12/14/2017

Large Scale Optimization

Vehicle Routing Problem

GIANNATOU EVA

Contents

Component 1

In the first component we need to prepare a java application for dealing with the Vehicle Routing Problem. This representation should include the depot, the customers, the set of routes, the solution and the distance matrix.

I created three new classes called Node, Route and Solution. The Node class represents one customer and contains three integer attributes and one boolean. These attributes contain the coordinates (long, lat) of the customer's position, the customer ID, his demand and a boolean attribute called "isRouted" which describes whether this customer has been added into the solution or not (isRouted = "True" if yes). The three numeric attributes were initialized to zero and the boolean attribute 'isRouted' was initialized to 'false'. The Route class contains the customers of each route, the route load and the route cost. This class contains two double attributes with the route cost and the route load and one arraylist with Node objects. This arraylist consists of the customers which belong to each route, their coordinates, ID and demand. All the numeric attributes were initialized to zero and a new empty arraylist of type Node was also initialized. Finally, the Solution class stores all the routes, the solution's total cost and total load. It contains two double attributes which store the total cost and the total load of the solution as well as an arraylist of Route objects containing all the routes. All the numeric attributes were initialized to zero and a new empty arraylist of type Route was also initialized.

In the main method of the Java program I have created objects for each one of the three classes mentioned above. The first Node object was named 'depot' and contained the coordinates and the demand of the depot, in this case depot.x=50, depot.y=50 and depot.demand=0. The second Node object was called 'customers' and it was used to store all the customers, their coordinates, id and demand. A random generator was used in order to randomly fill in all the customer's information.
The second Node object was named 'allNodes' and it was used to build the corresponding distance matrix which contained the distance between all customers and between the customers and the depot. The distance matrix is a 2-D array which holds the distances between node pairs. The [i][j] element of this array is the distance required for moving from the i-th node of allNodes (node with id : i) to the j-th node of allNodes list (node with id : j). Finally one Solution object called 's' was created and one Route object called 'route'. As it was mentioned earlier the Route object contains an arraylist of Node objects and the Solution object contains an arraylist of Route objects.

Component 2

In the main method of the Java program I have created one Integer attribute for the number of customers, one Integer attribute for the vehicle capacity, one for the total available number of vehicles, the vehicle number of the vehicles which are currently used, one attribute for the total solution load and one for the total solution cost. The numberOfCustomers=30, the vehicleCapacity=50, the totalVehicles=10, the vehicleNo=0, the total solution cost and the total solution load were initialized to zero.

Component 3

For this component we need to develop a constructive heuristic based on Nearest Neighbor to produce an initial solution for the VRP. The code used for this component is based on the nearest neighbor heuristic for the TSP which was taught during the lesson's labs. I repeated the TSP solution as many times as the number of vehicles needed in order to satisfy all customers. The TSP algorithm results to only one route while the VRP solution results to an arraylist of mulitple routes. Every route is being performed by only one vehicle and it begins and ends with a depot. Of course, in every iterations we have to check the capacity constraints because the load that each vehicle can carry is limited. As far as the VRP implemenation is concerned I created two new attributes the first attribute was called cntCust and it was used for counting the number of customers which have been inserted into the solution and a new boolean attribute called 'complete' which was initialized to 'false'. Complete will be true when the greedy VRP solution finishes and when all inter and intra relocations are completed. A new Node object called nodeSequence was created which is the arraylist of nodes contained in the route (s.rt). A new Route object called 'route' was also created and it will be the one of the routes contained in s. The rest of the main method will be inside a while loop (while(!complete)) and this loop will continue running until all customers have been inserted into the solution. In the first step of every iteration the program identifies the closest neighbor to the last customer that also respects the capacity. However, when the current vehicle is full but there are still customers left to be served we move to the top of the while loop and we begin the construction of the second route which will be performed by the second vehicle. Depots are added in the beginning and in the end of every route. At the end of each iteration we add the current route into the solution. At the end of every route we add the current route load to the total solution load and the route cost to the total solution cost.

