Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Selected Topics in Optimization
HW3: Heuristics for TSP

Oscar Felipe Renaut Vega

2226683

ITS

Group 002 – Tuesdays M4-M6

Dr. Roger Z. Rios Mercado

Due date: March 08, 2026

Introduction

The Traveling Salesman Problem also known as TSP is a well known problem in operations research. The idea is simple. A traveler must visit several cities and return to the starting city while visiting each city only once. The goal is to find the route with the smallest total distance. One way to see this is like planning a road trip where you want to visit every place but also avoid traveling more distance than necessary.

When the number of cities increases the problem becomes much harder. The number of possible routes grows very quickly. This means that checking every possible route is usually not practical. Because of this the TSP is considered an NP hard problem. In other words finding the exact best solution can take too much time for larger problems.

To deal with this difficulty people often use heuristic methods. A heuristic is a strategy that helps find a good solution without testing every route. These methods are faster and usually give solutions that are close to the best one. This helps solve the problem in a reasonable amount of time.

In this homework two heuristics are used to solve a symmetric TSP with eight cities

- Nearest Neighbor Heuristic

This method starts from one city and moves to the closest city that has not been visited yet. The process repeats until all cities are visited and the route returns to the start.

- Nearest Insertion Heuristic

This method starts with the two closest cities and forms a small route. After that new cities are inserted into the tour in the position that increases the distance the least.

The steps and calculations show how the route is built and help understand how each heuristic works.

Problem Data

City Coordinates

City	x	y
C1	86	37
C2	17	94
C3	3	65
C4	48	43
C5	78	70
C6	17	55
C7	62	91
C8	78	91

Distance Matrix

	C1	C2	C3	C4	C5	C6	C7	C8
C1	0	89	87	38	33	71	59	54
C2	89	0	32	59	65	39	45	61
C3	87	32	0	50	75	17	64	79
C4	38	59	50	0	40	33	50	56
C5	33	65	75	40	0	62	26	21
C6	71	39	17	33	62	0	57	70
C7	59	45	64	50	26	57	0	16
C8	54	61	79	56	21	70	16	0

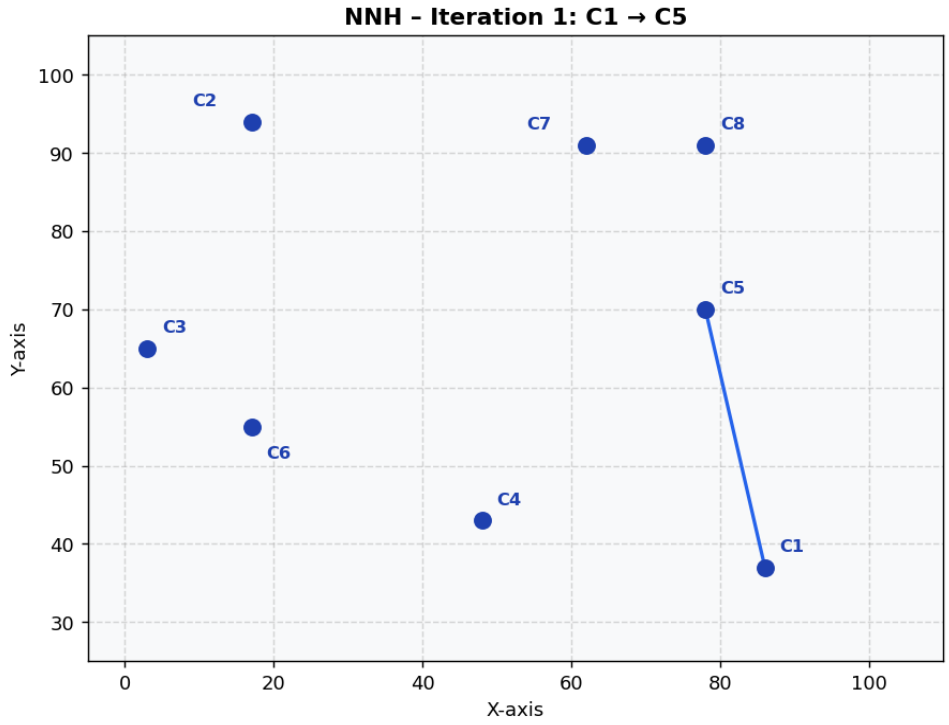
Note: distances are rounded Euclidean values. The matrix is symmetric (d(i,j) = d(j,i)) and the diagonal is zero.

Nearest Neighbor Heuristic (NNH) for the TSP

The algorithm starts at city C1 and at each step moves to the nearest unvisited city. Once all cities have been visited, it returns to city C1 to complete the tour.

