Electric Vehicle Scheduling in Public Transit with Capacitated Charging Stations

Marelot H. de Vos, Rolf N. van Lieshout, Twan Dollevoet

https://pubsonline.informs.org/journal/trsc

TRANSPORTATION SCIENCE
Vol. 58, No. 2, March–April 2024, pp. 279–294
ISSN 0041-1655 (print), ISSN 1526-5447 (online)

Electric Vehicle Scheduling in Public Transit with Capacitated Charging Stations

Marelot H. de Vos,[a] Rolf N. van Lieshout,[b,*] Twan Dollevoet[c]

[a] ORTEC, Data Science & Consulting Department, 2719 EA Zoetermeer, Netherlands; [b] Department of Operations, Planning, Accounting, and Control, School of Industrial Engineering, Eindhoven University of Technology, 5600 MB Eindhoven, Netherlands; [c] Econometric Institute and Erasmus Center for Optimization in Public Transport, Erasmus University Rotterdam, 3062 PA Rotterdam, Netherlands

*Corresponding author

Contact: marelotdevos@hotmail.com (MHdV); r.n.v.lieshout@tue.nl, https://orcid.org/0000-0001-9918-5962 (RNvL); dollevoet@ese.eur.nl, https://orcid.org/0000-0003-3193-823X (TD)

Received: July 27, 2022
Revised: April 28, 2023; October 27, 2023
Accepted: October 29, 2023
Published Online in Articles in Advance: December 12, 2023

https://doi.org/10.1287/trsc.2022.0253

Copyright: © 2023 INFORMS

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 a diving heuristic, including acceleration strategies. We test the approach using data from the concession Gooi en Vechtstreek in the Netherlands, containing up to 816 trips. The diving heuristic outperforms the other heuristic and solves the entire concession within seven hours of computation time with an optimality gap of less than 3%.

Supplemental Material: The online appendix is available at https://doi.org/10.1287/trsc.2022.0253.

Keywords: electric vehicles • bus scheduling • partial charging • column generation • discretization

1. Introduction

The benefits of electric buses are undisputed: replacing conventional combustion engine buses with 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 because 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 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 the E-VSP, primarily from the last three years (Perumal, Lusby, and 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 (nonlinear) charging, bridging the gap between theory and practice.

A number of other papers on the E-VSP consider charging station capacity. However, these papers either consider battery swapping or another form of constant-time charging. Another stream of literature considers partial charging but ignores charging station capacity. Literature on the E-VSP with both partial charging and capacitated charging stations is scarce and considers only small instances, or it comes with other limitations. 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 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 nonlinear charging functions and the impact of depth-of-discharge on battery lifetime, and it is therefore widely applicable.

Using the developed network structure, we formulate the problem as a path-based binary program (BP) with side constraints, whose linear programming (LP) relaxation can be solved with column generation. Because of 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 a diving heuristic. In price-and-branch, only the linear relaxation is solved using column generation, after which all generated paths are fed to a commercial mixed integer programming (MIP) solver that optimizes over this given subset of all paths. The diving heuristic performs a depth-first exploration of the branch-and-bound tree, where all nodes are solved using column generation. 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 it is therefore not a true bound of the underlying problem. Inspired by Boland et al. (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 in which 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 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 southeast of Amsterdam. We also generate smaller instances by taking random subsets of all trips. On the smaller instances, the diving heuristic 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 the diving heuristic up to an optimality gap of 2.7%. 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 the numerical results, including sensitivity analyses. Last, we conclude in Section 6.

2. Problem Description

The problem that we consider in this paper is a multidepot 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 (multidepot) vehicle scheduling problem (MD-)VSP (Bunte and Kliewer 2009). The input is a set of timetabled trips, where every trip has a fixed starting and ending location, as well as a fixed starting and ending time. 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. The schedule of a single vehicle is referred to as a *duty*. A duty is feasible if it starts and ends at the same depot and consists of a sequence of compatible trips. Each vehicle should be assigned to a feasible duty, such that all trips are performed.

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 locations and capacities. Charging stations are not required to be located at the starting or ending locations of trips, in which case vehicles need to deadhead to and from charging stations. The capacities of charging stations imply that only a limited number of charging actions can be scheduled simultaneously at each charging station. We do not require that every charging action fully recharges a vehicle's battery; that is, we allow for partial charging. The duration of a charging action correlates positively with the increase in SoC according to a known *charging function*. In addition, we consider a heterogeneous fleet, where the battery limit, charging function, and energy consumption rate are allowed to differ per type of vehicle. Finally, we assume that the objective is to minimize the total cost of ownership, which includes investment costs for the vehicles, variable costs per kilometer and per minute, and costs for consumed energy. Furthermore, the objective also includes a fixed penalty per charging action to avoid unnecessary charging.

3. Literature Review

For a general review on electric vehicle scheduling and related problems, we refer to Perumal, Lusby, and Larsen (2022). Here, we limit ourselves to discussing research on electric vehicle scheduling that considers the capacity of charging stations and/or partial charging.

