Proceedings of the of the 17th Conference on Computer Science and Intelligence Systems pp. 313–322 DOI: 10.15439/2022F186 ISSN 2300-5963 ACSIS, Vol. 30

The electric vehicle shortest path problem with time windows and prize collection

Antonio Cassia, Ola Jabali, Federico Malucelli
Politecnico di Milano, Italy
Email: {antonio.cassia, ola.jabali, federico.malucelli}@polimi.it

Marta Pascoal
University of Coimbra, CMUC, INESC-Coimbra, Portugal,
Politecnico di Milano, Italy
Email: marta.brazpascoal@polimi.it

***Abstract*—The Electric Vehicle Shortest Path Problem (EVSPP) aims at finding the shortest path for an electric vehicle (EV) from a given origin to a given destination. During long trips, the limited autonomy of the EV may imply several stops for recharging its battery. We consider combining such stops with visiting points of interest near charging stations (CSs). Specifically, we address a version of the EVSPP in which the charging decisions are harmonized with the driver's preferences. The goal is to maximize the total gained score (assigned by the driver to the CSs), while respecting the time windows and the EV autonomy constraints. We define the problem as a MILP and develop an A* search heuristic to solve it. We evaluate the method by means of extensive computational experiments on realistic instances.**

I. Introduction

DEVISING tools that facilitate the use of Electric Vehicles (EVs) is key to successfully electrifying transport, e.g., [5]. The limited autonomy of EVs coupled with the scarcity of charging stations (CSs) entails that long trips may involve several charging stops. Given that such stops are likely to be time consuming, we explore the idea of matching the charging stops with user preferences.

Considering an EV that needs to travel between an origin and a destination, we consider that the user attributes a score to each CS. Such scores are based on the vicinity to Points of Interest (POIs) to the CS. For example, a user may prefer to stop at cultural venues. In this case, a high score is given to CSs nearby such venues. Using those scores, we consider the problem of optimizing the shortest path respecting all energy feasibility constraints, while maximizing the total score that the user can achieve. To handle this problem we propose the Maximum Profit Model (**MPM**). To avoid excessively long trips, we limit the duration of the route. We establish this limit by solving the Shortest Path Model (**SPM**), which is the shortest EV path in time. We allow the **MPM** to deviated from the shortest path length within a given tolerance.

We adapt the Mixed Integer Linear programming (MILP) model proposed in [6] to handle the **MPM** and the **SPM**. Furthermore, we develop a heuristic algorithm based on the A* search which handles large instances of **MPM**. To this end, we propose the Maximum Discounted Profit Model (**MDPM**), which discounts the scores of the nodes into the arc costs that connect them. By doing so, we are able to efficiently adapt the A* search Algorithm to the **MPM**.

This work was partially supported by the Portuguese Foundation for Science and Technology (FCT) under project grants UID/MAT/00324/2020 and UID/MULTI/00308/2020.

In general, the Electric Vehicle Shortest Path Problem (EVSPP) is a shortest path problem that accounts for battery limitation and charging constraints. Due to their limited autonomy, EVs may need to detour to CSs in order to recharge their battery. This is particularly true in medium and long range routes, like in [11]. A key decision in this context is where and how much to charge the EVs. The problem of minimizing the overall trip time for EVs in road networks was studied by [1]. A heuristic algorithm for solving large EVSPP instances was proposed in [14]. Baum et al. [2] introduced a functional representation of the optimal energy consumption between two locations, which led to developing an efficient heuristic algorithm that computes energy optimal paths.

One of the main EVSPP modeling assumption relate to how the EV batteries are recharged. Some researchers assume that the EV must completely recharge before leaving a CS, e.g. [4], [9]. Other works (e.g., [6], [10], [12]) consider the charging quantity as a decision within the optimization. As in most studies we assume that the energy consumption is directly and exclusively related to the traveled distance.

In practice, the EV charging function is nonlinear with respect to time, and depends on the used charging technology. The nonlinear charging functions were modeled as piecewise linear concave functions by [6], [10].

Time window (TW) constraints have been introduced in EV problems by [12]. TWs oblige EVs to arrive in predetermined CSs before or during a particular time interval. Contrary to what is done in the literature, we assume that TW types are given (e.g., lunch breaks), yet their location is determined from a set of CSs which accommodate this type of TW. Furthermore, we consider required and weakly mandatory TWs.

Considering that CSs have different scores and a given maximum path length (in time), the aim of the **MPM** is to maximize the total score of visited CSs. As such, CSs that better match the user preferences are prioritized. We consider a single EV, with partial recharging decisions and nonlinear charging functions. Furthermore, we consider only public CSs, having different technologies and scores. We also consider TWs with a limit on the total travel time.

The contributions of this paper are threefold: 1) we propose an EV shortest path model that accounts for user preferences (**MPM**); 2) we develop a heuristic based on the A* algorithm to solve that problem; 3) and we verify the performance of both the MILP model and the heuristic algorithm on realistic test instances.

We introduce the **MPM** in Sec. II and develop an A* algorithm for it in Sec. III. We present our computational experiments in Sec. IV, and state our conclusions in Sec. V.

II. Problem definition

The **MPM** maximizes the score of the CSs visited by the EV, such that the resulting path is feasible and within a tolerance from the shortest EV path. In the literature, the goal of maximizing scores obtained by visiting nodes is often called prize collection [13]. To model the **MPM** we adapt the arc based MILP model of [6]. Due to space limitations, we do not present the full formulation. Instead, we give an overview of that work and then detail the major adaptations made to handle the **MPM**.

Froger et al. [6] introduced the Fixed Route Vehicle Charging Problem (FRVCP). Given a sequence of customer nodes (i.e., not CSs) to visit, the objective of the FRVCP is to determine the charging operations (which CSs to visit and how much to charge), in order to minimize the total route duration while satisfying the following conditions: The customers in the resulting route are visited according to the given order, the resulting route is energy feasible and satisfies a maximum duration limit $T^{\max}$. The state of charge (SoC) of the EV is tracked on each traversed arc and visited node. The EV may be partially recharged at a set of charging stations $\mathcal{S}$, which may have different technologies. For each technology we consider a nonlinear charging function, approximated via a piecewise linear function following the models by [10]. In the **MPM** the fixed sequence of nodes to visit goes from an origin $\mathcal{O}$ node to a destination $\mathcal{D}$ node.

Building on the FRVCP, we now describe the **MPM**. Let $\mathcal{G} := (\mathcal{S}_{\mathcal{O},\mathcal{D}}, \mathcal{A})$ be a directed graph, where $\mathcal{S}_{\mathcal{O},\mathcal{D}} := \mathcal{S} \cup \{\mathcal{O}, \mathcal{D}\}$ and $\mathcal{A}$ is the set of arcs that connect pairs of nodes in $\mathcal{S}_{\mathcal{O},\mathcal{D}}$. Let $t_{ij} \geq 0$ and $e_{ij} \geq 0$ be the driving time and energy consumption of arc $(i,j) \in \mathcal{A}$, both satisfying the triangular inequality. The EV departs from $\mathcal{O}$ with a fully charged battery of capacity Q. We impose that the SoC of the EV is at least $q_{\min}$ throughout the route. Since we consider long trip planning, we assume that the number of nights is a user input.

The user will spend some time in the neighborhood of the selected CS. Moreover, in certain moments of the day, the user may prefer to visit a CS near a POI, e.g., restaurants or hotels. CSs are typically strategically placed near those types of POIs. It is important that the user visits CSs that best suit her preferences. We attribute a score σ_j for $j \in \mathcal{S}$, which are input to the **MPM**. We assume that they can be derived based on the user's ranking of POIs. Furthermore, the user may personalize her trip by imposing TWs related to an activity (e.g., lunch breaks). The classical definition of TWs determines a time to visit a given node. In the **MPM**, such TWs may be realized

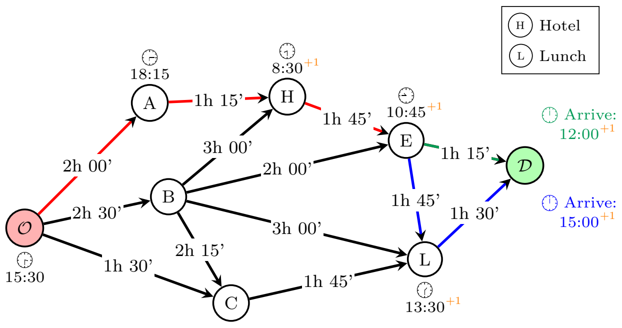

Figure 1: Path from $\mathcal{O}$ to $\mathcal{D}$ with a night at a hotel and lunch breaks. The timestamp near a node represents the corresponding departure time, including the recharging time (not reported in the figure). For $\mathcal{D}$, instead, the timestamps represent the arrival time. The number on each arc is its travel time. The red path is optimal, stopping in the hotel as required and reaching node E. If also the lunch break is required, then the EV is forced to arrive to $\mathcal{D}$ with stopping at L (blue path). Instead, with the weakly mandatory flag, from E the EV can go directly to $\mathcal{D}$ 3h in advance (green path).

at a number of CSs. We therefore denote the TWs related to our **MPM** as Activity-based Time Windows (ATWs). In our experiments in Sec. IV, we assign to each CS a randomly generated score between 0 and 5. In practice, these values will be established by the user in real applications.

