A Method for Optimizing for Charging Cost in Electric Vehicle Routing

Matthew Lehrer

Computer Science: Innovation for Change in a Digital Society
Two-year master’s program
30 credits
Department of Technology and Society
Spring 2023
Supervisor: Johan Holmgren
 Examiner: Nancy Russo
Abstract
Adoption of electric vehicles has been restrained by the availability of charging stations and consumer fear of being stranded with a depleted battery, far from the nearest charger. In many areas of the world, charging stations are now widely available and the transition from vehicles with internal combustion engines is accelerating, though still in a fairly early stage. For electric vehicle drivers in those areas, anxiety that they will not be able to find a charger (“range anxiety”) is subsiding. However, differences in charging speed and pricing between stations and different outlets at the same station can be large. Total trip duration can vary significantly based on the charging outlet selected. Prior research has developed methods for helping all drivers find the fastest route and for electric vehicle drivers to ensure that they are able to complete their trip. Additional research has explored other complexities of route selection for electric vehicles such as how to select optimal stations for charging based on the total trip duration, including driving and charging time. Pricing for recharging electric vehicles at public chargers is more complex and diverse than for gas filling stations due to the differences in charging rates and the relatively low competition. This research investigates those differences. Using design science research methodology, a method is presented for determining which charging stops result in the lowest possible charging cost for a given route. The method is demonstrated through experiment with random routes within Sweden. The experimental results show that the average cost savings as compared to the duration-optimal route is 15% and 139 SEK per additional hour of trip time. One possible direction for future work is to improve the performance of the algorithm for use in real-time consumer route planning applications.

Keywords – Electric Vehicle Routing, Electric Vehicles, Shortest Path Problem with Constraints, EV, Charging Stations, Cost Optimization
Popular science summary
Electric cars are becoming more popular, but anxieties still hold back adoption. The possibility of a depleted battery far from home still makes some people anxious. But in many places, there are now plenty of charging stations to choose from. There are significant differences between those charging stations, though. Most importantly, they differ in speed of charging and their pricing. Usually, using a slower charger is cheaper than using faster chargers. Choosing where to charge is a complex problem, with unique solutions for different cars and drivers. Previous research has explored fastest routes and the complications for electric vehicles. Other work has found methods for optimizing refueling stops for internal combustion vehicles. This thesis develops and tests a way to find the lowest cost route for electric vehicles. After developing that method, we use random routes in southern Sweden to compare its performance to the fastest route. In this experiment, people could save 15% of the cost of charging on road trips. But those savings come at the cost of 5% longer trips. Judged against the longer trip time, the savings are 139 SEK per hour. That may be worthwhile for many drivers. In the future, routing applications could use the method to guide consumers. Researchers building on this work could focus on a way to estimate the cost of the route, known as a heuristic, to improve its performance. That will be important as the number of charging stations increases in the coming years.
Appendix 2: Search Algorithm 43
Appendix 3: Vehicle Specifications 46
Appendix 4: Functions for force and power at the wheels 47
Appendix 5: Experiment 58
1. INTRODUCTION

The transportation sector produces a large percentage of greenhouse gas emissions (Lamb et al., 2022) due to the use of internal combustion engines (ICEs), which consume fossil fuels. So, a transition from ICEs to electric motors and battery electric vehicles (EVs) can significantly contribute to a solution for a more sustainable economy. Now, EVs frequently have a lower total cost of ownership than ICE vehicles due to lower operating and maintenance costs (Palmer et al., 2018; Yang et al., 2023). Governments around the world have incentivized the transition with purchase-price rebates, which decreased the initial purchase price premium that EVs have had (Broadbent et al., 2022; Coffman et al., 2017) and to help EVs reach some economies of scale.

Besides price, consumers have resisted switching to EVs because of uncertainty if battery capacity would be sufficient to reach their destinations and if they would be able to recharge en route with minimal inconvenience. This uncertainty and fear is known as “range anxiety” (Neubauer & Wood, 2014; Zhou et al., 2023).

Investment in charging infrastructure in order to reduce range anxiety and increase the practicality of EVs continues to be intense and competitive. EV manufacturers, especially Tesla, have built out charging networks and even included free charging in order to increase the desirability of their cars (The Tesla Team, 2017). Governments have incentivized infrastructure investment and standardized charging cables and connectors across brands. Private industry has rushed to capitalize on the new business model of charging for charging, investing in parking spaces and charging infrastructure and selling electricity at a markup (San Román et al., 2011).

According to HERE, a navigation and routing company headquartered in the Netherlands, many countries have expanded their EV charging station availability to such a degree that their navigation routing selects the same route for EV and ICE vehicles (HERE Developer, 2023). That list includes more than thirty countries, including Austria, Croatia, Denmark, France, Germany, Japan, Norway, Sweden, the United Kingdom, and Vietnam, as well as twenty US States. In those areas, there is no longer any concern about the presence of charging stations along the fastest route. As charging station infrastructure investment continues, that situation is improving and spreading. Range anxiety is subsiding, though slowly.

For longer trips, consumers typically use routing software to find the fastest route and estimate the trip duration. Popular routing tools like Google Maps and Waze include charging station locations and some, for example A Better Route Planner, can recommend places to charge. Google Maps added charging station location functionality in late 2022 (Perry, 2022). Waze added a similar feature in March 2023. These features include the ability to enter information about the user’s vehicle in order to filter charging stations by outlet type (Hawkins, 2023).

However, calculating the optimal locations for charging – precisely where a driver should stop to recharge along the route – is a complex problem that involves integrating many kinds of data with a process for searching for the solution. In this context, optimal usually means most efficient with regards to time. That is, the shortest total duration for the combination of driving and charging (Sweda & Klabjan, 2012). It can also be defined as minimizing the variability of travel time and the chance of long delays (Chen et al., 2021). This problem is even more difficult in practice due to the heterogeneity among available EVs in the market. Battery capacity is different for every car. And driving range varies with the type of route, air temperature, weight differences from the average driver, and traffic density amongst other variables (De Cauwer et al., 2020).
Driver preferences also complicate the presentation of routes with charging stations. There are tradeoffs between overall trip time, willingness to detour from a main route, and association with something to see or do at the charging stations, known as places of interest (POI). A stop to refuel an ICE vehicle may have left enough time to buy a bag of chips but not to eat it. Stopping to recharge for thirty minutes or more creates a different situation. People might seek out charging stations with POIs like a scenic overlook or a sit-down restaurant. On one hand, something to do while recharging is especially of interest where fast charging is not available. On the other hand, some consumers might seek out a less expensive slow charger if something appealing can occupy the time difference. These factors are on top of the typical routing choices, including avoidance of tolls, ferries, or highways, which might be avoided by necessity with certain kinds of vehicles or trailers or by choice for more diverse scenery.

Unlike for filling stations where competition has driven pricing toward parity, pricing from station to station for recharging is far from equal. As of this writing, the European Commission notes that recharging a Nissan Leaf with a 27.3 kWh battery from 10% to 80% varies between €6.55 and €18.02 in Sweden (European Alternative Fuels Observatory, 2022). Even at the same charging station outlet, different consumers can pay different prices. Some mobile apps have deals with charging station owners and some station owners have memberships with lower pricing. Hardman et al. (2018) found that some charging station owners require membership and that some regions have up to 20 owner networks, all of which might require membership to access their chargers. These differences further complicate the possibility of providing drivers with accurate information.

The availability of fast chargers, which provide more than 22 kW and “super-fast” chargers which provide 50 kW or more make a large difference in recharge time for consumers as compared with older, slower 6 kW or 11 kW chargers. Recharging that Nissan Leaf’s 27.3 kWh battery from 10% to 80% would take more than three hours at 6 kW and around 25 minutes at 50 kW. And that is for a car with a relatively small battery. A Polestar 2 with the long range battery option has an 82 kWh battery (Polestar Sverige, 2023) and would therefore take three times as long to charge. A charging station could generate much more revenue from faster charging outlets. They can sell more energy per hour and consumers are willing to pay a premium price for the ability to recharge and drive away sooner.

1.1 PURPOSE AND OBJECTIVES

There are many factors that influence the reduction in greenhouse gas emissions in the transportation sector. And, likewise, there are many factors that influence consumer adoption of electric vehicles, a part of that reduction. As the build out of charging infrastructure continues, range anxiety decreases. Other consumer anxieties remain. One is the greater complexity of route planning. Another is concern over the total cost of ownership for EVs.

Route planning is a complex problem even before accounting for the difficulties associated with EVs. It involves a large amount of data and unique conditions for each search (De Cauwer et al., 2020). The combination of computational complexity, consumer preferences, and limited aggregation of and access to accurate, reliable charging station data creates a difficult problem. Over time, Moore’s Law, improvements in algorithms, and digitalization are likely to gradually solve the computational complexity problem. Consumers will be better served for individual preferences through competition in different user interfaces and experiences. And further increases in the
supply of charging stations is likely to decrease differences in charging prices. These problems will be solved step by step.

The total cost of ownership for EVs is a less complicated problem. The initial costs are understood. They are mainly purchase price and financing costs, taxes, and potentially government rebates. Estimates for maintenance costs may not be as reliable as for ICE vehicles because there is less history as EVs are newer. Averages for maintenance costs should be close enough. But some operating costs depend on the specific circumstances of a consumer including where they drive and how frequently they need to charge at public charging stations. The costs of a daily commute can be calculated based on the local electricity prices that consumers pay at home and the energy used on the commute, based on the car and the roads used. For some consumers, longer trips that necessitate public charging stops may be a significant factor in their cost of ownership. These consumers have limited ways to find out what those trips cost and how much the prices they pay for charging on those trips might vary.

Both of these problems can be resolved with a shared solution. In order to encourage further transition from ICE vehicles to EVs, problems like these must be overcome. Consumers should be able to figure out how an electric vehicle will fit into their life and be able to estimate its cost of ownership. The purpose of this thesis is to develop a method that can aid route planning and shed light on the cost of ownership of EVs to reduce consumer anxieties.

To achieve that purpose, a few objectives must be met. The first objective to accomplish that is to develop an algorithm that can be used to find the lowest possible cost of recharging at public charging stations. That algorithm will take a base route, a vehicle specification, and a set of potential charging station stops as input. It will return an optimal path describing which charging stations to stop at and how much to charge there for the lowest possible trip cost. In some cases, it will determine that it is not possible to complete the route given the constraints of the vehicle battery. The second objective is to test that algorithm to determine its correctness. It will be compared to the results of the fastest route to ensure that the optimizations make sense, to the extent possible. The third objective is to compare the optimal costs with the fastest route in order to determine how much savings is possible and whether consumers should care.

