POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

GREEDY REGRET HEURISTICS

Piotr Balewski, 156037
Lidia Wiśniewska, 156063

POZNAŃ 2025

1 Problem Description

The problem involves a set of nodes, each defined by three attributes:

- x and y coordinates representing the node's position in a 2D plane,
- cost associated with the node.

The objective is to select exactly 50% of the nodes (rounded up if the total number of nodes is odd) and construct a Hamiltonian cycle through the selected nodes. The aim is to minimize the sum of:

1. The total length of the Hamiltonian cycle, and
2. The total cost of the selected nodes.

Node-to-node distances are computed as Euclidean distances, rounded to the nearest integer.

1.1 Regret Heuristics

This lab implements algorithms based on a "regret" heuristic. Unlike a simple greedy heuristic, a regret heuristic calculates a score for each candidate node based on two criteria:

1. **Insertion Cost** (bestDelta): What is the lowest cost to insert this node anywhere in the current path/cycle?
2. **Regret:** How much more expensive are the next-best insertion options?

High "regret" means that if the node's best spot is taken, inserting it elsewhere is much more costly, making the node "urgent". The score is calculated as a weighted sum:

$$score = \text{regretWeight} \times \text{regret} - (1 - \text{regretWeight}) \times \text{bestDelta}$$

We test two variations (using $k = 2$ for regret calculation):

- **Pure Regret** (w=1.0)
- **Weighted** (w=0.5)

2 Implemented Algorithms

2.1 Regret Nearest Any

Builds an open path. Starts with a single node and iteratively adds the node with the best "regret" score into its best-fitting position.

```
procedure regretNearestAny(inst, start, k, weight)
  selected = [start], remaining = all nodes except start
  while |selected| < (n + 1) / 2 do
    bestNode = null, bestScore = -Inf
    for each remaining node j:
      deltas = []
      for each position pos in selected (incl. ends):
        calculate cost delta of inserting j at pos
        add delta to deltas

      sortedDeltas = sort(deltas)
      bestDelta = sortedDeltas[0]
      regret = calculate_regret(sortedDeltas, k)
      score = weight*regret - (1-weight)*bestDelta

      if score > bestScore:
        bestScore = score, bestNode = j
        bestPos = original index of bestDelta

    selected.insert(bestPos, bestNode)
    remaining.remove(bestNode)
  return (selected, EVALUATE(selected, inst, form_cycle=true))
```

2.2 Regret Greedy Cycle

Builds a closed cycle. Starts with a 2-node cycle (start + nearest neighbor) and iteratively inserts the node with the best "regret" score.

```
procedure regretGreedyCycle(inst, start, k, weight)
  remaining = all nodes except start
  bestSecond = find nearest neighbor of start
  selected = [start, bestSecond]
  remaining.remove(bestSecond)

  while |selected| < (n + 1) / 2 do
    bestNode = null, bestScore = -Inf
    for each remaining node j:
      deltas = []
      for each edge (i, kNode) in selected cycle:
        calculate cost delta of inserting j
        delta = dist(i,j) + dist(j,kNode) - dist(i,kNode)
        add (delta + cost(j)) to deltas
```

```

    // Calculate score same as in regretNearestAny
    bestDelta, regret, score = ...

    if score > bestScore:
      bestScore = score, bestNode = j
      bestPos = original index of bestDelta + 1

  selected.insert(bestPos, bestNode)
  remaining.remove(bestNode)
return (selected, EVALUATE(selected, inst, true))
```

3 Experimental Results

3.1 TSPA: regretNN (w=1.0)

Results: min: 108151 max: 124921 avg: 117138
Best path: [16, 175, 56, 31, 38, 157, 17, 196, 91, 57, 52, 106, 185, 8, 165, 39, 90, 27, 71, 164, 7, 21, 132, 14, 102, 128, 167, 111, 130, 148, 15, 64, 114, 186, 23, 89, 183, 153, 0, 141, 66, 176, 79, 133, 151, 109, 118, 59, 197, 116, 43, 42, 5, 96, 115, 198, 46, 68, 93, 36, 67, 108, 69, 199, 20, 22, 146, 103, 34, 160, 48, 30, 104, 177, 10, 190, 4, 112, 35, 184, 166, 131, 24, 123, 127, 70, 6, 154, 158, 53, 136, 121, 100, 97, 152, 87, 2, 129, 82, 25]

