Solving the VRPTW through Constructive Algorithms

Thomas Martinod[1]

[1] *Heuristics, EAFIT University*

September 9, 2024

Introduction

This project applies three different methods to solve the Vehicle Routing Problem with Time Windows (VRPTW) across 18 instances. The methods are:

- Constructive Heuristic
- Reactive GRASP
- Ant Colony Optimization (ACO)

The goal is to compare the performance of each method in terms of vehicle efficiency, total distance, and computational time.

All results and code are available in the repository:
`https://github.com/thomas-martinod/heuristics-for-vehicle-routing-problem-with-time-windows`

VRPTW

The Vehicle Routing Problem with Time Windows (VRPTW) involves a fleet of K vehicles, each with a capacity Q, tasked with serving a set of customers on a graph $G = (V, E)$, starting and ending at a central depot. Every customer i has a demand q_i and a specific time window $[e_i, l_i]$ for service [1]. The following constraints must be met:

- Each customer is visited exactly once.
- Routes must begin and end at the depot.
- The total demand on any route must not exceed the vehicle's capacity Q.
- Service at each customer must start within their designated time window, and vehicles must return to the depot by l_0.

The objective is to minimize both the total distance traveled, D, and the number of routes (vehicles) used, K.

file_reader

We begin by introducing five common methods utilized across all solutions: the constructive method, Reactive GRASP, and ACO.

The `file_reader` method reads the `.txt` files located in the `Instances VRPTW` folder. These files contain the problem instances we will use to demonstrate the three methods discussed by the lecturer.

Using an object-oriented programming (OOP) class structure, we store the attributes of each node (including the depot), such as coordinates, time windows, and service times.

`file_writer` and `visualization`

These are our output generators. The `file_writer` method produces an Excel file that contains the results for the 18 instances in the format specified by [2].

Additionally, the `visualization` method graphically displays the generated routes for each instance using scatter plots from the matplotlib library.

As a note, all figures and Excel outputs are saved in the *figures* and *results* directories of the repository.

feasibility

This special file checks the two main constraints when adding a new node to a route.

- **Capacity Constraint:**

$$\sum_{i \in r} q_i + q_j \leq Q$$

- **Time Window Constraint:**

$$t_{r_j} = \max(t_{r_{j-1}} + T_{r_{j-1}, r_j}, e_j) + s_i$$

The condition to verify is:

$$t_{r_j} < e_i$$

The function naturally returns a boolean value.

distance_finder

This is the final auxiliary file, and it provides several utilities:

- It includes a function to calculate the Euclidean distance between two nodes.
- It calculates the distance from each node to every other node (using the Euclidean distance) and stores it in a matrix.
- It also allows us to find the maximum and minimum values of the sets $\{e_i\}$, $\{l_i\}$, and $\{t_{ij}\}$ for a fixed index, which are useful for normalizing these values later.

Lower Bound on Routes

Remember, we have two objective functions to minimize: the total distance traveled D and the number of vehicles used K. Since theoretical optima are only achievable in toy problems, we use lower bounds to measure how far our solutions are from an ideal solution.

- The function `lower_bound_routes` (in all 3 methods) calculates the minimum number of routes required which is based on the total demand of customers and the vehicle capacity.
- Mathematically:

$$\text{Lower Bound on Routes} = \text{LB Q} := \left\lceil \frac{\sum_{i=1}^{n} q_i}{Q} \right\rceil$$

where n is the number of non-depot nodes.

Lower Bound on Distance (MST)

- The function `lower_bound_mst` computes a lower bound on the total distance using the Minimum Spanning Tree (MST) method. The depot and customers are represented as a graph, with edges corresponding to the distances between nodes.
- Mathematically:

$$\text{Lower Bound Distance} = \text{LB D} := \sum_{\text{e} \in \text{MST}} e$$

Constructive Heuristic for Route Selection

Here we introduce the first method for route selection, the constructed method. Routes are constructed iteratively using a weighted criterion to select the next customer.

- The function `constructive_route_selection` minimizes a convex combination of the following weights:

$$c_{\text{distance}} \times \text{Distance between nodes}$$
$$c_{\text{inf}} \times \text{Lower time window}$$
$$c_{\text{sup}} \times \text{Upper time window}$$

- The goal is to build feasible routes until all customers have been served. A route is fixed, and the next node is chosen by minimizing the weighted sum until no other node is feasible.

