TU Delft

Bachelor Thesis

Applied mathematics

The influence of location on the profitability of a charging station

Author:
Matthijs Arnoldus

Supervisor:
Dr. ir. J.T. van Essen

Other committee members:
Prof. dr. ir. A.W. Heemink

November 26, 2021

TU Delft

Preface

This thesis was written for the final project of the bachelor in Applied Mathematics. It is extended from my bachelor project for my bachelor in Computer Science.

While writing this thesis, I assumed readers to have basic general mathematical knowledge. However, some more knowledge of mathematical optimisation is required to understand the model developed in Section 4.2.

I would like to thank my supervisor Dr. ir. J.T. van Essen for her supervision and mathematical support during the writing of this thesis, and my supervisor from the Computer Science Department Dr. V. Robu for his supervision on my Computer Science thesis, which forms a basis for this thesis.

Delft, 17 November 2021
Matthijs Arnoldus

Summary

With the increasing number of electric vehicles on the road, the routing problem has become more complex. As charging electric vehicles takes longer than fueling non-electric vehicles, congestion can occur at charging stations. This might lead to the shortest route not being the fastest route, due to long waiting times at the stations. By communicating the intentions of each vehicle, the vehicles can spread out over multiple stations. This thesis investigates the effect of such a routing system on the profitability of charging stations. In particular, the influence of a charging station's location on its profitability has been researched. In order to do this, a pricing model has been developed to extend the routing model. It has become clear that due to the intention-aware routing, the number of visits at stations is harder to predict, and there is not always a clear pattern to be found between the location and the number of visits. This research does, however, propose an optimisation model, which can decide where to build a new charging station, such that the number of visits will be the highest.

Contents

1. Introduction

In the past years, the number of electric cars has significantly increased with increases of 40% in 2019, 63% in 2018 and 58% in 2017 [1]. With this increasing number, new challenges arise. One of the drawbacks of electric vehicles over non-electric vehicles is the long charging time. This can cause long waiting times at charging stations if multiple vehicles go to the same station. In order to overcome this issue, De Weerdt et al.[2] have proposed a way of routing electric vehicles such that they take into account the waiting times at charging stations. This solution, called Intention-Aware Routing, keeps track of the intentions of each vehicle connected to the system. These vehicles can then retrieve the expected waiting times at each station from the system, and calculate their fastest route.

This, however, is only one way of looking at the electric vehicle routing problem. In other work that has been done so far, different approaches are considered, such as routing with minimal energy consumption [3] or routing based only on the competition in price between buyer and consumer, without taking into account route lengths [4]. What most of these papers have in common, is that they look at the perspective of the vehicles. But, optimizing the road network itself can also improve the efficiency of routing and charging those vehicles. Also, looking at the perspective of a charging station can give insights in how a charging station can raise its profits, or whether multiple charging stations can change their prices, capacity or queuing policies to reduce congestion.

In this thesis, the relation between the location of a charging station and its profitability is described. In other words, given a road network with charging stations, we want to determine which charging stations are more profitable than others. In order to do this, we extend the model of De Weerdt et al. with a pricing model. This model is then simulated using various graph topologies to obtain a relation between the location of a station and its profit.

The research question that will be answered is:

How does the location of a charging station for electric vehicles, within a road network modelled as a graph, affect its profitability?

This question is divided into four sub-questions, which are:

1. In a situation where prices are not taken into account (i.e. prices are equal in all stations), how does the location of a charging station influence the number of times it is visited?

2. How can we add the possibility to charge different prices into the model?

3. What is the effect of charging different prices based on the location of a charging station?

4. Given a road network, how can we determine the most profitable location to build a new charging station?

This thesis is structured in the following manner. Chapter 2 contains an analysis of the existing literature on routing and charging electric vehicles. The model of De Weerdt et al. [2] is explained in Chapter 3. Chapter 4 describes the methodology applied in this research. In Chapter 5, the obtained results are presented. A discussion of these results is given in Chapter 6. Finally, Chapter 7 contains the conclusions taken from this research and suggestions for future work.

2. Literature

In the past years, several studies on electric vehicles have been conducted. This chapter contains a review of the existing literature on routing and charging electric vehicles. Section 2.1 describes several routing algorithms with different approaches. In Section 2.2, graph theoretical background for finding the optimal location for a charging station is reviewed.

2.1. Routing algorithms

Routing electric vehicles can be done with different incentives. Minimising the duration or length in kilometers of a trip is what is usually done in navigation systems. Modern navigation systems also take into account traffic congestion. The paper by De Weerdt et al. [2] compares three different algorithms that try to determine the shortest route in terms of time for a single vehicle. These three algorithms are MIN, short for minimising the journey time, a randomised variant of MIN that uses a logistic distribution (LOGIT), and the intention-aware routing system (IARS). Two of these three stragtegies, MIN and LOGIT, do not care about congestion, whereas their newly proposed algorithm, IARS, does take into account congestion at charging stations. The routing model behind these three algorithms is the same, and will be discussed in Chapter 3, as this is used in this thesis as well. MIN and LOGIT both use this model without any information about what other cars are doing. Therefore they cannot take into account congestion when determining the route. The MIN algorithm just determines the shortest route from origin to destination. LOGIT, on the other hand, adds some randomness to the MIN algorithm, in order to make sure not all cars go to the same charging stations. Although LOGIT already performs a bit better than MIN, the IARS algorithm is much better at reducing congestion at charging stations. This algorithm namely takes into account the intentions of each vehicle when determining the route. Each vehicle subscribes to a central system and sends their initial route to that system. The system then computes the waiting times at each charging station and returns these to the vehicles. Also, whenever any waiting time is updated, each vehicle is notified. The individual vehicles then use these waiting times to recompute their route, such that they avoid crowded charging stations.

An alternative approach to IARS, is to calculate the optimal route without stations before-hand, and dynamically update the route if charging is necessary. This is done by Ghorpade et al. [5]. They have decomposed the electric vehicle routing problem into two problems, the optimal vehicle routing problem, which already has suitable algorithms, and a dynamic charging decision problem. The latter is their main contribution, and is the problem of deciding whether a vehicle has to go to a charging station while it is driving along the predetermined optimal route. This decision is made based on the predicted battery level at the next location, and if this is below a certain minimum, the vehicle is routed to the nearest charging station or the station that has the smallest deviation from the route, depending on whether that station can be reached.

Another goal for routing electric vehicles could be to minimise energy consumption. Zhang et al. [3] have developed a routing algorithm that tries to minimise the total amount of kWh spent. The problem that this research solves considers a set of electric vehicles that start at a depot, visit a set of customers, and return to the depot. On their way, these vehicles might need to charge several times. The proposed optimisation model considers, among other things, the weight and capacity of the vehicles, the different driving speeds at different arcs in the graph and the battery capacity of the vehicles. An ant colony algorithm is used to determine which vehicles to use and their optimal routes.

Another paper, by Chen et al.[6], proposes an optimisation model for a pickup and delivery system with a fleet of electric vehicles. This model calculates waiting times at charging stations and tries to minimise the passengers' travel distance, while maximising the energy efficiency.

2.2. Placing charging stations

In the previous section, we have seen the perspective of a vehicle, which can also be seen as the buyer. In this section, we consider the perspective of the seller, or the charging stations. When building a new charging stations, there are namely also a lot of things that influence the total traffic flow in the whole road network. Factors as the location, price or capacity of a charging station influence the amount of congestion in the road network or the profit of the station itself.

In graph theory, centrality measures can often be used to extract some information on locations in a graph. There are a lot of such measures, such as betweenness centrality or degree centrality [7]. It has been found that betweenness centrality can play a role in finding congestion in road networks, i.e. nodes with a higher betweenness centrality are often more congested [8]. The betweenness centrality of a node reflects the number of shortest paths going through that node and is given by:

$$g(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

where σ_{st} is the total number of shortest paths between s and t, and $\sigma_{st}(v)$ is the number of shortest paths between s and t going through v.

Several attempts in finding locations for building charging stations have been made. Sun et al. [9] have proposed a model that places both slow charging stations and fast recharging stations with the goal of capturing as much traffic flow as possible. The slow stations are built on nodes and intended for short distance commuters, whereas the fast stations are built along paths and intended for long distance commuters.

Another research, by Uslu et al. [10] proposes a mixed integer programming model for deciding where to build electric bus charging stations and their capacity, with the goal to ensure that all roads in the network can be reached by buses.

3. Routing model

The model described by De Weerdt et al. [2] is used as the routing model when investigating the influence of a charging station's location on its profitability. In order to understand the effect of placing stations at certain locations, this routing model and the algorithms using this routing model should be understood. This chapter describes the mathematical formulation of the model (Section 3.1) and the policy function (Section 3.2), which determines the decisions of individual vehicles. Next to that, we explain two routing algorithms that are based on this model (Section 3.3).

3.1. Model

The model describes both the road network with the charging stations, as well as the state of each car. The domain for the model is described by $\langle V, E, T, P, S, C\rangle$, and is from the perspective of a single car.

