UNIVERSITY OF ZAGREB
Faculty of Electrical Engineering and Computing

Master Programme

Heuristic Optimization Methods

REPORT – Project

Capacitated Vehicle Routing Problem with Time Windows

Dora Kašik

Katarina Mikulić

Ac. year 2025/2026

Contents

1 Problem description

The Capacitated Vehicle Routing Problem with Time Windows (CVRPTW) is an extension of the classical Vehicle Routing Problem (VRP), which is a well-known combinatorial optimization problem in logistics and transportation. The goal of VRP is to determine an **optimal set of routes** for a fleet of vehicles that serve a given set of customers, all starting and ending at a central depot, while minimizing a predefined objective function.

In the basic VRP formulation, each customer must be visited exactly once by a single vehicle, and all vehicle routes begin and end at the depot. CVRPTW extends this formulation by incorporating two additional constraints that frequently arise in real-world applications: **vehicle capacity** constraints and **customer time window** constraints.

The capacity constraint, inherited from the Capacitated Vehicle Routing Problem (CVRP), limits the total demand that a vehicle can serve on a single route. Each vehicle has a fixed maximum capacity, and the sum of demands of all customers assigned to that vehicle must not exceed this capacity.

The time window constraint, originating from the Vehicle Routing Problem with Time Windows (VRPTW), requires each customer to be serviced within a predefined time interval, known as the scheduling horizon or time window. A vehicle may arrive before the beginning of a customer's time window, in which case it must wait until service can begin, but arriving after the end of the time window is not allowed. Additionally, all routes must start and finish within the working hours of the depot.

CVRPTW combines both capacity and time window constraints, making it significantly more complex than the basic VRP. The problem is classified as **NP-hard**, and exact solution methods become computationally infeasible for larger instances. As a result, heuristic and metaheuristic optimization methods are commonly used to obtain high-quality approximate solutions within reasonable computation times.

2 Algorithm description(s)

There were many different algorithms implemented and tested, but two of them stood out with their solution quality. We decided to keep both and compare them in the rest of the report.

Both algorithms start from an initial feasible solution constructed by a greedy algorithm (more in section 2.3), which represents a complete set of vehicle routes satisfying all problem constraints. This solution is evaluated using a fitness function that incorporates the objective value and constraint feasibility.

The first algorithm is a **hybrid implementation of simulated annealing (SA)** metaheuristic which uses variable neighborhood search (VNS) to further explore the solution neighborhood. In this implementation, the current solution is set to the initial solution, and the best solution found so far is stored separately.

At each iteration, the algorithm generates a candidate solution from the current one using a short execution of a Variable Neighborhood Search (VNS) procedure. In this hybrid mode, VNS is applied for a limited time and with a restricted neighborhood size, enabling a deeper local search around the current solution before applying the SA acceptance criterion. This hybridization aims to combine the diversification capabilities of SA with the intensification strength of VNS.

Each candidate solution is evaluated for feasibility with respect to capacity and time window constraints. Only feasible solutions are considered for acceptance. The acceptance decision is based on the classical SA rule: if the candidate solution improves the fitness value, it is always accepted; otherwise, it may still be accepted with a probability that depends on the fitness difference and the current temperature.

The temperature parameter controls the level of randomness in the search. It is initialized to a predefined value and gradually reduced according to a geometric cooling schedule. To prevent premature convergence, a reheating mechanism is applied: when the temperature drops below a minimum threshold, it is reset to its initial value, allowing the search to regain exploration capability.

The algorithm terminates when a predefined time limit is reached. The best solution encountered during the search is returned as the final result, together with its routing structure and service schedule.

The second one is an implementation of **basic VNS** metaheuristic which uses relocate and swap operators. The initial solution is stored as the best solution found so far.

The search process is organized into two nested loops controlled by a time limit and a neighborhood index k. The outer loop ensures that the algorithm runs until the allocated computation time is exhausted. The inner loop iterates over neighborhood structures up to a predefined maximum neighborhood index k_{max}.

