Les Cahiers du GERAD ISSN: 0711–2440

A machine-learning-based column generation heuristic for electric bus scheduling

J. Gerbaux, Q. Cappart, G. Desaulniers

G–2024–13

January 2024

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

A machine-learning-based column generation heuristic for electric bus scheduling

Juliette Gerbaux [a]

Quentin Cappart [a]

Guy Desaulniers [a, b]

[a] *Département de mathématiques et de génie industriel, Polytechnique Montréal, Montréal, (Qc), Canada, H3T 1J4*

[b] *GERAD, Montréal (Qc), Canada, H3T 1J4*

January 2024
Les Cahiers du GERAD
G–2024–13

Abstract : Bus scheduling in public transit consists in determining a set of bus schedules to cover a set of timetabled trips at minimum cost. This planning process has evolved recently with the advent of electric buses that introduce constraints related to vehicle autonomy and battery charging process. In particular, column-generation algorithms have regained popularity for solving problems similar to the one considered in this paper, namely, the MDEVSP with a piecewise linear charging function and capacitated charging stations. To tackle large-scale MDEVSP instances, we design a column generation heuristic that relies on reduced-sized networks to generate the bus schedules. The reduction is achieved by selecting a priori a subset of the arcs. Multiple selection techniques are studied: some are based on a greedy heuristic and others exploit a supervised learning algorithm relying on a graph neural network. It turns out that combining both selection types yields the best computational results. On 405 artificial instances involving between 568 and 1474 trips and generated from real bus lines in Montreal, the network reduction technique produced an average computational time reduction of 71.6% while deteriorating solution cost by an average of 2.2%. On 8 larger instances containing more than 2500 trips on average, the proposed solution method also provided an average time saving of 52.5% with an average gap of 4.2% thanks to a transfer learning approach.

Keywords : Electric bus scheduling, column generation heuristic, network reduction, machine learning, greedy heuristic

Acknowledgements: We are thankful to the personnel of GIRO Inc. for providing initial datasets and discussing our progress throughout the project. This work was funded by GIRO Inc., the Natural Sciences and Engineering Research Council of Canada under the grant ALLRP 567169–21, and Prompt Québec under the grant PSO 171. This financial support was greatly appreciated.

1 Introduction

More and more cities are using battery-electric buses for public transit to reduce greenhouse gas emissions as well as pollution and noise. However, the deployment of electric buses requires rethinking the planning process of the bus operations due to new constraints imposed by the vehicle batteries. Depending on the recharging technology, different changes, described in Häll et al. (2019), have to be made.

Bus operations planning has been extensively studied in the literature and involves several important steps (Desaulniers and Hickman, 2007; Ibarra-Rojas et al., 2015): strategic planning, which establishes the transportation network by selecting the bus lines; tactical planning, which determines trip frequencies and schedules over the lines; and operational planning, which elaborates bus and driver schedules, among others. Each of these steps entails solving one or several combinatorial optimization problems to maximize service quality (passenger satisfaction) or minimize operational costs. The constraints considered in each problem can be infrastructure constraints (e.g., location of depots), budget constraints (e.g., maximum number of buses available) or operational constraints (e.g., a bus schedule must start and end at the same depot). One of the essential steps in this process is the vehicle scheduling problem (VSP). Given a set of timetabled trips and a set of available buses, the VSP consists in finding feasible bus schedules (sequences of trips) such that the total operating costs are minimized and all planned trips are serviced. The operating costs include, for example, fixed costs for each bus used as well as variable travel costs for empty bus movements (deadheads). The multi-depot VSP (MDVSP), which arises when the buses are assigned to multiple depots, is known to be NP-hard (Bertossi et al., 1987).

The algorithms developed for the VSP cannot be directly applied to the electric VSP (EVSP), which is also NP-hard (Sassi and Oulamara, 2017). Indeed, operating electric vehicles induces additional constraints such as limited vehicle autonomy due to battery capacity. The battery recharging process, which is non-linear, must also be taken into account in the mathematical formulation (Zhang et al., 2021a). Moreover, the location of the recharging stations and their available number of chargers must be considered.

Therefore, the EVSP needs to be further studied in order to efficiently use electric buses in practice. It is essential that the chosen model be realistic and that the applied solution method yields high-quality solutions in relatively short computational times. Although there has been several research works published in recent years (see below), several challenges still lie ahead before being able to address efficiently large-scale industrial problems. It is in this context that we conducted this research project on the multi-depot EVSP (MDEVSP) that was proposed to us by our industrial partner GIRO Inc., a leader in the commercialization of optimization software for public transit. Our goal is to accelerate a column generation (CG) heuristic, without deteriorating too much solution quality, by selecting a relatively small subset of the arcs to consider during the optimization process. To perform this arc selection, we explore the usage of a greedy heuristic and a deep learning architecture, namely, a graph neural network (GNN).

This paper is structured as follows. Section 2 presents a literature review on the EVSP and states our contributions. Section 3 defines the problem studied and proposes a mathematical formulation for it. Then, Sections 4 and 5 describes a CG heuristic for solving the EVSP and the arc selection techniques used to reduce heuristically the model size, respectively. Section 6 presents the computational experiments carried out and their results. Finally, conclusions are drawn in Section 7.

2 Literature review and contributions

The entire planning process at the strategic, tactical, and operational levels must be revisited to operate electric buses. For instance, decisions on bus acquisition and charging infrastructure design must be considered. An overview of papers addressing such issues can be found in Dirks et al. (2022)

and Perumal et al. (2022). Below, our literature review focuses on the (MD)EVSP. However, it is not exhaustive given the large number of works published recently and problem variants. More exhaustive reviews can be found in Perumal et al. (2022) and Gkiotsalitis et al. (2023).

Our review is divided in four parts: simplified recharging process (Section 2.1), nonlinear recharging process (Section 2.2), charger capacity constraints (Section 2.3), and GNNs in combinatorial optimization (Section 2.4). Our contributions with respect to this literature are presented in Section 2.5.

2.1 Works assuming a simplified recharging process

At first, only the autonomy of the vehicles and a constant recharging time of the batteries were taken into account. A first version of the EVSP was proposed by Chao and Xiaohong (2013), in which the vehicles are equipped with batteries that can be exchanged in a constant time throughout the day. It was solved using a genetic algorithm. Afterwards, several papers like Li (2014), Reuer et al. (2015), and Adler and Mirchandani (2017) developed heuristics for solving variants of the EVSP with a constant recharging time, always assuming that the battery is fully recharged during this time.

Next, less restrictive assumptions such as non-constant recharging time and partial recharging, have been considered. Wen et al. (2016) introduced a large neighborhood search (LNS) heuristic to solve an EVSP with a linear recharging time in function of the state-of-charge (SoC) increase and possible partial recharging. It can solve instances with up to 500 trips in about 20 minutes. Sassi and Oulamara (2017) modeled the recharging process using a 15-minute discretization: in each time interval, the charging power is a decision variable that determines the recharging speed. They considered a mixed fleet of electric and non-electric buses and developed a mixed-integer program as well as two matheuristics for solving their problem.

In the last few years, research on the EVSP has exploded with numerous works still assuming a linear recharging process. In particular, local-search based heuristics have been developed by Jovanovic et al. (2021), Jiang et al. (2022b) and Zhang et al. (2022). In Jovanovic et al. (2021), a greedy randomized adaptive search procedure that can solve large-scale real-life instances with up to a few thousand trips is developed. In Jiang et al. (2022b), the authors devised a LNS metaheuristic to solve an EVSP arising in Shenzhen, China, and involving close to 800 trips. In Zhang et al. (2022), another LNS algorithm is proposed to tackle a MDEVSP that considers multiple types of electric buses. This heuristic can solve instances with 200 trips in around 30 minutes.

The following matheuristics, i.e., heuristics based on mathematical programming tools, have also been proposed for solving EVSP variants. Wang et al. (2021) tackled the MDEVSP using a genetic algorithm that exploits the columns produced by a CG algorithm. On instances with up to 510 trips, their method turned out to be 40 times faster than a branch-and-price (BP) algorithm when duplicate coverage or non-coverage of some trips is allowed.

For a mixed-fleet MDEVSP with diesel and electric buses, Olsen et al. (2022) proposed a three-phase matheuristic that first determines a least-cost set of bus schedules without limited driving range by solving exactly a multi-commodity network flow problem, second rearranges these schedules using one of eight simple procedures to create a maximum number of schedules that can possibly be assigned to an electric bus, and third tries to insert charging operations to create feasible schedules for electric buses. This approach can solve instances with up to 10710 trips in 30 seconds but cannot guarantee the number of schedules that can be assigned to electric buses.

Jiang et al. (2022a) developed a BP algorithm that takes into account the recharging time in the CG subproblem, but the final recharging events are only planned in a post-processing step. With this approach, the authors report solving instances with up to 400 trips in less than 72 minutes. For such instances, the BP matheuristic yields better solutions than the LNS heuristic of Jiang et al. (2022b) but in larger computational times.

Finally, let us mention that several mixed-integer linear programming (MILP) models solvable exactly by a commercial solver have been proposed. However, these approaches do not scale well. To tackle larger instances, Alvo et al. (2021) devised a Benders decomposition algorithm that can solve exactly instances with up to 250 trips in less than one hour. The master problem assembles the trips into bus schedules and the Benders subproblems verify that these schedules are energy-feasible.

The impact of using a simplified recharging process is studied in Olsen and Kliewer (2020) and Zhang et al. (2021a). Furthermore, the interest of allowing partial recharging is demonstrated in Zhang et al. (2022).

2.2 Works with a nonlinear recharging process

In reality, the time required to recharge a battery is not linear with respect to the SoC variation (see Montoya et al., 2017). EVSP models considering nonlinear recharging time emerged in van Kooten Niekerk et al. (2017). These authors propose several MILP models and solution methods that rely on a discretization of the recharging process. The solution methods are: an exact branch-and-cut algorithm, a CG heuristic, and a Lagrangian relaxation heuristic combined with CG. The latter algorithm yields the best results and can solve instances with 543 trips in about 15 minutes.

With the main goal of analyzing the impact of considering nonlinear recharging times compared to linear ones, Olsen and Kliewer (2020) extended the local search heuristic of Adler and Mirchandani (2017) to take into account partial recharging and nonlinear recharging times. With this heuristic, they solved EVSP instances with up to 10710 trips.

Yıldırım and Yıldız (2021) tackled large-scale MDEVSP instances with several types of electric buses and diesel buses, several types of chargers, and partial nonlinear recharging. They developed a CG heuristic that can solve a first set of real-world instances with 6416 trips and 1685 buses in 3.5 hours on average and a second one with 846 trips and 99 buses in 1.7 hours on average.