The road network is represented as a graph with vertices $v \in V$ and edges $e = (v_i, v_j) \in E$. Both the roads and the charging stations are represented as edges, where a charging station is represented as a loop from a vertex to itself.

The other four variables in the domain of the model are used to describe the state of a car. The set $T = 1, ..., t_{max}$ is used to discretise the time. Each time point $t \in T$ is a moment at which a car can update its decisions on which stations to go to. Also, each $t \in T$ can represent the duration of driving over a road or charging a car, which is used by the function P. P is a probabilistic function indicating the driving or charging time for a certain edge. More specifically, $P(t|e, t_c)$ is the chance that driving over road e at time t_c, takes t time points (i.e. minutes).

The state of charge of a car is described by the set S. $S = 0, ..., s_{max}$ represents all possible states of charge, and intuitively this can be seen as a percentage of the battery that is left. The state of charge changes when a car drives over a road or goes to a charging station. This change is determined by the function C, where $C(e) \in S$ gives the charging cost of driving over an edge e.

3.2. Routing policy

The decision of a vehicle is determined by a routing policy $\pi : V \times T \times S \to V$. The policy determines, given the current vertex, current time and charging state of a car, which vertex to drive to. This routing policy determines its decision by maximizing the expected utility.

The expected utility is the expected reward of making a certain decision at a certain moment. This function is defined recursively and given by

$$EU(e_c = (v_c, w), t_c, s_c|\pi) =$$
$$\begin{cases} -\infty, & \text{if } s_c \leq 0 \\ \sum_{\Delta t \in T} P(\Delta t|e_c, t_c) \cdot U(t_c + \Delta t, s') & \text{if } w = v_{dest} \\ \sum_{\Delta t \in T} P(\Delta t|e_c, t_c) \cdot & \\ EU((w, \pi(w, t_c + \Delta t, s'), t_c + \Delta t, s'|\pi) & \text{otherwise} \end{cases}$$

Here s' is the state of charge after taking the edge e, and $U(t_c, s_c)$ represents the utility of arriving at time t_c with charge s_c. This utility function is given by

$$U(t_c, s_c) = \begin{cases} \infty, & \text{if } s_c \leq 0 \\ -t_c, & \text{otherwise} \end{cases}$$

In other words, the goal of the utility function is to minimise the travel time.

As can be seen in the equation, the expected utility contains three cases. In the first case, the utility is set to $-\infty$ if the current charge is smaller or equal to zero. This represents the fact that the car cannot drive any further due to an empty battery. The second case represents the expected utility when the considered edge gets the vehicle to its desired destination. For this edge, we consider all possible travel times Δt, and multiply the probability that taking that edge takes Δt time units with the utility of arriving after Δt time units. By taking the sum of all these values, we have the expected utility for taking the specified edge. In all other cases, the same principle is applied as in the second case, but the utility is replaced by the expected utility, which defines the recursive relation.

3.3. Algorithms

In this research, we consider two of the algorithms used by De Weerdt et al. [2]. Both algorithms work with the same routing model and are only different in the way in which they handle waiting times.

The first algorithm we consider is the MAX algorithm. In the paper by De Weerdt et al. [2], this was called MIN as it minimises the expected journey time, but we keep referring to it as MAX (for maximising utility), since the utility function is changed when the model is extended with pricing. The MAX algorithm is not used for our results, but it is the basis for the IARS algorithm, so explaining this helps the understanding of the IARS algorithm.

The second algorithm is the IARS algorithm, which essentially works in the same way. The only difference is the fact that IARS dynamically updates the waiting times of each charging station based on the intentions of each vehicle. Each vehicle sends the stations they intend to use to a central system whenever they change their route. The central system then recalculates the waiting times at each station, and notifies each vehicle of the changes. This algorithm, thus, tries to account for congestion at charging stations.

The algorithm used to determine the route of a vehicle is a dynamic programming algorithm and keeps track of the expected utilities using a three-dimensional array EU. This array stores the expected utilities for a single vehicle at every vertex, for every possible state of charge, at every possible point in time. Another three-dimensional array is used to keep track of the best node to go to, given a vertex, state of charge and point in time. In other words, this array stores the policy π of the vehicle.

At the start of a simulation, all values for these two arrays are computed. This is done in the following manner. Initially, each value in the expected utility array is set to $-\infty$, the penalty for being out of charge, and the policy array consists of undefined values, as nothing is known about the policy. The first step for the algorithm is to loop over all possible time points, from the latest possible time point to the earliest time point in the simulation. Then, for each time point, it loops over all possible charging states. Now, given a time point t and a state of charge s, the utility for arriving at the desired destination v_{dest}, given by $EU(v_{dest}, s, t)$, is calculated and stored. Using the same t and s, all possible edges in the road network are considered. For each edge that is traversable with the current state of charge s, the expected utility of taking that edge is calculated by multiplying the probabilities for the different durations of that edge by the values stored in the expected utility array at the resulting vertex, state of charge and time point. So, if we have an edge e from v_{from} to v_{to} with $C(e) = 1$ and possible durations 1 with a chance of 10% and 2 with a chance of 90%, the expected utility becomes the following:

$$EU(v_{from}, s, t) = \max\{EU(v_{from}, s, t), 0.1 \cdot EU(v_{to}, s-1, t+1) + 0.9 \cdot EU(v_{to}, s-1, t+2)\}$$

That is, if the expected utility of taking edge e is bigger than the value stored at $EU(v_{from}, s, t)$, the algorithm updates the stored value to the newly calculated expected utility. The policy is then also updated such that

$$\pi(v_{from}, s, t) = e.$$

By repeating this procedure for every possible state of charge and each time point, we obtain all expected utility values and the policy of a vehicle. It is important to loop backwards over all time points, since the expected utility is computed using time points in the future.

The above procedure shows the exact working of the MAX algorithm. The IARS algorithm uses this procedure but adds another element to it. The above procedure is namely only calculated at the start of the algorithm, so influences of other vehicles in the network are not taken into account. For the IARS algorithm, however, the routes of other vehicles are essential for the performance of the algorithm. Therefore, the computation of the expected utility array and the policy array is done multiple times. Every time a vehicle arrives at a new vertex, all vehicles that arrive at a vertex at the same time point recompute their expected utilities and their policy in the same way as before. The only difference here is that the charging edges now have a computed waiting time. Therefore, the new policy is based upon the queues that occur at charging stations. The waiting times are stored as a distribution per station per time point, where each distribution is based on the number of vehicles arriving at that station at that time point. This number of vehicles is determined using the intentions of each vehicle, which are shared with the system in IARS.

4. Methodology

This chapter aims to describe what models we used to obtain the results to our research questions. In Section 4.1, the pricing extension to the routing model by De Weerdt et al. [2] is presented. Next to that, the optimisation model used to obtain the most profitable location is developed in Section 4.2

4.1. Pricing extension

In this section, we extend the routing model by De Weerdt et al. [2] (described in Chapter 3), such that it can handle variable pricing per station. This extended model can then be used to determine the effect of charging different prices based on the location of a charging station. In order to extend the model with a pricing scheme, we have to extend the domain of the model (Subsection 4.1.1), and we have to update the utility function (Subsection 4.1.2).

4.1.1. Pricing model

The original domain of the model was described by $\langle V, E, T, P, S, C\rangle$. In order to extend the model with a pricing scheme, we need to extend this domain.

First, we add the set $M = 1, 2, ..., m_{max}$, which represents the set of possible values of amount of money spent. The state of a vehicle is, therefore, also extended with a variable m_c indicating the amount of money the vehicle has spent at that moment.

In order to update the money spent of a vehicle when it charges, a function F is added to the domain. This function represents the cost of fully charging a vehicle at a certain station and is given by

$$F(e) = \begin{cases} 0, & \text{for all } e \in E_{roads} \\ m_e, & \text{for all } e \in E_{stations} \end{cases}$$

where $m_e \in M$ is a fixed price which is determined by the station. Note that m_e can easily be replaced by a function of time to represent a dynamic pricing system.

4.1.2. Utility function

As the domain of the model has been extended, the routing policy should be updated accordingly. With the addition of the amount of money spent, the routing policy now becomes $\pi : V \times T \times S \times M \to V$. The current amount of money spent is added as a factor which plays a role in determining the next edge to take.

This policy still maximises the expected utility, which does not change, apart from the fact that the current amount of money spent is passed along. This, thus, results in the following expected utility function.

$$EU(e_c = (v_c, w), t_c, s_c, m_c|\pi) =$$
$$\begin{cases} -\infty, & \text{if } s_c \leq 0 \\ \sum_{\Delta t \in T} P(\Delta t|e_c, t_c) \cdot U(t_c + \Delta t, s', m') & \text{if } w = v_{dest} \\ \sum_{\Delta t \in T} P(\Delta t|e_c, t_c) \cdot & \\ EU((w, \pi(w, t_c + \Delta t, s', m'), t_c + \Delta t, s', m'|\pi) & \text{otherwise} \end{cases}$$

Just like with s', m' is the amount of money spent after taking the edge e.