By meeting these objectives, the algorithm could produce information that can reduce the remaining anxiety over charging, range, and trip planning in consumers. That will lower the barrier to EV adoption and further reduce transportation greenhouse gas emissions.

1.2 LIMITATIONS

The algorithm designed in this project can be customized according to the data and computational resources available. It is outside the scope of this project to prove that it finds the cost-optimal route. However, some tests are done to show that the results are appropriate and make sense to the extent possible.

The demonstration of the algorithm also has limitations. The energy use for route segments would be more accurate with additional data. The formulas are in place for including the slope of each road segment and how that would impact battery draw. However, elevation data has not been integrated. Additionally, drive cycles are not used and a simple method of instantaneous acceleration, explained below, is used. Without those, no segment has negative energy use due to deceleration or downhill segments. Given more time, these additional aspects could be included
and would increase the accuracy and precision of the experimental results. These changes do not impact the algorithm method, only the experimental results.

The API used as the source for charging station and price information does not provide enough route information for the estimation of energy use. It may be possible to reverse engineer or compute the necessary information but that is outside the scope of this project. A method of generating the route, discussed below in II(B) and V(A), is used which does include details necessary for estimating energy use. That method’s route occasionally differs significantly from the API for pricing data. In those cases, the charging stations available in the search graph may be so far from the main route that a successful route to the destination without depleting the battery is not found or the estimates of costs are distorted. Further, that API does not have complete information on all charging stations in its dataset. Stations without pricing data are excluded from the experiment. It is assumed for this project that the pricing data it has is accurate and, anecdotally, it is.

1.3 RESEARCH QUESTIONS

To aid consumers in route planning with electric vehicles given different prices at charging stations along the route, and to increase understanding of the impact of those different prices, the following research questions were formulated. Answering these questions will meet the objectives of developing an algorithm for optimal cost paths for EVs, test its correctness, and compare its results with the fastest paths.

RQ1: How can we optimize a driver’s financial cost of charging at public charging stations for electric vehicle routing?

In other words, given an origin and destination and a set of charging station outlets with various prices, how can we find the path with the lowest financial cost to complete the route?

RQ2: What is the variation in financial cost and trip duration between the fastest and cost-optimized routes in Sweden based on current charging station pricing?

Is there a financial benefit to the consumer of using that route, and if so, how big are those savings as compared with the fastest route?

1.4 THESIS OUTLINE

The remainder of this thesis will be structured as follows. Section 2 provides the theoretical background needed for understanding the technical aspects of the research area and methodology. Section 3 discusses the existing research that has been done on routing, electric vehicle routing, and price differences. It ends with a discussion of a gap in the research, which this thesis addresses. Section 4 presents the research methodology used to answer the research questions. Sections 5 and 6 describe the methods used to answer RQ1 and RQ2, respectively, the motivation for those methods, and how the research methodology is implemented. Section 7 presents the experimental data generated by the methods in Sections 5 and 6. And Sections 8 and 9 analyze and discuss that data and ideas for future work.
2. **THEORETICAL BACKGROUND**

2.1 **GRAPH THEORY**

Deciding how to move through the world is a problem everyone faces. Sometimes efficiency does not matter but many times it does. And relationships between people and things create networks all around us. In order to find efficient, reliable, or in some sense “optimal” ways of moving and to analyze and understand networks, we rely on an area of mathematics called graph theory (West, 2001).

A graph is, informally, a set of points with lines connecting some of them in some way. In graph theory, the points are known as vertices or nodes and the lines are known as edges. A graph is, formally, a set of nodes $V$ and a set of edges $E$ that join nodes in $V$ (Bondy & Murty, 1976).

Graphs can be used to model and understand both physical networks and intangible concepts. Physical networks that are commonly modeled as graphs include road and railway networks, electrical circuits, and organic molecules. Intangible concepts that are commonly modeled as graphs include online and offline social networks or the flow of control in software programs. (Gross & Yellen, 2005)

In some graphs, relationships are symmetrical. People who know each other and counties that are adjacent can be represented in this way and are known as undirected graphs. Road networks with one-way streets and social networks where one person can follow another without that being reciprocated cannot be represented that way. The edges have directionality because the relationships are asymmetric. These directional edges are called arcs and the graphs are known as directed graphs. (Gross & Yellen, 2005)

It is often necessary to add information to the graph. One of the most common methods is to associate a value with each arc, where the number is known as its weight, creating a weighted graph. For a graph representing an electrical circuit, the weights could be the resistance or power loss of that segment. For a graph representing a road network, the weights could be the distance or the amount of time it takes to travel between nodes. (Gross & Yellen, 2005)

2.2 **SHORTEST PATH PROBLEMS**

In many graphs, there are multiple paths or sequences of unique nodes that connect one node to another. In a weighted graph, we can compare those paths by taking the sum of the weights of the arcs along each path. Many times, there is a “best” or optimal path which is the path with, depending on context, the minimum or maximum sum of weights for all its arcs. More often than not, the minimum is best. This kind of problem is known as the shortest path problem (Gross & Yellen, 2005).

An example of a directed, weighted graph is depicted in Figure 1. The path from A to D through B has a total weight of 5. The path from A to D through C has a total weight of 6. The shortest path is A to B to D.
There are many ways to solve the shortest path problem and the list often begins with breadth-first search (BFS), a simple algorithm that is the basis for many others. In BFS, a graph and a source node are the input. The algorithm “discovers” all nodes that are reachable from the source, where a path of arcs exists from the source to a particular node, keeping track of the distance from the source for each node. The order that the algorithm uses for the search resulted in the name: nodes with the same distance from the source are discovered before further nodes. (Cormen et al., 2022)

Depth-first search (DFS) is similar, though, as its name implies, the order of nodes discovered is different. In DFS, nodes are searched by moving further from the source whenever that is possible. Then, the algorithm “backtracks” to a node closer to the source before moving outward again until all nodes are analyzed. Typically, DFS allows multiple source nodes whereas BFS is used with only one. BFS and DFS have similar performance where the processing time scales linearly with the size of the input graph (Cormen et al., 2022).

Finding the fastest or shortest route through a road network is a natural application of Dijkstra’s algorithm (Dijkstra, 1959), a solution to the shortest path problem which builds on ideas from BFS. For Dijkstra’s algorithm, the road network is modeled as a graph with intersections as nodes and road segments as arcs. The weights of the arcs are the travel time for the fastest route or distance for the shortest route it takes to traverse that route. With an origin, destination and road network graph as input, the output is the fastest route from origin to destination. As digitalization has progressed, solutions have improved in various ways.

Many methods have been developed to increase the performance of the shortest path problem algorithms. BFS, DFS, and Dijkstra’s algorithm all find the distance from a source to all other nodes in the graph. For the purposes of routing, stopping once the shortest path to the destination is found is an easy improvement. Significant improvement has been made by optimizing the priority queue data structure used by Dijkstra to keep track of the current set of optimal subroutes, from $O(n^2)$ to as fast as linear time (that is, proportional to the size of the graph instead of growing exponentially with the size of the graph) “with high probability” in some cases. (Schultes, 2008)

The A* algorithm uses a heuristic – essentially a guess of which direction is most likely best – to improve the probability that the optimal path will be found earlier in the search process (Hart et al., 1968). Bidirectional search algorithms improve on A* by searching from both origin and destination, further reducing the number of nodes that must be evaluated before finding the optimal path (Pohl, 1971; Pijls & Post, 2009).
2.3 APPLICATION TO ROAD NETWORKS

Other techniques take advantage of known attributes of road networks. Alleys and side streets are unlikely to be used in the middle of longer trips. Highways are more likely. That is, road networks are hierarchical. Separator-based, reach-based and other heuristic approaches can take advantage of the hierarchy by preprocessing the graph network and using data stored during preprocessing to further improve query efficiency (Schultes, 2008). Contraction Hierarchies, in which nodes are ordered by importance to the overall network (Geisberger et al., 2008), is one such technique now commonly used. Multiple techniques can also be used in combination.

Further, many details about the road network have been added to these solutions in order to increase the accuracy of the models and their output. Real-time traffic and temporary speed-limit changes, turn costs, road closures, points of interest, and other details are added in various combinations. (Delling et al., 2017; Gao & Chu, 2022)

2.4 CHALLENGES FOR ELECTRIC VEHICLES

The simplest solutions for routing for electric vehicles do not have to take all of the above road network complications into account. They do, however, have to solve for a key difference from ICE vehicles: the limited battery capacity of EVs. This means that a subset of the shortest path problem known as the shortest path problem with constraints concerns us here. In that set of problems, possible paths are restricted by placing some condition on subpaths (Joksch, 1966). For EV routing, a subpath cannot exceed the battery capacity and subpaths become unusable – usually denoted with infinite weight – if the energy use for that segment exceeds the state of charge as the vehicle would enter it (Storandt, 2012; Sun & Zhou, 2016).

Complicating matters even more, not all route segments drain EV batteries. Downhill segments and deceleration can have a net positive impact on battery usage. With that in mind, optimizing EV routes could focus on time or energy efficiency. A least-energy path algorithm cannot use Dijkstra’s algorithm directly, however, because it cannot handle negative arc weights. Other algorithms, including Bellman-Ford, can handle negative arc weights though they are much slower. (Artmeier et al., 2010)

The battery constraint is the most important and solutions rely on earlier work in solving the shortest path problem with constraints (Irminch & Desaulniers, 2005; Storandt, 2012). The main technique is to keep track of the battery’s state of charge (SoC) when it reaches each node and set the arc weight to infinity if the SoC is insufficient for traversing that route segment.

In terms of computing the fastest route, stops for refueling ICE vehicles can largely be ignored in route planning because they take little time and filling stations are widely available. Including charging in route planning for EVs is, however, necessary. Public charging stations are relatively more sparsely located and recharging can contribute a meaningful amount to the overall trip duration. There are some challenges with including EV charging stations in route models. First, different chargers take different amounts of time to recharge. When the primary objective of most routing applications is to find the fastest path, this time becomes important. Second, a stop at a charging station does not automatically result in a fully recharged battery. Long charging times mean drivers can consider whether a partial charge is more efficient in time or financial cost, if the battery can be charged cheaper at a later stop or after reaching the destination. Partial charging leads to an “infinite amount of different charging configurations, where all configurations result in
different paths” through the graph (Funke & Storandt, 2015). And charging time is non-linear. It does not necessarily take the same amount of time to charge from 25% to 35% as it does from 75% to 85%. Most EV batteries are lithium ion and charge more slowly as they near full charge (Kancharla & Ramadurai, 2020; Montoya et al., 2017).