Calculating the GAP

After generating routes, which are saved in different arrays, we calculate the GAP for both the number of routes and the total distance (this is common to all three methods).

The GAP for the number of routes is given by:

$$\text{GAP}_K = \frac{K - \text{LB K}}{\text{LB K}}$$

The GAP for the total distance is calculated as:

$$\text{GAP}_D = \frac{D - \text{LB D}}{\text{LB D}}$$

Execution and Results

The function `vrptw_solver` reads instances from a directory and solves each using the constructive heuristic. For each instance, it computes:

- Lower bounds for both routes and distances.
- Constructed routes and total distance.
- GAP for routes and distance.
- Execution time.

The results are saved in an Excel file and visualized through plots.

Now, we proceed to analyze the Reactive GRASP method.

Reactive GRASP

Reactive GRASP is an adaptive version of the GRASP algorithm that dynamically adjusts the parameter α during execution. The parameter α controls the balance between greediness and randomness when selecting the next element to add to the solution.

By adjusting α, the algorithm can explore different areas of the solution space, improving the chances of finding better solutions. The probabilities of α values are updated based on the quality of the solutions generated, favoring better-performing values as the search progresses.

Reactive GRASP: Assigning α

A set of possible values for α is defined as $\{\alpha_1, \alpha_2, ..., \alpha_m\}$. Initially, the probability of selecting each α is calculated as:

$$p_i = \frac{1}{m}, \quad \forall\ i \in \{1, 2, ..., m\}$$

For subsequent iterations, the probability is updated as follows:

$$p_i = \frac{q_i}{\sum_{j=1}^{m} q_j}, \qquad q_i = \frac{z^*}{A_i}$$

Here, z^* represents the best solution found, and A_i is the average value of all solutions obtained for α_i. After all this α definition for sorting, the constructive method is used.

Mathematical Development of ACO

Ants select their next move based on a combination of local heuristics and the amount of pheromone on the path [3]:

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} [\eta_{ij}]^{\beta}}{\sum_{l \in N_k(i)} [\tau_{il}(t)]^{\alpha} [\eta_{il}]^{\beta}} \qquad \forall\ j \in P$$

Where:

- $p_{ij}^{k}(t)$ is the probability that ant k selects node j from node i.
- $\eta_{ij} = \frac{1}{d_{ij}}$ is the heuristic information (inverse of the distance between i and j).
- α and β are parameters controlling the influence of pheromone and heuristic information, respectively.
- $N_k(i)$ is the set of available nodes for ant k at node i.

Next Movement Selection

Ants select their next movement based on a combination of local heuristic information and the amount of pheromone on the path [3]:

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} [\eta_{ij}]^{\beta}}{\sum_{l \in N_k(i)} [\tau_{il}(t)]^{\alpha} [\eta_{il}]^{\beta}} \qquad \forall\ j \in P$$

Where:

- $p_{ij}^{k}(t)$ is the probability that ant k selects node j from node i.
- $\eta_{ij} = \frac{1}{d_{ij}}$ is the heuristic information (inverse of the distance between i and j).
- α and β are parameters that control the influence of the pheromone and the heuristic information, respectively.
- $N_k(i)$ is the set of available nodes for ant k at node i.

