Report assignment 2 - MDVRP

Chromosome representation:

The chromosome is represented as an ArrayList with a linkedlist of customer number strings for each depot. The routes for each depot are stored in a separate list where each linkedList represents a depot. The integer values, r, mark the end of a route given by how many of the total customers have been traversed (last entry will be length of customer list). In addition, the current load of each depot is stored and the total depot cost for each depot is stored in separate lists. The cost value is the total value of the chromosome (sum of all depot costs). The information on each customer and depot is centralized, and the chromosomes reference the central information database when needed.

```
ArrayList<LinkedList<String>> depots;
ArrayList<LinkedList<Integer>> depotRoutePartitions;
ArrayList<Double> currentLoad;
ArrayList<Double> depotCosts;
double cost;
```

Figure 1: Chromosome representation

The code uses two different crossover operations. The first one is a simple order1 crossover where for each depot in the chromosome two parents create the child depot. Customers moved away to another depot in parent 1 are disregarded for crossover at that depot (they are removed from parent 2). After the order1 stage there is a possibility that ¼ of randomly chosen customers will be inserted at their best place in the current depot. This however was not a strong enough crossover which is why the second crossover was implemented.
After some research I found the BCRC by Ombuki, B., Ross, B., Hanshar, F. and decided to implement a version of it. It works as shown in figure 2. In my version the customers present in only p_2 are disregarded. Additionally the code only stores the best feasible and the best unfeasible (defined by breaking a route in two) instead of keeping a list of all the possible entries. The use of multiple crossovers keeps a better diversity in the population.

1. Randomly select $p_1, p_2, \in P$.
2. Randomly select a depot $\Phi \in V_d$ to undergo reproduction
3. Randomly select a route from each parent $r_1 \in p_{1_\Phi}, r_2 \in p_{2_\Phi}$
4. a) Remove all customers $c \in r_1$ from p_2
   b) Remove all customers $c \in r_2$ from p_1
5. For each $c \in r_1$
   - Compute the insertion cost of $c_l \in r_1$ into each location in p_{2_Φ} and store these in an ordered list. Also for each insertion location, store in this list whether the insertion maintained feasibility or infeasibility. (If insertion broke an existing route into two routes, this would be infeasible.) location in p_{2_Φ}.
   - Generate a random number $k \in (0, 1)$.
   - If $k \leq 0.8$, choose the first feasible insertion location. (If there exists no feasible insertion locations, create new route with c_l as the only customer.)
     Else if $k > 0.8$, choose the first entry in the list, regardless of feasibility.
6. Repeat for r_2 and p_{1_Φ} taking place of r_1 and p_{2_Φ} respectively in the above loop (from step 5)

Figure 2: The BCRC algorithm

There are five different mutation operators. **Reverse:** Select a random depot and then selects a random length of customers and reverses the customer list. Update depot values and partition indices. **Switch:** Select a random depot and select two random customers and swap their places. **Biasedswitch:** Random depot and select a random customer. Calculate the total depot cost for all insertion places of customer. Insert customer at the place which gives the lowest total score. **Random inter depot:** Select a random depot. Select a

customer from the closeToSeveral list (customers who are close to more than one depot can be switched). Retrieve the chosen customer's depot list (the depots it is close to) and try to insert customer to random of these depots. If the operator fails (due to load constraints) a given amount of times it gives up. **Biased inter depot:** The idea is to pick customers from depots with the most slack. The goal is to maximize each vehicle load. So the depot with the most slack will be the one where the last vehicle (given all previous vehicles are fully loaded) has the least amount of load. The assumption that the optimal solution is the one where all vehicles are fully loaded might be faulty. So the use of this operator is somewhat restricted. Often the simple random operator is preferred.

The first parent is selected by a binary tournament. The selection mechanism has an adjustable bias towards better solutions when picking the first parent. The second parent is randomly chosen from the population and competes with another parent in a tournament. To maintain diversity, the winner is the one who has the biggest discrepancy depot-wise with the first parent. The idea is to keep the "pareto-front" as wide as possible (considering each depot as a single objective). However, this picking routine has a flaw. The second parent will more often than not be from the lower half of the population. To account for this, the offspring based mainly on the second parent must compete with its parent, and only one of them is chosen. Both the offspring and parent from parent one are inserted to the main population.

2. Values which give best result: population size = 400, generation number 6000, crossover rate on average about 0.5 (1.5/3), and mutation rate = 0.4

3. The only constraints which may be violated by the crossovers is the max amount of vehicles. This is handled by giving a penalty for each route in a chromosome. Hopefully, the algorithm will improve on this constraint and fulfill the requirements by itself.

4. The fitness function is the total cost (distance traveled) by all vehicles plus a static penalty for each route.

5. Elitism is implemented. The next generation children are first added to the main population and sorted before the weakest individuals are removed. This way the strongest individuals will always have precedence over weaker individuals.

```
1371.543289325507
1 1   170.71    54.00   0 8 16 24 32 40 39 31 23 15 7 0
1 2   170.71    54.00   0 6 14 22 30 38 36 28 20 12 4 0
1 3   170.71    54.00   0 5 13 21 29 37 35 27 19 11 3 0
1 4   170.71    54.00   0 1 9 17 25 33 34 26 18 10 2 0
2 1   170.71    54.00   0 47 55 63 71 79 78 70 62 54 46 0
2 2   176.57    54.00   0 45 48 56 64 72 80 77 69 61 53 0
2 3   170.71    54.00   0 41 49 57 65 73 76 68 60 52 44 0
2 4   170.71    54.00   0 43 51 59 67 75 74 66 58 50 42 0
```

Figure 4: Solution for test problem 13 in the default data format

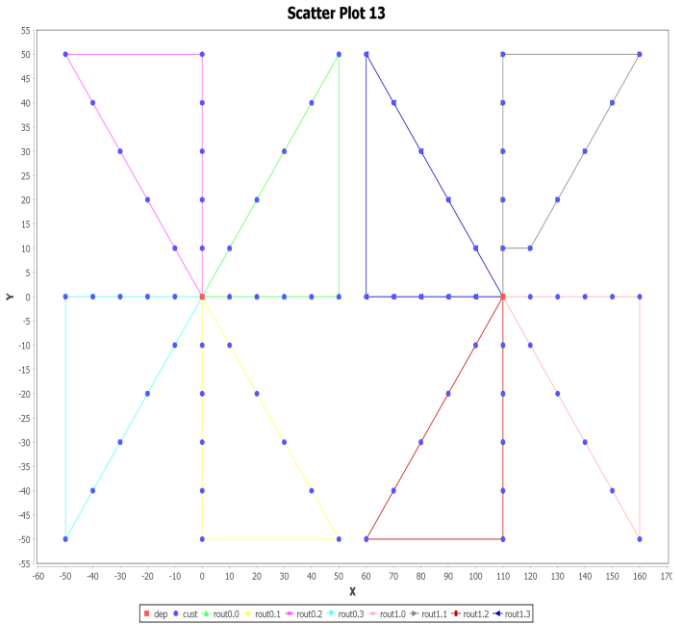

Figure 3: Graphical presentation of a solution for test problem 13