Heuristic Optimization Methods
Capacitated Vehicle Routing Problem with Time Windows

Čubek Matej
Paradžik Berislav

January 9, 2023

Contents

1 Problem description

The problem described in the text is the capacitated vehicle routing with time windows (CVRPTW) problem, which is a variant of the vehicle routing problem (VRP). It involves finding a set of routes for a fleet of vehicles serving customers from a depot while considering capacity and time window constraints. In this problem, the vehicles have a limited capacity, and customers must be served within their specified time windows and the working hours of the depot. The problem instances include information about the number of vehicles, the capacity of each vehicle, and data about each customer such as their coordinates, resource demands, schedule horizon, and the duration of the service time.

The problem defines the following constrains:

1. Each customer is served by exactly one vehicle/route, with the resource amounts.
2. The demand on each route must not exceed the capacity of the vehicle.
3. The vehicle servicing a certain customer must arrive at the customer location within the interval given for that customer. The duration of the service can exceed the interval.
4. Each vehicle starts and finishes its route in node 0 (customer 0 location; depot), within the time interval given for customer 0.

2 Description of the implemented heuristic algorithm

Greedy algorithm is used for construction phase, for local search is used Large Neighbourhood Search. Large Neighbourhood Search randomly removes n customers from solution and then puts them back in random routes to optimal positions. New solution is not accepted if it is worse or does not fulfil all constraints(except maximum number of routes).

Solution representation	List of routes, each route contains list of customers in exact order they are served. Each customer is served by exactly one vehicle/route.
Objective function	min(numberOfRoutes, totalDistance), ∞ if constraints are not meet. For example, when comparing 2 solutions solution1 and solution2, better solution is one that has lower number of routes, if both solution have same number of routes, better solution is the one which has lower total distance.
Fitness function(score)	$min(numberOfRoutes^{alpha} + getAllRoutesTime^{beta}), alpha = 3, beta = 1$
Construction of an initial solution	Greedy Algorithm
Termination criterion	Time constrain, in this project, time constrains were 1 minute, 5 minute and 6 hours.

3 Pseudocode of the implemented algorithm

Algorithm 1 Greedy Algorithm - construction phase

```
leftCustomers ← loadAllCustomers()
routes ← newList()
depot ← leftCustomers.removeFirst()
while notleftCustomers.isEmpty() do
  route ← newRoute(vehicleInstance.capacity())
  lastCustomer = depot
  route.addCustomerToRouteEnd(depot)
  while notleftCustomers.isEmpty() do
    lastCustomerFinal ← lastCustomer
    potentialNextStop = leftCustomers.stream().filter(satisfyallrouteconstraints).findClosest()

    if potentialNextStopisNone then
      break
    end if
    lastCustomer ← potentialNextStop
    route.addCustomerToRouteEnd(lastCustomer)
    leftCustomers.remove(lastCustomer)
  end while
  route.addCustomerToRouteEnd(depot)
  routes.add(route)
end while
return routes
```

Algorithm 2 Neighbourhood Iterator - function next

Input $solution, minCustomersToRemove, maxCustomersToRemove$

```
random ← newRandom()
numberOfCustomersToRemove = random.nextInt(minCustomersToRemove, maxCustomersToRemove+
1)
customersToRemove = pickRandomCustomersToRemove(numberOfCustomersToRemove)
routesCopy = copyRoutes()
routesWithRemovedCustomers = removeCustomersFromRoute(routesCopy, customersToRemove)

newRoutes = addCustomersToRandomRouteInOptimalList(routesWithRemovedCustomers, customersToRemove)

return newRoutes
```

Algorithm 3 Large Neighbourhood Search Algorithm

```
timer ← loadTimer()
neighbourhoodIteratorCreator ← newloadNeighbourhoodIteratorCreator()
incumbent ← GreedyAlgorithm()
incumbentScore ← newList()
while timer.isActive() do
  neighbourhoodIterator = neighbourhoodIteratorCreator.apply(incumbent, timer)
  while neighbourhoodIterator.hasNext() do
    neighbour = neighbourhoodIterator.next(solution, minCustomersToRemove, maxCustomersToRemove)

    neighbourScore = objectiveFunction.score(neighbour)
    if neighbourScore < incumbentScore then
      incumbent ← neighbour
      incumbentScore ← neighbourScore
      break
    end if
  end while
end while
return incumbent
```

4 Results

In this section are presented results obtained after 1 minute of algorithm execution, 5 minutes of algorithm execution and 6 hours of algorithm execution for each instance. For each instance and time constraint there is a graph how number of vehicles(routes) changed over iterations.

Best results for each instances

Instance	Max N vehicles	N Customers	Capacity	N vehicles	Total distance
i1[2]	50	200	200	27	8694.713865
i2[5]	100	400	200	42	12593.663622
i3[4]	150	600	700	33	28114.259405
i4[4]	200	800	200	93	45070.271740
i5[4]	250	1000	1000	42	80054.374287

Algorithm used in this project in each instance finds valid solutions. Large Neighbourhood Search performs better when number of customers matches capacity, while it struggles when number of customers increases without increment in capacity.