Component 4

In this component we need to design a local search method for improving the initial solution generated in Component 3. This local search method will consider intra-route relocations. This means that at each iteration, the method should explore all potential relocations of the customers within their routes. The relocation yielding the best solution cost improvement should be selected to be applied to the candidate solution. As it mentioned earlier the code is placed inside a while loop which will end when all customers have been inserted into the solution and when all intra and inter relocations are performed. This while route is repeated as many times as the number of vehicles used in the solution. For example in the first iteration of the algorithm the first route has been created and added into the solution. Before moving to the next vehicle, we examine whether the current route cost can be reduced by applying intra route relocation. In order to do so we use the findBestRelocationMove and applyRelocationMoves methods which were discussed during the lab sessions of this course. All cost reducing relocations are performed, the current route cost and the solution cost is updated and then we move to the next route and therefore the next vehicle. For the next route I find the initial VRP solution and then I apply all cost reducing relocation before moving again to the next vehicle. This procedure will be repeated several times, until all customers are inserted into the solution.

Component 5

In this component we need to design a local search method for improving the initial solution generated in Component 3. This local search method will consider all possible customer relocations (both intra- and inter-route). In component 4, we performed all cost reducing intra route relocations and we have added the improved routes into the solution. For the needs of this component I concatenated all the routes into one long route. I then applied the same procedure as I did in the fourth component. I applied intra route relocations in this long route which contained the paths of all the vehicles which passed through all the customers. This procedure was performed with respect to the constraints. For example the depots were never getting relocated. Finally, the relocation was only performed if the vehicle to which the node was about to relocate could satisfy this customer's demand.