3.1. Capacitated Charging Stations

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. Li, Lo, and Xiao (2019) deal 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; that is, partial charging is not considered. The authors formulate the problem as a MIP model and solve instances with 288 trips and two depots using a commercial solver. Tang, Lin, and He (2019) investigate the scheduling of electric buses in a stochastic setting, where the aim is to find schedules that are 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, in both static and 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 1,008 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 can solve instances with up to 500 trips. Olsen and Kliewer (2020) extend this heuristic to allow for nonlinear 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 Li et al. (2020), who develop an adaptive genetic algorithm that is used to solve instances with up to 867 trips. Van Kooten Niekerk, Van den Akker, and Hoogeveen (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 solvable only 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 they solve instances containing up to 1,200 trips. However, the authors use a heuristic pricing algorithm and therefore are unable to present optimality gaps. Parmentier, Martinelli, and Vidal (2023) develop a scalable column generation approach with an exact pricing algorithm based on bidirectional labeling and leverage a diving heuristic to find near-optimal solutions for instances with up to 500 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, Wang, and Qu (2021) study electric vehicle scheduling with partial (nonlinear) 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.

3.4. Our Contribution

An overview of the discussed literature is presented in Table 1. As can be seen, in terms of the problem scope, our contribution is that we consider a rich E-VSP with partial charging, multiple depots and vehicle types, and capacitated charging stations. Our methodology is an extension of the existing column generation approaches used by Van Aken and Hiemstra (2020) and Posthoorn (2016). However, in contrast to Van Aken and Hiemstra (2020), we use an exact pricing algorithm and have additional constraints in the master problem to model capacities. In contrast to Posthoorn (2016), we use an additional heuristic to find integer solutions. In addition, although many of the discussed papers use a solution approach based on a discretization of the SoC, none of these papers

Table 1. Overview of Included Aspects in Papers on the E-VSP

	Partial charging	Multiple depots	Multiple vehicle types	Charging station capacity	Solution method	Number of trips
Li (2014)				✓	CG	947
Posthoorn (2016)	✓			✓	CG	709
Wen et al. (2016)	✓	✓			MH	500
Van Kooten Niekerk, Van den Akker, and Hoogeveen (2017)	✓				CG	543
Janovec and Kohání (2019)	✓			✓	MIP	160
Li, Lo, and Xiao (2019)		✓	✓	✓	MIP	288
Tang, Lin, and He (2019)				✓	CG	96
Li et al. (2020)	✓	✓			MH	867
Olsen and Kliewer (2020)	✓	✓			MH	10,710
Rinaldi et al. (2020)			✓	✓	MIP	1,008
Van Aken and Hiemstra (2020)	✓	✓	✓		CG	1,200
Zhang, Wang, and Qu (2021)	✓			✓	CG	160
Wu et al. (2022)		✓		✓	CG	400
Parmentier, Martinelli, and Vidal (2023)	✓	✓			CG	500
This paper	✓	✓	✓	✓	CG	816

Note. CG, Column generation (including branch-and-price and CG-based heuristics); MH, metaheuristic.

quantifies the "cost" of this discretization. In this paper, we are able to do so by developing a method for computing a lower bound that is valid irrespective of the discretization.

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 because the set covering formulation is claimed to be numerically more stable (Barnhart et al. 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, because we solve the E-VSP, charging activities should be taken into account. Similar to Li (2014), we discretize the time horizon into time blocks (TBs) $\mathcal{B}$ 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, whereas 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 $\mathcal{P}$ as 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 one if trip $i \in \mathcal{T}$ is included in path $p \in \mathcal{P}$, and zero otherwise. Similarly, the coefficient $u_{r,b,p}$ is one 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:

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

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

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

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

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 $b \in \mathcal{B}$. Last, Constraint (4) provides the range of the decision variables.

4.2. Column Generation Algorithm

Because 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 cost because these can improve the objective value. The path with the most negative reduced cost is added to the set $\mathcal{P}'$ used in the RMP. If the pricing problem cannot provide a path with negative reduced cost, 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 cost 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 cost 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 cost. 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.

4.2.1.1. Network Structure.

We construct a connection-based network that allows us to create feasible vehicle duties. In particular, we create a separate network for each combination of 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 the lowest reduced cost in each network $k \in \mathcal{K}$ separately. If we find a duty with negative reduced cost in multiple networks, we add all of them to the RMP.

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 that 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.

4.2.1.2. Nodes.

In each network G^k, we include a source node and a sink node, denoted by 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, Van den Akker, and Hoogeveen (2017), to keep track of the SoC of vehicles along their paths. Mathematically, let $\mathcal{T}^k$ and $\mathcal{S}^k$ be the sets of trips and all possible SoC values for network k, respectively. Because 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

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

For each node, the value s represents the SoC of the vehicle at the moment it departs from the starting location of trip i, 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. Because this SoC value represents the SoC before the charging, we disregard the nodes for a fully charged SoC, represented by s^{full}, because 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 $k \in \mathcal{K}$.

