

**Master Programme**

**Heuristic Optimization Methods**

REPORT - Project  **Capacitated Vehicle Routing Problem with Time Windows**

**Petar Knežević**

**Ivan Derdić**

**Ac. year 2023/2024**

**Contents**

1 Summary of best-found results 3

2 Problem 4

2.1 Problem description 4

2.2 Problem instance 4

2.3 Problem formulation 5

3 Ant colony optimization 6

3.1 Pseudocode 6

3.2 Description 6

3.3 Analysis 7

3.4 Conclusion 7

# Summary of best-found results

**Instance 1**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 min | 5 min | No time constraints |
| Vehicles | 20 | 20 | 20 |
| Total Distance | 5357.85 | 5357.85 | 5357.85 |

**Instance 2**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 min | 5 min | No time constraints |
| Vehicles | 39 | 39 | 39 |
| Total Distance | 13404.93 | 13404.93 | 13404.93 |

**Instance 3**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 min | 5 min | No time constraints |
| Vehicles | 22 | 22 | 22 |
| Total Distance | 10370.73 | 10370.73 | 10370.73 |

**Instance 4**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 min | 5 min | No time constraints |
| Vehicles | 111 | 111 | 111 |
| Total Distance | 48925.61 | 48925.61 | 48925.61 |

**Instance 5**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 min | 5 min | No time constraints |
| Vehicles | 20 | 20 | 20 |
| Total Distance | 64535.12 | 64535.12 | 64535.12 |

**Instance 6**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 min | 5 min | No time constraints |
| Vehicles | 94 | 94 | 94 |
| Total Distance | 77369.22 | 77369.22 | 77369.22 |

Programming language: **Python**

# Problem

## Problem description

Capacitated vehicle routing problem with time windows (**CVRPTW**) belongs to the class of Vehicle Routing Problems (**VRP**), but, on top of the generic VRP formulation, defines additional constraints that occur in real-life scenarios. In VRP, the objective is to find a set of routes for a fleet of vehicles serving customers from a depot.

The *capacity constraint* in CVRP refers to the capacity of the vehicle: the total demand of all customers supplied on a single route must not exceed the vehicle capacity. On the other hand, the *time window constraint* in VRPTW refers to the interval in which a customer must be supplied (often called the “scheduling horizon”): all customers in a route need to be reached within their scheduling horizons, and the whole route needs to be started and finished within the working hours of the depot. CVRPTW includes both constraints.

## Problem instance

Problem instances define the number of vehicles, single vehicle capacity, and data about each customer: X and Y coordinates, resource demands, schedule horizon, and the duration of the service time. The data provided in each instance file is defined as follows:

|  |  |
| --- | --- |
| **- VEHICLE -** |  |
| NUMBER | Total number of available vehicles. |
| CAPACITY | Capacity of a single vehicle. All vehicles have the same capacity. Capacity is “spent” on fulfilling customer demands on a route serviced by that vehicle. |
| **- CUSTOMER -** |  |
| CUST NO. | Customer index. Index 0 corresponds to the depot, not a customer. |
| XCOORD | X coordinate of the customer location. |
| YCOORD | Y coordinate of the customer location. |
| DEMAND | Amount of resources that need to be delivered to a given customer by a vehicle on a route that services that customer. Demands “spend” the capacity of the vehicle on that route. For customer no. 0 (depot) demand equals 0. |
| READY TIME | The earliest time at which the start of the service can happen for a certain customer. For the depot (index 0) indicates the opening of the depot. |
| DUE TIME | The latest time at which the start of the service can happen for a certain customer. For the depot (index 0) indicates the closing of the depot. |
| SERVICE TIME | The duration of the service – the amount of time which the service vehicle needs to spend at the customer location. Note: Service only needs to start between READY TIME and DUE DATE, it can finish after the DUE DATE. |

## Problem formulation

The problem defines the following **constraints**:

1. Each customer is served by exactly one vehicle/route, with the resource amounts that equal their demands.

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's 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.

The time at which a vehicle starts the service at the location of a certain customer in a route equals the sum of: (1) the time at which the previous customer was serviced (or the time at which the depot was left, for the first customer), (2) SERVICE TIME at the previous location, and (3) the ceiling of the distance between previous and current location. The distance is calculated as Euclidean distance. If that value is smaller than READY TIME for the current customer, then the service of the current customer starts at READY TIME.

The primary **objective** is to minimize the number of vehicles by which all the customers can be serviced, while the secondary objective is to minimize the sum of distances on all routes.

# Ant colony optimization

## Pseudocode

Start Ant colony optimization algorithm

Initialize pheromone trails

While(!termination criterion):

For each ant:

Select initial customer si

S = all viable next customers to select from customer si

Select next customer j with probability pij

Update pheromone trails

Evaporation

Reinforcement using best solution

Return best solution

## Description

We decided to solve this problem using ant colony optimization. Ants use collective behavior to achieve difficult tasks such as finding the shortest path to a food source and the transport of the food. Similarly, in this problem, we need to find the minimum number of vehicles to serve all customers and do it in the shortest possible route.

For the solution representation, we decided to use a list of all vehicle routes, so that we can check if the solution is feasible.

For the fitness function, we used a combination of the number of total vehicles used and the total distances traveled. The fitness function should be good for solution evaluation and push the solution towards both objectives.

The termination criterion was either a specific time period or the convergence around the best solution.

The ant colony optimization design issues that we had to tackle were ant behavior, pheromone trails, and pheromone updating strategy.

We started constructing the ant behavior by finding a set of viable customers to visit next from the current customer, the ones that the ants can visit before the due time. To save some time, we told them to only visit customers whose ready time already started so we could avoid the meaningless waiting. We then navigated them away from the depot to serve customers that are far away sooner rather than later. We combined that with the information that we got from the pheromone trails using the parameters alpha and beta to find solutions.

Pheromone trails are used to memorize good solutions to guide new ants toward them. We implemented it as a dictionary of values starting at 1, which gets updated by the evaporation and reinforcement process.

In the evaporation process, all pheromones decrease by a fixed decay rate, and in the reinforcement process, the pheromone trails are updated based on the fitness function of the last constructed solution.

## Analysis

Let’s analyze our algorithm and the final solution. We had 6 instances with different customer numbers and 3 different time periods to try our algorithm on, so we got 18 solutions in total. What we can see straight away is that our algorithm finds a local optimum in the first few iterations, can’t escape it, and finally converges around it.

In theory, the smaller the alpha, the greedier the solution, and the smaller the beta, the solution becomes more dependent on the pheromone trails. However, our experiments with different alphas and betas only resulted in slightly different solutions. We couldn’t escape the early convergence.

Our algorithm is in general slow for large instances. For the smallest instance it makes around 25 iterations and for the largest instance the first iteration lasts longer than a minute.

## Conclusion

To sum up, our algorithm gives us solutions that are close to the theoretical optimum. However, it could be better. Nobody likes to get stuck in the local optimum. The thing that made the problem even harder was the time constraint due to the size of some instances, so we had to combine speed and efficiency. In terms of time, we can be thankful for the early convergence.

Some potential improvements for our algorithm could be using more ants or diversifying more in the start. Using more ants would make the algorithm even slower but could improve the solution for smaller instances. On the other hand, diversifying would give us worse results if we can’t speed up the algorithm. Using a faster programming language would make the code run faster and we would have more room for experimentation.