Let $\mathcal{W}$ be the set of all ATWs, which is partitioned into two subsets, $\mathcal{W}^R, \mathcal{W}^M \subseteq \mathcal{W}$, formed by the required and the weakly mandatory ATWs. The EV is forced to stop in each $k_R \in \mathcal{W}^R$, but it must stop in $k_M \in \mathcal{W}^M$ only if there exists at least one k_R such that $k_M \prec k_R$. Otherwise, k_M may be skipped. We assume that the sets $\mathcal{W}^R, \mathcal{W}^M$ are ordered by chronological order of starting time of the TWs. Each ATW is defined as:

$$k := (\gamma_k^L, \gamma_k^U, t_k^{\min}, o_k, \nu_k), \quad k \in \mathcal{W},$$

where the interval $[\gamma_k^L, \gamma_k^U]$ (with $\gamma_k^L < \gamma_k^U$) describes its initial and ending times (the EV must arrive before γ_k^U); $t_k^{\min}$ is the minimum time that the EV needs to stop during k; o_k is a binary value, equal to 1 if $k \in \mathcal{W}^M$, and 0 otherwise. The set $\mathcal{W}$ is ordered, thus, if $k \prec h$, then $\gamma_k^L < \gamma_h^L$, for any $k, h \in \mathcal{W}$. The EV is allowed to arrive at a node with a maximum anticipation time $\widetilde{\varphi}$. In this case, the activity will nevertheless start at γ_k^L. Thus, if the EV arrives in node i in the interval $\left[\gamma_k^L - \widetilde{\varphi}, \gamma_k^U\right]$, then it may decide to stop at i for at least $t_k^{\min}$ or instead perform k in a subsequent node. In addition, we use hard TWs, which means that it is not possible to perform the ATW k before $\gamma_k^L - \widetilde{\varphi}$ nor after γ_k^U. Finally, two ATWs can overlap, but they cannot be contained in one another. Each CSs is associated with multiple POIs, and each TW requires a specific POI. This information is stored in the label ν_k and is different for each TW. For instance, if $k \in \mathcal{W}$ refers to the first night, then $\nu_k =$ "Hotels" and the EV is forced to stop at a CSs near a hotel. So, it is possible to construct the set of chargers $\mathcal{S}_k \subseteq \mathcal{S}$ that have in their neighborhood the POI stated in ν_k.

To determine $T^{\max}$ we solve an EV Shortest Path Problem **SPM**, subject to the same constraints as **MPM**, but with the goal of minimizing the total trip duration (i.e., ignoring the scores). Let T^{opt} be the optimal objective function value of **SPM**, which is a theoretical lower bound on $T^{\max}$. Then, let T^{add} be the tolerance of the total additional detouring time from the shortest path to stop in nodes with higher scores. Therefore, we set $T^{\max}$ to $T^{\text{opt}} + T^{\text{add}}$.

To avoid routes with many stops at SCs, we introduce the parameter $r^{\min}$, as $r^{\min} := \lfloor 0.4\,(Q - q_{\min})/\eta \rfloor$, where η is the average energy consumption per kilometer (expressed in kWh/km), and is dependent on the EV type. The ratio $(Q - q_{\min})/\eta$ represents the maximum autonomy of the vehicle, excluding the minimal amount of energy that is always required. With this arrangement it is possible to prune all the arcs associated with a distance less than $r^{\min}$, which also avoids stops for charging the EV in consecutive locations that are very close to each other.

Let ξ denote a lower bound on the distance of a trip from $\mathcal{O}$ to $\mathcal{D}$ (its computation is described in Sec. IV). Then the maximum number of legs N in a path is defined as $\lceil 1.5 \lceil \xi / r^{\min} \rceil \rceil$.

III. A* SEARCH ALGORITHM

In the following we propose a heuristic algorithm for the **SPM** and extend it for solving the **MPM**. Recall that the **SPM** solution serves as an input to **MDPM**. The algorithm is based on the A* search, which finds a path from an origin $\mathcal{O}$ to a destination $\mathcal{D}$ with the smallest cost. To do that, it maintains the tree of all the paths originated from $\mathcal{O}$ and extends each of them one arc at a time until $\mathcal{D}$ is reached. It uses a best-first search by selecting the node that minimizes

$$f(i) := g(i) + h(i),$$

where i is the current node, $g(i)$ is the cost of the path from $\mathcal{O}$ to i, and $h(i)$ is an estimate of the cost of the shortest path from i to $\mathcal{D}$. If $h(i)$ never overestimates the real cost to reach $\mathcal{D}$ from i, for all i, then the A* algorithm finds the optimal solution.

The A* algorithm is based on minimizing cost mechanisms, thus it is not directly applicable to prize collection settings such as the **MDPM**. Therefore, we propose the **MDPM** which approximates the **MPM**, searching for a shortest path while maximizing the total score. To do that, we assign a weight $\tilde{s}_{ij}$ with each arc, defined as

$$\tilde{s}_{ij} := t_{ij} + \Delta_j - \mu\sigma_j,\ \forall (i,j) \in \mathcal{A} \ \text{ s.t. } \ i,j \in \mathcal{S},$$

where Δ_j is the the time spent waiting while the EV is charging at the CS j, and μ is a parameter that indicates the relative importance of the score with respect to the time required to go from i to j, charging time included. The objective function of the **MDPM** model is then

$$\min \sum_{(i,j)\in\mathcal{A},\ j\in\mathcal{S}} \tilde{s}_{ij} x_{ij}.$$

This expression is nonlinear, but it can be linearized by introducing new decision variables s_{ij}, and changing the objective function as

$$\min \sum_{(i,j)\in\mathcal{A},\ j\in\mathcal{S}} [(t_{ij} - \mu\sigma_j)x_{ij} + s_{ij}]$$

We first discuss the computation of the potentials used to estimate the heuristic function h. We then incorporate the TWs in the heuristic. Finally, we modify the algorithm to account for the scores in h.

Let $\underline{q}_i$ and $\overline{q}_i$ be the SoC when the EV arrives and departs from CS i. The variables $\underline{c}_i$ and $\overline{c}_i$ are respectively the start and end time for charging an EV. Variables $\underline{\tau}_i, \overline{\tau}_i$ track the time when the EV arrives and leaves CS $i \in \mathcal{S}$, respectively. There is also a tolerance φ_i that represents how much time in advance, with respect to γ_k^L, the EV can arrive in i, for any $i \in \mathcal{S}_{\mathcal{D}}$. The maximum anticipation time is set to $\widetilde{\varphi}$, but even if the EV arrives in advance, the minimum stopping time $t_k^{\min}$ starts at γ_k^L and not before.

Let x_{ij} be a binary variable which equals 1 if the EV arrives at node j from node i, 0 otherwise. The variable y_{jk} is also binary and it is 1 if the EV stops in j in TW k, 0 otherwise. The maximum duration of the trip is $T^{\max}$ (in Sec. III we show how to compute an upper bound for this value). The variable z_k is binary and is equal to 1 if the EV arrives in $\mathcal{D}$ after TW k, 0 otherwise, for any $k \in \mathcal{W}^M$. It is used to link the arrival time in node $\mathcal{D}$ and avoidable TWs.

A. Potentials

We now find an initial estimate of the total time from any CS i to $\mathcal{D}$, building on techniques used in [14]. We start by dropping the charging and the TW constraints, thus we obtain a problem that can be solved using the Dijkstra's algorithm. Let $\mathcal{G} := \langle \mathcal{S}_{\mathcal{O},\mathcal{D}}, \mathcal{A} \rangle$ be the directed graph from $\mathcal{O}$ to $\mathcal{D}$, where $\mathcal{S}_{\mathcal{O},\mathcal{D}}$ is the set of nodes and $\mathcal{A} := \mathcal{S}_{\mathcal{O}} \times \mathcal{S}_{\mathcal{D}}$ the set of arcs. We then apply the backward Dijkstra's algorithm, which operates on the reversed graph $\mathcal{G}' := \langle \mathcal{S}_{\mathcal{O},\mathcal{D}}, \mathcal{A}' \rangle$ where

$$\mathcal{A}' := \mathcal{S}_{\mathcal{D}} \times \mathcal{S}_{\mathcal{O}} \text{ such that } (i,j) \in \mathcal{A} \Rightarrow (j,i) \in \mathcal{A}',$$

while considering $\mathcal{D}$ as the origin and $\mathcal{O}$ as the destination. This allows us to obtain a lower bound on the driving time from any $i \in \mathcal{S}_{\mathcal{O},\mathcal{D}}$ to $\mathcal{D}$, which we denote by $\boldsymbol{\pi}_{\text{dr}}(i)$. To account for energy consumption, we apply the backward Dijkstra's algorithm considering the energy consumption as the weight for the arcs. This allows us to derive the minimum amount of energy required from $i \in \mathcal{S}_{\mathcal{O},\mathcal{D}}$ to $\mathcal{D}$. We denote this lower bound by $\boldsymbol{\pi}_{\text{cons}}(i)$.

