Electric Vehicle Scheduling with Capacitated Charging Stations and Partial Charging

Electric Vehicle Scheduling with Capacitated Charging Stations and Partial Charging

Marelot H. de Vos[1], Rolf N. van Lieshout[2,3], and Twan Dollevoet[4]

[1]ORTEC Data Science & Consulting, Zoetermeer, The Netherlands

[2]Corresponding author. Email: r.n.v.lieshout@tue.nl

[3]Department of Operations, Planning, Accounting, and Control, School of Industrial Engineering, Eindhoven University of Technology, The Netherlands

[4]Econometric Institute and ECOPT, Erasmus University Rotterdam, The Netherlands

August 1, 2022

Abstract

This paper considers the scheduling of electric vehicles in a public transit system. Our main innovation is that we take into account that charging stations have limited capacity, while also considering partial charging. To solve the problem, we expand a connection-based network in order to track the state of charge of vehicles and model recharging actions. We then formulate the electric vehicle scheduling problem as a path-based binary program, whose linear relaxation we solve using column generation. We find integer feasible solutions using two heuristics: price-and-branch and truncated column generation, including acceleration strategies. We test the approach using data of the concession Gooi en Vechtstreek in the Netherlands, containing up to 816 trips. The truncated column generation outperforms the other heuristic, and solves the entire concession within 28 hours of computation time with an optimality gap less than 3.5 percent.

Keywords: Electric Vehicles, Bus Scheduling, Column Generation, Discretization.

1 Introduction

The benefits of electric buses are undisputed: replacing conventional combustion engine buses by electric buses drastically reduces noise, pollution and greenhouse gas emissions. For these reasons, many public transit operators have started electrifying their fleets. However, the introduction of electric buses introduces new complexities in the transit planning chain, since limited battery capacity requires electric buses to recharge during the day. In order to prevent buses from depleting their batteries while minimizing operating costs and energy consumption, it is crucial to time these recharging actions carefully, establishing a clear need for solution methods that incorporate charging to support bus companies in these decisions.

Naturally, the demand for algorithmic support for the planning of electric buses has spurred the interest in the Electric Vehicle Scheduling Problem (E-VSP), which is the problem of constructing feasible electric bus duties to cover a set of timetabled trips. A recent survey on electric bus planning and scheduling identified over 20 papers on (variants of) the E-VSP, primarily from the last three years (Perumal, Lusby, & Larsen, 2022). The innovation of our paper is that we propose a solution approach that is capable of solving large instances of the E-VSP while considering both the capacity of charging stations and partial (non-linear) charging, bridging the gap between theory and practice.

A number of other papers on the E-VSP consider charging station capacity (J. Q. Li, 2014; L. Li, Lo, & Xiao, 2019; Rinaldi, Picarelli, D'Ariano, & Viti, 2020; Tang, Lin, & He, 2019; Wu, Lin, Liu, & Jin, 2022). However, these papers either consider battery swapping or another form of constant-time charging. Another stream of literature considers partial charging (X. Li et al., 2020; Olsen & Kliewer, 2020; Van Aken & Hiemstra, 2020; Van Kooten Niekerk, Van den Akker, & Hoogeveen, 2017; Wen, Linde, Ropke, Mirchandani, & Larsen, 2016). Literature on the E-VSP with both partial charging and capacitated charging stations is scarce (Janovec & Kohání, 2019; Posthoorn, 2016; Zhang, Wang, & Qu, 2021). Janovec and Kohání (2019) only consider relatively small instances. Posthoorn (2016) tests the proposed approach on larger (single-depot) instances, but does not report optimality gaps or computation times. Zhang et al. (2021) solve

medium-sized instances up to an optimality gap of 1%, but consider only a single depot that also serves as the charging station. Conversely, in this paper, we find provably high-quality solutions for large instances with up to 816 trips of a rich variant of the E-VSP with partial charging, multiple capacitated charging stations, multiple depots and multiple vehicle types.

Our solution approach is based on a discretization of the battery energy levels, which we combine with a connection-based network with nodes representing trips and charging actions. Every path in the resulting so-called *primal* network represents a feasible bus duty, respecting both the compatibility of trips and the battery capacity. Note that the converse is not true: there may exist feasible vehicle duties that are not represented in the primal network, because we connect nodes using a conservative rounding scheme to ensure feasibility. However, we are able to use a relatively fine discretization, achieving a good trade-off between solution time and solution quality. In addition, although we do not consider this in our numerical experiments, the proposed discretization scheme perfectly lends itself to non-linear charging functions and is therefore widely applicable.

Using the developed network structure, we formulate the problem as a path-based binary program with side constraints, whose linear relaxation can be solved with column generation. Due to the construction of the network, the pricing problem corresponds to a standard shortest path problem. To find integer solutions, we consider two heuristics: price-and-branch and truncated column generation. In price-and-branch, only the linear relaxation is solved using column generation, after which all generated paths are fed to a commercial MIP solver that optimizes over this given subset of all paths. Truncated column generation can be viewed as a diving heuristic in the branch-and-bound tree, where all nodes are solved using column generation: paths are fixed iteratively until the solution is integer feasible. We also develop acceleration strategies to reduce the computation time of the heuristics.

In general, a disadvantage of optimization models based on discretization is that the obtained dual bound is only valid for the discretized representation of the problem, and is therefore not a true bound of the underlying problem. Inspired by Boland, Hewitt, Marshall, and Savelsbergh

(2017), we propose to find true lower bounds for the E-VSP by solving a linear relaxation on a *dual* network, containing the same nodes as the primal network, but where nodes are connected using an *optimistic* rounding scheme. Every feasible vehicle duty corresponds to a path in this modified network, such that this procedure yields a lower bound that is valid irrespective of the discretization. Note that where Boland et al. (2017) apply this principle to time-discretized networks, we apply it to a network that is discretized in two dimensions: time and the battery energy levels.

We test our approach using real-life timetable data from the bus concession Gooi en Vechtstreek in the Netherlands, which consists of 816 trips connecting five medium-sized cities south-east of Amsterdam. We also generate smaller instances by taking (random) subsets of all trips. On the smaller instances, truncated column generation achieves an optimality gap smaller than 1.5%, outperforming price-and-branch. When paired with a dedicated acceleration strategy, we are able to solve the entire concession using truncated column generation up to an optimality gap of 3.4%. We also perform a sensitivity analysis, which shows that our level of discretization is adequate: using finer discretizations leads to a significant increase in computation time, without resulting in a significant decrease in costs.

The remainder of this paper is organized as follows: Section 2 gives a detailed description of the problem considered in this paper. Thereafter, Section 3 discusses literature related to this research. In Section 4, we present our solution approach, discussing both the network structure and the column generation based heuristics. In Section 5, we present numerical results, including sensitivity analyses. Lastly, we conclude in Section 6.

2 Problem Description

The problem that we consider in this paper is a multi-depot vehicle scheduling problem that includes range constraints of the vehicles in a heterogeneous vehicle fleet, capacitated charging stations and partial charging. In this section, we discuss these elements in more detail.

The core of the considered problem is the classical (multi-depot) vehicle scheduling problem

(MD-)VSP (Bunte & Kliewer, 2009). Here, the input is a set of timetabled trips, which should all be performed by a single vehicle. Every trip has fixed starting and ending locations, as well as fixed starting and ending times. A pair of trips (a, b) is called *compatible* if their starting and ending times and locations are such that trip b can be performed after trip a, potentially after an empty or *deadhead* trip between the ending location of a and the starting location of b. Every vehicle should be assigned a feasible duty, starting at (one of) the depot(s), performing a sequence of compatible trips, and returning to the (same) depot.

The problem considered in this paper is an extension of the MD-VSP with range constraints, (partial) recharging and capacitated charging stations. We assume that every electric vehicle is fully charged at the beginning of the day and that the energy consumption of every trip is known. Evidently, vehicles can only operate as long as their *state of charge* (SoC) is strictly positive. *Charging actions* may be scheduled at specified charging stations with given capacities. We do not require that every charging action fully recharges a vehicle's battery, i.e. we allow for partial charging. The duration of a charging activity correlates positively with the increase in the SoC, according to a known *charging function*. The capacity of charging stations implies that only a limited number of charging actions can be scheduled simultaneously at each charging station. In addition, we also consider a heterogeneous fleet, where the battery limit, charging function, and the energy consumption rate are allowed to differ per type of vehicle. Finally, we assume that the objective is to minimize the Total Costs of Ownership, including both fixed costs (investment costs for each vehicle) and operational costs (variable costs per kilometer to account for crew, energy consumption and maintenance).

3 Literature Review

For a general review on electric vehicle scheduling and related problems, we refer to Perumal et al. (2022). Here, we limit ourselves to discussing research on electric vehicle scheduling that considers the capacity of charging stations and/or partial charging. An overview of the included features in the discussed literature is presented in Table 3.1.

Table 3.1: Overview of included aspects in papers on the E-VSP. Abbrevations: CG=Column Generation (incl. branch-and-price and CG-based heuristics), MIP=Mixed Integer Programming, MH=Metaheuristic.

	Partial Charging	Multiple Depots	Multiple Vehicle Types	Charging Station Capacity	Solution Method	Number of Trips
J. Q. Li (2014)				✓	CG	947
Posthoorn (2016)	✓			✓	CG	709
Wen et al. (2016)	✓	✓			MH	500
Van Kooten Niekerk et al. (2017)	✓				CG	543
Janovec and Kohári (2019)	✓			✓	MIP	160
L. Li et al. (2019)		✓	✓	✓	MIP	288
Tang et al. (2019)				✓	CG	96
X. Li et al. (2020)	✓	✓			MH	867
Olsen and Kliewer (2020)	✓	✓			MH	10,710
Rinaldi et al. (2020)			✓	✓	MIP	1008
Van Aken and Hiemstra (2020)	✓	✓	✓		CG	1,200
Zhang et al. (2021)	✓			✓	CG	160
Wu et al. (2022)		✓		✓	CG	400
This paper	✓	✓	✓	✓	CG	816

3.1 Capacitated Charging Stations

J. Q. Li (2014) studies the E-VSP with battery swapping (or equivalently, fast-charging), taking the capacity of the charging station into account. In this setting, charging a vehicle takes a constant time. To solve the problem, the author develops exact and heuristic algorithms based on column generation. L. Li et al. (2019) deals with scheduling a mixed fleet of electric and conventional buses. It is assumed that full energy is restored after each refueling, which takes a constant time of 30 minutes, i.e. partial charging is not considered. The authors formulate the problem as a mixed-integer programming (MIP) model and solve instances with 288 trips and two depots using a commercial solver. Tang et al. (2019) investigate the scheduling of electric buses in a stochastic setting, where the aim is to find schedules robust against varying traffic conditions. The authors include charging station capacity, but only consider fast charging. Using branch-and-price, problem instances with up to 96 trips are solved, both in static and in dynamic fashion. Rinaldi et al. (2020) propose a MIP model to schedule a mixed-fleet of electric and diesel buses, assuming fast-charging. The authors also develop an ad-hoc decomposition scheme, which is tested on instances with up to 1008 trips. Wu et al. (2022) propose a branch-

and-price scheme to solve the E-VSP with capacitated charging stations, minimizing both costs and the overall peak load on the energy grid. The authors use the epsilon-constraint method to find (approximate) Pareto-efficient solutions with respect to the two objectives for instances with up to 400 trips.