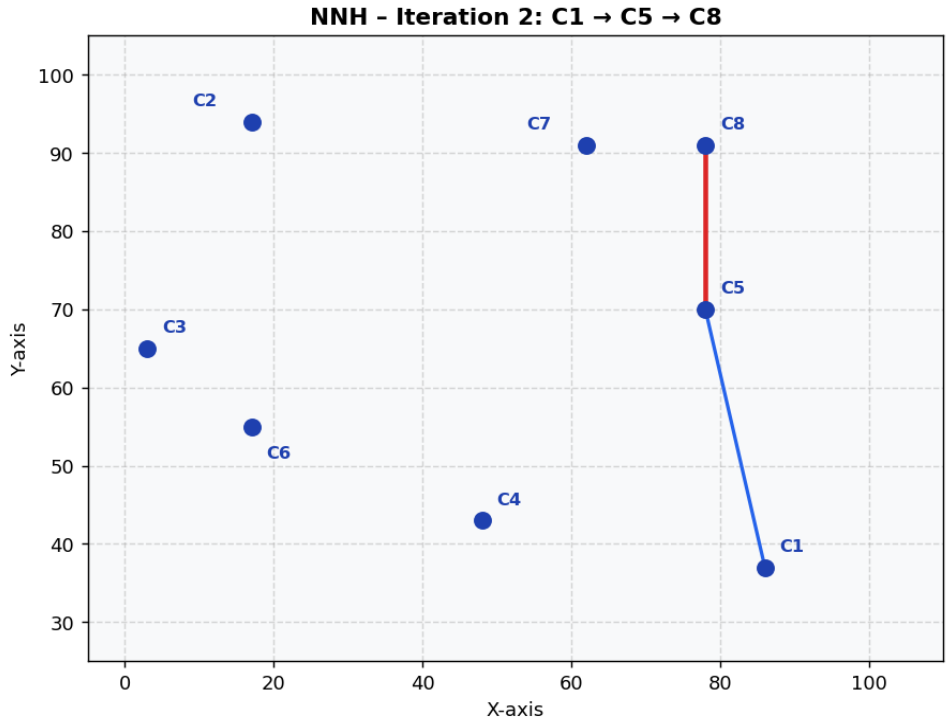
Iteration 1

Current city: C1 Unvisited: {C2, C3, C4, C5, C6, C7, C8}
Distances from C1 to all unvisited cities:
d(C1,C2) = 89 d(C1,C3) = 87 d(C1,C4) = 38
d(C1,C5) = 33 ← minimum d(C1,C6) = 71
d(C1,C7) = 59 d(C1,C8) = 54
Decision: Go to city C5 (d = 33).
Partial tour: C1 → C5 Accumulated distance: 33



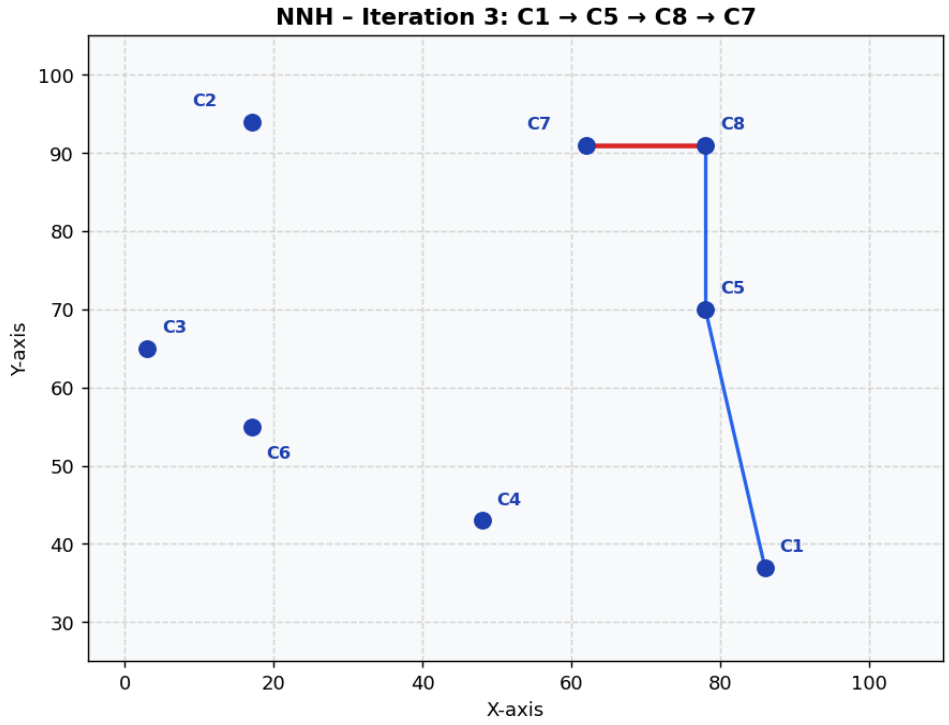
Iteration 2

Current city: C5 Unvisited: {C2, C3, C4, C6, C7, C8}
Distances from C5 to all unvisited cities:
d(C5,C2) = 65 d(C5,C3) = 75 d(C5,C4) = 40
d(C5,C6) = 62 d(C5,C7) = 26
d(C5,C8) = 21 ← minimum
Decision: Go to city C8 (d = 21).
Partial tour: C1 → C5 → C8 Accumulated distance: 33 + 21 = 54



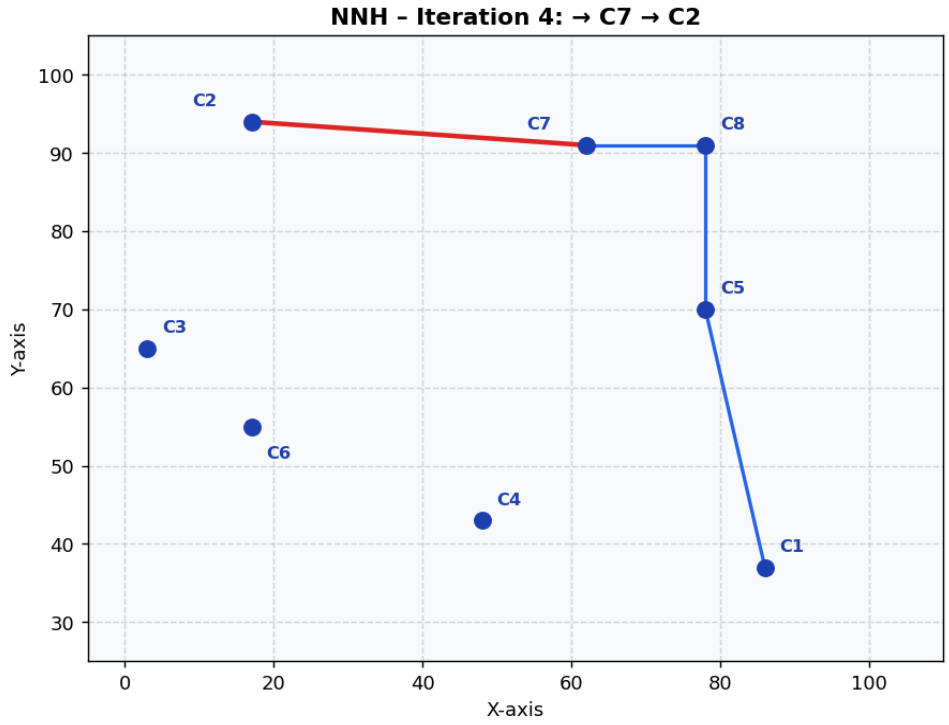
Iteration 3

Current city: C8 Unvisited: {C2, C3, C4, C6, C7}
Distances from C8 to all unvisited cities:
d(C8,C2) = 61 d(C8,C3) = 79 d(C8,C4) = 56
d(C8,C6) = 70
d(C8,C7) = 16 ← minimum
Decision: Go to city C7 (d = 16).
Partial tour: C1 → C5 → C8 → C7 Accumulated distance: 33 + 21 + 16 = 70