Zhang et al. (2021b) introduced an exact BP approach to solve the EVSP. Their model considers battery degradation in their objective function but assumes full recharging which allows to define an efficient labeling algorithm for solving the CG subproblem. The largest test instances have 160 trips and are solved in about 2 hours.

2.3 Works with charger capacity constraints

One feature that we will consider is a limit on the number of buses that can recharge simultaneously at each charging station. Such charger capacity constraints typically arise for two reasons, namely, the available number of parking spots equipped with a charger and the power available from the grid.

Not many papers on pure (MD)EVSP have considered these constraints due to the difficulty to handle them. To simplify their treatment, some authors like Li (2014), Sassi and Oulamara (2017) and Zhang et al. (2021b) have chosen to use a discretized time horizon for the recharging operations, i.e., a recharge covers an integer number of consecutive periods. On the other hand, Alvo et al. (2021) avoid this approximation and use a continuous time horizon in their Benders' decomposition algorithm by handling these constraints in the linear programming subproblem that is responsible to find a feasible schedule for each available charger.

2.4 Works leveraging machine learning in combinatorial optimization

In a different context, there has been a surge in the prevalence of deep learning architectures (LeCun et al., 2015) over the past decade. This rise is attributed to advancements in computational capabilities and the abundance of available data, enabling the development of increasingly complex neural architectures that have achieved remarkable results across a wide range of tasks, spanning computer vision to natural language processing. As the machine learning community continues to seek new

frontiers and challenges, there has been a gradual adoption of deep learning techniques to address combinatorial optimization problems, as discussed by Bengio et al. (2021). Related to the MDVSP, Dauer and Prata (2021) propose to reduce the number of variables on a time-space network using a supervised approach.

A recent and widely embraced architectural approach for tackling combinatorial optimization problems is the GNN (Kipf and Welling, 2017). Much like convolutional neural networks, which are tailored for learning from spatial data such as images, GNNs specialize in learning from data structured in the form of graphs. Numerous variations of architectures based on GNNs are available, including those incorporating sampling techniques (Hamilton et al., 2017), attention mechanisms (Veličković et al., 2018), or gating mechanisms (Ruiz et al., 2020). This architectural paradigm has progressively gained attention for diverse combinatorial optimization domains, including integer programming (Gasse et al., 2019), constraint programming (Cappart et al., 2021), SAT solving (Selsam and Bjørner, 2019), decision diagrams (Cappart et al., 2022), and even column generation (Morabit et al., 2021). For a comprehensive examination of GNNs and their application in combinatorial optimization, we direct the reader to the survey conducted by Cappart et al. (2023). To the best of our knowledge, GNNs have not been considered yet for the (MD)EVSP.

2.5 Our contributions

As one can observe from this literature review, several types of algorithms have been proposed to solve the (MD)EVSP. Among them, there are several CG-based algorithms that allow to solve large-scale instances, while providing dual bounds to assess solution quality. Such a CG heuristic is commercialized by our industrial partner GIRO for solving real-life instances. The goal of this paper is to develop an efficient arc selection technique to speed up such algorithms without deteriorating too much solution quality. To do so, we explore different techniques and apply them to solve the (MD)EVSP with a nonlinear recharging process and charging station capacity constraints.

Our main scientific contributions can be stated as follows:

- We develop a fast greedy heuristic that can include some randomness to generate a subset of (possibly infeasible) solutions;
- We introduce a GNN architecture, trained in a supervised fashion from previously solved instances, that can provide an approximation of the probability that an arc is part of an optimal solution;
- We propose different arc selection procedures that rely on the results of the greedy heuristic or the GNN or both of them;
- We apply these selection procedures to speed up a basic CG heuristic.

To assess the proposed solution methods, we report computational results obtained on (MD)EVSP instances generated from real bus lines.

3 Problem statement and mathematical model

In this section, we first define the MDEVSP in detail, before presenting a mathematical formulation for it.

3.1 Problem statement

In public transit, the MDEVSP is often defined over a single day. Let T be a set of n trips to cover with a single bus each. Each trip $t \in T$ is defined by its departure and end terminals l_t^D and l_t^E, its departure and end times q_t^D and q_t^E, and its energy consumption e_t. These trips must be covered by

a fleet of identical electric buses that are spread in a set D of depots. Depot $d \in D$ has v^d available buses. We assume that any bus can cover any trip.

While scheduling electric buses, we ensure that they do not run out of energy. Thus, we impose that the SoC of each bus remains in an interval $[\underline{\sigma}, \overline{\sigma}]$ and assume that the buses are fully charged at $\overline{\sigma}$ at the start of the day. To recharge the buses in the middle of the day, there is a set H of recharging stations (at the depots or elsewhere). Station $h \in H$ is equipped with u^h chargers and, thus, a maximum of u^h buses can recharge simultaneously there. As suggested by Montoya et al. (2017), the charging process is modeled by a piecewise linear function f which, from an initial battery SoC σ^I and a charging time Δ, gives the final SoC $\sigma^F = f(\sigma^I, \Delta)$ (see Figure 1).

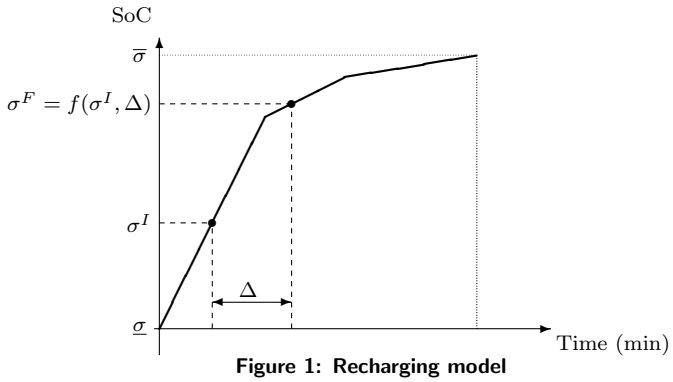


Figure 1: Recharging model

To track the number of buses present simultaneously at a charging station, the day is discretized into m consecutive δ-minute time intervals. Let $P = \{p_1, p_2, ..., p_m\}$ be the set of these periods and denote by τ_p^S and τ_p^E the start and end times of period $p \in P$, respectively, where $\tau_{p_i}^E = \tau_{p_{i+1}}^S$, $\forall i \in \{1, 2, \ldots, m-1\}$. We assume that any recharging operation must start at the beginning of a period and last an integer number of periods. In fact, recharging can stop before the last period if the SoC reaches $\overline{\sigma}$ but the vehicle continues to occupy the charger until the end of this period. Only one (non-preemptive) recharge is allowed per visit at a recharging station.

For each pair of points i and j in the transportation network (i.e., depots, recharging stations, start and end trip terminals), let ρ_{ij} and e_{ij} be the travel time and energy consumption between i and j when the bus is deadheading, i.e., without passengers.

A bus schedule is a sequence of trips, deadheads and recharging events. It is feasible if it starts and ends at the same depot, it respects the timetable of the visited trips, and the bus SoC always remains in $[\underline{\sigma}, \overline{\sigma}]$. Furthermore, to avoid long unproductive time for the drivers, we impose a maximum duration of β minutes between two trips t_1 and t_2 performed consecutively by the same bus (i.e., $q_{t_2}^D - q_{t_1}^E \leq \beta$). This rule applies only if no recharge or return to the depot (see below) is performed between t_1 and t_2. On the other hand, it is possible to return temporarily to any depot during the day, but if so, the bus must be parked there for a minimum time γ.

The cost structure is as follows. Each used bus incurs a fixed cost c^F and a variable cost associated with its schedule. The cost of a schedule is separated into four parts: a cost c^W for each time unit waiting outside the depots (including any time spent at the recharging stations), an empty travel cost c_{ij}^D for deadheading between locations i and j, a penalty cost c^R for each return to a depot during the day, and a penalty cost c^H for each charging event. There is no waiting cost at the depots because the driver can, typically, leave the vehicle without attendance and there is no travel cost on the trips because they must be covered exactly once, incurring an overall constant cost. The penalty costs c^R and c^H are considered to avoid unnecessary recharging operations and returns to the depot.

The goal of the MDEVSP is to find a set of feasible bus schedules that covers each trip once while minimizing the total cost and respecting the capacity of the depots and charging stations.

3.2 Mathematical formulation

The MDEVSP can be formulated as a set partitioning problem with side constraints. This formulation relies on the following additional notation. Let S^d be the set of feasible schedules for the buses housed in depot d. For a schedule $s \in S^d$, $d \in D$, we define the following parameters: c_s is the schedule cost; a_s^t is a binary parameter equal to 1 if trip $t \in T$ is covered by schedule s and 0 otherwise, and; b_s^{ph} is a binary parameter equal to 1 if schedule s uses a charger at station $h \in H$ in time period $p \in P$ and 0 otherwise. Furthermore, we associate a binary variable x_s that is equal to 1 if schedule s is used in the solution and 0 otherwise.

The MDEVSP can then be expressed as follows:

$$\min \sum_{d \in D} \sum_{s \in S^d} c_s x_s \tag{1}$$

$$\text{s.t.:} \sum_{d \in D} \sum_{s \in S^d} a_s^t x_s = 1, \qquad \forall t \in T, \tag{2}$$

$$\sum_{d \in D} \sum_{s \in S^d} b_s^{ph} x_s \leq u^h, \qquad \forall p \in P, h \in H, \tag{3}$$

$$\sum_{s \in S^d} x_s \leq v^d, \qquad \forall d \in D, \tag{4}$$

$$x_s \in \{0, 1\}, \qquad \forall d \in D, s \in S^d. \tag{5}$$

Objective function (1) aims at minimizing the total cost of the schedules. Constraints (2) ensure that each trip is serviced by exactly one bus. Constraints (3) model the capacity of the recharging stations in each time period. Constraints (4) enforce the capacity of the depots. Finally, constraints (5) impose binary requirements on the schedule variables. In practice, model (1)–(5) involves a huge number of variables, namely, one per feasible schedule. As discussed next, CG can be used to handle them.

4 Basic CG heuristic

For many years, CG heuristics have been used to solve various large-scale vehicle and crew scheduling problems (see, e.g., Desaulniers et al., 2005; Sadykov et al., 2019). Such a matheuristic applies CG to solve linear relaxations and derive integer solutions by partially exploring a branch-and-bound search tree. In this section, we first present our CG algorithm (Subection 4.1) before describing our branching strategy (Subsection 4.2). Finally, three basic speedup techniques are briefly exposed in Subsection 4.3.