3.2 Partial Charging

Wen et al. (2016) develop a MIP model and an Adaptive Large Neighborhood Search heuristic to solve the E-VSP with partial charging. The MIP can solve instances with 30 trips, and the heuristic with up to 500 trips. Olsen and Kliewer (2020) extend this heuristic to allow for non-linear charging processes and analyze the impact of assuming a constant charging time and/or a linear charging process on large instances with thousands of trips. An alternative heuristic approach is taken by X. Li et al. (2020), who develop an adaptive genetic algorithm, which is used to solve instances with up to 867 trips. Van Kooten Niekerk et al. (2017) present two MIP models for the E-VSP with partial charging. The first model assumes a linear charging process, so that the SoC can be tracked with continuous variables. In the second model this assumption is relaxed, which requires the SoC to be discretized. The authors apply column generation based heuristics to solve instances with 543 trips using the second model. The first model is only solvable for small instances. A similar discretization approach is taken by Van Aken and Hiemstra (2020), who, besides partial charging, also consider multiple depots and bus types, and are able to solve instances containing up to 1,200 trips.

3.3 Capacitated Charging Stations and Partial Charging

Posthoorn (2016) considers the E-VSP with partial charging and a single charging station with a limited capacity. The author discretizes the SoC and solves the linear relaxation using column generation. Subsequently, the generated paths are included in a MIP to find integer solutions. The approach is applied to instances with up to 709 trips. The discretization is relatively coarse and no gaps or computation times are reported. Janovec and Kohání (2019) develop an arc-

based MIP formulation for the E-VSP with partial charging and capacitated charging stations. The authors consider instances with up to 160 trips with nine buses and three to six chargers. Furthermore, for all instances, the results with electric buses are the same as with diesel buses, which suggests that the range and charging capacity constraints may not be restrictive. Zhang et al. (2021) study electric vehicle scheduling with partial (non-linear) charging from a single terminal, which also serves as the (capacitated) charging station. In addition, the authors also consider the impact of the schedule on battery-aging. Instances with up to 160 trips are solved using a branch-and-price algorithm.

4 Methodology

In order to solve the E-VSP, we first formulate the problem as a set covering problem with additional constraints. We then develop a column generation algorithm to solve its LP-relaxation. Finally, we present two heuristics, based on column generation, to obtain feasible solutions.

4.1 Mathematical Formulation

Our mathematical formulation is based on the set-partitioning model for the VSP as explained by Bunte and Kliewer (2009). In this formulation, the columns correspond to feasible vehicle duties, also called *paths*, which are sequences of compatible trips. Given that we consider electric vehicles, these paths include recharging actions as well. In our formulation, we consider set covering instead of set partitioning, since the set covering formulation is claimed to be numerically more stable (Barnhart, Johnson, Nemhauser, Savelsbergh, & Vance, 1998). Hence, each trip should be serviced by at least one, instead of precisely one vehicle. Without loss of generality, a double trip can be deleted from a vehicle duty without increasing the costs, and therefore an optimal solution to the set covering formulation is also optimal for the set partitioning formulation.

Moreover, since we solve the E-VSP, charging activities should be taken into account. Similar to J. Q. Li (2014), we discretize the time horizon into time blocks $\mathcal{B}$ with a fixed length l. These

time blocks are created to track the availability of charging stations over time and to incorporate the limited capacity of the charging stations. We assume that a vehicle occupies the charging station either during the entire block, or not at all in this block. Let t_b represent the starting time of time block $b \in \mathcal{B}$. A time block $b \in \mathcal{B}$ represents the time interval $[t_b, t_b + l)$.

We can now formulate the E-VSP. The set $\mathcal{T}$ represents all trips that should be serviced, while the set $\mathcal{R}$ contains all charging stations. The parameter M_r denotes the capacity of charging station $r \in \mathcal{R}$. We define the set $\mathcal{P}$ containing all possible paths. A path $p \in \mathcal{P}$ encodes a feasible vehicle duty in which trips to service and charging actions are included. We define a binary decision variable x_p that indicates whether path $p \in \mathcal{P}$ is selected in the solution. The parameter c_p represents the costs of path $p \in \mathcal{P}$, and each coefficient $a_{i,p}$ is 1 if trip $i \in \mathcal{T}$ is included in path $p \in \mathcal{P}$, and 0 otherwise. Similarly, the coefficient $u_{r,b,p}$ is 1 if charging station $r \in \mathcal{R}$ is visited during time block $b \in \mathcal{B}$ in path $p \in \mathcal{P}$. We use the following formulation for the E-VSP:

$$\min \quad \sum_{p \in \mathcal{P}} c_p x_p, \tag{1}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}} a_{i,p} x_p \geq 1 \qquad \forall\, i \in \mathcal{T}, \tag{2}$$

$$\sum_{p \in \mathcal{P}} u_{r,b,p} x_p \leq M_r \qquad \forall\, r \in \mathcal{R},\ b \in \mathcal{B}, \tag{3}$$

$$x_p \in \{0, 1\} \qquad \forall p \in \mathcal{P}. \tag{4}$$

The Objective (1) minimizes the total costs. Constraint (2) guarantees that each trip $i \in \mathcal{T}$ is executed by at least one vehicle. Constraint (3) is added to ensure that the capacity of each charging station $r \in \mathcal{R}$ is not exceeded in any of the time blocks $b \in \mathcal{B}$. Lastly, Constraint (4) provides the range of the decision variables.

4.2 Column Generation Algorithm

Since the path-based formulation (1)-(4) has exponentially many variables, we develop two heuristics that are based on column generation. In column generation, a solution to a linear program, called the *master problem* (MP), is found by iteratively solving a *restricted master problem* (RMP) and a *pricing problem*. In short, the MP is the LP-relaxation of the set covering model (1)-(4). The RMP is similar to the MP but uses only a subset of paths, denoted by $\mathcal{P}'$. Before the start of the column generation process, the set $\mathcal{P}'$ is initialized with paths that allow a feasible solution to the RMP. Afterward, in every iteration, the RMP is solved and the values of the dual variables are used as input for the pricing problem. The pricing problem searches for new paths with negative reduced costs, since these can improve the objective value. The path with the most negative reduced costs is added to the set $\mathcal{P}'$ used in the RMP. If the pricing problem cannot provide a path with negative reduced costs, the MP is solved to optimality and the column generation process is terminated. For a more detailed explanation of column generation, we refer to Desrosiers and Lübbecke (2005).

In the remainder of this section, we discuss the pricing problem of the column generation approach for the E-VSP in detail. Additionally, we discuss how the set $\mathcal{P}'$ is initialized. We then discuss how lower bounds on the optimal solution value can be obtained. Finally, we develop two column generation based heuristics that are used to obtain a feasible solution for the E-VSP.

4.2.1 Pricing Problem

In every iteration, the pricing problem searches for new variables with negative reduced costs based on the current values of the duals. From the RMP, we obtain optimal values for the dual variables σ_i for all trips $i \in \mathcal{T}$ and $\gamma_{r,b}$ for all charging stations $r \in \mathcal{R}$ and time blocks $b \in \mathcal{B}$. Using this dual information from the RMP, the reduced costs corresponding to path p can be calculated as follows:

$$RC(x_p) = c_p - \sum_{i \in \mathcal{T}} a_{i,p}\sigma_i - \sum_{r \in \mathcal{R}} \sum_{b \in \mathcal{B}} u_{r,b,p}\gamma_{r,b}. \tag{5}$$

The aim of the pricing problem is to find the variable corresponding to a path with the lowest reduced costs. In order to find the path $p \in \mathcal{P}$ with the lowest reduced cost, we solve a shortest path problem in a suitably chosen network, which we now describe in full detail.

Network Structure We construct a connection-based network which allows to create feasible vehicle duties. In particular, we create a separate network for each combination of a vehicle type and depot, as proposed by Gintner, Kliewer, and Suhl (2005). This allows us to ensure that each vehicle starts and ends at the same depot and to incorporate different characteristics for each vehicle type. We define $\mathcal{K}$ as the set of networks that are created. The aim is to find a vehicle duty with lowest reduced costs in each network $k \in \mathcal{K}$ separately.

We now describe the construction of the network $G^k(\mathcal{N}^k, \mathcal{A}^k)$ for a given combination $k \in \mathcal{K}$ of a vehicle type and a depot. We take the SoC of the vehicle into account by discretizing the possible SoC values and tracking the SoC of the vehicle along the path. Thus, the nodes in this network represent the depot, combinations of trips and SoC values, or combinations of charging stations, time blocks, and SoC values. Each compatible connection is explicitly modeled using a conservative rounding scheme for the SoC values. This ensures that all paths in the network correspond to a feasible vehicle duty. Therefore, we refer to this network as the *primal* network. However, as a consequence, some feasible vehicle duties cannot be represented as a path in our network. we introduce the concept of a *dual* network and this network can generate true lower bounds in Section 4.2.3. Furthermore, we study the impact of the discretization on the solution values in Section 5.4.2.

Nodes In each network G^k, we include a source node and sink node, denoted as d_k^σ and d_k^τ, respectively, that correspond to the depot. We include nodes for the timetabled trips in combination with discretized SoC values similar to Van Kooten Niekerk et al. (2017), to keep track of the SoC of vehicles along their path. Mathematically, let $\mathcal{T}^k$ and $\mathcal{S}^k$ be the sets of trips and all possible SoC values for network k, respectively. Since we discretize the SoC, $\mathcal{S}^k$ has a finite number of elements. Let $s_k^{\min}$ represent the minimum allowed SoC value. For trip i, we

define the set $\mathcal{S}_i^k$ that includes all SoC values $s \in \mathcal{S}^k$ that are at least $s_k^{\min}$ plus the SoC required for executing trip i, represented by f_i. This results in the node set

$$\mathcal{N}_k^{\text{trip}} = \{(i, s) | i \in \mathcal{T}^k, s \in \mathcal{S}_i^k\}.$$

For each node, the value s represents the SoC of the vehicle at the moment it departs from the start location of trip i, hence at the beginning of trip i.