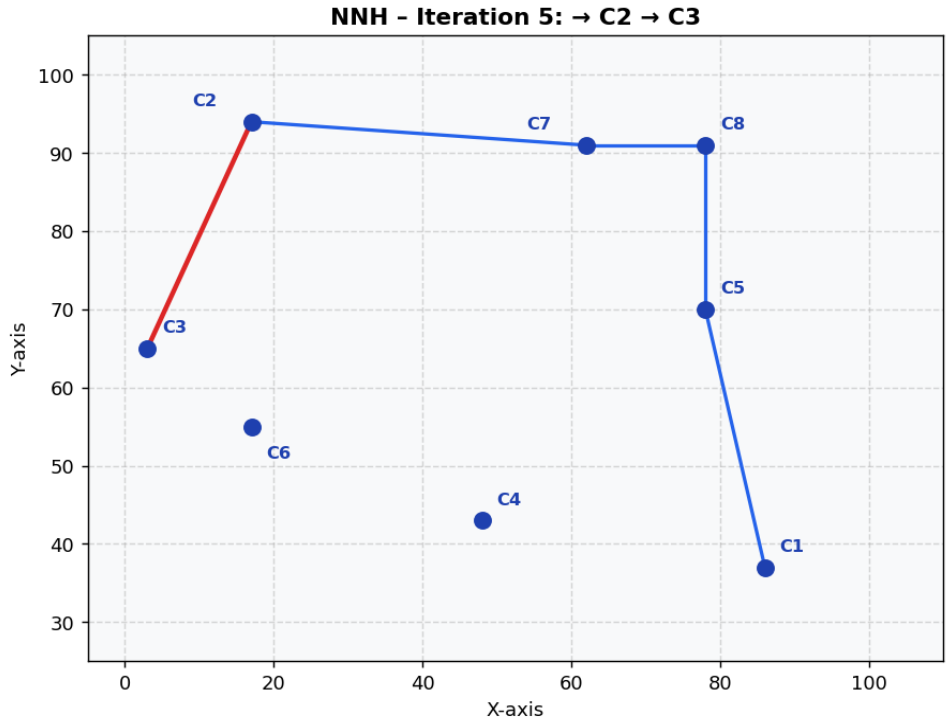
Iteration 4

Current city: C7 Unvisited: {C2, C3, C4, C6}
Distances from C7 to all unvisited cities:
d(C7,C2) = 45 ← minimum d(C7,C3) = 64
d(C7,C4) = 50 d(C7,C6) = 57
Decision: Go to city C2 (d = 45).
Partial tour: C1 → C5 → C8 → C7 → C2 Accumulated distance: 33 + 21 + 16 + 45 = 115



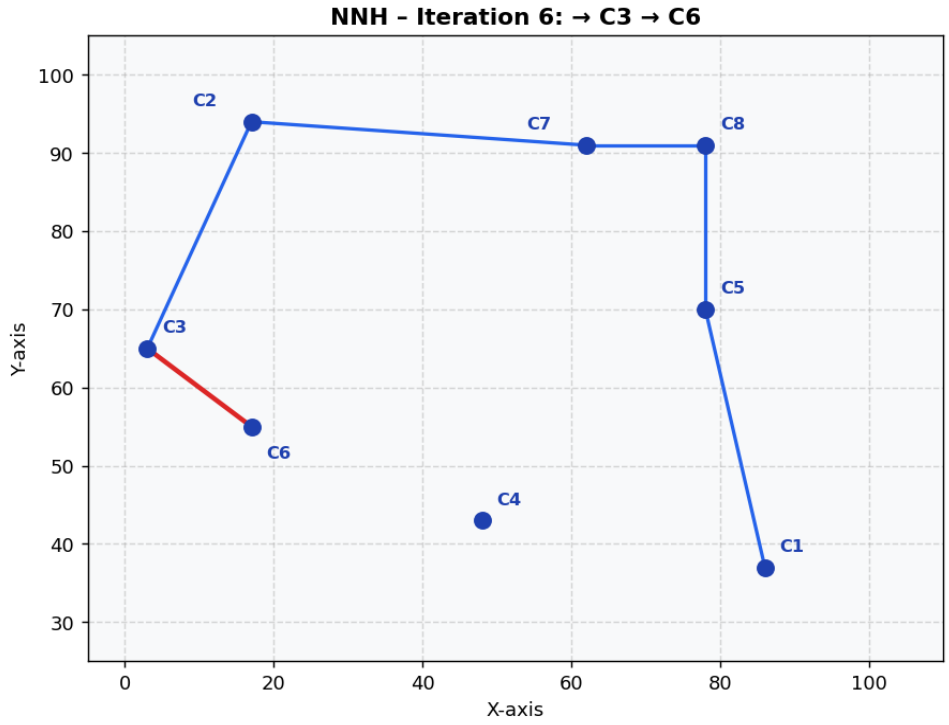
Iteration 5

Current city: C2 Unvisited: {C3, C4, C6}
Distances from C2 to all unvisited cities:
d(C2,C3) = 32 ← minimum d(C2,C4) = 59 d(C2,C6) = 39
Decision: Go to city C3 (d = 32).
Partial tour: C1 → C5 → C8 → C7 → C2 → C3 Accumulated distance: 33 + 21 + 16 + 45 + 32 = 147