4.1 Column generation

CG is an iterative algorithm that is first used to solve the linear relaxation of model (1)–(5) that is called the master problem (MP). At each iteration ℓ, CG solves the MP restricted to a subset $S_\ell^d \subset S^d$ of the feasible schedules for each depot $d \in D$. This problem, called the restricted MP (RMP) and denoted RMP^ℓ at iteration ℓ, writes as follows:

$$\min \sum_{d \in D} \sum_{s \in S_\ell^d} c_s x_s \tag{6}$$

$$\text{s.t.:} \sum_{d \in D} \sum_{s \in S_\ell^d} a_s^t x_s = 1, \qquad \forall t \in T, \tag{7}$$

$$\sum_{d\in D}\sum_{s\in S^d_\ell} b^{ph}_s x_s \le u^h, \qquad \forall p\in P, \forall h\in H, \tag{8}$$

$$\sum_{s\in S^d_\ell} x_s \le v^d, \qquad \forall d\in D, \tag{9}$$

$$x_s \ge 0, \qquad \forall d\in D, s\in S^d_\ell. \tag{10}$$

Solving RMP^ℓ yields a primal solution (assuming it is feasible) and a dual solution $(\pi^\ell, \nu^\ell, \eta^\ell)$, where π^ℓ, ν^ℓ, and η^ℓ are the vectors of dual variables associated with constraints (7)–(9), respectively. Then, CG solves a subproblem for each depot $d \in D$ whose goal is to find new variables (columns) with a negative reduced cost, where the reduced cost $\bar{c}^\ell_s$ of a variable x_s, $s \in S^d$, at iteration ℓ is

$$\bar{c}^\ell_s = c_s - \sum_{t\in T}\pi^\ell_t a^t_s - \sum_{p\in P}\sum_{h\in H}\nu^\ell_{ph} b^{ph}_s - \eta^\ell_d. \tag{11}$$

If no negative reduced cost columns are found for all depots, the optimal solution to RMP^ℓ is also optimal for the MP. Otherwise, negative reduced cost columns are added to RMP^ℓ to yield $RMP^{\ell+1}$, and a new iteration must be executed.

For the MDEVSP, each CG subproblem is a shortest path problem with resource constraints (SPPRC, see Irnich and Desaulniers, 2005) that is defined on a specific network and can be solved by a labeling algorithm. This network and this algorithm are described below.

4.1.1 Subproblem network and resource constraints

The SPPRC for depot $d \in D$ is defined on an acyclic network $\mathcal{G}^d = (\mathcal{V}^d, \mathcal{A}^d)$, where $\mathcal{V}^d$ and $\mathcal{A}^d$ are its vertex and arc sets, respectively. This network (see Figure 2) has the structure of a direct connection network between the trips (to easily deal with the time limit between consecutive trips) and that of a time-space network for the recharging stations and the intra-day depot returns.

In $\mathcal{V}^d$, there is a source node o^d and a sink node k^d that represent depot d at the beginning and the end of the day, respectively. There is a trip node t for each trip $t \in T$. For each charging station $h \in H$ and time period $p \in P$, a waiting node w^h_p and a recharging node r^h_p are created to represent a vehicle ready to wait or charge at station h at the beginning of period p. Furthermore, for each depot $d' \in D$ and period $p \in P$, there is a node $s^{d'}_p$ representing the possibility of idling at the depot. Hence, a discretized timeline is also used at the depots to obtain an acyclic network.

Arc set $\mathcal{A}^d$ is built as follows. For each trip $t \in T$, there is a (black) pull-out arc (o^d, t) and a (black) pull-in arc (t, k^d). For each pair of distinct trips $t_i \in T$ and $t_j \in T$ that can be feasibly covered consecutively by the same bus ($\rho_{l^E_{t_i}, l^D_{t_j}} \le q^D_{t_j} - q^E_{t_i} \le \beta$), there is a (red) connection arc (t_i, t_j). Next, for each trip $t \in T$ and each charging station $h \in H$, we create a (blue) from-station arc $(w^h_{p^L}, t)$, where p^L is the latest period in P such that $\tau^E_{p^L} + \rho_{h,l^D_t} \le q^D_t$. Similarly, a (blue) to-station arc $(t, w^h_{p^E})$ is created, where p^E is the earliest period in P such that $q^E_t + \rho_{l^E_t,h} \le \tau^S_{p^E}$. Then, for each trip $t \in T$ and depot $d' \in D$, a (green) from-depot arc $(s^{d'}_{p^L}, t)$ is defined, where p^L is the latest period in P such that $\tau^E_{p^L} + \rho_{d',l^D_t} \le q^D_t$. Moreover, we define a (green) to-depot arc $(t, s^{d'}_{p^E})$, where p^E is the earliest period in P such that $q^E_t + \rho_{l^E_t,d'} + \gamma \le \tau^S_{p^E}$, where γ ensures that the bus spends at least γ minutes in the depot. Also, for each recharging station $h \in H$ and time period $p_i \in P$, we create (brown) waiting arcs $(w^h_{p_i}, w^h_{p_{i+1}})$, (brown) start-of-recharging arcs $(w^h_{p_i}, r^h_{p_{i+1}})$ and (brown) recharging arcs $(r^h_{p_i}, r^h_{p_{i+1}})$ associated both with period p_i, as well as end-of-recharging arcs $(r^h_{p_i}, w^h_{p_i})$. As shown in Figure 2, some of these arcs are not defined when $p_i = p_1$ or $p_i = p_m$. For each station h, there is also a (black) pull-in arc $(w^h_{p_m}, k^d)$ to model the possibility to end a schedule by a recharging event to reach the depot. The symmetric option is not considered at the beginning of the day because we assume that all buses are fully recharged at night. Finally, for each depot $d' \in D$, there is a sequence of (orange) idle arcs $(s^{d'}_{p_i}, s^{d'}_{p_{i+1}})$, $i = 1, 2, \ldots, m-1\}$ allowing to extend the time spent at the depot in the middle of the day.

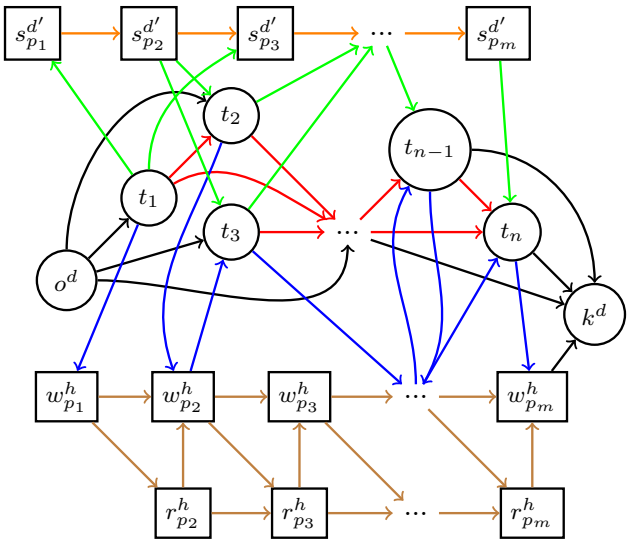


Figure 2: Subproblem network $\mathcal{G}^d$ for depot d

Each arc $(i,j) \in \mathcal{A}^d$ has a *real* cost c_{ij} that is computed according to the structure explained at the end of Section 3.1. In particular, the fixed cost c^F, the return-to-the-depot penalty c^R, and the charging-event penalty c^H are included in the cost of all pull-out arcs, to-depot arcs, and to-station arcs, respectively. To compute the reduced cost of the schedule variables x_s in the subproblem at iteration ℓ, these arc costs c_{ij} are replaced by *adjusted* costs $\bar{c}^\ell_{ij}$ that take into account the dual solution $(\pi^\ell, \nu^\ell, \eta^\ell)$ of RMP^ℓ as follows:

$$\bar{c}^\ell_{ij} = \begin{cases} c_{ij} - (\eta^\ell_d + \pi^\ell_j) & \text{if } (i,j) \text{ is a pull-out arc} \\ c_{ij} - \pi^\ell_j & \text{if } i \neq o^d \text{ and } j \text{ is a trip node} \\ c_{ij} - \nu^\ell_{p_k h} & \text{if } j = r^h_{p_{k+1}} \text{ is a recharging node} \\ c_{ij} & \text{otherwise.} \end{cases} \tag{12}$$

With these adjusted costs, the cost of any $o^d - k^d$ path corresponding to a feasible schedule $s \in S^d$ is equal to its reduced cost $\bar{c}^\ell_s$ as defined by Equation (11). With each arc $(i,j) \in \mathcal{A}^d$, we also associate an energy consumption

$$\tilde{e}_{ij} = \begin{cases} e_i + e_{l_i \bar{l}_j} & \text{if } i \text{ is a trip node} \\ e_{l_i \bar{l}_j} & \text{otherwise,} \end{cases} \tag{13}$$

where l_i is the terminal l^E_i if i is a trip node or the location associated with node i otherwise, while $\bar{l}_j$ is the terminal l^D_j if j is a trip node or the location associated with node i otherwise.

Every feasible schedule in S^d corresponds to an $o^d - k^d$ path in $\mathcal{G}^d$. However, not all $o^d - k^d$ paths represent a feasible schedule as the SoC and charging constraints are not explicitly taken into account in the network structure. These constraints are modeled as resource constraints using three different resources. The first resource is the SoC, the second ensures that any recharging event is not split in multiple parts, while the third forces a recharging event to take place at every visit to a recharging station. The usage of these resources is detailed in the next subsection.

4.1.2 Labeling algorithm

The SPPRC for depot $d \in D$ can be solved using a labeling algorithm (see Irnich and Desaulniers, 2005). Starting with an initial partial path containing only source node o^d, this algorithm extends iteratively partial paths until reaching sink node k^d. To avoid enumerating all possible paths, infeasible paths (those violating the resource constraints) and paths that are proven to be dominated by others are discarded during the course of the algorithm.

A partial path p ending at a node $i \in \mathcal{V}^d$ is represented by a label $\Lambda_p = (\Gamma_p^{rCost}, \Gamma_p^{SoC}, \Gamma_p^{NoR}, \Gamma_p^{RchR})$ associated with this node, where the label components provide the following information about path p:

Γ_p^{rCost}**:** Reduced cost up to node i;
Γ_p^{SoC}**:** SoC at node i;
Γ_p^{NoR}**:** Number of recharging events since the last trip;
Γ_p^{RchR}**:** Indicator equal to 1 if a recharging event is still required before the next trip and to 0 otherwise.