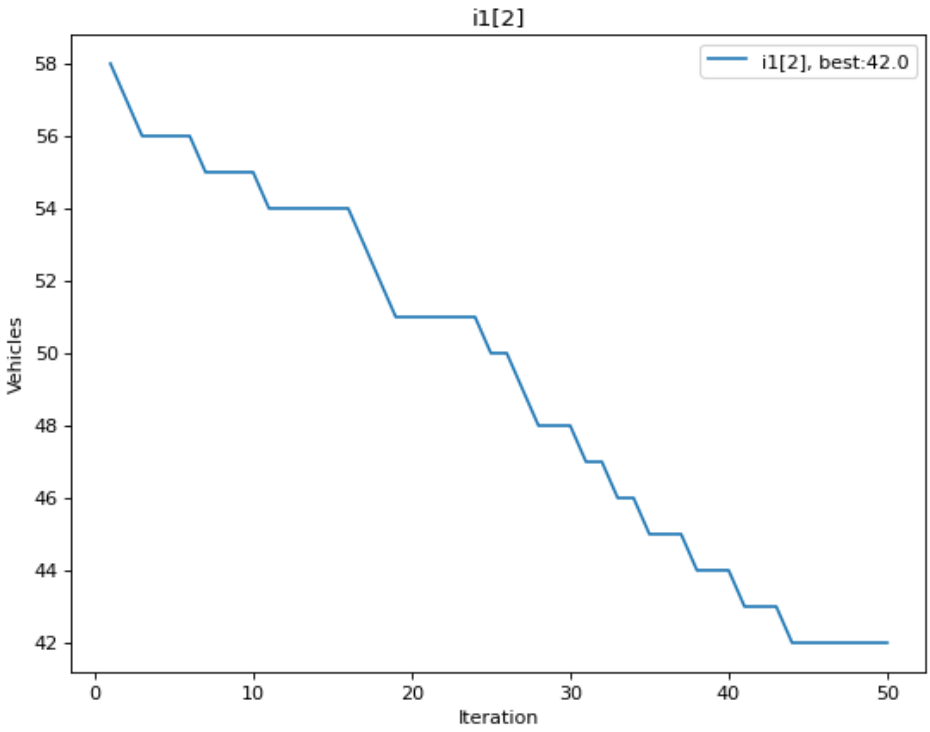

Figure 1: Time constraint - 1 minute

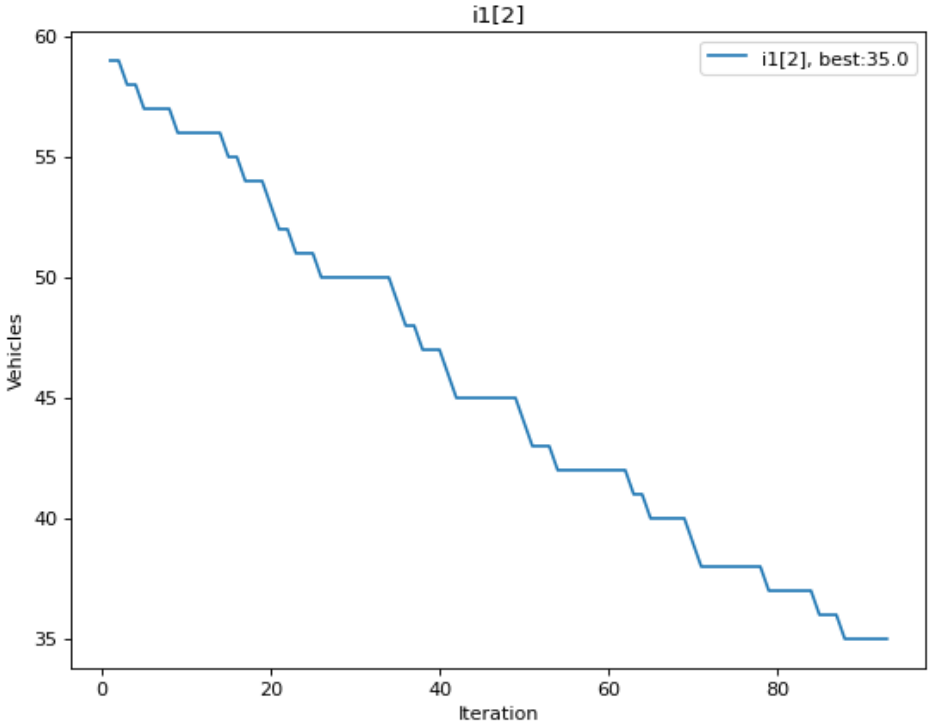

Figure 2: Time constraint - 5 minutes

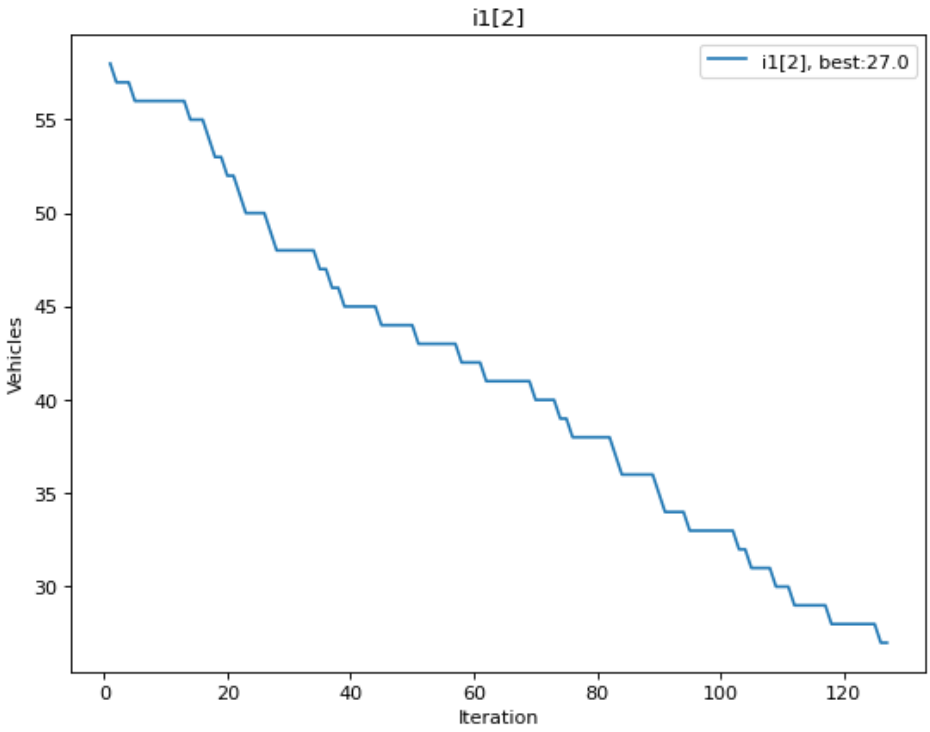

Figure 3: Time constraint - 6 hours

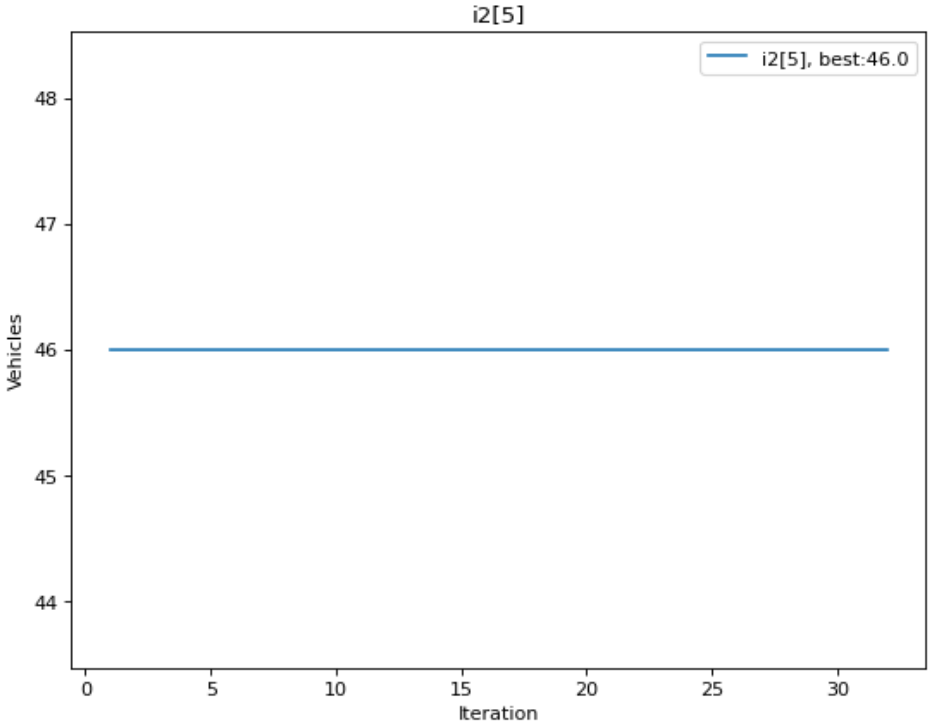