Results

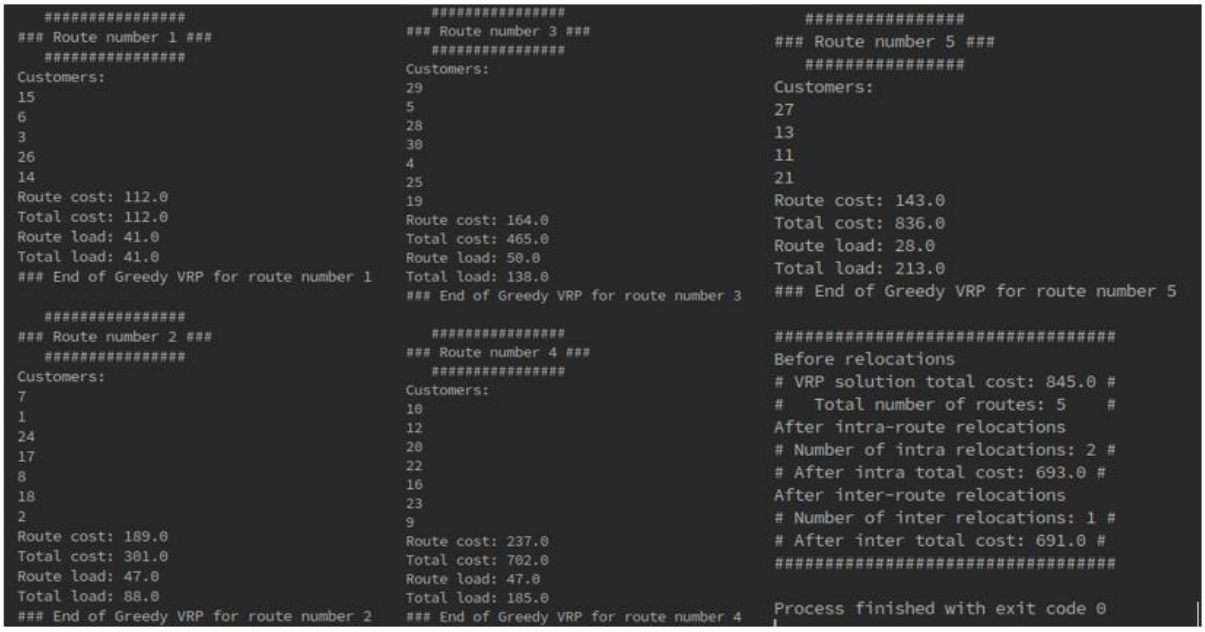

Figure 1 VRP solution output

Figure 1 show the output of the Java program which summarizes the results. As we can see in the initial solution there was a total cost=845. The first route was the 0-15-6-3-26-14-0 which had route cost=112, route load=41 and at this point the total cost and load of the solutions equals to the cost and load of the first route. The second route was the 0-7-1-24-17-8-18-2-0 with route cost=189, route load=47, total solution cost=301 and total solution load=88. The third route was the 0-29-5-28-30-4-25-19-0 with route cost=164, route load=50, total solution cost=465 and total solution load=138. The fourth route was the 0-10-12-20-22-16-23-9-0 with route cost=237, route load=47, total solution cost=702 and total solution load=185. The final route was the 0-27-11-13-21-0 with route cost=143, route load=28, total solution cost=836 and total solution load=213. The total solution cost for the initial solution with the 5 routes equals to 845. The total cost of the solution after the intra route relocations dropped to 693. Finally, the total solution cost after the inter route relocations was equal to 691.

In the next pages we can see the route visualizations before and after the relocations.