2.5 VEHICLE POWER USAGE

There is quite a bit of research into how to model power usage per route segment in order to accurately predict range and to plan the activities of EV fleets (Erdelić & Carić, 2019; Kucukoğlu et al., 2021; Xiao et al., 2021). Any method for computing the energy use for a route would rely on a representation of speed and acceleration for a vehicle traveling along that route. The most straightforward representation is to assume instantaneous acceleration and that a route segment is always traveled at the average speed for that segment. More accurate, complex, and computationally expensive methods include modeling “drive cycles” for each type of road and route segment (Alateef & Thomas, 2023).

A drive cycle is a second-by-second time series of the speed of the vehicle. From that list of speeds, acceleration can be calculated as the change in speed from one second to the next. Drive cycles are used in many kinds of simulation (Kharrazi et al., 2018). They are also used as guides for what speed vehicles should travel for consistent comparisons of emissions in environmental testing (André, 2004). In those tests, drivers match the drive cycle speeds as closely as possible and emissions are measured.

To go from speed to power usage requires a model. There are computationally expensive models with the goal of creating the highest fidelity model, matching reality as closely as possible. For example, the Future Automotive Systems Technology Simulator (FASTSim) is focused on modeling light, medium, and heavy duty vehicle powertrains. It uses data about vehicles and second-by-second vehicle speed lists known as “drive cycles” as input. It outputs data on vehicle efficiency, and fuel economy or battery life, depending on vehicle type. The drive cycles model different road types, that is urban, rural, highway, etc. FASTSim does not directly model specific routes through a road network. (Brooker et al., 2015)

The Simulation of Urban MOBility (SUMO) project’s software tool can be used to model traffic and simulate how vehicles move through a road network. It is used extensively in urban planning and other transportation simulations. It can be used to create many kinds of simulations including traffic analysis, how changes in road networks affect traffic use, multi-modal analyses, and generation of driving cycles by generating simulations of changes in speed throughout a route due to turns, traffic signals, and other road users. Setting it up for use in this type of application and experiment is a burden out of scope of this project. It could be used to generate drive cycles to gain an understanding of vehicle efficiency, but it is not designed to provide input for other real-time applications such as routing. (Behrisch et al., 2011)

Other methods for estimating battery use include a constant or speed-dependent regeneration factor where the amount of power necessary to move the vehicle is built up from the Newtonian physics at the wheel, through the power loss due to heat and efficiency at the motor and the battery (Genikomsakis & Mitrentsis, 2017). With a method for determining whether the battery’s SoC will allow traversing each route segment, a constraint in this shortest path problem can be added to the graph traversal algorithm.
3. RELATED WORK

With these main additional complications of EV routing, path-finding models have been created, iterated on, and improved in various ways. Many have retained the focus on trip duration in their optimizations. One example, particularly relevant to this thesis, is focused on the optimization of charging station choice and the duration of charging at each station (Huber et al., 2022; Huber & Bogenberger, 2015). In that work, which is described in more detail below in Section 5.1, a route with various potential charging stations is turned into a graph. Huber et al. (2022) developed an algorithm for finding the lowest trip duration based on which charging stations are stopped at for charging and how much the battery is recharged at each of those stations.

Other work has focused on energy-efficient paths (Chen et al., 2021; Erdoğan et al., 2023) where the path that consumes the least overall energy is defined as optimal. With that definition, energy consumption is approximately 10% less than when using the fastest path (Kluge et al., 2013). These optimal paths could have longer trip durations because they avoid steep climbs or roads with higher speeds where the energy consumed is comparatively inefficient.

Consumer charging costs have been explored, particularly in relation to the changing prices of residential electricity between day and night and the ways in which that will impact overall grid demand as more households transition from ICE vehicles to EVs (Jin et al., 2013). Studies of total costs of ownership include charging costs but generally assume one price per kWh, the prevailing price that most consumers would pay for overnight charging at home. That makes sense, because that is where and when most charging takes place. (Hagman et al., 2016; Palmer et al., 2018)

Managing the load on the grid through dynamic pricing for EV charging has also been studied (Cui et al., 2021; Valogianni et al., 2020). But that is from the perspective of the electricity providers and the grid. The perspective of the individual electric vehicle driver has largely been ignored in this regard.

Recent work by Lanz et al. (2022) explored the prices paid for charging across Europe and they found a “large variance of charging costs.” Their focus, however, was on overall prices across different types of charging stations and outlets and on breaking down the components of that pricing (for example, infrastructure and taxes), not on how those prices affect drivers on individual routes.

Decision support for refueling choices of ICE vehicles has been studied, with a focus on commercial trucking where a lot of fuel is consumed and there is a profit motive to seek out optimal solutions. Much of that research focuses on the fixed-route vehicle-refueling problem (FRVRP) in which the main route, along with the locations of and information about fueling stations, is an input to the problem of choosing which fueling stations along that route are optimal (Lin, 2011; Suzuki, 2014).

The variable-route vehicle-refueling problem expands on the FRVRP by exploring a wider area and different potential base routes for an optimal solution (Suzuki & Dai, 2013). Optimal solutions exist though they are not widely used in practice. Adoption is held back by computational complexity and the view of truck drivers that using these tools takes away their freedom (Suzuki et al., 2014). It appears that many times, the choice of truck stop is about more than just fuel and fuel pricing in much the same way that POIs can influence consumer choice.

FRVRP solutions are used by commercial software vendors serving the commercial trucking industry. They claim that across a company’s fleet, usage of the solution can save upwards of $1
million per year. These solutions are all implemented using heuristics instead of exact solutions because of the computational complexity involved. For a large-scale road network, the size of the search graph across all possible refueling stations is too large. There is, therefore, research into improving the quality of those heuristics and comparing the solutions found using the heuristic with an exact, optimal solution. Each one percent gap between the heuristic solution and the optimal solution is a difference of USD 1 billion for the trucking industry as a whole. (Schulz & Suzuki, 2023)

Outside of academia, the question facing potential EV owners of how much drivers pay for charging at public charging stations on longer trips has been remarked upon. The Washington Post recently partnered with the think tank Energy Innovation to explore the question, with the implication that EV adoption is held back because consumers view public charging as potentially more costly than refueling ICE vehicles on similar trips. Their analysis looked at two main routes, one across California where charging stations are plentiful and one across a longer stretch of the United States from Detroit to Miami where charging stations are sparse. Their focus was on comparing EV charging costs with refueling ICE vehicles, not on the optimal charging prices. (Coren, 2023)

This thesis is, therefore, concerned with the development of an algorithm for the optimization of the financial costs of charging electric vehicles through road networks given different prices at charging stations. As in FRVRP, the focus is on the choice of the charging stations and does not account for potentially varying the main route. The methodological framework and specific methods for addressing that optimization problem are presented next.
4. RESEARCH METHODOLOGY

A primary goal of this work is to produce a method for computing the optimal financial cost path given certain constraints. The method will be an artifact of this research (Hevner et al., 2004). To ensure “utility, quality, and efficacy,” and to guide the process and aid readers and reviewers of this work, the framework and template for design science research in Peffers et al. (2014) is used. Peffers’ framework was selected for two main reasons. First, it builds on a shared mental model for this type of research that readers may have encountered before. If so, mapping the components and assessing the quality of this work will be easier. Even if the Peffers framework is new to the reader, the framework sets peer-reviewed guidelines for process and evaluation. Second, the framework is a better fit for the research than alternatives.

4.1 PEFFERS FRAMEWORK AND MENTAL MODEL

The framework described in Peffers et al. (2014) was developed by consolidating the ideas from seven prior influential papers on design science research methodology. Those ideas were distilled into six “well-accepted elements” or activities. 1) Problem identification and motivation in which the research problem is defined and the value of a solution is justified. 2) Define the objectives for a solution in which quantitative or qualitative objectives are “inferred rationally” from the problem definition. 3) Design and development in which the artifact meets the definition from Hevner, et al. (2004) or Järvinen (2007). In Hevner et al. (2004), the artifact can be a construct, model, method, or instantiation. In Järvinen (2007), the artifact explains “new properties of technical, social, and/or informational resources.” 4) Demonstration in which the artifact is used in an appropriate activity to solve the problem. 5) Evaluation in which observations and measurements are made on how well the artifact solves the problem or supports a solution. 6) Communication in which the resulting knowledge – the problem, the artifact, and its effectiveness – is packaged for dissemination. Different activities can serve as a starting point for different types of problems or contexts. For example, a consulting engagement might begin with the observation of a solution in activity four, and then expand to “apply rigor to the process retroactively.” (Peffers et al., 2007)

4.2 ALTERNATIVE FRAMEWORKS

There are other well-regarded frameworks for design science research including Hevner et al. (2004) and Mettler et al. (2014). Because these frameworks all share the goal of developing “a system of principles, practices, and procedures” to guide practitioners of design science research (Peffers et al., 2007), there is significant overlap. That makes the choice of framework somewhat more difficult, though all of these frameworks are more than adequate. The following is a brief description of each with their differences highlighted.

Mettler et al. (2014) focuses on experiments in design science research. The goal of their framework is to design “an evaluation procedure for gaining generalized knowledge about the utility of distinct design alternatives.” While there was some iteration in the design of the artifact in this work, the focus is not on the evaluation of different possible artifacts in order to choose the best one.

As one of the seven papers that Peffers is based on, the ideas from Hevner et al. (2004) influenced and are included in the Peffers framework. Hevner’s framework includes seven
guidelines: 1) Design as an Artifact, 2) Problem Relevance, 3) Design Evaluation, 4) Research Contributions, 5) Research Rigor, 6) Design as a Search Process, and 7) Communication of Research. Guidelines 1, 4 and 6 are combined in Peffers as Design and Development. Peffers found that the search process of guideline 6 was unique among the seven papers it is based on. Guideline 2, on problem relevance, is split into the first two Peffers processes. The objectives of a solution, implicit in Hevner, become explicit in Peffers. Guideline 3, on evaluation, is also divided in Peffers, into separate demonstration and evaluation processes. Guideline 7, communicating the research, is essentially unchanged in Peffers. Peffers has, therefore, more clearly delineated steps making it easier to execute and to follow as a reader.