Additionally, to be able to model charging activities, we use copies of charging actions in combination with possible SoC values as nodes. We mean by *charging action* the charging of a vehicle at a charging station within a specific time block $b \in \mathcal{B}$. Let the set $\mathcal{R}^k$ correspond to all charging stations suitable for network k. As the SoC value at a specific node influences the compatibility to other nodes in the network, we combine these nodes with discretized SoC values. We include nodes per charging action for each possible value of the SoC $s \in \mathcal{S}^k$ that a vehicle has at the beginning of the charging action for all charging nodes. Since this SoC value represents the SoC before the charging, we disregard the nodes for a fully charged SoC, represented by s^{full}, since this value cannot be increased during a charging action. Thus, the nodes created for the charging actions can be summarized in the set

$$\mathcal{N}_k^{\text{charge}} = \{(r, b, s) | r \in \mathcal{R}^k, b \in \mathcal{B}, s \in \mathcal{S}^k \setminus \{s^{\text{full}}\}\}.$$

In conclusion, the nodes in network k are represented by the node set

$$\mathcal{N}^k = \{d_k^\sigma, d_k^\tau\} \cup \mathcal{N}_k^{\text{trip}} \cup \mathcal{N}_k^{\text{charge}}.$$

For the remainder of this paper, we call node n a *trip node* if $n \in \mathcal{N}_k^{\text{trip}}$ and a *charging node* if $n \in \mathcal{N}_k^{\text{charge}}$ in a particular network $k \in \mathcal{K}$.

Arcs The set of arcs $\mathcal{A}^k$ represent connections between nodes in the network. Because the time horizon and SoC values are discretized, a rounding scheme is needed. For bus companies,

it is important that an electric bus is always capable of finishing its duty and, thus, running out of battery must be prevented. Additionally, a bus arriving too early is preferred over that bus arriving too late. For these reasons, we apply a conservative rounding scheme. In particular, charging actions begin at the earliest in the first time interval $b \in \mathcal{B}$ that starts after the arrival of the vehicle. The vehicle idles in between its arrival and the start of the time block. Additionally, we round down the actual SoC value of a vehicle to the nearest SoC value $s \in \mathcal{S}^k$, which results in an underestimated SoC value. Consequently, it can occur that some feasible vehicle duties are excluded. However, the final schedule obtained using this conservative rounding scheme will certainly be feasible in practice.

Below, we briefly explain which arcs are created in network k. Here, we let $F^{\text{soc}}(\cdot)$ be a function that returns the nearest SoC value $s \in \mathcal{S}^k$ smaller than the SoC input. A more thorough description is provided in Appendix A.

- Given that vehicles leave the depot fully charged, we only include arcs from the source node d_k^σ to trip nodes. For these arcs, we take the SoC usage required for the deadheading trip from the depot to the start location of the corresponding trip into account, as well as a maximum deadheading time, if applicable.

- For trip nodes $n = (i, s)$, we include outgoing arcs to the sink node, to other trip nodes, and to charging nodes. An outgoing arc to the sink node is included if the node's SoC value is sufficient to execute the trip and then directly return to the depot. An outgoing arc to another trip node v is present if the trips are compatible and the SoC value s is sufficient to execute both trips, as well as the deadheading and idling between the trips, if applicable. Similarly, there is an outgoing arc from a trip node to a charging action v if the maximum deadheading and idling SoC usage and time are respected. For arcs towards a trip or charging node v, the SoC value s' of v must satisfy

$$s' = F^{\text{soc}}(s - f_i - \tau^{\text{soc}}_{(n,v)}),$$

where $\tau^{\text{soc}}_{(n,v)}$ represents the SoC required for the potential deadheading and idling between nodes n and v, if applicable.

- For charging nodes $n = (r, b, s)$, we include arcs to the sink node, to trip nodes, and to other charging nodes. We denote the amount of SoC recharged in node n by s^+_n. An outgoing arc to the sink node is included if the node's SoC value after recharging is sufficient to return to the depot, while this is not the case without recharging s^+_n. If it would be possible to return to the depot without the recharging of node n, vehicles could either return to the depot without recharging at all, or with a shorter recharging time. Outgoing arcs to trip nodes v are present if the vehicle can arrive timely at the start location of the trip. Moreover, the SoC value s' of the trip node must equal $s' = F^{\text{soc}}(s + s^+_n - \tau^{\text{soc}}_{(n,v)})$. Finally, an outgoing arc from one charging node to another is included if the corresponding charging stations coincide, the second charging node directly follows the first one in time, and $s' = F^{\text{soc}}(s + s^+_n) > s$.

Arc Costs The costs of a path $p \in \mathcal{P}$ can be distributed over the arcs. Each arc $(n, v) \in \mathcal{A}^k$ contains operational costs per driven kilometer and crew costs for deadheading and idling. These general costs for arcs are represented by $c^{\text{arc}}_{(n,v)}$. Arcs coming from the source node ($n = d^\sigma_k$) additionally include the investment costs corresponding to the vehicle of network k, represented by c^{invest}_k. Each arc that reaches a trip node includes operational costs per driven kilometer and the crew costs of the corresponding trip. Similarly, each arc that reaches a charging node, includes the operational costs of the corresponding charging action. Each arc that reaches a charging node coming from a trip node, includes an additional fixed penalty term c^{start}, which can be interpreted as starting costs for a charging activity. This penalty term should be small relative to the investment costs.

The reduced costs of a path $p \in \mathcal{P}$ can be distributed over the arcs as well. Besides the primal costs that are described above, we subtract the dual costs σ_i for all trip nodes $v = (i, s) \in \mathcal{N}^{\text{trip}}_k$ from the arc costs of all arcs (n, v) towards v, and we subtract the dual costs $\gamma_{r,b}$ for all charging

nodes $v = (r, b, s) \in \mathcal{N}_k^{\text{charge}}$ from the arc costs of all arcs (n, v).

Example An example of a resulting network is given in Figure 4.1. Here, we have two trips i and j that can be serviced by the vehicle type under consideration and a charging station r_1 that is available during two time blocks b_1 and b_2 between the end of trip i and the start of trip j. Both trips start at the depot. Trip i and j reduce the SoC by 40% and 80%, respectively. Trip i ends at a location from which the SoC is reduced by 20% to return to the depot. Trip j ends at the depot. Charging station r_1 is located at the end location of trip i and can increase the SoC in one time block with 20%, independently of the SoC value at the beginning of the charging action. We use SoC values between 0% to 100% in steps of 20%. Observe that some of the nodes and arcs cannot be included in a path that starts and ends at the depot. These nodes and arcs are depicted in a lighter color. In a preprocessing step, we remove these node and arcs from the network.

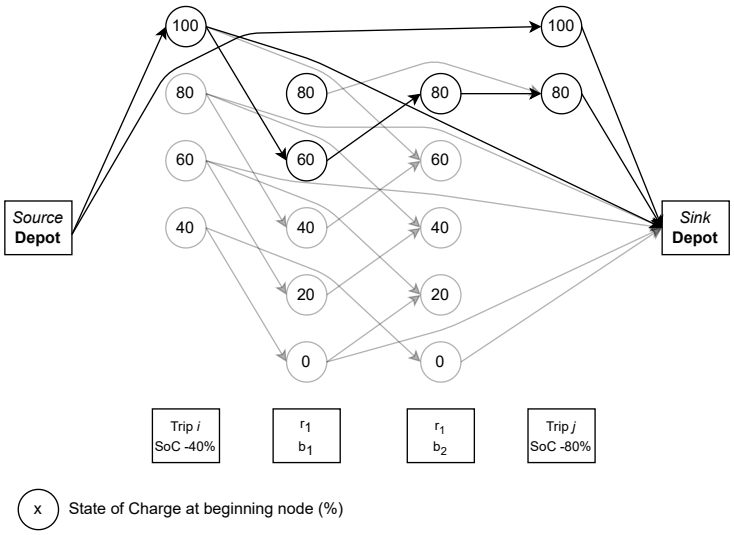

Figure 4.1: Network including all nodes and arcs before the preprocessing. The nodes and arcs with reduced opacity are not connected to the source node and/or sink node, and are removed in a preprocessing step.

4.2.2 Initialization

To initialize the column generation algorithm, we construct a feasible solution by including a separate path for each trip in $\mathcal{P}'$. In our test instances, this solution is always feasible and no

charging is necessary. In general, one can also avoid having to find a set of feasible paths for the initialization by using dummy variables in the trip covering constraints.

4.2.3 Lower bounds

The column generation algorithm terminates if paths with negative reduced costs can no longer be found. In that case, the MP has been solved to optimality, given the set $\mathcal{P}$ of columns under consideration. A lower bound on the MP's solution value can be obtained in each iteration of the column generation algorithm, if a value κ can be found that satisfies

$$\kappa \geq \sum_{p \in \mathcal{P}} x_p$$

for all optimal solutions of the MP. In that case, the value $z_{\text{RMP}} + \kappa c$ is a lower bound on the optimal solution value of the MP, where c is the lowest reduced costs over all paths $p \in \mathcal{P}$ and z_{RMP} is the optimal solution value of the RMP (see Desrosiers and Lübbecke (2005)).

As stated before, because of the conservative rounding scheme, there might be feasible vehicle duties that cannot be represented as a path in the networks defined above. Hence, the lower bound on the MP does not necessarily correspond to a true lower bound of the original problem.

To obtain a true lower bound, we construct an auxiliary dual network that has the same nodes as the primal network, but where nodes are connected using an optimistic rounding scheme instead. Such a scheme rounds SoC values up and allows to start charging in the last time block before the vehicle arrives at a charging station. It should be noted that not all paths in the dual network correspond to feasible vehicle duties. Any lower bound on the MP's solution value formulated on the dual network corresponds to a lower bound that is valid irrespective of the used discretization. We compute such a lower bound as a separate step from finding feasible solutions.

4.2.4 Obtaining Integral Solutions

We now discuss two heuristics to find integral solutions for the E-VSP. In both cases, we first obtain a solution to the master problem by using column generation.

(Truncated) Price-and-Branch Our first heuristic solves the MP, and then applies a commercial solver to the binary program (1)-(4) using all columns in $\mathcal{P}'$. Such a heuristic has been referred to as a *restricted master heuristic* or *price-and-branch* (Sadykov, Vanderbeck, Pessoa, Tahiri, & Uchoa, 2019). For this heuristic, we can either solve the MP to optimality, or terminate the column generation algorithm before an optimal solution to the MP has been obtained. In the latter case, we prevent the infamous tail-off effect of column generation. We terminate the column generation process if the objective value of the RMP has not improved sufficiently relative to the objective value of a fixed number of iterations earlier. Let the parameters $Z_{\min}$ and I represent the minimum percentage of improvement considered as sufficient and the number of iterations, respectively. Several existing papers use this method to early stop a column generation process (see e.g. Gamache, Soumis, Marquis, & Desrosiers, 1999; Wang, Zhou, & Yue, 2019). After the column generation process is terminated, we solve the binary program as in the price-and-branch heuristic. If the column generation process is terminated early, we refer to this heuristic as *truncated price-and-branch*.

