# A Branch-Cut-and-Price Algorithm for the Time-Dependent Electric Vehicle Routing Problem with Time Windows

•••

Python implementation of the paper

SM3800057
Tavano Matteo

- Adoption of **Electric vehicles (EVs)** within the <u>last-mile delivery</u> is considered one of the key transformations towards more suitable logistics.

- The inclusion of EVs introduces new **operational constraints** such as battery consumption, recharge during the routes and restricted driving ranges.

- This project aims to address the **Vehicle Routing Problem (VRP)**, starting from initial depot, all customers within the city must be served once and the route must end to the depot.

- Furthermore, customers have a specific **Time-Window (TW)** and service time, thus making the problem more realistic.

- The paper proposes a unifying framework to integrate other critical **time-dependent (TD)** times arising during the operations previously studied in the literature, such as the waiting and charging times.

- A general version of the **Time-Dependent Electric Vehicle Routing Problem with Time Windows (TDEVRPTW)** is proposed, which incorporates the time-dependent nature of the transportation network both in terms of travel times and the battery consumption.

- The **Branch Cut-and-Price** (**BCP**) algorithm is an effective approach for determining the optimal routing strategy for all EVs in the city, while efficiently satisfying all operational constraints.

- This project aims to replicate the authors' proposed solution using **Python** with **Gurobi** as the solver. Its purpose is to implement the problem specification and solution framework, enabling the practical resolution of small instances.

- **Scalability** tests will be conducted to verify the correct resolution of larger, scaled-up instances.

## Introduction

## Evolution of the Electric Vehicle Routing Problem

- Traditional **Vehicle Routing Problems (VRPs)** were extended to **Electric Vehicle Routing Problems (EVRPs)** to account for limited battery capacity and the need for intermediate recharging stops.
- Early EVRP models assumed energy consumption depended only on distance traveled and used constant charging times.
- Later studies revealed that travel speed, vehicle load, and terrain gradient significantly affect battery discharge, especially in congested urban areas.
- The **Time-Dependent EVRPTW (TDEVRPTW)** incorporates traffic-induced speed variations and integrates variable charging and waiting times for more realistic routing strategies.

## Literature Review: Time-Dependency and EVs

- Early work on **Green VRPs (GVRPs)** , such as Erdoğan and Miller-Hooks (2012), introduced alternative-fuel vehicles with limited driving ranges.

- The **EVRPTW** (Schneider et al., 2014) added battery constraints, time windows, and recharging strategies, later enhanced with partial recharge policies, charging station capacity limits, and more realistic energy consumption models.

- Parallel research on **Time-Dependent VRPs (TDVRPs)** addressed congestion effects by modeling variable travel speeds and times, requiring advanced exact and heuristic algorithms.

- Recent studies begin bridging EVRPTW and TDVRP, but comprehensive models integrating congestion effects into EV battery discharge remain scarce and challenging.

## TDEVRPTW

- Building on **Goeke & Schneider (2015)** and **Ichoua et al. (2003)**, generalize the EVRPTW to the **TDEVRPTW**, capturing congestion effects on both travel times and battery consumption.

- This framework extends classical time-dependency by integrating **variable charging times** and **waiting times**, modeled with continuous piecewise linear functions.

- A **state-of-the-art Branch-Cut-and-Price (BCP) algorithm has been developed** with a new labeling procedure, tailored preprocessing rules, and an improved branching scheme to reduce search complexity.

## Time-Dependent electric vehicle routing problem

## Problem Description

- The **TDEVRPTW** is modeled on a directed graph **D=(V,A)** , where each vertex $\mathbf{i \in V_i}$ represents either the depot, a customer, or a recharging station.
- Thus, $\mathbf{V=\{o,d\} \cup Vc \cup Vs}$ with **o** and **d** as the start and end depots, $\mathbf{V_c}$ the set of customers, and $\mathbf{V_s}$ the set of stations.
- Each arc $\mathbf{(i,j) \in A}$ corresponds to a path between two locations with travel cost $\mathbf{c_{ij}}$. Routes must start at **o** and finish at **d**, visiting each customer exactly once.
- Each customer $\mathbf{i \in Vc_i}$ has demand $\mathbf{q_i>0}$, a service time $\mathbf{s_i}$, and a time window $\mathbf{[a_i,b_i]}$ within which service must begin.
- Vehicles are homogeneous electric vehicles (EVs) with capacity **Q** and limited battery capacity **B** (kWh)

- Congestion is modeled using the following approach, where the planning horizon **[0,T]** is divided into intervals with **piecewise-constant speed functions $v_{ij}(t)$** for each arc. Travel times **$\tau_{ij}(t)$** are derived as continuous, piecewise-linear functions satisfying the **FIFO** property: delaying departure cannot result in an earlier arrival.
- Battery consumption depends on departure time, with **$\beta_{ij}(t)$** representing the energy used on **(i,j)** if leaving **i** at time **t**.
- Charging at stations is also time-dependent: each station $j \in V_s$ has an invertible, concave recharge function **$\hat{g}_j(t)$**.
- For a vehicle with battery level $w_0>0$ the charge gained after **t** units of time is

$$\mathbf{g_j(w_0,t)=\min\{B-w_0,\ \hat{g}_j(\hat{g}_j^{-1}(w_0)+t)-w0\}}$$

- These definitions generalize the classical EVRPTW, which is recovered when $\beta_{ij}(t)$ and g^j(t) are linear and $\tau_{ij}(t)$ is constant.