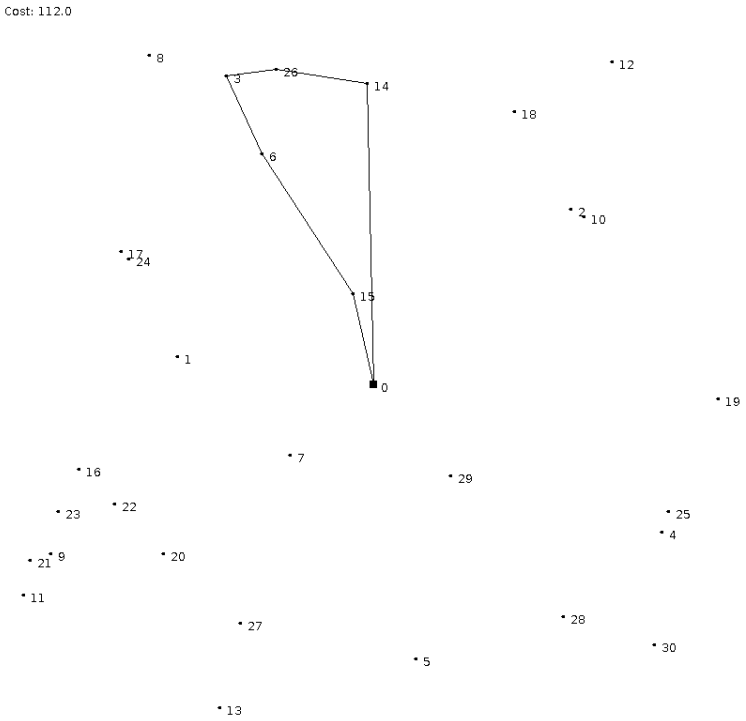

Figure 2 Route 1: before intra relocations

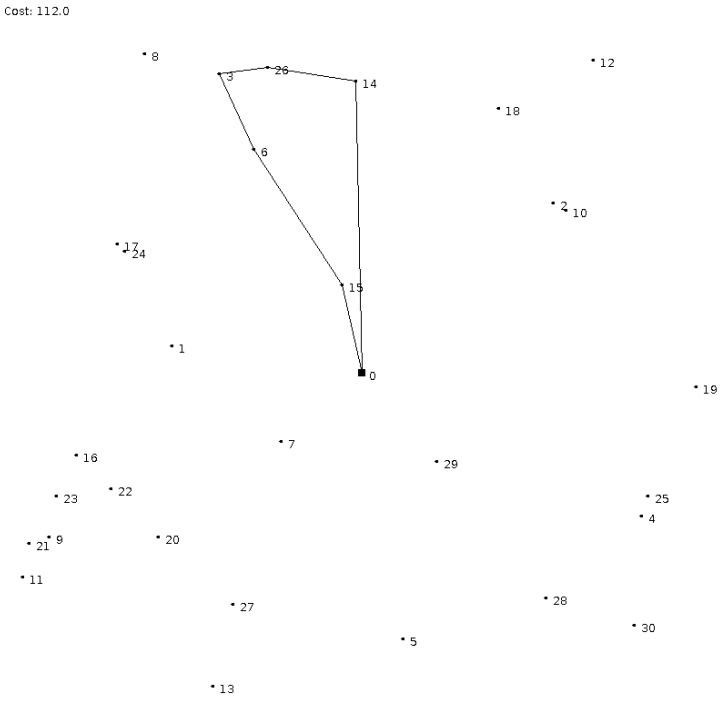

Figure 3 Route 1: after intra relocations (0 relocations)

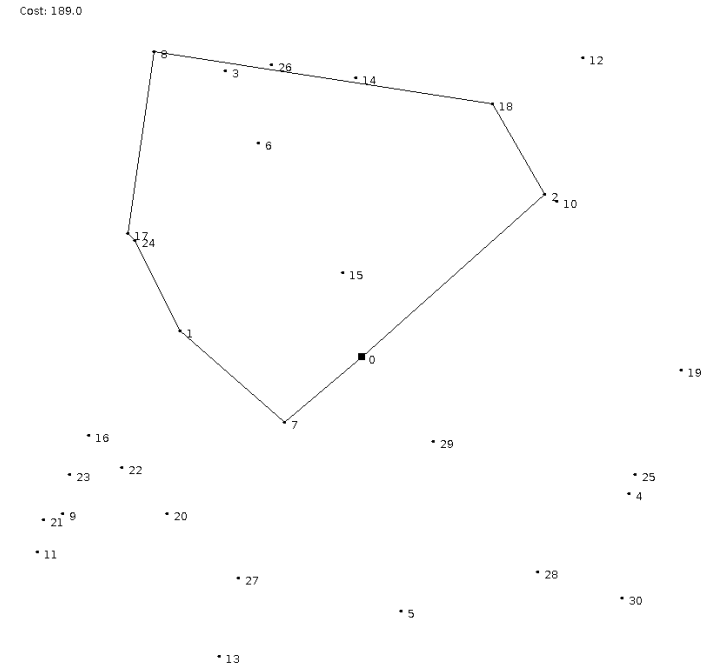

Figure 4 Route 2: before intra relocations

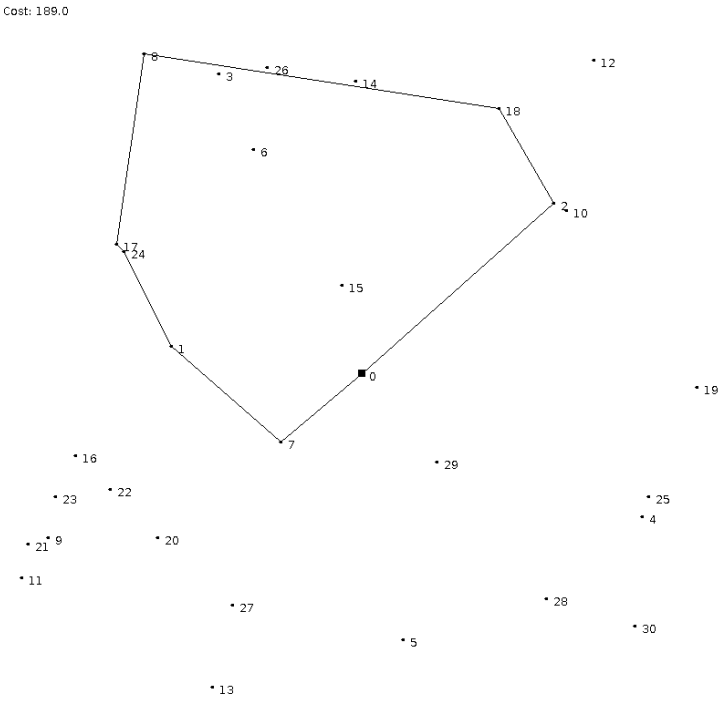

Figure 5 Route 2: after intra relocations (0 relocations)