Truncated Column Generation Our second heuristic is comparable to the truncated column generation as proposed by Pepin, Desaulniers, Hertz, and Huisman (2009) to solve the MD-VSP. In this heuristic, a column generation phase and a fixing phase are executed iteratively. Specific path variables in the fractional solution are fixed to one after early termination of the column generation phase. Afterward, the column-generation phase re-starts. This iterative process is repeated until the obtained solution is integral. This is in contrast to the first proposed heuristic that applies a commercial solver after the column generation process to obtain an integer solution. We use the same stopping criterion for each column generation process as for the truncated price-and-branch using the predefined parameters $Z_{\min}$ and I. This early termination

differs from Pepin et al. (2009) since we look at a relative difference, whereas Pepin et al. (2009) use an absolute difference. We use a relative difference to be able to apply a similar method to instances of varying sizes. After termination of each column generation phase, all path variables x_p in the set $\mathcal{P}'$ that have a solution value larger than a predefined threshold θ and that are no initial path are set to one in the RMP if the solution is fractional. If no such paths exist, we fix the path with the maximum value that is not an initial path. Then, we resolve the RMP and start the column generation process again. Each time the column generation process is restarted, at least I iterations should be executed before we early terminate the process. These steps are repeated until an integral solution is obtained.

To reduce the network size during the column generation process, we can extend the truncated column generation heuristic by executing an additional operation each time paths are fixed. This additional step aims to lower the computation time for solving the pricing problem. In this operation, certain nodes and their adjacent arcs are removed from the pricing networks. Firstly, the nodes corresponding to all trips included in the paths that are fixed are removed. Choosing new paths that also contain these trips results in executing trips more than once. It is unlikely that this gives an optimal solution. Hence, it is reasonable to delete these nodes and their adjacent arcs. Secondly, the nodes corresponding to each combination of a charging station and time block that reached the capacity of the charging station due to fixed paths can be deleted. Since the capacity constraints of these charging station and time block combinations are binding, newly added paths that include a charging action for such a combination can never be part of the solution. Lastly, the preprocessing step is repeated. In this step, all nodes that are unreachable from the source or cannot reach the sink are removed. If all nodes as described above and their adjacent arcs are deleted, the network size shrinks once paths are fixed. A potential disadvantage of the node removal is that the quality of the solution might decrease since the search region for new paths shrinks each time paths are fixed due to the network reduction. Note that this only holds true if the optimal solution contains empty trips, which is rare in practice.

5 Results

We now study the performance of our heuristics using a real-world data set. First, we describe the data set and explain how we derive instances of varying size from it. To evaluate the performances of the heuristics, we compare the heuristics by using fixed parameter values on some smaller instances. Thereafter, we tune the parameters of the best-performing heuristic on the smaller instances. Lastly, we solve the larger instances.

The heuristics are implemented in Python 3.7 including the CPLEX solver version 12.10.0. We use a MacBook Pro with a 1.4 GHz Quad-Core Intel Core i5 processor and 8 GB RAM.

5.1 Case

We now describe the data set used for the evaluation of our methodology. The E-VSP requires input on the timetabled trips, the depots, the charging locations, and the bus types. We use data on the timetable of the bus concession of Gooi en Vechtstreek provided by Lynxx to test our proposed heuristics. The data set is used to create several instances with a varying number of trips.

The bus concession of Gooi en Vechtstreek covers the public bus transit in Bussum, Hilversum, Huizen, Naarden, and Weesp (OV in Nederland, 2021). These cities are all located in the eastern part of the province of Noord-Holland. We consider the timetable of December the 3rd, 2019. In total, 816 trips are scheduled that day. The information regarding the trips comes from a General Transit Feed Specification (GTFS) data set. GTFS includes information about the public transit trips and corresponding geographic information.

The trips in this data set can be divided in several clusters. The *city lines* contain only stops that are located in Hilversum. The trips from the city lines are on average shorter than the trips from the *Rnet line*, which contains trips between Amsterdam and Hilversum. Trips corresponding to the *regional lines* depart from various places. The *rush hour lines* and *school lines* contain 27 and 3 trips, respectively. These lines are only used at specific times.

When viewing the number of trips operated simultaneously during the day, two peak mo-

Table 5.1: Bus characteristics of two bus types that are used in the case to provide a bus schedule for the concession of Gooi en Vechtstreek

Characteristics	**Type 1**	**Type 2**
Battery Capacity (kWh)	155	210
Consumption Rate (kWh/km)	1.3	1.4
Idling Consumption Rate (kWh/s)	0.00167	0.00167
Charging Rate (kWh/s)	0.0639	0.0889
Investment Costs (€)	50,000	52,500
Operational Costs (€/km)	1.0	1.05

ments occur. After the start of the service day, we see an increasing number of trips until a peak in the morning around 08:00. Afterward, the number of trips decreases, stabilizes, and increases again after approximately 15:00. The evening peak lasts until approximately 18:30. Thereafter, the number of trips decreases. The maximum number of trips that should be serviced simultaneously is 53. This means that at least 53 buses are required for the bus schedule of the entire concession.

The data set includes two depots, three charging stations, and two bus types. Both depots have a large capacity. Two of the charging stations are located close to the depots, and have a capacity of five and two buses, respectively. The third charging station has a capacity of two buses.

Finally, we consider two different bus types. Information about the bus types regarding their battery capacities, consumption and charging rates, and costs is given in Table 5.1. We translate the battery capacity to a value of 100% for the SoC level. Then, we convert the information that is given in kWh in Table 5.1 to SoC values in proportion to the battery capacity. In this case, we assume both bus types can be located at both depots, can operate all trips and can charge at every charging station.

5.2 Parameter Settings and Instances

We now first describe the parameter settings for our heuristics. Then, we use the data set described in Section 5.1 to create instances with varying numbers of trips.

5.2.1 Parameter Settings

We initially fix the discretization of the possible SoC values and the length of the time blocks. These fixed parameter values are used to compare the heuristics. In Section 5.4, we study the impact of coarser and finer discretizations. The discretization influences the number of nodes and arcs in the pricing networks. We use the same discretization for all networks.

To compare the heuristics, we use 27 possible SoC values, ranging from 22% to 100% in steps of 3%. We start from 22% to incorporate a minimum value for the SoC. Furthermore, we choose 5 minutes as the length of time blocks. We assume that the charging rate is independent of the SoC value at the beginning of the charging action, as it is often assumed in literature (see e.g. Van Kooten Niekerk et al., 2017). Note that this could be easily adjusted because of the way we construct the network.

When creating the network for each instance, we set the maximum deadhead time to 1 hour. The maximum time allowed for idling is set to 8 hours between trips and to 3 hours if a bus goes to or comes from a charging station. We determine the deadhead distance and times using The Open Source Routing Machine (OSRM). In the data set that we consider, all deadhead times are less than 1 hour. Moreover, we use energy costs of €0.1361 per kWh and crew costs of €0.67 per minute, similar to Van Aken and Hiemstra (2020). We use €10 as starting costs of a charging activity. Other cost parameters are given in Table 5.1

5.2.2 Instances

Several instances are used for the computational results. First, we create relatively smaller instances by picking random subsets of the set of all trips. Secondly, we use the clusters as described in Section 5.1 to create different instances. Characteristics of the instances are shown in Table 5.2. Here, Trips is the number of trips in the instance, TH is the length of the time horizon, and Nodes and Arcs represent the total number of nodes and arcs in the four networks, respectively. The time horizon, used for the charging nodes, depends on the length of the service for the considered trips. For each instance, we let the time horizon begin at the start of the

hour of the first trip that departs and end at the end of the hour of the last trip that finishes. Instance A represents 50 trips randomly chosen from all morning trips, containing all trips that finish before noon. Instance B represents 100 trips randomly chosen from all trips. Instances 1 to 5 are combinations of the clusters.

Table 5.2: Characteristics of the instances

Instance	**Trips used**	**Trips (#)**	**TH (h)**	**Nodes (#)**	**Arcs (#)**
A	Random Morning Trips	50	6[1]	13,691[1]	224,873[1]
B	Random Trips	100	20[1]	68,345[1]	1,229,069[1]
1	City line	119	19	70,507	2,084,615
2	Rnet line	185	21	74,671	1,339,960
3	City & Rnet lines	304	21	86,846	3,800,870
4	Rush hour- & School- & Regional Lines	512	21	103,134	9,170,356
5	All bus lines	816	21	125,344	15,589,878

[1] Average outcome of ten instances

In general, the number of nodes and arcs increases if the number of trips increases. However, we see that instance 1 contains more arcs in the final networks than instance 2. This can be explained by the fact that instance 1 includes the shorter City line trips, in contrast to the Rnet line trips that have a longer average duration in instance 2. This results in fewer arcs for instance 2 than for instance 1, since fewer trip pairs are compatible.

5.3 Comparison Heuristics

We now compare the performance of the heuristics we propose in Section 4.2.4. We first state the parameter values we use. Second, we show the computational results of the comparison.

5.3.1 Heuristic Parameters

For the truncated price-and-branch and the truncated column generation heuristics, we stop the column generation process if there is no improvement of at least 0.01% ($Z_{\min} = 0.01$) within 30 iterations ($I = 30$). We use a larger number of iterations than Pepin et al. (2009) to prevent early termination due to degeneracy. Additionally, since we use a relative decrease and the objective value is large at the beginning of the algorithm, a larger value of I prevents the algorithm from terminating too soon. The variables with a value higher than 0.70 are fixed in the truncated

column generation heuristic ($\theta = 0.70$), similar to Pepin et al. (2009). For the (truncated) price-and-branch heuristic, we set a time limit of 1 hour for CPLEX for solving the BP.

5.3.2 Results

In this section, we test the proposed heuristics on the smaller instances A, B, 1, 2, and 3. For instances A and B, the results are averaged over 10 randomly generated instances. The required time and final solution values of the heuristics are presented in Table 5.3. For all heuristics, Time is the total time needed for the heuristic to obtain a final integer solution, including the time needed to solve the BP for the (truncated) price-and-branch heuristic. Sol is the objective value of the final integer solution of the E-VSP, and G is a bound on the optimality gap. The bounds on the optimality gap of all heuristics are computed using a lower bound that is obtained using an optimistic rounding scheme, as explained in Section 4.2.3. The values of these lower bounds and more detailed results can be found in Appendix B.

Table 5.3: A comparison of the required time and final solution of the three proposed heuristics

	Price-and-Branch			**Truncated Price-and-Branch** ($Z_{\min} = 0.01, I = 30$)			**Truncated Column Generation** ($Z_{\min} = 0.01, I = 30, \theta = 0.70$)		
Instance	Time (s)	Sol	G (%)	Time (s)	Sol	G (%)	Time (s)	Sol	G (%)
A	35[1]	744,828[1]	0.869[1]	26[1]	744,896[1]	0.878[1]	37[1]	744,838[1]	0.870[1]
B	1,436[1]	569,503[1]	0.149[1]	451[1]	602,477[1]	6.165[1]	962[1]	569,614[1]	0.167[1]
1	9,504	304,320	19.901	792	304,320	19.901	1,977	254,525	0.282
2	6,163[2]	1,153,237[2]	6.027[2]	4,394[2]	1,319,157[2]	21.281[2]	2,274	1,102,030	1.319
3	22,204[2]	1,549,954[2]	20.278[2]	6,603[2]	1,773,327[2]	37.612[2]	10,134	1,299,192	0.818

[1] Average outcome of ten instances

[2] BP not solved to optimality due to reached time limit

Table 5.3 shows that all three heuristics obtain a high-quality solution for the A-instances. Both the price-and-branch and the truncated column generation heuristic provide a good solution to the B-instances as well. For the larger instances 1, 2, and 3, we see that the truncated column generation outperforms the other two heuristics based on solution quality.