Results Table

					Constructive					GRASP					ACO				
Instance	n	Q	LB K	LB D	K	D	GAP_K	GAP_D	Time (ms)	K	D	GAP_K	GAP_D	Time (ms)	K	D	GAP_K	GAP_D	Time (ms)
VRPTW1	25	200	3	98.991	3	257.922	0.000	1.606	1	4	259.744	0.333	1.624	111	4	230.351	0.333	1.327	7052
VRPTW2	25	700	1	141.823	2	328.895	1.000	1.319	3	2	215.543	1.000	0.520	152	2	257.412	1.000	0.815	8673
VRPTW3	25	200	2	276.782	8	634.907	3.000	1.294	7	9	715.537	3.500	1.585	93	9	633.437	3.500	1.289	6279
VRPTW4	25	1000	1	276.782	2	745.996	1.000	1.695	0	4	608.271	3.000	1.198	131	4	531.360	3.000	0.920	7150
VRPTW5	25	200	3	178.406	5	567.906	0.667	2.183	8	5	561.447	0.667	2.147	90	5	545.655	0.667	2.059	6970
VRPTW6	25	1000	1	178.406	2	681.966	1.000	2.823	7	3	558.563	2.000	2.131	121	4	446.500	3.000	1.503	7356
VRPTW7	50	200	5	195.661	5	493.598	0.000	1.523	7	9	680.576	0.800	2.478	384	7	440.278	0.400	1.250	24804
VRPTW8	50	700	2	264.105	2	515.023	0.000	0.950	16	5	657.717	1.500	1.490	500	4	421.801	1.000	0.597	37121
VRPTW9	50	200	4	417.383	13	1256.124	2.250	2.010	7	13	1232.530	2.250	1.953	347	15	1164.915	2.750	1.791	21122
VRPTW10	50	1000	1	417.383	3	1435.358	2.000	2.439	16	4	1124.526	3.000	1.694	713	7	983.105	6.000	1.355	26551
VRPTW11	50	200	5	315.738	9	1060.250	0.800	2.358	6	9	1095.061	0.800	2.468	418	11	1124.058	1.200	2.560	21399
VRPTW12	50	1000	1	315.738	3	1391.370	2.000	3.407	4	6	1032.763	5.000	2.271	503	7	979.574	6.000	2.103	25528
VRPTW13	100	200	10	417.299	10	889.047	0.000	1.131	24	16	1678.450	0.600	3.022	1608	13	1071.047	0.300	1.567	88009
VRPTW14	100	700	3	492.469	3	779.902	0.000	0.584	54	7	1343.666	1.333	1.728	2249	5	955.949	0.667	0.941	163632
VRPTW15	100	200	9	562.257	22	1962.402	1.750	2.490	21	22	1973.364	1.750	2.510	1497	27	1995.108	2.375	2.548	73173
VRPTW16	100	1000	2	562.257	5	2208.197	1.500	2.927	51	7	1833.461	2.500	2.261	2701	10	1572.657	4.000	1.797	115320
VRPTW17	100	200	9	563.998	18	2185.075	1.000	2.874	23	18	2106.481	1.000	2.735	1590	23	2162.244	1.556	2.834	77210
VRPTW18	100	1000	2	563.998	5	2938.973	1.500	4.211	31	6	1725.319	2.000	2.059	3021	11	1801.119	4.500	2.194	103949

Figure: Summary of the results obtained for each instance, including total distance, number of routes, and GAP values for both metrics.

Graphical Validation (Constructive)

It is first needed a graphical way to see if our solutions actually make sense. We show the solutions of the three methods in particular for instance 13.

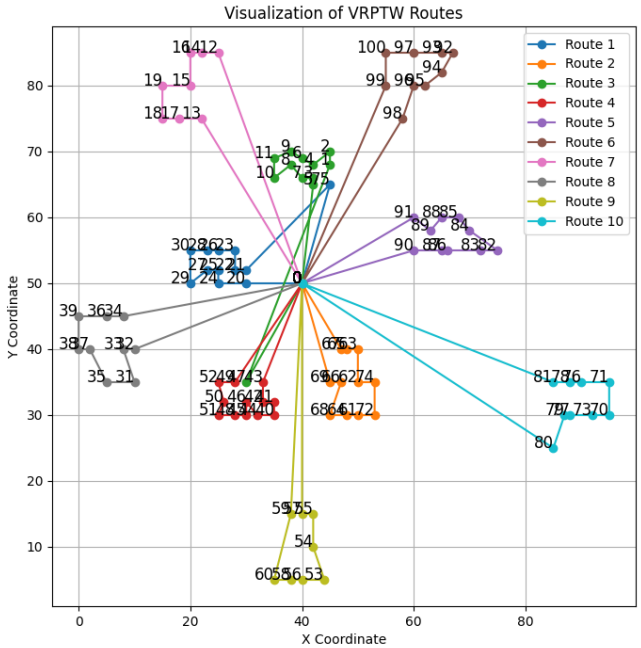

Figure: Instance 13 with the constructive method.