4.3 APPLICATION OF PEFFERS

Sections 1 and 3, above, cover the Peffers processes for identifying the problem and the motivation for a solution. This Section will describe how the rest of the thesis fulfills the methodology as described by Peffers et al. (2014) in the creation and evaluation of an artifact which can be used to find the lowest cost route for an EV driver within some constraints.

A solution to this problem will have the following characteristics. It will define the constraints within which it is appropriate to use and its limitations. Within those constraints, it will produce a single, optimal path from origin to destination. Information about that path will also be produced. That information will include which charging stations to charge at, the amount of battery capacity recharged at each of those stops, the total duration of the trip, and the total financial cost of the trip.

It is outside the scope of this project to include a performance threshold, so that the method could, for example, be used in real-time applications. A solution, therefore, does not need to be tested for its storage or computation characteristics.

The artifact that results from this research will be a method (Hevner et al., 2004; Peffers et al., 2007) for determining the financially-optimal strategy for charging electric vehicles along a long-distance route. The method takes a vehicle specification, route information, and a list of charging stations near the route as input and outputs an optimal charging strategy. It is specified in Section 5.

Demonstration will be used to show the applicability of the method. An evaluation through logical proof that the modifications of Dijkstra’s algorithm provided by Huber & Bogenberger (2015) as described below can be extended to include financial cost metrics is outside the scope of this project. A demonstration of its use in a specific scenario is discussed and analyzed in Sections 6 and 9, below, respectively.

4.4 IMPLEMENTATION APPROACH

The search algorithm, its tests, and the experiment are implemented in Typescript. The graph construction relies on a library called ngraph.graph, written primarily by Andrei Kashcha (Kashcha, 2013/2023). The priority queues, the primary data structure in the search process, use a library called Heap.js, written by Ignacio Lago (Lago, 2017/2023). Both of those libraries are distributed through the npm registry. The graph implementation is integrated with real-world data for the experiment. The road network data is from Open Street Maps and distributed by Geofabrik (Geofabrik Download Server, n.d.). That road network data is processed by the Open Source Routing Machine (OSRM)
(Project OSRM, n.d.) which is queried for the base, fastest route. Charging station pricing and location data is added via ChargeFinder (ChargeFinder, 2023). Further details are provided below in Section 6 and the appendices where a link to the full implementation on GitHub is also supplied.
5. FINANCIAL-COST OPTIMIZATION METHOD

In order to find the cost-optimal route, a directed graph is constructed from the route and charging station data. Then, a shortest path problem with constraints algorithm is used on that graph in which the constraint is the charge level of the battery. The objective of optimization is the lowest financial cost of charging. The algorithm for optimal charging station selection and the method of constructing the graph presented by Huber & Bogenberger (2015) are used as a starting point. Their notation for labeling graph nodes is used as well.

5.1 GRAPH CONSTRUCTION

In order to create the graph that will be searched for the optimal route, one main route is used as a foundation. That base route is selected by a shortest path algorithm, as described in Section 2.2. Charging stations along that route are added if and only if charging speed and pricing is available. The origin, each intersection along the base route that is closest to a charging station, and the destination are added to the search graph. While the base route may have many curves, turns, and speeds, the search graph can be simplified and modeled at this point as in Figure 2.

![Figure 2: The first nodes and edges are added to the search graph. The six nodes here are the origin (s) and destination (d), two charging stations (c₁ and c₂), and the intersections along the base route nearest those charging stations (a₁ and a₂).](image)

There are multiple requirements for paths through this graph, so each arc has some associated information about traversing it: the energy used, the duration, and the financial cost. The energy use is calculated as described in Section 2.5. The duration can be included as a fixed arc weight if no changes due to traffic are required. When ignoring traffic, the duration is part of the information returned in the routing methods described in Section 2.3. Traffic-dependent duration would have to be calculated during the graph search because the timing of arrival at different road segments depends on the starting time of the route and choices of when to charge. Here, traffic is ignored and fixed duration weights are added to the graph.

At this point, the graph’s nodes are only the charging stations and the intersections closest to them. However, the routing method generally divides the route from one turn to the next, so some work must be done to get the correct data for each arc since these intersections are, almost by definition, in between turns on the base route. The energy use and duration associated with these arcs are the sum of all the route segments from the previous intersection or from the origin.

Each charging station may have multiple outlet types which charge at different rates and have different pricing. For example, a single location might have eight parking spaces available for charging with four outlets which charge at 43kW for 7 SEK per kWh and four outlets which charge at 150kW for 8.5 SEK per kWh. Each outlet type – in this example, one for 43kW and one for 150kW – is added to the search graph. Each of these different outlet types is added as a complete set of...
nodes and arcs from the base route. If a charging station has multiple outlets, they are added as $c_i$ and $c_{i+1}$ and there is zero energy use and duration in traveling from $a_i$ to $a_{i+1}$.

Next, as mentioned in Section 2.4, there is a possibility of infinite different charging amounts. Instead, a finite set of potential charge levels that the vehicle could charge to and leave the station with are considered. Each of these is called a “target state” because the vehicle can enter with a lower charge state and leave when it reaches the target. A node is added to the graph for each target state. The computational resources, in terms of computation time and more importantly available memory, increase with smaller steps between those states of charge included in the search. For example, a 10% step size uses more resources than 20%. Any step size can work with this method, depending only on the available resources. A step size of 25% is shown in Figure 3, along with additional nodes and arcs breaking down the charging process. There are arcs in the search graph associated with traveling from the main route to the charging station, the time and cost of charging to different states of charge, the overhead time at the charging station, and travel time to return to the main route.

So, for each charging outlet $k$ along the route, nodes representing the state of the vehicle before ($i_k$) and after ($o_k$) charging as well as nodes representing each target state (“ts”) of charge ($c_{k}^{ts}$) are added to the graph. Metadata for the outlet’s charge rate, cost per kWh, and cost per minute of parking are attached. The node from which the vehicle would leave the main route in order to charge at charging station $k$ is denoted $a_k$ and the intersection at which the vehicle returns to the main route from that charging station is denoted $b_k$. The node at $b_k$ may represent the same location as $a_k$. Similarly, charging outlet $k+1$ may be a different outlet type at the same charging station as charging outlet $k$. In that case, $a_k$ and $a_{k+1}$ would represent the same physical location and the arcs from $a_k$ to $b_k$ to $a_{k+1}$ would have zero duration and energy use associated with traversing them.

The following arcs are also added. The arc from $a_k$ to $i_k$ represents the time, distance, and energy use from the main route to the charging station and it is labeled with weights accordingly.
The arc from $c_k^{ix}$ to $o_k$ represents the overhead of stopping at a charging station: no distance or financial cost but some time to park, get out of the car, setup payment, connect the car to the outlet, and get back in and go. The arc from $o_k$ to $b_k$ represents the time, distance, and energy use for the path back to the main route from the charging station and it is also labeled with weights accordingly.

The arc from $i_k$ to $c_k^{ix}$ is more complicated. It represents the time and financial cost of charging to that target state. It is associated with data about the pricing and current for that charger. It is further labeled during the algorithm runtime with time and financial cost of charging, which are not known while constructing the graph. They are dependent on the state of charge of the vehicle when it enters the charging station and, therefore, on the path through the graph to get there. For example, if the vehicle enters the charging station with a 27% SoC, the arcs from $i_k$ to, say, $c_k^{10}$ and $c_k^{20}$ have infinite weights and will not be included because the charger can only increase the charge. The time and financial cost of charging from 27% to any target states larger than 27% are included and the formulas for doing so are defined below. It is worth noting that these arcs are the only ones in the graph with a non-zero financial cost since these are the arcs that represent the process of charging.

At this point, the graph construction is complete. The implementation is provided in Appendix 1 and the method for using this graph to find the lowest cost path through it is described next in Section 5.2.

5.2 SEARCH ALGORITHM

In order to find the shortest path through the graph, some additional inputs are necessary besides the graph itself. The starting node and the destination node which were used as inputs for the graph construction are used again. The vehicle’s SoC at the starting node is a new requirement. The battery capacity for the vehicle whose specification was used for the energy use in the graph construction provides the constraint for the search algorithm. That is, the battery can never be empty, so the optimal path output by the search algorithm can never include a segment which requires more energy than the SoC of the vehicle as it enters that segment. Because of some variability in energy use and driver anxiety around getting stranded with a depleted battery, a minimum SoC above zero can also be used (Huber et al., 2022). Because we are ignoring traffic and using fixed durations for each road segment, the cost function for time on road segments is to reference the fixed weight in the graph. To calculate the time, in seconds, used for charging, the following function is used, where charge rate is the capacity of the charger adjusted for its efficiency (e.g., 43kW * 90%) and slow charge rate is the reduced charging rate associated with battery longevity as mentioned in Section 2.4 (e.g., 43kW * 90% * 50%):
charging duration = \begin{cases} 
0 & \text{if } SOC \geq ts \\
\frac{ts \times \text{battery capacity}}{\text{charge rate}}, & \text{if } ts \leq 80\%,
\end{cases}

\begin{align}
0.8 \times \frac{\text{battery capacity} - \text{SoC}}{\text{charge rate}} + \frac{(ts - 0.8) \times \text{battery capacity}}{\text{slow charge rate}}, & \text{otherwise.}
\end{align}

Those are the inputs used in the algorithm in Huber & Bogenberger (2015). Their algorithm is, as mentioned, based on Dijkstra’s algorithm and is shown below in pseudocode.

<table>
<thead>
<tr>
<th>Inputs: graph g, start node s, starting SoC initial_SoC, minimum SoC min_SoC, battery capacity batt_cap, and destination node d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
| 6 | Create a new label \( L_{\text{new}} = \{ \) \begin{align} 
\text{cumulative_duration: } & L_{\text{cur}}.\text{cumulative_duration} + \text{the duration of } e, \\
\text{cumulative_energy: } & c_E^{\text{new}}, \\
\text{previous_node: } & L_{\text{cur}}.\text{current_node}, \\
\text{previous_node_index: } & L_{\text{cur}}.\text{current_index}, \\
\text{current_node: the node } e \text{ goes to}, \\
\text{current_index: increment } & L_{\text{cur}}.\text{current_index}
\end{align} \}
| 7 | If \( \text{initial}_\text{SoC} - c_E^{\text{new}} \geq \text{min}_\text{SoC} \) |
| 8 | Add \( L_{\text{new}} \) to \( L_{\text{perm}} \) |
| 9 | End if. |
| 10 | End for. |
| 11 | End while. |
| 12 | If there is a label in \( L_{\text{perm}} \) associated with node \( d \), return it |
| 13 | Else, return null |
There are three major differences from Dijkstra’s algorithm, as noted by Huber & Bogenberger (2015). First, the “more complex notion of labels” which are added to the priority queue. These labels include six data points: 1) the cumulative time cumulative_duration, 2) the net cumulative energy used and recharged cumulative_energy, 3) the prior node used to reach this one previous_node, 4) an index for the label associated with this path stored in the data for that previous node previous_node_index, 5) the node this label is associated with current_node, and 6) similar to the index in 4), the index for the current label which may be referenced by the next node current_index. Second, the addition of the battery constraint in lines 7 through 9 here. And third, that “dominated labels” — those which have both a higher cumulative_duration and higher cumulative_energy for a given node — are not removed from the open node priority queue \( L_{\text{open}} \). Leaving those labels in the priority queue ensures that optimal paths are found. Unlike the use cases for which Dijkstra was originally designed, in electric vehicle routing it is possible for the subpath after a node to affect which subpath from the source to that node is optimal. For example, if a route requiring only one charging stop has a 150kW charger available 49% of the way through the route and a 43kW charger available 51% of the way, skipping the first charger will appear to take less time when half the graph has been analyzed. It is only when looking deeper in the graph that it becomes clear that the earlier stop takes less time for the full route.