The biggest differences occur in the utility function itself. Here, we want to model the trade-off between the time spent to reach the destination and the money spent on charging. Three approaches are considered on how to do this. The first approach is that every vehicle determines its budget for the journey, and time is minimised as long as you stay within that budget. The second approach uses a time deadline, and as long as the arrival is earlier than the deadline, the cost is minimised. The third approach models a trade-off between time and money, so every vehicle has a certain preference on whether they give more importance to time or to price and the utility is based upon that. This last option seems to fit the model best, as the other two approaches have significant drawbacks. The first approach, for instance, can spend all of the budget for a very small time benefit, whereas for the second approach it is hard to determine what to do when it is not possible to meet the deadline.

As a result of choosing the third approach, the time/money trade-off is modelled using the following utility function which has to be maximised:

$$U(t_c, s_c, m_c) =$$
$$\begin{cases} -\infty, & \text{if } s_c < 0 \\ \gamma * \frac{T_{max} - t_c}{T_{max} - T_{min}} + (1 - \gamma) * \frac{M_{max} - m_c}{M_{max} - M_{min}} & \text{otherwise} \end{cases}$$

In this function, γ represents the weights for price and travel time. γ equal to one indicates that the driver only cares about time, whereas γ equal to zero indicates a full preference for price. M_{min} and M_{max} are the minimum and maximum possible cost of the journey, respectively. The maximum possible cost could also be replaced by opportunity costs to obtain a more realistic value. T_{min} and T_{max} are the minimum possible journey time and the maximum arrival time, respectively. Both M_{min} and M_{max}, as well as T_{min} and T_{max} are normalisation factors. By normalising both the time and money spent, they are of equal importance. One could, therefore, see γ as a percentage of how much one favours time over price.

4.2. Determining the most profitable location for a new station

The last research question has the aim of finding the best location in a road network to build a new charging station. This section proposes an optimisation model with which we can find such a location if all vehicles use the IARS algorithm [2]. Appendix A shows an implementation of the model in Python. Within this model, we represent the graph of the road network with two sets. V is the set of all vertices, and E is the set of all edges. A subset $C \subseteq V$ consists of all vertices that are already occupied by a charging station. Furthermore, we have the set of vehicles represented by F, and the set of discrete time points given by T.

For each vehicle $f \in F$, we know the origin of its route o_f, the destination of its route d_f, and the maximum value of the state of charge z^f_{max}. For each edge $(i, j) \in E$, the travel time t_{ij} and the charge loss c_{ij} are known. Furthermore, for each vertex $i \in V$, we know the price p_i and the amount of vehicles that can charge at the same time q_i. These values also need to be present for vertices which are not a station, as a station can still be built there. We assume the charging time t_{charge} to be equal for every vehicle, as this is also done by the IARS algorithm. Finally, we use the values T_{min}, T_{max}, M_{min} and M_{max}, as described before, and the value M is used to represent a large positive number.

In order to determine the routes of every single vehicle $f \in F$, we introduce binary variables x^f_{ij}, which are one when vehicle f has edge $(i, j) \in E$ on its route, and zero otherwise. Next to that, for each vehicle, we keep track of its arrival time at every vertex with t^f_i, its waiting time at every vertex with $t^f_{wait,i}$, and its state of charge at every vertex with z^f_i. Moreover, we also use the binary variables h^f_{it} and b^f_{it}, which are one when vehicle f starts charging at station i at time point $t \in T$ and when vehicle f is busy charging at station i at time point t, respectively, and zero otherwise. The final decision variable is the binary variable s_i, which is one when the newly placed station is located at vertex i, and zero otherwise.

Route

As every vehicle needs to follow a route from its origin to its destination, a couple of constraints are formulated to ensure that every vehicle follows such a route. To model these constraints, we followed the approach of Chen et al.[6] Each route starts at the origin vertex, so from the origin vertex there should be exactly one outgoing edge. This is ensured by the following constraint.

$$\sum_{j \in V \setminus \{o_f\}} x^f_{ij} = 1, \qquad i \in \{o_f\}, \forall f \in F \tag{1}$$

The opposite of this, the destination vertex has exactly one incoming edge, is modelled with the next constraint.

$$\sum_{i \in V \setminus \{d_f\}} x^f_{ij} = 1, \qquad j \in \{d_f\}, \forall f \in F \tag{2}$$

For all other vertices, we should have that the number of incoming edges is equal to the number of outgoing edges, since this ensures that we will get a closed path from the origin to the destination. The following constraint ensures this.

$$\sum_{i \in V} x_{ij}^f = \sum_{k \in V} x_{jk}^f, \qquad \forall j \in V \setminus (\{o_f\} \cup \{d_f\}), \forall f \in F \tag{3}$$

With the above three constraints, every vehicle is guaranteed to have a route from its origin to its destination. Similar to the routing model described in Chapter 3, charging at a charging station is represented by taking the edge from a vertex to itself. As this adds up to both the incoming and outgoing edges, this is still possible in the formulation of the constraints, except for the origin and destination vertex (this is not necessary as a vehicle has maximum charge at its origin, and does not need to charge anymore at its destination). Also, visiting a station twice is possible, but this will not be expected to happen since the objective function will punish longer routes.

Time progression

To find out the duration of a trip of a vehicle, we need to keep track of the arrival and waiting times at every vertex for each vehicle. This is done with three sets of constraints, which we modelled according to the approach of Chen et al.[6]

$$t_{wait,i}^f \leq M x_{ii}^f, \qquad \forall i \in V, \forall f \in F \tag{4}$$

The above constraints enforce that a vehicle can only wait at a station if it also charges there. Next to that, we need to update the arrival times of vehicles based on the route lengths, waiting times and charging time. We split this up into two cases. The first constraints account for the situation where a vehicle travels from vertex i to vertex j without having charged at vertex i. In this case, the arrival time at vertex j should be the arrival time at vertex i plus the travel time between i and j. This is modelled with the following set of constraints.

$$-M(1 - x_{ij}^f + x_{ii}^f) \leq t_i^f + t_{ij} - t_j^f \leq M(1 - x_{ij}^f + x_{ii}^f), \qquad \forall i, j \in V, \forall f \in F \tag{5}$$

The other situation, in which the vehicle does charge at station i, also adds the waiting time and the charging time to the arrival time at station j. This results in the following.

$$-M(2 - x_{ij}^f - x_{ii}^f) \leq t_i^f + t_{wait,i}^f + t_{charge} + t_{ij} - t_j^f \leq M(2 - x_{ij}^f - x_{ii}^f), \qquad \forall i, j \in V, \forall f \in F \tag{6}$$

With these three sets of constraints, we are able to determine the arrival time of every vehicle at its destination.

State of charge

Similar to the time progression, we need to keep track of the state of charge of every vehicle. This is necessary in order to make sure no vehicle ends up being stuck with no charge left. In order to model this, we need two constraints. One for when the vehicle does not charge, and one for when the vehicle does charge. If the vehicle does not charge, the charge is only updated with the charge loss of taking the edge (i, j) and we get the following.

$$-M(1-x_{ij}^f+x_{ii}^f) \leq z_i^f-c_{ij}-z_j^f \leq M(1-x_{ij}^f+x_{ii}^f), \qquad \forall i, j \in V \text{ with } i \neq j, \forall f \in F \quad (7)$$

If the vehicle does charge, it just resets to its maximum charge and only loses the charge of taking the edge (i, j). This results into the next set of constraints.

$$-M(2-x_{ii}^f-x_{ij}^f) \leq z_{max}^f-c_{ij}-z_j^f \leq M(2-x_{ii}^f-x_{ij}^f), \qquad \forall i, j \in V \text{ with } i \neq j, \forall f \in F \quad (8)$$

In the implementation, we assume that each vehicle is fully charged at its origin.

Charging capacity

Since each charging station has limited capacity, the model needs to account for this. In order to do this, we followed the approach of Van Essen et al.[11] We first find out when each vehicle starts charging at a certain vertex, which is given by h_{it}^f. Note that it is never beneficial to charge twice at the same station, so we make sure this is not possible with the following constraints.

$$\sum_{t \in T} h_{it}^f \leq 1, \qquad \forall i \in V, \forall f \in F \quad (9)$$

Since we now have that h_{it}^f is only 1 at one point in time, we can make sure that each vehicle only starts charging after having arrived and having waited. This is done by the constraints below.

$$\sum_{t \in T} h_{it}^f \cdot t - M(1-x_{ii}^f) \leq t_i^f + t_{wait,i}^f \leq \sum_{t \in T} h_{it}^f \cdot t + M(1-x_{ii}^f), \qquad \forall i \in V, \forall f \in F \quad (10)$$

Now we know when each vehicle starts charging at every vertex, we can find out whether a vehicle is still charging at a certain time point using the charging time. This is necessary to find out how many vehicles are charging at the same time at the same station. The following constraints set the value b_{it}^f to 1 if vehicle f is busy charging at station i at time point t.

$$b_{it}^f = \sum_{\hat{t}=t-t_{charge}+1}^{t} h_{i\hat{t}}^f, \qquad \forall i \in V, \forall f \in F, \forall t \in T \quad (11)$$

We can now enforce that at any point in time no more than q_i vehicles (the capacity of the station) are charging at the same time with the next set of constraints.

$$\sum_{f \in F} b_{it}^f \leq q_i, \qquad \forall i \in V, \forall t \in T \quad (12)$$

New station

Since the goal of this model is to find the best location for a new station, we also need to include the possibility to add a station in our model. In order to do this we use the binary decision variable s_i. Since we only want to add one station, we apply the following constraint to ensure this.

$$\sum_{i \in V \setminus C} s_i \leq 1 \tag{13}$$

Note that this can be easily extended to adding more stations at the same time, by changing the 1 into a parameter.