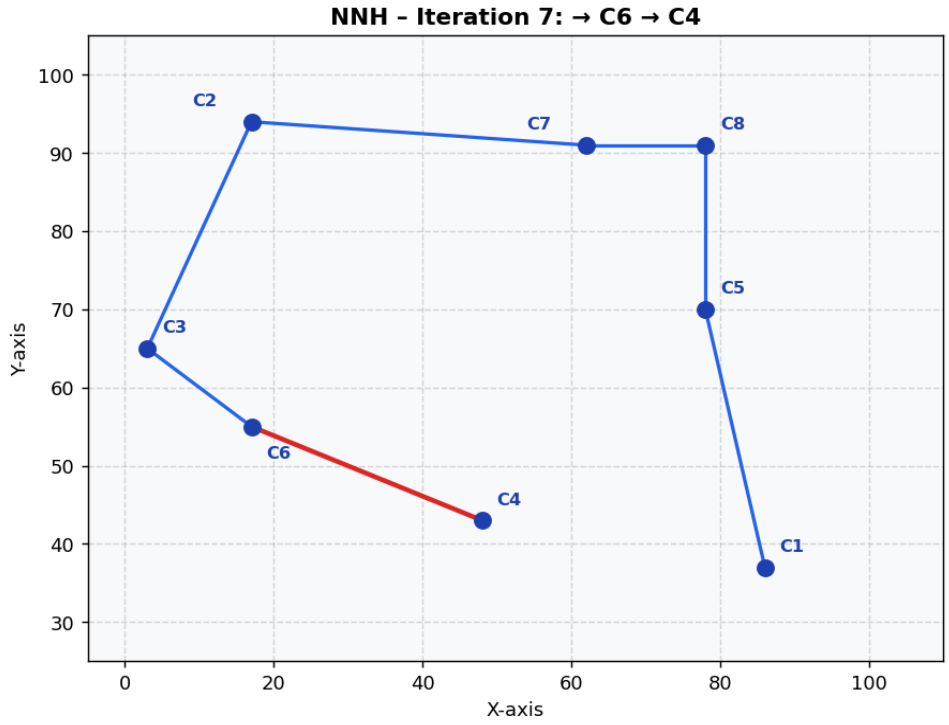
Iteration 6

Current city: C3 Unvisited: {C4, C6}
Distances from C3 to all unvisited cities:
d(C3,C4) = 50
d(C3,C6) = 17 ← minimum
Decision: Go to city C6 (d = 17).
Partial tour: C1 → ... → C2 → C3 → C6 Accumulated distance: 33 + 21 + 16 + 45 + 32 + 17 = 164



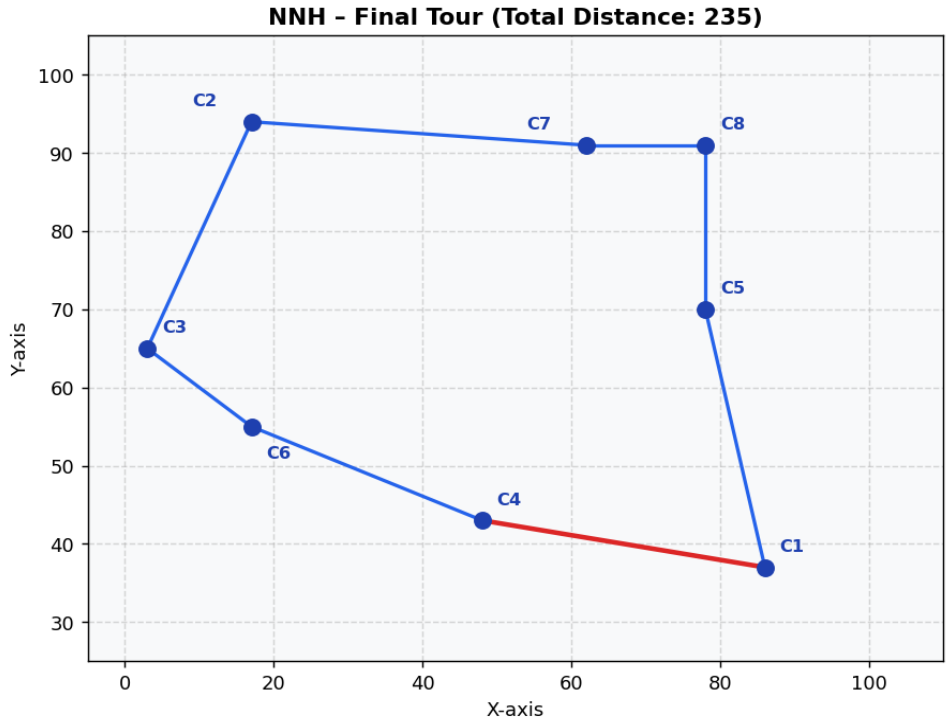
Iteration 7

Current city: C6 Unvisited: {C4}
Only one city remains:
d(C6,C4) = 33 ← only option
Decision: Go to city C4 (d = 33).
Partial tour: C1 → ... → C3 → C6 → C4 Accumulated distance: 33 + 21 + 16 + 45 + 32 + 17 + 33 = 197



Iteration 8 – Return to Start

All cities visited. Return from C4 to starting city C1:
d(C4,C1) = 38
Final tour: C1 → C5 → C8 → C7 → C2 → C3 → C6 → C4 → C1
Total distance: 33 + 21 + 16 + 45 + 32 + 17 + 33 + 38 = 235



NNH Summary

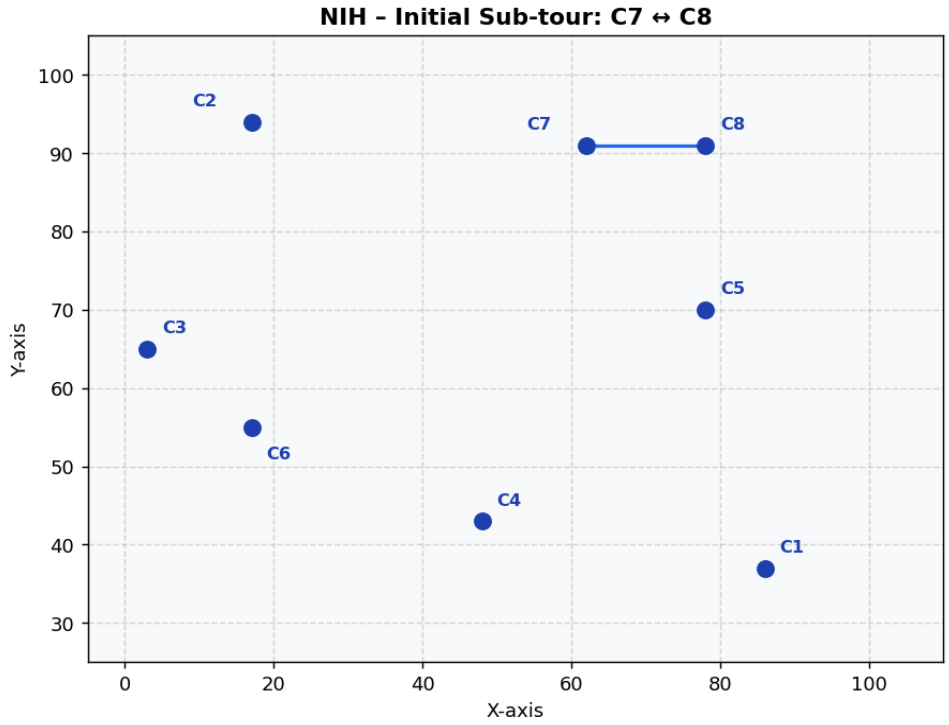
Segment	Distance
C1 → C5	33
C5 → C8	21
C8 → C7	16
C7 → C2	45
C2 → C3	32
C3 → C6	17
C6 → C4	33
C4 → C1	38
TOTAL	235