If a label for the destination \( d \) is found in the closed node priority queue, the optimal path has been found. The label returned in step 12 in that case includes the information on the cumulative duration but does not include which charging stations are included in that optimal path. This is done so that each label, each priority queue, and, therefore, the program as a whole uses as little memory and computational resources as possible. The full path and the charging stations included can be recreated by searching the closed node priority queue for the prior label which would have a current_node and current_index that match the previous_node and previous_node_index, respectively, of the destination node, and then continuing to do so recursively until reaching the origin.

To adapt this algorithm to optimize for financial cost, two changes are necessary. An additional field cumulative_financial_cost is added to each label. It is calculated according to the following function, where \( ts \) is the target state or the percentage charge level when the vehicle is done charging at this station, \( SoC \) is the percentage the battery is full before charging, charging_duration is the amount of time in seconds spent charging as defined above, cost_per_kWh is the cost in local currency per kilowatt hour of charging, and cost_per_min is the cost per minute spent connected to the charger for those outlets which charge for parking separately from the cost of electricity.

\[
\text{charging cost} = \frac{(ts - SoC) \times \text{battery capacity} \times \text{cost per kWh}}{1,000} + \frac{\text{charging duration} \times \text{cost per min}}{60} \quad (2)
\]

Next, the priority queues must be adapted. A naive approach — which was initially tried by the author — would select the label associated with the minimum financial cost for the node being evaluated from the priority queue in the same way that the algorithm above selects the minimum duration. However, almost all arcs have zero cost which results in two problems. First, routes through a charging station may be selected over routes that skip that charging station even when no charging takes place since both of those options would have zero financial cost. Second, the
performance of the algorithm is significantly degraded when so few arcs are preferred. This eliminates the main advantage of Dijkstra’s algorithm which pursues the most likely candidate paths first. Since so many paths are equal, all of them must be explored. The fix is simple. Preferring routes with lower cumulative duration when the financial cost is equal solves both problems.

Four changes are, therefore, made to the algorithm to optimize for financial cost. Lines 0, 2, and 6 are changed and line 5b is added after line 5, as follows. In line 5b, $c_{FC}^{\text{new}}$ represents the financial cost of the arc currently being evaluated and the cost per kWh and cost per minute are extracted from the node’s metadata.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initialize a label for node s (now with cumulative_financial_cost)</td>
</tr>
<tr>
<td></td>
<td>$L = { \text{cumulative_duration: } 0, \text{cumulative_financial_cost: } 0, \text{cumulative_energy: } 0, \text{previous_node: } \text{null}, \text{previous_node_index: } \text{null}, \text{current_node: } s, \text{current_index: } 1 }$. Create a priority queue for open nodes $L\text{temp} = [L]$ and for closed nodes $L\text{perm} = [\text{null}]$.</td>
</tr>
<tr>
<td>2</td>
<td>$L\text{cur} =$ next in the $L\text{temp}$ (i.e., either the smallest cumulative_financial_cost or cumulative_duration, depending on what the search is optimizing for)</td>
</tr>
<tr>
<td>5b</td>
<td>$c_{FC}^{\text{new}} = L\text{cur.cumulative_financial_cost} + ((ts - \text{SoC}) * \text{battery capacity} * \text{cost per kWh}) / 1,000 + (\text{charging duration} * \text{cost per min}) / 60$</td>
</tr>
<tr>
<td>6</td>
<td>Create a new label $L\text{new} =$ {</td>
</tr>
<tr>
<td></td>
<td>cumulative_duration: $L\text{cur.cumulative_duration} + \text{the duration of}$</td>
</tr>
<tr>
<td></td>
<td>cumulative_financial_cost: $L\text{cur.cumulative_financial_cost} + c_{FC}^{\text{new}}$</td>
</tr>
<tr>
<td></td>
<td>cumulative_energy: $c_{E}^{\text{new}}$,</td>
</tr>
<tr>
<td></td>
<td>previous_node: $L\text{cur.current_node}$,</td>
</tr>
<tr>
<td></td>
<td>previous_node_index: $L\text{cur.current_index}$,</td>
</tr>
<tr>
<td></td>
<td>current_node: the node $e$ goes to,</td>
</tr>
<tr>
<td></td>
<td>current_index: increment $L\text{cur.current_index}$ }</td>
</tr>
</tbody>
</table>

With this new set of data for each label, the algorithm can be used to find a path through the graph for either optimal time or financial cost. The implementation of this algorithm is provided in Appendix 2.
6. **SCENARIO STUDY**

To demonstrate the algorithm, a specific setting is assumed in which an individual EV driver needs to drive from point A to point B, both in the region of Sweden depicted in Figure 4. For many of these the route would consume more energy than the vehicle’s battery capacity. Therefore, one or more stops for recharging may be necessary to complete the trip. The driver desires to complete the trip with a minimum of time and cost.

Minimization of the financial cost of charging along EV routes requires some constraints. Taken to the extreme, an EV driver could use a solar panel and – eventually – reach their destination without paying for charging at all or just using free public charging stations. Here, charging is restricted to public charging infrastructure for which pricing data is available. Charging stations which allow free charging are included. A minimum charger capacity of 43kW is used.

6.1 **FASTEST ROUTE**

As described in Section 5.2, the search for the cost-optimal route is based on a single, fastest route for each origin and destination. That route is the fastest driving route for any vehicle, not just EVs. Detours to charging stations are added to that route. For Sweden and many other countries as determined by HERE, this does not pose a practical problem or shortcoming of this approach, as many charging stations are available across the country (HERE Developer, 2023).

To find the optimal path with charging stops, multiple pieces of data and software must be connected. A solution requires a sequence of road segments with associated energy requirements and travel time as well as charging stops with associated cost and duration. Selecting charging stations and their outlet and pricing information requires the base route selection. The base route and its associated energy use and duration requires a fastest path algorithm applied to road network data.

Multiple options for generating the fastest route exist. Paid solutions, including Google Maps, HERE, and TomTom, are available via application programming interface (API). These are integrated with proprietary road network data. A free, open source, reliable set of road network data is also available: Open Street Maps (OSM). The OSM project has been crowdsourcing geospatial data since 2004 and has an extensive ecosystem of contributors and tools (Brovelli & Zamboni, 2018; Ghosh & Bhattacharyya, 2020; Luxen & Vetter, 2011). OSM data for Sweden and other countries or continents can be downloaded from Geofabrik (Geofabrik Download Server, n.d.).

To generate the route based on the Open Street Maps data, the OSRM was selected because it is designed for use with OSM data and takes only milliseconds to produce routes even on continent-sized map data (Luxen & Vetter, 2011; Project OSRM, n.d.; Shamshad & Haq, 2020). It was configured using its default settings for car routing using contraction hierarchies. OSRM takes latitude and longitude for the origin and destination and returns turn-by-turn directions with details on each road segment including its distance and duration.

6.2 **ENERGY USE**

With the route selected, energy use about all road segments along the way can be generated. This is needed to determine how much charging is necessary to complete the route. Due to time constraints, elevation data was not integrated and all road segments are assumed to be flat. The area
of Sweden included in the experiment is fairly flat and not mountainous though the degree that this choice impacts the results is unknown.

Energy use for each route segment is based on the method from Genikomsakis & Mitrentsis (2017). That method was chosen because it is designed to be computationally efficient and it is validated to have a mean absolute percentage error of less than 4% as compared to the much more difficult to implement FASTSim. In the Genikomsakis & Mitrentsis method, energy use is built up from the force required at the wheels in order to move the vehicle at various speeds. It depends on vehicle mass, road gradient, current speed and acceleration, and other variables. An example vehicle, the same as used by Genikomsakis & Mitrentsis (2017), was used for the experiment. The amount of energy required at the wheels is provided by the motor through the vehicle powertrain. Loss of energy due to powertrain inefficiency is additional output from the motor. Similarly, efficiency loss in the motor is additional energy which must be provided by the battery and the roundtrip efficiency of the battery is also taken into account. For simplicity and due to time constraints, instantaneous acceleration is assumed. In other words, the experiment simulates that the vehicle is going exactly the average speed for the route segment for the entire route segment. There is no acceleration within each route segment.

The energy use of the accessories, including air conditioning system, dashboard and others, is estimated at a constant 300W for the duration of the route. With that, we can calculate the total energy use for each route segment.

The statistics as used in the experiment are available in Appendix 3 and the implementation of the energy use model is provided in Appendix 4.

6.3 CHARGING STATION DATA

Next, data about charging stations along the route is added. Options for adding charging station data include HERE and other paid APIs, Open Charge Map data, and data from ChargeFinder.com (ChargeFinder, 2023). The ChargeFinder API was chosen for its completeness and the availability of pricing for thousands of charging station outlets throughout Sweden. For charging outlets without price data, the station is not included as a potential stop. Due to computational constraints, discussed in more detail in Section 7, a target state step size of 50% is used. In other words, only target states of 50% and 100% are added to the search graph. For the overhead of stopping at a charging station – the arcs from \( c_k \) to \( o_k \) – a duration of five minutes is used.

6.4 SCENARIO PROCEDURE

With all of those pieces in place, we can construct the experiment. The following procedure is followed:

1. Randomly select origin/destination pairs within the area of Sweden shown in Figure 4.
2. Get the fastest route for that pair from OSRM.
3. Calculate the energy use for that route.
4. Get charging stations along the route and their charge speed and pricing information from ChargeFinder.
5. Construct the search graph.
6. Compute the financial-cost-optimal route.
7. Compute the time-optimal route.
8. Data for each step is saved in a SQLite database.