4.2.1.3. Arcs. The set of arcs $\mathcal{A}^k$ represents 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 a 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 Online Appendix A.

Given that vehicles leave the depot fully charged, we include only 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 toward a trip or charging node v, the SoC value s' of v must satisfy

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

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

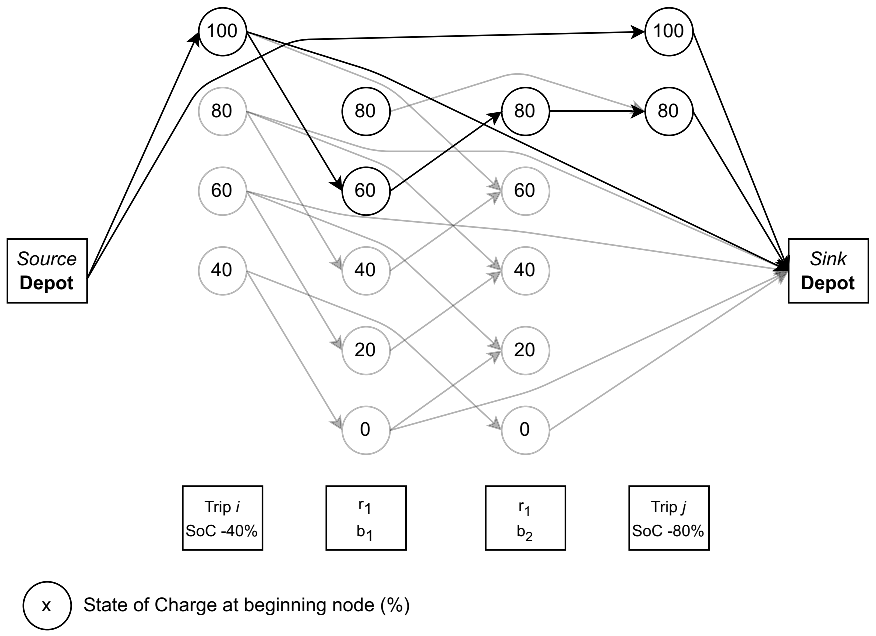
For charging nodes $n = (r,b,s)$, we include arcs to the sink node, to trip nodes, and to other charging nodes. We denote the amount of SoC recharged in node n by s_n^+. An outgoing arc to the sink node is included if the node's SoC value after recharging is sufficient to return to the depot, whereas this is not the case without recharging s_n^+. If it would be possible to return to the depot without recharging at 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 $s' = F^{\text{soc}}(s + s_n^+ - \tau_{(n,v)}^{\text{soc}})$. 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 $s' = F^{\text{soc}}(s + s_n^+) > s$.

4.2.1.4. 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 $c_{(n,v)}^{\text{arc}}$. Arcs coming from the source node $(n = d_k^\sigma)$ additionally include the investment costs corresponding to the vehicle of network k, represented by c_k^{invest}. 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.

The reduced cost 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 $v = (i,s) \in \mathcal{N}_k^{\text{trip}}$ from the arc costs of all arcs (n, v) toward 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).

4.2.1.5. Example. An example of a resulting network is given in Figure 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. Trips i and j reduce the SoC by 40% and 80%, respectively. Trip i ends at a location from which the SoC is reduced by

Figure 1. Network Including All Nodes and Arcs Before the Preprocessing



Note. 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.

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% and 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 shade. In a preprocessing step, we remove these node and arcs from the network.

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 cost 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 cost 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 in which nodes are connected using an optimistic rounding scheme instead. Such a scheme rounds SoC values up and allows charging to start 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.

4.2.4.1. (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 et al. 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 stop a column generation process early (see, e.g., Gamache et al. 1999; Wang, Zhou, and 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*.

4.2.4.2. Diving Heuristic. Our second heuristic is a depth-first exploration of the branch-and-bound tree, known as a diving heuristic in the literature. It is also comparable to the truncated column generation approach as proposed by Pepin et al. (2009) to solve the MD-VSP. In this heuristic, a column generation phase and a fixing phase are executed iteratively until we obtain an integral solution. After the termination of a column generation phase, all fractional path variables x_p that have a value larger than a predefined threshold θ and that do not correspond to one of the initial paths are fixed to one in the RMP. If no such paths exist, we fix the path with the maximum fractional value. Then, we resolve the RMP and start the column generation process again. We repeat these two steps until the solution contains no more fractional variables. We use the same stopping criterion for each column generation phase as for the truncated price-and-branch: each phase should run a minimum number of iterations I and terminate as soon as the relative improvement over the last I iterations drops below $Z_{\min}$. This early termination criterion differs from that of Pepin et al. (2009) because we look at a relative improvement, whereas Pepin et al. (2009) use an absolute improvement. We use a relative difference to be able to apply a similar method to instances of varying sizes.

