Greedy Regret 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

Extended version of the nearest neighbor heuristic and the greedy cycle heuristic:

- **Greedy 2-regret heuristics.**
- **Greedy heuristics with a weighted sum criterion** – 2-regret + best change of the objective function. By default, use equal weights but you can also experiment with other values

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: Greedy 2-Regret Nearest Neighbor
Input: instance, startNode
Output: solution

1. Initialize selected ← [startNode], order ← [startNode.id]
2. Initialize remaining ← all nodes except startNode
3. Calculate numToSelect ← ⌈n/2⌉
4. While selected.size < numToSelect AND remaining is not empty:
   a. Step 1: Find best candidate based on regret
   b. Set bestCandidateToAdd ← null, maxRegret ← -∞
   c. For each candidate in remaining:
      i. Set minDistanceToTour ← ∞, secondMinDistanceToTour ← ∞
      ii. For each inTour node in selected:
         - Get distance ← dist[inTour.id][candidate.id]
         - If distance < minDistanceToTour:
            - secondMinDistanceToTour ← minDistanceToTour
            - minDistanceToTour ← distance
         - Else if distance < secondMinDistanceToTour:
            - secondMinDistanceToTour ← distance
      iii. Calculate bestMetric ← minDistanceToTour + candidate.cost
      iv. Calculate secondBestMetric ← secondMinDistanceToTour + candidate.cost
      v. Calculate regret ← secondBestMetric − bestMetric
      vi. If regret > maxRegret:
         - maxRegret ← regret
         - bestCandidateToAdd ← candidate
   d. Step 2: Find best position to insert selected candidate
   e. If bestCandidateToAdd is not null:
      i. Set bestPosition ← -1, minIncrease ← ∞
      ii. For each position pos in cycle (0 to order.size):
         - Get prevNodeId ← order[pos]
         - Get nextNodeId ← order[(pos + 1) % order.size]
         - Calculate objectiveIncrease ← (dist[prev][candidate] + dist[candidate][next] - dist[prev][next]) + candidate.cost
         - If objectiveIncrease < minIncrease:
            - minIncrease ← objectiveIncrease
            - bestPosition ← pos + 1
      iii. Insert bestCandidateToAdd into selected at bestPosition
      iv. Insert bestCandidateToAdd.id into order at bestPosition
      v. Remove bestCandidateToAdd from remaining
   f. Else break
5. Calculate totalDistance by summing distances along cycle
6. Calculate totalNodeCost by summing node costs
7. Calculate totalCost ← totalDistance + totalNodeCost
8. Return solution with selected, order, totalCost, totalDistance

Algorithm: Greedy Weighted Regret Nearest Neighbor
Input: instance, startNode, weightRegret, weightObjective
Output: solution

1. Initialize selected ← [startNode], order ← [startNode.id]
2. Initialize remaining ← all nodes except startNode
3. Calculate numToSelect ← ⌈n/2⌉
4. While selected.size < numToSelect AND remaining is not empty:
   a. Set bestCandidateToAdd ← null, maxWeightedScore ← -∞
   b. For each candidate in remaining:
      i. Set minDistanceToTour ← ∞, secondMinDistanceToTour ← ∞
      ii. For each inTour node in selected:
         - Get distance ← dist[inTour.id][candidate.id]
         - If distance < minDistanceToTour:
            - secondMinDistanceToTour ← minDistanceToTour
            - minDistanceToTour ← distance
         - Else if distance < secondMinDistanceToTour:
            - secondMinDistanceToTour ← distance
      iii. Calculate bestMetric ← minDistanceToTour + candidate.cost
      iv. Calculate secondBestMetric ← secondMinDistanceToTour + candidate.cost
      v. Calculate regret ← secondBestMetric − bestMetric
      vi. Calculate weightedScore ← weightRegret × regret - weightObjective × bestMetric
      vii. If weightedScore > maxWeightedScore:
         - maxWeightedScore ← weightedScore
         - bestCandidateToAdd ← candidate
   c. If bestCandidateToAdd is not null:
      i. Set bestPosition ← -1, minIncrease ← ∞
      ii. For each position pos in cycle (0 to order.size):
         - Get prevNodeId ← order[pos]
         - Get nextNodeId ← order[(pos + 1) % order.size]
         - Calculate objectiveIncrease ← (dist[prev][candidate] + dist[candidate][next] - dist[prev][next]) + candidate.cost
         - If objectiveIncrease < minIncrease:
            - minIncrease ← objectiveIncrease
            - bestPosition ← pos + 1
      iii. Insert bestCandidateToAdd into selected at bestPosition
      iv. Insert bestCandidateToAdd.id into order at bestPosition
      v. Remove bestCandidateToAdd from remaining
   d. Else break
5. Calculate totalDistance by summing distances along cycle
6. Calculate totalNodeCost by summing node costs
7. Calculate totalCost ← totalDistance + totalNodeCost
8. Return solution with selected, order, totalCost, totalDistance