The EV arrives partially charged at each node, with an amount of energy equal to SoC(i). Since the minimal amount of SoC $q_{\min}$ needs to be respected at every node, we can think of SoC$(i) - q_{\min}$ as the available energy at node $i \in \mathcal{S}_{\mathcal{O},\mathcal{D}}$. Then, to compute the minimal amount of energy from i to $\mathcal{D}$, we define

$$\tilde{\boldsymbol{\pi}}_{\text{cons}}(i) := \boldsymbol{\pi}_{\text{cons}}(i) - (\text{SoC}(i) - q_{\min}).$$

We now compute a lower bound for the charging time, based on the minimal required energy at a node $i \in \mathcal{S}_{\mathcal{O},\mathcal{D}}$. We define $\mathcal{G}_i := \langle \mathcal{T}_i, \mathcal{A}_i \rangle$ as a subgraph of $\mathcal{G}$, where $\mathcal{T}_i$ is the set of reachable nodes from i, and $\mathcal{A}_i$ is the set of arcs comprising a path from i to any $j \in \mathcal{T}_i$ (see Fig. 2).

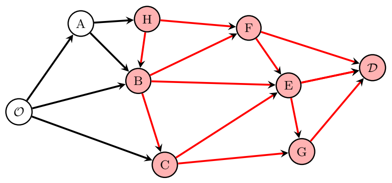

Figure 2: Illustration of $\mathcal{G}_H$

Let $s_{\max}(i)$ denote the maximum charging rate of all the CSs in $\mathcal{T}_i$. We denote the maximum charging rate of CS j as $\bar{\rho}_j$, which corresponds to the largest slope of the piecewise charging function of j. We then set $s_{\max}(i)$ as $\max\{\bar{\rho}_j : \forall j \in \mathcal{T}_i\}$. This improves the computation of the charging potential with respect to [14], where $s_{\max}$ is constant and does not depend on the nodes that are reachable from i.

If the energy available at i is greater than the remaining energy needed to reach $\mathcal{D}$, then $\tilde{\pi}_{\text{cons}}$ can be negative. Thus, two cases are considered when computing a lower bound for the charging time:

$$\pi_{\text{ch}}(i) := \begin{cases} \dfrac{\tilde{\pi}_{\text{cons}}(i)}{s_{\max}(i)} & \text{SoC}(i) - q_{\min} \le \pi_{\text{cons}}(i) \\ 0 & \text{otherwise} \end{cases}$$

This gives a potential that returns the minimal charging time from any node i to $\mathcal{D}$. Thus, a lower bound on the total trip time is given by

$$\tilde{\pi}_{\text{tt}}(i) := \pi_{\text{dr}}(i) + \pi_{\text{ch}}(i) \qquad \forall i \in \mathcal{S}_{\mathcal{O},\mathcal{D}}.$$

We improve this lower bound by accounting for TWs. Let $\pi_{\text{tw}}(i)$ be the minimal stopping time that the EV must perform from i to $\mathcal{D}$ according to the ATWs. The trip accounts for the sum of all the minimum stopping times. Let $\tilde{k}$ be the last TW in the ordered set $\mathcal{W}^R$. Suppose that, when at node i, the EV has not performed all TWs in $\mathcal{W}^R$. Then, $g(i) \le \gamma^L_{\tilde{k}} + t^{\min}_{\tilde{k}}$, where $g(i)$ is the arrival time at i and $\gamma^L_{\tilde{k}}$ is the starting time of TW $\tilde{k}$. In this case, we compute a lower bound for the TWs potential as the sum of all the stopping times that have not been performed by the time $g(i)$. If instead, in node i, the EV has already done all the TWs in $\mathcal{W}^R$, then $g(i) > \gamma^L_{\tilde{k}} + t^{\min}_{\tilde{k}}$ and so the potential for the TWs must be zero. Therefore, if $\pi_{\text{tw}}(i)$ is the TWs potential for node i, we have

$$\pi_{\text{tw}}(i) := \begin{cases} \displaystyle\sum_{k \in \mathcal{W}^R : g(i) < \gamma^L_k} t^{\min}_k & \text{if } g(i) \le \gamma^L_{\tilde{k}} + t^{\min}_{\tilde{k}} \\ 0 & \text{otherwise} \end{cases}$$

For instance, suppose that $\mathcal{W}^R$ contains a 1 hour stop for lunch, one tourism stop for 2 hours and one for sleeping for 11 hours. Then, before lunch $\pi_{\text{tw}}(i) = 14$, after lunch $\pi_{\text{tw}}(i) = 13$, and the next day $\pi_{\text{tw}}(i) = 0$.

We now incorporate π_{tw} in $\tilde{\pi}_{\text{tt}}$. The EV charges during each stop. Thus, we cannot simply sum π_{tw} and π_{ch}, since they may overlap. Instead, we can obtain a better lower bound considering the maximum between π_{ch} and π_{tw}, and then adding the driving potential π_{dr}. So,

$$\pi^1_{\text{tt}}(i) := \pi_{\text{dr}}(i) + \max\{\pi_{\text{ch}}(i), \pi_{\text{tw}}(i)\}$$

defines the lower bound for the total stopping time from i to $\mathcal{D}$. We are not overestimating the real cost, since taking the maximum we consider for each node the best possible charging scenario that at least the EV has to perform.

B. Labels

Label L_{j_m} represents the state of the EV when arriving at the m-th copy of node j, that is dynamically allocated. With this label we keep track of the state of the EV. For notational convenience, we denote $g(j_m)$ by g^m_j. Each label includes the total time needed to reach j_m, so it includes both driving and charging times. For instance, suppose that the EV goes from the n-th copy of i to the m-th copy of j, namely from i_n to j_m. Then the label L_{j_m} considers driving from i to j and the charge in i that is needed to reach j for the m-th time. It excludes the time the EV spends charging in j. The label of the m-th copy of node j, with i_n as its direct predecessor, is:

$$L_{j_m} := (i, g^m_j, h^m_j, f^m_j, p^m_j, \beta^m_j, q^m_j, \underline{q}^m_j, \lambda^m_j, \Delta^m_j, \omega^m_j)$$

where

- i is the node from which the EV arrives to j_m;
- g^m_j is the total travel time from $\mathcal{O}$ to j_m;
- h^m_j is the estimated travel remaining time from j_m to $\mathcal{D}$;
- f^m_j is the estimated arrival time at $\mathcal{D}$ if the path from $\mathcal{O}$ to j_m is performed. It is given by $f^m_j := g^m_j + h^m_j$;
- p^m_j is the label from which the EV arrives, i.e., L_{i_n}, which is the label of the n-th copy of node i, with $n \in \{1, \dots, M_i\}$;
- β^m_j is the additional time spent at j for charging, it is chosen from an ordered set $\boldsymbol{\beta} := \{\beta_1, \beta_2, \dots, \beta_s\}$;
- q^m_j is the amount of energy that is charged in the predecessor node i_n. It is computed to at least respect the consumption e_{ij} and is given by $q^m_j := \overline{q}^m_j - \underline{q}^m_j$, where
$$\overline{q}^m_j := \max\left\{ \underline{q}^n_i + e_{ij},\ Q \right\};$$
- $\underline{q}^m_j$ is the amount of energy of the EV when arriving at j_m. It is computed as $\underline{q}^m_j := \underline{q}^n_i + q^m_j - e_{ij}$;
- λ^m_j is the minimum time the EV must charge at j_m. This amount of time is considered in the next label and not in L_{j_m}. It is defined as

$$\lambda^m_j := \begin{cases} \max\{0,\ \gamma^L_k - g^m_j\} + t^{\min}_k & \text{if TW } k \text{ is performed in } i_n \\ 0 & \text{otherwise} \end{cases}$$

where γ_k^L is the starting time of TW k and $t_k^{\min}$ is its minimum stopping time. TW k is retrieved using ω_j^m (see below);

- Δ_j^m is the charging time in i_n. It is given by

$$\Delta_j^m := \max\left\{ \lambda_i^n,\ \overline{c}_j^m - \underline{c}_j^m \right\} + \beta_j^m,$$

  where $\overline{c}_j^m := \Phi_j^{-1}(\overline{q}_j^m)$ and $\underline{c}_j^m := \Phi_j^{-1}(\underline{q}_i^n)$, with Φ_j^{-1} the inverse of the charging function in node j. The term β_j^m is added to consider the cases in which the EV charges more than needed;
- ω_j^m is the index of the last TW k prior to node j_m.