We can reduce the network size during the execution of this heuristic by exploiting the fact that once a part of the solution is fixed, certain nodes and their adjacent arcs are unlikely to appear in the final solution. By removing these parts of the network, the computation time for solving the pricing problem decreases. First, we remove nodes corresponding to all trips included in the fixed paths. Selecting additional paths that also contain these trips results in executing trips more than once, which is unlikely to give an optimal solution. Hence, it is reasonable to delete these nodes and their adjacent arcs. Second, we delete nodes corresponding to each combination of a charging station and time block that reached the capacity of the charging station due to fixed paths. Because 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. Third, the preprocessing step is repeated. In this step, all nodes that are unreachable from the source or cannot reach the sink are removed. A potential disadvantage of the node removal is that the quality of the solution might decrease. However, this occurs only 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 first determine the best parameters for each heuristic and then compare them using their best parameter values on some smaller instances. Thereafter, we study the impact of the discretization. Last, we solve the larger instances and examine the impact of the battery capacity and of the charging station capacity on the costs of the solutions.

The heuristics are implemented in Python 3.9.13 and use CPLEX solver version 22.1.1 to solve the linear and integer programs. We use a laptop with a 12th generation Intel i7 processor at 2.3 GHz and with 32 GB of random access memory and running Windows for the experiments.

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 3, 2019. In total, 816 trips are scheduled that day. The information regarding the trips comes from a general transit feed specification data set. This includes information about the public transit trips and corresponding geographic information.

The trips in this data set can be divided into 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 *R-net line*, which contains trips between Amsterdam and Hilversum. Trips corresponding to the *regional lines* depart from various

Table 2. 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

places. The *rush hour lines* and *school lines* contain 27 and 3 trips, respectively. These lines are used only at specific times.

When viewing the number of trips operated simultaneously during the day, two peak moments occur. After the start of the service day, we see an increasing number of trips until a peak in the morning around 8:00 a.m. Afterward, the number of trips decreases, stabilizes, and increases again after approximately 3:00 p.m. The evening peak lasts until approximately 6:30 p.m. 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 capacities 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 2. We translate the battery capacity to a value of 100% for the SoC level. Then, we convert the information that is given in kilowatt hours in Table 2 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 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 five minutes as the length of the 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 the literature (see, e.g., Van Kooten Niekerk, Van den Akker, and Hoogeveen 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 one hour. The maximum time allowed for idling is set to eight hours between trips and to three 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. In the data set that we consider, all deadhead times are less than one 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 the starting cost of a charging activity. Other cost parameters are given in Table 2.

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. Second, we use the clusters as described in Section 5.1 to create different instances. Characteristics of the instances are shown in Table 3. 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 numbers of nodes and arcs in the four networks. 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.

In general, the numbers of nodes and arcs increase 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 R-net line trips that have a longer average duration in instance 2. This results in fewer arcs for instance 2 than for instance 1, because fewer trip pairs are compatible.

5.3. Comparison of the Heuristics

We now compare the performance of the heuristics we proposed in Section 4.2.4. We first determine the best parameters for each heuristic. Then, we compare the heuristics using the best values for their parameters.

Table 3. Characteristics of the Instances

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

[a]Average outcome of 10 instances.

5.3.1. Tuning the Parameters of the Heuristics. We first determine the best parameters for each of the heuristics. We do so by running the heuristics for various values of their parameters on instances A, B, 1, 2, and 3. For instances A and B, the results are averaged over 10 randomly generated instances.

For the (truncated) price-and-branch heuristic, we set a time limit of one hour for CPLEX for solving the binary program. We vary the minimum improvement and the number of iterations without sufficient improvement before the column generation process is truncated. We report detailed results for all instances in Table B.1 in Online Appendix B and summarize those for instance 3 in Table 4. In these and all subsequent tables, the column headings mean the following: "Time" is the total computation time to obtain the solution. For the (truncated) price-and-branch heuristic, "Time CG" and "Time BP" are the time devoted to the column generation process and to solving the binary program, respectively. "It" denotes the number of iterations. "Sol" indicates the solution value, whereas "G" denotes the gap to a lower bound that is obtained using the dual network. Finally, "B" is the number of buses in the solution.

We observe that the number of iterations, and thus the computation time, increases when I increases or when $Z_{\min}$ decreases. Generally, we also obtain better solutions for higher values of I or lower values of $Z_{\min}$. The best results are obtained if we do not truncate the generation of columns prematurely, but this also leads to the longest computation times. For the larger instances, we observe that the resulting binary programs are too large to be solved to optimality. We conclude that the best results are obtained with $Z_{\min} = 0.005$ and $I = 30$. The gaps for this setting are very similar to those obtained with $Z_{\min} = 0.010$ and $I = 90$, but the latter setting leads to higher running times. Finally, we note that truncating the column generation process indeed reduces the computation time, but it leads to worse solutions, especially for the larger instances.