Nearest Insertion Heuristic (NIH) for the TSP

The algorithm starts by finding the two closest cities in the entire distance matrix, forming an initial sub-tour. At each iteration it: (1) finds the unvisited city with minimum distance R(i) to the current tour, (2) determines the cheapest position to insert it using e(r) = d(u,q) + d(q,v) − d(u,v).

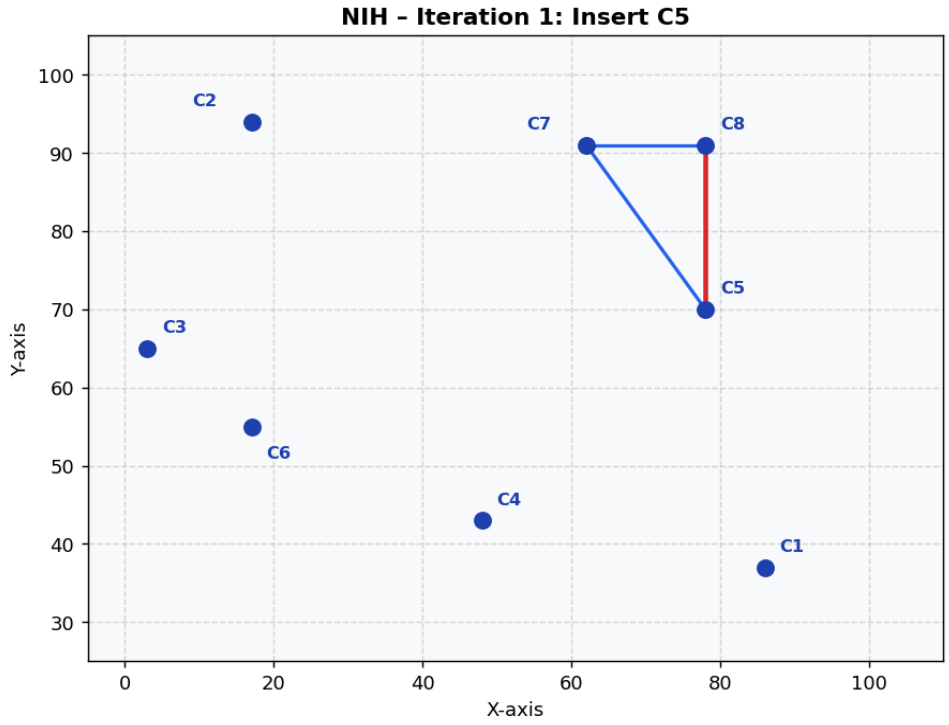
Initialization – Find the Two Closest Cities

Scanning the full distance matrix for the minimum off-diagonal entry:
d(C7,C8) = d(C8,C7) = 16 ← global minimum
Initial sub-tour: C7 → C8 → C7
Initial tour cost: d(C7,C8) + d(C8,C7) = 16 + 16 = 32



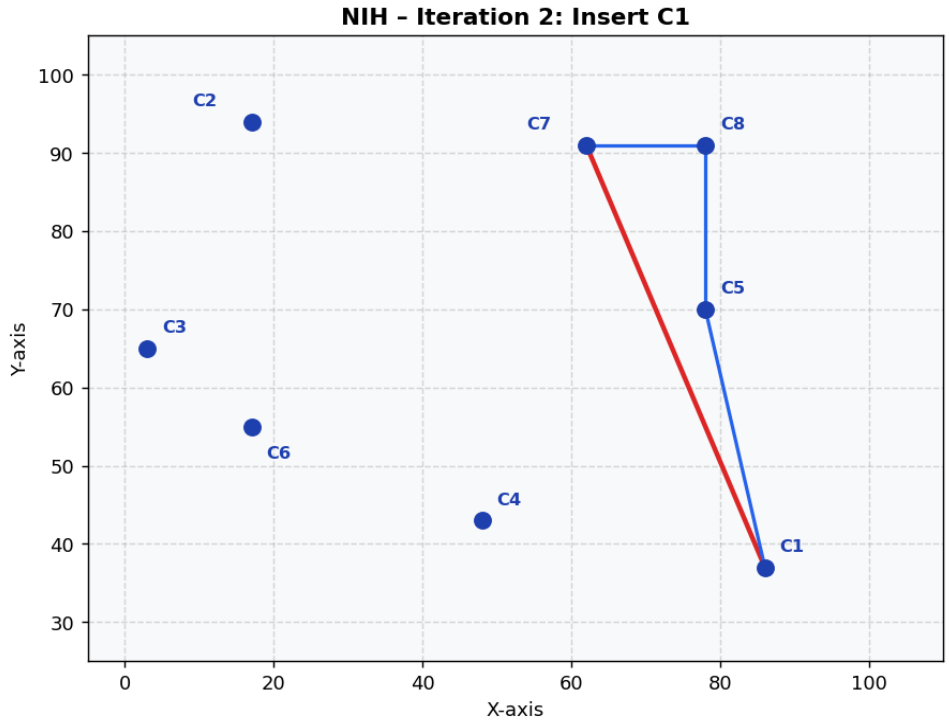
Iteration 1

Current tour set: {C7, C8} Unvisited: {C1, C2, C3, C4, C5, C6}
Compute R(i) = min distance from each unvisited city to the tour:
C1: min[d(C1,C7)=59, d(C1,C8)=54] = 54
C2: min[d(C2,C7)=45, d(C2,C8)=61] = 45
C3: min[d(C3,C7)=64, d(C3,C8)=79] = 64
C4: min[d(C4,C7)=50, d(C4,C8)=56] = 50
C5: min[d(C5,C7)=26, d(C5,C8)=21] = 21 ← minimum
C6: min[d(C6,C7)=57, d(C6,C8)=70] = 57
Decision: Choose city C5 (R = 21).
Only one edge to break (C7–C8). Insertion cost:
e = d(C7,C5) + d(C8,C5) − d(C7,C8) = 26 + 21 − 16 = 31
Updated tour: C7 → C8 → C5 → C7 Tour cost: 26 + 21 + 16 = 63