We also need to make sure that a vehicle can only charge at a station if it is selected as the new station. The following constraints enforce this.

$$x_{ii} \leq s_i, \qquad \forall i \in V \setminus C \tag{14}$$

Objective function

The objective of our model is directly related to the objective of each vehicle in the IARS model [2]. Since each vehicle wants to maximise their own utility function, our model maximises the sum of the utility functions. This results into the following objective function.

$$\max \qquad \sum_{f \in F} \left(\gamma \frac{T_{max} - t^f_{d_f}}{T_{max} - T_{min}} + (1-\gamma) \frac{M_{max} - \sum_{i \in V} x^f_{ii} p_i}{M_{max} - M_{min}} \right) \tag{15}$$

By applying this objective, the system starts to place a charging station at a certain vertex, such that the total utility of all vehicles is optimal. We can then retrieve this location from the results, which represents the best location to place a new station. Note that this objective function maximises the sum of all utilities, whereas in reality each individual vehicle maximises its own utility.

5. Results

This chapter shows the results of running the models presented in the previous two chapters on two different graph topologies. The first section (Section 5.1) shows the results of running the IARS algorithm [2] on two different topologies. In Section 5.2, the IARS with the pricing extension is run, whereas Section 5.3 shows the results of running the optimisation model presented in Section 4.2.

5.1. Most visited location

In order to answer the first research question, which is about determining the most frequently visited station with equal pricing, we run the IARS algorithm [2] several times, on two different types of graph topologies. For each topology, we experiment with different parameters, such as the length of the edges. By doing this, we can determine whether certain locations in a network are visited more often than others. We also calculate the betweenness centrality of all stations in order to find out if there is a relation between the centrality and the number of visits.

Bottleneck topology

The first graph topology we investigate is the bottleneck topology. This is a very simple graph with one origin, one destination, and one column with stations. A visualisation of this graph with four stations in the column, can be seen in Figure 1 below.

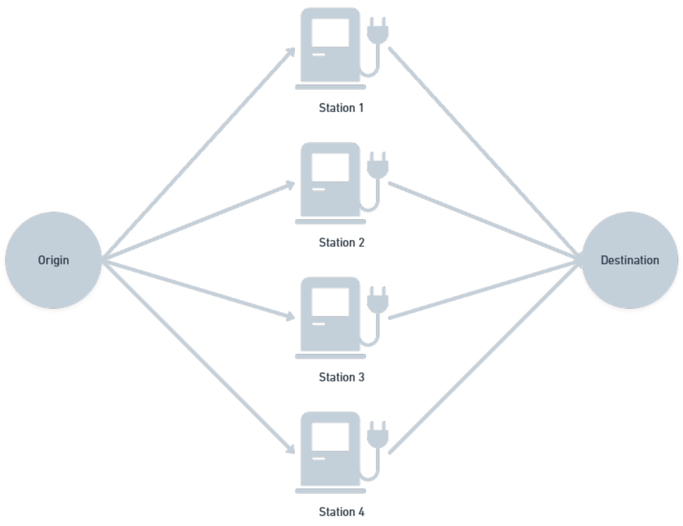

Figure 1: Bottleneck graph with four charging stations

When running the algorithm, each vehicle thus has to make a choice between the four stations, which might be on longer or shorter routes.

For our first run, we consider that all edges take 1 unit of time to travel along them, so each route from start to finish takes 2 units of driving time. We made sure every car has to charge on its route, by setting the charging capacity of a vehicle to 1. The amount of vehicles in this run is 500 and each station has a capacity of 2 vehicles that can charge at the same time. As the edge lengths are all 1, each node has a betweenness centrality of $\frac{1}{4}$. Table 1 shows the results of running the algorithm with the described setup.

Table 1: Station visits for bottleneck graph with equal edge lengths

Station	Betweenness centrality	Number of visits
1	0.25	125
2	0.25	125
3	0.25	125
4	0.25	125

We observe that each station is visited 125 times, so the vehicles equally spread out over all stations in order to reduce waiting times. This is a logical result, as all paths are equally long, so the driving time does not depend on which charging station the vehicle chooses. Also, the betweenness centrality is equal for every station, so in this situation this corresponds to the distribution over the stations.

It is more interesting to see what happens when some stations are on longer routes than others. For the bottleneck graph, we can investigate this by having different edge lengths. For the next run, we again have 500 vehicles and a capacity of 2 vehicles per charging station. This time, however, we have different edge lengths for each edge, which are random lengths between 1 and 10 time units. Table 2 shows the number of visits per station for a single run. Moreover, the route length indicates the sum of the driving times if you take the two roads that the vehicle takes if it charges at the corresponding station.

Table 2: Station visits for bottleneck graph with different edge lengths

Station	Route length	Betweenness centrality	Visits (IARS)
1	10	0	138
2	18	0	55
3	9	1	169
4	10	0	138

As there is only one shortest path from origin to destination, station 3 has a betweenness of 1, while the other stations have a betweenness of 0. However, not all vehicles go to the station with the highest betweenness. The reason for this is the fact that congestion will occur at station 3 if all vehicles go to that station. So in order to reduce their waiting time, and thus their total travel time, several vehicles also charge at another station. The influence of the waiting times thus reduces the number of visits for the station on the shorter route.

Although the waiting times make sure that the vehicles spread out over all stations, there does seem to be a relation between the route length and the number of visits. We can investigate this effect further by running the algorithm multiple times with different edge lengths. Again, the edges are taken randomly between 1 and 10 time units, and all other parameters stay the same. Figure 2 shows the results of 5 runs with different edge lengths. On the x-axis the route length through a certain station divided by the sum of all route lengths is displayed. The y-axis shows the number of visits of that station divided by the total number of visits for that simulation is displayed. The total number of visits is always 500 as every car has to charge to reach the goal.

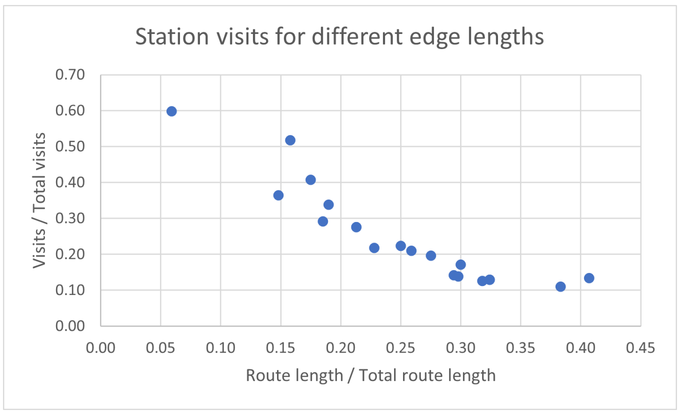

Figure 2: Results for simulations on bottleneck graph with different edge lengths

The graph in Figure 2 shows a clear downward trend, which means that a station on a longer route is visited less often. Even throughout different simulations, this trend is still clearly visible. This is as expected, but it does show that although charging capacity is a major blocking factor, with 500 cars and only 2 charging spots per station, the route length still has a big influence on the number of visits per station.

One more thing we can investigate for the bottleneck graph is the influence of the charging capacity parameter. We use the same edge lengths and parameters as for the results in Table 2, but we change the charging capacity parameter. This change is made for each station, so in every run all stations have the same capacity. Figure 3 shows the results for 7 runs with different charging capacities. The x-axis shows the capacity at each station, whereas the y-axis shows the number of visits per station.

