Authors: Weronika Zawadzka 151943 Eliza Czaplicka 151963

Date: October 20, 2024

Best solutions have been checked with the solution checker

Summary Report for w2_greedy_regret_heuristics

Based on methods from [our github project]

Problem description

We are given three columns of integers with a row for each node. The first two columns contain x and y coordinates of the node positions in a plane. The third column contains node costs. The goal is to select exactly 50% of the nodes (if the number of nodes is odd we round the number of nodes to be selected up) and form a Hamiltonian cycle (closed path) through this set of nodes such that the sum of the total length of the path plus the total cost of the selected nodes is minimized.

The distances between nodes are calculated as Euclidean distances rounded mathematically to integer values. The distance matrix should be calculated just after reading an instance, and then only the distance matrix (no nodes coordinates) should be accessed by optimization methods to allow instances defined only by distance matrices.

Pseudocode of implemented algorithms

- **Greedy 2-regret heuristics**

  // select randomly the starting index
  startNode = random node from nodeList
  Add startNode to currCycle
  Mark startNode as visited
  // **choose the nearest vertex and create two vertices cycle**
  Find the nearest neighbor to startNode and add to currCycle nearestNode
  Add nearestNode to currCycle
  Mark nearestNode as visited
  // **repeat until 50% of nodes in currCycle**
  for each **unvisited node j** do:
  // save the best position, lowest and second lowest increase:
  for each possible **position** in currCycle do:
  prevNode = node at current position in currCycle
  nextNode = node at next position
  increase = distanceMatrix(prevNode, j) + distanceMatrix(j, nextNode) - distanceMatrix(prevNode, nextNode) + nodeList.getCost(j)
  if increase < bestIncreaseforNode then:

secondBestIncrease = bestIncrease
bestIncrease = increase
nodePosition = position + 1
if increase<secondBestIncrease then:
secondBestIncrease = increase
regret = secondBestIncrease - bestIncrease
if regret > highestRegret then:
highestRegret = regret
bestNode = j
bestPosition = nodePosition
Insert bestNode into currCycle at bestPosition
Mark bestNode as visited
// **return currCycle**

- **Greedy 2-regret heuristics weighted sum**

// select randomly the starting index
startNode = random node from nodeList
Add startNode to currCycle
Mark startNode as visited
// **choose the nearest vertex and create two vertices cycle**
Find the nearest neighbor to startNode and add to currCycle nearestNode
Add nearestNode to currCycle
Mark nearestNode as visited
// **repeat until 50% of nodes in currCycle**
for each **unvisited node j** do:
// save the best position, lowest and second lowest increase:
for each **position** in currCycle
calculate increase in obj. function
// calculate regret
regret = secondBestIncrease - bestIncrease
// calculate weightedSum
weightedSum = (1-weight) * bestIncrease - weight * regret
// select the node with the smallest weighted sum
// **insert** the selected node into the best position in currCycle
insert bestNodeIndex at bestPosition in currCycle
Mark bestNodeIndex as visited
// **return currCycles**

Summary performance of each method

Instance: A

Summary for Objective Function Value

method	min	max	mean
Greedy2RegretMethod	105692	126951	116066.94
Greedy2RegretWeightedSumMethod	71108	73321	72123.08
GreedyCycleMethod	71719	75803	72987.8
NearNeighborEndMethod	83590	89433	85208.68
RandomMethod	239099	289258	264976.085

Instance: B

Summary for Objective Function Value

method	min	max	mean
Greedy2RegretMethod	68395	78406	72783.945
Greedy2RegretWeightedSumMethod	47144	55700	50904.245
GreedyCycleMethod	49001	57271	51568.455
NearNeighborEndMethod	52319	59030	54359.255
RandomMethod	189044	237132	213532.255

The difference in execution time of Greedy 2-Regret method and the Greedy 2-Regret with Weighted sum in not vary substantial but we can not say the same about the results. The

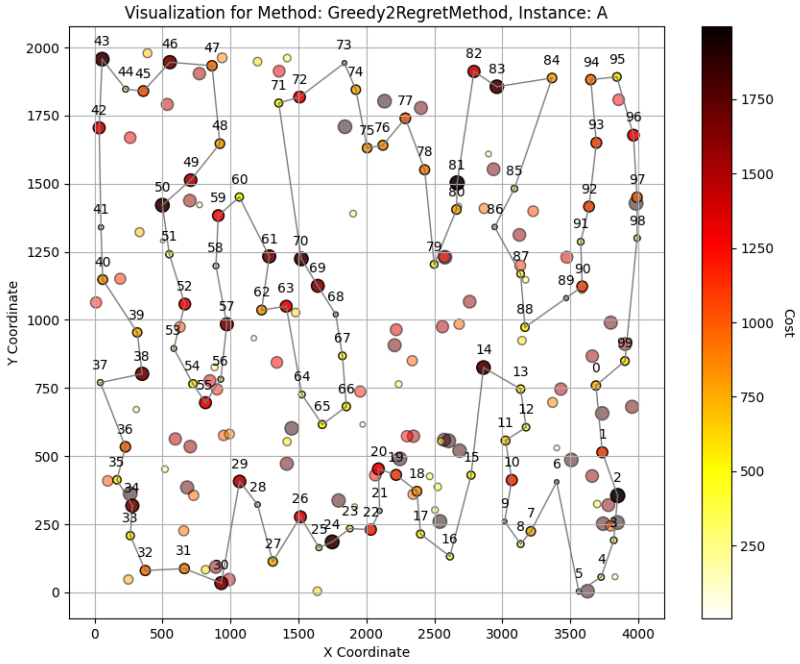
method with the weighted sum unsurprisingly has produced better results than the one basing on only regret.