The labeling algorithm starts with an initial label $\Lambda_{p_0} = (0, \overline{\sigma}, 0, 0)$ at node o^d that represents partial path $p_0 = o^d$. Extending a partial path p ending at node i or, equivalently, its label $\Lambda_p = (\Gamma_p^{rCost}, \Gamma_p^{SoC}, \Gamma_p^{NoR}, \Gamma_p^{RchR})$, along an arc $(i,j) \in \mathcal{A}^d$ yields a new label $\Lambda_{p'} = (\Gamma_{p'}^{rCost}, \Gamma_{p'}^{SoC}, \Gamma_{p'}^{NoR}, \Gamma_{p'}^{RchR})$ at node j whose components are computed using the following resource extension functions at iteration ℓ:

$$\Gamma_{p'}^{rCost} = \Gamma_p^{rCost} + \bar{c}_{ij}^{\ell} \tag{14}$$

$$\Gamma_{p'}^{SoC} = \begin{cases} \min\{\overline{\sigma}, f(\Gamma_p^{SoC}, \delta)\} & \text{if } (i,j) \in \mathcal{A}_{SR}^d \cup \mathcal{A}_R^d \\ \Gamma_p^{SoC} - \tilde{e}_{ij} & \text{otherwise} \end{cases} \tag{15}$$

$$\Gamma_{p'}^{NoR} = \begin{cases} \Gamma_p^{NoR} + 1 & \text{if } (i,j) \in \mathcal{A}_{SR}^d \\ \Gamma_p^{NoR} - 1 & \text{if } (i,j) \in \mathcal{A}_{FS}^d \\ \Gamma_p^{NoR} & \text{otherwise} \end{cases} \tag{16}$$

$$\Gamma_{p'}^{RchR} = \begin{cases} \Gamma_p^{RchR} + 1 & \text{if } (i,j) \in \mathcal{A}_{TS}^d \\ \Gamma_p^{RchR} - 1 & \text{if } (i,j) \in \mathcal{A}_{SR}^d \\ \Gamma_p^{RchR} & \text{otherwise} \end{cases} \tag{17}$$

where $\mathcal{A}_{SR}^d$, $\mathcal{A}_R^d$, $\mathcal{A}_{FS}^d$, and $\mathcal{A}_{TS}^d$ are the subsets of start-of-recharging, recharging, from-station, and to-station arcs, respectively. Observe that the extension function (15) allows to stay in a recharging mode after reaching $\overline{\sigma}$, the maximum SoC. In this case, the bus continues to occupy a charger but stops recharging. Label p' is deemed infeasible and discarded if at least one of the following conditions holds: i) $\Gamma_{p'}^{SoC} < \underline{\sigma}$, ii) $\Gamma_{p'}^{NoR} > 1$, and iii) j is a trip node and $\Gamma_{p'}^{RchR} = 1$. To avoid enumerating all feasible paths, the following dominance rule is applied.

Definition 1. Let $\Lambda_{p_i} = (\Gamma_{p_i}^{rCost}, \Gamma_{p_i}^{SoC}, \Gamma_{p_i}^{NoR}, \Gamma_{p_i}^{RchR})$, $i = 1, 2$, be two labels associated with the same node. Label Λ_{p_1} dominates label Λ_{p_2} if

$$\Gamma_{p_1}^{rCost} \leq \Gamma_{p_2}^{rCost}, \quad \Gamma_{p_1}^{SoC} \geq \Gamma_{p_2}^{SoC}, \quad \Gamma_{p_1}^{NoR} \leq \Gamma_{p_2}^{NoR}, \quad \Gamma_{p_1}^{RchR} \leq \Gamma_{p_2}^{RchR}.$$

All dominated labels are discarded except when two labels dominate each other, in which case one of them must be kept.

4.2 Integer solutions

To derive integer solutions, we partially explore a search tree. We can apply in order of priority the following three types of decisions (the parameter values below have been selected based on preliminary tests):

- Fixing to 1 the value of at most 10 variables x_s that take a fractional value larger than or equal to 0.7. A single child node is created.
- Fixing to 1 the (direct or indirect) flow between two consecutive trips for at most 50 pairs of trips whose flow is fractional and larger than or equal to 0.7, or for a single pair (that with the largest flow) if none have a fractional flow larger than or equal to 0.7. Such inter-trip constraints are imposed in the CG subproblem by modifying the labeling algorithm (see Irnich and Desaulniers, 2005). A single child node is created.

- Branching on an arc flow if it is fractional. The decisions are imposed by adding a constraint in the MP. Two child nodes are created.

Most of the times, the first two types are sufficient to complete the solution process and find a high-quality integer solution. This corresponds to a diving heuristic (see Sadykov et al., 2019). In rare cases, we need to resort to the third decision type. Note that CG is re-started after imposing the decisions.

4.3 Speedup techniques

Despite using CG and heuristic branching, the convergence of the algorithm can be very long for large instances, partly because they become highly degenerated. To fight degeneracy, perturbation of the trip coverage constraints (7), which allows to slightly under- and over-cover each trip, is applied as proposed in Desfontaines and Desaulniers (2018). Perturbation is deactivated when no decisions of types 1 and 2 (for a fractional value of at least 0.7) can be imposed.

To further accelerate the algorithm, CG is terminated prematurely at each node of the search tree when the optimal value of the current RMP has decreased by less than a predefined threshold (set to 100 for our tests) over a given number of iterations (3 for our tests).

Finally, for the very large instances for which solving the subproblem is highly time-consuming, we use an aggressive heuristic dominance rule in the labeling algorithm that considers only the test on the reduced cost component. In each node of the search tree, this heuristic rule is used to generate columns until the CG process starts to stall (the optimal value of the RMP has decreased by less than 100 in the last 5 iterations).

5 Heuristics for reducing the size of the networks

Despite the speedup techniques described previously, the execution time of the CG heuristic can still be prohibitive for solving very large instances. To tackle this issue, we propose to reduce the complexity of the SPPRC subproblem by shrinking the size of the related networks $\mathcal{G}^d = (\mathcal{V}^d, \mathcal{A}^d)$, $d \in D$ (see Figure 2). Instead of executing the labeling algorithm on a complete $\mathcal{G}^d$ network, our idea is to execute it on a subnetwork $\hat{\mathcal{G}}^d = (\mathcal{V}^d, \hat{\mathcal{A}}^d)$ such that $\hat{\mathcal{A}}^d \subset \mathcal{A}^d$. In other words, all the vertices are present in a subnetwork but only a subset of the arcs is included. The following two options can be considered: defining the subnetworks $\hat{\mathcal{G}}^d$, $d \in D$, only once before starting CG or redefining them at each CG iteration taking into account the current RMP dual solution. We have chosen the first option because the second one is more complex and, thus, requires devising a procedure that might induce a too large overhead at each iteration. Nevertheless, determining which arcs can be safely removed without altering the final solution is nontrivial. When we remove arcs, there is a potential risk of diminishing the solution quality as the SPPRC subproblems might not be able to generate all necessary paths. Another consideration is to determine *how many arcs should be kept in the subnetworks*. Pruning too many arcs may result in a poor-quality solution, but on the other hand, being too conservative will not have a significant impact on the execution time that we want to reduce. Therefore, there is an appropriate balance to achieve. Finally, even if this selection procedure is executed only once before starting CG, it should be relatively fast.

We introduce three approaches for reducing the subproblem networks by selecting appropriate arcs. The first approach is a greedy heuristic exploiting the problem structure (Section 5.1). The second one is based on supervised learning and leverages a GNN (Section 5.2). Finally, the last approach combines both of them (Section 5.3). These three arc selection approaches focus only on the connection, from-station, to-station, from-depot and to-depot arcs. These arcs are called the *selectable* arcs. All other (black, brown, and orange) arcs are always kept in the $\hat{\mathcal{G}}^d$ subnetworks to ensure feasibility.

5.1 Selection with a greedy heuristic

Inspired by Adler and Mirchandani (2017), we design a greedy heuristic for solving the MDEVSP without taking into account station capacity. Then, only the selectable arcs present in the solution are included in $\hat{\mathcal{A}}^d$ when building subnetwork $\hat{\mathcal{G}}^d$, independently of the depot associated with the bus traversing these arcs in the solution. Briefly, our greedy heuristic consists in maintaining a set of partial paths, and to append each trip to the path incurring a minimal immediate insertion cost. When there is no direct way to add a trip to a path, we complete the path with a detour towards a charging station or a depot. If it is still not possible to add the trip, we create a new path starting from a depot having an available bus. We realistically assume that the bus fleet is sufficient to cover all trips. Otherwise, we dynamically increase it. The heuristic is formalized in Algorithm 1, where z^d indicates the current number of buses used from depot $d \in D$ and the built paths are stored in a dictionary data structure $\mathcal{R}$ indexed by the depot d and vehicle v associated with the path.

Algorithm 1: Greedy heuristic for arc selection

1 ▷ **Pre:** D is the set of depots.
2 ▷ **Pre:** $\mathcal{G} = \{\mathcal{G}^1, \dots, \mathcal{G}^{|D|}\}$ are the networks for each depot.
3 ▷ **Pre:** T is the set of trips, sorted by departure time q_t^D.
4 ▷ **Pre:** o^d, k^d, v^d are parameters for each $d \in D$ (Section 3).
5
6 $\mathcal{R}[d, v] := o^d \quad \forall d \in D \wedge \forall v \in \{1, \dots, v^d\}$
7 $z^d := 0 \quad \forall d \in D$
8 **forall** $t \in T$ **do**
9 $\quad E_r := \mathsf{getAdmissiblePaths}(\mathcal{R}, t, \mathcal{G})$
10 $\quad E_c := \mathsf{getAdmissibleChargingStations}(\mathcal{R}, t, \mathcal{G})$
11 $\quad E_d := \mathsf{getAdmissibleDepots}(\mathcal{R}, \mathcal{G})$
12 $\quad$ **if** $E_r \neq \emptyset$ **then**
13 $\quad\quad (d, v) := \mathsf{getCheapestPath}(\mathcal{R}, t, E_r, \mathcal{G})$
14 $\quad\quad \mathcal{R}[d, v] := \mathcal{R}[d, v] \oplus t$
15 $\quad$ **else if** $E_c \cup E_d \neq \emptyset$ **then**
16 $\quad\quad (d, v, r) := \mathsf{getCheapestExtension}(\mathcal{R}, t, E_c, E_d, \mathcal{G})$
17 $\quad\quad \mathcal{R}[d, v] := \mathcal{R}[d, v] \oplus r \oplus t$
18 $\quad$ **else**
19 $\quad\quad d := \mathsf{argmin}_{d \in D}\{\mathsf{getCost}(o^d, t) \mid d \in D \wedge z^d < v^d\}$
20 $\quad\quad z^d := z^d + 1$
21 $\quad\quad \mathcal{R}[d, z^d] := \mathcal{R}[d, z^d] \oplus t$
22 **forall** $d \in D$ **do**
23 $\quad$ **forall** $v \in \{1, \dots, z^d\}$ **do**
24 $\quad\quad r := \mathsf{getCheapestFeasiblePathToDepot}(\mathcal{R}, d, v, \mathcal{G}^d)$
25 $\quad\quad \mathcal{R}[d, v] := \mathcal{R}[d, v] \oplus r \oplus k^d$
26 $\hat{\mathcal{Q}} := \mathtt{getAllSelectedArcs}(\mathcal{R}, \mathcal{G})$
27 **return** $\hat{\mathcal{Q}}$

First, a partial path corresponding to an empty schedule is initialized for each available vehicle v in each depot d (line 6) and the current number of vehicles used is set to 0 (line 7). Then, all trips $t \in T$, sorted in increasing order of their departure time, are considered for insertion in a partial path (line 8). Three sets are computed at each iteration: (1) E_r, which includes all the existing paths on which trip t can be directly appended without violating the energy and time constraints, (2) E_c, which includes the set of charging stations for which there exists at least one path that can be extended to cover trip t by going through this station, and (3) E_d, which includes the set of depots for which there exists at least one path that can be extended to cover trip t by letting the vehicle wait at this depot before trip t (lines 9 to 11).