Algorithm: Greedy 2-Regret Greedy Cycle
Input: instance, startNode
Output: solution

1. Initialize selected ← [startNode], order ← [startNode.id]
2. Initialize remaining ← all nodes except startNode
3. Calculate numToSelect ← ⌈n/2⌉
4. While selected.size < numToSelect AND remaining is not empty:
   a. Set bestCandidate ← null, bestPosition ← -1, maxRegret ← -∞
   b. For each candidate in remaining:
      i. Set bestIncrease ← ∞, secondBestIncrease ← ∞, bestPos ← -1
      ii. For each position pos in cycle (0 to order.size):
          - Get prevNodeId ← order[pos]
          - Get nextNodeId ← order[(pos + 1) % order.size]
          - Calculate objectiveIncrease ← (dist[prev][candidate] + dist[candidate][next] - dist[prev][next]) + candidate.cost
          - If objectiveIncrease < bestIncrease:
            - secondBestIncrease ← bestIncrease
            - bestIncrease ← objectiveIncrease
            - bestPos ← pos + 1
          - Else if objectiveIncrease < secondBestIncrease:
            - secondBestIncrease ← objectiveIncrease
      iii. Calculate regret ← (secondBestIncrease == ∞) ? bestIncrease : (secondBestIncrease - bestIncrease)
      iv. If regret > maxRegret:
          - maxRegret ← regret
          - bestCandidate ← candidate
          - bestPosition ← bestPos
   c. If bestCandidate is not null:
      - Insert bestCandidate into selected at bestPosition
      - Insert bestCandidate.id into order at bestPosition
      - Remove bestCandidate from remaining
   d. Else break
5. Calculate totalDistance by summing distances along cycle
6. Calculate totalNodeCost by summing node costs
7. Calculate totalCost ← totalDistance + totalNodeCost
8. Return solution with selected, order, totalCost, totalDistance

Algorithm: Greedy Weighted Regret Greedy Cycle
Input: instance, startNode, weightRegret, weightObjective
Output: solution

1. Initialize selected ← [startNode], order ← [startNode.id]
2. Initialize remaining ← all nodes except startNode
3. Calculate numToSelect ← ⌊n/2⌋
4. While selected.size < numToSelect AND remaining is not empty:
   a. Set bestCandidate ← null, bestPosition ← -1, maxWeightedScore ← -∞
   b. For each candidate in remaining:
      i. Set bestIncrease ← ∞, secondBestIncrease ← ∞, bestPos ← -1
      ii. For each position pos in cycle (0 to order.size):
         - Get prevNodeId ← order[pos]
         - Get nextNodeId ← order[(pos + 1) % order.size]
         - Calculate objectiveIncrease ← (dist[prev][candidate] + dist[candidate][next] - dist[prev][next]) + candidate.cost
         - If objectiveIncrease < bestIncrease:
            - secondBestIncrease ← bestIncrease
            - bestIncrease ← objectiveIncrease
            - bestPos ← pos + 1
         - Else if objectiveIncrease < secondBestIncrease:
            - secondBestIncrease ← objectiveIncrease
      iii. Calculate regret ← (secondBestIncrease == ∞) ? bestIncrease : (secondBestIncrease - bestIncrease)
      iv. Calculate weightedScore ← weightRegret × regret - weightObjective × bestIncrease
      v. If weightedScore > maxWeightedScore:
         - maxWeightedScore ← weightedScore
         - bestCandidate ← candidate
         - bestPosition ← bestPos
   c. If bestCandidate is not null:
      - Insert bestCandidate into selected at bestPosition
      - Insert bestCandidate.id into order at bestPosition
      - Remove bestCandidate from remaining
   d. Else break
5. Calculate totalDistance by summing distances along cycle
6. Calculate totalNodeCost by summing node costs
7. Calculate totalCost ← totalDistance + totalNodeCost
8. Return solution with selected, order, totalCost, totalDistance

Results

Method	Instance A	Instance B
greedy2RegretNN	121200.87 (111728 - 130895)	74916.32 (69992 – 79154)
greedyWeightedRegretNN	72555.99 (71373 – 73970)	48853.80 (46862 – 51266)
greedy2RegretGreedyCycle	116942.80 (110039 - 125138)	72817.32 (67658 – 77816)
greedyWeightedRegretGreedyCycle	71888.03 (70700 – 73381)	50278.91 (47609 – 54296)