The price-and-branch heuristic requires more than twice as much computation time as the truncated column generation for instances 2 and 3. Note that for instances 2 and 3, the time limit for solving the BP is reached for both the price-and-branch and the truncated price-and-branch heuristic. The price-and-branch heuristic also requires most time for Instance 1 and

the B-instances. Despite its longer computation time, the price-and-branch heuristic provides a poor optimality gap bound of about 20% for instance 1. The truncated price-and-branch heuristic is the fastest for all instances except instance 2. However, the truncated price-and-branch heuristic results in the solution with the highest costs for all instances and has a poor optimality gap bound ($> 19\%$) for the larger instances.

The required time can be partly explained by the number of iterations needed in the column generation process of all heuristics. We present in Table 5.4 the required number of iterations and the average time for solving the pricing problem and RMP per iteration. Here, It is the number of required iterations during the column generation process, PP is the average time of solving the pricing problem per iteration, and RMP is the average time of solving the RMP per iteration.

Table 5.4: The number of iterations and the average time per iteration for solving the pricing problem and the RMP of the three proposed heuristics

	Price-and-Branch			**Truncated Price-and-Branch** ($Z_{\min} = 0.01, I = 30$)			**Truncated Column Generation** ($Z_{\min} = 0.01, I = 30, \theta = 0.70$)		
Instance	It (#)	PP (s)	RMP (s)	It (#)	PP (s)	RMP (s)	It (#)	PP (s)	RMP (s)
A	154[1]	0.22[1]	0.004[1]	114[1]	0.22[1]	0.004[1]	169[1]	0.22[1]	0.004[1]
B	1,076[1]	1.24[1]	0.011[1]	258[1]	1.25[1]	0.010[1]	767[1]	1.26[1]	0.010[1]
1	2,947	2.12	0.016	369	2.13	0.010	936	2.11	0.011
2	1,835	1.37	0.015	580	1.33	0.013	1,673	1.35	0.014
3	4,601	3.97	0.056	754	3.95	0.016	2,657	3.81	0.021

[1] Average outcome of ten instances

As expected, the price-and-branch heuristic requires more iterations than the truncated price-and-branch heuristic due to the early stopping criterion of the truncated price-and-branch heuristic. Similarly, the truncated column generation heuristic requires more iterations than the truncated price-and-branch heuristic, since the truncated column generation heuristic continues the column generation process after the root node has been explored, in contrast to the truncated price-and-branch. We observe that the pricing problem dominates the computation time per iteration, and that the average time to solve one pricing problem is highly similar for the three heuristics. Thus, the computation time is determined mainly by the number of iterations.

Based on the above-described results and the trade-off between computation time and solution quality, we conclude that the truncated column generation heuristic is the best-performing

heuristic. Hence, we continue with the truncated column generation heuristic in the remainder of this section.

5.4 Parameter tuning

In this section, we perform a parameter tuning to study the influence of the chosen parameter values for the truncated column generation heuristic, using the three smallest instances (A, B, and 1). In contrast to the previous section, we now consider a single instance A and B. First, we solve the instances using different values for the parameters of the truncated column generation heuristic to test their influence on the solution quality. Afterward, we solve the instances using various discretizations for the underlying network to test how this impacts the final solution.

5.4.1 Heuristic Parameters

We test how the quality of the solution changes if other parameters are used for the truncated column generation heuristic. We fix the discretization and use possible SoC values in steps of 3% between 22% and 100% and time blocks of 5 minutes, similar to Section 5.3. We test different values for the minimum relative improvement ($Z_{\min}$), the number of iterations (I), and the threshold for fixing variables (θ) used in the truncated column generation heuristic. Note that the minimum allowed value for θ is 0.5. Smaller values could lead to violations of the capacity constraints.

The results for Instance 1 are presented in Table 5.5. Here, Time is the total computation time of the heuristic, It is the number of iterations, and PP and RMP are the average times per iteration to solve the pricing problem and the RMP, respectively. Sol, G, and B are the objective value, a bound on the optimality gap, and the number of buses of the final integer solution, respectively. For instance A and B, we obtained highly similar results for all settings. For this reason, we have reported the results for these instances in the appendix.

Table 5.5 demonstrates that using a value for I of 15 results in a worse solution than using larger values for I (30, 50, or 90). This can partly be explained by the occurrence of degeneracy,

Table 5.5: Results of the truncated column generation heuristic for differing parameters considering the minimum required relative improvement, the number of iterations, and the threshold as used in the fixing step

			Instance 1 (LB = 253, 809.6)						
$Z_{\min}(\%)$	I	θ	Time (s)	It (#)	PP (s)	RMP (s)	Sol	G (%)	B (#)
0.010	30	0.7	1,977	936	2.11	0.011	254,525	0.282	5
0.010	15	0.7	44	21	2.20	0.008	304,320	19.901	6
0.010	50	0.7	3,309	1,514	2.17	0.012	254,327	0.204	5
0.010	90	0.7	4,234	1,942	2.16	0.013	254,250	0.173	5
0.005	30	0.7	3,359	1,554	2.15	0.013	254,287	0.188	5
0.050	30	0.7	1,912	897	2.13	0.011	254,661	0.335	5
0.500	30	0.7	1,854	890	2.08	0.010	304,737	20.065	6
0.010	30	0.5	1,978	936	2.11	0.011	254,525	0.282	5
0.010	30	0.9	1,960	936	2.09	0.011	254,525	0.282	5

which results in the objective value not improving for several iterations. This causes the column generation process to be stopped too early if I is small. In general, increasing I results in a better solution. However, it also increases the number of required iterations and, as a consequence, the required computation time.

Decreasing the value of $Z_{\min}$ to 0.005 while keeping I and θ constant results in slightly better solutions compared to using $Z_{\min} = 0.01$. However, more iterations are executed, which again increases the computation time. Increasing $Z_{\min}$ from 0.01 to 0.05 or 0.5 results in a lower computation time, but also in an increased solution value.

The usage of different values for θ does not always result in a difference in the final solutions. This holds if none of the path variables are above the threshold of 0.7 when the early stopping criterion is met. In this case, the path variable with the maximum value is fixed. This is confirmed by the results for varying values of θ: all values we have tested result in the same solution.

5.4.2 Discretization

As introduced in Section 3, finding a balance between better solutions versus computational tractability plays a crucial role when determining the best discretization (Boland, Hewitt, Marshall, & Savelsbergh, 2019). We now study the impact of the discretization on the performance of the heuristic. In these tests, the heuristic parameters are kept constant with $Z_{\min} = 0.01$,

$I = 30$, and $\theta = 0.70$, similarly as in Section 5.3. Table 5.6 presents the results of the experiments. We test different values for the length of the time blocks (TB) and the step size of the SoC values (Steps). Range represents the interval of the possible SoC values.

Table 5.6: Results of the truncated column generation heuristic for different discretization levels.

			Instance A (LB = 653,319.3)						
TB (min)	Steps (%)	Range	Time (s)	It (#)	PP (s)	RMP (s)	Sol	G (%)	B (#)
5	3	22-100	45	181	0.25	0.004	655,957	0.404	13
2	3	22-100	125	205	0.61	0.006	655,986	0.408	13
10	3	22-100	25	210	0.12	0.003	705,972	8.059	14
30	3	22-100	8	206	0.04	0.002	658,618	0.811	13
1	1	22-100	931	213	4.44	0.012	655,901	0.395	13
5	1	22-100	168	236	0.71	0.004	655,928	0.399	13
5	6	22-100	25	191	0.12	0.004	655,991	0.409	13
5	10	20-100	14	191	0.07	0.003	703,452	7.674	14
5	20	20-100	3	120	0.02	0.003	1,236,037	89.193	24
			Instance B (LB = 559,350.2)						
TB (min)	Steps (%)	Range	Time (s)	It (#)	PP (s)	RMP (s)	Sol	G (%)	B (#)
5	3	22-100	850	700	1.21	0.010	559,863	0.092	11
2	3	22-100	2,301	801	2.87	0.017	560,079	0.130	11
10	3	22-100	470	762	0.61	0.007	560,371	0.183	11
30	3	22-100	165	796	0.20	0.004	564,990	1.008	11
5	1	22-100	3,320	766	4.37	0.011	559,776	0.076	11
5	6	22-100	366	707	0.51	0.009	560,107	0.135	11
5	10	20-100	220	801	0.26	0.009	562,810	0.619	11
5	20	20-100	24	283	0.08	0.008	2,623,753	369.072	51
			Instance 1 (LB = 253,809.6)						
TB (min)	Steps (%)	Range	Time (s)	It (#)	PP (s)	RMP (s)	Sol	G (%)	B (#)
5	3	22-100	1,977	936	2.11	0.011	254,525	0.282	5
2	3	22-100	3,941	1,175	3.32	0.018	254,555	0.294	5
10	3	22-100	45	37	1.25	0.009	304,725	20.060	6
30	3	22-100	616	1,143	0.53	0.007	304,639	20.027	6
5	1	22-100	8,444	1,078	7.85	0.012	254,459	0.256	5
5	6	22-100	34	37	0.93	0.008	304,670	20.039	6
5	10	20-100	18	37	0.50	0.008	304,991	20.165	6
5	20	20-100	76	458	0.16	0.009	2,003,821	689.498	40

First, note that time blocks of 5 minutes and a SoC step size of 3% results in small optimality gaps of 0.40%, 0.09% and 0.28% for instance A, B and 1, respectively. Since these gaps are obtained using a lower bound that does not depend on the used discretization, only relatively small improvements are theoretically possible if one uses a finer discretization. Indeed, we find that choosing a different length of the time blocks than 5 minutes while keeping the possible SoC step size constant at 3%, does rarely result in a better solution. Using time blocks of 2 minutes instead of 5 minutes even worsens the solution quality, despite requiring more iterations and increasing the computation time. This could be explained by the amount of charging

in the considered time block and the conservative rounding. In 5 minutes, the buses charge approximately 12.5%. In 2 minutes, the increase in SoC is roughly 5%. Using a step size of 3%, the amount that is discarded due to rounding between two consecutive charging actions is approximately 0.5% versus 2%, for time blocks of 5 versus 2 minutes, respectively. Hence, using time blocks of 2 minutes might result in a stronger underestimation of the SoC. This demonstrates the interplay between the discretization of time and that of the SoC values. In particular, it is not guaranteed that using a finer time discretization results in a better solution. The usage of a finer time discretization does result in a longer computation time, though.

An increased computation time can also be observed for time blocks of 1 minute and steps of 1% for the possible SoC values for instance A. The cost decrease is negligible, but the computation time is increased by a factor 20. To avoid computation times that are prohibitively long, the experiments with time blocks of 1 minute and steps of 1% for the possible SoC values are not executed for instances B and 1.