Parameters Δ, q and $\underline{q}$ are strictly related to $\boldsymbol{\beta}$. As a consequence, also g, h and f depend on $\boldsymbol{\beta}$. The value of λ instead depends strictly on ω.

C. *A* search algorithms for the* ***MPM*** *and* ***MDPM***

Using the labels described above we now outline the A* search algorithm. We start by implementing a heuristic approach to find the fastest path from $\mathcal{O}$ to $\mathcal{D}$, referring to this as **AsM**.

The origin node $\mathcal{O}$ is initialized as follows:

$$\begin{aligned} L_{\mathcal{O}}^1 \ := \ & (\ \mathcal{O}, g_{\mathcal{O}}^1 = t_{\text{start}}, h_{\mathcal{O}}^1 = h^1(\mathcal{O}), f_{\mathcal{O}}^1 = g_{\mathcal{O}}^1 + h_{\mathcal{O}}^1, \\ & p_{\mathcal{O}}^1 = -, 0, 0, \underline{q}_{\mathcal{O}}^1 = Q, 0, 0, 0), \end{aligned}$$

where $g_{\mathcal{O}}^1$ is the starting time of the trip. Thus, $f_{\mathcal{D}}^m$ represents the arrival time in $\mathcal{D}$ and not the duration of the trip. Let $\mathcal{L}$ be the set of all labels, and M_j be a counter of the number of copies of $j \in \mathcal{S}_{\mathcal{O},\mathcal{D}}$. The algorithm keeps track of open labels using a priority queue $\mathcal{Q}$. Every time a new label is created, it is added to $\mathcal{Q}$ in a way that the first element of $\mathcal{Q}$ is always the one with the lowest $f_{\overline{j}}^{\overline{m}}$, among all labels L_j^m, so

$$\overline{j}_{\overline{m}} := \underset{j_m : j \in \mathcal{S}_{\mathcal{O},\mathcal{D}},\ m=1,\ldots,M_j}{\arg\min} \left\{ f_j^m \right\}.$$

We now introduce some of the functions used later in the pseudocode. The function $\textsc{Pop}(\mathcal{Q})$ returns the label with the lowest f in $\mathcal{Q}$, while function $\textsc{Push}(\mathcal{Q}, L_{j_m})$ adds the label L_{j_m} to the queue. Function $\textsc{Star}(\mathcal{G}, j)$ returns the set of nodes $h \in \mathcal{S}_{\mathcal{D}}$ such that $(j, h) \in \mathcal{A}$, in descending order with respect to t_{jh}. The function $\textsc{NextTW}(\mathcal{W}^R, \omega_j^m)$ returns the next TW in $\mathcal{W}^R$ that is not visited when the EV arrives at node j_m. The EV is allowed to arrive at a node with a maximum anticipation time of $\widetilde{\varphi}$. The boolean function $\textsc{NodeHasPoi}(i, \nu)$ returns one if there is at least one POI of the category ν in the neighborhood of node i, and zero otherwise.

Function $\textsc{MaxSlope}(\dot{\mathcal{S}})$ applied to a generic subset $\dot{\mathcal{S}}$ of CS $\mathcal{S}$, returns the maximum slope between all the charging functions of nodes in $\dot{\mathcal{S}}$. To speed up the algorithm, all the subtrees are precomputed. The function $\textsc{Routing}(i, j)$ returns the pair (t_{ij}, e_{ij}), that are respectively the time and the energy required to go from i to j, for any $i \in \mathcal{S}_{\mathcal{O}}, j \in \mathcal{S}_{\mathcal{D}}$. The function $\textsc{MinStop}(g_i)$ returns $\boldsymbol{\pi}_{\text{tw}}(i)$.

The A* algorithm is described in Alg. 1. It starts by initializing the counters for all the copies and storing the subtree of each node. Then the label associated with the origin is created and added to the queue and to the set of all the labels.

Algorithm 1 ASTARSEARCH Algorithm

```
 1: function ASTARSEARCH($\mathcal{G}$, $Q$, $q_{\min}$, $t_{\text{start}}$, $t_{\text{end}}$, $\mathcal{W}^R$, $\boldsymbol{\beta}$)
 2:     for all node $h \in \mathcal{S}_{\mathcal{O},\mathcal{D}}$ do
 3:         $\mathcal{T}[h] := \textsc{SubGraph}(\mathcal{G}, h)$, $M_h := 0$
 4:     end for
 5:     $M_{\mathcal{O}} := 1$, Initialize $L_{\mathcal{O}}^1$; $\mathcal{L} := \{ L_{\mathcal{O}}^1 \}$; $\mathcal{Q} := \{ \mathcal{O} \}$
 6:     while $\mathcal{Q}$ do
 7:         $L_i^n := \textsc{Pop}(\mathcal{Q}) = (\bar{i}, g_i^n, h_i^n, f_i^n, p_i^n, \beta_i^n, q_i^n, \underline{q}_i^n, \lambda_i^n, \Delta_i^n, \omega_i^n)$
 8:         $k := \textsc{NextTW}(\mathcal{W}^R, \omega_i^n)$
 9:         if $i = \mathcal{D}$ then
10:             if $\omega_i^n < |\mathcal{W}^R|$ then go to 6
11:             return $L_i^n$, $\mathcal{L}$
12:         end if
13:         if $k$ is not NONE then                                   // See [A]
14:             IDX $:= \omega_i^n$; C $:= k$
15:             while C is not NONE do
16:                 if $\mathcal{T}[i] \cap \mathcal{S}_C = \emptyset$ then go to 6
17:                 IDX $:=$ IDX $+ 1$; C $:= \textsc{NextTW}(\mathcal{W}^R, \text{IDX})$
18:             end while
19:         end if
20:         NEIGHBORS $:= \textsc{Star}(\mathcal{G}, i)$
21:         for all $j \in$ NEIGHBORS do
22:             $t_{ij}, e_{ij} := \textsc{Routing}(i, j)$
23:             if $e_{ij} > Q - q_{\min}$ then go to 21
24:             for all $\beta \in \boldsymbol{\beta}$ do
25:                 $\Delta,\ q := \textsc{ChargingEnergy}(i, e_{ij}, \underline{q}_i^n)$
26:                 $\Delta := \max\{ \lambda_i^n,\ \Delta \} + \beta, q := \textsc{ChargingForTime}(i, \underline{q}_i^n, \Delta)$
27:                 $g_{\text{temp}} := g_i^n + \Delta + t_{ij}$
28:                 if $g_{\text{temp}} > t_{\text{end}}$ then go to 21
29:                 if $k$ is not NONE then
30:                     if $g_{\text{temp}} > \gamma_k^U$ then go to 21
31:                     if $\gamma_k^L - \widetilde{\varphi} \le g_{\text{temp}} \le \gamma_k^U$ then
32:                         if $j = \mathcal{D}$ or not $\textsc{NodeHasPoi}(j, \nu_k)$ then go to 39
33:                         $M_j := M_j + 1$; $m := M_j$
34:                         $\lambda := \max\left\{ 0,\ \gamma_k^U - g_{\text{temp}} \right\} + t_k^{\min}$
35:                         $L_j^m := \textsc{CreateLabel}(i, j, g_i^n, \underline{q}_i^n, \Delta, q, t_{ij}, e_{ij}, \lambda, \omega_i^n +$
    $1, \beta, L_i^n)$
36:                         $\mathcal{L} := \mathcal{L} \cup \left\{ L_j^m \right\}$; $\textsc{Push}(\mathcal{Q}, L_j^m)$
37:                     end if
38:                 end if
39:                 $M_j := M_j + 1$; $m := M_j$
40:                 $L_j^m := \textsc{CreateLabel}(i, j, g_i^n, \underline{q}_i^n, \Delta, q, t_{ij}, e_{ij}, 0, \omega_i^n, \beta, L_i^n)$
41:                 $\mathcal{L} := \mathcal{L} \cup \left\{ L_j^m \right\}$; $\textsc{Push}(\mathcal{Q}, L_j^m)$
42:             end for
43:         end for
44:     end while
45:     return NONE , NONE                                             // Node not found
46: end function
```

[A] Check if the subtree of current node contains the category of POI that is needed for the next TW and the subsequents ones. Otherwise, goes to the next element in the queue.

The label L_i^n with the lowest value of f is selected and then the algorithm finds the next TW k that must be performed. A check is performed to verify if the current label entails the arrival to the destination point: if so, we verify whether the index ω_i^n of the last visited TW is at least equal $|\mathcal{W}^R|$. If this is not the case, the EV has not visited yet all the non-avoidable TWs, and thus the current label must be discarded. Otherwise, the current label and the set of all the generated labels are returned and the search terminates.

At this point, if k is not NONE the algorithm checks whether or not there exists at least one node in $\mathcal{T}_i$ of the current node i in which the POI constraint of TW k is satisfied. Then it checks the same for all the TWs in $\mathcal{W}^R$ that must be performed after k.