For the diving heuristic, we report detailed results for all instances in Tables B.2 and B.3 in Online Appendix B. In the former table, we do not apply the node removal heuristic, whereas in the latter we do. The results for instance 3, obtained with the node removal heuristic, are reported in Table 5 as well. Both with and without the node removal heuristic, we observe that the computation time increases for higher values of I or lower values of $Z_{\min}$. Comparing the results in Tables B.2 and B.3, we see that the node removal heuristic leads to better solutions as well as lower computation times. The best trade-off between computation time and solution quality is obtained with the node removal heuristic and by using $I = 30$, $Z_{\min} = 0.01$, and $\theta = 0.7$.

We use a larger number of iterations without sufficient improvement than Pepin et al. (2009) to prevent early termination due to degeneracy. Additionally, because 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 0.7 are fixed in the diving

Table 4. Results of the (Truncated) Price-and-Branch Heuristic for Varying Parameters Considering the Minimum Required Relative Improvement and the Number of Iterations Without Sufficient Improvement

		Instance 3 (Lower bound = 1,288,645.26)						
$Z_{\min}$(%)	I	Time (s)	Time CG (s)	Time BP (s)	It (#)	Sol	G (%)	B (#)
0.010	15	4,105.32	498.17	3,607.12	397	1,776,757.37	37.9	34
0.010	30	4,208.09	600.78	3,607.29	480	1,775,940.42	37.8	34
0.010	50	4,387.87	784.74	3,603.13	630	1,763,050.65	36.8	34
0.010	90	4,609.04	1,004.54	3,604.49	779	1,667,645.22	29.4	32
0.500	30	3,877.10	263.13	3,613.94	207	1,844,736.90	43.2	35
0.050	30	4,034.38	429.17	3,605.10	341	1,742,360.63	35.2	33
0.005	30	4,413.03	806.29	3,606.74	634	1,666,118.85	29.3	32
0	—	5,681.50	2,078.06	3,603.44	1,588	1,500,923.52	16.5	29

Table 5. Results of the Diving Heuristic for Varying Parameters Considering the Minimum Required Relative Improvement, the Number of Iterations Without Sufficient Improvement, and the Threshold as Used in the Fixing Step

			Instance 3 (Lower bound = 1,288,645.26)						
$Z_{\min}$(%)	I	θ	Time (s)	It (#)	PP (s)	RMP (s)	Sol	G (%)	B (#)
0.010	15	0.7	1,162.99	1,265	0.840	0.039	1,296,170.60	0.6	25
0.010	30	0.7	1,523.37	1,545	0.888	0.049	1,295,483.00	0.5	25
0.010	50	0.7	2,002.45	2,084	0.834	0.058	1,297,757.53	0.7	25
0.010	90	0.7	3,216.37	3,004	0.900	0.073	1,299,968.82	0.9	25
0.500	30	0.7	859.80	1,008	0.793	0.027	1,297,252.90	0.7	25
0.050	30	0.7	1,096.33	1,225	0.818	0.038	1,295,897.80	0.6	25
0.010	30	0.7	1,523.37	1,545	0.888	0.049	1,295,483.00	0.5	25
0.005	30	0.7	1,857.99	1,883	0.872	0.054	1,295,369.28	0.5	25
0.010	30	0.5	1,404.01	1,541	0.816	0.044	1,295,545.94	0.5	25
0.010	30	0.7	1,523.37	1,545	0.888	0.049	1,295,483.00	0.5	25
0.010	30	0.9	1,544.48	1,630	0.847	0.048	1,343,659.68	4.3	26

Note. The node removal heuristic is applied.

heuristic ($\theta = 0.7$), similar to Pepin et al. (2009). We note that using $\theta = 0.9$ provides very similar results. Using $\theta = 0.5$ leads to solutions of worse quality and to the inability to find a feasible solution for some smaller instances.

5.3.2. Comparison of the Heuristics. We now present the results of the heuristics obtained using their best-performing parameters. The required time and final solution values obtained with each heuristic are presented in Table 6. Recall that 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 Online Appendix B.

Table 6 shows that all three heuristics obtain a high-quality solution for the A instances and the B instances. For the larger instances 1, 2, and 3, we see that the diving heuristic outperforms the other heuristics based on both solution quality and computation time. The price-and-branch heuristic requires more than three times as much computation time as the diving heuristic for these instances. Note that the time limit for solving the BP is reached for both the price-and-branch and truncated price-and-branch heuristics for these instances. Despite its longer computation time, the price-and-branch heuristic provides a poor optimality gap bound of over 15% for instances 1 and 3. The truncated price-and-branch heuristic is faster, but it also results in the solutions with the highest costs and a poor optimality gap bound.

The required time can be partly explained by the number of iterations needed in the column generation process of all the heuristics. We present in Table 7 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 to solve the pricing problem per iteration, and “RMP” is the average time to solve the RMP per iteration.