At each iteration, a candidate solution is generated by applying a neighborhood move to the current best solution. In this variant of the algorithm, two neighborhood operators are used: relocate and swap. These operators are alternated based on the neighborhood index, enabling systematic exploration of different local modifications.

Each candidate solution is evaluated for feasibility with respect to capacity and time window constraints. If the candidate solution is feasible and yields an improvement in the fitness value, it replaces the current best solution. In this case, the neighborhood index is reset to its initial value, intensifying the search around the new improved solution. If no improvement is achieved, the neighborhood index is incremented, and the algorithm proceeds to explore a larger or different neighborhood.

The algorithm terminates when the specified time limit is reached. The best solution found during the search is returned as the final result, including the corresponding routing configuration and service schedule.

2.1 Solution representation

A solution to the CVRPTW is represented as a set of vehicle routes, where each route corresponds to a single vehicle starting and ending at the depot. The solution representation explicitly encodes both the routing and scheduling aspects of the problem.

The first element in the solution file specifies the total number of routes/vehicles used. Each subsequent line defines one route as an ordered sequence of customer indices, beginning and ending with the depot (denoted by node 0). For each visited customer, the time of service start is included in parentheses, which allows direct verification of the time window constraints.

Formally, a route is represented as:

$$0(t_0) \rightarrow\ i_1(t_1) \rightarrow\ i_2(t_2) \rightarrow \cdots \rightarrow 0(t_{end})$$

where i_k denotes a customer and t_k the corresponding service start time.

The last line of the solution file contains the total travel distance of all routes, which corresponds to the value of the objective function.

2.2 Objective and fitness functions

There are two objectives, the **primary is to minimize the number of vehicles** by which all the customers can be serviced, while the **secondary is to minimize the sum of distances** on all routes. The implemented fitness function is defined further below. Formal representations of objective functions are as follows:

Primary objective function

$$\min V$$

Where V is the total number of vehicles used.

Secondary objective function

$$\min \sum_{1}^{n} \sum_{i,j\, \in K} d_{ij}$$

$$d_{ij} = \sqrt{\left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2}$$

Where i, j represent customer indices, n is the number of routes, x_i and x_j are x coordinates of a customer, y_i and y_j are y coordinates of a customer, K is a set of all customer indices, i, j are elements of K and d_{ij} represents euclidian distance between two customers.

Implemented fitness function

- Priority 1: Number of vehicles (weight = 1,000,000)
- Priority 2: Penalties for non-valid solutions
- Priority 3: Total distance

$$penalty = (total\_excess\_load \cdot penalty\_capacity) + (total\_time\_violation \cdot penalty\_time)$$

$$fitness = (V * 1000000) + D + \sum_{l \in L} \max(0, Q_l - Capacity) \ * 2000 + \sum_{i \in K} \max\,(0, A_i - DueDate_i) * 500$$

Where V is the number of used vehicles, D is Euclidean distance, Q_l is the total demand on route l and A_i is the arrival time at node i.

2.3 Initial solution construction

A greedy construction heuristic is used to generate an initial feasible solution for the CVRPTW. The algorithm follows a nearest-neighbor strategy while enforcing vehicle capacity constraints and customer time window constraints.

The algorithm maintains a set of unvisited customers and incrementally builds routes for individual vehicles. Each route starts at the depot with zero load and time. At each step, the algorithm selects the nearest feasible unvisited customer that can be served without violating the vehicle capacity constraint, the customer's time window, or the depot's working hours. If a vehicle arrives before the beginning of a customer's time window, waiting is allowed until service can start.

Once no additional customer can be feasibly inserted into the current route, the route is closed and a new vehicle is initialized. This process is repeated until all customers are assigned to routes or no further feasible assignments are possible.

2.4 Termination criterion

Both algorithms have a **time-based termination criterion**. The search process is terminated when the predefined time limit is reached (1 minute, 5 minutes, or "unlimited" runtime – set to 15 minutes because of the limited time we had for running final tests).