Graphical Validation (GRASP)

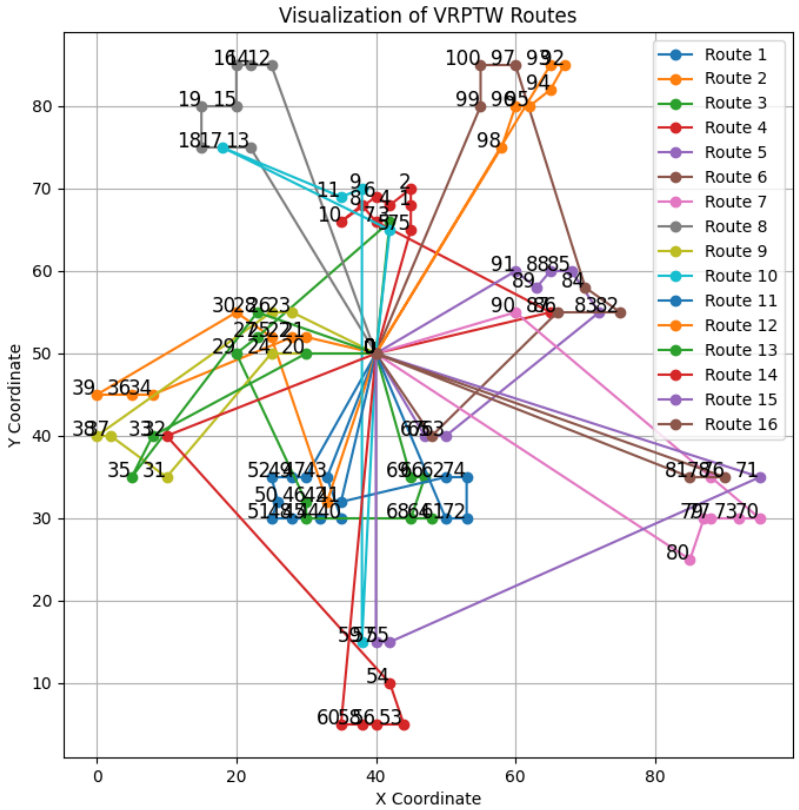

Figure: Instance 13 with the Reactive GRASP method.

Graphical Validation (ACO)

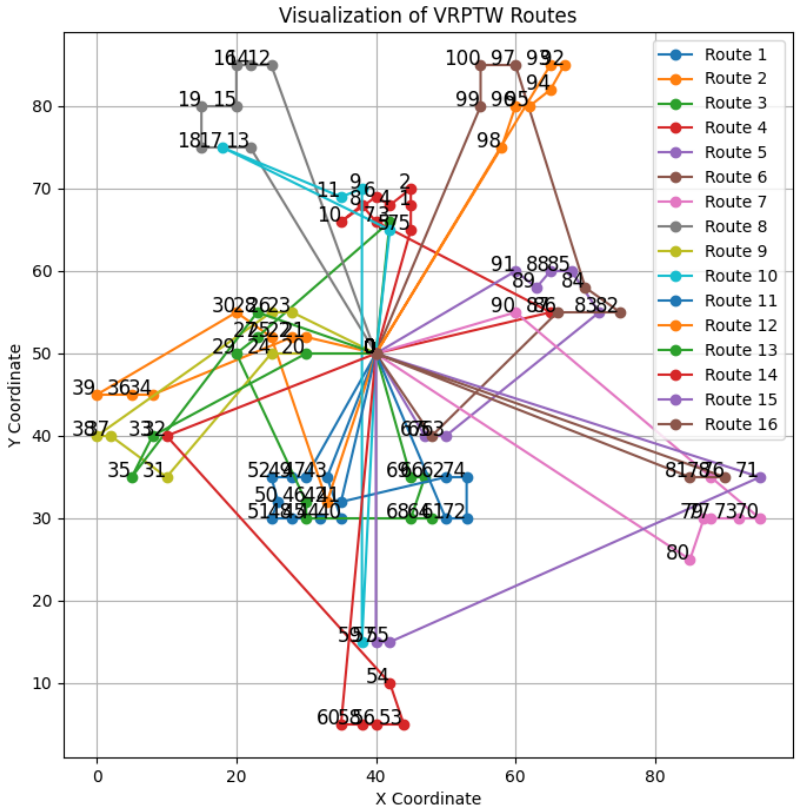

Figure: Instance 13 with the ACO method.

Methods Comparison

Here we show the mean GAPs for the three methods.

Method	Mean GAP in K	Mean GAP in D
Constructive	1.081	2.101
GRASP	1.835	1.993
ACO	2.347	1.636

Table: GAP values for the number of routes (K) and total distance (D)

We can observe that the constructive method focuses more on minimizing the number of routes (K), while ACO prioritizes minimizing the total distance (D). GRASP offers a balance between the two.

Weights Selection

The following table shows the impact of different weight combinations on the performance of both the constructive and GRASP methods.