There are three options for inserting a trip. First, if trip t can be directly appended to an existing path without detours (line 12), we get the partial path $\mathcal{R}[d, v]$ for which the insertion cost is minimal (line 13) and we add the trip to this path (line 14). The operator $x \oplus y$ represents extending a partial

path x with y. Second, if the trip can be inserted to an existing path only by resorting to a detour through a charging station or a depot (line 15), we get the path $\mathcal{R}[d, v]$ with the minimal insertion cost subject to a detour denoted r (line 16). The trip is then inserted into the path, after the detour (line 17). Third, if there is no option to insert the trip in an existing path (line 18), we start a new path from the cheapest depot that has an available vehicle (line 19). The number of available vehicles is then updated (line 20), and the trip is inserted (line 21).

At this step, we have a set of partial paths covering all trips in T. The last loop consists in closing each path by finding the cheapest feasible extension to the vehicle's depot (lines 22–25). We ensure that a vehicle can always go back to its depot by enforcing a safety margin on the battery level. Finally, we build the subset $\hat{\mathcal{Q}}$ of retained selectable arcs by taking all selectable arcs appearing in a path of the solution (line 26), and return this set $\hat{\mathcal{Q}}$ that applies to all depots $d \in D$ (line 27). Note that the number of chargers available at a charging station is not considered in the algorithm and that the solution obtained may, consequently, be infeasible. This is a reason why all non-selectable arcs are kept in the subnetworks. The execution time of this algorithm is negligible compared to the CG algorithm executed afterwards.

5.1.1 Adding randomness in the heuristic

Incorporating randomness is a common enhancement strategy to increase diversification in a greedy heuristic. We introduce randomness in Algorithm 1 as follows. First, instead of selecting the cheapest arcs for insertion, e.g. in line 19, we choose one arc from the set of the n^R most affordable arcs using a uniform distribution. Second, the algorithm is executed a total of n^S times. Here, n^R and n^S are predefined parameters. This random strategy is also applied on lines 13 and 16. The final set of selectable arcs returned encompasses the union of the arc subsets obtained through these n^S executions.

5.2 Selection with a graph neural network

A drawback of the greedy heuristic lies in its strategy for path extension, which primarily relies on selecting arcs with the least immediate insertion cost. In cases where finding an optimal path necessitates the inclusion of costly immediate arcs, the greedy approach may prove inefficient. Rather than seeking to identify these arcs through a manually crafted algorithm, we propose *learning* to identify them from historical data using a GNN.

Let us consider the *aggregated* network $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ underlying the subproblems, where $\mathcal{V} = \cup_{d \in D} \mathcal{V}^d$ and $\mathcal{A} = \cup_{d \in D} \mathcal{A}^d$. Let $y_{v,u} \in \{0, 1\}$ be a binary value associated with arc $(v, u) \in \mathcal{A}$ indicating whether arc (v, u) belongs to a good-quality MDEVSP solution obtained with the basic CG heuristic described in Section 4. We define k as the number of arcs in $\mathcal{A}$ and $\mathcal{Y} = \{y_1, \ldots, y_k\}$ as the set of all indicator arc values in network $\mathcal{G}$. Briefly, our supervised learning procedure consists in training a function $\Phi : \mathcal{G} \to [0, 1]^k$ from a dataset $\mathcal{D} = \{(\mathcal{G}_1, \mathcal{Y}_1), \ldots, (\mathcal{G}_m, \mathcal{Y}_m)\}$ of m previously solved MDEVSPs. We note that the number of vertices or arcs in the m networks from can be different. This function takes as input an aggregated network (i.e., an MDEVSP instance) and output a *probability* for each arc to be part of the solution obtained with the CG algorithm. We propose to build this function thanks to a GNN. This approach is elaborated next, starting with an overview of the feature information employed for training, followed by the GNN's architecture and the specifics of the training process.

5.2.1 Description of features

All vertices and arcs are decorated with a list of features. Intuitively, such features embed the characteristics of the MDEVSP that must be solved. Let $\phi_v^d \in \mathbb{R}^{9+2n^L}$ and $\psi_a^d \in \mathbb{R}^8$ be the vectors of features associated with a vertex $v \in \mathcal{V}^d$ and an arc $a = (v, u) \in \mathcal{A}^d$, respectively, where n^L is the number of physical locations (depots, terminals, charging stations) in the transportation network. The vertex features include the vertex type, the starting and ending times of the associated activity, the physical

locations associated with the vertex and, for the trip vertices, the numbers of trips that arrive shortly before the trip departure and that depart shortly after the trip arrival. On the other hand, the arc features include the arc cost, energy consumption, deadhead travel time, and waiting time. Furthermore, for trip-to-trip arcs, some cost and departure time rankings are also considered to assess the quality of this connection with respect to other possible ones. The detailed features are provided in A. Before using these features in the GNN, two preprocessing steps are carried out: first, each feature is considered as a real value (required for the training with a neural network) and second, values are normalized between 0 and 1 to stabilize the training.

5.2.2 Architecture of the graph neural network

In essence, the objective of a GNN is to transform the data within a network into a p-dimensional tensor for each vertex within the network. This transformation process entails an iterative refinement of the information associated with each vertex, achieved by gathering and consolidating information from neighboring vertices. Each step in this information aggregation process is commonly termed a *layer* of the GNN, and encompasses parameters that need to be learned.

Let $h_v^l \in \mathbb{R}^p$ be a p-dimensional vector representation of vertex $v \in \mathcal{V}$ at layer l and $g_{v,u}^l \in \mathbb{R}^q$ a q-dimensional representation of arc $(v,u) \in \mathcal{A}$ at layer $l \in \{0,1,\dots,L\}$. The inference process of a GNN consists in computing the next representations (h_v^{l+1} and $g_{u,v}^{l+1}$) from the previous ones for each vertex and arc. This operation can be executed through various approaches and is commonly referred to as *message passing*. In our specific context, we adopt the formalization proposed by Joshi et al. (2022). First, we set $h_v^0 = \theta_1 \phi_v$ and $g_{v,u}^0 = \theta_2 \psi_{v,u}$. This corresponds to a linear projection, where θ_1 and θ_2 are two tensors of weights that are determined during training. Then, the representations at each layer $l+1$ from 0 to $L-1$ are formalized as follows.

$$\mu_v^l = \max_{u \in \mathcal{N}(v)} \left(\sigma(g_{v,u}^l) \theta_3^l h_u^l \right) \qquad \forall v \in \mathcal{V} \tag{18}$$

$$h_v^{l+1} = h_v^l + \mathsf{ReLU}\Big(\mathsf{Norm}\big(\theta_4^l h_v^l + \mu_v^l \big) \Big) \qquad \forall v \in \mathcal{V} \tag{19}$$

$$g_{v,u}^{l+1} = g_{v,u}^l + \mathsf{ReLU}\Big(\mathsf{Norm}\big(\theta_5^l g_{v,u}^l + \theta_6^l h_v^l + \theta_7^l h_u^l \big) \Big) \qquad \forall (v,u) \in \mathcal{A} \tag{20}$$

Parameters $\theta_3^l, \dots, \theta_7^l$ are additional weight tensors for each layer l, σ is the sigmoid function, $\mathsf{ReLU}(x) = \max(0,x)$ is a non-linear activation function (Glorot et al., 2011), $\mathcal{N}(v)$ is the neighborhood of vertex v, and Norm is a normalization function. Such operations are executed for each vertex and each arc. The information from each neighbor is first aggregated by taking their maximum value in (18) which is then used in (19) for computing a vertex representation at the next layer. The representation for an arc is computed in (20) using the representations of the arc and its two vertices at the previous layer.

Let $\hat{y}_{v,u} \in [0,1]$ be the predicted probability that arc (v,u) is part of the MDEVSP solution. The last step is to compute such prediction from h_v^L as stated in the following equations:

$$h = \frac{1}{|\mathcal{V}|} \sum_{u \in \mathcal{V}} h_u^L \tag{21}$$

$$\hat{y}_{v,u} = \sigma\Big(\theta_8 \mathsf{ReLU}\big(\theta_9 (h \| h_v^L \| h_u^L) \big) \Big) \qquad \forall (v,u) \in \mathcal{A}. \tag{22}$$

A comprehensive representation of network $\mathcal{G}$ is generated by calculating the mean value of each vertex representation in (21). The prediction is subsequently derived by concatenating this mean value with the representations of the two vertices connected by the arc. This concatenated representation is then processed in (22) through a non-linear function that involves two additional tensors of weights θ_8 and θ_9. Note that the final sigmoid function is employed to ensure that the prediction falls within the range $[0,1]$. For more information about these equations, we refer the reader to Joshi et al. (2022). Our architecture consists of $L = 25$ layers, each of dimension $p = 128$.

5.2.3 Training algorithm

To find good values for the tensors $\theta_1, \ldots, \theta_9$, the model is trained using gradient back-propagation with a weighted binary cross-entropy loss. The loss is computed only using the selectable arcs and has a weight of 1 for arcs that are part of the MDEVSP solution, and of 0.1 for the others. Our dataset of solved instances is divided into three sets: 70 % are used for the training set, 15 % for the validation set, and 15 % for the test set. The training is carried out for a maximum of 200 epochs (roughly between 6 and 16 hours depending on the training dataset, see Section 6) using the Adam optimizer (Kingma and Ba, 2014) with a batch size of 5. We used the default values for the optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and no weight decay), except for the learning rate which was set to 10^{-4} instead of 10^{-3} to better stabilize the training. At the end of each epoch, the loss function is evaluated on the validation set. If this value has not decreased for the last 5 epochs, the learning is stopped. The numerical values have been set either using the recommended default values, or have been calibrated manually based on preliminary tests. The implementation is done in Python using PyTorch (Paszke et al., 2017) and DGL (Wang and Tang, 2021) libraries.