- The cost of a route $\mathbf{r=(o,v_1,\ldots,v_k,d)}$ is

$$c_r=\sum_{(i,j)\in r} c_{ij}$$

- A route is feasible if vehicle capacity is not exceeded, each customer is served within $[a_i,b_i]$ and the battery never depletes.

- Because travel times and consumption depend on speed and thus on departure times delaying departures can sometimes reduce energy use. (Limited to $b_i + s_i$ when visiting i).

- Finally, unlike many time-dependent VRPs that minimize makespan or duration, this formulation uses a **time-independent cost objective** , a choice motivated by both modeling considerations and algorithmic efficiency.

## Further characteristics of EVRP

- **Recharge policies** : Two main strategies exists, full recharge and partial recharge (vehicles can decide how much to charge, enlarging the solution space). **Desaulniers et al. (2016)** also introduced the single-recharge policy, limiting routes to one in-route recharge, in contrast to the multiple-recharge policy.

- **Station congestion** : immediate recharging upon arrival is unrealistic in peak hours. **Time-dependent waiting-time function** $\boldsymbol{\omega_s(t)}$ for each station $\boldsymbol{s \in V_s}$ to model expected waiting before a charger becomes available. These functions are continuous, piecewise-linear, and satisfy FIFO.

- **Recharge time models** : Originally, **g^(t)** was modeled as **linear** , implying constant recharge rates. Other studies, showed recharge times are **nonlinear** —the first 75% of charge takes about as long as the last 25%. They approximated this behavior with a **concave, continuous piecewise-linear function** with three segments.

- **Multiple charging modes** : Some EVs support different charging speeds or technologies. Each mode can be represented by its own recharge function, and stations with multiple modes can be duplicated in the graph, assigning fixed costs to each inbound arc to distinguish charging options.

## Battery Consumption Model

- Travel times $\boldsymbol{\tau_{ij}(t)}$ are derived from speed profiles $\mathbf{v_{ij}(t)}$ and distances $\mathbf{d_{ij}}$ they are continuous, piecewise-linear, and FIFO-compliant.
- Battery consumption $\boldsymbol{\beta_{ij}(t)}$ depends on speed-related energy use, incorporating aerodynamic drag, rolling resistance, and vehicle mass.
- Precomputing these functions reduces computational complexity during route feasibility checks.
- They enable fast evaluation in labeling and pricing steps of the BCP algorithm.

$$P(v,q) = \left(\frac{1}{2} \cdot cd \cdot \rho \cdot FS \cdot v^2 + (m_c + m_u q) \cdot g \cdot (\sin(\alpha) + cr \cdot \cos(\alpha))\right) \cdot v.$$

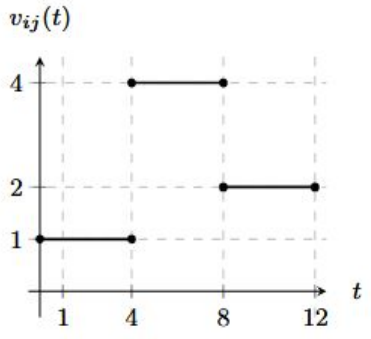
$$P(v) = h_1 v^3 + h_2 v.$$

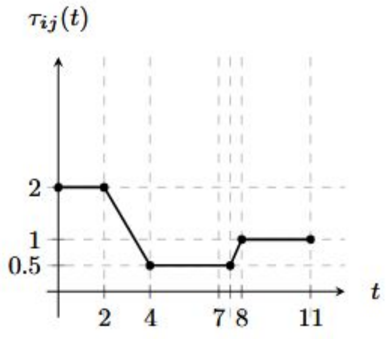
$h_1 \approx 1.54$ and $h_2 \approx 52.97$

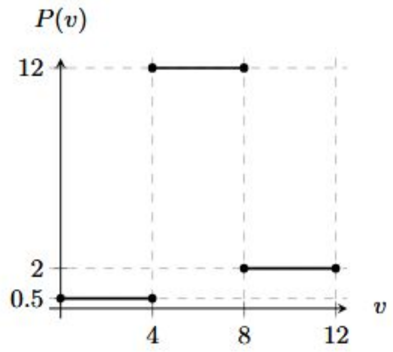
$$\beta_{ij}(t) = \int_t^{t+\tau_{ij}(t)} P(v_{ij}(x))dx$$