The algorithm proceeds by considering the nodes j such that $(i, j) \in \mathcal{A}$. A sequence of operations assures that the trip from i_n to j_m is feasible, where m is the index of the m-th copy of j that will be created if all the checks are passed. First the energy constraints. If the energy required on arc (i, j) is greater than the maximum amount for the EV, we discard this label, and the algorithm proceeds to the next node in $\textsc{Star}(\mathcal{G}, i)$. Otherwise, if $e_{ij} < Q - q_{\min}$, we consider charging a greater amount of energy with respect to e_{ij} by looping on β.

At line 25 of Alg. 1, the charging time and the charged energy are computed, given the current SoC $\underline{q}_i^n$ and the amount of energy required e_{ij}. The two values are then updated on line 26, taking the current SoC and the arrival time.

We then compute a temporary value g_{temp} of the arrival time in j. In case $g_{\text{temp}} > t_{\text{end}}$, the current label is discarded. We then verify if the selected TW k is in $\mathcal{W}^R$. If so, then the algorithm must satisfy the constraints associated with k. First, if $g_{\text{temp}} > \gamma_k^U$, then the arrival time at j will be after the ending time of k, which is not feasible. If instead, g_{temp} is included in the range $[\gamma_k^L - \varphi, \gamma_k^U]$, then the EV arrives at j during the TW k. In this situation, the user may decide to stop in j, and respect the TW constraints, or to continue driving to the next node h and see if it is possible to respect k in node h. This scenario models the case in which, for instance, instead of respecting the lunch constraints in node j, the user prefers to drive longer and eat at another place. In the case we stay in j to respect TW k, we check whether node j has the POI that is required for k. If so, a label L_j^m is created, imposing that $\omega_j^m := \omega_i^n + 1$ (from j_m on, TW k is respected) and $\lambda_j^m = \max\left\{0,\ \gamma_k^L - g_{\text{temp}}\right\} + t_k^{\min}$. The max function is used to compute how much time in advance the EV arrives in j, so the minimum stopping time imposed from any arc from j_m is λ_j^m. If instead node j does not have any POI of the given category ν_k, we can step over and create a label that goes from i_n to j_m without imposing a minimum stopping time λ_j^m. In this case $\lambda_j^m := 0$ and $\omega_j^m := \omega_i^n$. In both cases, the newly created label L_{j_m} is added to the set of labels $\mathcal{L}$ and pushed to the queue $\mathcal{Q}$. Finally, the loop on β continues after updating β. Alg.s 2 and 3 outline the creation of a new label, and its heuristic. Finally, if it is not possible to reach $\mathcal{D}$ respecting all the imposed constraints, the algorithm returns NONE .

Algorithm 2 CREATELABEL function. It creates the label from node a to node b, given the arrival time g in a, the SoC $\underline{q}$, the charging time Δ, the charged energy q, the driving time and energy, t, e, the minimum amount of time needed to charge in node b in the next label, the index of the last performed TW ω, the charger additional time β and the previous label L.

```
1: function CreateLabel($a$, $b$, $g$, $\underline{q}$, $\Delta$, $q$, $t$, $e$, $\lambda$, $\omega$, $\beta$, $L$)
2:     $g := g + \Delta + t$; $\underline{q} := \underline{q} + q - e$
3:     $h :=$ Heuristic($b, \underline{q}, g$)
4:     $\underset{\sim}{f} := g + f$
5:     $\widetilde{L} := (a, g, h, f, L, \beta, q, \underline{q}, \lambda, \Delta, \omega)$
6:     return $\widetilde{L}$
7: end function
```

We modify the previously described A* search algorithm for the **MDPM**. We refer to this algorithm as **AsDM**. We add

Algorithm 3 HEURISTIC function. It returns the estimated remaining time from node i to $\mathcal{D}$ in graph $\mathcal{G}$, considering the SoC $\underline{q}$ and arrival time g.

```
1: function Heuristic($i$, $\underline{q}$, $g$)
2:     $\boldsymbol{\pi}_{\text{tw}}(i) :=$ MinStop($g$)
3:     if $\underline{q} - q_{\min} \geq \boldsymbol{\pi}_{\text{cons}}(i)$ then
4:         $\boldsymbol{\pi}_{\text{ch}}(i) := 0$
5:     else
6:         $s_{\max} :=$ MaxSlope(SubGraph($\mathcal{G}, i$))
7:         $\boldsymbol{\pi}_{\text{ch}}(i) := [\boldsymbol{\pi}_{\text{cons}}(i) - (\underline{q} - q_{\min})]/s_{\max}$
8:     end if
9:     return $\boldsymbol{\pi}_{\text{dr}}(i) + \max\{\boldsymbol{\pi}_{\text{tw}}(i),\ \boldsymbol{\pi}_{\text{ch}}(i)\}$
10: end function
```

two parameters in the labels, as follows

$$L_{j_m} := (i, g_j^m, h_j^m, f_j^m, p_j^m, \beta_j^m, q_j^m, \underline{q}_j^m, \lambda_j^m, \Delta_j^m, \\ \omega_j^m, \Sigma_j^m, \underline{\tau}_j^m).$$

The first parameter is Σ_j^m, which represents the total score gained from $\mathcal{O}$ to the current label. For copy j_m,

$$\Sigma_j^m := \Sigma_i^n + \sigma_j, \quad \Sigma_{\mathcal{O}}^1 := 0.$$

The second parameter is $\underline{\tau}_j^m$, it represents the arrival time in copy j_m. All the TWs constraints are now satisfied using this parameter instead of the arrival time g. This means, for instance, that $\tau_{\text{temp}} = \underline{\tau}_j^m + \Delta - t_{ij}$, and every g_{temp} is replaced with τ_{temp}. The minimum waiting time becomes $\lambda_j^m := \max\left\{0,\ \gamma_k^L - \underline{\tau}_j^m\right\} + t_k^{\min}$. For the origin node we have $\underline{\tau}_{\mathcal{O}}^1 := t_{\text{start}}$ and $g_{\mathcal{O}}^1 := 0$. Finally, due to the different definition of labels, the function CREATELABEL is replaced by CREATELABELDISCOUNTED, and is outlined in Alg. 4.

Algorithm 4 CREATELABELDISCOUNTED function. It creates the label from node a to node b, given the discounted cost g to a, the SoC $\underline{q}$, the charging time, Δ, the charged energy q, the driving time and energy, t, e, the minimum amount of time needed to charge in node b in the next label, the index of the last performed TW ω, the charger additional time β, the previous label L, the arrival time $\underline{\tau}$, and the gained profit Σ.

```
1: function CreateLabelDiscounted($a$, $b$, $g$, $\underline{q}$, $\Delta$, $q$, $t$, $e$, $\lambda$, $\omega$, $\beta$, $L$,
   $\underline{\tau}$, $\Sigma$)
2:     $\underline{\tau} := \underline{\tau} + \Delta + t$; $\underline{q} := \underline{q} + q - e$
3:     $\sigma :=$ Score($b$)
4:     $g := g + \Delta + t - \mu\sigma$
5:     $h :=$ HeuristicDiscounted($B, \underline{q}, \underline{\tau}, \Sigma$)
6:     $\underset{\sim}{f} := g + f$
7:     $\widetilde{L} := (A, g, h, f, L, \beta, q, \underline{q}, \lambda, \Delta, \omega, \Sigma, \underline{\tau})$
8:     return $\widetilde{L}$
9: end function
```

Furthermore, function HEURISTIC, is replaced with HEURISTICDISCOUNTED, described in Alg. 5. The new function takes as input the additional parameter Σ and returns the following discounted estimated time to reach $\mathcal{D}$:

$$\boldsymbol{\pi}_{\text{tt}}^2(i) := \boldsymbol{\pi}_{\text{tt}}^1(i) - \mu\Sigma.$$

Thus, the final value of f computed for node $\mathcal{D}$ can be compared to the one computed with **MDPM**.

Algorithm 5 HeuristicDiscounted function. It returns the estimated time from node i to $\mathcal{D}$, considering the SoC $\underline{q}$, the arrival time g and the gained score Σ

```
1: function HeuristicDiscounted($i$, $\underline{q}$, $g$, $\Sigma$)
2:     $\pi_{\text{tw}}(i) := \max\left\{0,\ \gamma_{\bar{k}}^{L} - g + t_{\bar{k}}^{\min}\right\}$
3:     if $\underline{q} - q_{\min} \geq \pi_{\text{cons}}(i)$ then                // See [A]
4:         $\widetilde{\pi}_{\text{cons}}(i) := 0$
5:     else
6:         $s_{\max} := $ MaxSlope(SubGraph($\mathcal{G}, i$))
7:         $\pi_{\text{ch}}(i) := \dfrac{\pi_{\text{cons}}(i) - (\underline{q} - q_{\min})}{s_{\max}}$
8:     end if
9:     return $\pi_{\text{dr}}(i) + \max\{\pi_{\text{tw}}(i),\ \pi_{\text{ch}}\}(i) - \mu\Sigma$
10: end function
```

[A] Available energy is potentially sufficient to reach $\mathcal{D}$.