5.2.4 Selection algorithm

We recall that the GNN is a function $\Phi : \mathcal{G} \to [0,1]^k$ giving, for each of the k arcs of a network, an estimate of the probability that it is part of the solution obtained by the CG heuristic. Once the GNN is trained, we can use it to select relevant arcs from a new network $\mathcal{G}$. Additionally, we introduce a parameter $n^A \in \mathbb{N}$ defining the number of arcs to keep for each vertex. Our learning-based selection approach that finds a subset $\hat{\mathcal{Q}}$ of the selectable arcs is formalized in Algorithm 2. Set $\hat{\mathcal{Q}}$ is initially empty (line 6). Then, for each trip vertex, we choose the n^A outgoing selectable arcs having the highest probability to be in the solution, based on the prediction of GNN Φ (line 8). A similar selection is carried out for the incoming selectable arcs (line 9). Finally, subset $\hat{\mathcal{Q}}$ is returned.

Algorithm 2: GNN learning approach for arc selection

1 ▷ **Pre:** $\mathcal{G}$ is the aggregated network.
2 ▷ **Pre:** $\mathcal{V}_{\text{trip}}$ is the set of trip vertices in $\mathcal{G}$.
3 ▷ **Pre:** Φ is the GNN, previously trained.
4 ▷ **Pre:** $n^A \in \mathbb{N}$ is the threshold for the selection.
5
6 $\hat{\mathcal{Q}} := \emptyset$
7 **forall** $v \in \mathcal{V}_{\text{trip}}$ **do**
8 $\quad O_v := \text{getHighestOutArcs}(v, \Phi, n^A)$
9 $\quad I_v := \text{getHighestInArcs}(v, \Phi, n^A)$
10 $\quad \hat{\mathcal{Q}} := \hat{\mathcal{Q}} \cup O_v \cup I_v$
11 **return** $\hat{\mathcal{Q}}$

5.3 Selection with a hybrid heuristic

Both the greedy heuristic and the learning-based approach exhibit certain limitations. The former focuses on incorporating arcs with low immediate costs, potentially overlooking other important arcs. On the other hand, the latter, due to the inherent uncertainty of learning methods, may miss straightforward, yet significant arcs. These limitations become apparent in our experimental results. In response to this challenge, we introduce a third selection approach that simply combines both methods. Let $\hat{\mathcal{Q}}_{\text{GRD}}$ and $\hat{\mathcal{Q}}_{\text{GNN}}$ represent the sets of arcs selected by Algorithms 1 and 2, respectively. The hybrid heuristic returns $\hat{\mathcal{Q}} = \hat{\mathcal{Q}}_{\text{GRD}} \cup \hat{\mathcal{Q}}_{\text{GNN}}$. By doing so, we can benefit from the non-trivial selections made by the GNN while maintaining the guarantee that we do not overlook straightforward, yet significant arcs.

We also implemented another hybrid heuristic that applies the greedy heuristic but using the probabilities provided by the GNN instead of the arc costs. This approach was clearly outperformed by the above hybrid heuristic and, for this reason, will not be discussed in this paper.

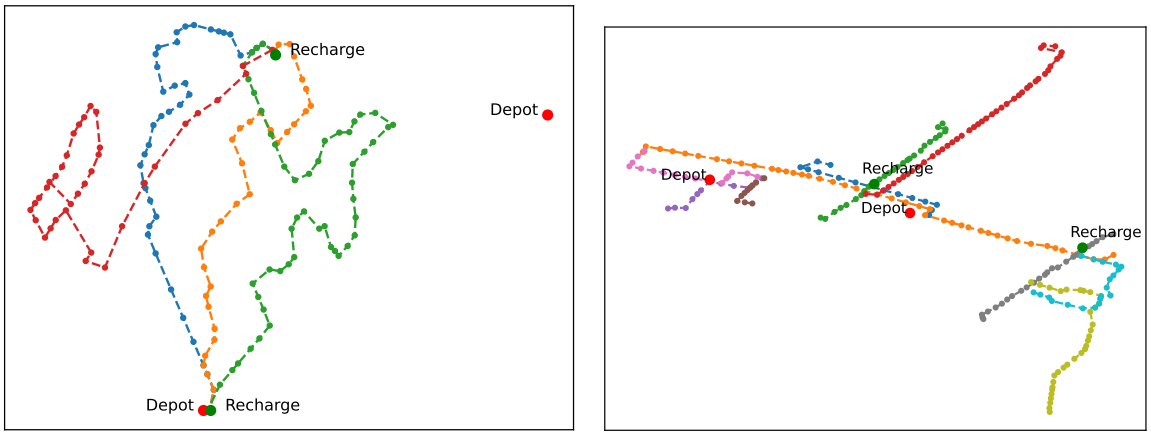
6 Computational experiments

This section presents the results of our computational experiments. All tests were run on a Linux machine equipped with an 16-core Intel i7-10700 processor with 64 GB and clocked at 2.9 GHz. A single thread was used for each test. The CG heuristic was implemented using the Gencol library (Desrosiers, 2010). The RMPs were solved using IBM Cplex version 20.1.

Below, we first describe the instances used for our experiments (Section 6.1). Then, we list the various CG heuristics used for our tests (Section 6.2). Finally, we report computational results to compare these heuristics and to assess the performance of the best ones on instances with varying characteristics and sizes (Section 6.3).

6.1 Test instances

To test the proposed algorithms, we use instances that were generated as in Brasseur (2022) from real-life timetables of the Montreal public transit agency. Two subsets of lines have been chosen to define two networks and create seven subsets of instances. Network 1 contains four lines, traveled in both directions, that are represented by different colors in Figure 3a. Network 2 has ten lines, also serviced in both directions (see Figure 3b). For each network, we define two depots and two charging stations. Some subsets of instances based on Network 1 have a reduced number of depots and/or a reduced number of recharging stations.



(a) Network 1 **(b) Network 2**

Figure 3: Networks used for our tests

To get various instances of an approximate target size for a given network, the original timetable is randomly modified such that a constant headway is imposed between the trips of each line in each hour, and the relative trip frequencies between one line and the others remain similar (see Brasseur, 2022). This method allows to have many instances with different timetables for the same network and a realistic distribution of the trips over time.

For all instances, we use the parameter values provided in Table 1. Note that the battery capacity $\overline{\sigma}$ of 100 KWh is consistent with the values given in Dirks et al. (2022). In addition, the recharging function f (see Figure 1) has three pieces defined by a recharging rate (in Wh per minute) of 1539.7

between 0 and 55 minutes of recharging time, of 716.0 between 55 and 69 minutes, and of 240.5 between 69 and 91 minutes.

Table 1: Parameter values

Parameter	Value
$\underline{\sigma}$	0 kWh
$\overline{\sigma}$	100 kWh
δ	15 minutes
β	45 minutes
γ	30 minutes
c^F	1000 per bus
c^W	2 per minute
c^R	30 per depot return
c^H	30 per recharge

The characteristics of the different instance subsets are displayed in Table 2. In order, the columns indicate: the subset identifier, the associated network, the number of depots, the number of charging stations, the average, minimum, and maximum numbers of trip per instance, the possibility to return to the depot for waiting during the day, the number of vehicles available per depot, the number of chargers available per station, and the number of instances in the subset. Note that subsets `A` to `D` share the same 500 trip timetables, but differ by the number of depots, the number of stations, and the possibility to return to the depot in the middle of the day.

Table 2: EVSP and MDEVSP instances

Instance Subset	Network	$\|D\|$	$\|H\|$	No. Trips			Return	v^d	u^h	No. Instances
				Avg	Min	Max				
`A`	1	1	1	687	568	893	Yes	60	4	500
`B`	1	1	2	687	568	893	Yes	60	2	500
`C`	1	2	2	687	568	893	Yes	30	2	500
`D`	1	1	1	687	568	893	No	60	4	500
`E`	2	2	2	787	669	880	Yes	50	2	500
`F`	2	2	2	1336	1152	1474	Yes	70	3	200
`G`	2	2	2	2600	2374	3115	Yes	200	5	50

6.2 Benchmarked column generation heuristics

For our computational tests, we compare the following CG algorithms.

`cgBasic`: The CG heuristic described in Section 4.

`cgUtopic`: A CG heuristic that keeps only the selectable arcs used in the solution found by the `cgBasic` heuristic. This heuristic is considered to estimate the maximum speedup that can be achieved.

`cgGRD`(n^R, n^S): Same as `cgBasic` except that a subset of the selectable arcs are removed. The ones kept are selected using the greedy selection procedure (Algorithm 1) and the parameters n^R and n^S defined in Section 5.1.1. Note that `cgGRD`$(1, 1)$ applies the pure greedy heuristic, i.e., without randomness, to select the removed arcs.

`cgGNN`$(\texttt{I}, n^A)$: Same as `cgGRD`(n^R, n^S) except that the selectable arcs kept are chosen using the GNN procedure (Algorithm 2) and the parameter n^A defined in Section 5.2.4. Set `I` indicates the training subset used.

`cgRND`(n^A): Same as `cgGNN`$(\texttt{I}, n^A)$ except that the selectable arcs kept are not chosen using a GNN model but at random.

`cgHBD`$(\texttt{I}, n^A, n^R, n^S)$: A hybrid of `cgGRD`$(n^R, n^S)$ and `cgGNN`$(\texttt{I}, n^A)$ that chooses the selectable arcs to keep as described in Section 5.3.

6.3 Computational results

In this section, we report average computational results obtained from three series of experiments. The first series allows to compare the different arc selection procedures on a subset of instances. The second aims at assessing the robustness of the best heuristic by testing it on more instances with different characteristics. Finally, the third set of experiments focuses on the largest instances. Detailed results are available in an online supplement.

To evaluate the performance of the network reduction procedures, we use the following two metrics. Let z_H and t_H be the cost of the solution obtained by a heuristic H and its total computational time. Then, we compute the time reduction and cost difference of heuristic H with respect to the `cgBasic` heuristic (B) as $\frac{(t_B - t_H)}{t_B}$ and $\frac{(z_H - z_B)}{z_B}$, respectively.

6.3.1 Selection procedure comparison

The first tests involve all CG heuristics listed in Section 6.2 (some with different parameter values) and is performed only on the instance subset `A` containing 75 test instances and involving a single depot and a single recharging station. The results obtained are reported in Table 3. Each row corresponds to a heuristic and provides its average results, namely, its total computational time in seconds (Time); the time reduction in percentage (Time Red.); the cost difference in percentage (Cost Diff.); the number of vehicles used in the solution (No. Veh.); and the number of arcs in the networks (No. Arcs).