2D visualisations of best solutions

Instance: A

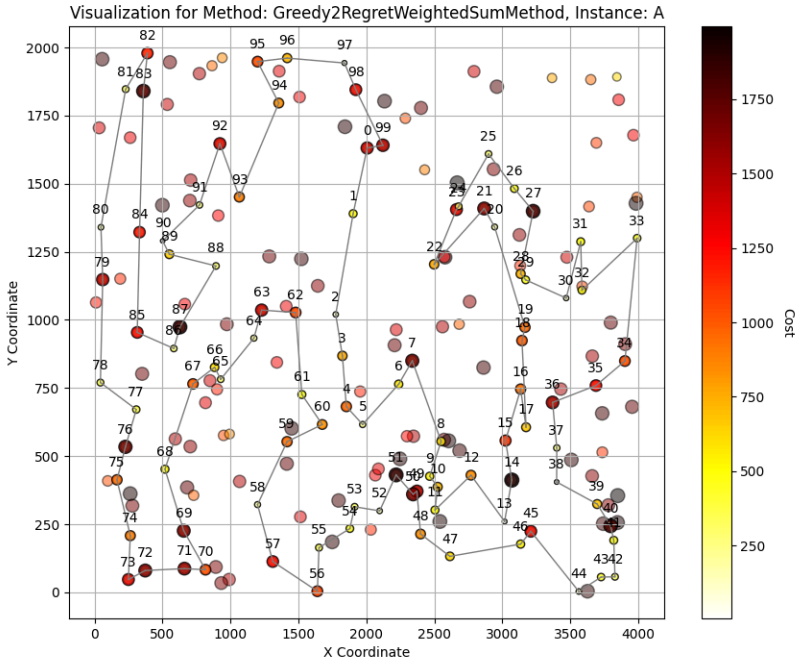
Method: Greedy2RegretMethod

Obj.f. value: 105692



Method: Greedy2RegretWeightedSumMethod

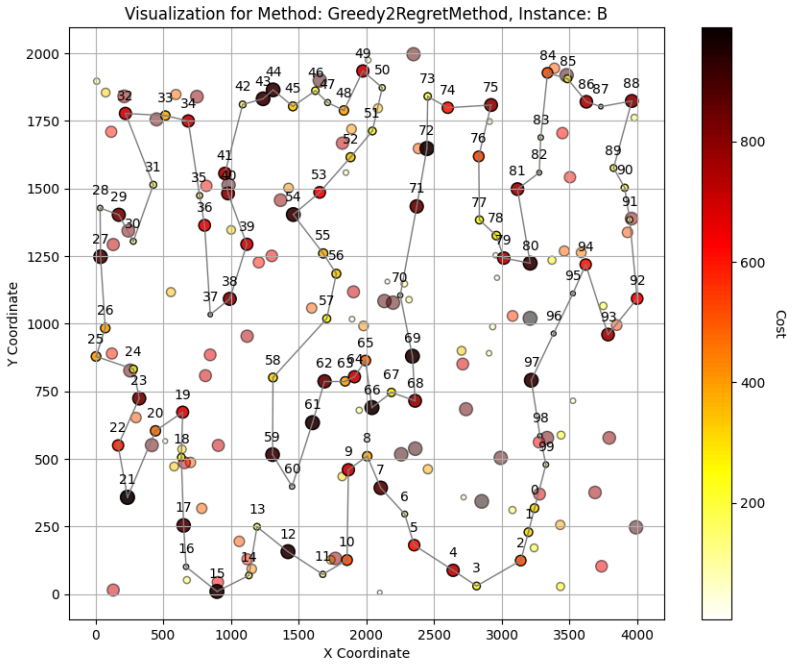
Obj.f. value: 71108



Instance: B

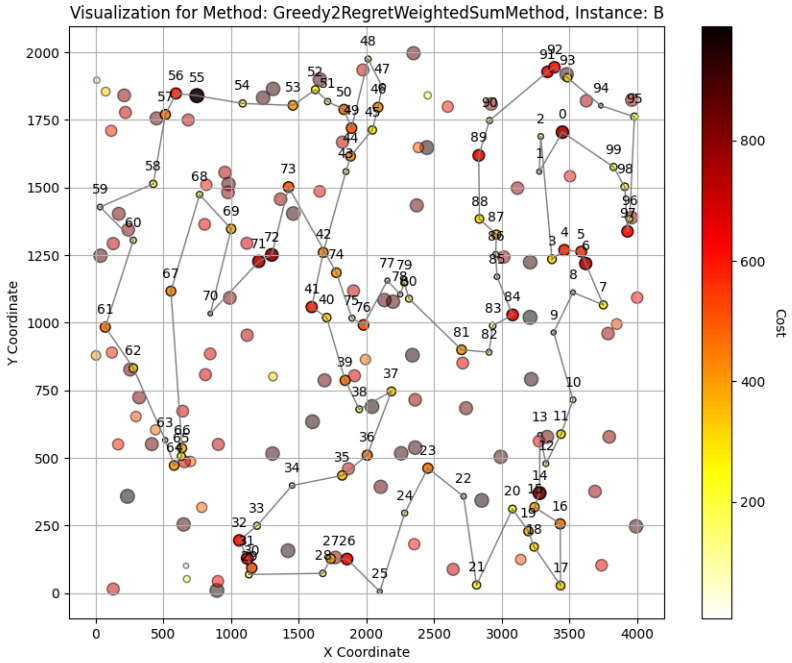
Method: Greedy2RegretMethod

Obj.f. value: 68395



Method: Greedy2RegretWeightedSumMethod

Obj.f. value: 47144



Best solutions, indices

Instance: A

Method: Greedy2RegretMethod

Lowest Objective Function Value (f_val): 105692

Solution:

196, 157, 188, 113, 171, 16, 78, 25, 44, 120, 82, 129, 92, 57, 172, 2, 75, 86, 26, 121, 182, 53, 158, 154, 6, 135, 194, 127, 123, 24, 156, 4, 190, 177, 104, 54, 48, 34, 192, 181, 146, 22, 20, 134, 18, 69, 67, 140, 68, 110, 142, 41, 96, 42, 43, 77, 65, 197, 115, 198, 46, 60, 118, 109, 151, 133, 79, 80, 176, 66, 141, 0, 153, 183, 89, 23, 186, 114, 15, 148, 9, 61, 73, 132, 21, 14, 49, 178, 52, 185, 119, 165, 39, 95, 7, 164, 71, 27, 90, 81

Method: Greedy2RegretWeightedSumMethod

Lowest Objective Function Value (f_val): 71108

Solution:

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, 43, 184, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 146, 22, 18, 108, 69, 159, 181, 42, 5, 115, 41, 193, 139, 68, 46, 0, 117, 143, 183, 89, 186

Instance: B

Method: Greedy2RegretMethod

Lowest Objective Function Value (f_val): 68395

Solution:

103, 89, 165, 187, 146, 97, 77, 58, 82, 87, 91, 36, 7, 5, 175, 46, 190, 105, 31, 151, 198, 196, 42, 24, 1, 27, 38, 92, 63, 96, 135, 122, 72, 133, 178, 90, 125, 121, 116, 98, 120, 71, 147, 192, 150, 6, 188, 169, 132, 161, 3, 145, 195, 43, 2, 139, 11, 138, 25, 123, 177, 171, 157, 104, 56, 144, 68, 111, 41, 37, 0, 69, 167, 155, 184, 53, 170, 34, 55, 83, 181, 174, 183, 140, 4, 28, 59, 20, 23, 148, 47, 94, 57, 52, 22, 185, 86, 64, 176, 113

Method: Greedy2RegretWeightedSumMethod

Lowest Objective Function Value (f_val): 47144

Solution:

199, 183, 140, 95, 130, 99, 22, 179, 185, 86, 166, 194, 113, 176, 26, 103, 114, 137, 127, 89, 163, 187, 153, 81, 77, 141, 91, 61, 36, 175, 78, 142, 45, 5, 177, 21, 82, 111, 8, 104, 138, 182, 139, 168, 195, 145, 15, 3, 70, 13, 132, 169, 188, 6, 147, 115, 10, 133, 122, 63, 135, 38, 1, 117, 193, 31, 54, 131, 90, 51, 121, 118, 74, 134, 11, 33, 160, 29, 0, 109, 35, 143, 106, 124, 128, 62, 18, 55, 34, 170, 152, 4, 149, 28, 20, 60, 94, 66, 47, 148

Conclusions

The execution of the Greedy 2-Regret method produced the best results in this weeks trials as well as overall methods. It was not surprising that taking into objective function both regret and the increase in the cost of the path worked better than only looking as regret in Greedy 2-Regret function, or looking at only the increase in the basic Greedy function. It was nice to see the bigger change in the path cost compared to the Greedy function for instance B as it was nearly 2000. As for instance A the decrease in path value was not so large but still significant as it was about 600. The graphs are a lot clearer than in the previous weeks' methods but comparable to the Greedy approach graphs.