Table 5.6 also shows the influence of choosing different step sizes than 3% for the SoC values for a time block length of 5 minutes. For instances A, B, and 1, choosing a step size of 1% results in bus schedules with slightly better solution values, for example due to fewer scheduled charging actions. However, this also approximately triples the average time needed to solve the pricing problem per iteration. This increases the required computation time significantly.

For instances A and B, a slightly worse solution is obtained by using a step size of 6% instead of 3%. Using a step size of 6% roughly halves the time needed to solve the pricing problem per iteration. This causes a significant reduction in the required computation time. We note that a step size of 6% results in the same amount of SoC discarded between two consecutive charging actions as a step size of 3%. Using a step size of 10% causes a solution that requires one more bus for instance A.

For all instances, using a step size of 20% results in a solution that requires significantly more buses. This can be explained by the fact that both buses can increase their SoC by approximately 12.5% during a charging action of 5 minutes. Thus, in this setting, charging actions do actually

not result in an increase in the SoC using this network structure. Hence, choosing a step size for the possible SoC values larger than the amount of SoC increase after charging one time block results in solutions with high costs and requiring more buses.

To conclude, using time blocks of 5 minutes in combination with a step size of 3% gives the best solutions compared to using different lengths of the time blocks. In general, using a step size of 1% results in slightly better solutions than using 3%, but also increases the required computation time. On the other hand, enlarging the step size to 6% shortens the average time to solve the pricing problem per iteration. However, it also can slightly worsen the solution value.

5.5 Larger Instances

In this section, we solve the two larger instances. First, we use instance 4 and the results of Section 5.4 to compare different discretized values for this larger instance using the truncated column generation heuristic. Additionally, we also test the truncated column generation with node removal heuristic. Afterward, we solve the whole concession using the insights obtained for instance 4.

5.5.1 Instance 4

In this section, we solve instance 4 using various techniques. We do not deviate from the used parameter values in Section 5.3 for the truncated column generation heuristic. Hence, we use $Z_{\min} = 0.01$, $I = 30$, and $\theta = 0.70$. Using these parameter settings, we obtained good solutions for instances A, B, 1, 2, and 3 as shown in Section 5.3. All achieved optimality gap bounds are less than 1.5%.

For the discretized values, we solve instance 4 using time blocks of 5 minutes and a step size of 3% between 22% and 100% for the SoC values. Based on the results in Section 5.4.2, the time blocks of 5 minutes give the best solutions. Besides, we prefer using a step size of 3% because of the lower computation times compared to a step size of 1%, despite the slightly worse solutions.

Table 5.7: Results of the truncated column generation (with and without node removal) heuristic for instance 4 using $Z_{\min} = 0.01$, $I = 30$, $\theta = 0.70$, TB = 5 min. and Range = 22-100. The lower bound for this instance equals 1,528,574.

Steps (%)	Node Removal	Time (h)	It (#)	PP (s)	RMP (s)	Sol	G (%)	B (#)
3	No	26.00	9,348	9.87	0.13	1,598,917	4.60	31
6	No	10.66	8,555	4.35	0.12	1,622,100	6.12	31
3	Yes	13.83	8,100	5.96	0.13	1,543,099	0.95	30
6	Yes	7.20	9,472	2.56	0.13	1,573,957	2.97	30

To obtain a solution in less time, we also solve instance 4 using a step size of 6% instead of 3% for the possible SoC values. Doing so leads to smaller networks, and allows the pricing problem to be solved faster. Additionally, we test the node removal heuristic to further reduce the average time needed to solve the pricing problem. This extension removes specific nodes and arcs during the column generation process each time paths are fixed.

The results are presented in Table 5.7. We use the same abbreviations as in Table 5.5. We observe that the truncated column generation heuristic using a step size of 3% requires 26 hours of computation time to solve instance 4. A large part of this computation time is needed to solve the pricing problem, which takes on average 9.87 seconds per iteration. This can be explained by the large number of nodes and arcs in the corresponding network (see Table 5.2).

Using a step size of 6%, the time needed to solve the pricing problem is significantly reduced to 4.35 seconds on average. This is more than two times faster than solving the pricing problem using a step size of 3%. However, we also see an increase in the solution value of €23,183, which is an increase of 1.45%. The total time needed for the column generation process is decreased to less than 11 hours.

Using the truncated column generation with node removal heuristic, we see a further decrease in the average time needed to solve the pricing problem. Now, on average 5.96 and 2.56 seconds are needed to solve the pricing problem per iteration, for steps of 3% and 6%, respectively. Thus, the node removal heuristic significantly reduces the computation time. In particular, using steps of 6%, the total computation time is reduced to 7 hours.

We also observe a decrease in the solution value if nodes are removed from the network. We believe this is due to the heuristic nature of the solution method.

5.5.2 Instance 5: The Entire Concession

Based on the results of Section 5.5.1, we use the truncated column generation with node removal heuristic to solve the entire concession (instance 5) with 816 trips. For the heuristic parameters we use $Z_{\min} = 0.01$, $I = 30$, and $\theta = 0.70$. For the underlying network, we use time blocks with a length of 5 minutes and SoC values between 22% and 100% with a step size of 6%. This results in a network containing in total 63,035 nodes and 7,487,553 arcs. Note that this is a significant reduction compared to the number of nodes and arcs reported in Table 5.2.

Table 5.8: Results of the truncated column generation with node removal heuristic for instance 5 using $Z_{\min} = 0.01$, $I = 30$, and $\theta = 0.70$

TB (min)	Steps (%)	Range	Time (h)	It (#)	PP (s)	RMP (s)	Sol	LB	G (%)	B (#)
5	6	22-100	28.00	20,266	4.30	0.59	2,794,568	2,702,417	3.41	53

We present the results of solving the entire concession in Table 5.8. The truncated column generation with node removal heuristic requires 28 hours to solve instance 5. The optimality gap is below 3.5%. The optimal solution value of the RMP equals 2,755,589. Thus, roughly half of the optimality gap can be explained by the discretization, and the other half by the heuristic to obtain integer solutions. The feasible bus schedule requires 53 buses. This equals the minimum number of buses required based on the number of trips that are serviced simultaneously as explained in Section 5.1. In total, 824 trips are scheduled. This means that 8 trips are driven 'empty'.

Below, we discuss a few characteristics of the bus duties that comprise the final solution. Each bus charges around 2 hours on average during its duty. The average times of deadheading and idling per bus duty are approximately 79 and 166 minutes, respectively. Due to the usage of discretized SoC values in our methodology, the SoC is rounded down in the resulting bus duties before a charging action or before servicing a trip. Per bus duty, the amount of SoC that is discarded as a result adds up to 66 percentage points on average. If we track the SoC of the bus duties without rounding down, we see that the average minimum real SoC value is 35.7%, with an overall minimum of 27.5%. This is higher than the minimum allowed SoC value of 22%. In contrast, if the SoC is rounded down, all bus duties reach the minimum allowed SoC value

of 22% at some point in time. The higher actual minimum SoC values show that the resulting bus duties are feasible, but possibly also suboptimal, since they do not exploit the entire range of allowed SoC values in reality.

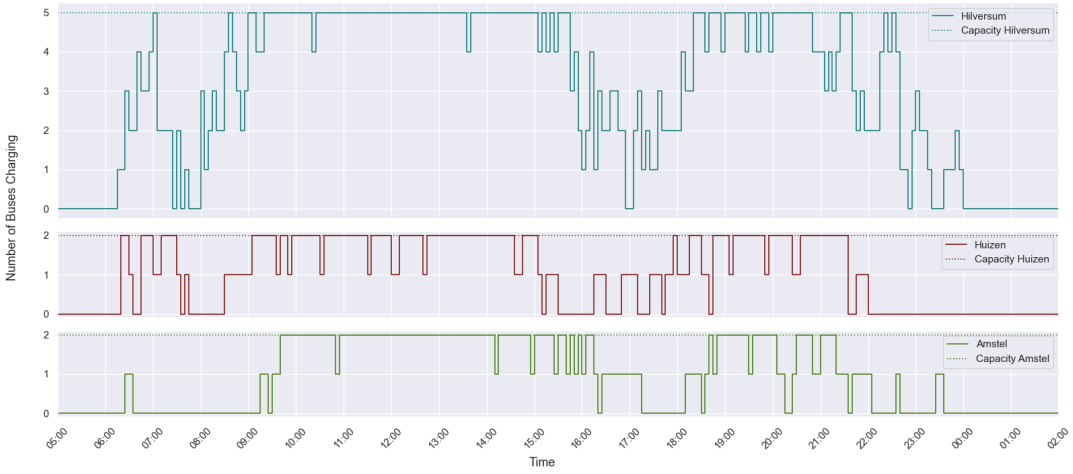

Figure 5.1: The occupation of the charging stations in the resulting bus schedule

An important aspect of a feasible bus schedule is that the capacities of the charging stations are never exceeded. We illustrate the usage of the charging stations during the day in Figure 5.1. Because we incorporated the charging capacities in our methodology, the capacities of the charging stations are never exceeded. This can be seen in Figure 5.1. Additionally, we see that the charging stations are often fully occupied, especially after the morning peak and evening peak hours. In contrast, during these peak moments, the number of buses that are charging is low.

6 Conclusion and Future Research

In this paper, we studied the Electric Vehicle Scheduling Problem considering both the capacity of charging stations and partial charging. To solve the problem, we developed a network structure in which every path represents a feasible duty, and presented two heuristics based on column generation. Valid lower bounds were obtained by solving an auxiliary problem on a slightly altered network.

Computational results based on a bus concession in the Netherlands showed that the truncated column generation outperforms price-and-branch, achieving an optimality gap below 1.5%

on the smaller instances. When applied in combination with the node removal heuristic to speed up the pricing problem, truncated column generation solved the entire concession with 816 trips to an optimality gap of 3.4%.

There are numerous promising directions for future research. It is likely that, in addition to the node removal heuristic, the pricing problem can be sped up further by developing other dedicated acceleration strategies. It would also be interesting to jointly optimize the vehicle schedule with the driver schedule, as the possibility to coordinate charging actions and meal breaks introduces interesting dynamics into the problem. Finally, our method to compute true lower bounds irrespective of the discretization could serve as the starting point for a full-fledged dynamic discretization discovery algorithm, potentially finding better solutions.

Acknowledgements We would like to thank the company Lynxx for facilitating and supervising the project that inspired this paper. Marelot H. de Vos did most of the work on this project while she was at Erasmus School of Economics, Erasmus University Rotterdam.

References

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., & Vance, P. H. (1998). Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, *46*(3), 316–329.

Boland, N., Hewitt, M., Marshall, L., & Savelsbergh, M. (2017). The Continuous Time Service Network Design Problem. *Operations Research*, *65*(5), 1303–1321.

Boland, N., Hewitt, M., Marshall, L., & Savelsbergh, M. (2019). The price of discretizing time: a study in service network design. *EURO Journal on Transportation and Logistics*, *8*(2), 195–216.