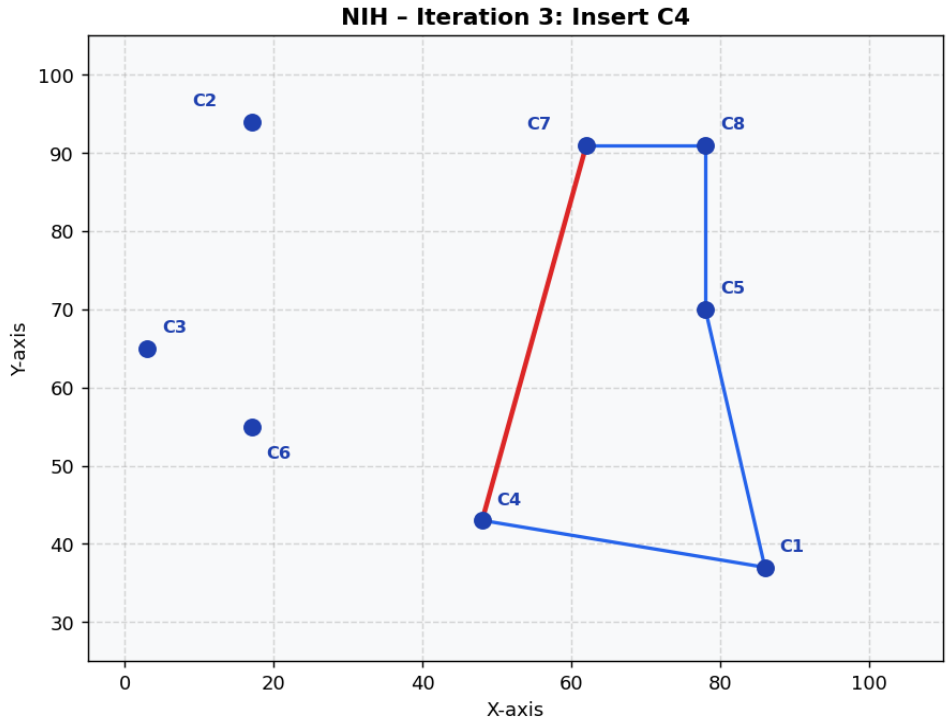
Iteration 2

Current tour: C7 → C8 → C5 → C7 Unvisited: {C1, C2, C3, C4, C6}
C1: min[d(C1,C7)=59, d(C1,C8)=54, d(C1,C5)=33] = 33 ← minimum
C2: min[d(C2,C7)=45, d(C2,C8)=61, d(C2,C5)=65] = 45
C3: min[d(C3,C7)=64, d(C3,C8)=79, d(C3,C5)=75] = 64
C4: min[d(C4,C7)=50, d(C4,C8)=56, d(C4,C5)=40] = 40
C6: min[d(C6,C7)=57, d(C6,C8)=70, d(C6,C5)=62] = 57
Decision: Choose city C1 (R = 33).
Compute insertion cost e(r) for each edge:
Between C7 & C8: d(C7,C1)+d(C1,C8)−d(C7,C8) = 59+54−16 = 97
Between C8 & C5: d(C8,C1)+d(C1,C5)−d(C8,C5) = 54+33−21 = 66
Between C5 & C7: d(C5,C1)+d(C1,C7)−d(C5,C7) = 33+59−26 = 66 ← tie (first chosen)
Decision: Insert C1 between C7 and C5 (or equivalently between C8 and C5 – tie).
Updated tour: C7 → C1 → C5 → C8 → C7 Tour cost: 59 + 33 + 21 + 16 = 129



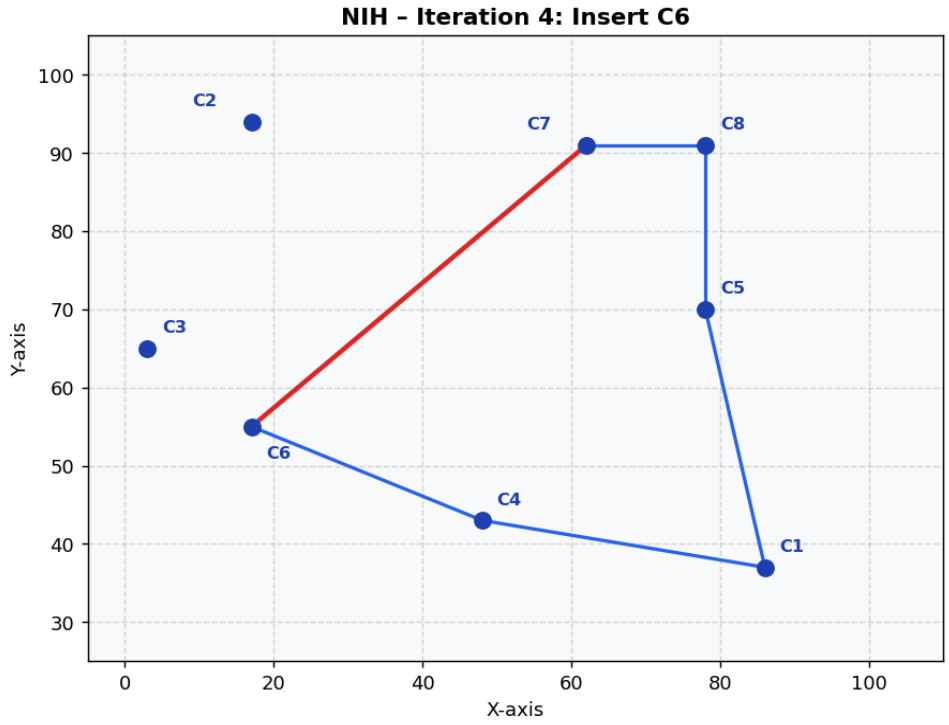
Iteration 3

Current tour: C7 → C1 → C5 → C8 → C7 Unvisited: {C2, C3, C4, C6}
C2: min[d(C2,C7)=45, d(C2,C1)=89, d(C2,C5)=65, d(C2,C8)=61] = 45
C3: min[d(C3,C7)=64, d(C3,C1)=87, d(C3,C5)=75, d(C3,C8)=79] = 64
C4: min[d(C4,C7)=50, d(C4,C1)=38, d(C4,C5)=40, d(C4,C8)=56] = 38 ← minimum
C6: min[d(C6,C7)=57, d(C6,C1)=71, d(C6,C5)=62, d(C6,C8)=70] = 57
Decision: Choose city C4 (R = 38).
Insertion cost e(r):
Between C7 & C1: d(C7,C4)+d(C4,C1)−d(C7,C1) = 50+38−59 = 29 ← minimum
Between C1 & C5: d(C1,C4)+d(C4,C5)−d(C1,C5) = 38+40−33 = 45
Between C5 & C8: d(C5,C4)+d(C4,C8)−d(C5,C8) = 40+56−21 = 75
Between C8 & C7: d(C8,C4)+d(C4,C7)−d(C8,C7) = 56+50−16 = 90
Decision: Insert C4 between C7 and C1.
Updated tour: C7 → C4 → C1 → C5 → C8 → C7 Tour cost: 50 + 38 + 33 + 21 + 16 = 158



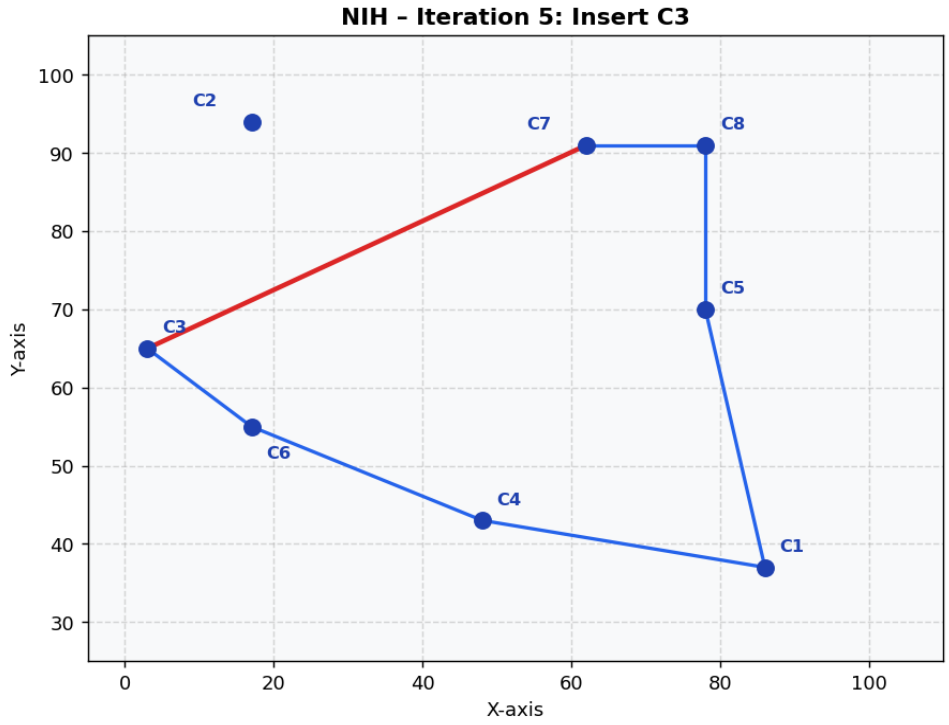
Iteration 4

Current tour: C7 → C4 → C1 → C5 → C8 → C7 Unvisited: {C2, C3, C6}
C2: min[d(C2,C7)=45, d(C2,C4)=59, d(C2,C1)=89, d(C2,C5)=65, d(C2,C8)=61] = 45
C3: min[d(C3,C7)=64, d(C3,C4)=50, d(C3,C1)=87, d(C3,C5)=75, d(C3,C8)=79] = 50
C6: min[d(C6,C7)=57, d(C6,C4)=33, d(C6,C1)=71, d(C6,C5)=62, d(C6,C8)=70] = 33 ← minimum
Decision: Choose city C6 (R = 33).
Insertion cost e(r):
Between C7 & C4: d(C7,C6)+d(C6,C4)−d(C7,C4) = 57+33−50 = 40 ← minimum
Between C4 & C1: d(C4,C6)+d(C6,C1)−d(C4,C1) = 33+71−38 = 66
Between C1 & C5: d(C1,C6)+d(C6,C5)−d(C1,C5) = 71+62−33 = 100
Between C5 & C8: d(C5,C6)+d(C6,C8)−d(C5,C8) = 62+70−21 = 111
Between C8 & C7: d(C8,C6)+d(C6,C7)−d(C8,C7) = 70+57−16 = 111
Decision: Insert C6 between C7 and C4.
Updated tour: C7 → C6 → C4 → C1 → C5 → C8 → C7 Tour cost: 57 + 33 + 38 + 33 + 21 + 16 = 198