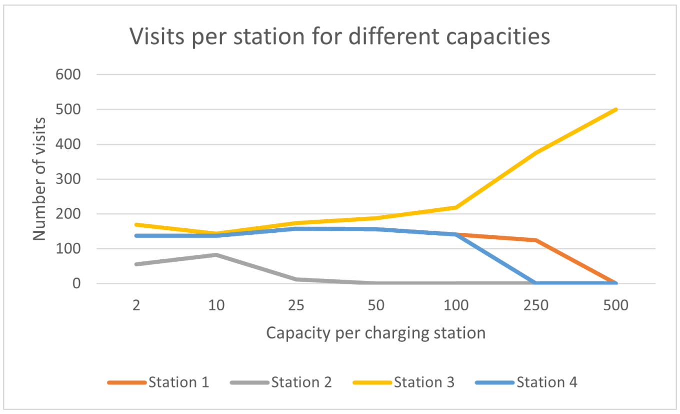

Figure 3: Results for simulations on bottleneck graph with different station capacities

These results clearly support the fact that waiting time has a large influence on the decisions taken by the vehicles using IARS. The general trend in these results is that with a bigger capacity, the stations on shorter routes get visited more often, until only the station on the shortest route is being visited. This indicates that stations on a shorter route benefit from expanding their capacity, whereas for stations on a longer route, this would only increase their costs.

Grid topology

The second topology, the grid topology, contains multiple origins and multiple destinations. Also several columns of stations seperate the column of origins and destinations. Each node connects with the adjacent nodes in the next column. In Figure 4, a grid graph with two columns of four stations is shown.

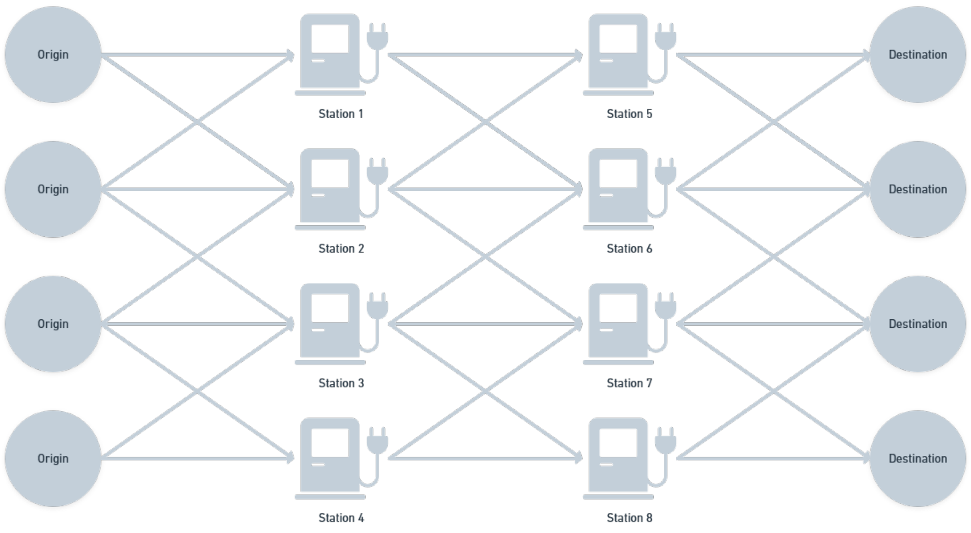

Figure 4: Grid graph with two columns of four stations

What can be seen from this graph, is that the stations in the middle rows have a higher number of routes going from any origin to any destination. Therefore, it is interesting to see whether these stations will have more visits than the stations in the top or bottom row.

We ran the algorithm ten times with different edge lengths, taken random between 1 and 10. For each run, all other factors stay the same, and are as before. So, the number of cars is 500 and the charging capacity of a station is 2. Each vehicle starts driving at the same time, and all vehicles are equally divided over the origins and the destinations, so one quarter of the vehicles starts at each origin. For each run, we determine the betweenness centrality of each station and compare this to the number of visits. As each run gave a similar result, we show the result of a single run in the following graph (Figure 5).

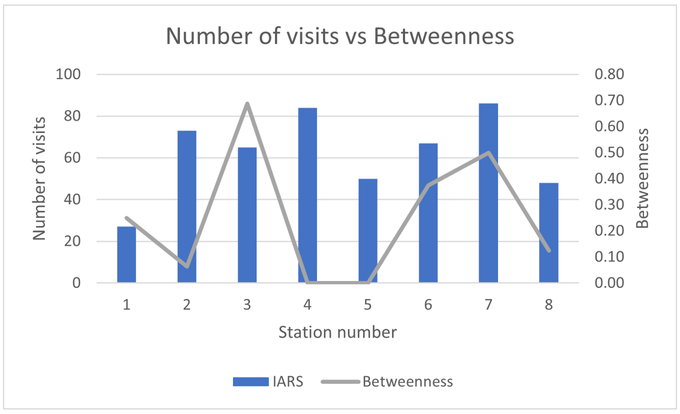

Figure 5: Comparison between the number of visits of a station and its betweenness

From these results, it becomes clear that there is no clear relation between the number of visits and the betweenness centrality. No clear pattern can be obtained from these results, and the reason for this could be that the large amount of possibilities cannot be captured by the betweenness centrality, as too many factors play a role. As betweenness centrality only looks at the number of shortest paths going through a station, factors as the charging capacities and waiting time are not taken into account. Also, the lengths of the routes that are an alternative to the shortest route for a vehicle might play a role in whether a certain station is visited or not.

5.2. Effect of variable pricing

With the newly added pricing model (Section 4.2), we can determine the effect of stations being able to charge different prices. A station in a better location might decide to charge a higher price to increase its profit. The goal here is to find out for which locations this could be successful.

Again we run the algorithm for the bottleneck and grid topologies, and calculate the profits obtained for each station. In each of the runs, one station has a different price from all other stations, which charge the same price. This station is either the station which originally had the highest number of visits or the station which originally had the lowest number of visits. In case it is the one with the highest number of visits, we increase its price, and we decrease it otherwise. Furthermore, each vehicle has $\lambda = 0.5$, so they equally value money and time.

By comparing the results to those in the previous section (Section 5.1), we can determine whether it is beneficial for a certain station to start charging a higher or lower price.

Bottleneck topology

Again, we first run the algorithm on the bottleneck topology. We use the exact same parameters and edge lengths as for the results in Table 2. So, we have 500 vehicles and each station has a capacity of 2. The only change, is that each station now charges a price of 50, instead of not charging price at all. As station 3 was the station that had the highest number of visits, we increase its price to see the effect on the number of visits. Figure 6 shows the result of these increases.

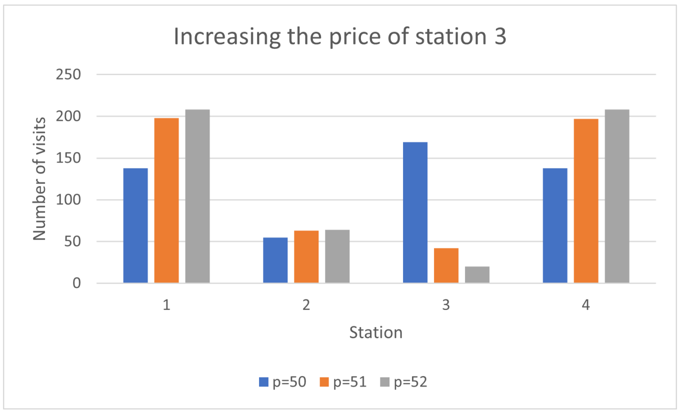

Figure 6: Increasing the price of the station on the shortest route

It can be seen that only a small increase in price, already leads to a strong decrease in number of visits. This might be due to the fact that the travel time does not increase that much, as the vehicles can spread over three other stations. It is also interesting to see that the station with the lowest number of visits profits the least from the increase in price by station 3.

Apart from increasing the price of the most profitable station, we can also decrease the price of the least profitable station. Figure 7 shows the result of decreasing the price.

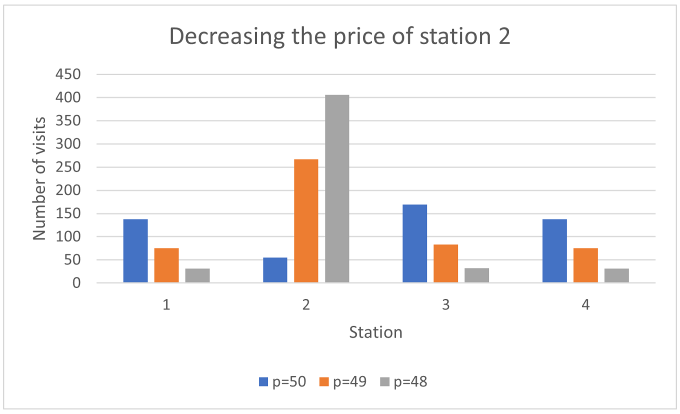

Figure 7: Decreasing the price of the station on the longest route

Again, we see a strong increase in number of visits for a small decrease in the price. So, for a station on a long route, it can be beneficial to lower the price to generate a higher profit, while a station on a short route does not benefit as much from setting a higher price.

Grid topology

For the grid topology, we do the same as for the bottleneck topology. So, we have one run in which we increase the price of the most profitable station, and one run in which we decrease the price of the least profitable station. As IARS with pricing has a longer run time, we reduce the number of vehicles to 100. The rest of the parameters stay the same. Also, we use a graph with random edge lengths, but of course the same graph is used when we change the price. Figure 8 shows the results of increasing the price of the most profitable station, which was station 6.

