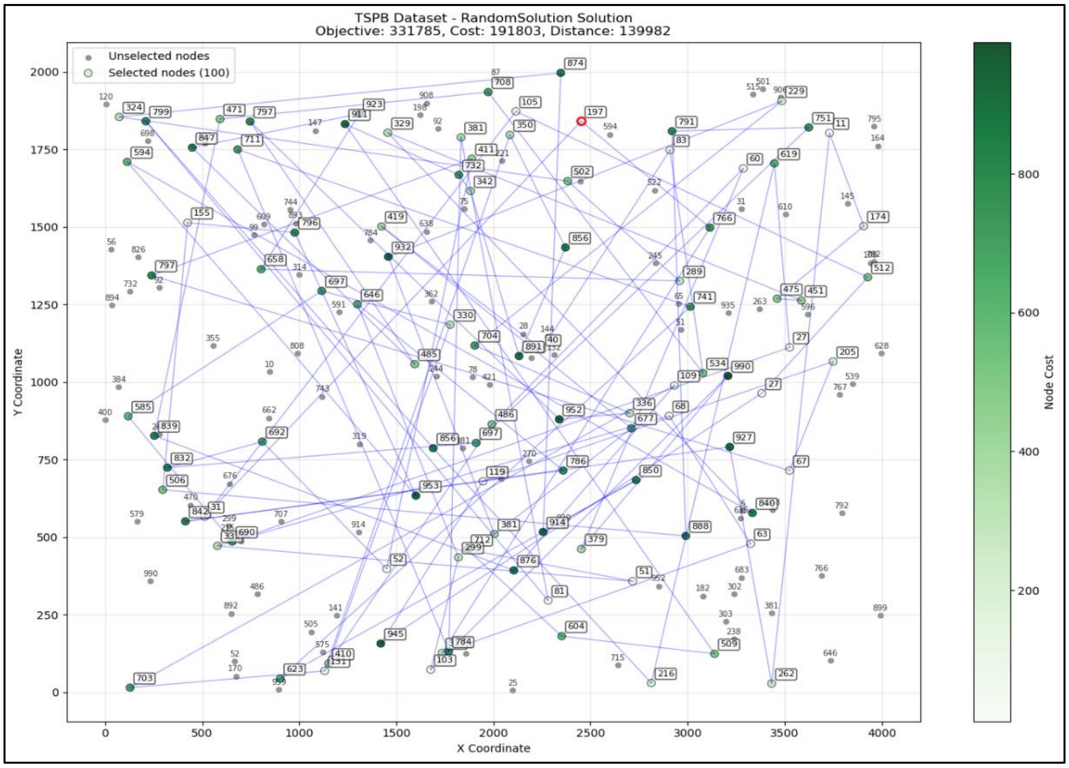


Greedy Cycle

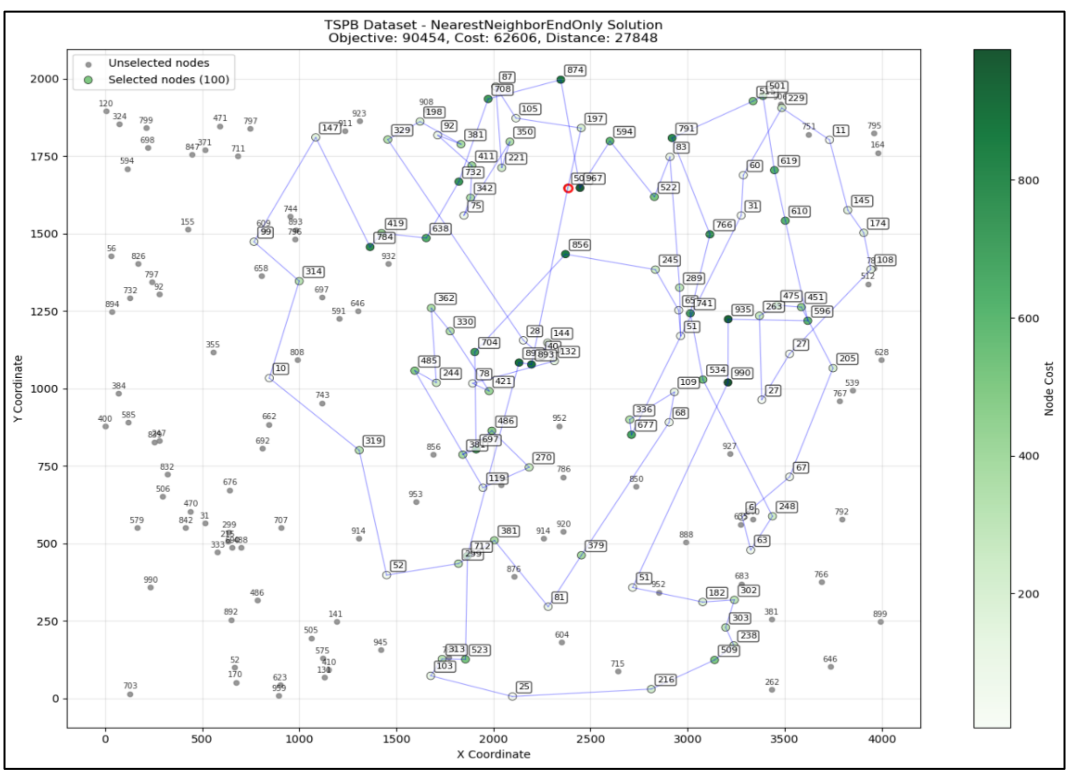


Instance B

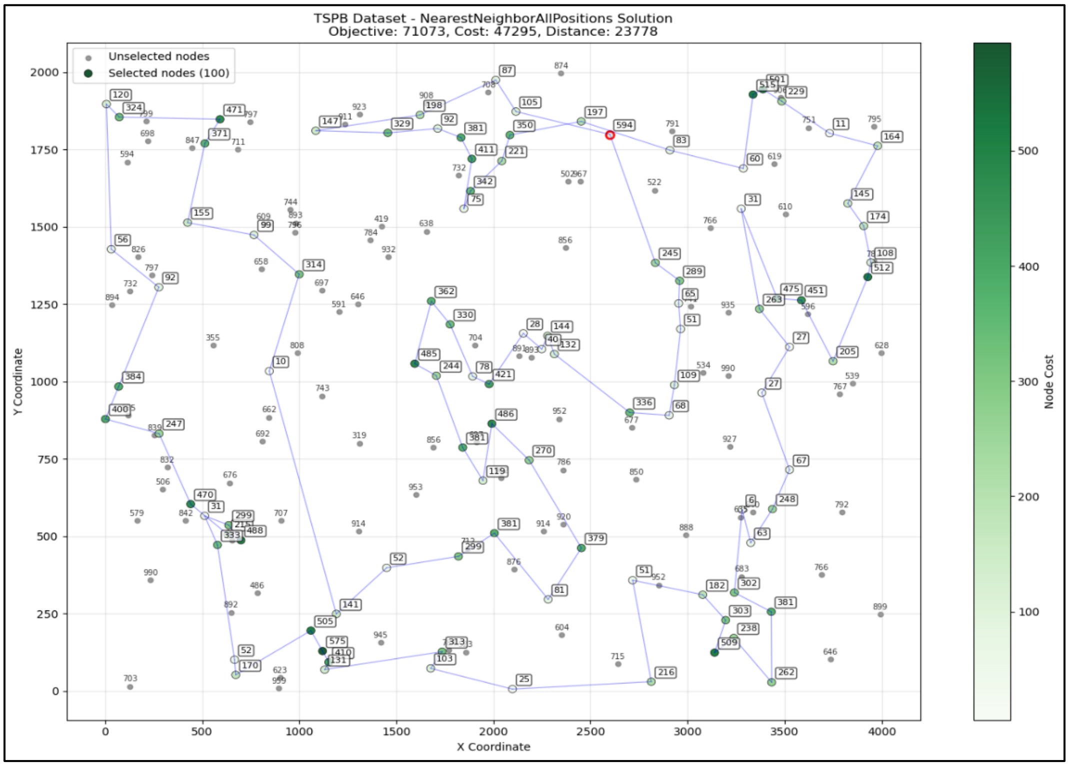
Random Solution



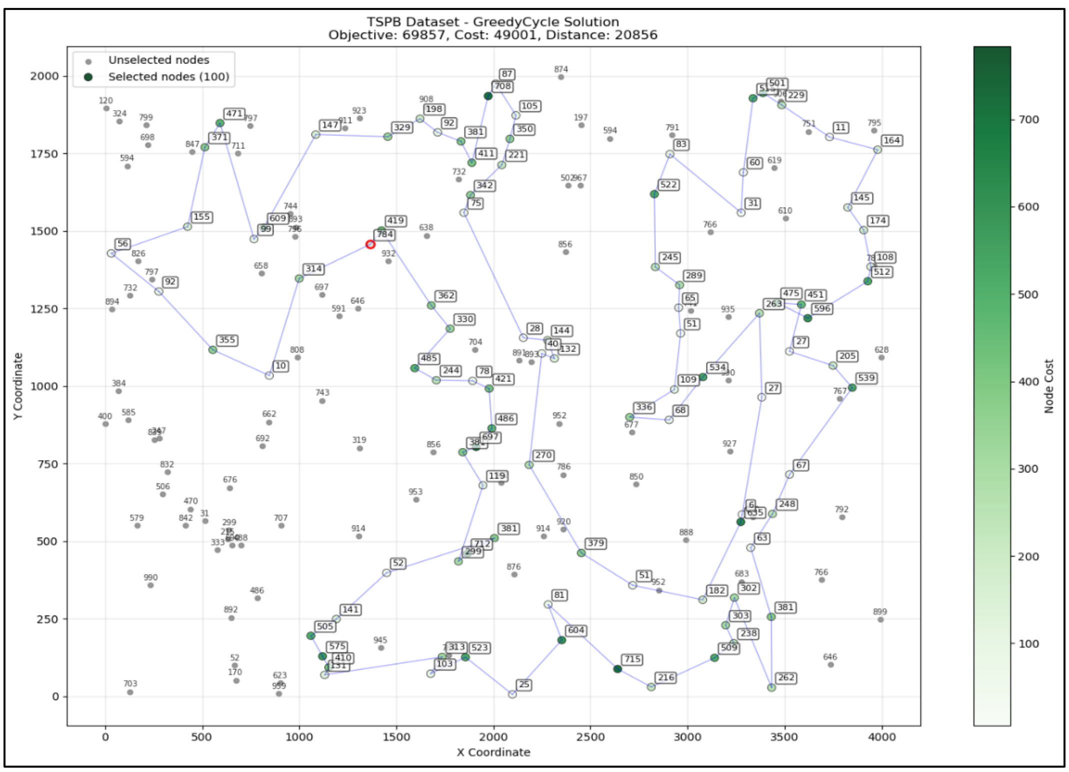
Nearest Neighbor End Only



Nearest Neighbor All Positions



Greedy Cycle



Cycles

Instance A

Random Solution

48-198-117-194-151-23-17-148-146-121-6-127-180-80-42-134-149-109-50-68-53-86-131-33-75-41-184-49-25-44-174-39-19-164-168-38-196-105-115-130-47-46-114-154-56-14-18-22-59-78-101-91-129-73-58-93-199-24-185-43-1-176-162-193-92-158-126-155-152-61-65-40-37-139-156-173-124-31-89-128-161-135-9-137-70-0-45-67-60-112-54-147-71-182-21-120-186-57-153-100

Nearest Neighbor End Only

73-144-49-62-14-106-178-185-165-40-52-55-148-9-102-15-183-89-137-176-80-63-94-152-97-1-101-120-78-145-92-57-129-2-167-37-114-186-23-143-0-117-46-115-59-151-133-79-53-180-154-135-123-162-51-118-65-116-139-193-41-42-43-184-160-34-22-18-108-93-140-68-153-170-64-21-7-164-90-27-95-138-3-32-155-132-39-119-8-196-81-31-113-175-16-171-44-25-179-91