Bunte, S., & Kliewer, N. (2009). An overview on vehicle scheduling models. *Public Transport*, *1*(4), 299–317.

Desrosiers, J., & Lübbecke, M. E. (2005). A primer in column generation. In G. Desaulniers,

J. Desrosiers, & M. M. Solomon (Eds.), *Column generation* (pp. 1–32). Boston, MA: Springer US.

Gamache, M., Soumis, F., Marquis, G., & Desrosiers, J. (1999). Column generation approach for large-scale aircrew rostering problems. *Operations Research*, *47*(2), 247–262.

Gintner, V., Kliewer, N., & Suhl, L. (2005). Solving large multiple-depot multiple-vehicle-type bus scheduling problems in practice. *OR Spectrum*, *27*(4), 507–523.

Janovec, M., & Kohári, M. (2019). Exact approach to the electric bus fleet scheduling. *Transportation Research Procedia*, *40*, 1380–1387.

Li, J. Q. (2014). Transit bus scheduling with limited energy. *Transportation Science*, *48*(4), 521–539.

Li, L., Lo, H. K., & Xiao, F. (2019). Mixed bus fleet scheduling under range and refueling constraints. *Transportation Research Part C: Emerging Technologies*, *104*, 443–462.

Li, X., Wang, T., Li, L., Feng, F., Wang, W., & Cheng, C. (2020). Joint optimization of regular charging electric bus transit network schedule and stationary charger deployment considering partial charging policy and time-of-use electricity prices. *Journal of Advanced Transportation*, *2020*, 8863905.

Olsen, N., & Kliewer, N. (2020). Scheduling electric buses in public transport: Modeling of the charging process and analysis of assumptions. *Logistics Research*, *13*(1), 4.

OV in Nederland. (2021). *Concessie Gooi- en Vechtstreek (2011-2021).* Retrieved from `https://wiki.ovinnederland.nl/wiki/Concessie_Gooi-_en_Vechtstreek_(2011-2021)`

Pepin, A. S., Desaulniers, G., Hertz, A., & Huisman, D. (2009). A comparison of five heuristics for the multiple depot vehicle scheduling problem. *Journal of Scheduling*, *12*(1), 17–30.

Perumal, S. S., Lusby, R. M., & Larsen, J. (2022). Electric bus planning & scheduling: A review of related problems and methodologies. *European Journal of Operational Research*, *301*, 395–413.

Posthoorn, C. (2016). *Vehicle Scheduling of Electric City Buses: A Column Generation Approach.* (MSc Thesis, Delft University of Technology). Retrieved from `http://`

repository.tudelft.nl/

Rinaldi, M., Picarelli, E., D'Ariano, A., & Viti, F. (2020). Mixed-fleet single-terminal bus scheduling problem: Modelling, solution scheme and potential applications. *Omega*, *96*, 102070.

Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I., & Uchoa, E. (2019). Primal Heuristics for Branch-and-Price: the assets of diving methods. *Journal on Computing*, *31*(2), 251–267.

Tang, X., Lin, X., & He, F. (2019). Robust scheduling strategies of electric buses under stochastic traffic conditions. *Transportation Research Part C: Emerging Technologies*, *105*, 163–182.

Van Aken, S., & Hiemstra, D. (2020). Strategische keuzes bij zero-emissiebusvervoer met een beslissingsondersteunend algoritme Zero Emission Optimizer. In *Nationaal Verkeerskunde Congres.*

Van Kooten Niekerk, M. E., Van den Akker, J. M., & Hoogeveen, J. A. (2017). Scheduling electric vehicles. *Public Transport*, *9*(1-2), 155–176.

Wang, J., Zhou, L., & Yue, Y. (2019). Column generation accelerated algorithm and optimisation for a high-speed railway train timetabling problem. *Symmetry*, *11*(8), 983.

Wen, M., Linde, E., Ropke, S., Mirchandani, P., & Larsen, A. (2016). An adaptive large neighborhood search heuristic for the Electric Vehicle Scheduling Problem. *Computers and Operations Research*, *76*, 73–83.

Wu, W., Lin, Y., Liu, R., & Jin, W. (2022). The multi-depot electric vehicle scheduling problem with power grid characteristics. *Transportation Research Part B: Methodological*, *155*, 322–347.

Zhang, L., Wang, S., & Qu, X. (2021). Optimal electric bus fleet scheduling considering battery degradation and non-linear charging profile. *Transportation Research Part E: Logistics and Transportation Review*, *154*, 102445.

A Detailed Description of Arcs

In this section, we provide a detailed description of the arcs that are present in the primal and in the dual network.

Consider a given combination $k \in \mathcal{K}$ of a vehicle type and a depot. In the primal network, we apply a conservative rounding scheme. Here, we define the function $F^{\text{soc}}(\cdot)$ that returns the largest SoC value $s \in \mathcal{S}^k$ smaller than or equal to the SoC input. In the dual network, we apply an optimistic rounding scheme. There, the function $F^{\text{soc}}(\cdot)$ is redefined and returns the smallest SoC value $s \in \mathcal{S}^k$ that is larger than or equal to the SoC input.

The compatibility of two nodes $n \in \mathcal{N}^k$ and $v \in \mathcal{N}^k$ depends on several characteristics. Recall that $\tau^{\text{soc}}_{(n,v)}$ denotes the SoC required for deadheading and idling between nodes n and v. Similarly, we define $\phi^{\text{time}}_{(n,v)}$ and $\chi^{\text{time}}_{(n,v)}$ as the required deadheading and idling time between nodes n and v, respectively. Moreover, let $\phi^{\max}_{(n,v)}$ and $\chi^{\max}_{(n,v)}$ denote the maximum allowed deadhead and idle times between nodes n and v. These quantities depend on the combination n and v to be able to impose different restrictions per location. The parameters b_i and e_i denote the begin and end time of trip $i \in \mathcal{T}$.

Below, we explain which arcs are created in network k while discussing the outgoing arcs from the source node, the trip nodes, and the charging nodes separately. Accordingly, we define the sets $A_{d^\sigma_k}$, $\mathcal{A}_{\text{trip}_k}$, and A_{charge_k}. The complete set of arcs of network k is then given by

$$\mathcal{A}^k = A_{d^\sigma_k} \cup \mathcal{A}_{\text{trip}_k} \cup A_{\text{charge}_k}.$$

Source Node First, we consider the source node d^σ_k. By our assumption that vehicles leave the depot fully charged, we do not include arcs that go from the source node to a charging node. When creating outgoing arcs to trip nodes $v \in \mathcal{N}^{\text{trip}}_k$, we consider the required SoC usage for the deadheading from the depot to the start location of the corresponding trip. An arc is created if and only if the maximum deadhead time is not exceeded. Given that the vehicle can depart from the depot such that it arrives precisely in time to operate the trip, idling need not

be considered. We obtain the set

$$A_{d_k^\sigma} = \{(n,v) | n = d_k^\sigma, v = (i,s) \in \mathcal{N}_k^{\text{trip}}, s = F^{\text{soc}}(s^{\text{full}} - \tau_{(n,v)}^{\text{soc}}), \phi_{(n,v)}^{\text{time}} \leq \phi_{(n,v)}^{\max}\}.$$

Trip Node Next, we explain the outgoing arcs from trip nodes $n \in \mathcal{N}_k^{\text{trip}}$ by considering outgoing arcs to the sink node, other trip nodes, and charging nodes separately, defining the sets $\mathcal{A}_{\text{trip}_k}^{\text{sink}}$, $\mathcal{A}_{\text{trip}_k}^{\text{trip}}$, and $\mathcal{A}_{\text{trip}_k}^{\text{charge}}$, respectively.

From trip node $n = (i,s)$, an outgoing arc to the sink node is created if the corresponding SoC value s suffices to return to the depot after performing the trip. Here, we consider the SoC required for the trip, for the deadheading back to the depot, and take into account that the SoC value should always stay above $s_k^{\min}$. This results in the set

$$\mathcal{A}_{\text{trip}_k}^{\text{sink}} = \{(n,v) | n = (i,s) \in \mathcal{N}_k^{\text{trip}}, v = d_k^\tau, s \geq f_i + \tau_{(n,v)}^{\text{soc}} + s_k^{\min}, \phi_{(n,v)}^{\text{time}} \leq \phi_{(n,v)}^{\max}\}.$$

Outgoing arcs from trip node $n = (i,s)$ to other trip nodes $v = (j,s') \in \mathcal{N}_k^{\text{trip}}$ are only possible if the two considered trips are compatible while not exceeding the maximum deadhead and idle times. The idle time is defined as $\chi_{(n,v)}^{\text{time}} = b_j - e_i - \phi_{(n,v)}^{\text{time}}$. We denote $i \to j$ for the requirements $\phi_{(n,v)}^{\text{time}} \leq \phi_{(n,v)}^{\max}$ and $0 \leq \chi_{(n,v)}^{\text{time}} \leq \chi_{(n,v)}^{\max}$. Additionally, from the SoC s of trip node n, the SoC required for the trip corresponding to node n and for the deadheading and idling between the nodes is subtracted to determine the SoC s' of the successor node $v \in \mathcal{N}_k^{\text{trip}}$. This results in the set

$$\mathcal{A}_{\text{trip}_k}^{\text{trip}} = \{(n,v) | n = (i,s) \in \mathcal{N}_k^{\text{trip}}, v = (j,s') \in \mathcal{N}_k^{\text{trip}}, i \to j, s' = F^{\text{soc}}(s - f_i - \tau_{(n,v)}^{\text{soc}})\}.$$

A charging node $v = (r,b,s') \in \mathcal{N}_k^{\text{charge}}$ can directly succeed the trip node n if $\phi_{(n,v)}^{\text{time}} \leq \phi_{(n,v)}^{\max}$ and $0 \leq \chi_{(n,v)}^{\text{time}} \leq \chi_{(n,v)}^{\max}$, where we now compute the idle time as $\chi_{(n,v)}^{\text{time}} = t_b - e_i - \phi_{(n,v)}^{\text{time}}$. Thus, the begin time t_b of time block b should be later than the end time of trip i plus the deadheading time from the end location of trip i to the location of the charging station r. Additionally, the

maximum allowed deadhead and idle time cannot be exceeded. For brevity, we denote these requirements by $i \to (r, b)$. For the SoC value s' of the charging node v, the required SoC for the trip of node n as well as the SoC needed for the deadheading and idling from the end location of the trip to the charging station is subtracted from the SoC value s belonging to node n. This results in the arcs from the set

$$\mathcal{A}^{\text{charge}}_{\text{trip}_k} = \{(n, v) | n = (i, s) \in \mathcal{N}^{\text{trip}}_k, v = (r, b, s') \in \mathcal{N}^{\text{charge}}_k, i \to (r, b), s' = F^{\text{soc}}(s - f_i - \tau^{\text{soc}}_{(n,v)})\}.$$

In the dual network, we want to allow a vehicle to start charging immediately when arriving at the charging station. For that reason, if $0 \leq \chi^{\text{time}}_{(n,v)} < l$, we add the amount of SoC that can be charged during idling. In particular, the last requirement is replaced by $s' = F^{\text{soc}}(s + \hat{s} - f_i - \tau^{\text{soc}}_{(n,v)})$, where $\hat{s}$ is the amount of SoC that could be charged while the vehicle is idling, and where F^{soc} now uses an optimistic rounding scheme. In practice, this corresponds to the situation where the vehicle starts charging during the time block before time block b, immediately after arriving. The vehicle is considered not to occupy the charging station during this time block.