Figure 4: Time constraint - 1 minute

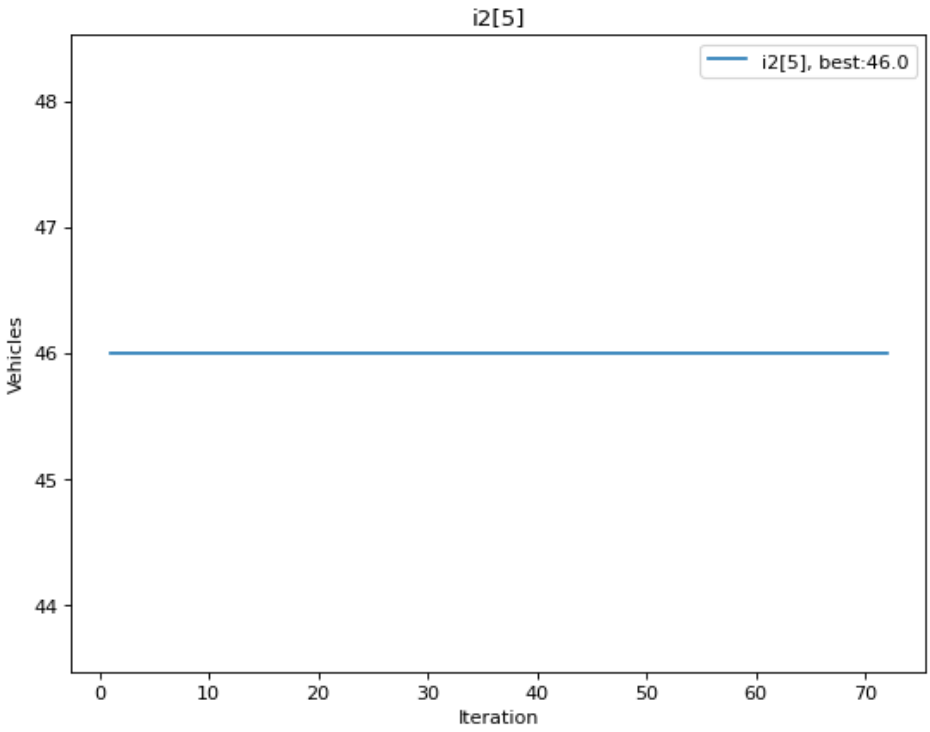

Figure 5: Time constraint - 5 minutes

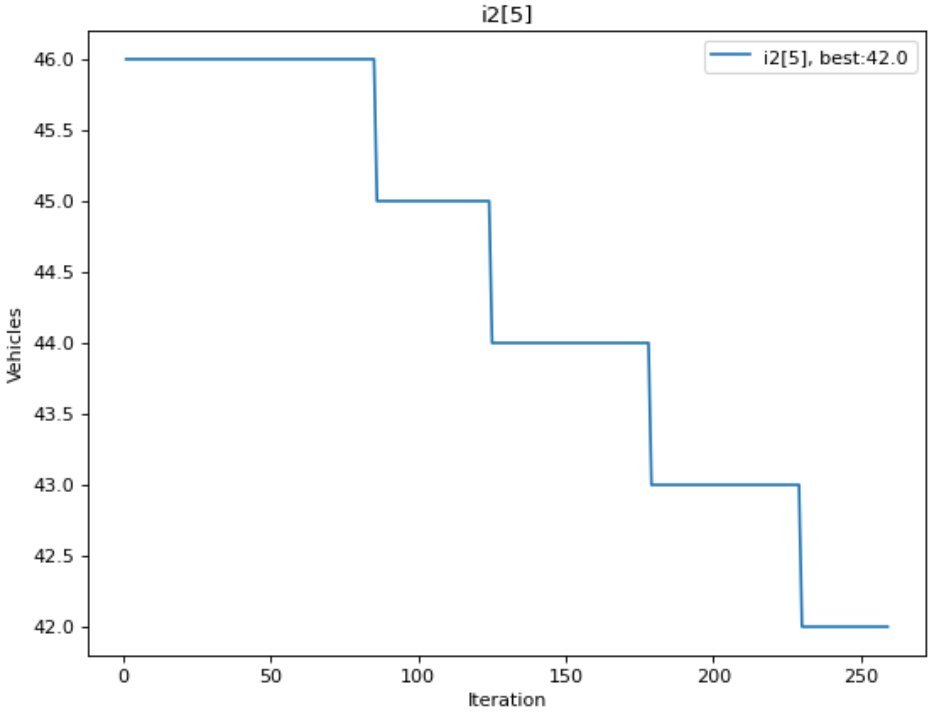

Figure 6: Time constraint - 6 hours

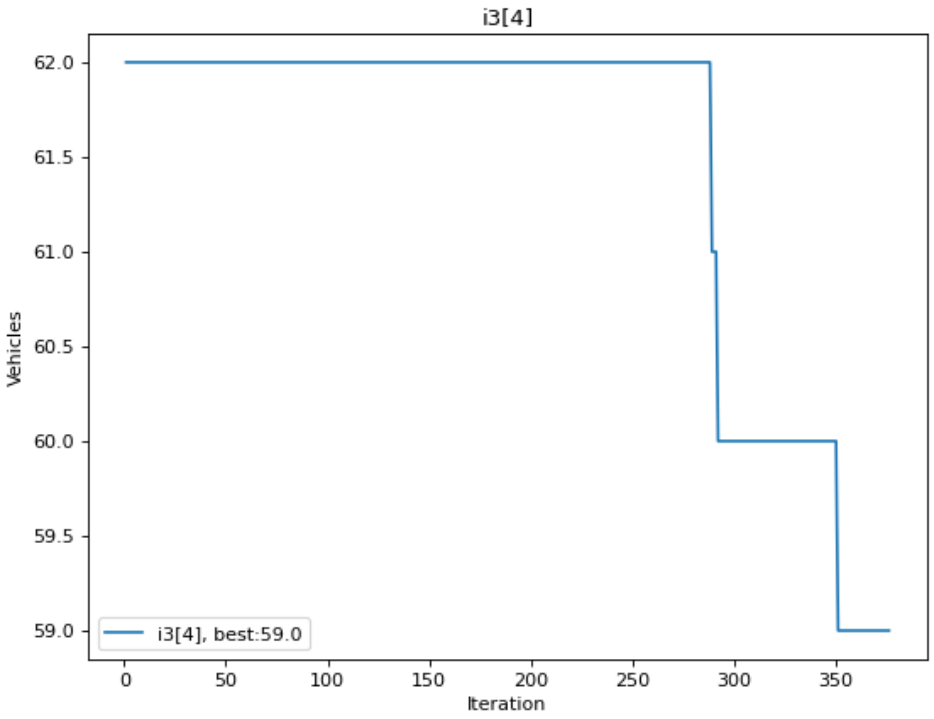