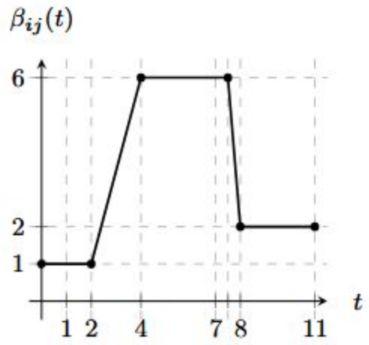
| Notation | Value | Description |
|---|---|---|
| $g$ | 9.81 meters per square second | Gravitational constant |
| $\rho$ | 1.42 kilograms per cubic meter | Air density |
| $cr$ | 0.006 | Coefficient rolling resistance |
| $cd$ | 0.9 | Coefficient aerodynamic drag |
| $FS$ | 2.40 square meters | Frontal surface |
| $m_c$ | 900 kilograms | Curb mass |

variables definitions


$v_{ij}(t)$
4
2
1
$t$
1
4
8
12


$\tau_{ij}(t)$
2
1
0.5
$t$
2
4
7
8
11


$P(v)$
12
2
0.5
$v$
4
8
12


$\beta_{ij}(t)$
6
2
1
$t$
1
2
4
7
8
11

## Battery consumption and feasible routes - TRV

- This represents a key concept within the model, as the feasibility of a route depends on maintaining a positive battery level during the entire trip.
- For an arc (i,j), let λ(t) be the **maximum battery** available if you are ready to leave i at time t. Then **TRV** gives the **maximum battery at j when you arrive at time t**.
- It allows for **leaving earlier and waiting**: define **H(t)** as all departure times **t′** from **i** that reach **j** no later than **t**

$$TRV_{ij}^{\lambda}(t) = \max_{t' \in H(t)} \{\lambda(t') - \beta_{ij}(t')\}.$$

- Intuitively, TRV **already represents waiting decisions** (arrive early, wait until the time window) without adding extra variables.

## Battery consumption and feasible routes - CHG

- Let µ(t) be the **maximum battery** if you are ready to **start charging** at station i at time t. Then **CHG** gives the **maximum battery when charging finishes at time t** :

$$CHG_i^{\mu}(t) = \max_{t' \le t, t' \in \mathrm{dom}(\mu)} \{\mu(t') + g_i(\mu(t'), t - t')\}.$$

- This formulation **allows starting earlier and waiting** : since charging has no cost, it is always better to charge.
- Intuitively, **CHG encodes "how long to charge"** decisions **without adding extra variables** , while preserving the convex, piecewise-linear structure needed by the labeling algorithm.

## Battery consumption and feasible routes - Implementation

- This represents a key concept within the model, as the feasibility of a route depends on maintaining a positive battery level during the entire trip.
- I model time‑dependent physics with a single core routine that computes both travel time and energy on each arc from departure time and speed profiles, ensuring consistency across the solver. (*get_travel_time_and_consumptions*)
- In the labeling step, **TRV** is realized implicitly when extending labels (arrival can include arrive earlier and wait), and **CHG** is handled at stations via piecewise-linear recharge functions (inverse for full charges), plus optional station waiting times.
- Strong dominance rules and an MBR based feasibility are also used.

## Preprocessing techiniques

## Handling service times

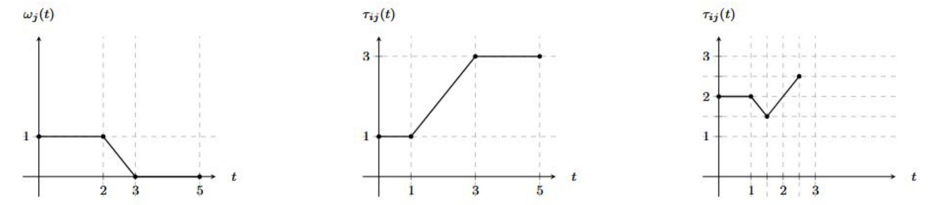
- Service time $\mathbf{s_i}$ of a customer $\mathbf{i} \in \mathbf{V_c}$ can be directly encoded into the travel time and battery discharge functions to simplify the model and definitions.

$$\tau_{ij}(t) := s_i + \tau_{ij}(t + s_i)$$
$$\beta_{ij}(t) := \beta_{ij}(t + s_i).$$

## Handling charging waiting times

- Time-dependent travel times can also naturally incorporate these waiting times by following a similar procedure than the one described for service times. Given a vertex **j** $\in$ **V** , then the travel time function for arc **(i, j)** $\in$ **A** is redefined as:

$$\tau_{ij}(t) := \tau_{ij}(t) + \omega_j(t + \tau_{ij}(t)).$$



(a) (b) (c)

## Reducing arcs and time windows

Briefly, an arc **$(i, j) \in A$** is removed if is either capacity infeasible, i.e. **$qi + qj > Q$** , or time infeasible, i.e. **$a_i + \tau_{ij} (a_i) > b_j$** .

## Minimum Battery Required (MBR)

- For each vertex $i \in V$, **MBR(i)** is defined as the minimum battery required to reach any recharge station (or eventually the depot), from vertex i without depleting the battery.
- Let $\boldsymbol{\beta_{ij}^{min} = \min\{ \beta_{ij}(t) \mid t \in dom(\beta_{ij})\}}$ be a lower bound on the energy consumption for arc $(i, j) \in A$. Then, for $i \in V$, a lower bound MBR(i) can be computed via a standard <u>shortest-path algorithm</u> using $\beta_{ij}$ as arc weights.
- Note that MBR(i) = 0 for $i \in V \setminus V_c$. In addition, arcs $(i, j) \in A$ satisfying $\boldsymbol{B - \beta_{ij} <}$ **MBR(j)** indicate that the battery level when reaching j cannot be enough to either reach another station or the depot, and therefore can be <u>safely discarded</u>.

## Exact Algorithm

## Set-partitioning formulation

- BCP algorithms and extended formulations based on the set-partitioning model stand as one of the most effective approaches to tackle different variants of the VRP.

- Let $\boldsymbol{\Omega}$ be the set of all the feasible routes for the TDEVRPTW. For each route $r \in \Omega$, $\mathbf{c_r}$ represents its cost and the constant $\mathbf{a_{ir}}$ indicates if route r visits customer $i \in V$ . Let $y_{\mathbf{r}}$ be a binary variable indicating whether a route $r \in \Omega$ is selected in the optimal solution

$$
\begin{aligned}
\min \sum_{r\in\Omega} c_r y_r \\
\text{s.t.} \sum_{r\in\Omega} a_{ir} y_r = 1, i \in V_c \\
y_r \in \{0,1\}, r \in \Omega.
\end{aligned}
$$

## Set-partitioning formulation: LP relaxation

A classical approach is to solve the **LP relaxation** at each node by using column generation.

**RMP** is generated using a subset of routes Ω'. For initialization, in Ω' is included an artificial route $r_{art}$ with high cost to guarantee the feasibility of the LP relaxation.

New columns are added (column generator) until the algorithm converges to a fractional optimal solution.

So, at each iteration LP relaxation on **$y \in \{0,1\}$ is computed to obtain the duals $\pi_i$ associated to the constraints.**

## Set-partitioning formulation: feasible routes

- If a feasible route with reduced cost $\mathbf{c_r\hat{} = c_r - \sum_{i\in r} \pi_i}$ **exists, then is added to the** RMP and the procedure is repeated.
- For the **TDEVRPTW** this pricing problem (found columns with reduced cost) is known as **TDEESPPRC** .
- Generally, this pricing problem is solved with labelling algorithms.
- Heuristics are used to speed up the performance of pricing and the exact algorithm is used when no negative reduced cost routes are found.

## Branching

- When no negative reduced-cost columns are found but the LP solution remains fractional, we apply a **branching scheme** to drive the solution to integrality. Starting from a variable with $0 < y_r < 1$, the solver branches and reruns column generation on two subproblems augmented with additional constraints.

- Two branching strategies are employed: the **Arc Branching Rule (ABR)** and the **Customer Branching Rule (CBR)**.

## Branching: ABR

Given an optimal fractional solution $\mathbf{y}^*$ of the LP relaxation, for each arc is defined:

$x^*_{ij} = \sum_{r \in \Omega, (i,j) \in r} y^*_r$

Therefore, branching consists in finding a variable $x_{ij}$ with non-integral value and opening two branches setting its value to 0 or 1.

In this implementation, ABR **forbids an arc** or **require it** (forbid all other successors of i and predecessors of j), so pricing simply avoids expanding along forbidden arcs.

## Branching: ABR

- With **charging stations** , arc variables must account for stations that can be revisited. Even if customer incident arcs are binary, the RMP may remain fractional. In the paper this is handled by adding LP bounds on $x_{ij}$ (introducing duals) rather than physically removing arcs, though that can impact efficiency.
- Solver prefers for this purpose **CBR** and falls-back to **ABR**.
- branch constraints are propagated to pricing and **filters candidates routes** against them before adding to the master.

## Branching: CBR

- CBR branches on **customer-successor** decisions: variable $z_{ij}$ indicates whether customer j is visited **immediately after** customer i in the customer sequence (stations are ignored).
- From an implementation point of view, first compute consecutive customer flows from the fractional LP solution and pick the pair (u,v whose flow is most fractional (closest to 0.5).
- The left branch **forbids** v as the next customer after u (and, for safety, forbids arc (u,v)); the right branch **requires** v as the next customer after u and forbids all other next-customer choices for u.

## Branching: CBR

- These branch constraints are carried into the pricing: label extensions that violate them are not generated, and candidate routes are filtered before being added to the master.
- CBR aligns well with set-partitioning over **elementary routes** , because it acts directly on the customer order, independent of how many station visits occur between customers.
- Compared to **ABR**, **CBR** targets the true combinatorial ambiguity (customer sequencing), which often reduces the number of B&B nodes explored in practice.

## The Pricing problem

- A **forward labeling algorithm** explores all feasible paths.
  The method implicitly generates an enumeration tree in which each node, called a label, represents a partial path starting from the initial depot o.

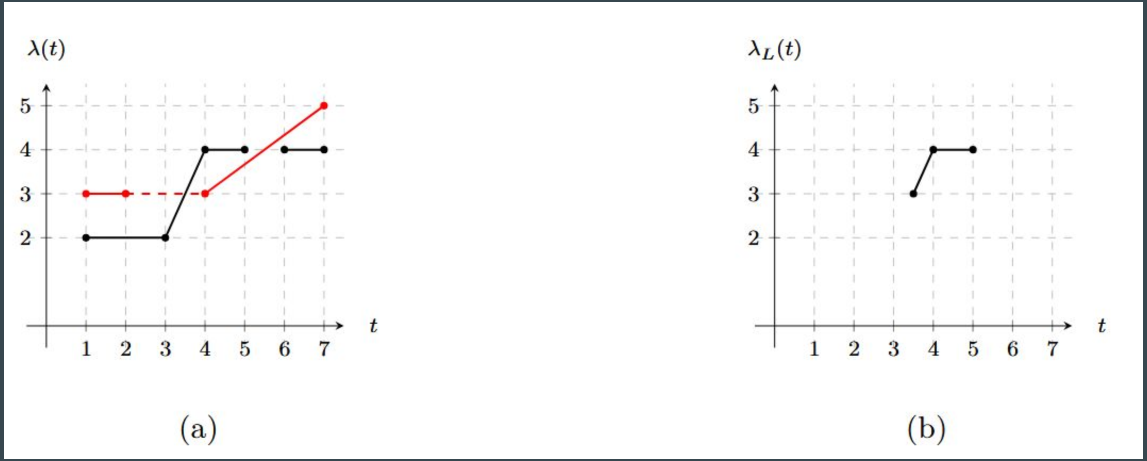
- To overcome exponential growth, pruning rules are incorporated to reduce the number of labels enumerated.

## Labeling Algorithm

- In a TD context, each label is usually either discarded or processed.
- Each label is presented as **L = (prev, v, S, q, c,   λ)** for each partial path in v (last vertex).
- λ(t) in this case is the maximum battery level at ready-time t.
- To avoid exponential growth of labels, cutting and dominance rules are applied.
- <u>MBR - Feasibility rule</u>: Discard time instances t of L, in which $\lambda_L(t) < MBR(v)$, i.e. it is infeasible because it cannot reach recharging station or depot. (if domain is empty, L is infeasible)

# Labeling Algorithm: Dominance

Let L and M be two labels satisfying: $v(L) = v(M)$, $q(M) \leq q(L)$, $c(M) \leq c(L)$, and $S(M) \subseteq S(L)$. Then, any time $t \in \mathrm{dom}(\lambda_L) \cap \mathrm{dom}(\lambda_M)$ such that $\lambda_L(t) \leq \lambda_M(t)$ can be discarded from dom(L). If $\mathrm{dom}(\lambda_L) = \varnothing$ then we say L is <u>fully dominated</u> and can be safely discarded.



## Label implementation

- The **Label class** stores the reduced cost, departure time, battery level, served demand, current node, last visited customer, the set of visited customers, and a pointer to its parent label.
- During extension (*extend_label*), the algorithm applies the **(MBR)** rule at departure, evaluates station waiting time $\omega_j(t)$, performs recharging, enforces time windows, reapplies MBR at the arrival node, and updates the reduced cost.
- Labels are maintained in a **min-heap** (priority queue), and complete routes are **reconstructed** when a label returns to the depot.
- **ABR** constraints are enforced by rejecting forbidden arcs during expansion; the expansion logic also prevents **station-to-station** chains.

## Label implementation

- *check_dominance* prunes labels by verifying cost, time, battery, demand, visited-set inclusion, and MBR slack conditions.

- The **MBR array** is precomputed at initialization and reused in feasibility and dominance checks.
- CBR in pricing is simplified, in order to have a lighter implementation.

## Pricing heuristics - paper

- At each pricing iteration, graph reduces keeping for each node only the k outcoming arcs with smallest reduced cost. Now labeling is performed on smallest graph to find faster negative columns.

- Relax-S / Relax-B: in the dominance rules described some assumptions are relaxed in order to prevent dominance of some Labels. Relax-S does not apply the condition $S(M) \subseteq S(L)$ visited customers, while Relax-B does not request anymore $\lambda_L(t) \leq \lambda_M(t)$.

## Pricing heuristics - implementation

- Pipeline heuristic has been applied, following the order Relax-S, K-shrink (3, 7, 12), relax-B, relax-S and finally k-shrink (k=7). If no reduced cost columns are found, exact pricing with timeout is invoked.
- K-shrink actually implements columns of the type **0-c-0** (may be infeasible according to the paper), instead of reducing the graph.
- Then, the algorithm takes the top-k customers, calculating TD and injecting them in the RMP iff they have negative reduced cost. This is a faster and simpler version of k-shrink.
- Relax-B and Relax-S are similar, but when exact pricing is invoked, max_labels is restricted to **3000** and **max_neg_routes=10** .

## Cutting Planes

- **Subset Row Inequalities (SRC)** are applied with k = 2 on small set of customers S, |S| = 3... $S_{max}$ .
- A fast **heuristic separation** selects the most <u>fractional customers</u> (coverage near 0.5) and high <u>co-occurrence</u> pairs, then enumerates candidate subsets S.
- For each S, we evaluate the violation: $\sum_r \lfloor |S \cap r|/2 \rfloor x_r \leq \lfloor |S|/2 \rfloor$, and keep the most **violated** .
- Chosen cuts are added directly to <u>RMP</u> (Gurobi) using current columns, avoiding duplicates
- These cuts **strengthen bounds** , **reduce B&B nodes** , and **guide pricing** toward more useful columns.

## Computational Experiments

## Computational Experiments - Environment

- **Computer. Intel® Core™ i7** CPU; experiments run single-threaded unless stated (Gurobi Threads=1) for fair comparison.
- **OS & Python. Ubuntu 22.04 LTS** , **Python 3.10** .
- **Solver. Gurobi Optimizer 10.x** with Method=1 (dual simplex) to exploit reoptimization in column generation; tolerances FeasibilityTol=1e-6, OptimalityTol=1e-9.
- **Reproducibility.** Fixed random seed (0) for any randomized routines; identical solver parameters across runs; wall-clock times measured with time.monotonic().
- **Implementation.** Python code for BCP (RMP + pricing/labeling + cuts); all LPs solved by Gurobi via gurobipy.

## Experimental Setup

Instances use a **time horizon [0,T]** with **time-dependent speeds** : each arc follows one of three stepwise profiles: **high** , **normal** , or **low** that change at predefined fractions of T.

The **speed function** $\mathbf{v_p(t)}$ is **piecewise constant** over the day, capturing congestion waves and yielding time-dependent travel times.

**Battery recharging** at stations follows a **nonlinear, concave** function g^(t) approximated piecewise-linearly, with diminishing marginal charge as the battery approaches **B** (e.g., breakpoints near **0.85 B** and **0.95 B** ).

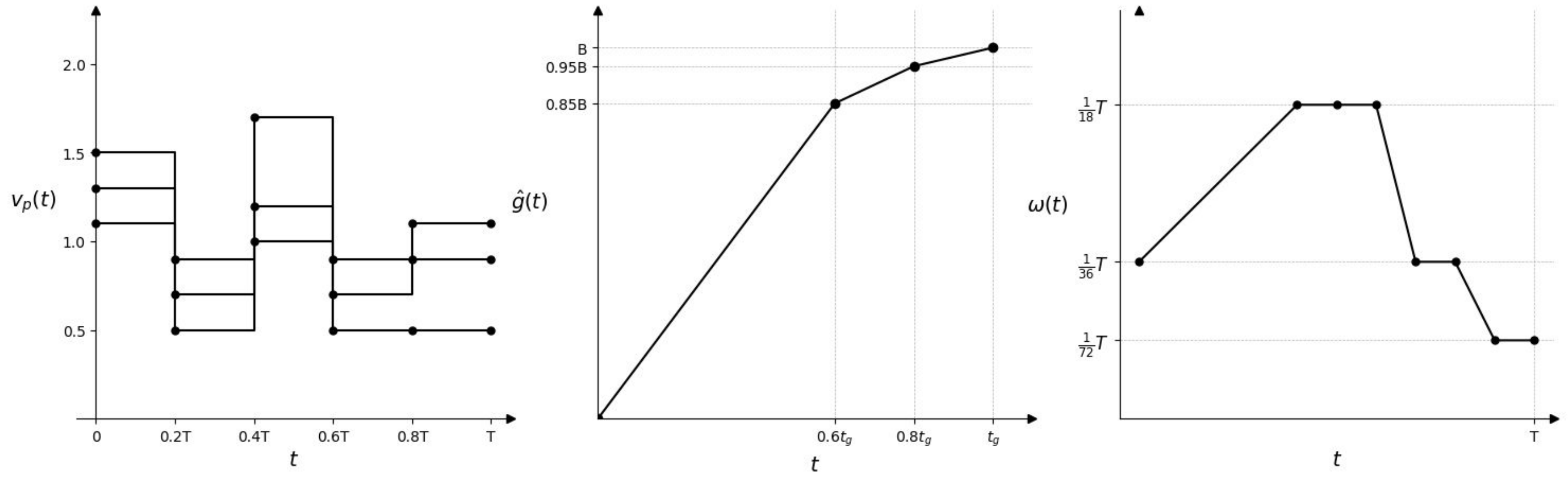
## Data Processing

Waiting times at charging stations are time-dependent and use the **TD-Smooth-Long** pattern, representing moderate, realistic congestion.

The waiting-time profile **$\omega(t)$** is scaled to the planning horizon of each instance so that peaks align with typical busy periods.

Instances of **EVRPTW** are available at: https://data.mendeley.com/datasets/h3mrm5dhxw/1

They has been manually elaborated through a python script.

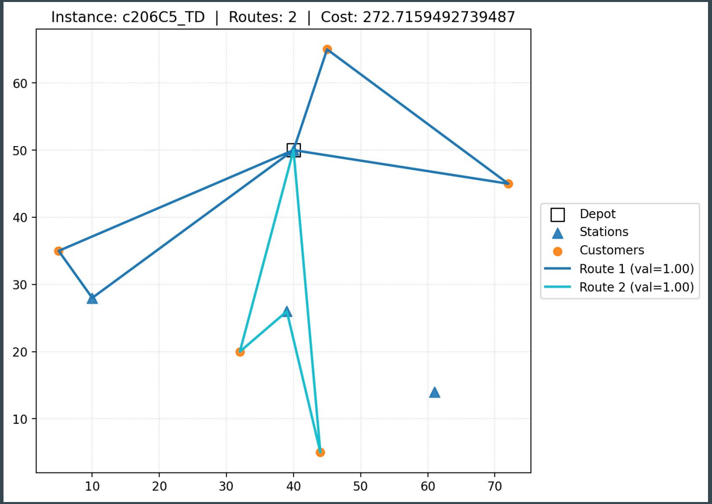


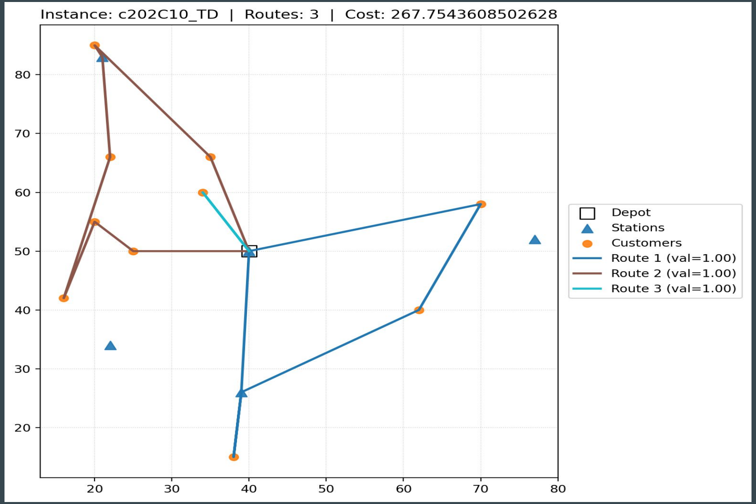
$v_p(t)$
2.0
1.5
1.0
0.5
0
0.2T
0.4T
0.6T
0.8T
T
$t$
$\hat{g}(t)$
B
0.95B
0.85B
$0.6t_g$
$0.8t_g$
$t_g$
$t$
$\omega(t)$
$\frac{1}{18}T$
$\frac{1}{36}T$
$\frac{1}{72}T$
T
$t$

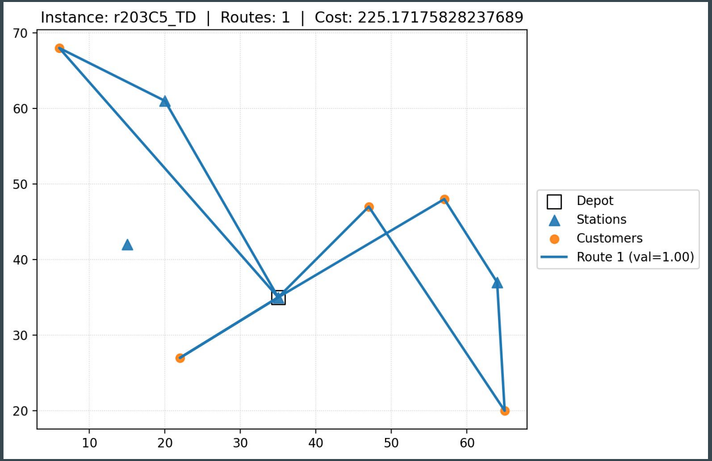
## Results

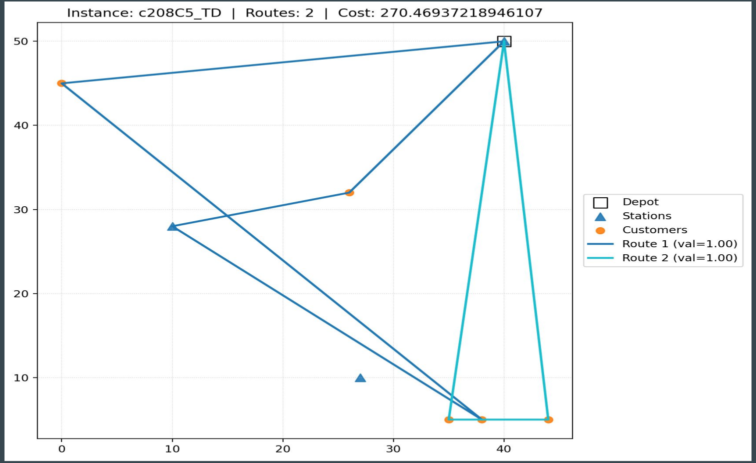
## Test on small Instances

- A small test is proposed to evaluate the correct working of the replicated algorithm related to **small instances** (5 to 10 customers).
- During the test, optimal routes has been tested for the correct requirements (battery never depleted during trips, time-windows respected, all-customer served once, and depot as final destination).
- The results are reported using 2d plots and python style lists.


Instance: c206C5_TD | Routes: 2 | Cost: 272.7159492739487
Depot
Stations
Customers
Route 1 (val=1.00)
Route 2 (val=1.00)


Instance: c202C10_TD | Routes: 3 | Cost: 267.7543608502628
Depot
Stations
Customers
Route 1 (val=1.00)
Route 2 (val=1.00)
Route 3 (val=1.00)


Instance: r203C5_TD | Routes: 1 | Cost: 225.17175828237689
Depot
Stations
Customers
Route 1 (val=1.00)


Instance: c208C5_TD | Routes: 2 | Cost: 270.46937218946107
Depot
Stations
Customers
Route 1 (val=1.00)
Route 2 (val=1.00)

## Scalability test

Scalability tests were conducted to verify the algorithm's correctness as instance size varies. To evaluate scaling up, a script incrementally adds five customers at a time to a fixed base instance while keeping battery, load, and all other parameters constant; new customer locations are generated by applying random **jitter** to their x and y coordinates.