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)

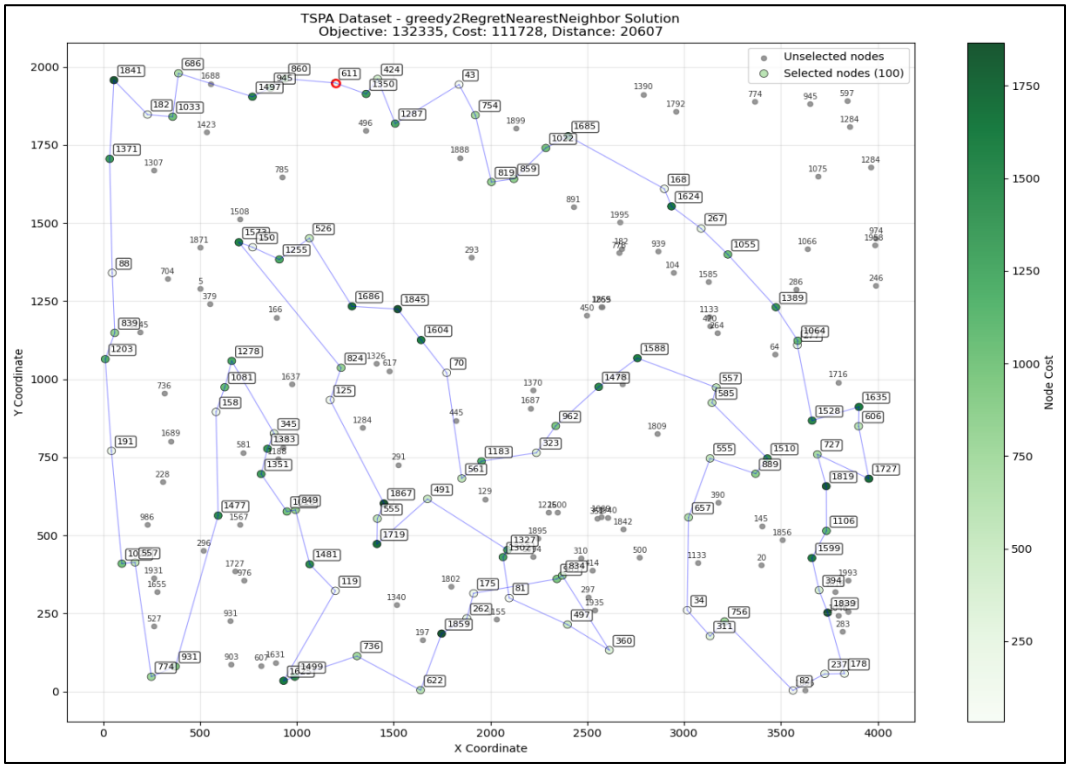
Method	Instance A	Instance B
greedy2RegretNN	4.01 (1 - 43)	5.01 (1 - 69)
greedyWeightedRegretNN	3.88 (1 - 34)	5.02 (1 - 62)
greedy2RegretGreedyCycle	5.28 (3 - 59)	5.95 (3 - 46)
greedyWeightedRegretGreedyCycle	5.40 (3 - 105)	5.89 (2 - 105)

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)