First, we observe from the `cgUtopic` heuristic results that there is a huge potential to reduce the computational time by removing arcs from the subproblem network as an average time reduction of more than 99% can be achieved without deteriorating solution quality. The next four lines in Table 3 compare four variants of the `cgGRD`(n^R, n^S) heuristic. Variant `cgGRD`$(1,1)$ indicates that a pure greedy arc selection procedure provides a huge average time reduction (97.2%) but a poor-quality solution (average cost increase of 27.5%). This may be due to the fact that the greedy solution obtained by Algorithm 1 might not be feasible. The random variants `cgGRD`$(3, n^S)$, $n^S = 1, 3, 5$, yield, on average, improved solutions as n^S increases, but at the expense of more selected arcs and larger computational times.

Table 3: Comparative average results for instance subset `A`

Heuristic	Time (s)	Time Red. (%)	Cost Diff. (%)	No. Veh.	No. Arcs
`cgBasic`	454.2	-	-	41.9	20695
`cgUtopic`	3.7	99.1	0.0	41.9	2513
`cgGRD`$(1,1)$	11.6	97.2	27.5	45.1	2562
`cgGRD`$(3,1)$	11.4	97.3	29.9	45.3	2555
`cgGRD`$(3,3)$	83.8	80.9	8.2	42.9	3381
`cgGRD`$(3,5)$	128.3	70.8	4.3	42.3	3862
`cgGNN`$(\mathtt{A},1)$	9.9	97.7	161.3	124.8	2819
`cgGNN`$(\mathtt{A},2)$	85.5	80.1	12.4	48.6	3621
`cgGNN`$(\mathtt{A},3)$	187.1	57.2	1.6	42.6	4379
`cgRND`(2)	47.9	88.6	111.4	75.1	3776
`cgHBD`$(\mathtt{A},1,3,3)$	112.7	74.2	2.3	42.2	3769
`cgHBD`$(\mathtt{A},2,1,1)$	126.3	70.8	1.4	42.2	3850

Then, the `cgGNN`$(\mathtt{A}, n^A)$ variants with $n^A = 1, 2, 3$ also exhibit the trade-off between average reduced computational time and average cost difference. Again, as the number of selected arcs increases with the value of n^A, both the time and the solution quality increase to reach an average cost difference of only 1.6% for an average time reduction of near 60% with `cgGNN`$(\mathtt{A}, 3)$. On the other hand, we observe that, for a similar time reduction, `cgGRD`$(3,3)$ yields, on average, better solutions than `cgGNN`$(\mathtt{A}, 2)$ (cost difference of 8.3% vs 12.4%). This is due to the fact that it is difficult to have a global view of

the problem when making predictions with a machine learning model. Nevertheless, comparing the results of cgRND(2) and cgGNN(A, 2) that select approximately the same number of arcs indicate that the GNN model has learned from the training datasets and is making relatively good predictions.

Finally, the last two rows disclose the results of two cgHBD(A, n^A, n^R, n^S) variants. Compared to the first with $n^A = 1$ and $n^R = n^S = 3$, the second variant with $n^A = 2$ and $n^R = n^S = 1$ selects more arcs with the GNN procedure but less with the greedy one. On average, both variants select approximately the same number of arcs (3769 and 3850) and yield similar computational times (reductions of 74.2% and 70.8%). However, cgHBD(A, 2, 1, 1) computes much better quality solutions with an average cost difference of only 1.4%. This is similar to cgGNN(A, 3) which required larger computational times, and much better than cgGRD(3, 5) that required similar times. Consequently, cgHBD(A, 2, 1, 1) seems to offer the best compromise as it outperforms the standalone heuristics. This is explained by the fact that the GNN predictions identify interesting arcs without ensuring the global consistency of the schedules, while the greedy algorithm identifies arcs that are interesting for the construction of entire schedules.

A final word about the numbers of vehicles reported in Table 3. They are directly proportional to solution quality because they incur a large fixed cost per bus ($c^F = 1000$). Furthermore, given that there are 686 trips on average in the A instances and the optimal average number of vehicles is close to 42, we deduce that each used vehicle covers around 16 trips on average yielding high degeneracy and justifying the need to develop an efficient CG heuristic.

6.3.2 Robustness assessment

The second test series aims at evaluating the robustness of cgHBD(I, 2, 1, 1) when applied to other instance subsets I $\in \{$B, . . . , F$\}$. Table 4 reports the average results obtained by this heuristic, by the methods that compose it (cgGRD(1, 1) and cgGNN(I, 2)), and by cgBasic. First, we observe a slight deterioration of the performance of cgHBD(I, 2, 1, 1) on the subsets I $\in \{$B, C, D$\}$ (average cost difference varying between 1.6% and 2.1%) compared to cgHBD(A, 2, 1, 1) on subset A (1.4%). Because such a deterioration is also observed for cgGNN(I, 2), it can be explained by less accurate predictions made by the GNN model when there are more arcs to choose from (due to an increase of the number of charging stations or depots in subsets B and C) or returning to the depot is forbidden (subset D). For subsets B and C, this counterperformance is mitigated by an improvement of the greedy selection procedure as shown by the average cost differences yielded by cgGRD(1, 1). For all those subsets I, cgHBD(I, 2, 1, 1) clearly outperforms the methods that compose it when applied separately and still yields interesting average speedups compared to cgBasic.

For the subsets I $\in \{$E, F$\}$ that involve more bus trips, more bus lines, and two depots, the story is somewhat different. Indeed, cgHBD(I, 2, 1, 1) still outperforms cgGRD(1, 1) and cgGNN(I, 2) but by a much smaller margin for cgGNN(I, 2). It seems that the GNN predictions are much better for these instances which allow to cover much more trips per vehicle (close to 30 on average compared to 16 for subset A). The average time reductions decrease for these larger instances but remain substantial (close to 60%).

Consequently, all these results show that the proposed hybrid CG heuristic is robust to some variations of the instance characteristics.

6.3.3 Results on larger instances

With our last experiments, we test the hybrid CG heuristic on the very large instances in subset G, which involve 2 depots, 2 charging stations, and an average of 2600 trips. As reported in the first row of Table 5, the cgBasic heuristic took 16266 seconds to solve these instances on average, providing solutions with an average of 30.8 trips per vehicle. The next three rows present the results obtained by cgGRD(1, 1), cgGNN(G, 2), and cgHBD(G, 2, 1, 1), which indicate a decrease in solution quality compared to the previous datasets. In particular, cgHBD(G, 2, 1, 1) exhibits its worst performance with an average cost difference of 5.1% but still preserves a substantial time reduction of 54.5%. These results are not

Table 4: Average results for the instance subsets A to F

Instance Subset	Heuristic	Time (s)	Time Red. (%)	Cost Diff. (%)	No. Veh.	No. Arcs
A	cgBasic	454.2	-	-	41.9	20695
A	cgGRD(1, 1)	11.6	97.2	27.5	45.1	2562
A	cgGNN(A, 2)	85.5	80.1	12.4	48.6	3621
A	cgHBD(A, 2, 1, 1)	126.3	70.8	1.4	42.2	3850
B	cgBasic	463.1	-	-	42.0	22224
B	cgGRD(1, 1)	12.4	97.1	18.6	43.5	2901
B	cgGNN(B, 2)	76.2	82.5	16.2	50.1	3909
B	cgHBD(B, 2, 1, 1)	122.6	72.0	2.1	42.4	4137
C	cgBasic	505.4	-	-	41.7	25609
C	cgGRD(1, 1)	12.0	97.5	18.7	42.7	4380
C	cgGNN(C, 2)	75.6	84.8	15.0	48.8	5436
C	cgHBD(C, 2, 1, 1)	126.5	74.4	2.1	42.1	5643
D	cgBasic	1479.7	-	-	41.9	19726
D	cgGRD(1, 1)	10.9	99.0	26.3	45.1	2479
D	cgGNN(D, 2)	56.9	94.8	21.2	57.2	3601
D	cgHBD(D, 2, 1, 1)	149.2	87.5	1.6	42.3	3866
E	cgBasic	564.4	-	-	29.1	23904
E	cgGRD(1, 1)	27.5	95.1	57.7	41.8	4962
E	cgGNN(E, 2)	186.8	66.7	5.4	31.3	6106
E	cgHBD(E, 2, 1, 1)	232.5	58.6	3.7	30.3	6371
F	cgBasic	2571.9	-	-	45.3	55709
F	cgGRD(1, 1)	85.0	96.7	63.9	66.9	7765
F	cgGNN(F, 2)	809.6	68.4	7.0	50.3	9633
F	cgHBD(F, 2, 1, 1)	1019.3	58.9	3.3	46.9	10021

surprising given the size of those instances and the smaller number of training instances used (only 35 compared to 350 for subsets A to E). To overcome this difficulty, we tried two strategies. The first is to train the GNN model on instance subset E defined also on Network 2 and containing much more training instances. This approach gives unsatisfactory results with an average cost difference exceeding 40%, excluding one instance for which the algorithm did not even return a feasible solution. The second idea is to apply transfer learning (denoted E+G) that consists in training first the GNN model on subset E before fine tuning it on subset G. Compared to cgHBD(G, 2, 1, 1), variant cgHBD(E+G, 2, 1, 1) improves solution quality (cost difference of 4.2%) while yielding a similar time reduction of 52.5%. Thus, the proposed hybrid method still works on large instances, thanks to transfer learning, but the gain in computational time is less important.

Table 5: Results for the subset G of large instances

Heuristic	Time (s)	Time Red. (%)	Cost Diff. (%)	No. Veh.	No. Arcs
cgBasic	16266.3	-	-	84.5	167520
cgGRD(1, 1)	716.8	95.5	66.0	126.6	13775
cgGNN(G, 2)	5111.9	67.8	21.6	114.5	17078
cgHBD(G, 2, 1, 1)	7261.4	54.5	5.1	88.3	17864
cgHBD(E, 2, 1, 1)	4082.5	74.3	40.6	108.0	23846
cgHBD(E+G, 2, 1, 1)	7584.8	52.5	4.2	87.1	17948

7 Conclusion

In this paper, we have developed different techniques to reduce the size of the subproblem networks and accelerate a CG heuristic for solving the MDEVSP. These procedures identify network arcs to keep from a set of potential arcs to remove. Three types of procedures are explored: the first is based on a

greedy heuristic that quickly computes (possibly infeasible) solutions; the second relies on a GNN to estimate the probability that an arc is part of an optimal solution; and the third combines the previous two.

Our computational results showed that the hybrid technique yields the best results. For medium-sized instances (with less than 1500 trips), this method reduces the computational time by an average of at least 58% while limiting the cost increase to an average of up to 3.7%. For larger instances with up to 3000 trips, transfer learning helps to obtain solutions with an average cost increase of 4.2% while cutting the computational by an average factor of 2. Overall, our results show that combining a greedy heuristic that can identify good sequences of arcs and a GNN model that can make good predictions produce the best arc selection method.