Nearest Neighbor All Positions

151-51-149-131-65-116-43-42-184-84-112-4-190-10-177-30-54-160-34-181-146-22-18-108-159-193-41-139-115-59-118-46-68-140-93-0-117-143-183-89-186-23-137-176-80-79-94-63-152-97-1-124-148-9-62-102-144-14-49-178-106-185-165-90-81-196-40-119-52-55-57-129-92-179-145-78-31-56-113-175-171-16-25-44-120-2-75-101-86-26-100-53-180-154-135-70-127-123-162-133

Greedy Cycle

117-0-46-68-139-193-41-115-5-42-181-159-69-108-18-22-146-34-160-48-54-30-177-10-190-4-112-84-35-184-43-116-65-59-118-51-151-133-162-123-127-70-135-180-154-53-100-26-86-75-44-25-16-171-175-113-56-31-78-145-179-92-57-52-185-119-40-196-81-90-165-106-178-14-144-62-9-148-102-49-55-129-120-2-101-1-97-152-124-94-63-79-80-176-137-23-186-89-183-143

Instance B

Random Solution

155-174-61-113-106-193-153-87-97-165-192-126-195-39-69-15-175-108-124-58-197-10-161-128-152-0-134-159-82-2-28-49-143-37-115-182-44-150-125-55-189-107-84-50-119-162-83-178-79-132-77-17-157-187-199-185-164-16-98-11-78-144-140-81-110-36-21-74-41-30-64-137-20-47-130-99-6-88-100-171-120-122-156-129-53-59-56-24-19-177-13-32-166-179-8-7-86-66-3-117

Nearest Neighbor End Only

189-155-3-70-145-15-168-195-13-169-132-188-6-29-0-109-35-33-160-11-139-138-182-8-111-144-104-56-49-69-34-18-62-55-152-170-184-167-84-161-126-43-134-85-147-90-51-121-25-177-21-82-77-81-106-124-143-159-183-140-28-20-148-47-94-185-86-95-130-99-179-166-176-113-194-128-83-174-53-4-149-199-9-22-181-110-153-163-103-89-127-165-187-141-36-61-91-87-39-12

Nearest Neighbor All Positions

184-34-55-18-62-124-106-143-35-109-0-29-160-33-11-139-182-138-104-8-144-111-81-77-82-21-177-5-121-51-90-122-133-10-107-40-63-135-38-27-1-198-31-73-54-117-193-190-80-45-142-78-175-61-36-141-187-153-163-89-165-127-137-114-103-176-113-194-166-86-185-95-183-130-99-179-66-94-47-148-60-20-28-149-4-140-152-155-15-145-195-168-13-132-169-6-147-188-70-3

Greedy Cycle

85-51-121-131-135-63-122-133-10-90-191-147-6-188-169-132-13-161-70-3-15-145-195-168-29-109-35-0-111-81-153-163-180-176-86-95-128-106-143-124-62-18-55-34-170-152-183-140-4-149-28-20-60-148-47-94-66-22-130-99-185-179-172-166-194-113-114-137-103-89-127-165-187-146-77-97-141-91-36-61-175-78-142-45-5-177-82-87-21-8-104-56-144-160-33-138-182-11-139-134

Conclusion

- After the experiment it reveals a clear performance of **Greedy Cycle** and **Nearest Neighbor with All Positions** having similar results but **NN-all positions** has achieved the best solution quality for both TSPA and TSPB instances.
- **Nearest Neighbor with All Positions** insertion demonstrated significantly better performance than its **End Only variant**, highlighting the importance of considering multiple insertion points during solution construction rather than restricting nodes to path endpoints.
- The **Random Solution** method, while computationally trivial, produced substantially inferior results, serving primarily as a baseline to demonstrate the value of intelligent heuristic design.