IV. Computational experiments

A. Data description and preprocessing

We obtain CS locations and specifications from a company, whose name we cannot disclosed due to privacy reasons. The CSs were extracted from a bounding box that goes from 43.55° to 49.05° latitude and from 8.68° to 13.11° longitude, covering parts of Italy, Germany, Austria, Switzerland and the totality of Liechtenstein and San Marino (see Fig. 3).

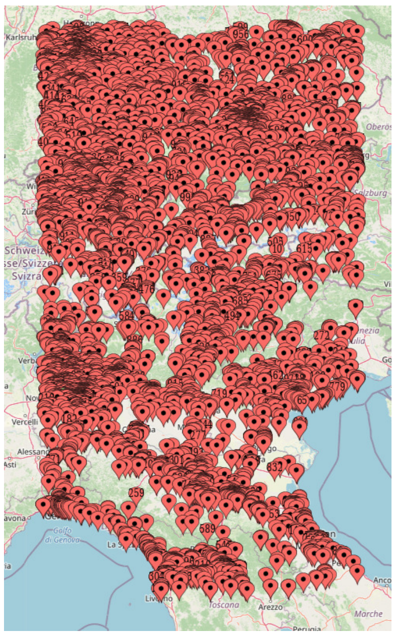

Figure 3: Distribution of the CSs (created with mapcustomizer.com [7]).

CS i has a charging power p_i. We categorize the CSs as follows:

- slow: $p_i \leq 11\,\text{kW}$,
- medium: $11\,\text{kW} < p_i \leq 30\,\text{kW}$,
- fast: $p_i > 30\,\text{kW}$.

We have performed a number of preprocessing actions on the provided data set. The first step was to group CSs at similar locations, for each group the fastest charger within that group was considered. We then considered that CSs within a radius of $r_M = 100\,\text{m}$ form a cluster and if one CS belongs to two or more clusters, then all of them are merged. This reduction is based on the Haversine formula for computing the distance between each pair of nodes and assumes that the distances are symmetric. Merging all the CSs into a cluster produces a new CS that replaces the other CSs in the cluster. The cluster position is set as the average of the GPS coordinates of the CSs that comprise it, while its charging speed is set as the maximum among the CSs that comprise it. The final dataset is denoted by Γ. The final number of CSs is summarized in Tab. I.

Table I: Distribution of CSs for each country.

	Original # CSs	# CSs after grouping	# CSs after clustering
Germany	59 339	5 955	4 135
Italy	15 009	5 713	4 911
Switzerland	5 400	1 583	1 334
Austria	3 679	1 488	1 219
Liechtenstein	65	33	27
San Marino	34	17	15
Total	**83 526**	**14 789**	**11 641**

Using the post-processed database, we computed the distance and time matrix for each pair of CSs, obtained from multiple requests to the Open Source Routing Machine API (OSRM, [8]) server. The energy required by each arc was instead given by the product of the arc length and the average consumption per kilometer, as in [10].

The sets of required $\mathcal{W}^R$, and weakly mandatory time windows $\mathcal{W}^M$ are given as input. Furthermore, we consider that tourism AWTs $\mathcal{W}^T$ are also given as input. For $k \in \mathcal{W}^T$ a single CS location $\mathcal{P}_k$ is given. To construct the set $\mathcal{S}_k$, the Haversine distance is computed by searching for CSs within a radius r_T centered in $\mathcal{P}_k$. The CSs with a distance less than r_T are included in $\mathcal{S}_k$. If no eligible nodes are present, the radius is iteratively increased by δ_T and the search is repeated, up to a maximum of $\widetilde{r}_T$. If $\mathcal{S}_k$ is still empty, the computation is resumed and an error informs that it is not possible to reach $\mathcal{P}_k$ unless the EV is left more than $\widetilde{r}_T$ away. This process simulates the approach that a user needs to search for a CS near the location she wants to visit. If no CS is available at a reasonable distance from $\mathcal{P}_k$, then the user can decide to remove or change that tourism stop. For our experiments, we used $r_T = 2\,\text{km}$, $\delta_T = 200\,\text{m}$ and $\widetilde{r}_T = 4\,\text{km}$.

Given the origin $\mathcal{O}$, the destination $\mathcal{D}$ and the set of tourism stops $\mathcal{W}^T$, we construct the set of chargers $\mathcal{S}$ and the set of arcs $\mathcal{A}$. Using OSRM, the optimal path without charging stops is computed and is used as a lower bound for **SPM**. All the CSs that are within a range of 5 km centered on the OSRM optimal route form the set $\mathcal{S}$. Finally, we construct $\mathcal{A}$ as the set of all arcs (i, j) connecting nodes in $\mathcal{S}_{\mathcal{O},\mathcal{D}}$.

In addition, we remove from $\mathcal{S}$ all the nodes that have a distance from $\mathcal{O}$ that is less then $r^{\min}$, due to the fact that we want a solution with a small number of stops. Selecting one of those would have caused the EV to stop for just a few kilometers from the origin point, which is not desirable. The same reasoning is applied to $\mathcal{D}$.

We also remove arcs that are going in the opposite direction with respect $\mathcal{D}$. More precisely, for each arc $(i, j) \in \mathcal{A}$, if by going from i to j the distance to $\mathcal{D}$ is reduced, then (i, j) is kept, otherwise it is deleted. This process nearly halves the amount of edges included in $\mathcal{A}$.

Since the EV needs to respect the battery capacity Q and we want the total travel time to be small, we consider only the arcs (i, j) such that $r^{\min} \leq d_{ij} \leq r^{\max}$, where

$$r^{\max} := \lfloor (Q - q_{\min})/\eta \rfloor.$$

Finally, nodes that were not reachable from any other CS, as well as the corresponding arcs, are deleted.

B. Experiments and results

The codes were implemented in Python, using IBM ILOG CPLEX Optimization Studio 20.1. The tests ran on a single core of an Apple MacBook Pro with 8 core Apple M1 processor of 3.2 GHz, with 8 GB of LPDDR4 RAM.

The considered EV was a Škoda Enyaq iV 60, with a net battery capacity of $Q := 58\,\mathrm{kWh}$ and a maximum average consumption of $\eta := 0.187\,\mathrm{kWh/km}$ [3]. It has a maximum power charge $P := 40\,\mathrm{kW}$, and we set the minimum required energy level $q_{\min}$ to $15\,\mathrm{kWh}$.

Two subsets of CSs were created: a small one, Γ_1, with 650 CSs, used to compare the exact solution obtained with the MILP models with the A* search algorithms, and a larger one, Γ_2, with 5 813 CSs, used to test the A* search. Both datasets are extracted with uniform probability from Γ, so we must guarantee that in each tourism stop there is at least one CS. We created a $5\,\mathrm{km} \times 5\,\mathrm{km}$ square centered in the coordinates of each tourism stop and uniformly extracted four CSs. Γ_1 and Γ_2 were obtained by selecting uniformly a certain amount of CSs from Γ and adding them to the ones selected for the tourism stops.

For both sets of CSs we use the same set of instances defined as trips. Since our CSs lay in a rectangular area that covers part of central Europe, those instances are chosen so that they fully lay in the same geographical area. We generate three main trips, with some variations, such as starting time, presence of tourism stops and minimum stopping times, presence or not of lunch (1 hour) and night (11 hours) ATWs. The ATW of lunch is [12:00, 13:30], while that of the night is [19:00, 22:30]. The instances have the following $\mathcal{O} - \mathcal{D}$ pairs:

- from Genoa to Zürich denoted by `GeZu`;
- from Livorno to Regensburg denoted by `LiRe`;
- Stuttgart to Ancona denoted by `StAn`

The remaining details are summarized in Tab. II. The name of the $\mathcal{O} - \mathcal{D}$ is followed by a running number to distinguish the instances. Overall, we created 20 instances.