$(c_{\text{distance}}, c_{\text{inf}}, c_{\text{sup}})$	**Mean GAP K**	**Mean GAP D**
(0.5, 0.4, 0.1)	108.148%	210.125%
(0.5, 0.25, 0.25)	110.926%	213.521%
(0.4, 0.3, 0.3)	113.704%	227.656%
(0.3, 0.4, 0.3)	124.043%	244.242%
(0.33, 0.33, 0.33)	118.488%	235.212%

Table: Mean GAP for the number of routes (K) and total distance (D) under different weight combinations.

Proper calibration of the weights c_{distance}, c_{inf}, and c_{sup} is essential in any multiobjective optimization problem, as shown by the varying impact of different weight configurations, nevevrtheless they are not essential on the quality.

On the Length of α

The specific values of α are not as critical as the number of α values considered during the process. A larger set of α values provides more flexibility in balancing greediness and randomness, allowing the algorithm to explore a broader range of solutions.

In our experiments, we tested lengths of 3, 5, 7, and 9 α values. We found that a length of 7 strikes the best balance between greediness and exploration, leading to more robust results. Length 3 produced a 5% higher mean GAP, length 5 showed about a 1% difference, while length 9 resulted in approximately a 1% improvement.

ACO Parameter Influence Ranking

Due to the long runtime of the ACO code, a detailed parameter analysis in terms of the GAP was not conducted. However, graphical observations revealed the following parameter influences:

- **Beta (β)**: The most influential parameter, it controls the importance of the heuristic (distance) in decision-making.
- **Alpha (α)**: The second most influential, it determines the influence of pheromones in the route selection process.
- **Rho (ρ)**: Controls pheromone evaporation, which impacts convergence and exploration. While not the most influential, it is the most sensitive. With high evaporation, ants get lost and choose random paths, and with low evaporation, they tend to follow the same route repeatedly.

Other parameters are not as vital.

Discussion

- The lower bounds are quite strict and may not provide a close approximation to the optimal theoretical solution, as evidenced by the high GAP values. However, they are a valuable mathematical tool to gauge how close a solution is to being optimal.
- In general, introducing more vehicles tends to relax the time window constraints, as shown in Table 1. However, restricting the number of vehicles is the critical decision that ensures a feasible solution. Our experimentation shows that most instances become infeasible when the number of vehicles is too limited.

Discussion

- It is clear that the straightforward logic of the constructive and GRASP methods allows for a lower runtime while keeping the number of vehicles used to a minimum. This is crucial when considering the cost of adding a new vehicle. However, if vehicle costs are negligible and runtime is not a concern (as in data bus connections), ACO might be the best method for optimizing node interconnections.
- Although it was not explicitly mentioned, in problems of this magnitude, it is essential to have a toy problem with a known solution to ensure that the algorithm is performing as expected.

Conclusion

In this project, we explored three different methods to solve the Vehicle Routing Problem with Time Windows (VRPTW): the constructive heuristic, Reactive GRASP, and Ant Colony Optimization (ACO). Each method was evaluated based on two key metrics: the number of vehicles (K) and the total distance traveled (D).

The results show that:

- The constructive method performed best in minimizing the number of vehicles (K), making it ideal when vehicle cost is a critical factor.
- ACO excelled in minimizing the total distance traveled (D), making it more suitable for scenarios where efficiency and connection optimization are prioritized.
- GRASP provided a balance between the two, offering a good compromise in terms of both vehicle count and distance.

Overall, the choice of method depends on the specific priorities of the problem, such as cost, time constraints, or solution accuracy.

References

[1] J. C. R. Agudelo, *Métodos constructivos y aleatorizados*, Curso: Heurística – CM0439, Universidad EAFIT, Medellín, Colombia, Jul. 15, 2024.

[2] J. C. R. Agudelo, *Métodos constructivos y aleatorizados*, Curso: Heurística – CM0439, 2024.

[3] Wikipedia contributors, *Ant colony optimization algorithms — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms, Accessed: 2024-09-06, 2023.

[4] J. C. R. Agudelo, *Heurística: Construcciones grasp*, Diapositivas, 2023.

[5] J. C. R. Agudelo, *Colonias de hormigas heurística*, 08 - ACO, 2023.

[6] OpenAI, *Chatgpt: A conversational ai model*, https://chat.openai.com/, Accessed: 2024-09-08, 2024.