The union of the sets $\mathcal{A}^{\text{sink}}_{\text{trip}_k}$, $\mathcal{A}^{\text{trip}}_{\text{trip}_k}$, and $\mathcal{A}^{\text{charge}}_{\text{trip}_k}$ provides the set $\mathcal{A}_{\text{trip}_k}$ that includes all outgoing arcs from the trip nodes $n \in \mathcal{N}^{\text{trip}}_k$.

Charging Node Lastly, we discuss the outgoing arcs from each charging node $n \in \mathcal{N}^{\text{charge}}_k$ by again considering outgoing arcs to the sink node, trip nodes, and other charging nodes separately. Recall that s^+_n represents the SoC increase during the charging action corresponding to charging node n. The amount of increase depends on the length of the time block of charging node n and the charging rate.

Since each vehicle can charge at its depot, an outgoing arc from charging node n to the sink node is only created if the corresponding SoC value without the charging of node n is too low for deadheading back to the depot, while it is high enough after the charging. Hence, returning to the depot from a charging station is only possible if it is necessary to charge before returning

to the depot. This results in the set

$$\mathcal{A}^{\text{sink}}_{\text{charge}_k} = \{(n,v) | n = (r,t,s) \in \mathcal{N}^{\text{charge}}_k, v = d^{\tau}_k, s < \tau^{\text{soc}}_{(n,v)} + s^{\text{min}}_k \leq s + s^{+}_n, \phi^{\text{time}}_{(n,v)} \leq \phi^{\text{max}}_{(n,v)}\}.$$

A trip node $v = (i, s') \in \mathcal{N}^{\text{trip}}_k$ can directly succeed charging node $n = (r, b, s)$ if $\phi^{\text{time}}_{(n,v)} \leq \phi^{\text{max}}_{(n,v)}$ and $0 \leq \chi^{\text{time}}_{(n,v)} \leq \chi^{\text{max}}_{(n,v)}$, where we compute the idle time as $\chi^{\text{time}}_{(n,v)} = b_i - t_b - l - \phi^{\text{time}}_{(n,v)}$. Thus, the begin time of the trip is later than the end time of the time block corresponding to node n plus the required deadhead time, and the maximum allowed deadhead and idle times are not exceeded. We denote these requirements by $(r, b) \rightarrow i$. The SoC value s' of the trip node v should be equal to the highest possible SoC value smaller than the SoC value s of the charging node n plus the increase due to the charging and minus the required SoC for the deadheading and idling. This results in the set of outgoing arcs

$$\mathcal{A}^{\text{trip}}_{\text{charge}_k} = \{(n,v) | n = (r,b,s) \in \mathcal{N}^{\text{charge}}_k, v = (i,s') \in \mathcal{N}^{\text{trip}}_k, (r,b) \rightarrow i, s' = F^{\text{soc}}(s + s^{+}_n - \tau^{\text{soc}}_{(n,v)})\}.$$

In the dual network, the arc (n, v) is present if the vehicle can execute the trip by departing during time block b, instead of at the end of time block b. In particular, $0 \leq \chi^{\text{time}}_{(n,v)}$ is then replaced by $-l > \chi^{\text{time}}_{(n,v)}$. If $\chi^{\text{time}}_{(n,v)} < 0$, the vehicle charges only during the time it is actually present at the charging station. Thus, s^{+}_n is then replaced by $\hat{s}$, where $\hat{s}$ is the amount of SoC that can be charged during the time $l + \chi^{\text{time}}_{(n,v)}$.

We assume that a charging activity cannot be paused and cannot continue at a different location. Accordingly, an outgoing arc from charging node n to another charging node $v \in \mathcal{N}^{\text{charge}}_k$ is created only if both nodes have the same charging station and the time block of charging node v begins immediately when the time block of charging node n ends. Due to this, a charging activity can take multiple time blocks, but it is not allowed for a vehicle to switch between different charging stations or for the charging activity to contain a break. Additionally, an outgoing

arc to charging node v is created only if the value of the SoC increases. This results in the set

$$\mathcal{A}^{\text{charge}}_{\text{charge}_k} = \{(n,v) | n = (r,b,s) \in \mathcal{N}^{\text{charge}}_k, v = (r',b',s') \in \mathcal{N}^{\text{charge}}_k, n \to v, s' = F^{\text{soc}}(s + s^+_n) > s\},$$

where $n \to v$ denotes the requirements $r = r'$, $t_{b'} = t_b + l$.

The union of the sets $\mathcal{A}^{\text{sink}}_{\text{charge}_k}$, $\mathcal{A}^{\text{trip}}_{\text{charge}_k}$, and $\mathcal{A}^{\text{charge}}_{\text{charge}_k}$ provides the set A_{charge_k} including all outgoing arcs from the charging nodes.

B Detailed Results Comparison Heuristics

Table B.1 presents the detailed results of the runs that are discussed in Section 5.3.2. We first report the computation times. For the (truncated) price-and-branch heuristics, Time CG is the time needed for the column generation process and Time BP is the time needed to solve the binary program. For the truncated column generation, Time is the total time needed to solve the instances. Moreover, It is the number of iterations of the column generation process, PP is the average time for solving the pricing problem per iteration, and RMP is the average time for solving the RMP per iteration. LB is a lower bound on the optimal solution value. Sol is the objective value of the E-VSP corresponding to the final integer solution of the heuristic, G is the optimality gap bound, and B is the number of buses used in the final bus schedule.

Table B.2 presents the results that are obtained for various heuristic parameter settings on Instance A and Instance B, as discussed in Section 5.4.1 Here, Time is the total computation time of the heuristic, It is the number of iterations, and PP and RMP are the average times per iteration to solve the pricing problem and the RMP, respectively. Sol, G, and B are the objective value, a bound on the optimality gap, and the number of buses of the final integer solution, respectively.

Table B.1: Detailed results of all three proposed heuristics

	Price-and-Branch								
Instance	Time CG (s)	Time BP(s)	It (#)	PP (s)	RMP (s)	LB	Sol	G (%)	B (#)
A	35.20[1]	0.02[1]	154.40[1]	0.22[1]	0.004[1]	738,971.06[1]	744,828.45[1]	0.869[1]	14.80[1]
B	1,356.78[1]	78.79[1]	1,076.10[1]	1.24[1]	0.011[1]	568,726.21[1]	569,502.93[1]	0.149[1]	11.20[1]
1	6,346.01	3,158.03	2,947.00	2.12	0.016	253,809.57	304,320.39	19.901	6.00
2	2,555.74	3,607.29[2]	1,835.00	1.37	0.015	1,087,685.65	1,153,236.86	6.027	22.00
3	18,604.00	3,600.10[2]	4,601.00	3.97	0.056	1,288,645.26	1,549,954.37	20.278	30.00
	Truncated Price-and-Branch ($Z_{\min} = 0.01, I = 30$)								
Instance	Time CG (s)	Time BP(s)	It (#)	PP (s)	RMP (s)	LB	Sol	G (%)	B (#)
A	25.72[1]	0.04[1]	113.50[1]	0.22[1]	0.004[1]	738,971.06[1]	744,895.65[1]	0.878[1]	14.80[1]
B	326.29[1]	125.19[1]	258.40[1]	1.25[1]	0.010[1]	568,726.21[1]	602,477.38[1]	6.165[1]	11.70[1]
1	791.20	0.98	369.00	2.13	0.010	253,809.57	304,320.39	19.901	6.00
2	782.47	3,611.40[2]	580.00	1.33	0.013	1,087,685.65	1,319,156.97	21.281	25.00
3	2,998.59	3,604.73[2]	754.00	3.95	0.016	1,288,645.26	1,773,326.53	37.612	34.00
	Truncated Column Generation ($Z_{\min} = 0.01, I = 30, \theta = 0.70$)								
Instance	Time (s)		It (#)	PP (s)	RMP (s)	LB	Sol	G (%)	B (#)
A	37.33[1]		168.90[1]	0.22[1]	0.004[1]	738,971.06[1]	744,837.94[1]	0.870[1]	14.80[1]
B	962.00[1]		766.90[1]	1.26[1]	0.010[1]	568,726.21[1]	569,613.50[1]	0.167[1]	11.20[1]
1	1,976.95		936.00	2.11	0.011	253,809.57	254,524.52	0.282	5.00
2	2,273.95		1,673.00	1.35	0.014	1,087,685.65	1,102,030.14	1.319	21.00
3	10,134.22		2,657.00	3.81	0.021	1,288,645.26	1,299,191.73	0.818	25.00

[1] Average outcome of 10 instances

[2] BP not solved to optimality due to reached time limit

Table B.2: Results of the truncated column generation heuristic for differing parameters considering the minimum required relative improvement, the number of iterations, and the threshold as used in the fixing step

			Instance A (LB = 653,319.3)						
$Z_{\min}$(%)	I	θ	Time (s)	It (#)	PP (s)	RMP (s)	Sol	G (%)	B (#)
0.010	30	0.7	46	181	0.25	0.004	655,957	0.404	13
0.010	15	0.7	53	215	0.25	0.004	656,012	0.412	13
0.010	50	0.7	44	171	0.25	0.004	655,941	0.401	13
0.010	90	0.7	47	186	0.25	0.004	655,946	0.402	13
0.005	30	0.7	48	191	0.25	0.004	655,947	0.402	13
0.050	30	0.7	55	221	0.25	0.004	655,950	0.402	13
0.500	30	0.7	48	189	0.26	0.004	655,961	0.404	13
0.010	30	0.5	42	162	0.25	0.004	655,965	0.405	13
0.010	30	0.9	43	172	0.25	0.004	655,941	0.401	13
			Instance B (LB = 559,350.2)						
$Z_{\min}$(%)	I	θ	Time (s)	It (#)	PP (s)	RMP (s)	Sol	G (%)	B (#)
0.010	30	0.7	850	700	1.21	0.010	559,863	0.091	11
0.010	15	0.7	612	513	1.20	0.009	560,638	0.230	11
0.010	50	0.7	1,199	975	1.22	0.010	559,819	0.083	11
0.010	90	0.7	1,570	1,277	1.22	0.011	559,635	0.051	11
0.005	30	0.7	1,108	884	1.25	0.010	559,775	0.076	11
0.050	30	0.7	495	416	1.20	0.009	560,379	0.184	11
0.500	30	0.7	424	361	1.19	0.009	560,350	0.178	11
0.010	30	0.5	1,006	807	1.24	0.010	559,852	0.089	11
0.010	30	0.9	893	700	1.38	0.010	559,863	0.091	11