As future work, we envision the development of an improved GNN model or a better local search heuristic that would allow to select less arcs and obtain better solutions. Another interesting research avenue is to try to limit the recharging possibilities by predicting when and where the recharges should occur and their duration. This information could be used to further reduced the subproblems' solution space and speedup the CG heuristic.

A Detailed features

Table 6 presents the features $\phi_v^d \in \mathbb{R}^{9+2n^L}$ and $\psi_a^d \in \mathbb{R}^8$ attached to the vertices and the arcs of a network $\mathcal{G}^d$ that are considered by the proposed GNN model in Section 5.2.

Table 6: Detailed features used to represent a network $\mathcal{G}^d$

Features attached to a vertex $v \in \mathcal{N}^d$		
Feature	Domain	Description
$\phi_{v,1}^d$	One-hot	Type of vertex v (trip, depot, charging station, etc.)
$\phi_{v,2}^d$	Integer	Starting time of the activity associated with vertex v
$\phi_{v,3}^d$	Integer	Ending time of the activity associated with vertex v
$\phi_{v,4}^d$	Integer	Duration of the activity associated with vertex v
$\phi_{v,5}^d$	Integer	For a trip vertex v, number of trips leaving l_v^D within 10 minutes after v
$\phi_{v,6}^d$	Integer	Same as $\phi_{v,5}^d$ but considering only the trips that also end at l_v^E
$\phi_{v,7}^d$	Integer	For a trip vertex v, number of trips arriving at l_v^E within 10 minutes before v
$\phi_{v,8}^d$	Integer	Same as $\phi_{v,7}^d$ but considering only the trips that also start from l_v^S
$\phi_{v,9}^d$	Integer	Number of trips in the instance (same value for each vertex v)
$\phi_{v,8+2k}^d, k = 1, \ldots, n^L$	Binary	Equal to 1 if k is the starting location of the activity associated with vertex v
$\phi_{v,9+2k}^d, k = 1, \ldots, n^L$	Binary	Equal to 1 if k is the ending location of the activity associated with vertex v
Features attached to an arc $a = (v, u) \in \mathcal{A}^d$		
Feature	Domain	Description
$\psi_{a,1}^d$	Real	Cost of arc a
$\psi_{a,2}^d$	Real	Energy consumption on arc a (0 for charging arcs)
$\psi_{a,3}^d$	Integer	Deadhead travel time on arc a
$\psi_{a,4}^d$	Integer	Waiting time on arc a
$\psi_{a,5}^d$	Integer	Total elapsed time excluding trip time on arc a
$\psi_{a,6}^d$	Integer	For a trip-to-trip arc (v, u), departure time ranking of arc (v, u) among all trip-to-trip arcs (v, u')
$\psi_{a,7}^d$	Integer	Same as $\psi_{a,6}^d$ but considering only the arcs (v, u') with $l_{u'}^D = l_u^D$
$\psi_{a,8}^d$	Integer	Cost ranking of arc (v, u) among all arcs (v, u')

References

Adler, J.D., Mirchandani, P.B., 2017. The vehicle scheduling problem for fleets with alternative-fuel vehicles. Transportation Science 51, 441–456.

Alvo, M., Angulo, G., Klapp, M.A., 2021. An exact solution approach for an electric bus dispatch problem. Transportation Research Part E: Logistics and Transportation Review 156, 102528.

Bengio, Y., Lodi, A., Prouvost, A., 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research 290, 405–421.

Bertossi, A.A., Carraresi, P., Gallo, G., 1987. On some matching problems arising in vehicle scheduling models. Networks 17, 271–281.

Brasseur, J., 2022. Accélération d'une méthode d'agrégation dynamique de contraintes par apprentissage automatique pour le problème de construction d'horaires de conducteurs d'autobus. Master's thesis. Ecole Polytechnique, Montréal (Canada).

Cappart, Q., Bergman, D., Rousseau, L.M., Prémont-Schwarz, I., Parjadis, A., 2022. Improving variable orderings of approximate decision diagrams using reinforcement learning. INFORMS Journal on Computing 34, 2552–2570.

Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Veličković, P., 2023. Combinatorial optimization and reasoning with graph neural networks. Journal of Machine Learning Research 24, 1–61.

Cappart, Q., Moisan, T., Rousseau, L.M., Prémont-Schwarz, I., Cire, A.A., 2021. Combining reinforcement learning and constraint programming for combinatorial optimization, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 3677–3687.

Chao, Z., Xiaohong, C., 2013. Optimizing battery electric bus transit vehicle scheduling with battery exchanging: Model and case study. Procedia-Social and Behavioral Sciences 96, 2725–2736.

Dauer, A.T., Prata, B.d.A., 2021. Variable fixing heuristics for solving multiple depot vehicle scheduling problem with heterogeneous fleet and time windows. Optimization Letters 15, 153–170.

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

Desaulniers, G., Hickman, M.D., 2007. Public transit, in: Barnhart, C., Laporte, G. (Eds.), Handbooks in Operations Research and Management Science. Elsevier, Amsterdam. volume 14, pp. 69–127.

Desfontaines, L., Desaulniers, G., 2018. Multiple depot vehicle scheduling with controlled trip shifting. Transportation Research Part B: Methodological 113, 34–53.

Desrosiers, J., 2010. GENCOL : Une équipe et un logiciel d'optimisation, in: Bui, A., Tseveendorj, I. (Eds.), Combinatorial Optimization in Practice. Hermann, Paris. volume 8, pp. 61–96.

Dirks, N., Schiffer, M., Walther, G., 2022. On the integration of battery electric buses into urban bus networks. Transportation Research Part C: Emerging Technologies 139, 103628.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A., 2019. Exact combinatorial optimization with graph convolutional neural networks. Advances in neural information processing systems 32.

Gkiotsalitis, K., Iliopoulou, C., Kepaptsoglou, K., 2023. An exact approach for the multi-depot electric bus scheduling problem with time windows. European Journal of Operational Research 306, 189–202.

Glorot, X., Bordes, A., Bengio, Y., 2011. Deep sparse rectifier neural networks, in: Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp. 315–323.

Häll, C.H., Ceder, A., Ekström, J., Quttineh, N.H., 2019. Adjustments of public transit operations planning process for the use of electric buses. Journal of Intelligent Transportation Systems 23, 216–230.

Hamilton, W., Ying, Z., Leskovec, J., 2017. Inductive representation learning on large graphs. Advances in neural information processing systems 30.

Ibarra-Rojas, O., Delgado, F., Giesen, R., Muñoz, J., 2015. Planning, operation, and control of bus transport systems: A literature review. Transportation Research Part B 77, 38–75.

Irnich, S., Desaulniers, G., 2005. Shortest path problems with resource contraints, in: Desaulniers, G., Desrosiers, J., Solomon, M.M. (Eds.), Column generation. Springer Science & Business Media. volume 5, pp. 33–65.

Jiang, M., Zhang, Y., Zhang, Y., 2022a. A branch-and-price algorithm for large-scale multidepot electric bus scheduling. IEEE Transactions on Intelligent Transportation Systems. In Press .

Jiang, M., Zhang, Y., Zhang, Y., 2022b. Multi-depot electric bus scheduling considering operational constraint and partial charging: A case study in Shenzhen, China. Sustainability 14, 255.

Joshi, C.K., Cappart, Q., Rousseau, L.M., Laurent, T., 2022. Learning the travelling salesperson problem requires rethinking generalization. Constraints 27, 70–98.

Jovanovic, R., Bayram, I.S., Bayhan, S., Voß, S., 2021. A GRASP approach for solving large-scale electric bus scheduling problems. Energies 14, 6610.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 0.

Kipf, T.N., Welling, M., 2017. Semi-supervised classification with graph convolutional networks, in: International Conference on Learning Representations.

van Kooten Niekerk, M.E., van den Akker, J., Hoogeveen, J., 2017. Scheduling electric vehicles. Public Transport 9, 155–176.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. nature 521, 436–444.

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

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

Morabit, M., Desaulniers, G., Lodi, A., 2021. Machine-learning–based column selection for column generation. Transportation Science 55, 815–831.

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

Olsen, N., Kliewer, N., Wolbeck, L., 2022. A study on flow decomposition methods for scheduling of electric buses in public transport based on aggregated time–space network models. Central European Journal of Operations Research 30, 883—919.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A., 2017. Automatic differentiation in PyTorch.

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.

Reuer, J., Kliewer, N., Wolbeck, L., 2015. The electric vehicle scheduling problem: A study on time-space network based and heuristic solution, in: Proceedings of the Conference on Advanced Systems in Public Transport (CASPT), pp. 1–15.

Ruiz, L., Gama, F., Ribeiro, A., 2020. Gated graph recurrent neural networks. IEEE Transactions on Signal Processing 68, 6303–6318.

Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I., Uchoa, E., 2019. Primal heuristics for branch and price: The assets of diving methods. INFORMS Journal on Computing 31, 251–267.

Sassi, O., Oulamara, A., 2017. Electric vehicle scheduling and optimal charging problem: complexity, exact and heuristic approaches. International Journal of Production Research 55, 519–535.

Selsam, D., Bjørner, N., 2019. Guiding high-performance sat solvers with unsat-core predictions, in: Theory and Applications of Satisfiability Testing, Springer. pp. 336–353.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y., 2018. Graph attention networks, in: International Conference on Learning Representations.

Wang, C., Guo, C., Zuo, X., 2021. Solving multi-depot electric vehicle scheduling problem by column generation and genetic algorithm. Applied Soft Computing 112, 107774.

Wang, Q., Tang, C., 2021. Deep reinforcement learning for transportation network combinatorial optimization: A survey. Knowledge-Based Systems 233, 107526.

Wen, M., Linde, E., Ropke, S., Mirchandani, P., Larsen, A., 2016. An adaptive large neighborhood search heuristic for the electric vehicle scheduling problem. Computers & Operations Research 76, 73–83.

Yıldırım, Ş., Yıldız, B., 2021. Electric bus fleet composition and scheduling. Transportation Research Part C: Emerging Technologies 129, 103197.

Zhang, A., Li, T., Tu, R., Dong, C., Chen, H., Gao, J., Liu, Y., 2021a. The effect of nonlinear charging function and line change constraints on electric bus scheduling. Promet-Traffic&Transportation 33, 527–538.

Zhang, A., Li, T., Zheng, Y., Li, X., Abdullah, M.G., Dong, C., 2022. Mixed electric bus fleet scheduling problem with partial mixed-route and partial recharging. International Journal of Sustainable Transportation 16, 73–83.

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