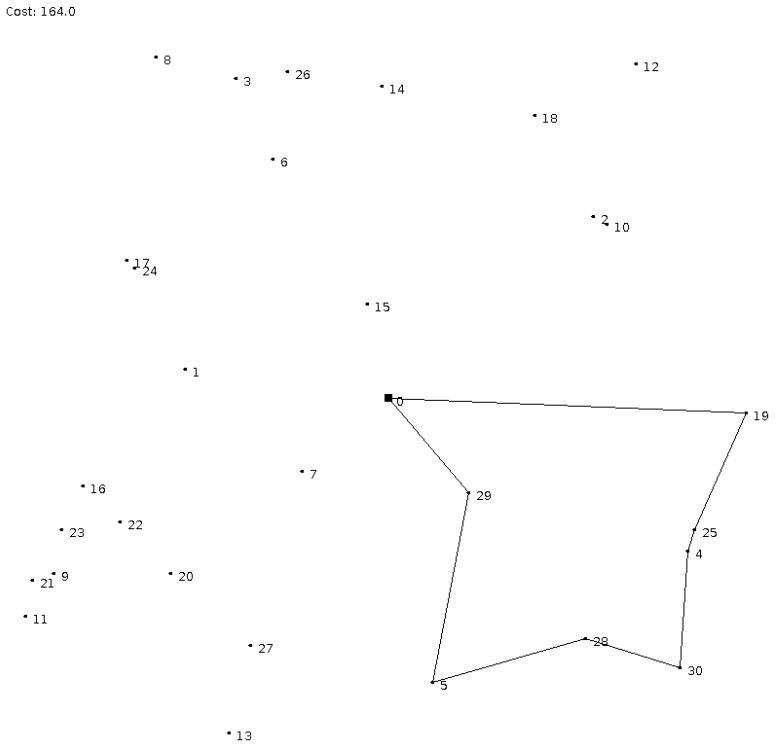

Figure 6 Route 3: before intra relocations

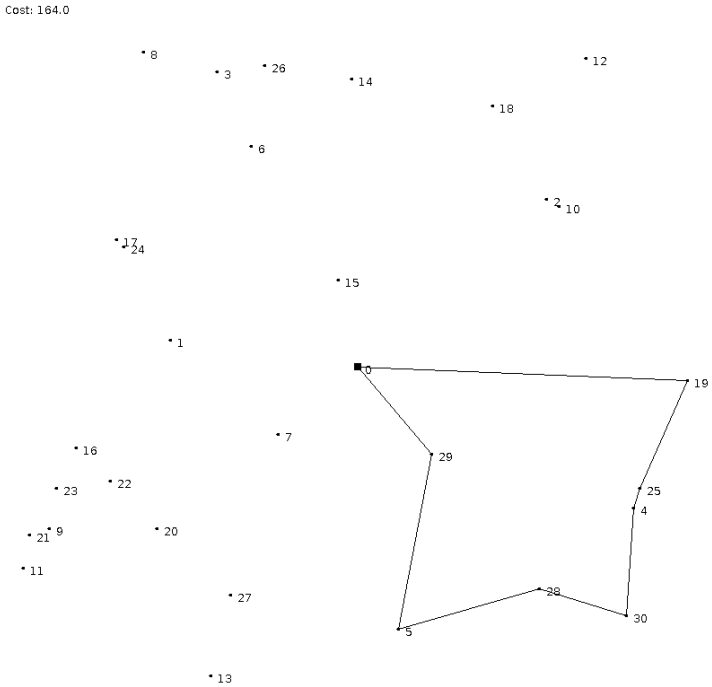

Figure 7 Route 3: after intra relocations (0 relocations)