The implementation of the experiment is provided in Appendix 5.

Figure 4: A map of part of Sweden. In this scenario, all routes have an origin and destination within the region of Sweden shown. This map was created using tile images from Thunderforest and data from OpenStreetMap using Leaflet and Svelte Leaflet.
7. RESULTS AND ANALYSIS

A total of 247 routes were run through the experiment and successfully retrieved charging station data for the route. Nearly 60% of the 247 completed all the steps in the experiment’s scenario procedure while the remainder crashed during the first or second pass through the graph search due to limitations in system resources. In 160 routes, the search algorithm returned a result when searching for a financial-cost-optimized path and in nine a path to the destination which did not deplete the battery could not be found. In 113 of the 160, a positive financial cost was found and in 47, the cost to complete the route was zero because there were no charging stops. The experiment did not find a route that was completed at no cost because a stop at a free public charger was enough to complete the route. In 143 of the 160, the search algorithm also found a duration-optimized path. And in 95 routes, including 17 of the 160 where financial-cost-optimized paths were found, the priority queue grew larger than the system resources allowed which caused a heap overflow. All experiment runs were done on Hetzner CX51 Intel x86_64 servers with 32GB of RAM running Ubuntu 22.04.

7.1 ALGORITHM CORRECTNESS

To show that the financial cost graph search algorithm is working as intended, two initial checks are done. First, for the same route and graph, the duration-optimized search should never find a lower financial cost than the cost-optimized search. That is, the duration-optimized cost should be greater than or equal to the optimized cost. Indeed, that is what is found, as shown in Figure 5. Second and similarly, Figure 6 shows that cost-optimized duration is always greater than or equal to the optimized duration. The gray lines in these figures have a slope of one, where the costs are equal. All values in the experimental data are in the appropriate half of these graphs.

The 247 routes run through the experiment ranged from 21.1 km to 669.4 km with an average of 299.5 km and a median of 292.7 km. More details and statistics are presented in Table 1. The count row shows how many routes have data for the statistic in that column. Longer routes were more likely to unsuccessfully complete the path finding algorithms, due to a heap overflow. Longer

![Figure 5: A scatterplot of the financial cost of a route when duration optimized or cost optimized. For the same route, the duration-optimized cost is always greater than or equal to the cost-optimized cost.](image5)

![Figure 6: A scatterplot of the duration of a route when duration optimized or cost optimized. For the same route, the cost-optimized duration is always greater than or equal to the optimized duration.](image6)
routes are associated with more need for charging stops and more charging stations along the route. Table 2 shows the difference between routes which resulted in successful paths for both cost- and duration-optimized searches and those which resulted in a heap overflow. The average route which caused a heap overflow had a duration of 20 185 seconds (more than 5.6 hours) and a distance of 412.5 km. That was almost double those for successful routes which were 11 987 seconds (3.3 hours) and 220.7 km, respectively. This aligns with the intuitive understanding that longer routes result in larger graphs and require more computational resources to analyze. And as shown in Figure 7, financial cost goes up on longer routes, as expected.

### Table 1: Summary statistics for all routes in the experiment

<table>
<thead>
<tr>
<th></th>
<th>Route Duration (s)</th>
<th>Route Distance (m)</th>
<th>Total Power (W)</th>
<th>Optimized Cost (SEK)</th>
<th>Cost-Optimized Duration (s)</th>
<th>Duration-Optimized Cost (SEK)</th>
<th>Optimized Duration (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>247.08</td>
<td>247.08</td>
<td>247.08</td>
<td>168.08</td>
<td>143.08</td>
<td>143.08</td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>15 481,63</td>
<td>299 487,54</td>
<td>34 790,15</td>
<td>77,68</td>
<td>16 086,38</td>
<td>83,76</td>
<td>14 687,47</td>
</tr>
<tr>
<td>std deviation</td>
<td>6 846,72</td>
<td>145 236,04</td>
<td>17 952,19</td>
<td>75,00</td>
<td>8 318,26</td>
<td>84,43</td>
<td>7 578,21</td>
</tr>
<tr>
<td>min</td>
<td>2 832,68</td>
<td>21 143,88</td>
<td>2 813,49</td>
<td>0,00</td>
<td>2 122,48</td>
<td>0,00</td>
<td>2 122,48</td>
</tr>
<tr>
<td>25 %</td>
<td>18 871,08</td>
<td>184 979,88</td>
<td>21 145,11</td>
<td>0,00</td>
<td>9 391,58</td>
<td>0,00</td>
<td>8 833,28</td>
</tr>
<tr>
<td>50 %</td>
<td>15 138,78</td>
<td>292 695,98</td>
<td>33 601,04</td>
<td>63,94</td>
<td>14 584,56</td>
<td>61,54</td>
<td>13 368,56</td>
</tr>
<tr>
<td>75 %</td>
<td>20 333,65</td>
<td>388 462,98</td>
<td>45 709,59</td>
<td>121,58</td>
<td>21 498,19</td>
<td>134,16</td>
<td>19 398,03</td>
</tr>
<tr>
<td>max</td>
<td>33 354,78</td>
<td>669 441,68</td>
<td>87 259,95</td>
<td>319,18</td>
<td>37 951,91</td>
<td>315,18</td>
<td>33 709,71</td>
</tr>
</tbody>
</table>

### Table 2: Summary statistics separated by whether the algorithm found cost-optimal and duration-optimal paths

<table>
<thead>
<tr>
<th></th>
<th>Routes Found (143 Routes)</th>
<th>Heap Overflows (95 Routes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Route Duration (s)</td>
<td>Route Distance (m)</td>
</tr>
<tr>
<td>mean</td>
<td>11 987,39</td>
<td>220 747,38</td>
</tr>
<tr>
<td>std deviation</td>
<td>5 577,33</td>
<td>106 914,87</td>
</tr>
<tr>
<td>min</td>
<td>2 832,68</td>
<td>21 143,88</td>
</tr>
<tr>
<td>25 %</td>
<td>8 819,38</td>
<td>145 816,95</td>
</tr>
<tr>
<td>50 %</td>
<td>11 019,28</td>
<td>215 009,58</td>
</tr>
<tr>
<td>75 %</td>
<td>15 480,30</td>
<td>293 947,55</td>
</tr>
<tr>
<td>max</td>
<td>26 068,38</td>
<td>471 896,58</td>
</tr>
</tbody>
</table>
7.2 COST AND DURATION DIFFERENCES

The average of the 143 routes which completed both the cost- and duration-optimized searches was a cost of 83,78 SEK when optimizing for duration and 71,17 SEK when optimizing for cost, a savings of 15%. That includes 47 routes with zero cost. The average of the 96 routes with a non-zero cost was 124,79 SEK when optimizing for duration and 106,02 SEK when optimizing for cost, still a savings of 15%. Those savings are a trade-off for extra time spent on the trip overall. The average cost-optimized duration was 15 411 seconds (4.28 hours) and the optimized duration averaged 14 687 seconds (4.08 hours). That is an increase in trip duration of 4.9%. More details on the distribution of costs and durations are presented in Table 3. The cost per kWh paid is higher on duration-optimized routes which have essentially a minimum cost of 6 SEK per kWh in this data, as shown in Figure 8. There is no pattern of minimum cost per kWh for cost-optimized routes and six of the 96 are below 5 SEK per kWh.

Table 3: Summary statistics separated by cost-optimal and duration-optimal paths for the 143 routes which found both

<table>
<thead>
<tr>
<th></th>
<th>Optimized Cost (SEK)</th>
<th>Cost-Optimized Duration (s)</th>
<th>Duration-Optimized Cost (SEK)</th>
<th>Optimized Duration (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>71,17</td>
<td>15 411,06</td>
<td>83,78</td>
<td>14 687,47</td>
</tr>
<tr>
<td>std</td>
<td>71,21</td>
<td>15 818,74</td>
<td>84,43</td>
<td>15 778,21</td>
</tr>
<tr>
<td>min</td>
<td>0,00</td>
<td>2 122,40</td>
<td>0,00</td>
<td>2 122,40</td>
</tr>
<tr>
<td>25 %</td>
<td>0,00</td>
<td>8 833,20</td>
<td>0,00</td>
<td>8 833,20</td>
</tr>
<tr>
<td>50 %</td>
<td>54,48</td>
<td>13 821,71</td>
<td>61,54</td>
<td>13 360,56</td>
</tr>
<tr>
<td>75 %</td>
<td>117,95</td>
<td>20 909,18</td>
<td>134,16</td>
<td>19 398,03</td>
</tr>
<tr>
<td>max</td>
<td>281,72</td>
<td>35 021,23</td>
<td>315,18</td>
<td>33 709,71</td>
</tr>
</tbody>
</table>
Potential savings were available on all route lengths long enough to require a charging stop. Nominal savings increase with more charging stops as shown in Figure 9, while shorter routes have the largest range of percentage savings, as shown in Figure 10. As shorter routes are more likely to rely on a single charging stop, the available pricing for that stop can have a larger effect on the percentage savings. For longer routes, there is less chance that all of the charging stops will have large savings available. Figure 11 shows the differences in time spent charging for the different optimizations. That difference accounts for the majority of the additional trip durations when optimizing for cost, as shown in Figure 12. If the search graph was expanded to allow for more out of the way chargers, this relationship might change.

Figure 8: The cost per kWh on duration-optimized vs cost-optimized paths. The cost-optimized cost per kWh is always lower than that for duration-optimized routes. There is no clear trend when accounting for the number of charging stops made.

Figure 9: Nominal savings on cost-optimized routes versus number of charging stops. Nominal savings has an increased range with more charging stops and greater distance.

Figure 10: Percentage savings on cost-optimized routes as compared to fastest routes versus route distance. Shorter routes have more variability in percentage savings.
7.3 COST-DURATION TRADE-OFF

So, the financial-cost optimization method can find savings in exchange for increased trip duration. The average savings was 139.4 SEK per hour for routes which made any change from the duration-optimized path. In other words, for each hour of additional travel time as compared to the fastest path, there was an average savings of 139.4 SEK. More details are shown in Table 4. Are those savings worthwhile and worth more to the driver than the time? Figure 13 shows the relationship.
between the average hourly pre-tax wages in Sweden of 187,90 SEK (Statistikmyndigheten SCB, n.d.) with the number of Swedish Kronor saved per extra hour spent on these trips. After 32% taxes, that average wage is 127.77 SEK per hour, 9% less than the average savings per hour. This is only one way to look at the value. This time is mostly spent at charging stations, as shown in Figure 12. The time could be spent communicating with friends or family, reading, or taking advantage of a nearby point of interest, in some cases. It is, therefore, not time “lost” for everyone in these situations.