On the other side, scalability was also assessed by **scaling down** the instance, removing five customers at a time and re-running the algorithm. This procedure tests robustness and performance across increasing and decreasing problem sizes under controlled conditions.

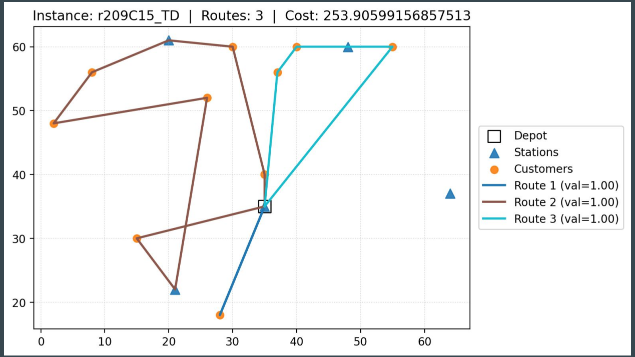
## Scalability results

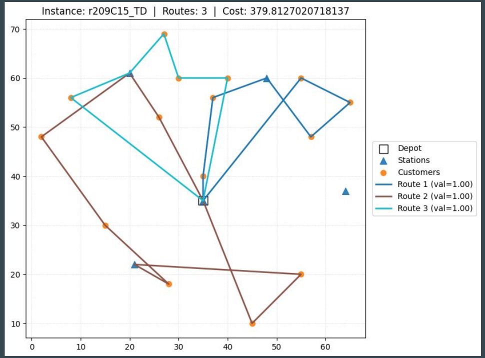
| instance name | $\lvert V_c \rvert$ | $\lvert V_s \rvert$ | cost | n. of routes found | feasibility | n. of nodes explored |
|---|---|---|---|---|---|---|
| r209C10_TD | 10 | 5 | 253.906 | 3 | True | 1 |
| r209C15_TD | 15 | 5 | 379.813 | 3 | True | 5 |
| r209C20_TD | 20 | 5 | 585.910 | 5 | True | 9 |

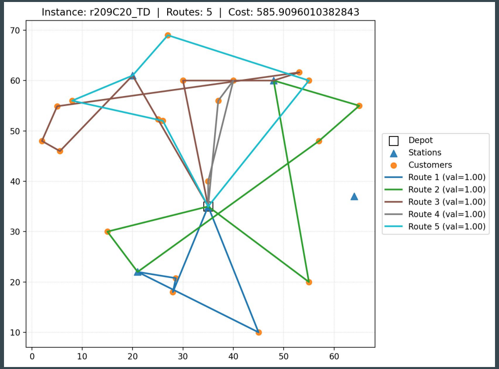
## Scalability results

- As customer count grows, total cost rises and the number of vehicles (routes) increases from 3 to 5, this is expected under TW, TD and battery constraints.
- The BCP solves the 10 customer case at the root (1 node) and needs only modest branching for 15 and 20 customers (5 and 9 nodes), which is typical when the LP relaxation and cuts are strong.
- Feasibility is consistently **True**, indicating every reported solution passes the time-dependent travel, battery, and time-window checks.
- Cost per customer stays stable then increases up at 20 customers, consistent with extra routes and added customers.

# Visualization of results







## Conclusions

## Future Implementations

The current solver omits **partial recharge decisions** and **multiple charging modes** described in the paper. Future work could extend the recharge model for a better implementation framework.

Waiting time deferral before departing a node (delayed departure) is limited by preprocessed time windows rather than fully exploiting the trade-off between speed and battery consumption discussed in the paper.

## Conclusions

The current branch-cut-and-price framework successfully implements a robust and modular solution for the Time-Dependent Electric Vehicle Routing Problem with Time Windows (TDERPVTW), following the main ideas of the reference paper while remaining computationally efficient.

## References

1. Lera-Romero, G., Miranda-Bront, J. J., & Soulignac, F. J. (2023). *A branch-cut-and-price algorithm for the time-dependent electric vehicle routing problem with time windows*. **European Journal of Operational Research** , 315(1), 72–90. https://www.sciencedirect.com/science/article/abs/pii/S037722172300509X

2. Gurobi Optimization, LLC. (2025). *Gurobi Optimizer: gurobipy Python API*. Retrieved September 14, 2025, from https://www.gurobi.com/

3. Schneider, M., Stenger, A., & Goeke, D. (2014). *Electric vehicle routing problem with time windows (E-VRPTW) instances*. Mendeley Data, V1. https://data.mendeley.com/datasets/h3mrm5dhxw/1

4. Castelli, M. (n.d.). *Mathematical Optimization – Exam Rules*. University of Trieste. Retrieved September 14, 2025, from https://sites.units.it/castelli/didattica/?file=mathopt.html

Notes: AI tools has been used to refactor code, debugging and fixing errors while executing code

Thanks for listening!