Algorithm Design - Assignment 2

[redacted]
Wessel Boosman (589273)

December 19, 2021

Question 1

(a) *Develop a greedy constructive heuristic. Provide a clear but concise explanation of the main steps of your algorithm. Implement the greedy values for all instances.*
Our greedy heuristic is made in the following way. We calculate the distances between every node. Then we pick the customer which is closest to the depot. From there we continue and pick the closest customer from that customer. We repeat this procedure until the next customer we would pick has a demand greater than the remaining capacity of the truck, in that case we will return to the depot. We keep track of two lists, one for which customer is served and one for which is not. So after a route is completed, all the customers of that route are added to the list of served customers and deleted from the list of customers which are not served. Thus, the first customer in the next route is the closest one to the depot which is not served yet. When all customers are served we return to the depot and terminate our greedy heuristic. The resulting objective values following this heuristic are summarized in table 1.

Table 1: Table summarizing the objective values of each of the instances using the greedy heuristic as explained above.

Instance	1	2	3	4	5
Objective Value	1547.84	1298.69	1706.56	1432.37	2182.09

(b) *Define at least three neighborhoods for a CVRP solution and determine the size of each neighborhood.*
We define the following three neighborhoods:

- 2-Opt
- 3-Opt
- Randomly destroy 10% of the solution (so just remove 10% of the nodes from the current solution) and then repair them by adding them one by one back in the route which increases the objective value the least.

The size of the 2-Opt neighborhood is $\mathcal{O}(n^2)$ since for every arc in your route there are, at most, *(n-1)* possible other arcs to apply an exchange with. Following the same reasoning, we determine that the 3-Opt neighborhood has size $\mathcal{O}(n^3)$. Therefore, the 2-Opt as well as the 3-Opt are considered small neighborhoods since the size of each neighborhood is less or equal than order cubed in the input size. The destroy and repair method is considered a large neighborhood, because it grows exponentially in the input size. The possible combinations of randomly picking a set of customers to exclude form the solution is obviously exponential in the input size.

(c) *Use the neighborhoods to develop a Variable Neighbourhood Descent (VND) heuristic. Provide a clear but concise explanation of your algorithm. Implement the VND heuristic, where you use the solution from the first part as starting solution. Report the objective values for each instance.*

Our Variable Neighborhood Descent is implemented in the following way. First we determine an initial solution using the greedy heuristic. Then we perform the 2-Opt algorithm on each route to see if we can do any better. If two arcs are exchanged successfully, we start our 2-Opt algorithm on that route again, till the moment we can not make any improvements anymore. From that moment onward we try our 3-opt algorithm and check if we can make an improvement. If so, we return to our 2-opt algorithm and restart the process. At one moment the 2-opt as well as the 3-opt algorithm are unable to make exchanges such that the solution improves. Then we continue to our largest neighborhood, the random destroy and repair method. We select randomly a set of customers from the solution, remove them, and create new routes excluding the determined set of customers. Then, considering one customer at the time, we check in which route and at which position the customer can be inserted best. That means, at which position is the increased objective value as small as possible. We do this until all *'destroyed'* customers are added back in the solution again. If we make an improvement using this method we go back to the 2-Opt algorithm. In the case we don't make an improvement we try the random and destroy method, at most, three more times. This is done because the 'destroy and repair' method is actually a large neighborhood search method in which a random component of the solution is destroyed. Intuitively it is easy to understand that not all destroyed parts are resulting in an improvement. Therefore we determined that we try the repair and destroy method multiple times to check if we were just 'unlucky'. And if one of those tries results in a better objective value we return to the 2-Opt method and start all over. If not, we terminate the heuristic. Of course, trying the destroy and repair method at most four times is a design choice, just as the choice of destroying 10%. The latter was determined empirically, destroying 10% gave us, in general, a relative good improvement of the objective value. The choice of trying the largest neighborhood four times is based on multiple reasons. First of all, the objective value will probably improve (it won't decrease at least) if you try the destroy and repair method even more times, but so will the running time. If we allow the VND to perform a large neighborhood search very

often, then it is not really a 'Variable Neighborhood Descent' anymore, but more just a large neighborhood search. Secondly, when implementing the VND in the the GRASP heuristic, there is a trade-off between investigating a certain solution and investigating the solution space when there is a limited amount of running time. This will be explained in more detail in the next question. The objective values resulting from the VND on the initial greedy heuristic solution are provided in table 2.

Table 2: Table summarizing the objective values after VND is performed on the solution constructed by the greedy heuristic

Instance	1	2	3	4	5
Objective Value	883.25	747.00	949.27	934.19	1239.22

(d) *Develop a GRASP heuristic, where you use the VND as the local search component. Again provide a clear but concise explanation of the main steps.*

Our GRASP heuristic is in essence the Variable Neighborhood Descent heuristic, but starting from different area's of the solution space each time. This is done in order to escape from possible local minima. In previous questions the initial solution was based on the greedy heuristic, now we adjust this algorithm a bit such that each initial solution we construct is based on random components. We do this in the following way, instead of picking the customer closest to the depot as in the greedy heuristic, we pick a random customer from the set of customers which is not yet served as our first customer in the route. From there we consider the 10% of the input size closest customers from the set of customers which has not been served yet and pick randomly one of those customers[1]. That will be our next customer. We repeat this procedure until the next customer we pick has a demand which exceeds the remaining capacity. In that we do go back to the depot and start a new route. Again we do this until all customers are served. Our GRASP algorithm starts with a random initial solution constructed using the approach just explained and therefore starts from a different area of the solution space each iteration. Starting from that solution, we apply the VND heuristic as explained in the previous part. Now it should also be clear that there is a trade-off between investigating a certain solution and the solution space. Starting from a random solution, running time can be used to improve that solution by trying, for example, the destroy and repair method more often. But this will result in additional running which can't be used for checking different areas of the solution space (equivalent of doing more iterations of the GRASP heuristic). After this heuristic terminates, we store the objective value found with the corresponding routes. Then we start all over with a new randomly constructed initial solution, apply the VND heuristic, check whether the new solution is better then the previous one and if so, store the new solution. Then we start again with a new randomly constructed

[1]Unless, there are less customers than 10% of the input size left to serve. In that case we just pick a random customer from the remaining customers.

initial solution, etc. We perform this procedure for ten minutes, and then terminate. The final optimal objective values are summarized in table 3. The optimal routes are provided in a separate file.

Table 3: Table summarizing the objective values of the GRASP heuristic.

Instance	1	2	3	4	5
Objective Value	710.96	646.12	862.58	793.03	1136.78

(e) *Compare the developed algorithms and report on your findings.*

Using our VND heuristic on the solution constructed by the greedy heuristic gives a significant improvement as can be seen from the comparison between table 1 and 2. Note that the solution improvement by the VND would be different every time you run the algorithm because in the largest neighborhood a random part of the solution is destroyed and repaired again. Performing the VND on (partly) randomly constructed initial solutions as in the GRASP heuristic results in even better objective values as can be seen in table 3. Therefore, we conclude that it actually pays off to investigate different areas of the solution space in order to find the optimal objective value. A possible explanation for this behaviour could be that performing our VND on the solution corresponding to the greedy heuristic resulted in a local minimum. In contrast, the GRASP heuristic was possibly able to escape this local minimum, resulting in better objective values.