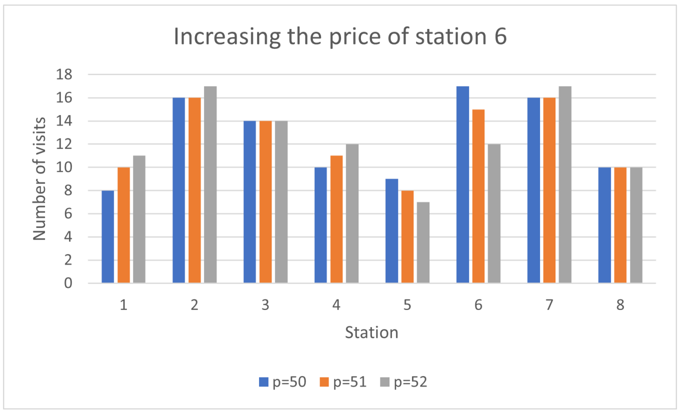

Figure 8: Increasing the price of the station with the highest number of visits

In comparison to the bottleneck topology, we see that increasing the price has a smaller effect on the number of visits. However, there is still a significant decrease. The reason for the smaller decrease could be that the alternative options for the vehicles charging at station 6 are significantly worse, but we cannot be sure of this.

Again, we also investigate what happens if we decrease the price of the least profitable station. The results of this are shown in Figure 9.

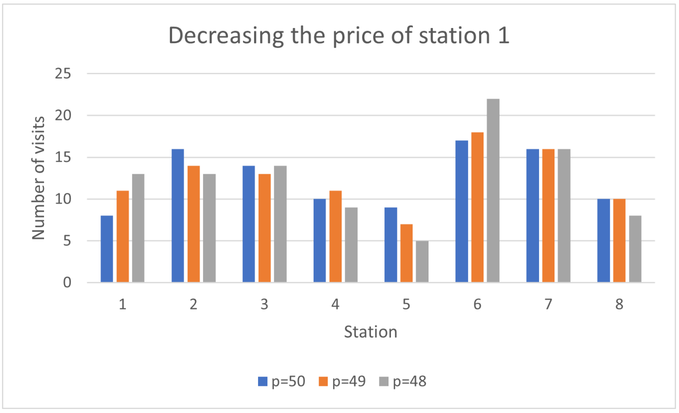

Figure 9: Decreasing the price of the station with the lowest number of visits

These results show a similar pattern as the previous figure (Figure 8). The increase in number of visits due to the lower price, is namely not as strong as for the bottleneck graph. What is also interesting, is the fact that some stations that do not change their price, also get a higher number of visits. This might be caused by the fact that different vehicles have different origin-destination pairs, which makes some stations unreachable by certain vehicles. Thus, when a change occurs at a certain station, this influences all other stations.

5.3. Optimal location for a new station

This section shows the results of running the optimisation model that finds the best location to build a new station, which has been described in Section 4.4. We then compare these results to simulations with the IARS model [2]. By doing this, we investigate whether we have developed a suitable model for finding the best location for a new charging station if all vehicles use the Intention-Aware Routing System.

As the optimisation model can take quite some time to run on larger graphs, we set a time limit of 15 minutes. This might lead to a sub-optimal solution, but since the model also optimises the routes of every vehicle, a sub-optimal solution could also mean that the utility functions of every vehicle are not completely optimised, but the right station is still chosen. We also show the optimality gap which is output of CPLEX. This gap shows the percentage difference between the best solution found within the time limit and the best bound found within the time limit. Here, the best bound is the maximal value any integer solution could still have, based on the information the solver has already discovered during the branch-and-bound.

Apart from the time limit, there are a couple of parameters that remain constant throughout all simulations of the model in order to isolate the effect of changing the other parameters. First of all, the charging capacity of a station is one in every simulation. Also the time it takes to charge is one, as well as the charge loss for every edge in the graph.

Bottleneck topology

The first graph topology we test our optimisation model on, is again the bottleneck graph. This graph has one origin node, one destination node, and four nodes in between the origin and destination node. In all our simulations, we started with one station, and tried to find which of the other three locations was most suitable to build an extra station. The initial situation is shown in the figure below.

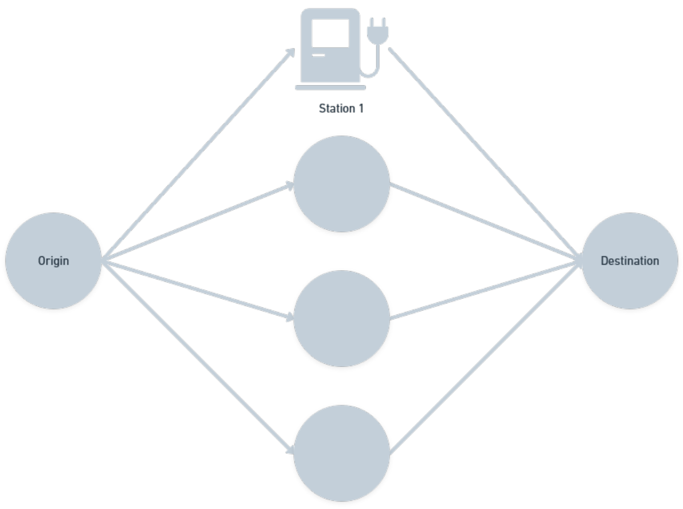

Figure 10: Bottleneck graph with one charging station and three possible locations for a new station

We first ran the model ten times with ten vehicles where price did not vary, but the length of the edges did. Each edge length is a random number between 1 and 10 time units. For each run, our optimisation model gave the best location to build the new station, and the average utility of the vehicles. We then used the same edge lengths in the IARS algorithm to find out which of the stations gets the most visits. This means that we ran the IARS algorithm three times, one time for each possible new station. We compared the optimal results of our model to the station that gave the best outcome in IARS, and found that in nine out of the ten simulations the chosen station by our model gave the best result in the IARS simulation. In the other run, all stations gave an equal result using IARS, and our model picked one of the three. Note that in only one run, the model reached the time limit and had an optimality gap of 3.20%. In all other runs the result was proven to be optimal.

We also compared the computed utilities of our model and the IARS algorithm in each simulation. Figure 11 shows the average utility per vehicle given by both the model and the IARS simulation.

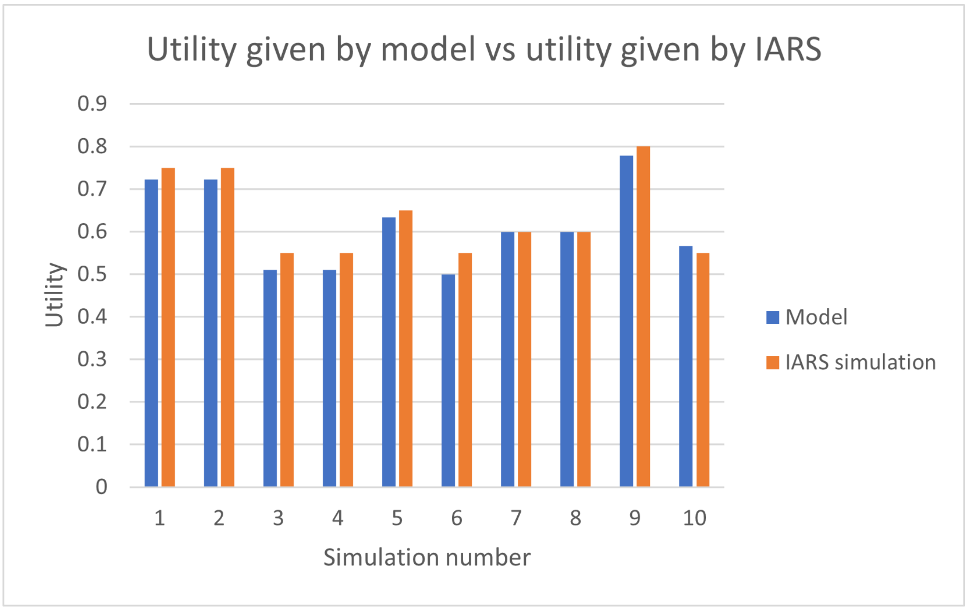

Figure 11: Comparison of the utilities given by the optimisation model and the simulations with IARS

Figure 11 shows that the utilities of both models lie quite close together. This is also supported by the fact that the maximum difference over all ten runs is 0.05, and the average difference is 0.024.

Next, we added random prices. Each station gets a random price between 1 and 5 that it charges, and we repeat the same experiment as above. We did 3 runs and each time, the model and IARS chose the same station as the one with the highest number of visits. In none of the runs, the model reached the time limit. The figure below shows the comparison of the utilities.