As expected, the price-and-branch heuristic requires more iterations than the truncated price-and-branch heuristic because of the early stopping criterion of the truncated price-and-branch heuristic. Similarly, the diving heuristic requires more iterations than the truncated price-and-branch heuristic because the diving heuristic continues the column generation process after the root node has been explored, in contrast to the truncated price-and-branch heuristic. 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,

Table 6. A Comparison of the Required Time and Final Solution of the Three Proposed Heuristics

	Price-and-branch			Truncated price-and-branch ($Z_{\min} = 0.005, I = 30$)			Diving heuristic ($Z_{\min} = 0.01, I = 30, \theta = 0.70$)		
Instance	Time (s)	Sol	G (%)	Time (s)	Sol	G (%)	Time (s)	Sol	G (%)
A[a]	7.72	744,833.73	0.9	7.56	744,834.58	0.9	7.46	749,826.43	1.5
B[a]	282.68	569,447.48	0.1	322.07	571,678.03	0.6	204.07	569,405.92	0.1
1	5,469.47[b]	304,785.38[b]	20.1[b]	3,992.41[b]	355,723.58[b]	40.2[b]	539.11	254,407.95	0.2
2	3,881.14[b]	1,151,663.22[b]	5.9[b]	3,774.77[b]	1,207,394.26[b]	11.0[b]	336.35	1,103,707.65	1.5
3	5,681.50[b]	1,500,923.52[b]	16.5[b]	4,413.03[b]	1,666,118.85[b]	29.3[b]	1,523.37	1,295,483.00	0.5

[a]Average outcome of 10 instances.
[b]The BP is not solved to optimality because of reaching the time limit.

Table 7. 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 ($Z_{\min} = 0.005, I = 30$)			Diving heuristic ($Z_{\min} = 0.01, I = 30, \theta = 0.70$)		
Instance	It (#)	PP (s)	RMP (s)	It (#)	PP (s)	RMP (s)	It (#)	PP (s)	RMP (s)
A[a]	97.2	0.074	0.002	94.5	0.075	0.002	96.5	0.073	0.002
B[a]	625.6	0.400	0.005	280.0	0.410	0.004	618.9	0.315	0.005
1	2,605	0.676	0.026	565	0.666	0.006	920	0.558	0.009
2	580	0.424	0.008	360	0.428	0.008	874	0.342	0.013
3	1,588	1.207	0.056	634	1.232	0.015	1,545	0.888	0.049

[a]Average outcome of 10 instances.

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 diving heuristic is the best-performing heuristic. Hence, we continue with this algorithm in the remainder of this section.

5.4. Impact of Discretization

As introduced in Section 3, finding a balance between better solutions and computational tractability plays a crucial role when determining the best discretization (Boland et al. 2019). We now study the impact of the discretization on the performance of the heuristic. Based on the results in Section 5.3.2, the heuristic parameters are kept constant, with $Z_{\min} = 0.01$, $I = 30$, and $\theta = 0.70$, and we use the node removal heuristic. Table 8 presents the results of the experiments with instance 3. The results for the other instances are reported in Table B.4 in Online Appendix B. We test different values for the lengths of the time blocks and the step size of the SoC values (Steps). Range represents the interval of the possible SoC values.

First, note that time blocks of five minutes and a SoC step size of 3% result in small optimality gaps of at most 1.5% for instances A, B, 1, 2, and 3. Because 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 five minutes while keeping the possible SoC step size constant at 3% rarely results in a better solution. Using time blocks of two minutes instead of five 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 five minutes, the buses charge approximately 12.5%. In two 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 five minutes versus two minutes, respectively. Hence, using time blocks of two 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 one 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 one minute and steps of 1% for the possible SoC values are not executed for the other instances.

Table 8. Results of the Diving Heuristic for Different Discretization Levels

			Instance 3 (Lower bound = 1,288,645.26)						
TB (min)	Steps (%)	Range	Time (s)	It (#)	PP (s)	RMP (s)	Sol	G (%)	B (#)
2	3	22–100	4,567.74	2,047	1.814	0.113	1,296,698.42	0.6	25
5	3	22–100	1,535.45	1,545	0.892	0.052	1,295,483.00	0.5	25
10	3	22–100	1,075.97	1,782	0.516	0.051	1,303,144.58	1.1	25
30	3	22–100	602.33	1,785	0.272	0.043	1,380,834.87	7.2	26
5	1	22–100	5,615.26	1,693	3.222	0.052	1,342,841.02	4.2	26
5	3	22–100	1,535.45	1,545	0.892	0.052	1,295,483.00	0.5	25
5	6	22–100	902.11	1,863	0.358	0.058	1,298,415.51	0.8	25
5	10	20–100	677.28	2,015	0.180	0.066	1,390,020.58	7.9	27
5	20	20–100	14.87	236	0.056	0.004	10,836,975.37	741.0	208

Table 8 and Table B.4 in Online Appendix B also show the influence of choosing different step sizes than 3% for the SoC values for a time block length of five minutes. For instances A, B, 1, and 2, choosing a step size of 1% results in bus schedules with slightly better solution values, for example, because of fewer scheduled charging actions. For instance 3, the solution actually becomes worse. Moreover, this also more than triples the average time needed to solve the pricing problem per iteration. This increases the required computation time significantly.