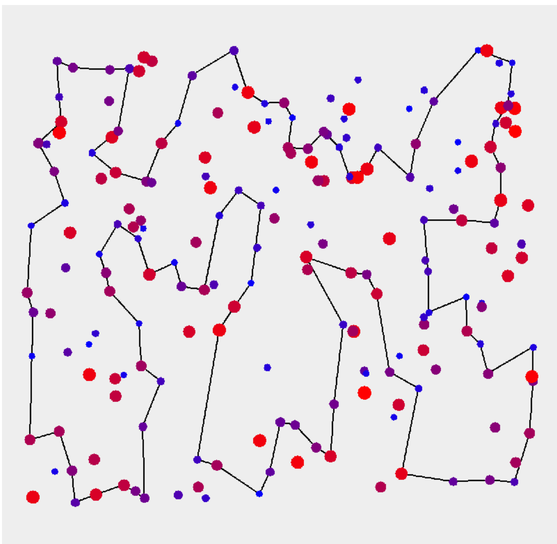

Figure 1: Best solution for regretNN (w=1.0), TSPA.

3.2 TSPA: regretGC (w=1.0)

Results: min: 105692 max: 126951 avg: 115579

Best path: [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]

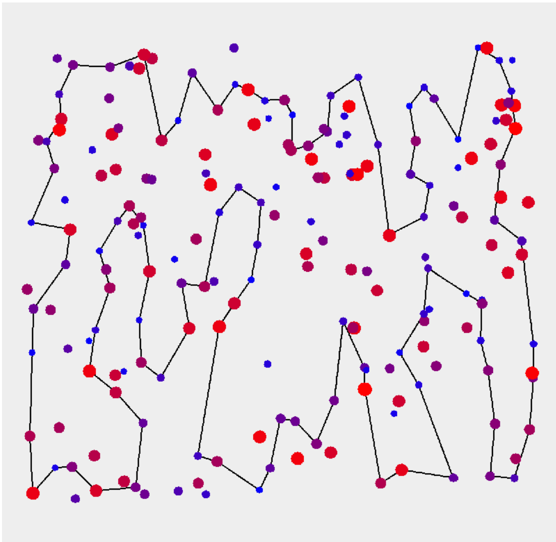

Figure 2: Best solution for regretGC (w=1.0), TSPA.

3.3 TSPA: NNregretWeight (w=0.5)

Results: min: 70010 max: 75452 avg: 72401
Best path: [108, 18, 199, 159, 22, 146, 181, 34, 160, 48, 54, 177, 184, 84, 4, 112, 35, 131, 149, 65, 116, 43, 42, 5, 41, 193, 139, 68, 46, 115, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 154, 180, 53, 121, 100, 26, 86, 75, 101, 1, 97, 152, 2, 120, 44, 25, 16, 171, 175, 113, 56, 31, 78, 145, 179, 92, 129, 57, 55, 52, 185, 40, 196, 81, 90, 165, 106, 178, 14, 49, 102, 144, 62, 9, 148, 124, 94, 63, 79, 80, 176, 137, 23, 89, 183, 143, 0, 117, 93, 140]

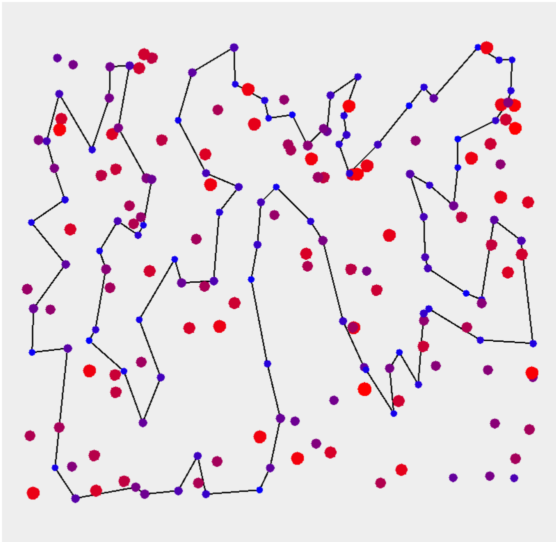

FIGURE 3: Best solution for NNregretWeight (w=0.5), TSPA.