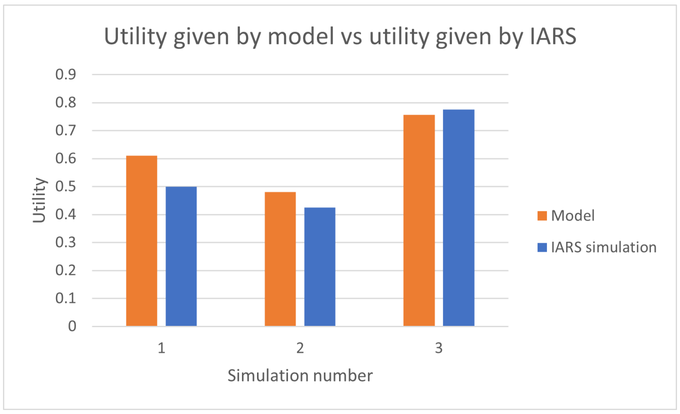

Figure 12: Comparison of the utilities given by the optimisation model and the simulations with IARS for the bottleneck graph with variable pricing

The utilities now show a bit more differences, but are still quite close together.

Grid topology

As in previous sections, we also run the model on the grid graph. This time, however, we use a smaller grid to account for the long runtime of the optimisation model. The graph is shown in Figure 13 and has two possible locations to open a new station.

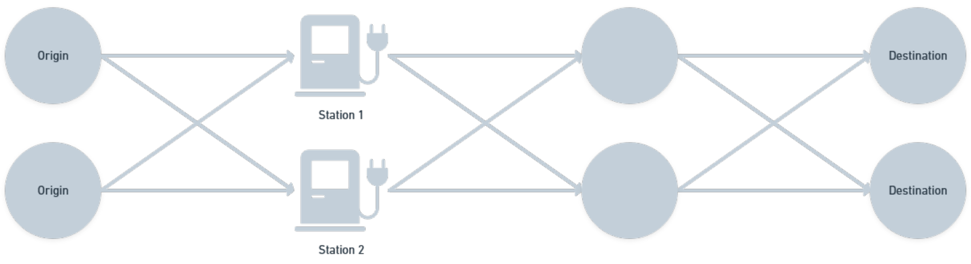

Figure 13: Grid graph with two charging stations and two possible locations for a new station

We ran the model three times with eight vehicles. For every origin-destination pair, there are exactly two vehicles that have that pair. We again used random prices and random edge lengths, and did the same comparison as for the bottleneck graph. In all runs, the model and IARS chose the same station as the optimal location. This time, however, all runs reached the time limit, and the optimality gaps varied between 4 and 30%. The following figure shows the comparison of the utililties.

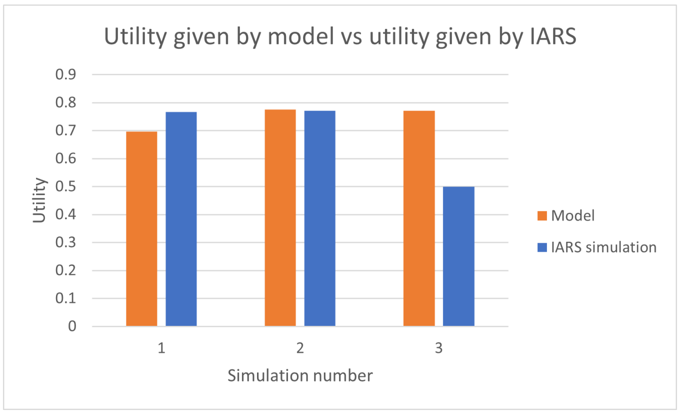

Figure 14: Comparison of the utilities given by the optimisation model and the simulations with IARS for the grid graph graph with variable pricing

We now see a greater difference between the utilities, especially for the third simulation, but this might have to do with the optimality gap. It is interesting to see that, althought the utilities are less accurate, the correct station is still chosen.

6. Conclusions and future work

The goal of this thesis was to investigate the effect of a charging station's location on its profitability, given that the vehicles use the IARS routing algorithm. This goal has been split up into several parts, which are investigating the effect of location without pricing, the addition of a pricing model, investigating the effects of pricing, and finding the best location to build a new station.

The analysis of the profitability of charging stations in the context of their location has led to several conclusions. First of all, in a situation without pricing, it became clear that for a small bottleneck graph there was a clear pattern, where a station on a longer route had a lower number of visits. However, for the grid graph, this pattern was not clear anymore. Although there were differences there was no relation between the number of visits and the betweenness centrality. When adding the pricing model, we have seen that a small change in price had a large influence. This was especially the case for the bottleneck graph. Thus, a station with a low number of visits can decrease its price to gain a lot of visitors.

The pricing extension also opens opportunities for new research. We have already seen the extension being used for the IARS algorithm, but it could also be used as an inspiration for alternative routing algorithms. It could be interesting to compare how different routing algorithms are influenced by changes in price. Also, many more relations could be investigated, such as the influence of the price/money trade-off parameter γ, or even the possibility to reduce congestion by setting different prices at different stations. Also, the pricing model could easily be improved to add the possibility of dynamic pricing based on time of day or current number of vehicles at a station.

The last contribution of this thesis is the optimisation model that finds the best location for a new charging station. From the runs with this model, we can conclude that the model accurately predicts the location that will obtain the highest number of visits if a station were to be built there, on small graphs. On the bottleneck topology, it also accurately predicted the utilities of the vehicles if a new station was built. Also, with varying prices, the model still predicted the station correctly.

For future work, the optimisation model could be very interesting. Especially improving the efficiency of the model could give interesting insights on larger graphs. Also, the model can easily be extended with dynamic pricing, and different strategies, such as reducing congestion, could be implemented by changing the objective function.

7. Discussion

This thesis has shown several results, of which the reliability is discussed in this chapter. The three main results that are discussed are the results of the runs in which pricing was not taken into account, the large effect of pricing, and the results produced by the optimisation model.

Without pricing, only a pattern was found for the bottleneck graph and not for the grid graph. However, it cannot be concluded already that such a relation does not exist. As the grid graph is larger than the bottleneck graph, the number of visits is affected by all other stations. This might be the cause that a clear pattern could not be observed. However, one could isolate each factor and find a pattern for every single factor.

When applying the formulated pricing extension, we found that price had a huge effect on the station choice, even though the drivers equally valued price and time. This indicates that a station's location has a lower influence than its price, but care needs to be taken when concluding this. Especially the influence of the normalisation factors in the pricing model has to be taken into account. Since these normalisation factors are based on the road network itself (i.e. M_{max} is the maximum possible price to pay on the route), a change in a station not on the route could change the utility of a vehicle. For instance, when a station that is not used by any of the vehicles increases its price significantly, the vehicles might start to value time a bit more due to the fact that the price factor is divided by a larger M_{max}. Therefore, the influence of these normalisation parameters should be researched more in order to verify the conclusions on the effect of price in this paper.

The optimisation model gave some promising results. The choices for the new station were always in line with the results the IARS algorithm gave. Also, the calculated utilities were fairly close for the bottleneck graph. For the grid graph, however, these utilities were not as close, but this is likely to be caused by the high optimality gap due to the model not finishing within the allocated run time. Also, since we were only able to run the model on simple and small graphs, one cannot say yet if such model would work in a practical situation. This would require more experiments on larger and more complex graph topologies.

References

[1] Global EV Outlook 2020. (2020). doi:10.1787/d394399e-en

[2] de Weerdt, M. M., Stein, S., Gerding, E. H., Robu, V., & Jennings, N. R. (2016). Intention-Aware Routing of Electric Vehicles. IEEE Transactions on Intelligent Transportation Systems, 17(5), 1472–1482. https://doi.org/10.1109/tits.2015.2506900

[3] Zhang, S., Gajpal, Y., Appadoo, S. S., & Abdulkader, M. M. S. (2018). Electric vehicle routing problem with recharging stations for minimizing energy consumption. International Journal of Production Economics, 203, 404–413. https://doi.org/10.1016/j.ijpe.2018.07.016

[4] Gerding, E., Stein, S., Robu, V., Zhao, D. and Jennings, N. R. (2013). Two-sided online markets for electric vehicle charging. Proc. 12th Int. Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2013), St. Paul, Minnesota, pp. 989-996.

[5] Ghorpade, T., Rangaraj, N., & Singh, T. (2020). Real-time charging decision with Stochastic battery performance for Commercial Electric Vehicles. Transportation Research Procedia, 47, 267-274. doi: 10.1016/j.trpro.2020.03.098

[6] Chen, T., Zhang, B., Pourbabak, H., Kavousi-Fard, A., & Su, W. (2018). Optimal Routing and Charging of an Electric Vehicle Fleet for High-Efficiency Dynamic Transit Systems. IEEE Transactions On Smart Grid, 9(4), 3563-3572. doi: 10.1109/tsg.2016.2635025

[7] Klein, D. J. (2010). Centrality measure in graphs. Journal of mathematical chemistry, 47(4), 1209-1223.