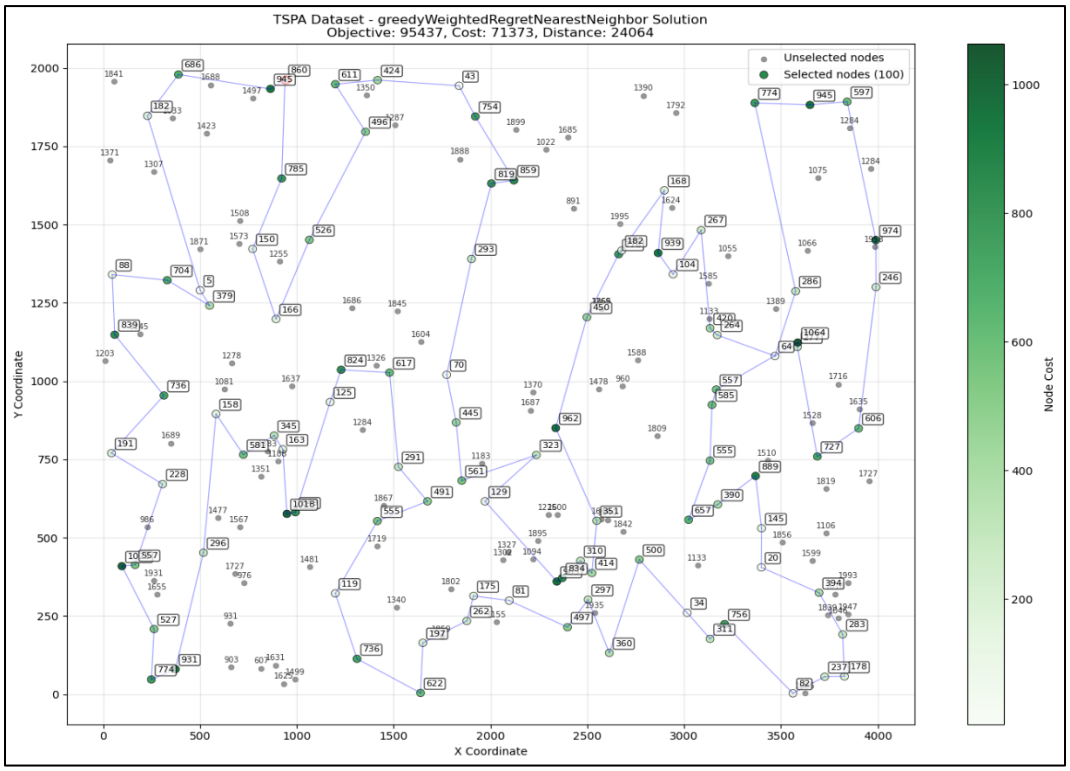
Best Solution

Instance A

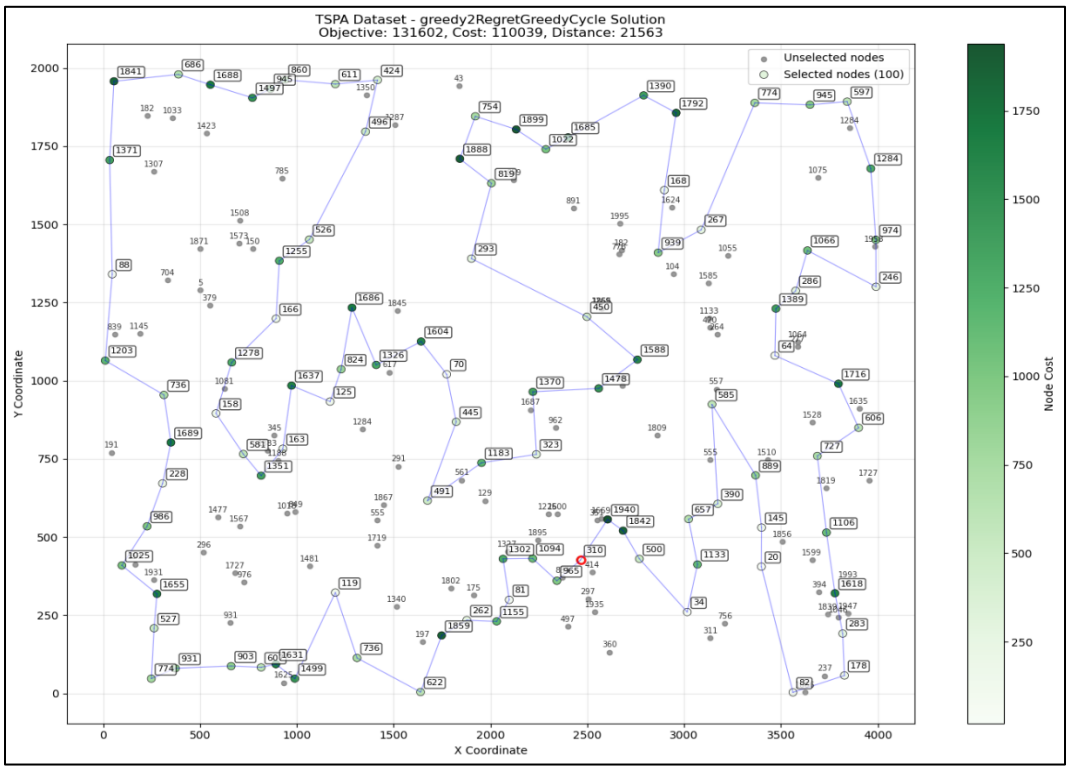
Greedy 2 Regret NN



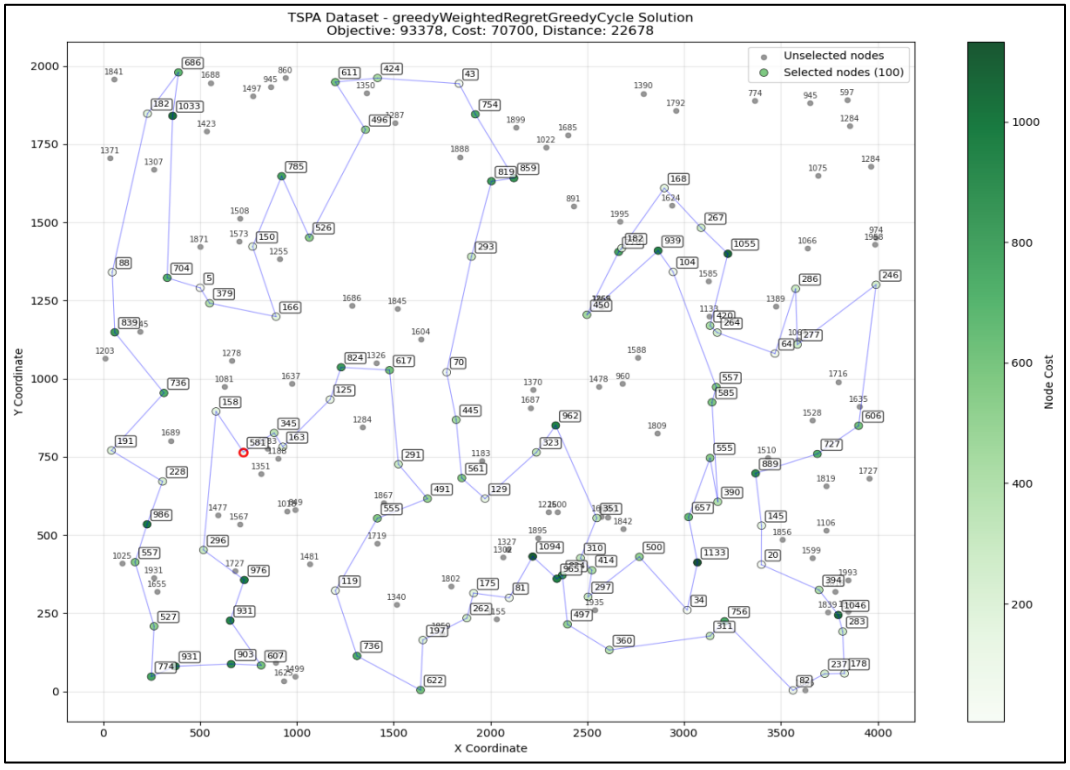
Greedy Weighted Regret NN



Greedy 2 Regret GeedyCycle