Table 4: Cost savings per hour of additional trip duration

<table>
<thead>
<tr>
<th>Savings per Extra Hour of Trip Duration (SEK)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>71.00</td>
</tr>
<tr>
<td>mean</td>
<td>136.41</td>
</tr>
<tr>
<td>std</td>
<td>211.32</td>
</tr>
<tr>
<td>min</td>
<td>3.45</td>
</tr>
<tr>
<td>25 %</td>
<td>21.84</td>
</tr>
<tr>
<td>50 %</td>
<td>62.01</td>
</tr>
<tr>
<td>75 %</td>
<td>132.12</td>
</tr>
<tr>
<td>max</td>
<td>1,165.57</td>
</tr>
</tbody>
</table>

Figure 13: Savings per hour versus route distance. The average hourly wage in Sweden, 187.90 SEK, is similar to the savings per hour found in the experiment.
8. DISCUSSION

The financial-cost optimization algorithm works. It answers the first research question: How can we optimize a driver’s financial cost of charging at public charging stations for electric vehicle routing? And with that method for answering the first research questions, we can collect data in order to answer the second: What is the variation in financial cost and trip duration between the fastest and cost-optimized routes in Sweden based on current charging station pricing? Experimental results show that potential cost savings are worthy of consideration.

Many aspects of the solution are only relevant or even possible because of recent developments. Most of the world does not yet have the number of charging stations available to make this algorithm useful (HERE Developer, 2023), but an increasing share does. The availability of data about the charging stations on the route is expanding quickly and tools to use that data most effectively are needed. The use of digital tools in transportation and routing is not new. Consumers are accustomed to and welcoming of digitalization in this part of their lives. Solving this problem helps consumers understand how the electrification of their personal transportation can fit into their life and budget. That increases the sustainability of personal vehicles and the lifestyles of those who use them. And the scale of the computation problem – frequently involving tens of millions of data points – is not possible without a digital solution.

8.1 ALGORITHM EFFICIENCY INVESTIGATIONS

Even with digital tools and significant resources, the problem is hard. Further innovation is required. The computational requirements to run the algorithm on real routes are significant and further investigation is necessary for its real-world use. Two ideas were investigated. Huber & Bogenberger (2015) have a more efficient version of their algorithm which they note “does not necessarily return time-optimal solutions” and that it might not find any solution even though a solution does exist. Because the computational resources needed to run the first version of the algorithm were so large, including 12+ hour running times on a MacBook Air M1 with 8GB RAM, the use of the more efficient version was explored. Initial testing did show fast results measured in milliseconds rather than minutes or hours. However, in many of the first few, short test routes, no solution was found or the time-optimal route took longer than a financial-cost-optimal route. Therefore, further testing was not conducted using that version of the algorithm. Its results were too unreliable.

Next, the potential of bit packing the labels was attempted. With bit packing, precise control of memory usage is taken from the programming language’s interpreter taking advantage of our understanding of the precision requirements for each statistic about the state of the vehicle and its battery along the route. For example, exactly 20 bits can be used for storing the duration data which is enough for routes up to 29 hours measured in tenths of a second. With one fewer bit, accurate data would be restricted to routes of 14.5 hours or less, which may not be enough for some routes. There are many data points associated with each label and the loss of precision in financial cost, duration, and energy use is not appropriate here. A revised label structure using six 32-bit unsigned integers was implemented. Again, early testing on short routes and small search graphs did not show promising results. While the output of the algorithm was the same as the non-bit-packed version, the time for completion was ten to twenty times longer. The memory usage may have
improved and the solution may be useful in some circumstances. Other implementations using more efficient programming languages than JavaScript are likely to be more fruitful.

After those investigations of the computational requirements and tests of improving the algorithm, ways of reducing the size of the search graph looked like the way forward. Tests on the full map of Sweden were reduced iteratively until the final map was chosen, as shown in Figure 4. The graph was further reduced by increasing the step size between target states of charge from 10% to 20, 25, 33, and finally 50% used in the results discussed. Similarly, the minimum charger capacity was increased from 22 kW to 43 kW. With those changes, random routes could be tried without wasting time where the vast majority of random routes result in heap overflows.

8.2 FUTURE WORK

Further reductions in the size of the search graph might improve or even guarantee success under the constraints of the computer systems used. However, the accuracy of the results would be degraded. A primary area of work that could improve confidence in the cost savings discussed here is to test the impact of changes in the step size and charger capacity. There are many other potential areas of exploration. Improving accuracy of the estimates of energy use along the route would integrate well. It would increase confidence that the charging stops recommended are well-timed and maximally productive. The use of drive cycles are worthy of consideration in this regard.

As the number of charging stations continues to grow, the computational complexity of the problem will increase as it has in the FRVRP facing ICE vehicles. Heuristics have been used to increase search performance that can be used in real-world, real-time applications while pursuing outcomes that are as close as possible to the optimal solution. If reasonable performance is possible with the current number of charging stations by implementing the search algorithm in C++, Rust, or another performance-optimized programming language, the number of charging stations may outpace those optimizations. In that case, the use of heuristics should be pursued in the same way as it has for the FRVRP.

Extending the algorithm to include multiple objectives or to allow drivers to configure personal refinements could make it a more appealing tool. One such possible objective is to aim for charging stops at appealing locations because they are associated with points of interest. This area could build on work by the vehicle parts and technology company Bosch (Luetttin et al., 2019). Another possible multiple-objective algorithm could be developed in which charging stops are recommended based on less overall energy use for the route or by preferring stops at chargers with renewable energy sources. These types of integrations and improvements would enhance the environmental sustainability of the solutions and the users’ lifestyles.

Research on incorporating this kind of work into consumer route-planning applications should also be considered. This information is only useful if it has a chance to impact the decisions of potential and current EV owners. The extent to which drivers will alter their route for cost savings or other objectives and which types of consumers are most likely to do so are key questions. The user experience of such applications could also be a fruitful area of examination. In particular, how much control over what the search algorithm is optimizing for given multiple variables and compromises is the right amount, an amount that leads to satisfaction and confidence and not a sense of being overwhelmed by choice?
9. **CONCLUSION**

This project has shown that an algorithm can be developed to find the lowest financial cost path for charging an electric vehicle along a route and that it can handle the constraints of the battery. That path includes where and how much to charge, helping the driver choose from the possible charging stations along their route, in order to optimize for cost. It is adaptable to different available computational resources by changing some of the parameters in the construction of the search graph.

The algorithm was demonstrated through random selection of routes within Sweden. It was shown that optimizing for cost results in a path that always has less than or equal cost as compared to the path when optimizing for the shortest duration. Similarly, it was shown that those cost-optimal paths always have greater than or equal durations as compared to the duration-optimized paths along the same route. The development of that algorithm and its demonstration answers the first research question on how to optimize an electric vehicle driver's cost of charging at public charging stations.

For the routes in the experiment, the total price for charging for cost-optimal routes was 15% lower than that for duration-optimal paths. Those savings are a trade-off for the extra time to complete the route, and most of that extra time is spent at charging stations. For each hour of additional trip duration due to optimizing for cost, the average savings was 139 SEK. That is enough that it may be worthwhile for many drivers. That is the answer to the second research question on the difference in cost and duration when optimizing for cost versus time.

Because the potential savings are significant and worth thinking about, there is a lot of opportunity to build on this work. Researchers and practitioners can improve the performance and reduce resource requirements. Entrepreneurs can integrate this kind of feature into route planning software. Consumer knowledge about the potential cost differences will encourage charging station owners to compete for customers in new ways. And if consumers have access to route planning software that incorporates this kind of algorithm, price competition will reduce consumer costs. With lower operating costs, more consumers will want to transition from internal combustion engine vehicles to electric, improving the sustainability of the sector. This project is one step forward for research on electric vehicle operating costs and one step in reducing consumer uncertainty about their transportation choices.
ACKNOWLEDGMENTS

For the software development part of this project, I would not have been successful without a tremendous amount of work from others. Andrei Kashcha’s work on ngraph and the related libraries and examples pointed me in the right direction on many issues and his library for graph construction was a big help. The documentation, tooling, and data around the Open Street Map and Open Source Routing Machine projects were occasionally frustrating but I cannot imagine this project without them.

I found it immensely satisfying to implement the work of Genikomsakis & Mitrentsis, converting data about the road network into energy usage for the vehicle. Their paper took me from essentially no understanding to a vibrant, flowing mental model of the energy flows as I implemented their formulas and unit tested them. I want to thank them for their clear and interesting paper.

I might still be trying to decide on a research topic without the guidance of my supervisors Johan Holmgren and Henrik Fredriksson. I am very thankful for their ideas on where to look and how to shape this work and for their patience as I figured it out.

And I cannot thank Blythe, Vivian, and Stellan, and Tessa enough for their love and support.
REFERENCES


Transactions on Smart Grid, 12(4), 3278–3288. https://doi.org/10.1109/TSG.2021.3053026
Gao, L., & Chu, F. (2022, February 23). An efficient bi-objective personalized route planning over two-fold time-dependent road networks. 23ème congrès annuel de la Société Française de Recherche Opérationnelle et d’Aide à la Décision. https://hal.science/hal-03595240
Waze is adding electric vehicle charging stations to its route planning tool. The Verge.
https://www.theverge.com/2023/3/14/23640203/waze-ev-charging-navigation-tool-google


Lamb, W. F., Grubb, M., Diluiso, F., & Minx, J. C. (2022). Countries with sustained greenhouse...