For most instances, 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 more buses for instances A, 2, and 3.

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 five minutes. Thus, in this setting, charging actions do not actually 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 five minutes in combination with a step size of 3% gives the best solutions compared with using different lengths of 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 diving heuristic. We again also test the influence of the 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 turn our attention to solving instance 4. We do not deviate from the parameter values used in Section 5.3 for the diving heuristic. Hence, we use $Z_{\min} = 0.01$, $I = 30$, and $\theta = 0.70$. Using these parameter settings, we obtain 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, the time blocks of five minutes give the best solutions. We prefer using a step size of 3% because of the lower computation times compared with a step size of 1%, despite the slightly worse solutions.

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 influence of the node removal heuristic on the average time needed to solve the pricing problem. Recall that this extension removes specific nodes and arcs during the column generation process each time paths are fixed.

The results are presented in Table 9. We observe that the diving heuristic using a step size of 3% requires over six 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 3.21 seconds per iteration. This can be explained by the large number of nodes and arcs in the corresponding network (see Table 3).

Using a step size of 6%, the time needed to solve the pricing problem is significantly reduced to 1.28 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 about 5%. The total time needed for the column generation process is decreased to less than 2.5 hours.

Using the diving heuristic with node removal, we see a further decrease in the average time needed to solve the pricing problem. Now, on average 2.12 and 0.86 seconds are needed to solve the pricing problem per iteration for steps of 3% and 6%, respectively. Thus, the node removal

Table 9. Results of the Diving Heuristic (with and Without Node Removal) for Instance 4 Using $Z_{\min} = 0.01$, $I = 30$, $\theta = 0.70$, TB = 5 min, and Range = 22–100

Steps (%)	Node removal	Time (h)	It (#)	PP (s)	RMP (s)	Sol	G (%)	B (#)
3	No	6.10	6,068	3.21	0.22	1,546,398.05	1.2	30
6	No	2.33	5,080	1.28	0.21	1,627,588.56	6.5	31
3	Yes	3.45	4,965	2.12	0.22	1,546,634.28	1.2	30
6	Yes	1.43	4,269	0.86	0.20	1,571,575.71	2.8	30

Note. The lower bound for this instance equals 1,528,573.95.

Table 10. Results of the Diving Heuristic with Node Removal 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	Lower bound	G (%)	B (#)
5	6	22–100	6.76	10,665	1.29	0.58	2,776,499.51	2,702,417.41	2.7	53

heuristic significantly reduces the computation time. In particular, when using steps of 6%, the total computation time is reduced to less than 1.5 hours.

5.5.2. Instance 5: The Entire Concession. Based on the results of Section 5.5.1, we use the diving heuristic with node removal 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 five 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 with the number of nodes and arcs reported in Table 3.

We present the results of solving the entire concession in Table 10. The diving heuristic with node removal requires less than seven hours to solve instance 5. The optimality gap is below 3%. The optimal solution value of the RMP equals 2,755,589. Thus, roughly two-thirds of the optimality gap can be explained by the discretization and one-third 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, 823 trips are scheduled. This means that seven trips are driven empty.

Below, we discuss a few characteristics of the bus duties that comprise the final solution. Each bus charges around two hours on average during its duty. The average times of deadheading and idling per bus duty are approximately 65 and 134 minutes, respectively. Because of 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 64 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 34.7%, with an overall minimum of 27.9%. This is higher than the minimum allowed SoC value of 22%. In contrast, if the SoC is rounded down, all bus duties but one 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 because they do not exploit the entire range of allowed SoC values.

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 2. Because we incorporate the charging capacities in our methodology, the capacities of the charging stations are never exceeded. This can be seen in Figure 2. 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.

5.5.3. Practical Insights. Our solution method can be used to study the impact of the battery capacity or the number of chargers at the charging stations. To illustrate this, we apply our solution method to instances with

Figure 2. (Color online) The Occupation of the Charging Stations in the Resulting Bus Schedule

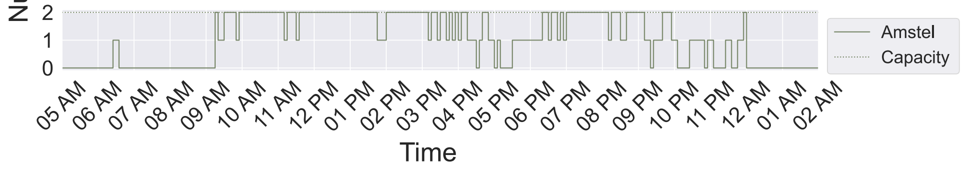
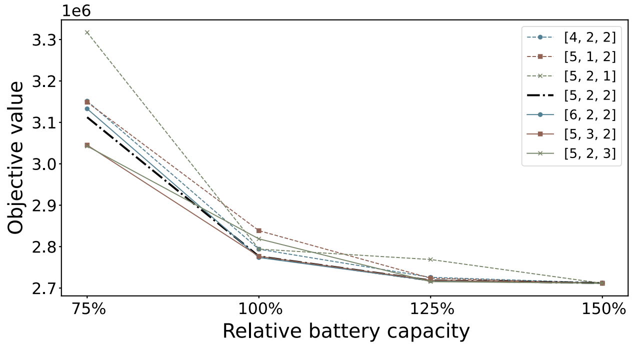