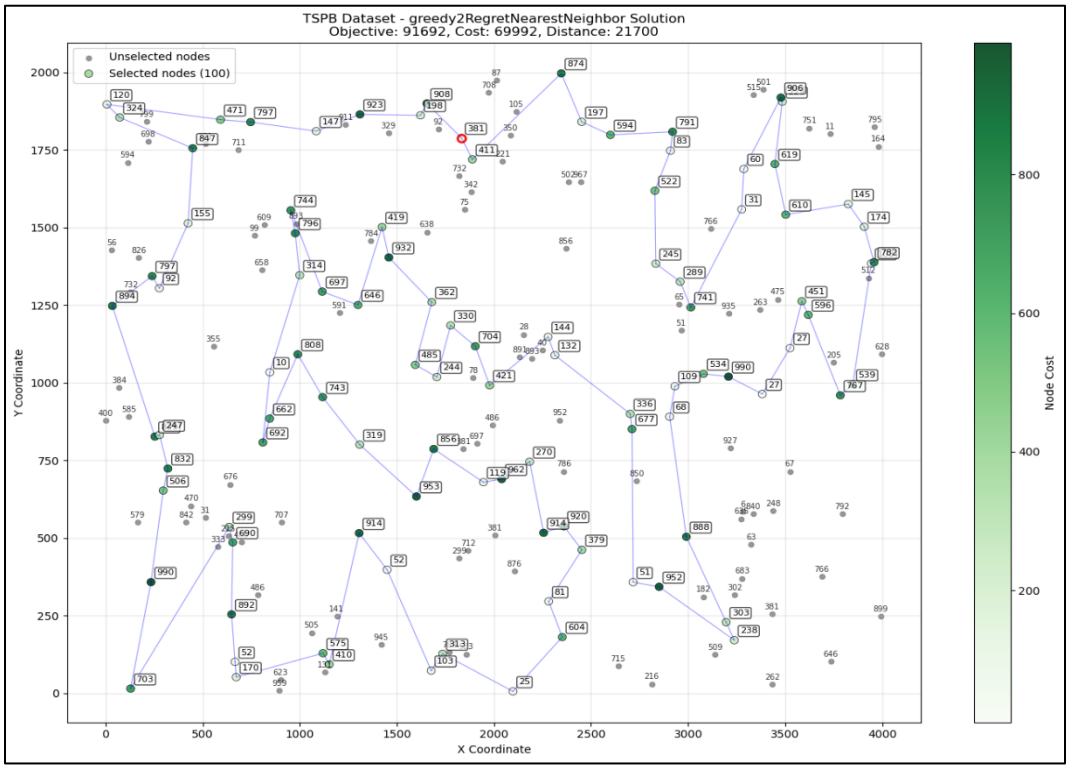


Greedy Weighted Regret Greedy Cycle

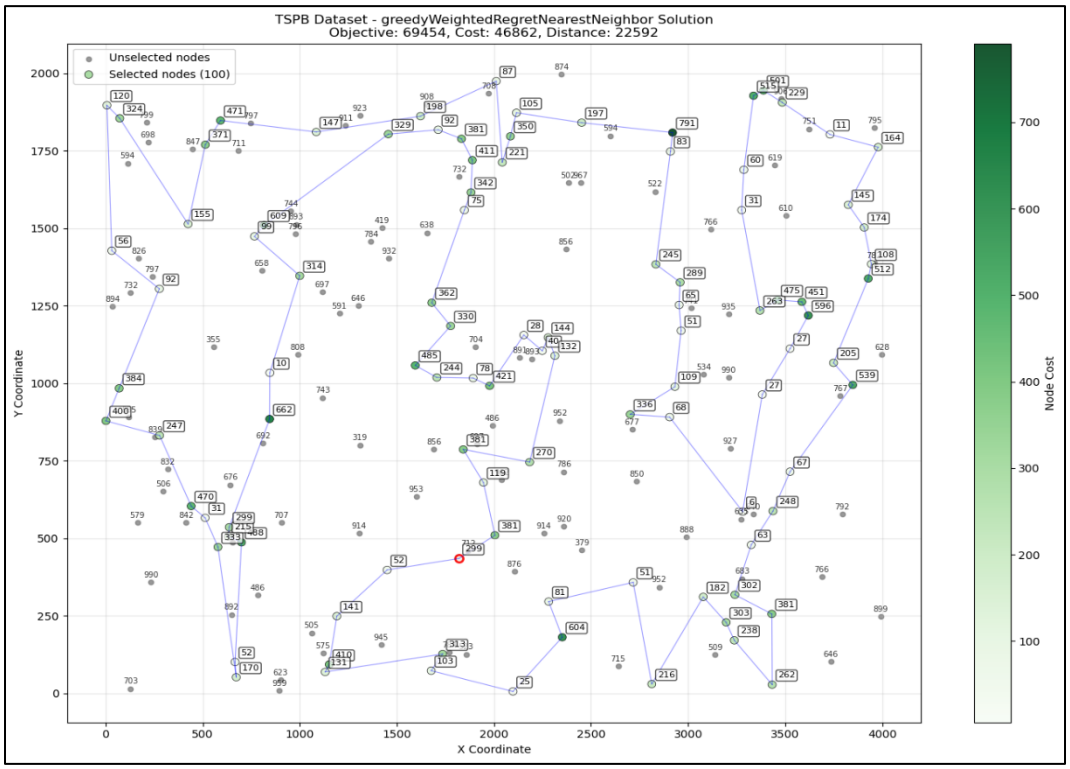


Instance B

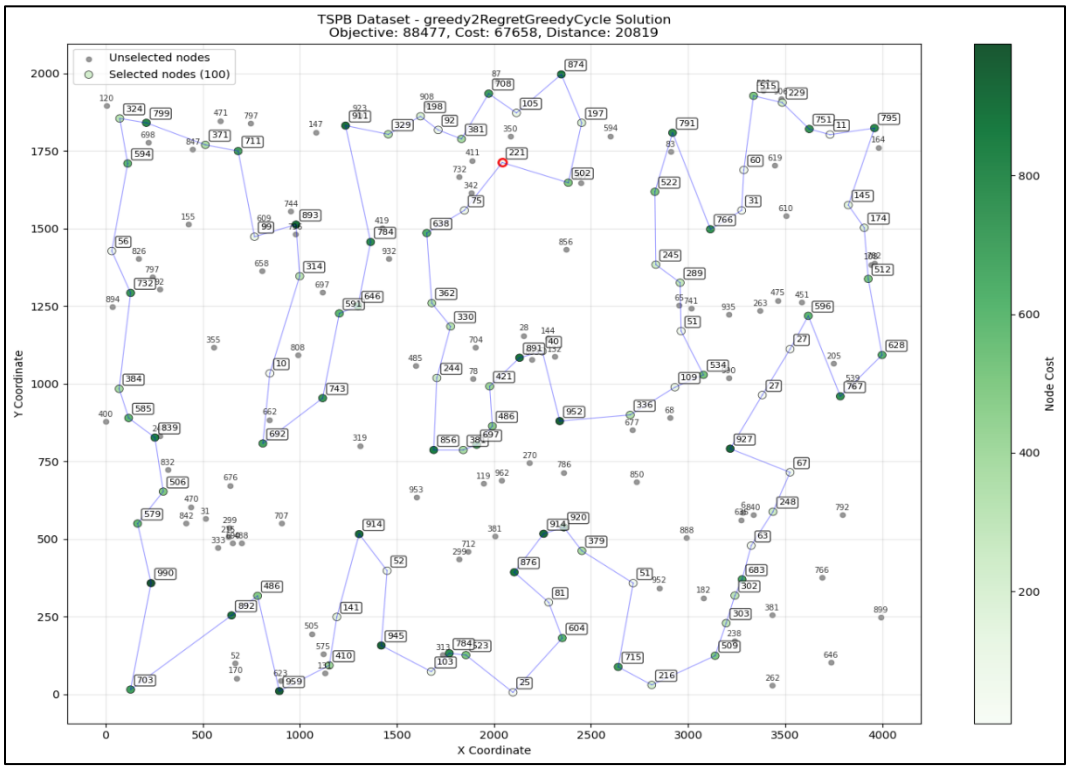
Greedy 2 Regret NN



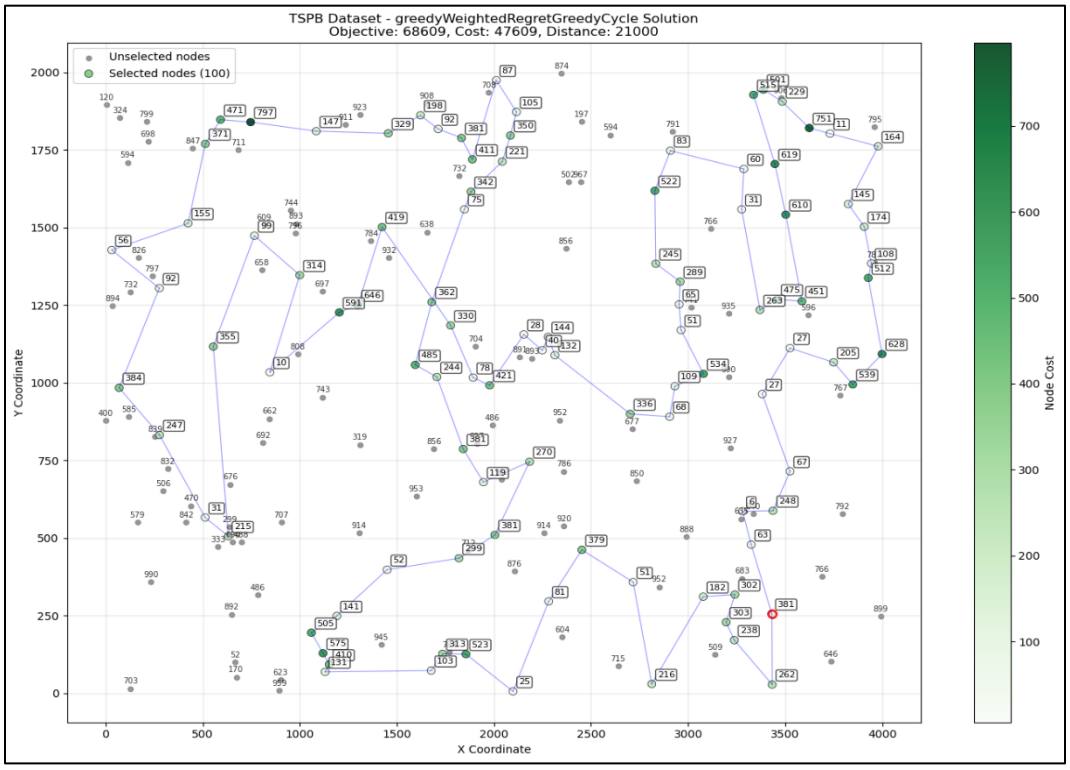
Greedy Weighted Regret NN



Greedy 2 Regret Greedy Cycle



Greedy Weighted Regret Greedy Cycle



Cycles

Instance A

Greedy 2 Regret NN

117-93-140-36-108-69-18-134-20-22-146-103-34-30-54-10-190-28-42-5-96-116-105-77-131-149-24-123-156-29-127-70-6-154-180-100-26-75-86-53-136-182-133-161-162-45-59-118-191-139-198-46-60-141-66-176-79-122-94-124-111-128-52-55-91-179-57-129-120-44-25-16-171-175-13-31-38-157-17-196-98-81-187-169-40-119-8-138-14-155-144-64-114-186-23-89-183-153-143-170