We tested the MILP shortest path model, **SPM**, and maximum profit model, **MPM**, as well as both A* heuristics, the A* search algorithm for the **SPM**, **AsM**, and its variant for the maximum discount profit model, **AsDM**, for set Γ_1. In addition, only the heuristics **AsM** and **AsDM** were applied to

Table II: Description of the instances

ID	Departure	Arrival	Tourism Stops			Lunch	Night
			At	When	Min stop		
GeZu1	10:00	$18{:}30^{+1}$	Lugano	14:00-17:30	2h	-	-
GeZu2	10:00	$18{:}30^{+1}$	Lugano	14:00-17:30	2h	Yes	-
LiRe1	10:00	$18{:}30^{+1}$	-	-	-	Yes	-
LiRe2	10:00	$18{:}30^{+1}$	-	-	-	-	Yes
LiRe3	10:00	$18{:}30^{+1}$	Verona	14:00-17:30	2.5h	Yes	-
LiRe4	10:00	$18{:}30^{+1}$	Verona	14:00-17:30	2.5h	-	Yes
LiRe5	4:00	$18{:}30^{+1}$	Verona	14:00-17:30	2.5h	Yes	Yes
LiRe6	6:00	$18{:}30^{+1}$	Verona	14:00-17:30	2.5h	Yes	Yes
LiRe7	8:00	$18{:}30^{+1}$	Verona	14:00-17:30	2.5h	Yes	Yes
LiRe8	10:00	$18{:}30^{+1}$	Verona	14:00-17:30	2.5h	Yes	Yes
LiRe9	10:00	$18{:}30^{+1}$	Verona	14:00-17:30	2h	Yes	Yes
LiRe10	10:00	$18{:}30^{+1}$	Verona	14:00-17:30	3h	Yes	Yes
StAn1	10:00	$18{:}30^{+1}$	-	-	-	Yes	-
StAn2	10:00	$18{:}30^{+1}$	Vaduz	14:00-17:30	2h	Yes	-
StAn3	20:00	$18{:}30^{+1}$	Bologna	$7{:}00\text{-}10{:}30^{+1}$	2h	Yes	-
StAn4	10:00	$18{:}30^{+1}$	Vaduz	14:00-17:30	2h	Yes	Yes
StAn5	10:00	$18{:}30^{+1}$	Bologna	$7{:}00\text{-}10{:}30^{+1}$	2h	Yes	Yes
StAn6	6:00	$2{:}00^{+1}$	Vaduz	9:30-13:00	2h	-	-
			Bologna	18:30-22:00	2h		
StAn7	10:00	$22{:}00^{+1}$	Vaduz	14:00-17:30	2h	-	Yes
			Bologna	$9{:}30\text{-}12{:}30^{+1}$	2h		
StAn8	13:00	$22{:}00^{+1}$	Vaduz	14:00-17:30	2h	Yes	Yes
			Bologna	$14{:}00\text{-}17{:}30^{+1}$	2h		

set Γ_2. Then for all **AsDM** models we solved **SPM** imposing that the EV must use the arcs selected by the heuristic. CPLEX was unable to solve any instance in Γ_2 within one hour. We denote by GS_x^y (GT_x^y) the relative gap between the total score (trip time) of the models x and y. Also, to ease the notation, in the following we denote by $\mathbf{AsDM}_{\mu_0}$ the application of **AsDM** for $\mu = \mu_0$, and use TS for the total score, TT for the trip time, and RT for the run time.

As seen in Tab. II, some instances describe the same trip, with different timings, and thus lead to the same set of unconstrained shortest optimal paths, and to the same set of CSs. For this reason, we subdivide the instances, and for both Γ_1 and Γ_2 extract the set of CSs related to each subdivision and store them. In this way, instances in the same subdivision use the same graph for all the models. In Tab. III we list the size for each subdivision.

Table III: Number of nodes and arcs for each instance.

Instances	Dataset Γ_1				Dataset Γ_2			
	Before		After		Before		After	
	Nodes	Arcs	Nodes	Arcs	Nodes	Arcs	Nodes	Arcs
GeZu1, GeZu2	36	1 260	22	52	296	87 320	212	3 236
LiRe1, LiRe2	43	1 806	36	157	480	229 920	391	22 230
LiRe3 to LiRe10	49	2 352	42	244	492	241 572	403	23 889
StAn1	84	6 972	54	276	667	444 222	415	19 762
StAn2, StAn4	104	10 712	73	629	820	671 580	555	34 622
StAn3, StAn5	83	6 806	54	276	666	442 890	414	19 698
StAn6 to StAn8	103	10 506	73	629	819	669 942	554	34 542

We use the dataset Γ_1 to analyze the performance of the A* search with respect to the exact solution found. Initially we find the shortest path value T^{opt} of **SPM** and **AsM**. Then we compute the maximum relaxed time $T^{\max} = T^{\text{opt}} + T^{\text{add}}$ and solve the **MPM** model. For the evaluation, we fix the value of T^{add} to 1h 30'. The result is then compared to the score gained with **AsDM** for different values of the score multiplier μ, $\mu = 1, 2$. For the heuristic algorithms, we set $\boldsymbol{\beta} := \{0, 1, 4\}$.

a) Small dataset: We solve the **MPM** model and apply the A* algorithm for $\mu = 2$. The results are in Tab. IV and V.

First, the solutions obtained with the A* search are refined using a MILP formulation. In particular, we create another **SPM** in which we set $x_{ij} = 1$ for each arc (i, j) selected in the final solution of **AsDM** for $\mu = 2$. The total trip times obtained by both approaches are quite similar. The results are summarized in Tab. IV and compared with the shortest path model **SPM**. The total trip time increases in average 3.3% when the refined **SPM** is applied, which is a small detour with respect to shortest path.

Table IV: Comparison of total trip time (in hours) between **SPM** and the refined **SPM** in Γ_1

ID	**SPM**	Ref. **SPM**	$GT^{\text{Ref. SPM}}_{\text{SPM}}$
	TT	TT	×100 [%]
`GeZu1`	8.6	9.0	4.7
`GeZu2`	9.0	9.0	0.0
`LiRe1`	14.1	15.7	11.2
`LiRe2`	23.6	24.4	3.4
`LiRe3`	15.7	16.1	2.5
`LiRe4`	24.6	25.5	3.7
`LiRe5`	30.6	30.9	1.0
`LiRe6`	28.6	28.6	0.0
`LiRe7`	26.6	26.7	0.4
`LiRe8`	25.3	25.4	0.4
`LiRe9`	25.1	26.2	4.4
`LiRe10`	27.7	26.6	3.5
`StAn1`	15.8	17.4	10.1
`StAn2`	18.2	19.5	7.1
`StAn3`	16.9	17.3	2.4
`StAn4`	29.0	29.8	3.8
`StAn5`	26.3	27.3	3.8
`StAn6`	19.0	19.3	1.6
`StAn7`	29.0	29.2	0.7
`StAn8`	29.8	29.9	0.3
Average			3.3

In Tab. V we compare the total score gained and the run time of **MPM** and **AsDM** for $\mu = 2$. In average, the relative change between the maximum score computed with **MPM** and the one computed with **AsDM** there is an average of 11.9% worse scores with respect to the optimal solution found by **MPM**. This last value is quite large, with some instances having a relative change of over 25%. A possible approach to obtain better results may be to rely on different values of the parameter μ. The run time is quite low for all the models, but it is worth noting that we are considering the small graph created with the CSs in Γ_1.

b) Medium dataset: We want to test the A* search algorithm for the medium dataset Γ_2. We tested all the instances using only the heuristic approaches, **AsM** and **AsDM**. The results are reported in Tab. VI and VII. According to Tab. VI, we obtain an average increase of the total score with $\mu = 1$ with respect to $\mu = 2$. The difference is due to the fact that with $\mu = 2$ the shortest arcs become more important, even

Table V: Comparison of total score and run time (in sec.) between **MPM** and **AsDM** with $\mu = 2$ in Γ_1

ID	**MPM**		**AsDM**$_2$		$GS^{\textbf{AsDM}_2}_{\textbf{MPM}}$
	RT	TS	RT	TS	×100 [%]
`GeZu1`	0.0	7.7	0.0	7.7	0.0
`GeZu2`	0.0	4.6	0.0	4.6	0.0
`LiRe1`	0.5	19.2	0.0	12.8	33.3
`LiRe2`	1.3	16.6	0.0	13.8	16.9
`LiRe3`	1.2	23.2	0.1	22.6	2.6
`LiRe4`	1.3	22.6	0.1	18.6	17.7
`LiRe5`	0.8	22.8	0.0	22.8	0.0
`LiRe6`	0.6	20.9	0.0	15.1	27.8
`LiRe7`	0.7	21.6	0.0	21.3	1.4
`LiRe8`	1.4	22.8	0.0	17.7	22.4
`LiRe9`	1.5	23.9	0.0	21.4	10.5
`LiRe10`	0.7	17.4	0.0	14.7	15.5
`StAn1`	1.4	17.1	0.0	15.2	11.1
`StAn2`	3.1	18.5	0.0	16.1	13.0
`StAn3`	1.4	15.2	0.1	11.4	25.0
`StAn4`	2.6	13.7	0.0	13.3	2.9
`StAn5`	2.0	17.5	0.2	14.6	16.6
`StAn6`	0.5	18.2	0.0	18.0	1.1
`StAn7`	0.3	19.3	0.0	19.2	0.5
`StAn8`	0.3	19.3	0.0	15.6	19.2
Average	1.1		2.1		11.9

with a lower score in the arrival node. The final solution might improve by tuning μ, also dynamically for each arc. The run time for the **AsM** model is quite high, with an average of 118.8 sec for the trip from Livorno to Regensburg and of 81.4 sec for the trip from Stuttgart to Ancona. If instead we analyze the **AsDM** model, we see a meaningful drop in the run time, with some exceptions. In particular, the instances that continue to have higher run times are the ones that have large time intervals without any TWs constraints. For instance, after the lunch break `StAn1` has no other TW that constrains the problem, so the research for a best bound solution is more time demanding.