Figure 7: Time constraint - 1 minute

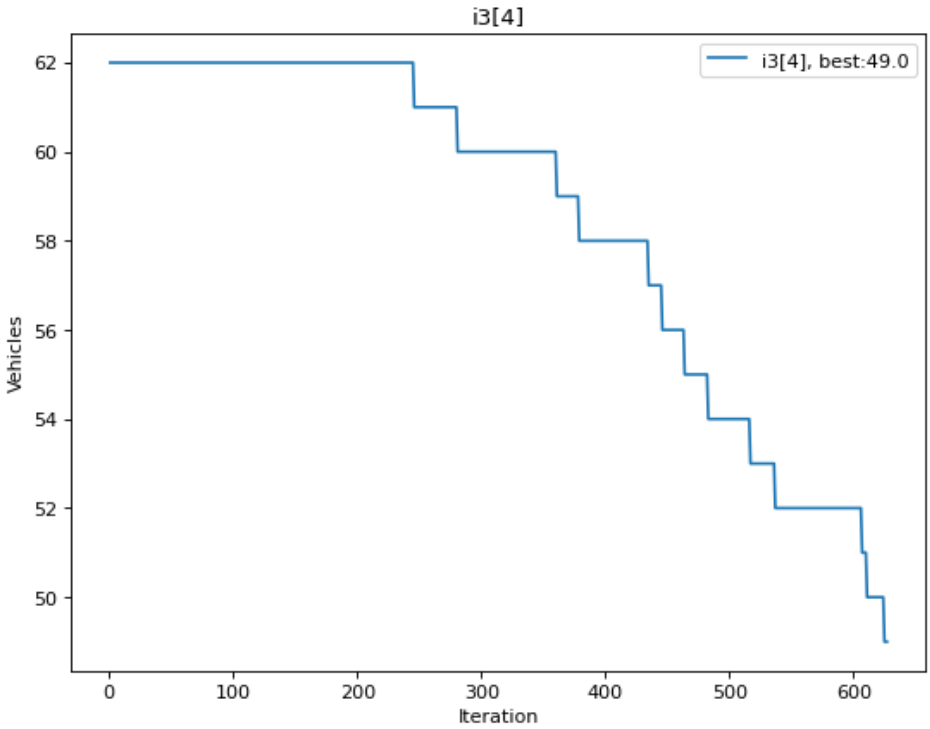

Figure 8: Time constraint - 5 minutes

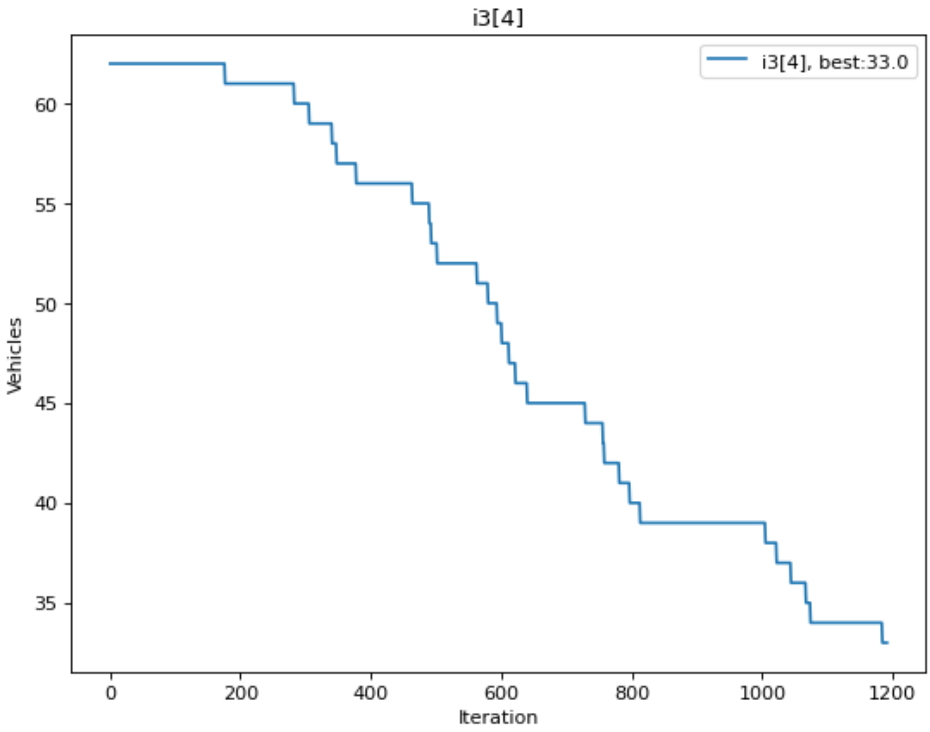

Figure 9: Time constraint - 6 hours

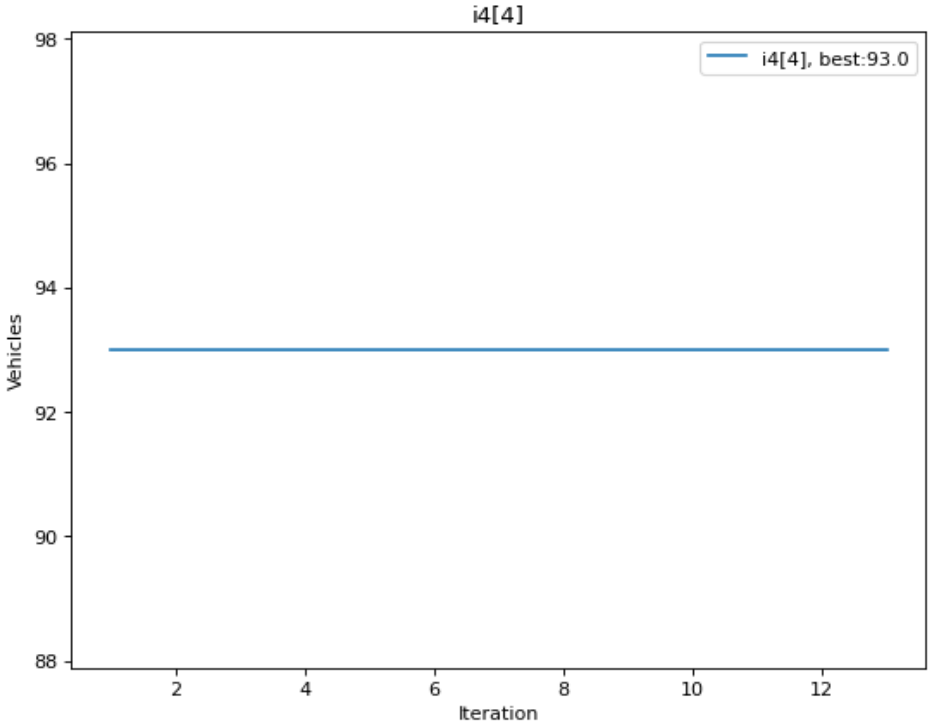

Figure 10: Time constraint - 1 minute