Greedy Weighted Regret NN

93-68-139-115-46-0-117-143-183-89-186-23-137-176-80-79-94-63-100-26-97-1-152-124-148-9-62-144-102-49-14-178-106-185-165-21-7-164-27-90-81-196-40-119-52-55-57-129-92-179-145-78-31-113-175-171-16-25-44-120-2-75-101-86-53-180-154-135-70-127-123-162-133-151-51-118-59-149-131-65-116-43-42-184-190-10-177-30-54-160-34-181-146-22-159-41-193-18-108-140

Greedy 2 Regret Greedy Cycle

97-125-87-2-120-82-129-92-55-179-145-78-16-175-113-85-157-196-81-174-185-8-165-39-90-27-71-164-7-21-14-102-144-132-73-64-114-83-89-76-23-137-148-128-111-12-94-122-133-80-176-66-109-60-118-59-197-65-77-43-42-96-115-198-46-0-143-117-93-140-36-67-108-134-20-22-103-181-192-160-48-30-104-177-10-190-4-112-126-29-123-127-70-6-154-158-53-136-121-100

Greedy Weighted Regret Greedy Cycle

43-42-184-35-84-112-4-190-10-177-54-48-160-34-181-146-22-18-108-69-159-193-41-115-139-68-46-0-117-143-183-89-186-23-137-176-80-79-63-94-124-152-97-1-101-2-120-82-129-57-92-55-52-49-102-148-9-62-144-14-138-178-106-185-165-40-90-81-196-179-145-78-31-56-113-175-171-16-25-44-75-86-26-100-121-53-180-154-135-70-127-123-162-133-151-51-118-59-65-116

Instance B

Greedy 2 Regret NN

132-65-188-150-147-115-10-40-107-44-122-135-32-92-197-1-24-156-196-108-54-164-105-190-80-142-78-123-177-36-61-141-97-77-81-14-50-111-68-8-157-171-25-158-116-112-19-121-51-120-71-98-74-134-2-139-182-138-11-49-160-109-35-143-159-153-186-127-89-129-106-124-128-110-86-185-99-22-52-172-94-154-47-148-9-199-28-101-140-183-83-55-34-170-152-53-184-155-84-13

Greedy Weighted Regret NN

21-177-5-78-175-61-36-141-97-77-153-187-163-89-127-137-114-103-113-194-166-172-179-66-94-47-148-60-20-28-149-4-140-183-95-130-99-22-185-86-176-106-143-124-62-18-55-34-152-53-155-3-15-145-70-188-147-10-133-122-107-40-63-135-38-27-1-198-117-193-80-190-73-31-54-112-121-51-90-191-6-169-132-13-195-168-139-11-182-138-33-160-29-0-109-35-111-104-8-82

Greedy 2 Regret Greedy Cycle

145-168-43-139-11-138-157-104-56-144-160-39-0-37-143-124-128-62-55-34-170-53-174-183-140-4-28-59-20-23-148-47-66-57-52-22-185-86-64-166-194-113-26-103-89-165-187-146-153-81-14-50-58-77-97-141-91-79-36-7-177-123-5-78-46-136-105-108-196-42-156-197-16-38-102-63-100-107-17-133-178-90-67-51-121-19-158-118-74-85-192-6-188-169-132-161-3-84-155-189

Greedy Weighted Regret Greedy Cycle

114-113-176-194-166-86-185-179-172-57-66-94-47-148-60-20-59-28-149-4-199-9-99-130-95-183-140-152-170-34-55-18-62-128-124-106-143-35-109-0-29-160-33-11-134-74-118-121-51-90-131-31-117-1-38-135-63-122-133-10-115-147-6-188-169-132-13-70-3-15-145-195-168-139-182-138-104-8-111-82-21-177-5-45-142-78-175-36-61-91-141-77-81-153-187-163-103-89-127-137

Conclusion

- The experimental evaluation of regret-based construction heuristics demonstrates the effectiveness of incorporating opportunity cost considerations into greedy decision-making revealing that regret-based methods consistently produce competitive solutions compared to their standard greedy counterparts, in this case having **weighted + 2 regret methods** solutions has good results an and even the best one for instance A, but for instance B **NN-all** possible is still the best.
- The weighted regret approaches demonstrate that balancing immediate gains with opportunity costs through equal weights (or experimentally tuned weights) can effectively navigate the trade-off between exploiting current best options and preserving future construction flexibility.
- Visualization of best solutions reveals that regret-based methods tend to select nodes more strategically, avoiding premature commitment to locally attractive but globally suboptimal choices that would constrain subsequent decisions.