3.4 TSPA: GCregretWeight (w=0.5)

Results: min: 71108 max: 73395 avg: 72130

Best path: [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, 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]

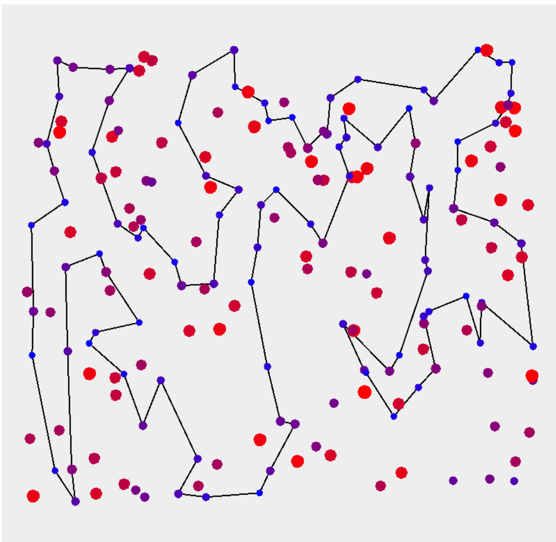

FIGURE 4: Best solution for GCregretWeight (w=0.5), TSPA.

3.5 TSPB: regretNN (w=1.0)

Results: min: 69933 max: 80278 avg: 74444

Best path: [129, 119, 159, 37, 41, 81, 77, 97, 146, 187, 165, 127, 137, 75, 93, 76, 194, 166, 86, 110, 128, 124, 62, 18, 34, 174, 183, 9, 99, 185, 179, 172, 57, 66, 47, 148, 23, 20, 59, 28, 4, 152, 184, 155, 84, 3, 15, 145, 13, 132, 169, 188, 6, 150, 147, 134, 2, 43, 139, 11, 0, 33, 104, 8, 82, 87, 79, 36, 7, 177, 123, 5, 78, 162, 80, 108, 196, 42, 156, 30, 117, 151, 173, 19, 112, 121, 116, 98, 51, 125, 191, 178, 10, 133, 44, 72, 40, 63, 92, 38]

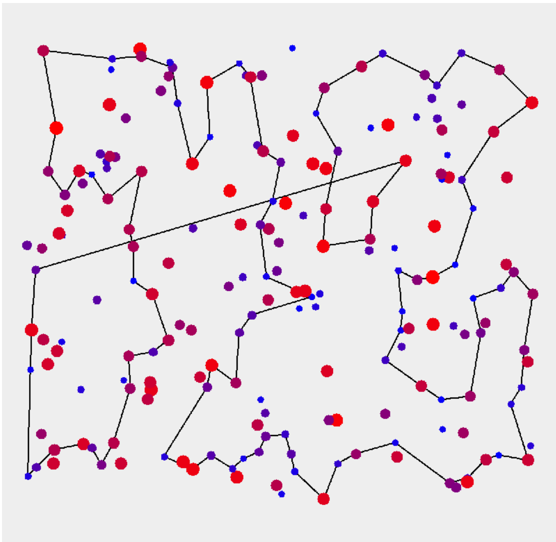

FIGURE 5: Best solution for regretNN (w=1.0), TSPB.

3.6 TSPB: regretGC (w=1.0)

Results: min: 67809 max: 78406 avg: 72740

Best path: [18, 34, 174, 183, 9, 99, 185, 179, 172, 57, 66, 47, 60, 20, 59, 28, 4, 53, 170, 184, 155, 84, 70, 132, 169, 188, 6, 192, 134, 2, 74, 118, 98, 51, 120, 71, 178, 10, 44, 17, 107, 100, 63, 102, 135, 131, 121, 112, 19, 173, 31, 117, 198, 24, 1, 27, 42, 196, 108, 80, 162, 142, 5, 123, 7, 36, 79, 91, 141, 97, 77, 58, 82, 68, 104, 33, 49, 29, 0, 41, 143, 119, 153, 186, 163, 103, 127, 137, 75, 93, 48, 166, 194, 180, 64, 86, 110, 128, 124, 62]