2.5 Heuristic specific design elements

Several heuristic-specific design elements were incorporated in order to improve the effectiveness and robustness of the proposed optimization approach:

- The **acceptance strategy** differs between the applied heuristics. Hybrid SA employs a **probabilistic acceptance mechanism** that allows the acceptance of non-improving solutions depending on the current temperature, enabling better diversification and escape from local optima. In contrast, VNS applies a **strict improvement rule** and accepts only solutions that improve the current best fitness value.
- A **geometric cooling schedule** is used in the hybrid SA algorithm, with an additional **reheating** mechanism that resets the temperature when it drops below a predefined threshold. This design choice helps prevent premature convergence.

3 Algorithm pseudocode(s)

3.1 Hybrid SA

```
Start hybrid SA
  Set current solution to initial solution  // curr_r = initial
  Check constraints and fitness of current solution // evaluator.py
  Set initial temperature parameter
  While time limit not reached do:
      Get solution candidate from VNS
      Get candidate validity, fitness, distance and detailed routes from evaluator // evaluator.py
      If candidate is valid:
          Calculate delta  // fitness – current_fitness
          If delta < 0 or random number < $e^{-\frac{\Delta}{T}}$ :
              Set current solution to candidate  // curr_r = cand
              Set current fitness to fitness  // curr_f = f
              If fitness is better than best fitness:  // f < best_f
                  Update best solution, fitness, distance and detailed routes to
                  candidate, fitness, distance and detailed routes
      Apply alpha parameter for cooling  // T = T * alpha, geometric cooling
      If temperature < 0.1:
          Set temperature back to initial temperature  // Reheating
  Return best solution, distance and detailed routes  // best_r, best_d, best_det
End hybrid SA
```

3.2 VNS

```
Start VNS
  Set best routes to the initial solution // best_routes = initial
  Get best fitness, best distance and best detailed routes from evaluator  // evaluator.py
  While time limit not reached do:
      Set neighborhood index k to 1  // k = 1
      While k is less than or equal to k_max do:
          Get candidate by applying relocate move if k is odd or swap move if k is even
          Get candidate validity, fitness, distance and detailed routes from evaluator
          If candidate is valid and fitness is better than best fitness  // f < best_f
              Update best routes, fitness, distance and detailed routes to
              candidate, fitness, distance and detailed routes
              Set k to 1
          Else:
              Increment k by 1  // k = k + 1
          Check if time has gone over the time limit, if yes, stop
  Return best routes, distance and detailed routes  // best_routes, best_d, best_det
End VNS
```

4 Computing resources

Two different computers were used to run tests in order to obtain solutions more quickly. The specifications of a computer on which the first three insances were tested are:

CPU	13th Gen Intel(R) Core(TM) i7-13700H (2.40 GHz)
RAM	16 GB
OS	64-bit Windows 11
GPU	NVIDIA GeForce RTX 4060, 8 GB

The specifications of a computer on which the remaining three instances were tested are:

CPU	12th Gen Intel(R) Core(TM) i7-12700H (2.30 GHz)
RAM	16 GB
OS	64-bit Windows 11
GPU	NVIDIA GeForce RTX 3060, 6 GB

5 Analysis of results and discussion

The performance of the proposed heuristic algorithms was evaluated with respect to solution quality and execution time under the given computational constraints. Due to the stochastic nature of the applied metaheuristics, the obtained results varied between runs so particular attention was paid to parameter selection. Parameters whose values were changed to observe their influence on the solution were initial temperature (T_init) and alpha for hybrid SA, and k_{max} for VNS.

Based on many of the previous test runs, a fixed set of parameter values was selected for the final experiments. These values demonstrated a good balance between solution quality and computational effort and produced the best solutions in most cases during the previous runs. The final experiment was run with following values:

T_init = 1000, alpha = 0.97 for hybrid SA and k_{max} = 20 for VNS.