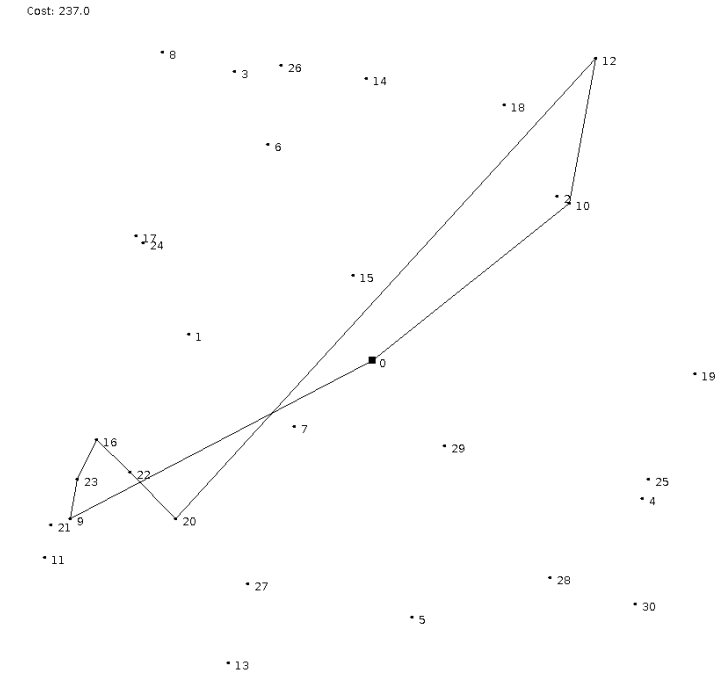

Figure 8 Route 4: before intra relocations

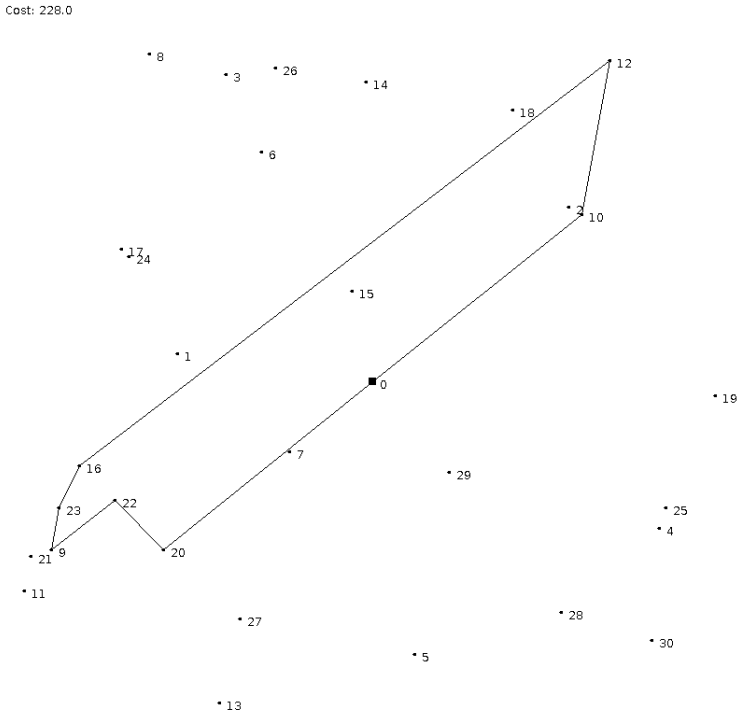

Figure 9 Route 4: after intra relocations (2 relocations)