Figure 3. (Color online) The Influence of the Battery Capacity and the Capacity of the Charging Stations on the Solution Value



varying battery capacities and charging station capacities. In particular, we multiply both the battery capacity and the charging rate by 0.75, 1.0, 1.25, or 1.5. We change the battery capacity and the charging rate at the same time to ensure that the discretization does not influence the results (see the discussion in Section 5.4). Moreover, we either increase or decrease the number of chargers by one at each of the charging stations separately. The results of these experiments are depicted in Figure 3. In this figure, we plot the solution value against the relative battery capacity, for seven different settings. The settings differ in the number of chargers per charging location. Recall that the default instance is the one with five chargers at Hilversum and two chargers at both Huizen and Amsterdam.

Figure 3 demonstrates the influence of battery capacity, especially when the battery capacity is decreased to 75%, which increases the costs of the solution significantly. Overall, increasing the battery capacity leads to lower costs, although one can clearly see that this effect is diminishing.

The impact of the number of chargers is less prominent. Because of the heuristic nature of our solution method, adding one charger does not always lead to lower costs. For example, the costs obtained for the instance with 100% of the battery capacity and with five chargers at Hilversum, two at Huizen, and three at Amsterdam are higher than those for the default instance with two chargers in Amsterdam. Nevertheless, the costs for the instances with fewer chargers seem higher in general than those for instances with more chargers. The effect of the number of chargers is much smaller than the effect of the battery capacity, though.

6. Conclusion and Future Research

In this paper, we study the E-VSP considering both the capacity of charging stations and partial charging. To solve the problem, we develop a network structure in which every path represents a feasible duty and present two heuristics based on column generation. Valid lower bounds are obtained by solving an auxiliary problem on a slightly altered network.

Computational results based on a bus concession in the Netherlands show that the diving heuristic 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, the diving heuristic solves the entire concession with 816 trips to an optimality gap of 2.7%.

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.

Acknowledgments

The authors 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 EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):316–329.

Boland N, Hewitt M, Marshall L, Savelsbergh M (2017) The continuous time service network design problem. *Oper. Res.* 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 J. Transportaion Logist.* 8(2):195–216.

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

Desrosiers J, Lübbecke ME (2005) A primer in column generation. Desaulniers G, Desrosiers J, Solomon MM, eds. *Column Generation* (Springer, Boston), 1–32.

Gamache M, Soumis F, Marquis G, Desrosiers J (1999) Column generation approach for large-scale aircrew rostering problems. *Oper. Res.* 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, Koháni M (2019) Exact approach to the electric bus fleet scheduling. *Transportation Res. Procedia* 40:1380–1387.

Li JQ (2014) Transit bus scheduling with limited energy. *Transportation Sci.* 48(4):521–539.

Li L, Lo HK, Xiao F (2019) Mixed bus fleet scheduling under range and refueling constraints. *Transportation Res. Part C: Emerging Tech.* 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. *J. Advanced Transportation* 2020:8863905.

Olsen N, Kliewer N (2020) Scheduling electric buses in public transport: Modeling of the charging process and analysis of assumptions. *Logist. Res.* 13(4):1–17.

OV in Nederland (2021) Concessie Gooi- en Vechtstreek (2011–2021). Retrieved September 15, https://wiki.ovinnederland.nl/wiki/Concessie_Gooi-_en_Vechtstreek_(2011-2021)

Parmentier A, Martinelli R, Vidal T (2023) Electric vehicle fleets: Scalable route and recharge scheduling through column generation. *Transportation Sci.* 57(3):631–646.

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

Perumal SS, Lusby RM, Larsen J (2022) Electric bus planning & scheduling: A review of related problems and methodologies. *Eur. J. Oper. Res.* 301(2):395–413.

Posthoorn C (2016) Vehicle scheduling of electric city buses: A column generation approach. Unpublished master’s thesis, Delft University of Technology, Delft, Netherlands.

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. *INFORMS J. Comput.* 31(2):251–267.

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

Van Aken S, Hiemstra D (2020) Strategische keuzes bij zero-emissiebusvervoer met een beslissingsondersteunend algoritme Zero Emission Optimizer. Retrieved September 15, https://www.nationaalverkeerskundecongres.nl/papers-2020.

Van Kooten Niekerk ME, Van den Akker JM, Hoogeveen JA (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. *Comput. Oper. Res.* 76:73–83.

Wu W, Lin Y, Liu R, Jin W (2022) The multi-depot electric vehicle scheduling problem with power grid characteristics. *Transportation Res. 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 Res. Part E: Logist. Transportation Rev.* 154:102445.