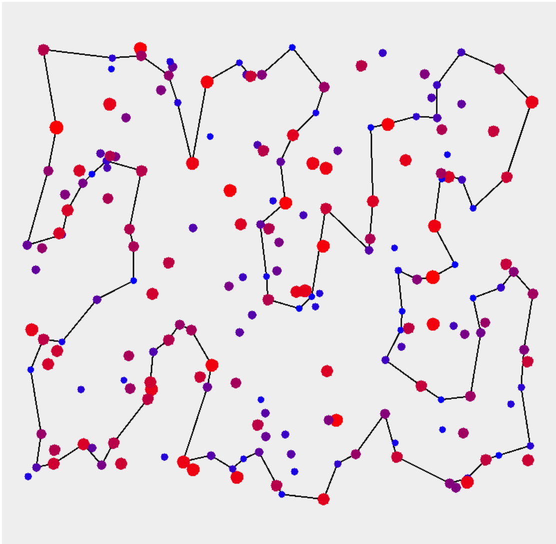

FIGURE 6: Best solution for regretGC (w=1.0), TSPB.

3.7 TSPB: NNregretWeight (w=0.5)

Results: min: 44891 max: 55247 avg: 47664

Best path: [131, 121, 51, 90, 147, 6, 188, 169, 132, 13, 168, 195, 145, 15, 70, 3, 155, 184, 152, 170, 34, 55, 18, 62, 124, 106, 128, 95, 130, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 57, 172, 179, 185, 86, 166, 194, 176, 113, 103, 127, 89, 163, 187, 153, 81, 77, 141, 91, 61, 36, 21, 82, 8, 104, 33, 160, 0, 35, 109, 29, 11, 138, 182, 25, 177, 5, 78, 175, 162, 80, 190, 136, 73, 31, 54, 193, 117, 198, 156, 1, 16, 27, 38, 135, 122, 63, 100, 107, 40]

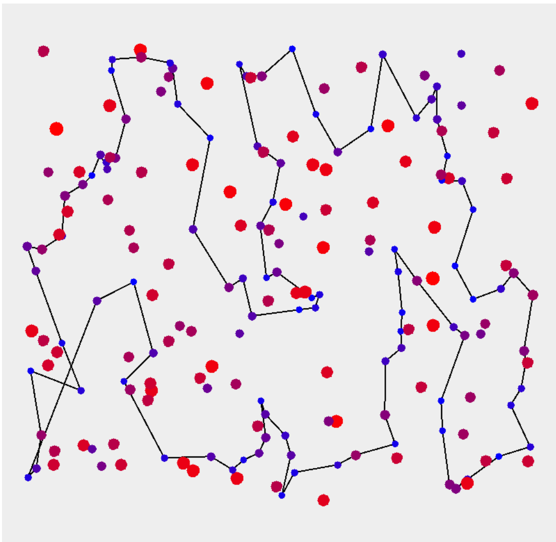

FIGURE 7: Best solution for NNregretWeight (w=0.5), TSPB.

3.8 TSPB: GCregretWeight (w=0.5)

Results: min: 47144 max: 55700 avg: 50897

Best path: [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]

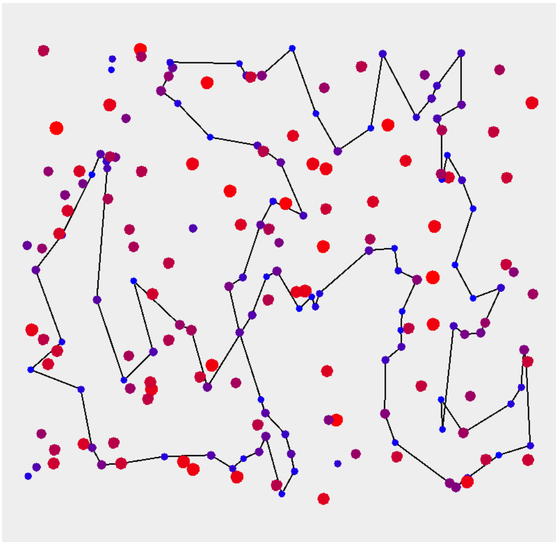

FIGURE 8: Best solution for GCregretWeight (w=0.5), TSPB.

3.9 Regret weight experiments

We experimented with different weight values for the TSPA instance. Based on the minimum values, the best results were obtained with $w = 0.6$ for Nearest Neighbor Any and $w = 0.7$ for Greedy Cycle.