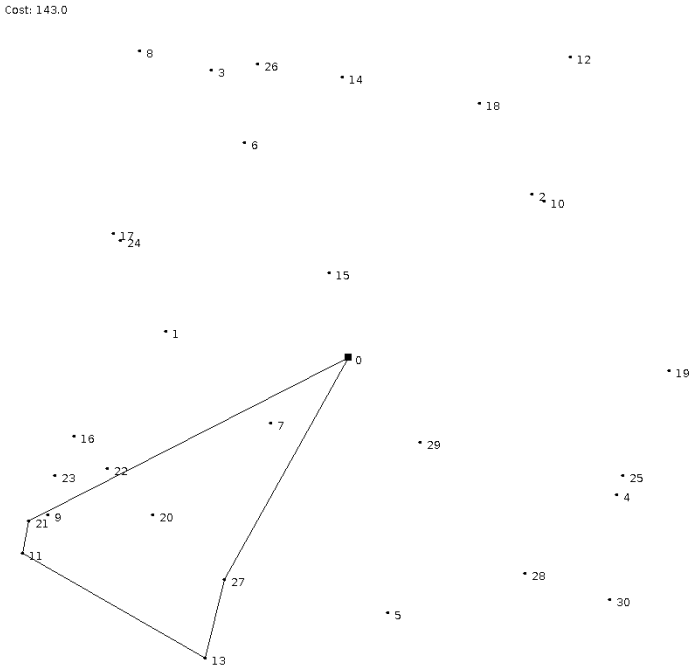

Figure 10 Route 5: before intra relocations

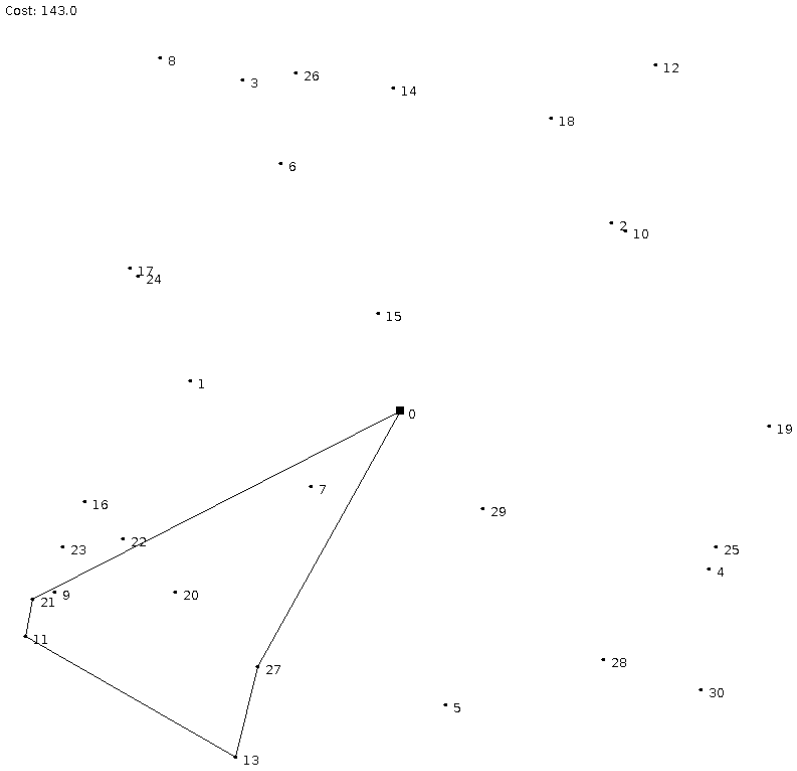

Figure 11 Route 5: after intra relocations (0 relocations)

