Highlights

Branch-and-Cut-and-Price for the Electric Vehicle Routing Problem with Time Windows, Piecewise-Linear Recharging and Capacitated Recharging Stations

Edward Lam, Guy Desaulniers, Peter J. Stuckey

- This paper considers the routing of electric vehicles and their recharging en route.
- Recharging the batteries of the vehicles is governed by a piecewise-linear function.
- Vehicles must recharge at stations with a limited number of chargers.
- Hence, the vehicles must be scheduled around the availability of a charger.
- The problem is tackled using a branch-and-cut-and-price algorithm.
- The algorithm decomposes the routing to integer programming and the scheduling to constraint programming.
- Results show that this hybrid algorithm solves instances with up to 100 customers.

Branch-and-Cut-and-Price for the Electric Vehicle Routing Problem with Time Windows, Piecewise-Linear Recharging and Capacitated Recharging Stations

Edward Lam[a], Guy Desaulniers[b], Peter J. Stuckey[a]

[a]*Monash University, Melbourne, Victoria, Australia*
[b]*Polytechnique Montreal and GERAD, Montreal, Quebec, Canada*

Abstract

The Electric Vehicle Routing Problem with Time Windows, Piecewise-Linear Recharging and Capacitated Recharging Stations aims to design minimum-cost routes for a fleet of electric vehicles subject to intra-route and inter-route constraints. Every vehicle is equipped with a rechargeable battery that depletes while it transports goods along its route. A vehicle must detour to a recharging station to recharge before draining its battery. To approximate a real recharging process, the amount of energy restored is modeled as a piecewise-linear function of the time spent recharging. Furthermore, each station has a small number of chargers, and hence, when and where a vehicle can recharge must be scheduled around the availability of a charger. This interaction between vehicles does not appear in classical vehicle routing problems and motivates the development of new methods that can exploit the joint routing and scheduling structure. This paper proposes a branch-and-cut-and-price algorithm that designates the routing to integer programming using Dantzig-Wolfe decomposition and the scheduling to constraint programming using logic-based Benders decomposition. Experimental results indicate that this hybrid method solves 34% of the instances with 100 customers.

Keywords:
vehicle routing problem, synchronization, scheduling, Dantzig-Wolfe decomposition, logic-based Benders decomposition, conflict-driven clause learning

1. Introduction

This paper proposes the Electric Vehicle Routing Problem with Time Windows, Piecewise-Linear Recharging and Capacitated Recharging Stations (EVRPTW-PLR-CRS). The EVRPTW-PLR-CRS adds two complications to the Electric Vehicle Routing Problem with Time Windows (EVRPTW) which is itself based on the classical Vehicle Routing Problem with Time Windows (VRPTW).

The VRPTW appoints a fleet of identical vehicles to deliver packages from a central depot to customers before returning to the depot at the end of their routes (see Vigo and Toth,

Email addresses: edward.lam@monash.edu (Edward Lam), guy.desaulniers@gerad.ca (Guy Desaulniers), peter.stuckey@monash.edu (Peter J. Stuckey)
URL: https://ed-lam.com (Edward Lam), https://www.gerad.ca/~guyd/gdeng.html (Guy Desaulniers), https://people.eng.unimelb.edu.au/pstuckey (Peter J. Stuckey)

2014; Costa et al., 2019). Every package is associated with a quantity of *load*. Each vehicle can carry packages up to a maximum total load called the *vehicle capacity*. The customer of every package is given a time frame, called the *time window*, within which the package will be delivered. Vehicles can arrive at a customer prior to the time window but must wait until the window opens before commencing delivery. Performing a delivery requires an amount of time called its *service duration*. The goal of the VRPTW is to compute routes that deliver every package while minimizing the total travel distance.

The EVRPTW extends the VRPTW to electric vehicles, each equipped with a rechargeable battery (Desaulniers et al., 2016). The battery of a vehicle is initially fully charged at the depot and depletes while the vehicle travels along its route. The battery must maintain a minimum amount of energy, denoted as 0, at all times. Vehicles must detour to a recharging station to recharge before draining their battery. The recharge rate is assumed to be constant. Therefore, the energy replenished is a linear function of the time spent recharging.

In practice, the recharge rate progressively slows to avoid damaging the battery. It can be accurately modeled as a piecewise-constant function (Montoya et al., 2017), which leads to a piecewise-linear function for representing the energy restored. The EVRPTW-PLR-CRS includes this piecewise-linear recharging function absent in the EVRPTW.

Much of the literature on routing electric vehicles assumes that the recharging stations can simultaneously recharge an unlimited number of vehicles. In practice, the stations are often equipped with only a few chargers. Hence, a vehicle may arrive at a recharging station to find that all chargers are already in use by other vehicles and, consequently, has to wait until a charger becomes available. The issue of waiting is exacerbated by the slow recharging of batteries today. The EVRPTW-PLR-CRS supplements the routing structure of the EVRPTW with a novel scheduling structure at each recharging station that both determines when a vehicle should recharge and bounds the total number of vehicles simultaneously recharging. This type of scheduling is seen at privately-owned recharging facilities where the operator has full control of the chargers. Appendix A presents an example that illustrates the combined routing and scheduling.

The scheduling structure in the EVRPTW-PLR-CRS substantially complicates the VRPTW. Every vehicle in the VRPTW can be considered independent: as long as the packages delivered along a route fit within the vehicle capacity and can be delivered on-time, delaying the route does not impact the routes of the other vehicles. In the EVRPTW, adding the energy constraints retains the independence of the vehicles: as long as a vehicle does not completely drain its battery, the routes of the other vehicles remain unaffected. However, the recharging station scheduling of the EVRPTW-PLR-CRS makes the problem much more difficult. The vehicles are no longer independent as they interact at the charging stations. To deliver the on-board packages on time, the vehicles are required to self-organize at the stations to determine the order in which the vehicles recharge.

The main contributions of this paper are an exponential-size formulation of the EVRPTW-PLR-CRS and an accompanying exact optimization algorithm that elegantly decomposes the joint routing and scheduling structure to the best technologies available for exploiting

these structures. It is well-known that integer programming and particularly branch-and-cut-and-price (BCP) power the state-of-the-art exact methods for many vehicle routing problems including the VRPTW (see Vigo and Toth, 2014; Costa et al., 2019) and the EVRPTW (Desaulniers et al., 2020). It is also well-known that constraint programming generally outperforms integer programming at scheduling problems (Schutt et al., 2010; Heinz, 2018). This paper introduces a hybrid BCP algorithm that uses integer programming for routing via a Dantzig-Wolfe decomposition (see Lübbecke and Desrosiers, 2005; Desaulniers et al., 2005) and defers the scheduling to a constraint programming subproblem using the generic form (Lam and Van Hentenryck, 2017; Davies et al., 2017; Lam et al., 2020) of logic-based Benders decomposition (Hooker and Ottosson, 2003). The paper also presents an ad-hoc technique to strengthen the Benders cuts and a polyhedral analysis to further lift the Benders cuts. Experimental results indicate that the BCP algorithm solves 34% of the instances with 100 customers.

The remainder of the paper is organized as follows. Section 2 reviews related problems and solution methods. Section 3 presents a mathematical formulation of the EVRPTW-PLR-CRS and describes the hybrid BCP algorithm. Section 4 compares the empirical performance of the algorithm on different variants of the problem. Section 5 concludes this paper.

2. Background and Related Work

This section reviews background material and surveys relevant studies.

2.1. Constraint Programming and Job Scheduling Problems

Constraint programming is a technology developed in the field of artificial intelligence for combinatorial optimization. Constraint programming is widely used across many application domains but it is especially successful at scheduling, at which it generally outperforms integer programming. Most notably, Schutt et al. (2010) close many benchmark instances of the Resource-Constrained Project Scheduling Problem using constraint programming. Integer programming and constraint programming hybrids are only starting to challenge standalone constraint programming on a limited number of benchmark instances (Heinz, 2018). The poor performance of integer programming arises from the disjunctive nature of job scheduling problems since the jobs are often scheduled before or after another.

High-performance constraint programming models often use *global constraints*, which are high-level declarations that encapsulate an entire combinatorial structure and are implemented by specialized algorithms. The DiffN global constraint is one of several workhorses for scheduling. It enforces rectangle packings and is used for scheduling jobs that cannot be paused and resumed (i.e., no preemption) and that cannot switch between different machines once started. Consider $N \in \mathbb{N}$ jobs and let $\mathcal{N} = \{1, \ldots, N\}$ be the set of jobs. Given the vectors $\mathbf{s}, \mathbf{d}, \mathbf{m}, \mathbf{r} \in \mathbb{Z}_+^N$ of variables respectively representing the start time of each job, the duration of each job, the index of the first machine assigned to each job and the number of machines simultaneously required by each job, all of which are bounded, the $\textsc{DiffN}(\mathbf{s}, \mathbf{d}, \mathbf{m}, \mathbf{r})$

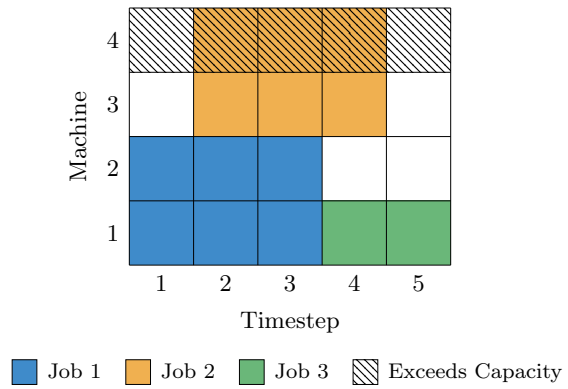

Figure 1: The DIFFN global constraint is used to pack smaller rectangles into a larger rectangle. The constraint is unsatisfiable because jobs 1 and 2 require four machines simultaneously.

constraint ensures that the rectangles with corners (s_i, m_i), $(s_i + d_i - 1, m_i)$, $(s_i, m_i + r_i - 1)$ and $(s_i + d_i - 1, m_i + r_i - 1)$, $i \in \mathcal{N}$, do not overlap. Formally, it requires

$$d_i = 0 \vee d_j = 0 \vee r_i = 0 \vee r_j = 0 \vee s_i + d_i \leq s_j \vee s_j + d_j \leq s_i \vee m_i + r_i \leq m_j \vee m_j + r_j \leq m_i$$
$$\forall i \in \{1, \ldots, n-1\}, j \in \{i+1, \ldots, n\}.$$

In the general case, $\mathbf{s}$, $\mathbf{d}$, $\mathbf{m}$ and $\mathbf{r}$ are vectors of variables. However, in many scheduling problems, the job durations $\mathbf{d}$ and resource consumptions $\mathbf{r}$ are known and fixed. Then, the DIFFN constraint only needs to find feasible start times $\mathbf{s}$ and machine allocations $\mathbf{m}$, given the constants $\mathbf{d}$ and $\mathbf{r}$. Figure 1 shows an example with three machines and three such jobs. Notice that job 2 cannot be scheduled on the three machines as it would exceed the capacity.

2.2. Logic-based Benders Decomposition

Hooker and Ottosson (2003) devise Logic-based Benders decomposition which generalizes the linear programming subproblem in the common Benders decomposition to any type of subproblem for which an *inference dual* can be defined. This includes discrete problems such as integer programming and constraint programming. Due to its generality, the Benders cuts in logic-based Benders decomposition do not have a specific form, such as the one dictated by the dual variables in standard Benders decomposition. Instead, the form of the Benders cuts must be carefully derived for every different problem. For many problems, naive combinatorial Benders cuts (i.e., cuts that allow at most $n - 1$ binary variables to be used given a set of n variables) are applicable (Codato and Fischetti, 2006) but results in an extremely weak linear relaxation.

Lam and Van Hentenryck (2017) and Davies et al. (2017) independently invent a generic mechanism that can derive combinatorial Benders cuts stronger than the naive cuts. Using the proof system borrowed from Boolean satisfiability known as *conflict analysis* or *conflict-driven clause learning* (Marques Silva and Sakallah, 1996; Ohrimenko et al., 2009; Feydy and Stuckey, 2009), their system tracks the sequence of *propagations* (i.e., the changes to the domain of the variables) in the subproblem and upon detecting infeasibility, traces these propagations backward to generate a combinatorial Benders cut in the master problem. The cut concerns

a small, but not necessarily minimal, set of master problem variables that implicate the infeasibility in the subproblem. A detailed walk-through of the procedure for the VRPTW is given by Lam and Van Hentenryck (2017). The use of conflict analysis to find combinatorial Benders cuts was recently developed into an automatic decomposition solver named Nutmeg by the two teams together (Lam et al., 2020), on which this work is based.

2.3. Electric Vehicle Routing Problems

Electric vehicle routing problems and green vehicle routing problems append constraints to classical vehicle routing problems to model issues relevant to a low-carbon society.

Desaulniers et al. (2016) design BCP algorithms for four variants of the EVRPTW: with or without partial recharges and with or without a maximum of one recharge per route. They propose pricing algorithms that include sophisticated resource extension functions for connecting the current battery level to the future energy needs of a route. They also develop dominance rules that can handle the interconnected time and charge resources. Their algorithms can solve instances with 100 customers. Desaulniers et al. (2020) then improve these algorithms by using reduced cost variable fixing to disable routes containing sequences of two arcs.

Montoya et al. (2017) study an electric vehicle routing problem with piecewise-linear recharging. They examine a common recharging process called *constant current-constant voltage* and conclude that it closely follows a piecewise-linear function. They then design a mixed integer linear programming model that could solve instances with 20 customers as well as a bespoke heuristic solver. They conclude that ignoring the non-linearity of recharging leads to infeasible or overly expensive solutions. They also notice that the literature up to this point ignore the capacity of stations and propose to extend the study to capacitated stations in future work. Based on the work by Montoya et al. (2017), Froger et al. (2019) develop two improved formulations and a heuristic approach that find the best-known solutions.

Lee (2020) develop a branch-and-price algorithm for an electric vehicle routing problem that uses a network whose nodes are recharging stations and arcs correspond to segments of a route. They include the exact non-linear recharging function without resorting to approximations but conclude that the method cannot yet handle other time constraints, such as time windows, because of the interaction with the non-linear recharging function.

Green vehicle routing problems include constraints relevant to vehicles running on alternative fuels such as hydrogen vehicles and electric vehicles. Bruglieri et al. (2019) propose exact methods for a green vehicle routing problem without time windows but with capacity constraints at the refueling stations. All vehicles are assumed to refuel for a fixed amount of time to its maximum capacity upon visiting a station. They develop two integer programming models: an arc-based model and a path-based model that couples together segments of the routes. They complement the path-based model with a cutting plane algorithm that first relaxes some of the constraints and iteratively adds them if they are violated. Their results indicate that the cutting plane approach over the huge path-based model is superior by solving the hardest instances with 15 vehicles, 15 customers and one refueling station. Noticing that the practicality of this model is limited to two refueling pumps at the station,

Bruglieri et al. (2021) later improve their cutting plane technique to handle more pumps by instead modeling the number of simultaneous refueling operations. This new algorithm can solve instances with up to 30 customers and five pumps. There are a few differences to our problem setting (e.g., no time windows) but the key difference is that we do not assume that the vehicles recharge to its full capacity in a constant amount of time but rather incorporate decisions on the amount to recharge and the duration to recharge, which is complicated by the non-linear recharging rate.

Froger et al. (2021) consider a problem similar to the EVRPTW-PLR-CRS that excludes time windows but includes stations equipped with different technologies, each with a different piecewise-linear recharging rate. They propose a very elaborate path-based mixed integer linear programming model and a two-phase local search matheuristic. Using the mixed integer linear programming model, they could exactly solve instances with up to 10 customers and 2 stations, each with 1 or 2 chargers. In contrast, the heuristic finds feasible solutions to instances with up to 320 customers. They conclude that explicitly considering the recharging station scheduling is crucial as ignoring the scheduling can lead to infeasible solutions in the presence of a limited number of chargers.

To simulate a limited capacity at the charging stations, several other studies (e.g., Kullman et al., 2021; Keskin et al., 2021, 2019) propose models and solution methods that include waiting queues at charging stations. There appears to be no other literature that develop exact solution methods to the EVRPTW-PLR-CRS, presumably due to the difficulty in handling the joint routing and scheduling structure.

2.4. Vehicle Routing Problems with Synchronization

The vehicles in classical vehicle routing problems can be considered independent as delaying a route has little impact on the other routes. However, in rich vehicle routing problems, modifying a route can have severe and detrimental effects on other routes. For instance, some problems require the vehicles to co-operate to rendezvous or share resources. This interaction is called *synchronization*. The EVRPTW-PLR-CRS falls into this category. Vehicles are interdependent due to the interaction at the recharging stations. Vehicle routing problems with synchronization are extremely difficult due to the consequences of one decision on another. Three studies with similar routing and scheduling structure are mentioned below. Drexl (2012) survey other vehicle routing problems with synchronization.

El Hachemi et al. (2011) investigate a joint routing and scheduling problem for scheduling vehicles in the timber industry. They develop an ad-hoc decomposition technique somewhat similar to logic-based Benders decomposition. They acknowledge that integer programming is inadequate at scheduling and propose a constraint programming master problem for scheduling and an integer programming subproblem for routing. Information on route infeasibility is communicated back to the constraint programming master problem using global constraints.

Lam and Van Hentenryck (2016) study a routing and scheduling problem that models airplanes parking in a limited number of bays while loading or unloading goods. The problem schedules the arrival and departure of the airplanes in consideration of the scarce parking bays. They develop a path-based integer programming master problem solved using column

generation for routing the vehicles, which are then scheduled using constraint programming. They enforce the scheduling constraints using naive combinatorial Benders cuts, which simply state that at most $n-1$ of n binary decisions can be chosen. They do not attempt to lift or strengthen these cuts.

Jungwirth et al. (2020) consider a scheduling problem in the healthcare domain. The problem is modeled as a vehicle routing problem with scheduling constraints. They develop a column generation algorithm for the problem and handle the scheduling structure using two methods: one based on branching and the other based on naive combinatorial Benders cuts. They also do not attempt to lift these cuts and conclude that branching performs better experimentally.

3. The Branch-and-Cut-and-Price Algorithm

This section presents an exponential-size model of the EVRPTW-PLR-CRS and an accompanying BCP algorithm for solving it.

3.1. Simplifying Assumptions

The following assumptions are imposed to simplify the problem.

Electric vehicles are not yet used for long-distance deliveries that require more than one recharge between two customers. Therefore, a route is prevented from containing a subsequence of two or more consecutive recharging stations. That is, a vehicle must arrive at a station from the depot or a customer and depart the station to a customer or the depot.

A vehicle is assumed to conduct exactly one recharge between the time that it arrives and departs a station. This prevents a vehicle from sharing its charger with another vehicle by pausing its recharge. In the terminology of scheduling, preemption is prohibited.

Every vehicle begins its day fully charged. In addition, vehicles can return to the depot with zero energy remaining as there is ample time overnight to recharge before beginning the next day. With a further assumption that energy costs are proportional to the recharge amount, then energy costs can be ignored since the costs are incurred either along a route or after returning to the depot.

The energy consumed while in motion is assumed to be proportional to the distance traveled. In reality, energy consumption is a non-linear function of the speed of the vehicle, the load on-board and the slope of the road, among others.

Montoya et al. (2017) analyze the recharging characteristics of real batteries and propose to model the recharging as a piecewise-linear function. They acknowledge that the recharging process is described by differential equations but argue that piecewise-linear functions are an accurate approximation. The same three-piece piecewise-linear function, as shown in Figure 2, is considered for the EVRPTW-PLR-CRS. Note that a battery level of 0 does not mean that the battery is fully depleted but rather represents a minimal state of charge.

3.2. The Model

Consider $K \in \mathbb{N}$ available vehicles, where K can be set to an arbitrarily large value if the number of vehicles is unlimited. Let $\mathcal{R}^*$ be the set of all possible routes. Let $\mathcal{C}$ be the set of

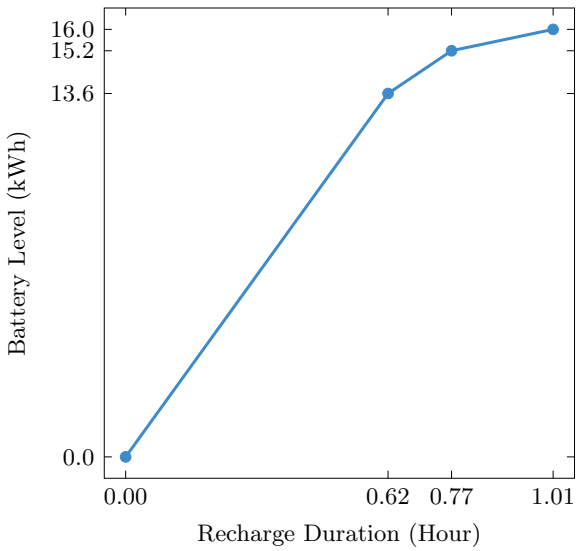

Figure 2: The piecewise-linear function governing the amount of energy restored as a function of the duration spent recharging according to Montoya et al. (2017).

customers and let $A_r^c \in \mathbb{Z}_+$ be a constant representing the number of times that route $r \in \mathcal{R}^*$ visits customer $c \in \mathcal{C}$. Associate a cost $t_r^{\text{cost}} \in \mathbb{Z}_+$ with every route $r \in \mathcal{R}^*$ and a variable $x_r \in \{0, 1\}$ that indicates whether r is chosen. A formulation of the EVRPTW-PLR-CRS is then given as follows:

$$\min \sum_{r \in \mathcal{R}^*} t_r^{\text{cost}} x_r \tag{1a}$$

subject to

$$\sum_{r \in \mathcal{R}^*} A_r^c x_r = 1 \qquad \forall c \in \mathcal{C}, \tag{1b}$$

$$\sum_{r \in \mathcal{R}^*} B_b^r x_r \geq B_b \qquad \forall b \in \mathcal{B}^*, \tag{1c}$$

$$\sum_{r \in \mathcal{R}^*} x_r \leq K, \tag{1d}$$

$$x_r \in \mathbb{Z}_+ \qquad \forall r \in \mathcal{R}^*. \tag{1e}$$

Objective Function (1a) minimizes the total cost of all chosen routes. Constraints (1b) ensure that every customer is visited exactly once. Constraints (1c) indirectly enforce the capacity of the recharging stations by requiring combinations of routes that have a feasible recharging schedule. The set $\mathcal{B}^*$ is the Benders cuts and every Benders cut $b \in \mathcal{B}^*$ is associated with a coefficient $B_b^r \in \mathbb{Z}$ for every route $r \in \mathcal{R}^*$ and a constant $B_b \in \mathbb{Z}$. Constraint (1d) bounds the number of routes. Constraints (1e) define the domain of the variables. Together, Constraints (1b) and (1e) ensure that $0 \leq x_r \leq 1$ for all $r \in \mathcal{R}^*$.

As $\mathcal{R}^*$ and $\mathcal{B}^*$ are exponential in the number of vertices, the model is simply too large to be declared upfront and solved in a static manner using an integer programming solver.

Instead, the variables and constraints are dynamically constructed using a BCP algorithm. In the context of this algorithm, the above formulation is referred to as the *integer master problem*.

3.3. The Algorithm

Algorithm 1 shows a generic BCP algorithm for solving the integer master problem. The algorithm begins by creating an initial incumbent solution (Line 1) and a priority queue (Line 2) to store the open nodes of the branch-and-bound tree. It then adds the root node to the priority queue (Line 3). Then, for each node in the priority queue (Lines 4 and 5), it performs column generation (Lines 7 to 21) to solve the linear relaxation of the integer master problem. In each iteration of column generation, it solves the restricted master problem (RMP) (Line 11) and the pricing subproblem (Line 19), and may call the checking subproblem (Line 13). The RMP is a linear program that, given a set of routes, determines the proportion that each route is selected in a candidate solution. If this solution is integral and can become the incumbent (Line 12, where *routes.obj* and *incumbent_routes.obj* denote the cost of this solution and that of the best solution found so far, respectively), the algorithm proceeds to the checking subproblem (Line 13), which handles the inter-route constraints, namely the synchronization between the vehicles at the recharging stations. The checking subproblem ensures that the routes chosen by the RMP have corresponding schedules that respect the capacity of every station. If so (Line 14), the routes from the RMP and the schedules from the checking subproblem constitute a feasible integer solution to the EVRPTW-PLR-CRS (Line 15). If not (Line 16), the checking subproblem generates a combinatorial Benders cut that is added to the RMP (Line 17), forcing the RMP to choose a different set of routes (Lines 9 and 18). Once cuts are not found, the algorithm proceeds to the pricing subproblem (Line 19), which attempts to find better routes for inclusion in the RMP (Line 21). The pricing subproblem manages the intra-route constraints, i.e., the load, time window and piecewise-linear recharging constraints. When the pricing subproblem declares that better routes are not available, the column generation process stops and the algorithm proceeds to branching if the RMP solution is fractional and potentially better than the incumbent (Lines 22 to 25).

The RMP is initially infeasible as it is initialized with no routes. Farkas pricing (e.g., Andersen, 2014) is used to bring the RMP to primal feasibility, both initially at the root node and after branching. The RMP is also augmented with valid inequalities to improve performance.

3.4. The Restricted Master Problem

The RMP is the linear relaxation of the integer master problem and replaces the huge sets of routes $\mathcal{R}^*$ and Benders cuts $\mathcal{B}^*$ with smaller but still exponential-size subsets $\mathcal{R} \subseteq \mathcal{R}^*$ and $\mathcal{B} \subseteq \mathcal{B}^*$, which are initially empty and are progressively filled by the pricing subproblem and checking subproblem, respectively. The RMP is shown below:

$$\min \sum_{r \in \mathcal{R}} t_r^{\text{cost}} x_r \tag{2a}$$

Algorithm 1 The branch-and-cut-and-price algorithm.

```
1: (incumbent_routes, incumbent_schedules) ← CreateDummySolution()
2: open ← CreatePriorityQueue()
3: open.Add(CreateRootNode())
4: while ¬open.IsEmpty() do
5:     node ← open.Pop()
6:     priced ← true
7:     while priced do
8:         separated ← true
9:         while separated do
10:            separated ← false
11:            (routes, duals) ← SolveRestrictedMasterProblem(node.master)
12:            if routes.IsInteger() ∧ routes.obj < incumbent_routes.obj then
13:                (schedules, cut) ← SolveCheckingSubproblem(node.master, routes)
14:                if schedules.IsFeasible() then
15:                    (incumbent_routes, incumbent_schedules) ← (routes, schedules)
16:                else
17:                    node.master.AddCut(cut)
18:                    separated ← true
19:        (priced, columns) ← SolvePricingSubproblem(node.master, duals)
20:        if priced then
21:            node.master.AddColumns(columns)
22:    if routes.IsFractional() ∧ routes.obj < incumbent_routes.obj then
23:        (left_child, right_child) ← CreateChildren(node.master, routes)
24:        open.Add(left_child)
25:        open.Add(right_child)
```

subject to

$$\sum_{r \in \mathcal{R}} A_r^c x_r = 1 \qquad \forall c \in \mathcal{C}, \quad (2b)$$

$$\sum_{r \in \mathcal{R}} B_b^r x_r \geq B_b \qquad \forall b \in \mathcal{B}, \quad (2c)$$

$$\sum_{r \in \mathcal{R}} x_r \leq K, \qquad (2d)$$

$$x_r \geq 0 \qquad \forall r \in \mathcal{R}. \quad (2e)$$

Define $\alpha_c, \beta_b, \gamma$ as the dual variables of Constraints (2b) to (2d), respectively.

3.5. The Pricing Subproblem

The pricing subproblem handles the routing of a single vehicle and has no direct knowledge about the interactions at the recharging stations. It optimistically assumes that vehicles can recharge whenever necessary, i.e., that the stations have no capacity. It is the dual variables β_b associated with the generated Benders cuts (Constraints (2c)) that indirectly influence the choice of the arcs in the routes generated by the pricing subproblem.

Every vehicle is associated with a vehicle capacity $T^{\text{load}} \in \mathbb{Z}_+$. Let $T^{\text{time}} \in \mathbb{Z}_+$ be the time horizon, at which point all vehicles must have returned to the depot. Every vehicle is equipped with a battery storing an amount of energy between zero (representing a minimum amount) and a maximum amount $T^{\text{energy}} \in \mathbb{Z}_+$. The battery is recharged according to a piecewise-linear function with $P \in \mathbb{N}$ pieces. Let $\mathcal{P} = \{1, \ldots, P\}$ be the set of pieces. Each piece $p \in \mathcal{P}$ begins at a battery level $\sigma_{p-1} \in \mathbb{Z}_+$ and ends at $\sigma_p \in \mathbb{Z}_+$, where $0 = \sigma_0 \leq \sigma_1 \leq \ldots \leq \sigma_P = T^{\text{energy}}$. Define $\theta_p \in \mathbb{Q}$ as the recharging rate (energy per timestep) within piece $p \in \mathcal{P}$ such that $\theta_1 \geq \theta_2 \geq \ldots \geq \theta_P > 0$ and $1/\theta_p \in \mathbb{N}$. To accommodate integrality limitations of constraint programming, as explained in Section 3.6, $1/\theta_p$ is required to be integer.

To implicitly model all feasible routes in the pricing subproblem, consider a directed graph $\mathcal{G}_{\text{PS}} = (\mathcal{V}, \mathcal{A}_{\text{PS}})$, with vertex set $\mathcal{V}$ and arc set $\mathcal{A}_{\text{PS}}$, that is customized for handling the duals of Constraints (2c). The vertex set $\mathcal{V} = \mathcal{C} \cup \mathcal{S}' \cup \mathcal{D}^+ \cup \mathcal{D}^-$ is the union of the customer set $\mathcal{C}$, a set $\mathcal{S}'$ of station vertices expanded from an original set $\mathcal{S}$ of stations, a set $\mathcal{D}^+$ of source vertices corresponding to the depot at which the vehicles begin their routes and a set $\mathcal{D}^-$ of sink vertices corresponding to the depot at which the vehicles end their routes. Note that we use sets $\mathcal{D}^+$ and $\mathcal{D}^-$ to denote the depot vertices to be consistent with the notation used in the next section, even if they each contain a single vertex.

Instantiate $\mathcal{C} = \{0, \ldots, C-1\}$ and $\mathcal{S} = \{0, \ldots, S-1\}$ where $C \in \mathbb{N}$ is the number of customers and $S \in \mathbb{N}$ is the number of stations. Each station $s \in \mathcal{S}$ has $M_s \in \mathbb{N}$ chargers. Let the set of expanded station vertices $\mathcal{S}' = \mathcal{S}^+ \cup \mathcal{S}^-$ where the post-customer station vertices in $\mathcal{S}^+ = \{C, \ldots, C+C \cdot S-1\}$ are the station vertices accessible only immediately after a customer and the pre-customer station vertices in $\mathcal{S}^- = \{C + C \cdot S, \ldots, C + 2C \cdot S - 1\}$ are the station vertices accessible only immediately before a customer, but as we will see later, accessible only immediately before any customer is visited. The vertex $i = C + C \cdot s + c \in \mathcal{S}^+$ is associated with station $s \in \mathcal{S}$ and is only accessible immediately after visiting customer $c \in \mathcal{C}$, i.e., i has

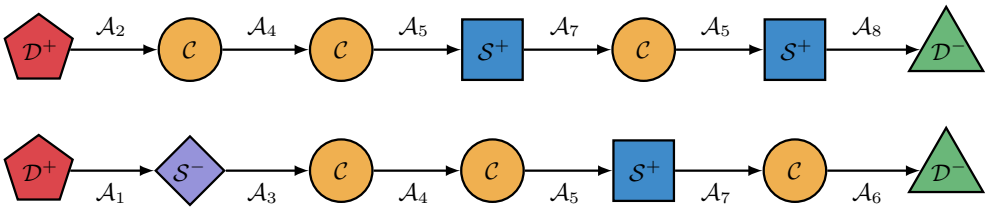

Figure 3: Two routes illustrating the network $\mathcal{G}_{\text{PS}}$. Each node and arc is labeled with the set that it is contained in.

only one incoming arc (c, i). Similarly, the set $\mathcal{S}^-$ clones the stations for visits only before one particular customer. The vertex $i = C + C \cdot S + C \cdot s + c \in \mathcal{S}^-$ is associated with station $s \in \mathcal{S}$ and is only accessible immediately before visiting customer $c \in \mathcal{C}$, i.e., i has only one outgoing arc (i, c). Also let $\mathcal{S}'_s = \{C + C \cdot s + c : c \in \mathcal{C}\} \cup \{C + C \cdot S + C \cdot s + c : c \in \mathcal{C}\} \subset \mathcal{S}'$ be the set of station vertices expanded from the original station $s \in \mathcal{S}$. Only the expanded station vertices appear within the graph; the original stations never appear as vertices. Denote the vertices in the singleton depot vertex sets as $\mathcal{D}^+ = \{C + 2C \cdot S\}$ and $\mathcal{D}^- = \{C + 2C \cdot S + 1\}$. Even though this expansion leads to a large number of vertices, the arc set is highly constrained, as explained below.

A route $r = (r_1, r_2, \ldots, r_n)$ of length $n \in \mathbb{N}$ is represented in $\mathcal{G}_{\text{PS}}$ as a sequence of n vertices beginning at a source vertex (i.e., $r_1 \in \mathcal{D}^+$), through a number of unique customer and station vertices (i.e., $r_2, \ldots, r_{n-1} \in \mathcal{C} \cup \mathcal{S}'$ and $r_i \neq r_j$ for $i \in \{2, \ldots, n-2\}$ and $j \in \{i+1, \ldots, n-1\}$) and then ending at a sink vertex (i.e., $r_n \in \mathcal{D}^-$) such that $(r_i, r_{i+1}) \in \mathcal{A}_{\text{PS}}$ for all $i \in \{1, \ldots, n-1\}$. Route r is said to *visit* the vertices $r_1, \ldots, r_n$ and *traverse* the arcs $(r_1, r_2), \ldots, (r_{n-1}, r_n)$.

We introduce more notation before describing the arc set $\mathcal{A}_{\text{PS}}$. Every vertex $i \in \mathcal{V}$ is associated with a load $t_i^{\text{load}} \in \{0, \ldots, T^{\text{load}}\}$, time window opening $t_i^{\text{open}} \in \{0, \ldots, T^{\text{time}}\}$ and time window closing $t_i^{\text{close}} \in \{0, \ldots, T^{\text{time}}\}$, where $t_i^{\text{open}} \leq t_i^{\text{close}}$. For every non-customer vertex $i \in \mathcal{V} \setminus \mathcal{C}$, assume that the load $t_i^{\text{load}} = 0$, the time window opening $t_i^{\text{open}} = 0$ and the time window closing $t_i^{\text{close}} = T^{\text{time}}$.

For every pair of vertices $i, j \in \mathcal{V}$, let $t_{i,j}^{\text{cost}} \in \mathbb{Z}_+$ be the distance from i to j, $t_{i,j}^{\text{travel}} \in \{0, \ldots, T^{\text{time}}\}$ be the travel time from i to j, including the service time at i if any, and $t_{i,j}^{\text{energy}} \in \{0, \ldots, T^{\text{energy}}\}$ be the amount of energy discharged while traveling from i to j. For a route $r = (r_1, \ldots, r_n)$, associate with it a cost $t_r^{\text{cost}} = \sum_{i=1}^{n-1} t_{r_i, r_{i+1}}^{\text{cost}}$.

Recall from Section 3.1 the assumption that a route cannot contain a subsequence of two or more consecutive stations. This limitation is enforced by requiring all arcs incoming to $\mathcal{S}^-$ to be outgoing from a source vertex. That is, if a pre-customer station vertex $j \in \mathcal{S}^-$ appears in a route, it must be the first vertex immediately after a source vertex (i.e., $r_3, \ldots, r_n \notin \mathcal{S}^-$). Hence, a vehicle cannot visit a post-customer station vertex $i \in \mathcal{S}^+$ associated with a customer $c_1 \in \mathcal{C}$ and followed immediately by a pre-customer station vertex $j \in \mathcal{S}^-$ associated with a different customer $c_2 \in \mathcal{C}$, $c_2 \neq c_1$.

Figure 3 summarizes the network with two routes. All routes begin at a source vertex ($\mathcal{D}^+$). Arcs outgoing from a source vertex ($\mathcal{D}^+$) only lead to a customer ($\mathcal{C}$) or a pre-customer station vertex ($\mathcal{S}^-$). Arcs outgoing from a customer ($\mathcal{C}$) lead to other customers ($\mathcal{C}$), post-

customer stations ($\mathcal{S}^+$) or sink vertices ($\mathcal{D}^-$). Pre-customer stations ($\mathcal{S}^-$) have exactly one arc leading to their associated customer ($\mathcal{C}$). Arcs outgoing from the post-customer stations ($\mathcal{S}^+$) lead to customers ($\mathcal{C}$) or the sink vertices ($\mathcal{D}^-$).

The arc set $\mathcal{A}_{\text{PS}} = (\mathcal{A}_1 \cup \ldots \cup \mathcal{A}_8) \setminus (\mathcal{A}_9 \cup \ldots \cup \mathcal{A}_{14})$ is constructed as follows. First, the arcs are built from the union of the following subsets:

- $\mathcal{A}_1 = \{(i,j) : i \in \mathcal{D}^+, j \in \mathcal{S}^-\}$ contains the arcs from the start depot to the pre-customer stations.
- $\mathcal{A}_2 = \{(i,j) : i \in \mathcal{D}^+, j \in \mathcal{C}\}$ contains the arcs from the start depot to the customers.
- $\mathcal{A}_3 = \{(i,j) : i \in \mathcal{S}^-, j \in \mathcal{C}, j = i \bmod C\}$ contains the arcs from each pre-customer station to its corresponding customer.
- $\mathcal{A}_4 = \{(i,j) : i,j \in \mathcal{C}\}$ contains the arcs between customers.
- $\mathcal{A}_5 = \{(i,j) : i \in \mathcal{C}, j \in \mathcal{S}^+, i = j \bmod C\}$ contains the arcs from each customer to its corresponding post-customer stations.
- $\mathcal{A}_6 = \{(i,j) : i \in \mathcal{C}, j \in \mathcal{D}^-\}$ contains the arcs from the customers to the end depot.
- $\mathcal{A}_7 = \{(i,j) : i \in \mathcal{S}^+, j \in \mathcal{C}, j \neq i \bmod C\}$ contains the arcs from the post-customer stations to other customers.
- $\mathcal{A}_8 = \{(i,j) : i \in \mathcal{S}^+, j \in \mathcal{D}^-\}$ contains the arcs from the post-customer stations to the end depot.

Next, the following subsets of infeasible or redundant arcs are removed:

- $\mathcal{A}_9 = \{(i,i) : i \in \mathcal{C}\}$ contains the loops at the customers.
- $\mathcal{A}_{10} = \{(i,j) : i,j \in \mathcal{V}, t_i^{\text{load}} + t_j^{\text{load}} > T^{\text{load}}\}$ contains the arcs where including the packages of customers i and j on the same vehicle will exceed its vehicle capacity.
- $\mathcal{A}_{11} = \{(i,j) : i,j \in \mathcal{V}, t_i^{\text{open}} + t_{i,j}^{\text{travel}} > t_j^{\text{close}}\}$ contains the arcs that exceed the time window at j given the earliest possible departure at i.
- $\mathcal{A}_{12} = \{(i,j) : i,j \in \mathcal{V}, t_{i,j}^{\text{energy}} > T^{\text{energy}}\}$ contains the arcs that discharge more than the battery capacity.
- $\mathcal{A}_{13} = \{(i,j) : i \in \mathcal{D}^+, j \in \mathcal{S}^-, t_{i,j}^{\text{cost}} = 0\}$ contains the arcs from the start depot to a station at the same location.
- $\mathcal{A}_{14} = \{(i,j) : i \in \mathcal{S}^+, j \in \mathcal{D}^-, t_{i,j}^{\text{cost}} = 0\}$ contains the arcs from a station at the same location as the depot to the end depot.

Excluding $\mathcal{A}_{13}$ and $\mathcal{A}_{14}$ is not strictly necessary but reduces some redundancy in the model because there is no benefit in recharging at the same location as the depot immediately after starting a route or immediately before ending a route. Some of these subsets are shown in Figure 3.

The pricing subproblem is an elementary resource-constrained shortest path problem on $\mathcal{G}_{\text{PS}}$. It is solved using a labeling algorithm (see Irnich and Desaulniers, 2005). The labeling algorithm is initialized with a partial path both starting and ending at a source vertex and iteratively extends the partial path forward along all arcs outgoing from the source and the subsequently visited vertices, generating more partial paths in the process. The algorithm loops through all partial paths until a stopping criterion is satisfied, such as discovering a path from the source to the sink with negative reduced cost. Any path found with negative reduced cost is then added to $\mathcal{R}$ in the RMP.

Every partial path from the source to some vertex i is associated with a *label*, a vector of the resource consumptions of the partial path, including its reduced cost. If the resource consumptions of a partial path are infeasible, the path is discarded. In addition, if a path is *dominated*, it is also discarded. Given two partial paths r_1 and r_2 from the source to a common vertex i, r_2 is dominated if every possible extension of r_2 to the sink is also a valid extension of r_1 that yields an equal or better reduced cost.

A label $l_i = (l_i^{\text{cost}}, l_i^{\text{load}}, l_i^{\text{time}}, l_i^{\text{energy}}, (l_i^{\text{mrt}_p})_{p\in\mathcal{P}}, (l_i^{\text{cust}_c})_{c\in\mathcal{C}})$ associated with a partial path ending at vertex i contains the components:

- l_i^{cost}: the reduced cost of the route up to i,
- l_i^{load}: the load delivered up to i,
- l_i^{time}: the earliest time for starting service at i,
- l_i^{energy}: the battery level at i after minimally recharging the battery along the partial path,
- $l_i^{\text{mrt}_p}$: the maximum amount of time for recharging at rate θ_p, $p \in \mathcal{P}$, at any station along the partial path that is still available when reaching i, and
- $l_i^{\text{cust}_c}$: a 0/1 resource indicating whether customer $c \in \mathcal{C}$ is unreachable (already visited or necessarily exceeds the vehicle capacity or time window) in the remainder of the partial path.

The labeling algorithm initially creates a label l_i representing the source depot vertex $i = C + 2C \cdot S$ with $l_i^{\text{cost}} = -\gamma$, $l_i^{\text{load}} = 0$, $l_i^{\text{time}} = 0$, $l_i^{\text{energy}} = T^{\text{energy}}$, $l_i^{\text{mrt}_p} = 0\,\forall p \in \mathcal{P}$, and $l_i^{\text{cust}_c} = 0\,\forall c \in \mathcal{C}$.

Given a set of existing labels, the labeling algorithm loops through every existing label l_i (associated with a partial path ending at vertex i) to extend the partial path along every outgoing arc $(i, j) \in \mathcal{A}_{\text{PS}}$. The labeling algorithm computes the resource consumptions of the new partial path ending at vertex j and stores them in a new label $l_j = (l_j^{\text{cost}}, l_j^{\text{load}}, l_j^{\text{time}}, l_j^{\text{energy}}, (l_j^{\text{mrt}_p})_{p\in\mathcal{P}}, (l_j^{\text{cust}_c})_{c\in\mathcal{C}})$. The resource consumptions of l_j are calculated using Algorithm 2.

As explained later, the constant B_b^r in Constraints (1c) and (2c) can be decomposed into a weighted sum on the arcs $B_b^r = \sum_{(i,j)\in\mathcal{A}_{\text{PS}}} B_b^{i,j} A_r^{i,j}$ where $B_b^{i,j} \in \mathbb{Z}$ is a constant associated with arc $(i, j) \in \mathcal{A}_{\text{PS}}$ and $A_r^{i,j} \in \mathbb{Z}_+$ is a constant representing the number of times that

Algorithm 2 Label extension procedure for extending a partial path ending at vertex i along an arc (i, j)

$l_j^{\text{cost}} \leftarrow l_i^{\text{cost}} + t_{i,j}^{\text{cost}} - \alpha_j - \sum_{b \in \mathcal{B}} \beta_b B_b^{i,j}$
if $j \in \mathcal{C}$ **then**
    $l_j^{\text{cust}_j} \leftarrow l_i^{\text{cust}_j} + 1$
    if $l_j^{\text{cust}_j} > 1$ **then**
        exit (extension is infeasible)
$l_j^{\text{load}} \leftarrow l_i^{\text{load}} + t_j^{\text{load}}$
$l_j^{\text{time}} \leftarrow \max\{l_i^{\text{time}} + t_{i,j}^{\text{travel}}, t_j^{\text{open}}\}$
if $l_j^{\text{load}} > T^{\text{load}} \vee l_j^{\text{time}} > t_j^{\text{close}}$ **then**
    exit (extension is infeasible)
$w \leftarrow \max\{t_j^{\text{open}} - (l_i^{\text{time}} + t_{i,j}^{\text{travel}}), 0\}$
for $p = 1, \ldots, P$ **do**
    $\delta^{\text{mrt}_p} \leftarrow \min\{w - \sum_{\mu=1}^{p-1} \delta^{\text{mrt}_\mu}, l_i^{\text{mrt}_p}\}$
    $l_j^{\text{mrt}_p} \leftarrow l_i^{\text{mrt}_p} - \delta^{\text{mrt}_p}$
$l_j^{\text{energy}} \leftarrow l_i^{\text{energy}} - t_{i,j}^{\text{energy}} + \sum_{p \in \mathcal{P}} \theta_p \delta^{\text{mrt}_p}$
if $l_j^{\text{energy}} < 0$ **then**
    $p \leftarrow 1$
    while $l_j^{\text{energy}} < 0$ and $p \leq P$ **do**
        $\delta^{\text{mrt}_p} \leftarrow \min\{l_j^{\text{mrt}_p}, -l_j^{\text{energy}}/\theta_p\}$
        $l_j^{\text{time}} \leftarrow l_j^{\text{time}} + \delta^{\text{mrt}_p}$
        $l_j^{\text{mrt}_p} \leftarrow l_j^{\text{mrt}_p} - \delta^{\text{mrt}_p}$
        $l_j^{\text{energy}} \leftarrow l_j^{\text{energy}} + \theta_p \delta^{\text{mrt}_p}$
        $p \leftarrow p + 1$
    if $l_j^{\text{energy}} < 0 \vee l_j^{\text{time}} > t_j^{\text{close}}$ **then**
        exit (extension is infeasible)
if $j \in \mathcal{S}'$ **then**
    for $p = 1, \ldots, P$ **do**
        $l_j^{\text{mrt}_p} \leftarrow \min\{(\sigma_p - \sigma_{p-1})/\theta_p, \max\{(\sigma_p - l_j^{\text{energy}})/\theta_p, 0\}\}$
for $p = 1, \ldots, P$ **do**
    $l_j^{\text{mrt}_p} \leftarrow \min\{l_j^{\text{mrt}_p}, t_j^{\text{close}} - l_j^{\text{time}} - \sum_{\mu=1}^{p-1} l_j^{\text{mrt}_\mu}\}$
for $c \in \mathcal{C}$ **do**
    if $l_j^{\text{cust}_c} = 0 \wedge (l_j^{\text{load}} + t_c^{\text{load}} > T^{\text{load}} \vee l_j^{\text{time}} + t_{j,c}^{\text{travel}} > t_c^{\text{close}})$ **then**
        $l_j^{\text{cust}_c} \leftarrow 1$

route $r \in \mathcal{R}$ traverses (i, j). Then, Line 1 updates the reduced cost of the partial path as an accumulation of reduced costs on the arcs. Assume that $\alpha_j = 0$ if $j \notin \mathcal{C}$.

If vertex j is a customer (Line 2), Line 3 increments the number of times that j is visited. If customer j is visited more than once (Line 4), Line 5 exits the algorithm. Line 6 increments the load. Line 7 computes a lower bound on the service start time l_j^{time}, which can be the arrival time but can also be delayed until the time window opens. If extending the partial path along the arc exceeds the vehicle capacity or the arrival time is later than the closure of the time window (Line 8), Line 9 exits the algorithm.

A vehicle may arrive at vertex j before the time window opens. Instead of waiting unproductively for the time window to open, the waiting time may be replaced by additional recharging time at a station previously visited along the route. This is always possible if (1) at least one station is visited before vertex j, (2) the battery is not full at one of these visits and (3) the time window of the customers visited since then are not yet closed. Lines 10 to 14 convert the waiting time into recharging time. Line 10 calculates the waiting time w at vertex j, if any. For each piece $p \in \mathcal{P}$ in order (Line 11), Line 12 computes the maximum amount of time δ^{mrt_p} that can be used to recharge at an earlier station to make the waiting time vanish, favoring the faster rates. Line 13 reduces the amount of recharging time available in piece p by the duration calculated in the previous line. Line 14 discharges the battery due to traveling on the arc and then recharges the battery using the waiting time, if any.

If the battery holds a negative amount of energy (Line 15), then the vehicle is insufficiently charged and cannot reach vertex j. Lines 15 to 24 recharge the vehicle until it holds 0 energy upon arriving at j. Recall that Lines 10 to 14 delay the vehicle until its arrival time at vertex j is the time window opening, and hence, the recharge has no consequence on the route. In contrast, Lines 15 to 24 delay the vehicle after the time window opens and may impact the time window constraints. Line 16 initializes the loop variable. Line 17 loops through each piece $p \in \mathcal{P}$, starting at the pieces representing the lower battery levels. Line 18 computes the amount of recharging time required in piece p until the battery holds 0 energy, capped at the amount of time available. Line 19 delays the vehicle by this amount of time. Line 20 reduces the available recharging time of piece p. Line 21 recharges the vehicle for this amount of time. Line 22 advances to the next piece. If the amount of additional recharging time required to reach vertex j is unavailable or performing this recharge exceeds its time window (Line 23), Line 24 exits the algorithm.

Upon reaching a station, Lines 25 to 27 update the amount of time available for recharging. The amount of time available for recharging in each piece is calculated as either the duration to recharge the full piece if the battery is in a lower piece, or the duration to reach the end of the piece if the battery is in the piece.

Lines 28 and 29 ensure that the closure of the time window at vertex j is respected in later extensions by reducing the time available to recharge in each piece.

If visiting a customer in the future exceeds the vehicle capacity of the vehicle or the time window of the customer, Lines 30 to 32 mark the customer as unreachable.

The components $l_i^{\text{time}}, l_i^{\text{energy}}, (l_i^{\text{mrt}_p})_{p \in \mathcal{P}}$ of label l_i are parameters that define the recharg-

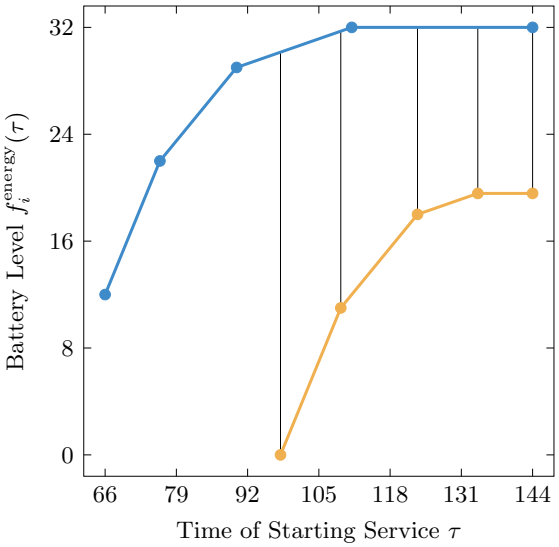

Figure 4: Two recharging functions governing the time of starting service and the battery level at arrival to a common vertex.

ing function $f_i^{\text{energy}}(\tau)$, $\tau \in [l_i^{\text{time}}, t_i^{\text{close}}]$, which relates the time of starting service at vertex i to the battery level upon arrival at i. The minimum battery level is given by l_i^{energy} and is associated with the earliest service start time l_i^{time}. The vehicle can arrive with a higher battery level by delaying the service start time at vertex i to recharge more at the stations visited previously. Thus, for $\tau \in [l_i^{\text{time}}, t_i^{\text{close}}]$, we get:

$$f_i^{\text{energy}}(\tau) = \begin{cases} l_i^{\text{energy}} + \tau^- \theta_1 & \text{if } \tau^- < l_i^{\text{mrt}_1}, \\ l_i^{\text{energy}} + \sum_{\mu=1}^{\rho(\tau)-1} l_i^{\text{mrt}_\mu} \theta_\mu + \left(\tau^- - \sum_{\mu=1}^{\rho(\tau)-1} l_i^{\text{mrt}_\mu}\right)\theta_{\rho(\tau)} & \text{if } l_i^{\text{mrt}_1} \le \tau^- < \sum_{\mu=1}^{P} l_i^{\text{mrt}_\mu}, \\ l_i^{\text{energy}} + \sum_{\mu=1}^{P} l_i^{\text{mrt}_\mu} \theta_\mu & \text{otherwise,} \end{cases}$$

where $\tau^- = \tau - l_i^{\text{time}}$ and $\rho(\tau) \in \mathcal{P}$ such that $\sum_{\mu=1}^{\rho(\tau)-1} l_i^{\text{mrt}_\mu} < \tau^- \le \sum_{\mu=1}^{\rho(\tau)} l_i^{\text{mrt}_\mu}$. Two examples of this function are illustrated in Figure 4. It can be shown that $f_i^{\text{energy}}(\tau)$ is continuous, piecewise-linear, concave and non-decreasing. The last case ("otherwise") is a constant function representing no more time to recharge due to either a binding time window or a fully charged battery.

A feasible extension is found after Algorithm 2 completes. The labeling algorithm then proceeds to determine whether an existing label dominates the new label or the new label dominates an existing label. A label $l_1 = (l_1^{\text{cost}}, l_1^{\text{load}}, l_1^{\text{time}}, l_1^{\text{energy}}, (l_1^{\text{mrt}_p})_{p\in\mathcal{P}}, (l_1^{\text{cust}_c})_{c\in\mathcal{C}})$ dominates another label $l_2 = (l_2^{\text{cost}}, l_2^{\text{load}}, l_2^{\text{time}}, l_2^{\text{energy}}, (l_2^{\text{mrt}_p})_{p\in\mathcal{P}}, (l_2^{\text{cust}_c})_{c\in\mathcal{C}})$ if both labels are associated with partial paths ending at the same vertex i and all of the following conditions hold:

$$\begin{aligned} l_1^{\text{cost}} &\le l_2^{\text{cost}} \\ l_1^{\text{load}} &\le l_2^{\text{load}} \\ l_1^{\text{time}} &\le l_2^{\text{time}} \end{aligned}$$

$$f_1^{\text{energy}}(\tau) \geq f_2^{\text{energy}}(\tau) \quad \forall \tau \in [l_2^{\text{time}}, t_i^{\text{close}}] \qquad (3)$$
$$l_1^{\text{cust}_c} \leq l_2^{\text{cust}_c} \quad \forall c \in \mathcal{C}.$$

If label l_2 is dominated by l_1, then l_2 can be removed from further consideration as any feasible extension of l_2 is also a feasible extension of l_1 that yields an equal or better reduced cost. In the case that both l_1 dominates l_2 and l_2 dominates l_1, then one of them must kept to ensure that the algorithm is exact.

Testing Condition (3) requires comparing two piecewise-linear functions. This can be accomplished by comparing the two functions at the endpoints and at the breakpoints of either function, i.e.,

$$f_1^{\text{energy}}(\tau) \geq f_2^{\text{energy}}(\tau) \quad \forall \tau \in \left\{ l_2^{\text{time}}, t_i^{\text{close}} \right\} \cup \left\{ l_2^{\text{time}} + \sum_{\mu=1}^{p} l_2^{\text{mrt}_\mu} : p \in \mathcal{P} \wedge l_2^{\text{time}} + \sum_{\mu=1}^{p} l_2^{\text{mrt}_\mu} < t_i^{\text{close}} \right\}.$$

In Figure 4, the blue label could dominate the yellow label because the blue function is greater than the yellow function at all of its breakpoints.

3.6. The Checking Subproblem

Because the RMP (2a)–(2e) does not include, in general, all Constraints (1c), there is no guarantee that there exists a feasible charging schedule for any integral solution to the RMP. Consequently, when an RMP integral solution is found, one must first determine whether it can be associated with a feasible charging schedule. This is accomplished by solving a checking subproblem that can be stated as follows. Given a set of vehicle routes that services each customer exactly once and that includes visits to charging stations, determine if there exists a feasible charging schedule for each vehicle such that each route is energy- and time-feasible and the capacity of all charging stations are not exceeded. Note that charging can only occur during the visits that were already planned in the proposed set of routes.

The checking subproblem is expressed as a constraint programming model based on successor variables over the giant tour representation (e.g., Kilby and Shaw, 2006). Appearing very frequently in constraint programming models of vehicle routing problems, this representation concatenates the routes into a single tour by linking the return to the depot of every route to the departure from the depot of another route. The successor variables are used to identify the successor vertex of every vertex. In our case, the successor variables are fixed according to the solution from the RMP. Nevertheless, given that these variables are required to determine the Benders cuts as explained in the next section, we present a constraint programming model that involves them as if they were not fixed.

Let $\mathcal{K} = \{0, \ldots, K-1\}$ be the set of vehicles. The checking subproblem can be defined over a directed graph $\mathcal{G}_{\text{CS}} = (\mathcal{V}, \mathcal{A}_{\text{CS}})$, which differs slightly from graph $\mathcal{G}_{\text{PS}}$ described in the previous section. The vertex set is again given by $\mathcal{V} = \mathcal{C} \cup \mathcal{S}' \cup \mathcal{D}^+ \cup \mathcal{D}^-$. However, for every vehicle, the giant tour representation requires one source vertex and one sink vertex. Hence, the depot vertex sets are redefined as $\mathcal{D}^+ = \{C + 2C \cdot S, \ldots, C + 2C \cdot S + K - 1\}$ and $\mathcal{D}^- = \{C+2C\cdot S+K, \ldots, C+2C\cdot S+2K-1\}$, where source depot vertex $i = C+2C\cdot S+k \in \mathcal{D}^+$

and sink depot vertex $j = C + 2C \cdot S + K + k \in \mathcal{D}^-$ are both associated with vehicle $k \in \mathcal{K}$. The giant tour representation links the sink depot vertex of vehicle $k \in \{0, \ldots, K-2\}$ to the source depot vertex of vehicle $k+1$, and the sink depot of the last vehicle $K-1$ to the source depot of the first vehicle 0. Hence, the routes can be viewed as one giant tour.

The giant tour formulation uses loops to indicate unvisited vertices. Any vertex not visited by a vehicle will be excluded from the giant tour and will have itself as its immediate predecessor and immediate successor. The network includes loops at the station vertices, which may be unvisited, whereas loops are prohibited at the customer vertices because every customer must be visited.

The arc set $\mathcal{A}_{\text{CS}} = (\mathcal{A}_1 \cup \ldots \cup \mathcal{A}_8 \cup \mathcal{A}_{15} \cup \ldots \cup \mathcal{A}_{18}) \setminus (\mathcal{A}_9 \cup \ldots \cup \mathcal{A}_{14})$ is essentially identical to the arc set $\mathcal{A}_{\text{PS}}$ used in the pricing subproblem but is modified to accommodate the giant tour representation by allowing loops at the stations and by linking each sink depot vertex to the source depot vertex of the next vehicle. Here, the arc subsets $\mathcal{A}_1$ to $\mathcal{A}_{14}$ are defined as in Section 3.5, assuming that $\mathcal{D}^+$ and $\mathcal{D}^-$ contain multiple vertices. The other subsets are given by:

- $\mathcal{A}_{15} = \{(i,j) : i \in \mathcal{D}^+, j \in \mathcal{D}^-, k \in \mathcal{K}, i = C + 2C \cdot S + k, j = C + 2C \cdot S + K + k\}$ contains the arcs from the source depot vertex of each vehicle to its sink depot vertex.
- $\mathcal{A}_{16} = \{(i,i) : i \in \mathcal{S}'\}$ contains the loops at the stations.
- $\mathcal{A}_{17} = \{(i,j) : i \in \mathcal{D}^-, j \in \mathcal{D}^+, k \in \mathcal{K} \setminus \{K-1\}, i = C + 2C \cdot S + K + k, j = C + 2C \cdot S + (k+1)\}$ contains the arcs from the sink depot vertex of each vehicle to the source depot vertex of the next vehicle.
- $\mathcal{A}_{18} = \{(i,j) : i \in \mathcal{D}^-, j \in \mathcal{D}^+, i = C + 2C \cdot S + K + K, j = C + 2C \cdot S\}$ contains the arc from the sink depot vertex of the last vehicle to the source depot vertex of the first vehicle.

Let $p_i \in \mathcal{V}$ and $n_i \in \mathcal{V}$ be integer variables storing the immediate predecessor and immediate successor of vertex $i \in \mathcal{V}$ respectively. That is, p_i is the vertex previous to i in a route and n_i is the next vertex after i. If $n_i = i$ and $p_i = i$, then i does not appear in any route. When solving the checking subproblem, all p_i and n_i variables are fixed to the values that specify the routes in the solution to the RMP. Then, the constraint programming solver will be able to propagate these values to the other constraints and in turn derive a conflict on these variables.

Let $k_i \in \mathcal{K}$ be a variable storing the vehicle that visits vertex $i \in \mathcal{V}$. Let $l_i \in \{0, \ldots, T^{\text{load}}\}$ be the total load delivered up to and including vertex $i \in \mathcal{V}$. The model assumes that recharging is possible at any vertex but the recharge duration is 0 at non-station vertices. Let $t_i \in \{t_i^{\text{open}}, \ldots, t_i^{\text{close}}\}$ and $t_i' \in \{0, \ldots, T^{\text{time}}\}$ be the time of starting and completing recharge at $i \in \mathcal{V}$, respectively. Let $e_i, e_i' \in \{0, \ldots, T^{\text{energy}}\}$ be the battery level before and after recharging at $i \in \mathcal{V}$, respectively. Define $d_i \in \{0, \ldots, T^{\text{time}}\}$ as the recharge duration at $i \in \mathcal{V}$. For every station $s \in \mathcal{S}$ and its expanded vertices $i \in \mathcal{S}_s'$, let $m_i \in \{0, \ldots, M_s - 1\}$ be the charger used by the vehicle at i.

The constraint programming model is given as follows:

$$\textsc{Inverse}(\mathbf{n}, \mathbf{p}), \tag{4a}$$

$$\textsc{SubCircuit}(\mathbf{n}), \tag{4b}$$

$$k_{n_i} = k_i \qquad \forall i \in \mathcal{C} \cup \mathcal{S}' \cup \mathcal{D}^+, \tag{4c}$$

$$k_{C+2C\cdot S+k} = k \qquad \forall k \in \mathcal{K}, \tag{4d}$$

$$k_{C+2C\cdot S+K+k} = k \qquad \forall k \in \mathcal{K}, \tag{4e}$$

$$n_{C+2C\cdot S+k} < n_{C+2C\cdot S+k+1} \qquad \forall k \in \{0, \dots, K-2\}, \tag{4f}$$

$$t'_i = t_i \qquad \forall i \in \mathcal{V} \setminus \mathcal{S}', \tag{4g}$$

$$e'_i = e_i \qquad \forall i \in \mathcal{V} \setminus \mathcal{S}', \tag{4h}$$

$$d_i = \sum_{w \in \mathcal{P}} (1/\theta_w) \cdot \max\{\min\{\sigma_w, e'_i\} - \max\{\sigma_{w-1}, e_i\}, 0\} \qquad \forall i \in \mathcal{S}', \tag{4i}$$

$$t'_i = t_i + d_i \qquad \forall i \in \mathcal{S}', \tag{4j}$$

$$e'_i \geq e_i \qquad \forall i \in \mathcal{S}', \tag{4k}$$

$$\textsc{DiffN}((t_i)_{i \in \mathcal{S}'_s}, (d_i)_{i \in \mathcal{S}'_s}, (m_i)_{i \in \mathcal{S}'_s}, \mathbf{1}) \qquad \forall s \in \mathcal{S}, \tag{4l}$$

$$l_i = l_{p_i} + t_i^{\text{load}} \qquad \forall i \in \mathcal{C} \cup \mathcal{S}' \cup \mathcal{D}^-, \tag{4m}$$

$$t_{n_i} \geq t'_i + t_{i,n_i}^{\text{travel}} \qquad \forall i \in \mathcal{C} \cup \mathcal{S}' \cup \mathcal{D}^+, \tag{4n}$$

$$e_{n_i} = e'_i - t_{i,n_i}^{\text{energy}} \qquad \forall i \in \mathcal{C} \cup \mathcal{S}' \cup \mathcal{D}^+, \tag{4o}$$

$$l_i = t_i = 0 \qquad \forall i \in \mathcal{D}^+, \tag{4p}$$

$$e_i = T^{\text{energy}} \qquad \forall i \in \mathcal{D}^+, \tag{4q}$$

$$p_j \in \{i : (i,j) \in \mathcal{A}_{\text{CS}}\} \qquad \forall j \in \mathcal{V}, \tag{4r}$$

$$n_i \in \{j : (i,j) \in \mathcal{A}_{\text{CS}}\} \qquad \forall i \in \mathcal{V}, \tag{4s}$$

$$k_i \in \mathcal{K} \qquad \forall i \in \mathcal{V}, \tag{4t}$$

$$l_i \in \{0, \dots, T^{\text{load}}\} \qquad \forall i \in \mathcal{V}, \tag{4u}$$

$$t_i \in \{t_i^{\text{open}}, \dots, t_i^{\text{close}}\} \qquad \forall i \in \mathcal{V}, \tag{4v}$$

$$t'_i \in \{0, \dots, T^{\text{time}}\} \qquad \forall i \in \mathcal{V}, \tag{4w}$$

$$d_i \in \{0, \dots, T^{\text{time}}\} \qquad \forall i \in \mathcal{V}, \tag{4x}$$

$$e_i, e'_i \in \{0, \dots, T^{\text{energy}}\} \qquad \forall i \in \mathcal{V}, \tag{4y}$$

$$m_i \in \{0, \dots, M_s - 1\} \qquad \forall s \in \mathcal{S}, i \in \mathcal{S}'_s. \tag{4z}$$

Constraint (4a) links the successor and predecessor variables, where $\mathbf{n} = (n_0, \dots, n_{C+2C\cdot S+2K-1})$ and $\mathbf{p} = (p_0, \dots, p_{C+2C\cdot S+2K-1})$ are the vectors of the successor variables and predecessor variables, respectively. Formally, this constraint ensures that $n_i = j \leftrightarrow p_j = i$ for all $i, j \in \mathcal{V}$. This constraint also implicitly requires the successor variables to take unique values and the predecessor variables to take unique values; i.e., it contains $\textsc{AllDifferent}(\mathbf{n})$ and $\textsc{AllDifferent}(\mathbf{p})$ as necessary conditions. Constraint (4b) enforces the giant tour.

Constraints (4c) track each vehicle along its route by equating the index of the vehicle visiting vertex i to the index of the vehicle visiting the successor of i. Constraints (4d)

and (4e) require the vehicle visiting a source or sink depot vertex to be exactly the vehicle associated with the vertex. The giant tour formulation ties a particular vehicle index to every source and sink depot vertex. This introduces symmetry due to permutations of the vehicle index. That is, permuting the routes and assigning them to different vehicles give exponentially-many equivalent solutions. Constraints (4f) break this vehicle symmetry by imposing an ordering on the first vertex (successor of the source depot vertex) across all routes. Given a solution, permuting the routes is now infeasible.

Constraints (4g) and (4h) forbid recharging at non-station vertices by equating the time and battery level before and after recharging. Constraints (4i) sum the amount of time spent recharging in each piece by computing the amount of energy replenished in each piece, which may be clamped by the start and end battery levels of the piece. Constraints (4j) calculate the timestep when recharging is completed. Constraints (4k) permit recharging at a station vertex.

Constraints (4l) schedule the recharges at each recharging station. The DIFFN global constraint is handled natively in constraint programming using a specialized algorithm. The vector $\mathbf{1}$ is the unit vector with appropriate dimension.

Constraints (4m) track the load along a route. These constraints state that the total load delivered up to vertex i is the total load delivered up to the predecessor of i incremented by the amount delivered at i. Constraints (4n) track the time along a route by requiring the recharge start time at the successor of i to be later than the recharge completion time at i plus the travel time. Similarly, Constraints (4o) track the battery level along a route.

Constraints (4p) and (4q) initialize the load, time and battery level at the start of a route. Constraints (4r) to (4z) restrict the domain of the variables.

A nuance here is that most constraint programming solvers only support integer variables and do not support floating point variables because computing the lower bound and upper bound of the domain of a floating point variable in exact arithmetic is both tricky and slow. To accommodate this limitation, all input data and variables in the checking subproblem are restricted to integer values. In particular, the time variables t_i, t'_i and the energy variables e_i, e'_i at each vertex $i \in \mathcal{V}$ and the $1/\theta_w$ coefficient in Constraints (4i) must take integral values. This is not required for the pricing subproblem. However, in practice, the time data (time windows, travel times, recharging durations, etc.) are often assumed to be integer. Furthermore, an amount of energy can be approximated by a scaled integer value obtained by first rounding this amount to a suitable number of decimals η before multiplying it by 10^η.

3.7. Translating Master Problem Candidate Solutions and Nogoods

This section describes how a set of routes from an integer feasible solution to the RMP can be translated into a partial solution in the checking subproblem, and how a nogood in the checking subproblem can be translated into a Benders cut in the RMP.

The automatic logic-based Benders decomposition and branch-and-check algorithm of Lam and Van Hentenryck (2017); Davies et al. (2017) and Lam et al. (2020) require the integer programming problem and the constraint programming problem to communicate through a set of identical variables. Hence, the constraint programming problem historically

uses the same representation as the integer programming problem and therefore cannot exploit the capabilities of global constraints.

The route variables in the RMP and the successor variables in the checking subproblem are clearly not identical. To remedy this issue, first consider an intermediary: a network flow representation with arc variables $x_{i,j} \in \{0,1\}$ that indicate whether an arc $(i,j) \in \mathcal{A}_{\text{PS}}$ is used. This is the well-known *two-index* formulation commonly used in the Traveling Salesman Problem and branch-and-cut arc-based models of vehicle routing problems. Observe that any feasible solution $\hat{\mathbf{x}}$ to the RMP has an equivalent solution in the network flow representation:

$$\hat{x}_{i,j} = \sum_{r \in \mathcal{R}} A_r^{i,j} \hat{x}_r.$$

Even though the arc variables in the network flow representation and the successor variables in the giant tour representation are equivalent, the depot vertices $\mathcal{D}^+ \cup \mathcal{D}^-$, and hence the arc sets $\mathcal{A}_{\text{PS}}$ and $\mathcal{A}_{\text{CS}}$, differ substantially. In particular, the network flow representation has no vehicle symmetry, whereas the giant tour representation duplicates the depot vertex for every vehicle, which consequently introduces symmetry due to permutations of the vehicle index.

Consider the subset $\mathcal{A}_\cap = \mathcal{A}_{\text{PS}} \cap \mathcal{A}_{\text{CS}} = \{(i,j) \in \mathcal{A}_{\text{PS}} : i,j \in \mathcal{C} \cup \mathcal{S}'\}$ of arcs that exists in both models. These arcs have customer or station end-points. For these arcs $(i,j) \in \mathcal{A}_\cap$, it is simply a matter of fixing $n_i = j$ whenever $\hat{x}_{i,j} = 1$ and then running the propagation algorithms, which will either (1) declare that the solution $(\hat{x}_{i,j})_{(i,j)\in\mathcal{A}_\cap}$ is schedule-feasible and return a schedule $((\hat{t}_i)_{i\in\mathcal{V}}, (\hat{t}'_i)_{i\in\mathcal{V}})$, or (2) declare that the solution $(\hat{x}_{i,j})_{(i,j)\in\mathcal{A}_\cap}$ is schedule-infeasible and return an *explanation*, i.e., a conjunction of several variable-value pairs that implicate the infeasibility:

$$\bigwedge_{(i,j)\in\mathcal{A}_1} \neg[\![n_i = j]\!] \wedge \bigwedge_{(i,j)\in\mathcal{A}_2} [\![n_i = j]\!] \rightarrow \text{infeasible},$$

where $\mathcal{A}_1, \mathcal{A}_2 \subset \mathcal{A}_\cap$ and $\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset$. This explanation states that no feasible schedule exists whenever all conditions $n_i \neq j$ for $(i,j) \in \mathcal{A}_1$ and $n_i = j$ for $(i,j) \in \mathcal{A}_2$ hold simultaneously. Negating the explanation yields a *nogood*, i.e., a constraint that prevents the same infeasibility from reoccurring:

$$\bigvee_{(i,j)\in\mathcal{A}_1} [\![n_i = j]\!] \vee \bigvee_{(i,j)\in\mathcal{A}_2} \neg[\![n_i = j]\!].$$

The nogood can then be translated into the combinatorial Benders cut

$$\sum_{(i,j)\in\mathcal{A}_1} x_{i,j} + \sum_{(i,j)\in\mathcal{A}_2} (1 - x_{i,j}) \geq 1,$$

which can be rearranged to

$$\sum_{(i,j)\in\mathcal{A}_1} x_{i,j} - \sum_{(i,j)\in\mathcal{A}_2} x_{i,j} \geq 1 - |\mathcal{A}_2|.$$

This Benders cut expressed in the network flow representation must then be rewritten in terms of the master problem variables (see Desaulniers et al., 2011):

$$\sum_{r \in \mathcal{R}^*} \left(\sum_{(i,j) \in \mathcal{A}_1} A_r^{i,j} - \sum_{(i,j) \in \mathcal{A}_2} A_r^{i,j} \right) x_r \geq 1 - |\mathcal{A}_2|.$$

It corresponds to a new $b \in \mathcal{B}$ in Constraints (2c), where $B_b = 1 - |\mathcal{A}_2|$,

$$B_b^r = \sum_{(i,j) \in \mathcal{A}_1} A_r^{i,j} - \sum_{(i,j) \in \mathcal{A}_2} A_r^{i,j} = \sum_{(i,j) \in \mathcal{A}} B_b^{i,j} A_r^{i,j}$$

and

$$B_b^{i,j} = \begin{cases} 1 & \text{if } (i,j) \in \mathcal{A}_1 \\ -1 & \text{if } (i,j) \in \mathcal{A}_2 \\ 0 & \text{otherwise.} \end{cases}$$

The main argument here is that the naive mapping $x_{i,j} = 1 \leftrightarrow n_i = j$ is valid because $(i,j) \in \mathcal{A}_\cap$ exists in both models.

The only remaining issue concerns the vehicle-specific depot vertices because explanations over vehicle-specific arcs cannot be translated into a Benders cut over arcs not tied to a particular vehicle. Specifically, prohibiting one vehicle from taking an arc does not preclude a different vehicle from taking the same arc. Hence, the naive translation described above requires some modifications.

Observe that the depot vertices $\mathcal{D}^+ \cup \mathcal{D}^-$ are numbered higher than the station vertices, from $C + 2C \cdot S$ onward. Then, the mapping between the two models is not especially difficult:

- Arcs outgoing from the source depot vertex: $x_{C+2C \cdot S,i} = 1 \leftrightarrow p_i \geq C + 2C \cdot S$ for any $i \in \mathcal{V}$.
- Arcs incoming to the sink depot vertex: $x_{i,C+2C \cdot S+1} = 1 \leftrightarrow n_i \geq C + 2C \cdot S + K$ for any $i \in \mathcal{V}$.

For example, if $\hat{x}_{C+2C \cdot S,i} = 1$ for some vertex $i \in \mathcal{V}$, the bound $p_i \geq C + 2C \cdot S$ is placed in the constraint programming model and propagation runs as usual. If it returns an explanation containing $[\![p_i \geq C + 2C \cdot S]\!]$ (or nogood containing $[\![p_i < C + 2C \cdot S]\!]$), then this literal is manually replaced with the term $(1 - \hat{x}_{C+2C \cdot S,i})$ in the Benders cut, i.e., $(C + 2C \cdot S, i) \in \mathcal{A}_2$.

3.8. Reducing the Number of Arcs in a Combinatorial Benders Cut

Conflict analysis stores the propagations (the changes to the domain of each variable) made by each constraint by dynamically constructing a graph called the *implication graph* (Ohrimenko et al., 2009; Feydy and Stuckey, 2009). Tightening the domain of one variable often triggers propagations on other variables, tightening the domain of these variables in turn. The sequence of propagations can eventually lead to a conflict. Upon detecting a conflict, conflict analysis traces the implication graph to deduce a set of variable-value pairs that provoke the conflict and then creates a combinatorial Benders cut that prevents these variable-value pairs

from occurring again. By construction, the cut generated by conflict analysis cannot contain more variables than the naive combinatorial Benders cut which contains all variable-value pairs of the master problem candidate solution fixed in the checking subproblem. Therefore, Benders cuts found using conflict analysis are stronger than the naive combinatorial Benders cuts commonly seen in the literature simply because they concern a smaller set of variables. However, conflict analysis does not guarantee that the cut contains the fewest number of variables for explaining the conflict. The cut can contain propagations in the chain leading to the conflict but not directly implicating the conflict. Hence, arcs in the cut not necessary for explaining the conflict can be removed in a post-processing step.

Once conflict analysis has found a set of $Q \leq K$ jointly infeasible partial routes $\hat{\mathcal{R}} = \{r^1, \ldots, r^Q\}$ such that $r^q = (r^q_1, \ldots, r^q_{n_q})$ for all $q \in \{1, \ldots, Q\}$, Algorithm 3 attempts to isolate the infeasibility to one station. Line 1 loops through each station s in turn. Line 2 extracts the routes that visit station s. If s is visited by more routes than its number of chargers (Line 3), Line 4 solves the checking subproblem again with this subset of routes. If these routes are infeasible (Line 5), the conflict can be isolated to station s, and hence, Line 6 creates a cut over the smaller route set $\hat{\mathcal{R}}_s$ before exiting on Line 7. If the cut cannot be isolated to one station, Line 8 creates a cut over the original arcs.

Algorithm 3 Procedure for reducing the number of arcs in a Benders cut

1: **for** $s \in \mathcal{S}$ **do**
2: $\quad \hat{\mathcal{R}}_s = \{r^q \in \hat{\mathcal{R}} : \exists i \in \{1, \ldots, n_q\} \wedge r^q_i \in \mathcal{S}'_s\}$
3: $\quad$ **if** $|\hat{\mathcal{R}}_s| > M_s$ **then**
4: $\quad\quad$ solve the checking subproblem again with some successor variables fixed to $\hat{\mathcal{R}}_s$
5: $\quad\quad$ **if** infeasible **then**
6: $\quad\quad\quad$ create a combinatorial Benders cut over $\hat{\mathcal{R}}_s$
7: $\quad\quad\quad$ exit
8: create a combinatorial Benders cut over $\hat{\mathcal{R}}$

Line 4 is very time-consuming. To reduce the computation time, the post-processing procedure is only run up to a certain depth in the branch-and-bound tree. A depth limit of five is chosen for our experiments.

3.9. Lifting the Combinatorial Benders Cuts to Multi-Tournament Cuts

The Benders cuts generated by logical inference do not exploit the polyhedral structure of the network flow representation nor the set partition representation. This section presents minor polyhedral analysis on the network flow model to lift the Benders cuts into a stronger form.

Ascheuer et al. (2000) study the polyhedron of the Asymmetric Traveling Salesman Problem, which is closely related to the single-vehicle VRPTW. They show that whenever a partial route $r = (r_1, \ldots, r_n)$ is infeasible, the constraint

$$\sum_{i=1}^{n-1} x_{r_i, r_{i+1}} \leq n - 2,$$

which prohibits at least one of the $n-1$ arcs of r, can be lifted to the *tournament inequality*

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_{r_i,r_j} \leq n-2.$$

A combinatorial Benders cut that prohibits at least one arc from multiple partial routes can also be lifted using the same idea, as shown in Proposition 1.

Proposition 1. Given a set of $Q \leq K$ jointly infeasible partial routes $\hat{\mathcal{R}} = \{r^1, \ldots, r^Q\}$ such that $r^q = (r_1^q, \ldots, r_{n_q}^q)$ for all $q \in \{1, \ldots, Q\}$, the combinatorial Benders cut

$$\sum_{q=1}^{Q} \sum_{i=1}^{n_q-1} x_{r_i^q, r_{i+1}^q} \leq \sum_{q=1}^{Q} (n_q - 1) - 1$$

can be lifted to the *multi-tournament inequality*

$$\sum_{q=1}^{Q} f(r^q) \leq \sum_{q=1}^{Q} (n_q - 1) - 1 \tag{5}$$

where

$$f(r^q) = \begin{cases} \displaystyle\sum_{i=1}^{n_q-1} \sum_{j=i+1}^{n_q} x_{r_i^q, r_j^q} & \text{if } r_1^q \notin \mathcal{D}^+, r_{n_q}^q \notin \mathcal{D}^-, \\ x_{r_1^q, r_2^q} + \displaystyle\sum_{i=2}^{n_q-1} \sum_{j=i+1}^{n_q} x_{r_i^q, r_j^q} & \text{if } r_1^q \in \mathcal{D}^+, r_{n_q}^q \notin \mathcal{D}^-, \\ \displaystyle\sum_{i=1}^{n_q-2} \sum_{j=i+1}^{n_q-1} x_{r_i^q, r_j^q} + x_{r_{n_q-1}^q, r_{n_q}^q} & \text{if } r_1^q \notin \mathcal{D}^+, r_{n_q}^q \in \mathcal{D}^-, \\ x_{r_1^q, r_2^q} + \displaystyle\sum_{i=2}^{n_q-2} \sum_{j=i+1}^{n_q-1} x_{r_i^q, r_j^q} + x_{r_{n_q-1}^q, r_{n_q}^q} & \text{if } r_1^q \in \mathcal{D}^+, r_{n_q}^q \in \mathcal{D}^-. \end{cases}$$

Proof. Consider the first case for all routes, i.e., $r_1^q \notin \mathcal{D}^+, r_{n_q}^q \notin \mathcal{D}^-$ for all $q \in \{1, \ldots, Q\}$. In other words, $r_i^q \in \mathcal{C} \cup \mathcal{S}'$ for all $q \in \{1, \ldots, Q\}$ and $i \in \{1, \ldots, n_q\}$. By substitution, Constraint (5) is simplified to

$$\sum_{q=1}^{Q} \sum_{i=1}^{n_q-1} \sum_{j=i+1}^{n_q} x_{r_i^q, r_j^q} \leq \sum_{q=1}^{Q} (n_q - 1) - 1. \tag{6}$$

Assume the complement

$$\sum_{q=1}^{Q} \sum_{i=1}^{n_q-1} \sum_{j=i+1}^{n_q} x_{r_i^q, r_j^q} > \sum_{q=1}^{Q} (n_q - 1) - 1. \tag{7}$$

The pricing subproblem will find routes that visit a customer or station vertex at most once,

implying that the network flow formulation contains the *degree constraints*:

$$\sum_{j:(i,j)\in\mathcal{A}_{\mathrm{PS}}} x_{i,j} \leq 1 \qquad \forall i \in \mathcal{C} \cup \mathcal{S}' \quad (8)$$

$$\sum_{h:(h,i)\in\mathcal{A}_{\mathrm{PS}}} x_{h,i} \leq 1 \qquad \forall i \in \mathcal{C} \cup \mathcal{S}'. \quad (9)$$

Given that the variable domain $x_{i,j} \leq 1$ for all $(i,j) \in \mathcal{A}_{\mathrm{PS}}$, Constraints (7) and (8) together imply that

$$\sum_{j=i+1}^{n_q} x_{r_i^q, r_j^q} = 1$$

for all $q \in \{1,\ldots,Q\}$ and $i \in \{1,\ldots,n_q-1\}$, which can be written explicitly as

$$x_{r^q_{n_q-1}, r^q_{n_q}} = 1, \quad (10)$$

$$x_{r^q_{n_q-2}, r^q_{n_q-1}} + x_{r^q_{n_q-2}, r^q_{n_q}} = 1, \quad (11)$$

$$\vdots$$

$$x_{r_1^q, r_2^q} + \ldots + x_{r_1^q, r_{n_q}^q} = 1 \quad (12)$$

for all $q \in \{1,\ldots,Q\}$. By Constraints (9) and (10), $x_{r^q_{n_q-2}, r^q_{n_q}} = 0$ and hence Constraint (11) can be simplified to $x_{r^q_{n_q-2}, r^q_{n_q-1}} = 1$. Repeating the same logic, Constraints (10) to (12) imply that $x_{r_i^q, r_{i+1}^q} = 1$ for all $q \in \{1,\ldots,Q\}$ and $i \in \{1,\ldots,n_q-1\}$, which is exactly the original conflict. Therefore Constraint (6) is valid by contradiction. The proofs for the other three cases are identical but respectively exclude the first vertex, the last vertex or both the first and last vertices because the degree constraints do not apply on the depot vertices. □

3.10. Branching Rules

As the RMP is a linear program, integrality is enforced by branching. Many vehicle routing solvers based on column generation implement a hierarchical branching rule that first branches on the number of routes and then on an individual arc. This hierarchical branching rule is augmented with two additional branching rules after branching on the number of routes but before branching on arcs. Given a solution $\hat{\mathbf{x}}$ to the RMP such that the number of visits to all stations $\sum_{r\in\mathcal{R}} \sum_{i\in\mathcal{S}'} A_r^i \hat{x}_r$ is fractional, integrality is enforced by branching on

$$\sum_{r\in\mathcal{R}^*} \sum_{i\in\mathcal{S}'} A_r^i x_r.$$

Next, for a station $s \in \mathcal{S}$, integrality of the number of visits to s is enforced by branching on

$$\sum_{r\in\hat{\mathcal{R}}} \sum_{i\in\mathcal{S}'_s} A_r^i x_r.$$

3.11. Valid Inequalities and Acceleration Techniques

The literature abounds with countless cutting planes and other techniques for boosting the performance of BCP algorithms. The following improvements are implemented.

- **Rounded capacity inequalities:** Laporte et al. (1985) generalize the well-known subtour elimination constraints to the rounded capacity cuts which additionally includes the vehicle capacity constraint. Our solver uses the separator from the CVRPSEP package by Lysgaard et al. (2004).

- **Two-path inequalities:** Kohl et al. (1999) invent the two-path cuts. Given a subset of customers, the two-path cuts state that if one vehicle cannot visit all the customers in the set due to, e.g., a violated load or time window constraint, then those customers must be covered by at least two vehicles.

- **Subset row inequalities:** Jepsen et al. (2008) devise the subset row cuts, which tighten the set packing polyhedron in the RMP (i.e., half of Constraints (2b)), as opposed to the rounded capacity and two-path cuts, which handle the load and time window side constraints in the network flow representation.

- **Heuristic pricing:** In the early rounds of pricing, it is not necessary to solve the pricing subproblem to optimality as discovering any route with negative reduced cost is sufficient. To accelerate the pricing, the labeling algorithm is called on a smaller graph in which every vertex has a limited number of incoming and outgoing arcs. In the implementation, every vertex is given the five incoming arcs and the five outgoing arcs with the lowest reduced cost. If heuristic pricing fails, i.e., if it cannot find a route with negative reduced cost, then exact pricing comprising the full graph proceeds.

- **ng-route pricing:** The elementarity constraint in the pricing subproblem, which requires every customer to be visited at most once, significantly increases the difficulty of the problem because the chance that the conditions on the l^{cust_c} components in the dominance rule are met is substantially reduced. In ng-route pricing, the elementarity constraint is partially relaxed by occasionally resetting l^{cust_c} to 0 for some $c \in \mathcal{C}'$ according to neighborhoods $\mathcal{C}' \subset \mathcal{C}$ that are defined for each customer vertex (Baldacci et al., 2011). This procedure trades off the tightness of the linear relaxation bound for improved performance in the labeling algorithm.

4. Experimental Results

This section reports the performance of the BCP algorithm and analyzes the impacts of piecewise-linear recharging and the station capacity constraints.

4.1. The Instances

Schneider et al. (2014)[1] created instances for the EVRPTW with 21 recharging stations by modifying the 100-customer benchmark instances of Solomon (1987)[2] for the VRPTW. Preliminary experiments indicate that the station capacity constraints are not binding in these instances simply because there are too many stations. Consequently, new instances are generated by replacing all stations with fewer randomly-located stations. Based on these stations, one set of instances is created with one charger at all stations and another set is created with two chargers at all stations. Like the Solomon benchmarks, smaller instances with 25, 50 and 75 customers are then created from the 100-customer instances by discarding the higher-numbered customers. In total, there are 224 instances in each of the two sets. These instances are available online.[3]

4.2. The Solver

The implementation calls SCIP 7.0.3 (Gamrath et al., 2020) for branch-and-bound, Geas (Gange et al., 2020) for constraint programming and Gurobi 9.5.1 for linear programming. The labeling algorithm is implemented in C++. The solver is single-threaded and is run on an Intel Xeon Platinum 8260 CPU at 2.4 GHz with a time limit of one hour and a memory limit of 16 GB.

4.3. The Impact of Piecewise-Linear Recharging and Station Capacity Constraints

Table 1 compares the performance of the BCP algorithm on the baseline EVRPTW (linear recharging and no station capacity) against the new variants with piecewise-linear (PWL) recharging and station capacity constraints. The first two columns identify the problem variant, comprising the recharging function and the number of chargers available at each station. The third column shows the number of customers in the statistics in the remainder of the row. The fourth column reports the number of instances solved to optimality or proved infeasible within the time limit. The remaining columns report the mean solve time in seconds, optimality gap, number of branch-and-bound nodes, number of columns generated and number of Benders cuts generated. For calculating the mean, the solve time of timed-out instances is recorded as 3600 seconds and the optimality gap of instances with no primal or dual bound is recorded as 100%.

In the baseline, the linear recharging function assumes that the battery can be fully recharged using the fastest recharging rate (the first piece). The results indicate that adding piecewise-linear recharging makes the problem easier because it is more constrained by the time windows due to delays from the slowest piece. The algorithm solves an additional 8 instances (171 vs. 163) and achieves an average runtime 122 seconds shorter (1069 vs. 1191).

In contrast, adding station capacity constraints makes the problem harder. Restricting the stations to one charger reduces the number of instances solved by 28 (135 vs. 163) and

[1] `https://data.mendeley.com/datasets/h3mrm5dhxw/1`
[2] `http://web.cba.neu.edu/~msolomon/problems.htm`
[3] `https://ed-lam.com`

Recharging Function	Chargers	Customers	Solved	Time	Gap	Nodes	Columns	Benders Cuts
Linear	∞	25	56/56	49	0%	219	1136	0
		50	46/56	767	5%	897	2572	0
		75	40/56	1370	11%	957	4192	0
		100	21/56	2577	27%	1741	6402	0
		All	163/224	1191	11%	953	3575	0
PWL	∞	25	56/56	13	0%	108	1016	0
		50	45/56	824	5%	607	2366	0
		75	44/56	1129	15%	430	3989	0
		100	26/56	2310	32%	1524	5901	0
		All	171/224	1069	13%	667	3318	0
Linear	1	25	55/56	105	2%	620	1135	9
		50	39/56	1214	14%	2495	3173	55
		75	29/56	2123	29%	2811	5740	31
		100	12/56	3014	45%	2157	7526	13
		All	135/224	1614	22%	2021	4394	27
Linear	2	25	56/56	40	0%	187	1118	0
		50	46/56	782	5%	958	2592	0
		75	35/56	1603	14%	1481	4454	5
		100	20/56	2671	34%	1836	6730	3
		All	157/224	1274	13%	1116	3724	2
PWL	1	25	55/56	79	2%	654	1064	5
		50	38/56	1337	19%	2293	3194	59
		75	29/56	1973	35%	2387	5374	44
		100	17/56	2744	48%	2222	7026	13
		All	139/224	1533	26%	1889	4165	30
PWL	2	25	56/56	13	0%	96	1002	0
		50	44/56	899	7%	852	2398	3
		75	39/56	1413	21%	928	4301	6
		100	21/56	2512	38%	1711	6166	4
		All	160/224	1209	16%	897	3467	3

Table 1: Average statistics for all instances. Rounded to the nearest integer.

increases the average runtime by 423 seconds (1614 vs. 1191). The station capacity constraints are not as tight with two chargers and hence the impact is not as dramatic. Adding the second charger allows 22 more instances (157 vs. 135) to be solved. Incorporating piecewise-linear recharging into the problem with station capacity constraints again improves the performance measures by solving a few more instances (139 vs. 135 and 160 vs. 157).

Table 2 presents statistics on the subset of instances with at least one Benders cut found. The fourth column reports the number of solved instances from this subset of instances. In contrast to other work, these results suggest that the station capacity constraints are not binding in most instances. With piecewise-linear recharging and one charger at every station, Benders cuts appear in even the smallest instances (25 customers) but are found in only 32 of the 224 instances.

The sixth to eighth columns show the proportion of time spent within the restricted master, pricing and the checking problems. Note that these percentages do not sum to 100% because they do not include time spent in the other components of branch-and-bound such as branching, node selection, etc. A lot more time is spent pricing than checking for the instances with one charger. With two chargers, the checking problem becomes much more time-consuming. This is because the problem is less constrained, making infeasibility harder to detect without a thorough search. The average optimality gap for the subset of instances that requires scheduling one charger is significantly worse (39% vs. 26%). Similar findings are observed for two chargers (27% vs. 16%).

Observe that the number of Benders cuts peaks at 50 customers for the problem with piecewise-linear recharging and one charger at every station. This occurs for several reasons. Instances with few customers do not require many vehicles, so there is less need for multiple vehicles to recharge at the same time. That is, whenever a vehicle wants to charge at a particular station, it is likely that the station is available. Furthermore, a fewer number of customers can be covered with shorter routes and a vehicle taking a short route likely means that the battery has enough initial charge for the entire route. At 50 customers, charging scheduling starts to become necessary, as seen in the large number of Benders cuts. As the problem becomes harder with more customers, the number of Benders cuts decreases because the algorithm spends significantly more time in pricing (trying to get to an optimal RMP solution before starting checking) and because the checking subproblem is called fewer times since integral solutions are harder to come by.

The last four columns report the average number of vehicles used, the average number of stations used (i.e., receives at least one vehicle), the average proportion of time that the used stations are charging at least one vehicle and the average proportion of time that the used stations are at capacity. The results show that the stations are charging at least one vehicle for a significant amount of time. In the case of one charger, a station is at capacity whenever it is in use. The stations are in use around 1/3rd of the time. For the larger instances with two chargers, the stations are at capacity around 1/10th of the time. Adding one additional charger to every station substantially alleviates the capacity limitations.

Overall, the hybrid BCP algorithm solves 30% (17 of 56) of the 100-customer instances

Recharging Function	Chargers	Customers	Solved	Time	Time in Master	Time in Pricing	Time in Checking	Gap	Nodes	Columns	Benders Cuts	Vehicles Used	Stations Used	Station in Use	Station at Capacity
Linear	1	25	2/3	1202	2%	25%	34%	33%	8186	567	167	9	4/10	39%	39%
		50	12/20	1542	3%	47%	18%	25%	5737	2675	153	15	3/7	35%	35%
		75	11/24	2421	5%	72%	8%	38%	6153	5592	73	20	3/7	30%	30%
		100	5/21	2885	3%	87%	3%	38%	4729	6575	36	27	3/6	33%	33%
		All	30/68	2252	4%	67%	10%	34%	5681	4816	89	20	3/7	33%	33%
Linear	2	50	3/3	522	1%	83%	12%	0%	2276	1616	4	18	9/13	33%	7%
		75	2/7	2630	5%	75%	12%	15%	6071	4110	36	23	7/10	39%	16%
		100	1/7	3507	2%	91%	4%	44%	5130	6574	23	33	6/8	43%	18%
		All	6/17	2619	3%	83%	8%	24%	5014	4685	25	25	7/10	39%	14%
PWL	1	25	3/4	930	2%	55%	11%	25%	7819	1206	70	6	3/10	35%	35%
		50	13/21	1641	4%	49%	15%	38%	5710	3082	158	15	2/8	38%	38%
		75	9/24	2427	4%	70%	10%	42%	5350	5406	103	20	3/7	30%	30%
		100	7/21	2683	3%	86%	3%	39%	5103	6569	34	27	3/5	28%	28%
		All	32/70	2182	4%	68%	9%	39%	5525	4818	97	20	3/7	32%	32%
PWL	2	25	1/1	5	0%	12%	86%	0%	13	241	1	7	6/8	24%	1%
		50	3/5	1660	1%	44%	23%	22%	7546	1461	37	19	7/9	36%	15%
		75	4/8	2026	2%	44%	45%	26%	3910	3440	44	25	8/9	37%	11%
		100	1/9	3235	1%	87%	8%	34%	5018	5755	23	25	5/7	43%	17%
		All	9/23	2332	1%	60%	28%	27%	4964	3777	32	22	6/8	38%	14%

Table 2: Average statistics for the subset of instances with at least one Benders cut. Rounded to the nearest integer.

Customers	Solved	Time	Gap	Nodes	Columns	Benders Cuts
25	54/56	150	13%	1177	792	1
50	45/56	790	33%	688	2580	27
75	42/56	1034	59%	95	3734	5
100	32/56	1658	70%	313	4817	0
All	173/224	908	44%	568	2981	8

Table 3: Average statistics for full recharging and one charger at every station. Rounded to the nearest integer.

with piecewise-linear recharging and one charger at every station and 38% (21 of 56) of the instances with two chargers. Considering the subset of 100-customer instances with binding station capacity constraints, the algorithm solves 33% (7 of 21) and 11% (1 of 9), respectively. These results signify that the station capacity constraints remain a significant challenge. Nonetheless, the results demonstrate that the proposed method displays substantially better scalability than the existing methods for related but different problems. The results confirm that the route-then-schedule methodology is appropriate for the EVRPTW-PLR-CRS because the EVRPTW-PLR-CRS is predominantly a routing problem. The scheduling component becomes relatively easy once the routes are fixed as the routes have few, if any, visits to the recharging stations.

4.4. The Impact of Full Recharges

This section analyzes the impact of always requiring vehicles to recharge to full when visiting a station. This problem variant is the simplest of four cases studied by Desaulniers et al. (2016). The modifications to the BCP algorithm are described in Appendix B. Table 3 shows the results for the instances with one charger at every station. The previous section found that replacing the linear recharging function with a piecewise-linear function that has slower pieces significantly delays the vehicles, making many routes time-infeasible. Full recharging makes even more routes time-infeasible, and sometimes whole instances infeasible, because the vehicles must discharge and recharge through the slowest piece in the higher states of charge. In contrast, vehicles can maintain their states of charge in the fastest piece when partial recharges are allowed. The key finding is that slow recharging delays the vehicles, making the problem easier to solve because the time constraints are more likely to be violated and hence the search space is smaller.

5. Conclusion and Future Work

This paper proposes an exact method for solving the EVRPTW-PLR-CRS. The EVRPTW-PLR-CRS includes a more realistic non-linear recharging procedure and recharging stations with a limited number of chargers which must be shared among the vehicles. To ensure that the vehicles achieve their collective goal of delivering the packages to all customers on time, the vehicles are required to self-organize at the recharging stations. This interaction leads to a novel scheduling structure rarely seen in vehicle routing problems.

This paper develops a BCP algorithm for the EVRPTW-PLR-CRS that exploits the best available technologies for routing and scheduling by distributing the routing to integer

programming and the scheduling to constraint programming. The method deploys a very-recent generic form of logic-based Benders decomposition which can separate combinatorial Benders cuts overarching all vehicles. The framework makes solving vehicle routing problems with synchronization relatively easy, as otherwise, handling the interaction/synchronization between vehicles is extremely difficult and usually problem-specific.

Previous work on automatic logic-based Benders decomposition require the integer programming problem and the constraint programming problem to communicate through a set of identical variables. This paper develops a new translation that enables the checking subproblem to use the giant tour formulation more friendly to constraint programming technology. The paper also proposes two simple lifting techniques to strengthen the combinatorial Benders cuts. The hybrid BCP algorithm solves 299 of all 448 instances (51%) and 38 of the 112 100-customer instances (34%).

Lam and Van Hentenryck (2017) speculate that the generic form of logic-based Benders decomposition is ineffective in column generation solvers of classical vehicle routing problems because the intra-route constraints (e.g., time windows) are almost always binding and suggest that the approach may be useful in rich vehicle routing problems when synchronization constraints are seldom binding. This paper validates their hypothesis.

As the generic form of logic-based Benders decomposition is only invented recently, many directions for future work are available. In particular, developing a deeper connection between the master, pricing and checking subproblems is a promising topic. Presently, the master problem completely ignores the scheduling constraints in the absence of the combinatorial Benders cuts. In many studies using logic-based Benders decomposition, the branch-and-cut master problem is equipped with a relaxation of the omitted constraints to drive it towards candidate solutions that are more likely to be feasible in the checking subproblem. In the context of branch-and-price, this relaxation in the master problem is associated with a new dual variable that imposes additional structure in the pricing subproblem that destroys the pricing algorithm. Hence, it is extremely difficult to implement a relaxation of the scheduling constraints in the master problem even though the relaxations themselves are well-known.

Appendix A. An Example of the EVRPTW-PLR-CRS

Figure A.5 illustrates the routes of two vehicles visiting three customers with a detour via a common recharging station. One vehicle follows the solid arcs on the top and the other vehicle follows the dashed arcs on the bottom. Both recharging and discharging rates are equal to 1 unit of energy per timestep. The time window of each customer is given on the right. All customers have zero service duration. The two numbers above (resp. below) a vertex provide the time of starting service and the charge of the top (resp. bottom) vehicle. The arrows at the stations indicate the time and charge before and after recharging. The number on each arc is the travel time and the identical energy consumption.

The top route begins at the depot D^+ at timestep 0 with 15 units of energy in its battery. It consumes 10 timesteps and 10 units of energy to reach station S at timestep 10. It then recharges until timestep 12 to gain 2 units of energy. It spends 2 timesteps and 2 units of

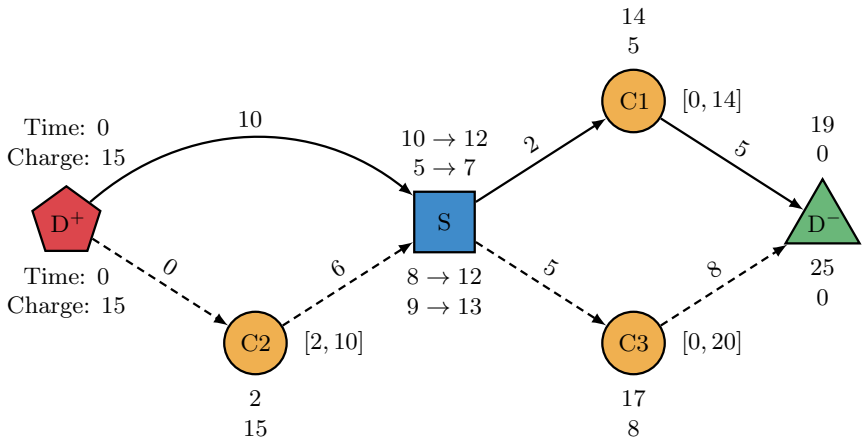

Figure A.5: The routes of two vehicles concurrently recharging at a station.

energy to arrive at customer C1 at timestep 14 with 5 units of energy remaining. It finally finishes its route at the depot D^- at timestep 19 with no energy remaining.

The bottom route also begins at the depot D^+ at timestep 0 with 15 units of energy. It delivers a package to customer C2 at the same location, costing 0 units of time and energy, but it is delayed from unloading until the time window opens at timestep 2. It then consumes 6 timesteps and 6 units of energy to reach station S at timestep 8 with 9 units of energy remaining. It recharges for 4 timesteps and finishes with a total of 13 units of energy. It spends 5 timesteps traveling to customer C3 and arrives at timestep 17 with 8 units of energy leftover. It ends at the depot D^- at timestep 25 with an empty battery.

After arriving at station S, the top vehicle requires 7 units of energy for the remainder of its route. It arrives at the station with 5 units of energy, which means that it must recharge for 2 timesteps to acquire the 7 units of energy to complete its route. The vehicle arrives at timestep 10 and must depart by timestep 12 to reach customer C1 before the time window closes. Hence, it must recharge during timesteps 10 and 11.

The bottom vehicle arrives at station S with 9 units of energy and requires a total of 13 units of energy to complete its route. Hence, it must recharge for 4 timesteps. It arrives at the station at timestep 8 and must depart by timestep 15 to arrive at customer C3 before the time window closes at timestep 20. Because it must recharge for 4 timesteps within this time interval, it has a slack of 3 timesteps for delaying the recharge.

Assuming that the station has a single charger, Figure A.6 demonstrates that regardless of whether the bottom vehicle recharges at the earliest (left) or latest (right) opportunity, no feasible schedule that satisfies the arrival times at customers C1 and C3 exists.

Appendix B. Modifications to Full Recharges

This section presents modifications to always fully recharge a vehicle. In the labeling algorithm (Algorithm 2), Lines 10 to 29 are replaced with Algorithm 4. If the destination is a station (Line 4), for each piece $p \in \mathcal{P}$ in order (Line 5), Line 6 calculates the energy recharged in the piece, Line 7 increases the energy available by this amount and Line 8 delays

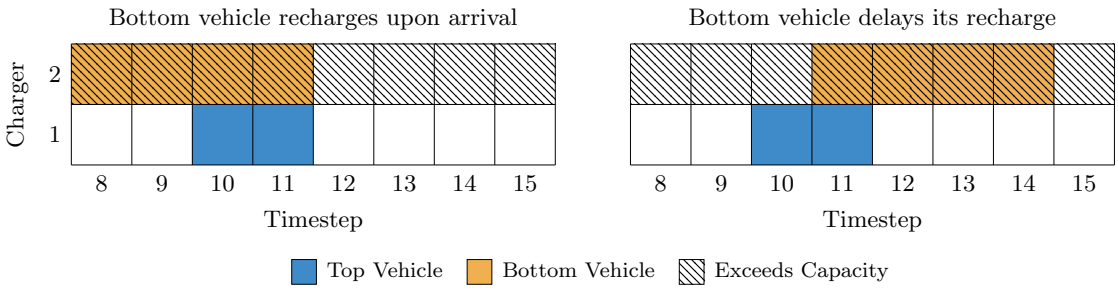

Figure A.6: Scheduling the two vehicles in Figure A.5 to recharge at station S. No feasible schedule exists because the two vehicles are trying to use one charger simultaneously.

the vehicle by the time to recharge this amount.

Algorithm 4 Changes to the labeling algorithm to require full recharges.

$l_j^{\text{energy}} \leftarrow l_i^{\text{energy}} - t_{i,j}^{\text{energy}}$
if $l_j^{\text{energy}} < 0 \vee l_j^{\text{time}} > t_j^{\text{close}}$ **then**
    exit (extension is infeasible)
if $j \in \mathcal{S}'$ **then**
    for $p = 1, \ldots, P$ **do**
        $\Delta \leftarrow \max\{\sigma_p - \max\{\sigma_{p-1}, l_j^{\text{energy}}\}, 0\}$
        $l_j^{\text{energy}} \leftarrow l_j^{\text{energy}} + \Delta$
        $l_j^{\text{time}} \leftarrow l_j^{\text{time}} + \Delta/\theta_p$
if $l_j^{\text{time}} > t_j^{\text{close}}$ **then**
    exit (extension is infeasible)

In the checking subproblem, Constraints (B.1) is added to enforce full recharges. This constraint states that the energy level must be full when departing a station.

$$e_i' = T^{\text{energy}} \qquad \forall i \in \mathcal{S}'. \quad \text{(B.1)}$$

Funding Sources

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

Andersen, E.D., 2014. How to use Farkas' lemma to say something important about infeasible linear problems. Technical Report. MOSEK. `https://docs.mosek.com/whitepapers/infeas.pdf`.

Ascheuer, N., Fischetti, M., Grötschel, M., 2000. A polyhedral study of the asymmetric traveling salesman problem with time windows. Networks 36, 69–79.

Baldacci, R., Mingozzi, A., Roberti, R., 2011. New route relaxation and pricing strategies for the vehicle routing problem. Operations Research 59, 1269–1283.

Bruglieri, M., Mancini, S., Pisacane, O., 2019. The green vehicle routing problem with capacitated alternative fuel stations. Computers & Operations Research 112, 104759.

Bruglieri, M., Mancini, S., Pisacane, O., 2021. A more efficient cutting planes approach for the green vehicle routing problem with capacitated alternative fuel stations. Optimization Letters 15, 2813–2829.

Codato, G., Fischetti, M., 2006. Combinatorial Benders' cuts for mixed-integer linear programming. Operations Research 54, 756–766.

Costa, L., Contardo, C., Desaulniers, G., 2019. Exact branch-price-and-cut algorithms for vehicle routing. Transportation Science 53, 946–985.

Davies, T.O., Gange, G., Stuckey, P.J., 2017. Automatic logic-based Benders decomposition with MiniZinc, in: Thirty-First AAAI Conference on Artificial Intelligence, pp. 787–793.

Desaulniers, G., Desrosiers, J., Solomon, M.M. (Eds.), 2005. Column Generation. Springer.

Desaulniers, G., Desrosiers, J., Spoorendonk, S., 2011. Cutting planes for branch-and-price algorithms. Networks 58, 301–310.

Desaulniers, G., Errico, F., Irnich, S., Schneider, M., 2016. Exact algorithms for electric vehicle-routing problems with time windows. Operations Research 64, 1388–1405.

Desaulniers, G., Gschwind, T., Irnich, S., 2020. Variable fixing for two-arc sequences in branch-price-and-cut algorithms on path-based models. Transportation Science 54, 1170–1188.

Drexl, M., 2012. Synchronization in vehicle routing–a survey of VRPs with multiple synchronization constraints. Transportation Science 46, 297–316.

El Hachemi, N., Gendreau, M., Rousseau, L.M., 2011. A hybrid constraint programming approach to the log-truck scheduling problem. Annals of Operations Research 184, 163–178.

Feydy, T., Stuckey, P.J., 2009. Lazy clause generation reengineered, in: Gent, I.P. (Ed.), Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP2009), Springer. pp. 352–366.

Froger, A., Jabali, O., Mendoza, J.E., Laporte, G., 2021. The electric vehicle routing problem with capacitated charging stations. Transportation Science .

Froger, A., Mendoza, J.E., Jabali, O., Laporte, G., 2019. Improved formulations and algorithmic components for the electric vehicle routing problem with nonlinear charging functions. Computers & Operations Research 104, 256–294.

Gamrath, G., Anderson, D., Bestuzheva, K., Chen, W.K., Eifler, L., Gasse, M., Gemander, P., Gleixner, A., Gottwald, L., Halbig, K., Hendel, G., Hojny, C., Koch, T., Le Bodic, P., Maher, S.J., Matter, F., Miltenberger, M., Mühmer, E., Müller, B., Pfetsch, M.E.,

Schlösser, F., Serrano, F., Shinano, Y., Tawfik, C., Vigerske, S., Wegscheider, F., Weninger, D., Witzig, J., 2020. The SCIP Optimization Suite 7.0. Technical Report. Optimization Online. http://www.optimization-online.org/DB_HTML/2020/03/7705.html.

Gange, G., Berg, J., Demirović, E., Stuckey, P.J., 2020. Core-guided and core-boosted search for CP, in: Hebrard, E., Musliu, N. (Eds.), Proceedings of the 16th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR2019), Springer. pp. 205–221.

Heinz, S., 2018. Presolving techniques and linear relaxations for cumulative scheduling. Ph.D. thesis. Technische Universität Berlin. Berlin.

Hooker, J.N., Ottosson, G., 2003. Logic-based Benders decomposition. Mathematical Programming 96, 33–60.

Irnich, S., Desaulniers, G., 2005. Shortest Path Problems with Resource Constraints. Springer. chapter 2. Column generation, pp. 33–65.

Jepsen, M., Petersen, B., Spoorendonk, S., Pisinger, D., 2008. Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research 56, 497–511.

Jungwirth, A., Desaulniers, G., Frey, M., Kolisch, R., 2020. Exact branch-price-and-cut for a hospital therapist scheduling problem with flexible service locations and time-dependent location capacity. https://mediatum.ub.tum.de/1427721.

Keskin, M., Laporte, G., Çatay, B., 2019. Electric vehicle routing problem with time-dependent waiting times at recharging stations. Computers & Operations Research 107, 77–94.

Keskin, M., Çatay, B., Laporte, G., 2021. A simulation-based heuristic for the electric vehicle routing problem with time windows and stochastic waiting times at recharging stations. Computers & Operations Research 125, 105060.

Kilby, P., Shaw, P., 2006. Vehicle routing. Elsevier. chapter 23. Handbook of constraint programming, pp. 799–834.

Kohl, N., Desrosiers, J., Madsen, O.B.G., Solomon, M.M., Soumis, F., 1999. 2-path cuts for the vehicle routing problem with time windows. Transportation Science 33, 101–116.

Kullman, N.D., Goodson, J.C., Mendoza, J.E., 2021. Electric vehicle routing with public charging stations. Transportation Science 55, 637–659.

Lam, E., Gange, G., Stuckey, P.J., Van Hentenryck, P., Dekker, J.J., 2020. Nutmeg: A MIP and CP hybrid solver using branch-and-check. SN Operations Research Forum 1, 22.

Lam, E., Van Hentenryck, P., 2016. A branch-and-price-and-check model for the vehicle routing problem with location congestion. Constraints 21, 394–412.

Lam, E., Van Hentenryck, P., 2017. Branch-and-check with explanations for the Vehicle Routing Problem with Time Windows, in: Beck, J.C. (Ed.), Proceedings of the 23rd International Conference on Principles and Practice of Constraint Programming (CP2017), Springer. pp. 579–595.

Laporte, G., Nobert, Y., Desrochers, M., 1985. Optimal routing under capacity and distance restrictions. Operations Research 33, 1050–1073.

Lee, C., 2020. An exact algorithm for the electric-vehicle routing problem with nonlinear charging time. Journal of the Operational Research Society , 1–24.

Lübbecke, M.E., Desrosiers, J., 2005. Selected topics in column generation. Operations Research 53, 1007–1023.

Lysgaard, J., Letchford, A.N., Eglese, R.W., 2004. A new branch-and-cut algorithm for the capacitated vehicle routing problem. Mathematical Programming 100, 423–445.

Marques Silva, J.a.P., Sakallah, K.A., 1996. GRASP-a new search algorithm for satisfiability, in: Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design, IEEE Computer Society. pp. 220–227.

Montoya, A., Guéret, C., Mendoza, J.E., Villegas, J.G., 2017. The electric vehicle routing problem with nonlinear charging function. Transportation Research Part B: Methodological 103, 87–110.

Ohrimenko, O., Stuckey, P.J., Codish, M., 2009. Propagation via lazy clause generation. Constraints 14, 357–391.

Schneider, M., Stenger, A., Goeke, D., 2014. The electric vehicle-routing problem with time windows and recharging stations. Transportation Science 48, 500–520.

Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G., 2010. Explaining the cumulative propagator. Constraints 16, 250–282.

Solomon, M.M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35, 254–265.

Vigo, D., Toth, P. (Eds.), 2014. Vehicle Routing: Problems, Methods, and Applications. MOS-SIAM Series on Optimization. second ed., Society for Industrial and Applied Mathematics.