[8] Zhao, L., Lai, Y., Park, K., & Ye, N. (2005). Onset of traffic congestion in complex networks. Physical Review E, 71(2). doi: 10.1103/physreve.71.026125

[9] Sun, Z., Gao, W., Li, B., & Wang, L. (2020). Locating charging stations for electric vehicles. Transport Policy, 98, 48-54. doi: 10.1016/j.tranpol.2018.07.009

[10] Uslu, T., & Kaya, O. (2021). Location and capacity decisions for electric bus charging stations considering waiting times. Transportation Research Part D: Transport And Environment, 90, 102645. doi: 10.1016/j.trd.2020.102645

[11] van Essen, J., Hurink, J., Hartholt, W., & van den Akker, B. (2012). Decision support system for the operating room rescheduling problem. Health Care Management Science, 15(4), 355-372. doi: 10.1007/s10729-012-9202-2

A. Python code for solving optimisation model

```
from pulp import *
import random

# Input parameters

# Bottleneck graph

nVertices = 6
nVehicles = 10
C = [1]
t_travel = [[-1] * nVertices for i in range(nVertices)]
for i in range(nVertices):
    t_travel[i][i] = 0
t_travel[0][1] = 3
t_travel[0][2] = 7
t_travel[0][3] = 6
t_travel[0][4] = 7
t_travel[1][5] = 3
t_travel[2][5] = 7
t_travel[3][5] = 5
t_travel[4][5] = 7
o = [0] * nVehicles
d = [5] * nVehicles
c_cap = [1] * nVertices
c_l = [[-1] * nVertices for i in range(nVertices)]
for i in range(nVertices):
    c_l[i][i] = 0
c_l[0][1] = 1
c_l[0][2] = 1
c_l[0][3] = 1
c_l[0][4] = 1
c_l[1][5] = 1
c_l[2][5] = 1
c_l[3][5] = 1
c_l[4][5] = 1

t_charge = 1
z_max = [1] * nVehicles
p = [1, random.randint(1, 10), random.randint(1, 10), random.
   randint(1, 10), random.randint(1, 10), 1]
print(p)
```

```
T_min = 7
T_max = 16
M_min = min(p)
M_max = max(p)
gamma = 1
M = 10000

# Grid graph

# nVertices = 8
# nVehicles = 8
# C = [2, 3]
# t_travel = [[-1] * nVertices for i in range(nVertices)]
# for i in range(nVertices):
#     t_travel[i][i] = 0
# t_travel[0][2] = 4
# t_travel[0][3] = 5
# t_travel[1][2] = 3
# t_travel[1][3] = 4
# t_travel[2][4] = 4
# t_travel[2][5] = 3
# t_travel[3][4] = 3
# t_travel[3][5] = 1
# t_travel[4][6] = 3
# t_travel[4][7] = 2
# t_travel[5][6] = 3
# t_travel[5][7] = 5
#
# o = [0, 0, 0, 0, 1, 1, 1, 1]
# d = [6, 6, 7, 7, 6, 6, 7, 7]
#
# print(o)
# print(d)
# c_cap = [1] * nVertices
# c_l = [[-1] * nVertices for i in range(nVertices)]
# for i in range(nVertices):
#     c_l[i][i] = 0
# c_l[0][2] = 1
# c_l[0][3] = 1
# c_l[1][2] = 1
# c_l[1][3] = 1
# c_l[2][4] = 1
# c_l[2][5] = 1
# c_l[3][4] = 1
# c_l[3][5] = 1
# c_l[4][6] = 1
# c_l[4][7] = 1
# c_l[5][6] = 1
# c_l[5][7] = 1
#
# t_charge = 1
# z_max = [2] * nVehicles
# p = [0, 0, 1, 1, 2, 1, 0, 0]
# T_min = 9
# T_max = 15
# M_min = 0
# M_max = 2
# gamma = 1
# M = 10000

problem = LpProblem("Facility_Location", LpMaximize)

# Decision variables
x = LpVariable.dicts("route", (range(nVehicles), range(
    nVertices), range(nVertices)), 0, 1, LpInteger)
s = LpVariable.dicts("new_station", (range(nVertices)), 0, 1,
    LpInteger)
t_arrival = LpVariable.dicts("arrival_time", (range(nVehicles),
     range(nVertices)), 0, None, LpInteger)
t_wait = LpVariable.dicts("waiting_time", (range(nVehicles),
    range(nVertices)), 0, None, LpInteger)
z = LpVariable.dicts("state_of_charge", (range(nVehicles),
    range(nVertices)), 0, None, LpInteger)
c = LpVariable.dicts("charge_start", (range(nVehicles), range(
    nVertices), range(T_max)), 0, 1, LpInteger)
b = LpVariable.dicts("charge_current", (range(nVehicles), range
    (nVertices), range(T_max)), 0, 1, LpInteger)

# Objective function
problem += lpSum([gamma * (T_max - t_arrival[f][d[f]]) / (T_max
     - T_min) + (1 - gamma) * (M_max - lpSum([x[f][i][i] * p[i]
    for i in range(nVertices)])) / (M_max - M_min) for f in
    range(nVehicles)]), "Utility"

# Constraints:

# Making sure travel from i to j is only possible if there is
    an edge
```

```
for f in range(nVehicles):
    for i in range(nVertices):
        for j in range(nVertices):
            if t_travel[i][j] == -1 and i != j:
                problem += x[f][i][j] == 0

# Origin only has one outgoing edge
for f in range(nVehicles):
    problem += lpSum([x[f][o[f]][j] for j in range(nVertices)
       if j != o[f]]) == 1

# Destination only has one incoming edge
for f in range(nVehicles):
    problem += lpSum([x[f][i][d[f]] for i in range(nVertices)
       if i != d[f]]) == 1

# Every vertex has as many incoming edges as outgoing edges
for f in range(nVehicles):
    for j in range(nVertices):
        if j == o[f] or j == d[f]:
            continue
        problem += lpSum([x[f][i][j] for i in range(nVertices)
           ]) == lpSum([x[f][j][k] for k in range(nVertices)])

# A vehicle can only have a waiting time at a station if it
   charges there
for f in range(nVehicles):
    for i in range(nVertices):
        problem += t_wait[f][i] <= M * x[f][i][i]

# Constraints for time progression
for f in range(nVehicles):
    for i in range(nVertices):
        for j in range(nVertices):
            if i == j:
                continue
            problem += -M * (1 - x[f][i][j] + x[f][i][i]) <=
               t_arrival[f][i] + t_wait[f][i] + t_travel[i][j]
               - t_arrival[f][j]
            problem += t_arrival[f][i] + t_wait[f][i] +
               t_travel[i][j] - t_arrival[f][j] <= M * (1 - x[f
               ][i][j] + x[f][i][i])
            problem += -M * (2 - x[f][i][j] - x[f][i][i]) <=
               t_arrival[f][i] + t_wait[f][i] + t_charge +
                t_travel[i][j] - t_arrival[f][j]
            problem += t_arrival[f][i] + t_wait[f][i] +
                t_charge + t_travel[i][j] - t_arrival[f][j] <= M
                 * (2 - x[f][i][j] - x[f][i][i])

# Constraints used for charging capacity
for f in range(nVehicles):
    for i in range(nVertices):
        problem += t_arrival[f][i] + t_wait[f][i] >= lpSum(c[f
            ][i][t] * t for t in range(T_max)) - M * (1 - x[f][i
            ][i])
        problem += t_arrival[f][i] + t_wait[f][i] <= lpSum(c[f
            ][i][t] * t for t in range(T_max)) + M * (1 - x[f][i
            ][i])

for f in range(nVehicles):
    for i in range(nVertices):
        problem += lpSum(c[f][i][t] for t in range(T_max)) <= 1

for f in range(nVehicles):
    for i in range(nVertices):
        for t in range(T_max):
            problem += b[f][i][t] == lpSum(c[f][i][t_hat] for
                t_hat in range(max(0, t - t_charge + 1), t + 1))

for i in range(nVertices):
    for t in range(T_max):
        problem += lpSum(b[f][i][t] for f in range(nVehicles))
           <= c_cap[i]

# Constraints for progression of state of charge
for f in range(nVehicles):
    for i in range(nVertices):
        for j in range(nVertices):
            if i == j:
                continue
            problem += z[f][i] - z[f][j] - c_l[i][j] <= M * (1
                - x[f][i][j] + x[f][i][i])
            problem += -M * (1 - x[f][i][j] + x[f][i][i]) <= z[
                f][i] - z[f][j] - c_l[i][j]
            problem += z_max[f] - z[f][j] - c_l[i][j] <= M * (2
                 - x[f][i][j] - x[f][i][i])
            problem += -M * (2 - x[f][i][j] - x[f][i][i]) <=
                z_max[f] - z[f][j] - c_l[i][j]
```

```
# Making sure only one station is built
problem += lpSum(s[i] for i in range(nVertices)) <= 1

# Making sure non-charging stations can only charge if the new
   station is built there
for f in range(nVehicles):
    for i in range(nVertices):
        if i in C:
            continue
        problem += x[f][i][i] <= s[i]

# Making sure vehicles do not go above maximum charge
for f in range(nVehicles):
    for i in range(nVertices):
        problem += z[f][i] <= z_max[f]



# Solve problem
problem.solve(CPLEX_CMD(timeLimit=900))
```