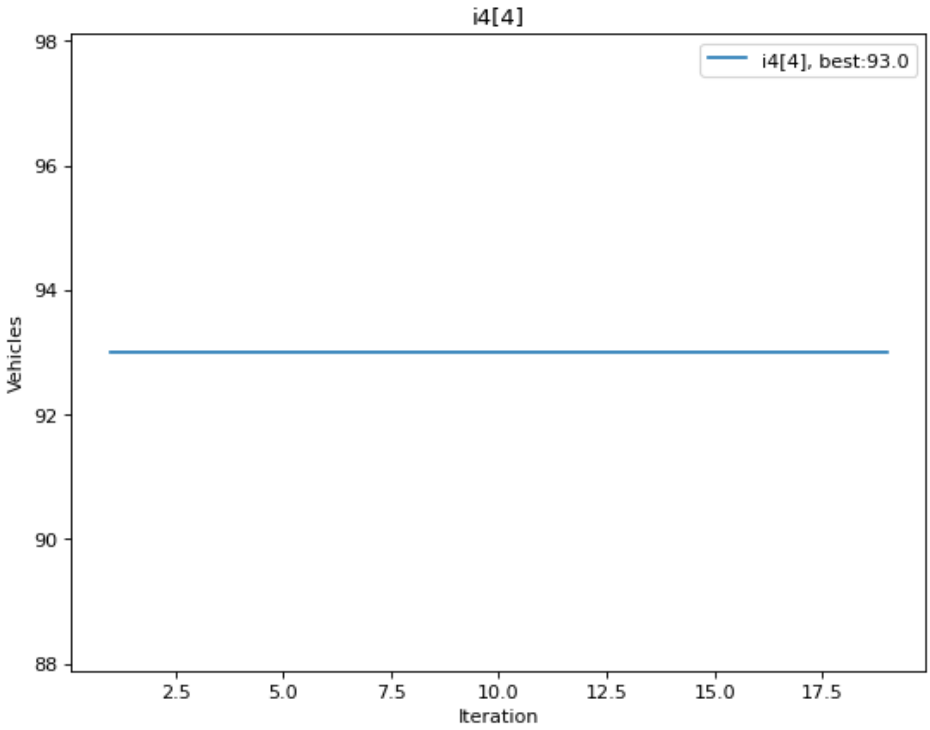

Figure 11: Time constraint - 5 minutes

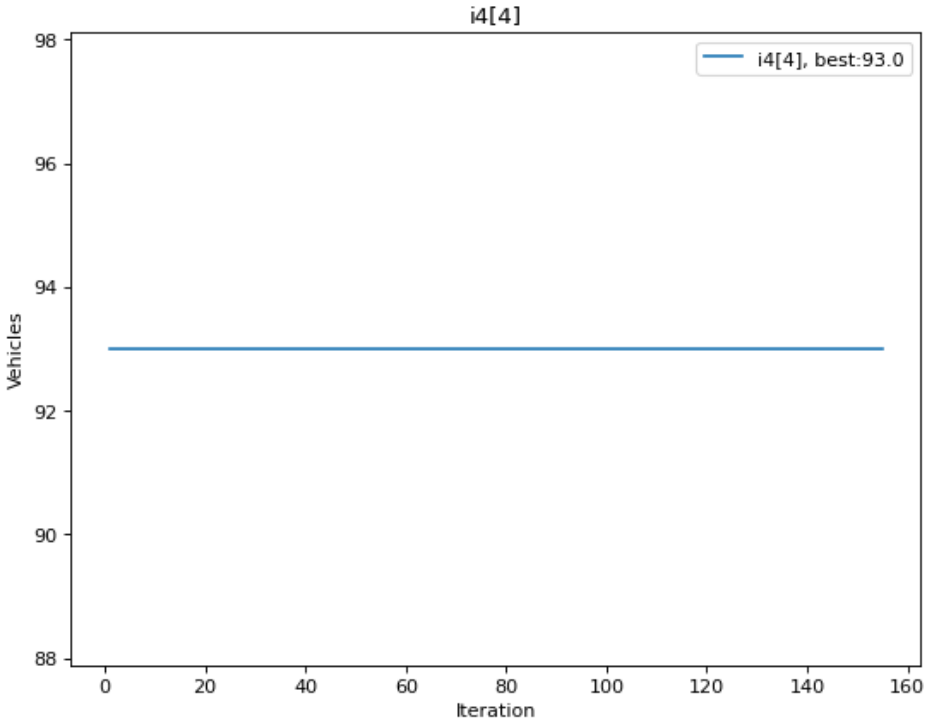

Figure 12: Time constraint - 6 hours

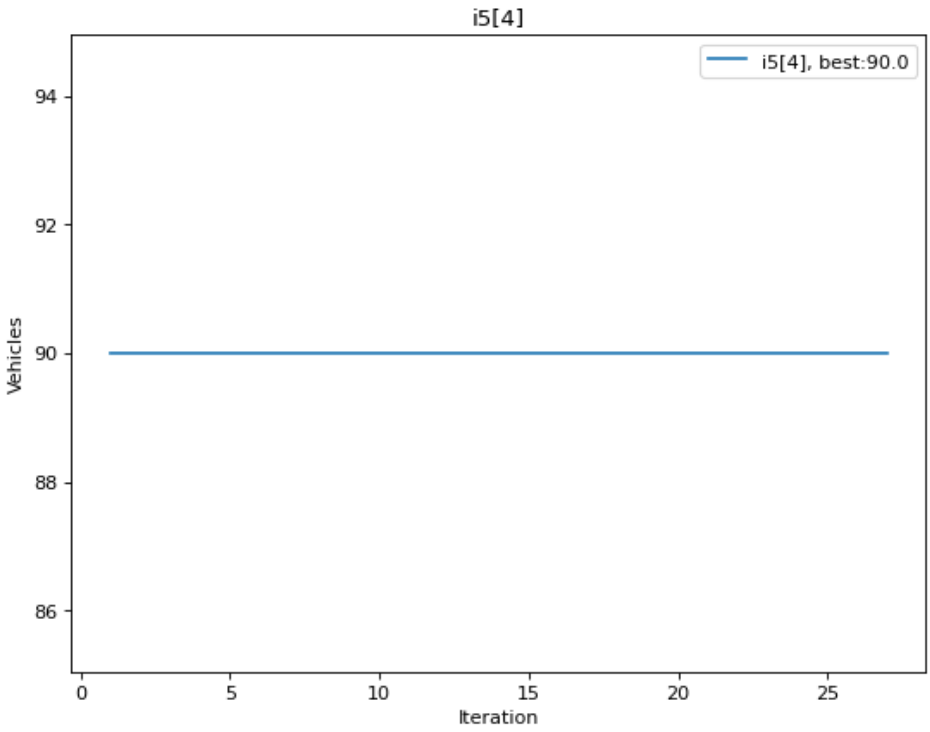

Figure 13: Time constraint - 1 minute

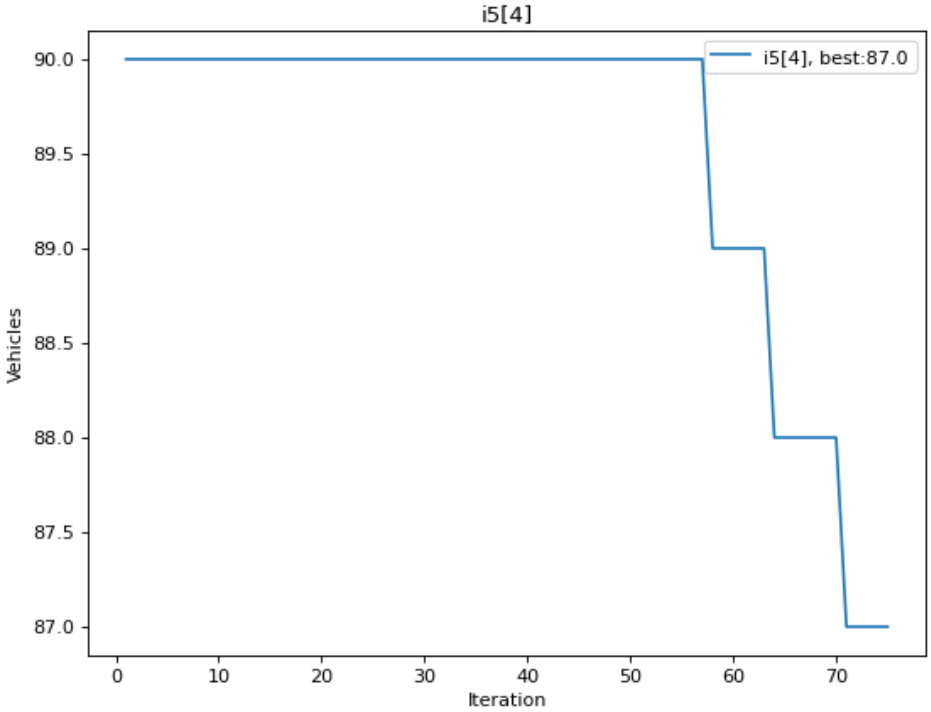