The same holds for `LiRe1`, `LiRe2` and `StAn6`. Instead, `StAn8`, the instance with more TWs, is solved in less than 1 sec. in each discounted model. When the TWs are balanced along the trip, with not too many uncovered time intervals, the computation with the A* search is very fast, even in medium sized graphs. In Tab. VII we can see an average total trip time variation of less than 6.0% in the **AsDM** models with respect to the **AsM**.

V. Conclusions

We investigate the problem of finding a path for an EV performing a a long trip. The objective is to select CSs that better match the user preferences while respecting ATWs constraints.

We proposed an A* algorithm **AsM** for the shortest path version of our problem. We then proposed the **AsDM** algorithm which privileges POIs preferred by the user. This algorithm was quite fast in finding a shortest path solution with high total score. The algorithm performed fairly well with respect

Table VI: Comparison of total score and run time (in sec.) between **AsM** and **AsDM** with $\mu = 1, 2$ in Γ_2

ID	AsM		AsDM$_1$		AsDM$_2$		$GS^{\text{AsDM}_1}_{\text{AsM}}$	$GS^{\text{AsDM}_2}_{\text{AsM}}$
	RT	TS	RT	TS	RT	TS	×100 [%]	×100 [%]
GeZu1	0.1	8.5	0.0	9.7	0.0	9.7	14.1	14.1
GeZu2	0.1	7.1	0.0	8.5	0.0	9.1	19.7	28.2
LiRe1	0.7	6.2	0.1	22.8	71.9	22.8	267.7	267.7
LiRe2	175.5	11.7	14.0	21.0	25.7	21.0	79.5	79.5
LiRe3	36.5	4.7	0.5	23.2	0.7	23.3	393.6	395.7
LiRe4	77.3	8.6	0.3	27.3	0.4	19.3	217.4	124.4
LiRe5	196.6	11.0	6.8	26.2	4.4	26.2	138.2	138.2
LiRe6	463.7	14.9	22.7	20.0	25.8	20.0	34.2	34.2
LiRe7	213.2	16.5	0.4	27.4	0.3	27.4	66.1	66.1
LiRe8	132.1	8.8	0.3	23.9	0.1	24.5	171.6	178.4
LiRe9	78.6	12.9	0.5	19.4	0.3	23.6	50.4	82.9
LiRe10	128.4	7.6	0.1	23.7	0.4	17.5	211.8	130.3
StAn1	0.7	8.1	134.4	19.2	540.4	19.2	137.0	137.0
StAn2	2.8	5.6	0.5	21.2	0.2	17.0	278.6	203.6
StAn3	99.8	9.7	15.0	19.0	40.2	19.0	95.9	95.9
StAn4	90.6	18.6	62.2	26.1	96.0	26.1	40.3	40.3
StAn5	92.7	12.8	2.7	15.6	4.7	15.6	21.9	21.9
StAn6	30.2	4.1	34.4	14.3	36.5	10.5	248.8	156.1
StAn7	148.9	6.9	37.9	12.9	38.4	19.0	87.0	175.4
StAn8	185.6	11.4	0.2	18.6	0.1	18.8	63.2	64.9
Average	107.7		16.6		44.3		131.9	121.7

Table VII: Comparison of total trip time (in hours) between **AsM** and **AsDM** with $\mu = 1, 2$ in Γ_2

ID	AsM	AsDM$_1$	AsDM$_2$	$GT^{\text{AsDM}_1}_{\text{AsM}}$	$GT^{\text{AsDM}_2}_{\text{AsM}}$
	TT	TT	TT	×100 [%]	×100 [%]
GeZu1	8.6	8.6	8.6	0.0	0.0
GeZu2	8.8	8.9	9.5	1.1	8.0
LiRe1	13.5	14.9	14.9	10.4	10.4
LiRe2	23.1	24.5	24.5	6.1	6.1
LiRe3	15.4	16.8	16.7	9.1	8.4
LiRe4	24.8	26.3	26.3	6.0	6.0
LiRe5	31.0	32.0	32.0	3.2	3.2
LiRe6	28.6	30.0	30.0	4.9	4.9
LiRe7	26.6	27.6	27.6	3.8	3.8
LiRe8	25.0	26.4	26.4	5.6	5.6
LiRe9	24.6	25.5	26.0	3.7	5.7
LiRe10	25.5	26.8	26.9	5.1	5.5
StAn1	15.1	16.6	16.6	9.9	9.9
StAn2	17.5	19.0	19.0	8.6	8.6
StAn3	15.8	17.0	17.0	7.6	7.6
StAn4	27.3	28.7	28.7	5.1	5.1
StAn5	26.2	27.4	27.4	4.6	4.6
StAn6	17.8	19.0	19.2	6.7	7.9
StAn7	28.0	29.5	29.1	5.4	3.9
StAn8	29.7	30.2	31.0	1.7	4.4
Average				5.4	6.0

to the exact solutions. This comparison was only possible on small instances, but the run times tend to increase with the size of the graph. If instead, the chosen trip has many TWs, then the computation is very fast even in medium sized graphs.

The run time of the MILP formulations increases exponentially in the size of the graph, so it can be computationally infeasible to solve even for medium size graphs. Both **AsM** and **AsDM** solve this problem by storing only the promising states of the EV. However, the heuristics cannot manage real values of the additional charging time, so we must discretize those values using the set β. Solving again the **SPM** model with the arcs selected by the heuristics helps to optimize the charging times on the final solution.

Despite the promising results obtained by **AsM**, **AsDM**, there is potential for further improvements. More efficient heuristics can be developed to account for the potentials, leading to speeding up the A* search algorithm. Such speeding ups may allow exploring more labels, yielding improved solutions.

References

[1] M. Baum, J. Dibbelt, A. Gemsa, D. Wagner, and T. Zündorf. Shortest feasible paths with charging stops for battery electric vehicles. In *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems*, pages 1–10, 2015. https://doi.org/10.1145/2820783.2820826.

[2] M. Baum, J. Dibbelt, T. Pajor, J. Sauer, D. Wagner, and T. Zündorf. Energy-optimal routes for battery electric vehicles. *Algorithmica*, 82:1490–1546, 2020. https://doi.org/10.1007/s00453-019-00655-9.

[3] API ChargePrice.com. Open EV Data, Chargeprice.app API. https://github.com/chargeprice/open-ev-data. Accessed: 18-03-2022.

[4] S. Erdoğan and E. Miller-Hooks. A green vehicle routing problem. *Transportation research part E: logistics and transportation review*, 48:100–114, 2012. https://doi.org/10.1016/j.tre.2011.08.001.

[5] E. Fadda, D. Manerba, G. Cabodi, P. Camurati, and R. Tadei. Kpis for optimal location of charging stations for electric vehicles: the biella case-study. In *2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 123–126. IEEE, 2019. http://dx.doi.org/10.15439/2019F171.

[6] A. Froger, J. E. Mendoza, O. Jabali, and G. Laporte. Improved formulations and algorithmic components for the electric vehicle routing problem with nonlinear charging functions. *Computers & Operations Research*, 104:256–294, 2019. https://doi.org/10.1016/j.cor.2018.12.013.

[7] P. Kaeding. MapCustomizer. https://www.mapcustomizer.com. Accessed: 12-03-2022.

[8] D. Luxen and C. Vetter. Real-time routing with openstreetmap data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 513–516, New York, NY, USA, 2011. ACM. https://doi.org/10.1145/2093973.2094062.

[9] A. Montoya, C. Guéret, J. E. Mendoza, and J. G. Villegas. A multi-space sampling heuristic for the green vehicle routing problem. *Transportation Research Part C: Emerging Technologies*, 70:113–128, 2016. https://doi.org/10.1016/j.trc.2015.09.009.

[10] A. Montoya, C. Guéret, J. E. Mendoza, and J. G. Villegas. The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological*, 103:87–110, 2017. https://doi.org/10.1016/j.trb.2017.02.004.

[11] M. Schiffer, S. Stütz, and G. Walther. Electric commercial vehicles in mid-haul logistics networks. In *Behaviour of Lithium-Ion Batteries in Electric Vehicles*, pages 153–173. Springer, 2018. https://doi.org/10.1007/978-3-319-69950-9_7.

[12] M. Schneider, A. Stenger, and D. Goeke. The electric vehicle-routing problem with time windows and recharging stations. *Transportation science*, 48:500–520, 2014. https://doi.org/10.1287/trsc.2013.0490.

[13] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231:1–21, 2013. https://doi.org/10.1016/j.ejor.2013.02.053.

[14] T. Zündorf. Electric vehicle routing with realistic recharging models. *Unpublished Master's thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany*, 2014.