Project description Captivated Vehicle Routing Problem

In the capacitated vehicle routing problem (CVRP), a fleet of delivery vehicles with uniform capacity must service customers with known demand for a single commodity. I included a file named A-n39-k5.vrp. This is from the A-set of problems that I would like to be able to solve. In this file you see 39 nodes/cities, with corresponding demand.

- Node 1 represents the depot.
- The distance between cities are calculated by Euclidean distance
- The vehicles start and end their routes at a common depot.
- Capacity of vehicles is given (=100)
- Demand per client is given
- Each customer can only be served by one vehicle.
- The objectives are to minimize the fleet size and assign a sequence of customers to each truck of the fleet minimizing the total distance travelled such that all customers are served and the total demand served by each truck does not exceed its capacity.

I would like this to be solved in 2 phases: initial solution and improved solution.

1. **Initial solution**

I would like to have the initial solution to be a greedy algorithm with nearest neighbour search. So we start with 1 vehicle and make a route based on its nearest neighbour until the capacity limit is reached, then we go back to depot. Then we start a new vehicle and go to the nearest neighbour, and then again the nearest neighbour etc. until his capacity is reached. We repeat this until all cities are visited once. Then we display the routes of the vehicles that have been used and the total costs/distance.

My knowledge on coding is limited, but I tried to make the pseudo code as good as possible:

1. Initialization: Start with a partial tour from the depot
2. Selection: Let (1, ... , k) be the current partial tour (k<n). Find city k+1 that is not in the current tour and was not in a previous tour, that is closest to k.
3. Test if demand of chosen point k+1 can be satisfied (capacity constraint)
   a. If no: go back to the depot and end the partial tour. Continue with 1 (initialization of a new partial tour, thus a new vehicle).
   b. If yes: continue with 4.
4. Insertion: Insert k+1 at the end of the partial tour.
5. If all cities are inserted then STOP, else go back to 2.
6. Show the routes of the different vehicles, and show the total costs.

2. **Improved solution (using Tabu Search)**

Of course, the initial solution is not optimal. The solution can be improved by swapping edges, however, it might happen that the solution will be stuck in a local optimum. I would like to implement Tabu Search to cope with this.