The results also highlight the importance of runtime as a termination criterion. Increasing the allowed execution time generally led to improvements in solution quality. This behavior is consistent with the expected convergence characteristics of metaheuristic optimization methods. Furthermore, the hybridization of Simulated Annealing with a short Variable Neighborhood Search phase proved beneficial in enhancing local search intensification.

Overall, the obtained results confirm that the selected heuristic design and parameter configuration are well suited for the CVRPTW instances considered in this project. Despite limited parameter variation in the final evaluation phase, the chosen setup provides reliable performance and represents a reasonable trade-off between solution quality and execution time.

6 Conclusion and future improvements

In this project, we explored a wide range of metaheuristic strategies for the vehicle routing problem with capacity and time constraints. While several algorithms were implemented during the development phase, our analysis and comparative testing led us to focus on the **Variable Neighborhood Search (VNS)** and a **Hybrid Simulated Annealing (SA)** approach as the most effective solutions. Our results confirm that while a greedy heuristic provides a necessary baseline for feasibility, metaheuristic intensification is essential for minimizing the primary objective which is the total vehicle count. The hybrid approach, which incorporates a local search phase within a probabilistic acceptance framework, proved to be a robust method for escaping local optima through its systematic cooling and reheating schedule.

Despite the high quality of the results, several advanced improvements could further enhance the solver's performance to meet state-of-the-art standards. The most significant architectural improvement would be transitioning from a single-point search to a population-based approach, such as **Hybrid Genetic Search (HGS)**. Implementing a Dual-Population strategy (maintaining separate populations for feasible and infeasible solutions) would allow the algorithm to explore the "boundaries of feasibility" which is crucial for complex routing problems. To avoid premature convergence within this population, a Biased Fitness mechanism could be introduced to rank solutions based on both their objective value and their contribution to diversity, using advanced metrics like the Broken-pairs distance.

Regarding local search, introducing more complex exchange operators would allow for moving nodes between different routes without fixing their insertion positions. This explores a significantly larger neighborhood of potential moves in a fraction of the time, greatly increasing the chances of finding a global optimum. To handle high-density scenarios more efficiently, a granular search could be applied by restricting the optimization to only the nearest neighbors for each customer. Furthermore, an **Adaptive Large Neighborhood Search (ALNS)** layer could be added to dynamically choose "Ruin-and-Recreate" operators (such as Shaw Removal) based on their past success in improving the solution, which would drastically increase the algorithm's diversification capabilities.

Finally, overall performance could be improved through numerical and implementation optimizations. Moving toward a constant-time evaluation method by pre-calculating data for route segments would allow the algorithm to test millions of additional moves per second. Additionally, an adaptive penalty mechanism could dynamically adjust the weights for capacity and time violations to better guide the search through the solution space. To eliminate floating-point precision issues and the computational overhead of repeated calculations during distance evaluations, scaling the coordinate system would allow the entire search to be conducted using integer arithmetic, leading to faster and more consistent results.

7 References

- Arnold, F., & Sörensen, K. (2019). Knowledge-guided local search for the vehicle routing problem.
- Christiaens, J., & Vanden Berghe, G. (2020). Slack induction by string removals for vehicle routing problems.
- CVRPLIB. The Capacitated Vehicle Routing Problem Library. [https://galgos.inf.puc-rio.br/cvrplib/]
- Google OR-Tools. Routing Library. [https://developers.google.com/optimization/routing]
- Helsgaun, K. (2017). An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained vehicle routing problems.
- Toth, P., & Vigo, D. (2014). Vehicle Routing: Problems, Methods, and Applications (2nd ed.). SIAM.
- Uchoa, E., et al. (2026). The XL Instances for the Capacitated Vehicle Routing Problem. ArXiv preprint arXiv:2601.11467.
- Vidal, T. (2022). Hybrid genetic search for the CVRP: Open-source implementation and SWAP neighborhood*.
- Wouda, N. A., Lan, L., & Kool, W. (2024). PyVRP: a high-performance VRP solver package.
- Zheng, J., et al. (2024). UDC: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems.