Our program is a popular topic in modeling, which is Travelling Salesman Problem. When we were assigned to the project our first approach was to find as many as possible viable similar programs. We have even watch lectures on the internet about Travelling Salesman Problem just to have a deeper understanding. After that, we have searched for similar code written in visual basic but we were not able to find many sources on that. Since we couldn’t find a beneficial source on VBA we changed our approach to other languages. There were a lot of application of TSP in C/C++ and Python. From the information, we have gathered with our research we decided to apply an algorithm on Nearest Neighbor like this,

1. Find each node's distance to every other node.

2. Set your starting point to since it is random

3. Create a binary tableau based on whether that road is available to travel or not (true for traveled false for not traveled)

4. Find the minimum distance to the starting point. If it is an available destination then go, if not find the next minimum.

5. Update the Route Array

6. Update the Binary tableau, Set True every binary in the End Point column (It means you cannot travel to this node from anywhere)

7. If every node is travelled exit function. If not repeat from step 4

1. Sub TSP()
2. Dim Size As Integer
3. Dim distance() As Single
4. Dim traveled() As Boolean
5. Dim Route() As Integer
6. Dim totalDistance As Single
7. Dim MatrixYcounter As Integer
8. Dim StartPoint As Integer
9. Dim EndPoint As Integer
10. Dim minDistance As Single 'just to use in the distance loop
11. Dim i As Integer, j As Integer, k As Integer, l As Integer
12. Dim x1 As Integer, x2 As Integer, y1 As Integer, y2 As Integer
13. Dim temp\_dist As Single
14. Dim coords As Range 'this is the table of coordinates
16. '2 opt variables
18. Dim Newroute() As Integer
19. Dim Temproute1 As Integer
20. Dim Temproute2 As Integer
21. Dim Newroutedistance As Single
22. Dim Iterationcounter As Integer
23. Dim Iterationflag As Boolean

26. ' determining how many cities are there?
27. Size = Range(Range("a1").Offset(1, 0), Range("a1").Offset(1, 0).End(xlDown)).Rows.Count
28. 'now that we know the number of cities, redimension distance array
29. ReDim distance(1 To Size, 1 To Size)
30. 'take the coordinates as a range
31. Set coords = Range(Range("a2"), Range("a2").End(xlDown)).Resize(, 3)

34. 'control not any two nodes are the same node
35. For i = 1 To Size
36. For j = i + 1 To Size
37. If Range("b1").Offset(i).Value = Range("b1").Offset(j).Value Then
38. If Range("c1").Offset(i).Value = Range("c1").Offset(j).Value Then
39. MsgBox "You have entered a node more than once"
40. Exit Sub
41. End If
42. End If
43. Next j
44. Next i
46. 'put in the first arm of the matrix
47. Range("H3") = "City"
48. Range("H3").Font.Bold = True
49. Range("H1") = "Distance Matrix"
50. Range("H1").Font.Bold = True
51. With Range("H3")
52. For i = 1 To Size
53. .Offset(i, 0) = i
54. .Offset(i, 0).Font.Bold = True
55. Next
56. 'second arm of the matrix
57. For j = 1 To Size
58. .Offset(0, j) = j
59. .Offset(0, j).Font.Bold = True
60. Next
61. 'fill it in with distances
62. For i = 1 To Size
63. For j = 1 To Size
64. 'the default value is 0
65. If i = j Then
66. Range("H3").Offset(i, j) = 0
67. 'otherwise look for euclidean distance
68. Else
69. 'search for the coordinates for each value
70. x1 = WorksheetFunction.VLookup(i, coords, 2, False) 'x of i
71. y1 = WorksheetFunction.VLookup(i, coords, 3, False) 'y of i
72. x2 = WorksheetFunction.VLookup(j, coords, 2, False) 'x of j
73. y2 = WorksheetFunction.VLookup(j, coords, 3, False) 'y of j
74. temp\_dist = Sqr(((x1 - x2) ^ 2) + ((y1 - y2) ^ 2))
75. distance(i, j) = temp\_dist                  'reading the distance
76. Range("H3").Offset(i, j) = temp\_dist
77. End If
78. Next
79. Next
80. End With
81. 'Array where route will be stored. Starts and ends in City 1
82. ReDim Route(1 To Size + 1)
83. Route(1) = 1
84. Route(Size + 1) = Route(1)
85. 'Boolean array indicating whether each city was already visited or not. Initialize all cities (except City 1) to False
86. ReDim traveled(1 To Size)
87. traveled(1) = True
88. For i = 2 To Size
89. traveled(i) = False
90. Next
91. 'Total distance traveled is initially 0. Initial current city is City 1
92. totalDistance = 0
93. StartPoint = 1



98. For MatrixYcounter = 2 To Size
99. 'initialize maxDistance to 0
100. minDistance = 9999999
101. For i = 1 To Size
102. If i <> StartPoint And Not traveled(i) Then
103. If distance(StartPoint, i) < minDistance Then
104. EndPoint = i
105. minDistance = Range("H3").Offset(StartPoint, i)
106. End If
107. End If
108. Next i
109. 'store the next city to be visited in the route array
110. Route(MatrixYcounter) = EndPoint
111. traveled(EndPoint) = True
112. 'update total distance travelled
113. totalDistance = totalDistance + minDistance
114. 'update current city
115. StartPoint = EndPoint
116. Next MatrixYcounter
117. 'Update total distance traveled with the distance between the last city visited and the initial city, City 1.
118. totalDistance = totalDistance + distance(StartPoint, 1)
119. 'Print Results
120. With Range("A2").Offset(Size + 5, 0)
121. .Offset(0, 0).Value = "Nearest neighbor route"
122. .Offset(1, 0).Value = "Stop #"
123. .Offset(1, 1).Value = "City"
124. For MatrixYcounter = 1 To Size + 1
125. .Offset(MatrixYcounter + 1, 0).Value = MatrixYcounter
126. .Offset(MatrixYcounter + 1, 1).Value = Route(MatrixYcounter)
127. Next MatrixYcounter
128. .Offset(Size + 4, 0).Value = "Total distance is " & totalDistance
130. For i = 1 To Size + 1
132. .Offset(Size + 5 + i, 0).Value = Application.WorksheetFunction.VLookup(Route(i), coords, 2, True)
133. .Offset(Size + 5 + i, 1).Value = Application.WorksheetFunction.VLookup(Route(i), coords, 3, True)
135. Next i
137. .Offset(Size + 6).Select
138. Range(Selection, Selection.End(xlToRight)).Select
139. Range(Selection, Selection.End(xlDown)).Select
140. ActiveSheet.Shapes.AddChart2(240, xlXYScatterLines).Select
142. ActiveChart.ChartTitle.Text = "Initial basic feasible route by Nearest Neighbour"
144. End With

After we found a basic initial feasible solution. We improved our result with the 2-opt algorithm. ‘opt algorithm was more challenging than we were expecting. Partly by we couldn’t remember the structure of the algorithm that we saw from the lecture but it was mostly about the lack of source on the internet. After several failed or semi-success attempts, we decided to meet and break the algorithm part by part to internalize what’s going on how can we convert it to a code section. We struggled until we realized the in between the focus points should be reversed. From that point on we could move towards a successful code bit by bit by breaking what we are doing, what is ought to be and what is happening by debugging. But in the end, we were successful and the algorithm goes like this:

1. Create a comparison array which will be the indicator of the difference

2. Construct a for loop with complexity n^2

3. Set New route

4. Find new routes distance

5. Compare it to the old route if smaller set old route with new route if not go to step 3

1. ReDim Newroute(1 To Size + 1)
3. Iterationcounter = 0
5. Iterationflag = False

8. For i = 1 To Size
9. Newroute(i) = Route(i)
10. Next i
12. repatfromthestart:
13. Iterationflag = False
14. For i = 1 To Size
15. For j = i + 1 To Size
16. Newroute(i) = Route(j)
17. Newroute(j) = Route(i)
18. Newroute(Size + 1) = Newroute(1)
19. For k = 1 To (j - i)
20. Newroute(i + k) = Route(j - k)
21. Next k
22. For k = 1 To Size
23. Newroutedistance = Newroutedistance + distance(Newroute(k), Newroute(k + 1))
24. Next k
26. If Newroutedistance < totalDistance Then
27. For k = 1 To Size + 1
28. Route(k) = Newroute(k)
29. Next k

32. totalDistance = Newroutedistance
33. Iterationcounter = Iterationcounter + 1
34. Iterationflag = True
35. With Range("A2").Offset(Size + 5, 3 \* Iterationcounter)
36. .Offset(0, 0).Value = "Nearest neighbor route"
37. .Offset(1, 0).Value = "Stop #"
38. .Offset(1, 1).Value = "City"
39. For MatrixYcounter = 1 To Size + 1
40. .Offset(MatrixYcounter + 1, 0).Value = MatrixYcounter
41. .Offset(MatrixYcounter + 1, 1).Value = Route(MatrixYcounter)
42. Next MatrixYcounter
43. .Offset(Size + 4, 0).Value = "Total distance is " & totalDistance
44. .Offset(Size + 5, 0).Value = "Iterationcounter = " & Iterationcounter
46. For l = 1 To Size + 1
48. .Offset(Size + 5 + l, 0).Value = Application.WorksheetFunction.VLookup(Route(l), coords, 2, True)
49. .Offset(Size + 5 + l, 1).Value = Application.WorksheetFunction.VLookup(Route(l), coords, 3, True)
51. Next l
53. .Offset(Size + 6).Select
54. Range(Selection, Selection.End(xlToRight)).Select
55. Range(Selection, Selection.End(xlDown)).Select
56. ActiveSheet.Shapes.AddChart2(240, xlXYScatterLines).Select
57. ActiveChart.ChartTitle.Text = "Iteration" & Iterationcounter
58. End With
60. End If
61. Newroutedistance = 0
62. If Iterationflag Then GoTo repatfromthestart
63. Next j
65. For k = 1 To Size + 1
66. Newroute(k) = Route(k)
67. Next k
68. Next i

Upon the second heuristic, even though we struggled a lot we were not able to figure if the new route creates sub-circles out. But we create a code that can separate if a node has more than one entering and exiting. For that, first, we create an array with 2 helper array, which helper arrays indicate from which node to which note the edge is, and hold their distances. We extract from the array loopy ones and the reciprocal ones and obtained a better manageable smaller array. To prevent more than two outgoing and incoming we look at the helper function if it is used more than twice or not. Because we do not have the complete code we are not adding to this paper but it will be in our Excel file since we believe it was near complete.

For the Bonus part, we used a lot of record macro. We tried to construct a model and use it with the solver. Although we could construct a model with 8 nodes or less, we couldn’t understand how the solver works with VBA. So, we stuck at the optimization part. But If optimize button is clicked we create a sheet that can be easily optimized with just opening the solver and pressing solve.

The user interface was the easiest and the most straightforward part since it doesn’t require much skill from the developer.

**Conclusion**

Our project was mostly hard earned. But we enjoyed the experience since it pushed us to our limits and research beyond our capabilities. Our discussion was mostly about how the VBA is so huge and there are lots of ways to do the something. One can easily forget or not remember a way to do a thing much more easily. So, knowing the steps what you want to build and researching during coding is a vital part of the experience.