Figure 14: Time constraint - 5 minutes

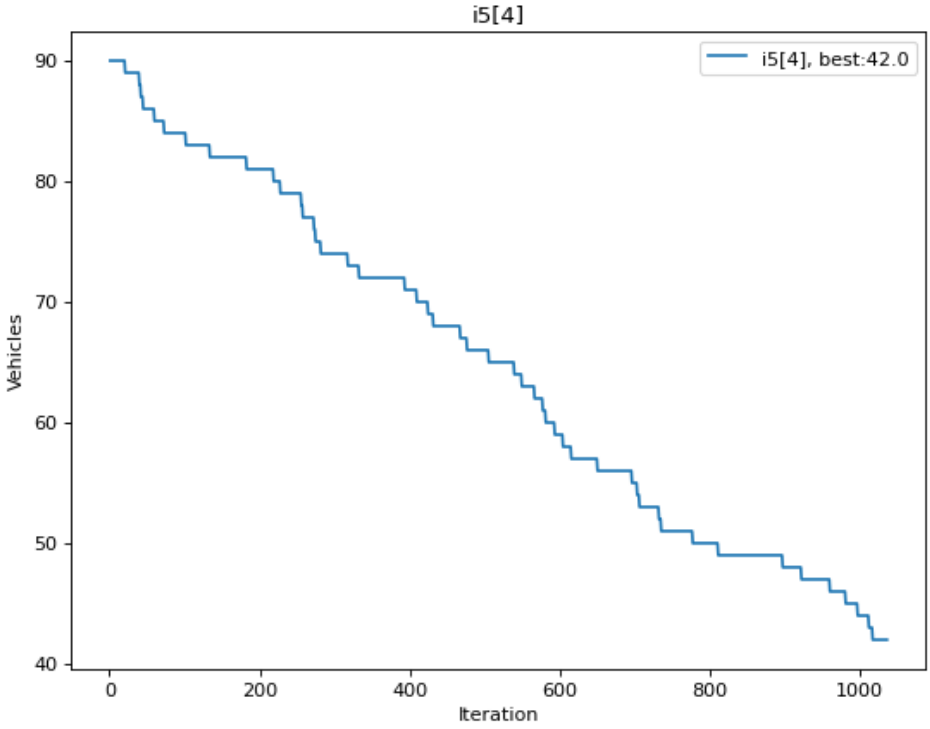

Figure 15: Time constraint - 6 hours

5 Conclusion

The CVRPTW problem involves finding routes for a fleet of vehicles to serve customers while considering capacity and time window constraints. A greedy algorithm was used for the construction phase and a large neighbourhood search was used for local search. The large neighbourhood search improves the solution by randomly removing and reinserting customers into different routes, but it performs better when the number of customers is closely matched with the capacity of the vehicles and may struggle with a higher number of customers without a corresponding increase in capacity. The algorithm was able to find valid solutions for each problem instance.