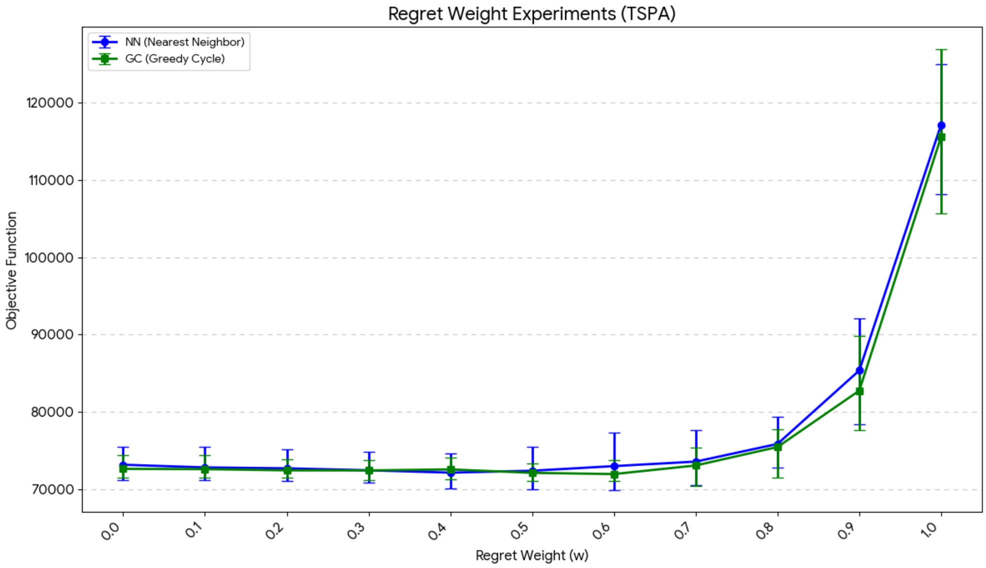

Figure 9: Results of weight experiments for TSPA.

Table 1: Results Summary

TSPA (Problem A)				TSPB (Problem B)			
Method	Min	Average	Max	Method	Min	Average	Max
Lab 1: Simple Greedy Heuristics				*Lab 1: Simple Greedy Heuristics*			
Random Solution	237322	262995	291386	Random Solution	185591	212627	240630
Nearest End	83182	85109	89433	Nearest End	52319	54390	59030
Nearest Any	71179	73179	75450	Nearest Any	44417	45870	53438
Greedy Cycle	71488	72646	74410	Greedy Cycle	49001	51401	57324
Lab 2: Regret Heuristics				*Lab 2: Regret Heuristics*			
`regretNN (w=1.0)`	108151	117138	124921	`regretNN (w=1.0)`	69933	74444	80278
`regretGC (w=1.0)`	105692	115579	126951	`regretGC (w=1.0)`	67809	72740	78406
`NNregretWeight (w=0.5)`	70010	72401	75452	`NNregretWeight (w=0.5)`	44891	47664	55247
`GCregretWeight (w=0.5)`	71108	72130	73395	`GCregretWeight (w=0.5)`	47144	50897	55700
`NNregretWeight (w=0.6)`	69932	73001	77323	`NNregretWeight (w=0.6)`	-	-	-
`GCregretWeight (w=0.7)`	70475	73082	75365	`GCregretWeight (w=0.7)`	-	-	-

4 Conclusions

This lab introduced regret-based heuristics to improve upon the simple greedy algorithms from the previous lab. The new methods, ‘regretNN‘ and ‘regretGC‘, were tested with two different scoring configurations: a pure regret model (weight=1.0) and a balanced model (weight=0.5).

- The choice of weighting was critical. Pure regret models (w=1.0) performed very poorly. In contrast, weighted heuristics (w=0.5 or w=0.7) that balance insertion cost and regret were highly effective.
- Regret Heuristics outperformed Greedy algorithms. This demonstrates that a well-tuned regret heuristic can find better solutions than a simple greedy approach.

Solution Checker: All best solutions obtained were verified using the provided solution checker.
Source Code: https://github.com/PBalewski/EvolutionaryComputation/tree/main/lab2