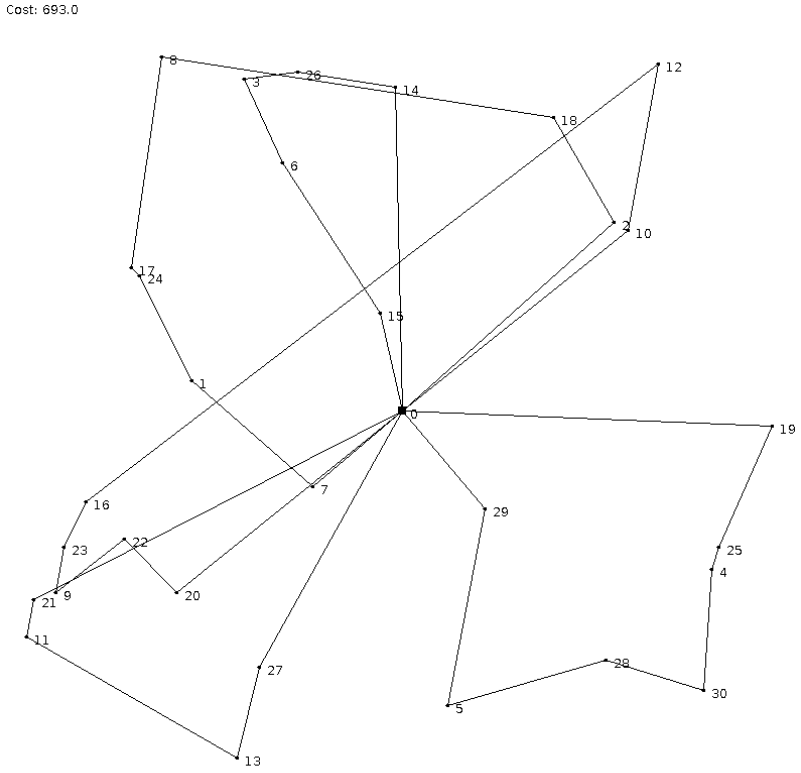

Figure 12 Full solution after 2 intra route relocations

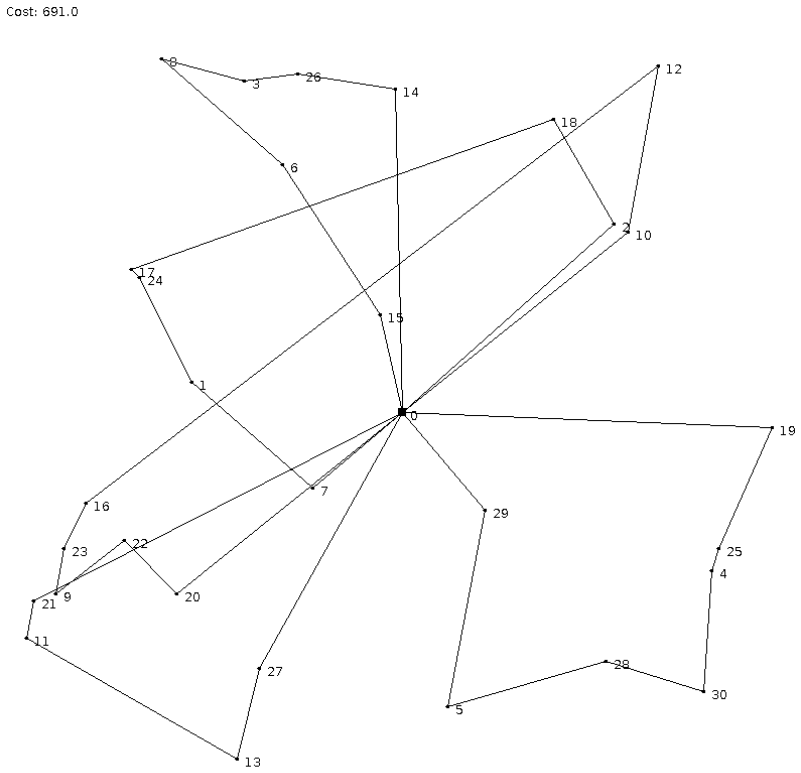

Figure 13 Full solution after 1 inter route relocation