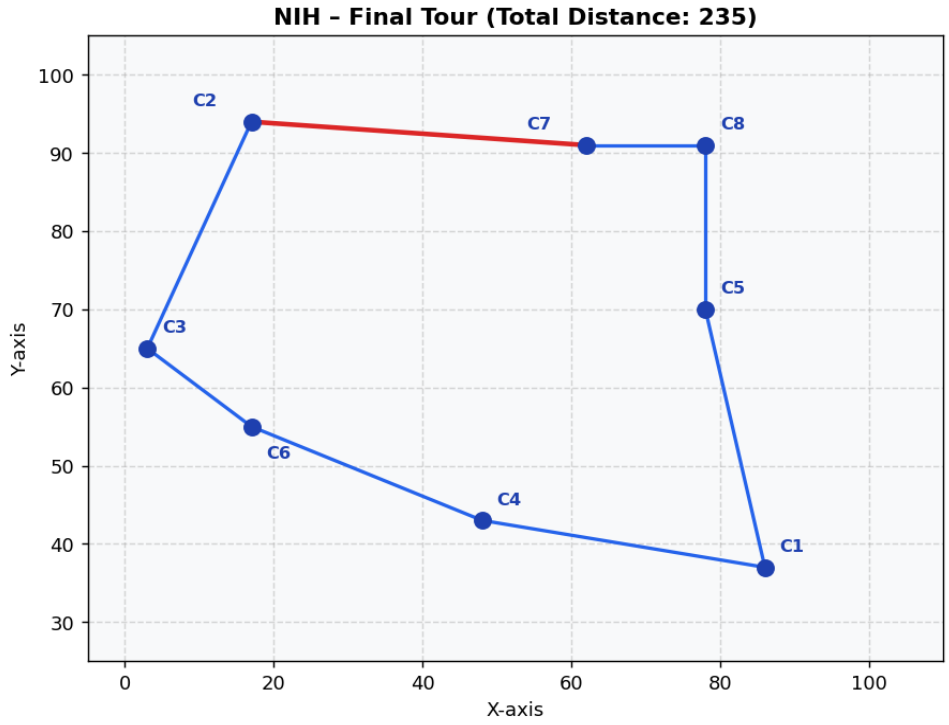
Iteration 5

Current tour: C7 → C6 → C4 → C1 → C5 → C8 → C7 Unvisited: {C2, C3}
C2: min[d(C2,C7)=45, d(C2,C6)=39, d(C2,C4)=59, d(C2,C1)=89, d(C2,C5)=65, d(C2,C8)=61] = 39
C3: min[d(C3,C7)=64, d(C3,C6)=17, d(C3,C4)=50, d(C3,C1)=87, d(C3,C5)=75, d(C3,C8)=79] = 17 ← min
Decision: Choose city C3 (R = 17).
Insertion cost e(r):
Between C7 & C6: d(C7,C3)+d(C3,C6)−d(C7,C6) = 64+17−57 = 24 ← minimum
Between C6 & C4: d(C6,C3)+d(C3,C4)−d(C6,C4) = 17+50−33 = 34
Between C4 & C1: d(C4,C3)+d(C3,C1)−d(C4,C1) = 50+87−38 = 99
Between C1 & C5: d(C1,C3)+d(C3,C5)−d(C1,C5) = 87+75−33 = 129
Between C5 & C8: d(C5,C3)+d(C3,C8)−d(C5,C8) = 75+79−21 = 133
Between C8 & C7: d(C8,C3)+d(C3,C7)−d(C8,C7) = 79+64−16 = 127
Decision: Insert C3 between C7 and C6.
Updated tour: C7 → C3 → C6 → C4 → C1 → C5 → C8 → C7 Tour cost: 64 + 17 + 33 + 38 + 33 + 21 + 16 = 222



Iteration 6 – Final Insertion

Current tour: C7 → C3 → C6 → C4 → C1 → C5 → C8 → C7 Unvisited: {C2}
Only city C2 remains. Compute insertion cost for all edges:
Between C7 & C3: d(C7,C2)+d(C2,C3)−d(C7,C3) = 45+32−64 = 13 ← minimum
Between C3 & C6: d(C3,C2)+d(C2,C6)−d(C3,C6) = 32+39−17 = 54
Between C6 & C4: d(C6,C2)+d(C2,C4)−d(C6,C4) = 39+59−33 = 65
Between C4 & C1: d(C4,C2)+d(C2,C1)−d(C4,C1) = 59+89−38 = 110
Between C1 & C5: d(C1,C2)+d(C2,C5)−d(C1,C5) = 89+65−33 = 121
Between C5 & C8: d(C5,C2)+d(C2,C8)−d(C5,C8) = 65+61−21 = 105
Between C8 & C7: d(C8,C2)+d(C2,C7)−d(C8,C7) = 61+45−16 = 90
Decision: Insert C2 between C7 and C3.
Final tour: C7 → C2 → C3 → C6 → C4 → C1 → C5 → C8 → C7
Total distance: 45 + 32 + 17 + 33 + 38 + 33 + 21 + 16 = 235



NIH Summary

Segment	Distance
C7 → C2	45
C2 → C3	32
C3 → C6	17
C6 → C4	33
C4 → C1	38
C1 → C5	33
C5 → C8	21
C8 → C7	16
TOTAL	235

Comparison of Results

Heuristic	Final Tour	Total Cost
NNH	C1→C5→C8→C7→C2→C3→C6→C4→C1	235
NIH	C7→C2→C3→C6→C4→C1→C5→C8→C7	235

In this instance, both heuristics produced a tour with the same total distance of 235 units. However, their tour structures differ: NNH follows the path
C1→C5→C8→C7→C2→C3→C6→C4→C1, while NIH produces
C7→C2→C3→C6→C4→C1→C5→C8→C7. Since both are valid representations of the same underlying cycle, it is worth noting that the two tours are actually the same cycle traversed from different starting points.

Conclusion

This homework showed how two classical heuristics can be used to solve the Traveling Salesman Problem with eight cities. The objective was to build a route that visits every city once and returns to the starting point while keeping the total distance as small as possible. Instead of testing every possible route the problem was solved using two constructive strategies that build the tour step by step.

The first method was the Nearest Neighbor Heuristic. This approach is simple and fast because it always chooses the closest city that has not been visited yet. This helps build a route quickly and is easy to follow. However sometimes this method can make decisions early that are not the best for the whole route. In other words it focuses only on the next step and not on the overall structure of the tour.

The second method was the Nearest Insertion Heuristic. This method builds a small route first using the two closest cities and then inserts the remaining cities one by one. Each city is placed where it increases the total distance the least. Because of this the route usually grows in a more organized way and often produces better shaped tours.

In this particular case both heuristics produced the same total distance of 235 units. This shows that for small problems both methods can give very similar results. In my opinion these heuristics are very useful because they help understand how the solution is built step by step and they make the problem easier to approach. They also work well as a starting point since the route they produce can later be improved with more advanced optimization methods.