Appendices

All code is available at https://github.com/mattlehrer/ev-routing

Appendix 1: Graph Construction

```javascript
/**
* Create a graph data structure from a route and a list of charging stations.
* @param intersections the intersections along the route
* @param stations the list of charging stations
* @param overheadDuration the time it takes to enter/exit the vehicle, set up payment, etc.
* @returns the constructed graph
*/
export function createGraphFromRouteAndChargingStations({
  intersections,
  stations,
  overheadDuration = 5 * 60, // 5 minutes
}) {
  const g = newGraph<NodeType>();
  // add a node for the beginning of the route
  g.addNode('s');
  let previous = 's';
  let previousLonLat = intersections[0].intersection.location;

  const sortedStations = stations.map((station) => {
    ...station,
    closest: findClosestIntersectionOnRouteToChargingStation({
      intersections,
      station
    }),
  }).sort((a, b) => a.closest.properties.featureIndex - b.closest.properties.featureIndex);

  // for each station,
  sortedStations.forEach((station, i) => {
    // find the closest intersection on the route
    const closest = station.closest;
    // add a node for that intersection - ai
    g.addNode(`a${i}`, { type: 'a', coordinates: closest.geometry.coordinates });
    // console.log({ closest, intersections });

    const statsFromPrevToA = cumulativeStatsAlongRoute({
      intersections,
      start: previousLonLat,
      end: closest.geometry.coordinates,
    });
  });
```

38
// add an edge from the previous station to this intersection
g.addLink(previous, `a${i}`, {
  distance: statsFromPrevToA.distance,
  duration: statsFromPrevToA.duration,
  power: statsFromPrevToA.power,
  financial: 0,
});
previous = `a${i}`;

// add a node for the station - ii
g.addNode(`i${i}`, { type: 'i', station });

// add an edge from intersection to station
const distanceToStation = closest.properties.distanceToPoint; // as the crow flies, should compute a route
const duration = (distanceToStation * 60 * 60) / 30000; // 30km/h
// TODO: base on route
const power = calc_route_segment_battery_power_flow({
  vehicle: TestVehicle,
  distance: distanceToStation, // as the crow flies, should compute a route
  duration: duration,
  elevation_start: 0,
  elevation_end: 0,
  density_of_air: 1.225,
});
power = calc_route_segment_battery_power_flow({
  vehicle: TestVehicle,
  distance: distanceToStation, // as the crow flies, should compute a route
  duration: duration,
  elevation_start: 0,
  elevation_end: 0,
  density_of_air: 1.225,
});

// add a node for oi that will have edges from all battery levels and back to the route
// TODO: factor in multiple outlets per station, different charging rates and prices
for (let j = 10; j <= 100; j += 10) {
  const chargeLevel = `c${i}-${j}`;
  // add a node for each charge level
  g.addNode(chargeLevel), { type: 'c', chargeLevel: j };
// add a node for bi that has edges from ai and oi, same location as ai
g.addNode(`b${i}`, { type: 'b', coordinates: closest.geometry.coordinates });

g.addLink(`o${i}`, `b${i}`, {
distance: closest.properties.distanceToPoint, // as the crow flies, should compute a route
duration: (closest.properties.distanceToPoint * 60 * 60) / 30000, // figure out an estimate
// TODO: base on route
distance: closest.properties.distanceToPoint, // as the crow flies, should compute a route
duration: (closest.properties.distanceToPoint * 60 * 60) / 30000, // figure out an estimate
power: calc_route_segment_battery_power_flow({
distance: closest.properties.distanceToPoint, // as the crow flies, should compute a route
duration: (closest.properties.distanceToPoint * 60 * 60) / 30000, // figure out an estimate
elevation_start: 0,
elevation_end: 0,
vehicle: TestVehicle,
density_of_air: 1.225,
}),
financial: 0,
});

g.addLink(`a${i}`, `b${i}`, {
distance: 0,
duration: 0,
power: 0,
financial: 0,
});

previous = `b${i}`;
previousLonLat = closest.geometry.coordinates;
});

// add node for destination
g.addNode('d');

const statsFromPrevToA = cumulativeStatsAlongRoute({
intersections,
start: previousLonLat,
end: intersections[intersections.length - 1].intersection.location,
});

// add edge to the destination from bn
g.addLink(previous, 'd', {
distance: statsFromPrevToA.distance,
duration: statsFromPrevToA.duration,
power: statsFromPrevToA.power,
financial: 0,
});

return g;
}
/**
 * Find the closest intersection from a list of intersections to a charging station.
 * @param intersections a list of intersections with LatLng coordinates
 * @param station a charging station with LatLng coordinates
 */
* @returns the intersection closest to the station  
*/

export function findClosestIntersectionOnRouteToChargingStation({
    intersections,
    station,
}: {
    intersections: ReturnType<typeof convertRouteFromStepsToIntersections>;
    station: ChargingStationBasic;
}) {
    const { latitude, longitude } = station.location;
    const points = featureCollection(
        intersections.map((intersection) => point(intersection.intersection.location)),
    );

    const nearestPointToStation = nearestPoint([longitude, latitude], points);
    return nearestPointToStation;
}

/**
 * Sum stats about a route between two positions.
 * @param route The route to search
 * @param start The start position
 * @param end The end position
 * @returns An object with the distance, in meters, duration, in s, and the power use, in Wh
 */

export function cumulativeStatsAlongRoute({
    intersections,
    start,
    end,
}: {
    intersections: ReturnType<typeof convertRouteFromStepsToIntersections>;
    start: Position;
    end: Position;
}): { distance: number; duration: number; power: number } {
    let distance = 0;
    let duration = 0;
    let power = 0;

    // console.log({ start, end });

    if (start[0] === end[0] && start[1] === end[1]) {
        return { distance, duration, power };  
    }

    let isAfterStart = false;

    for (const intersection of intersections) {
        if (!isAfterStart) {
            // if the intersection is the start position,
            if (intersection.intersection.location[0] === start[0] &&
                intersection.intersection.location[1] === start[1]) {
                // set the flag to true
                isAfterStart = true;
                // step.distance is the distance to the next maneuver, so we need to start
adding now
    distance += intersection.distance;
duration += intersection.duration;
    power += intersection.power ? intersection.power : 0;
}
} else {
    // if the intersection is the end position,
    if (intersection.intersection.location[0] === end[0] &&
        intersection.intersection.location[1] === end[1]) {
        // and break out of the loop and return the stats
        // break;
        return { distance, duration, power };
    } else {
        distance += intersection.distance;
duration += intersection.duration;
power += intersection.power ? intersection.power : 0;
    }
}
}

if (!isAfterStart) {
    throw new Error('Start position not found on route');
} else {
    throw new Error('End position not found on route');
}


**type** NodeLabel = {
cumulativeDuration: number;
cumulativePower: number;
cumulativeFinancialCost: number;
cumulativeDistance: number;
precedingNode: string | null;
prevLabelIndex: number; // "which of the labels belonging to the preceding node
is relevant for getting the currently considered label"
currentNode: string;
currentLabelIndex: number;
};

**type** NodeType =
| {  
|   type: 's' | 'd' | 'a' | 'b';
|   coordinates: Coordinate;
|   |
|   type: 'i';
|   station: ChargingStationBasic;
| |
|   type: 'c';
|   chargeLevel: number;
| |
|   type: 'o';
| |}
/**
 * Find a path through the road network graph optimizing for duration or financial cost
 * @param g the graph
 * @param type the type of cost function to optimize for
 * @param initialSoC the initial state of charge of the vehicle, in kWh
 * @param s the starting node
 * @param d the destination node
 * @param minSoC the minimum state of charge to allow, in kWh
 * @param batteryCapacity the battery capacity of the vehicle, in kWh
 * @returns the path
 */

export function findPathInGraphWithCostFunction({
g, type, initialSoC, s = 's', d = 'd',
minSoC = 0.1 * TestVehicle.battery_capacity,
batteryCapacity = TestVehicle.battery_capacity,
fastMode = false,
}) {
  const lTemp =
    type === 'cumulativeFinancialCost'
    ? new Heap < NodeLabel > (financialCostComparator) :
    new Heap < NodeLabel > ((a, b) => a[type] - b[type]); // opened nodes
  const lPerm =
    type === 'cumulativeFinancialCost'
    ? new Heap < NodeLabel > (financialCostComparator) :
    new Heap < NodeLabel > ((a, b) => a[type] - b[type]); // closed nodes
  let hasReachedDestination = false;

  lTemp.add({
    cumulativeDuration: 0,
    cumulativeDistance: 0,
    cumulativePower: 0,
    cumulativeFinancialCost: 0,
    chargingDuration: 0,
    chargingKw: 0,
    chargingStops: 0,
    precedingNode: null,
    prevLabelIndex: '',
  })

  while (!hasReachedDestination) {
    const node = lTemp.pop();
    if (node.prevLabelIndex !== '') {
      hasReachedDestination = true;
      break;
    }
    lPerm.add(node);
    for (const neighbor of g[node.label].neighbors) {
      const newNode = {...node, type: type};
      newNode[type] += neighbor[type];
      newNode.precedingNode = node;
      newNode.prevLabelIndex = node.prevLabelIndex;
      if (newNode[type] <= 0) {
        hasReachedDestination = true;
        break;
      }
      if (newNode[type] < minSoC) {
        hasReachedDestination = true;
        break;
      }
      if (newNode[type] < initialSoC) {
        hasReachedDestination = true;
        break;
      }
      const newNodeType = type === 'cumulativeDuration' ? 'cumulativeFinancialCost' : 'cumulativeDuration';
      lTemp.add(newNode);
    }
  }

  return lTemp[0];
}
let lCurrent: NodeLabel | undefined = undefined;

// line 1 from Huber 2015 Algorithm A
while (lTemp.size() > 0 && !hasReachedDestination) {
  // lines 2 & 3
  lCurrent = lTemp.pop();
  if (!lCurrent) throw new Error('lCurrent is undefined');
  lPerm.add(lCurrent);
  if (!(lPerm.size() % 1048576)) {
    console.log(`Temp Labels: ${lTemp.size()} | Perm Labels: ${lPerm.size()}`);
  }
  if (lCurrent.currentNode === d) {
    hasReachedDestination = true;
  }
  // line 4: for all outgoing edges of lCurrent.currentNode
  g.forEachLinkedNode(lCurrent.currentNode, (node, link) => {
    const edge = link.data;
    if (!lCurrent) throw new Error('lCurrent is undefined');
    // line 7
    const newLabel = {
      currentNode: String(node.id),
      cumulativeDuration: lCurrent.cumulativeDuration,
      cumulativeDistance: lCurrent.cumulativeDistance + edge.distance,
      cumulativePower: lCurrent.cumulativePower,
      cumulativeFinancialCost: lCurrent.cumulativeFinancialCost,
      chargingDuration: lCurrent.chargingDuration,
      chargingKw: lCurrent.chargingKw,
      chargingStops: lCurrent.chargingStops,
      precedingNode: lCurrent.currentNode,
      prevLabelIndex: lCurrent.currentLabelIndex,
      currentLabelIndex: uid(),
    };
    const soc = initialSoC - lCurrent.cumulativePower; // in kWh
    // lines 5: calculate the duration for this edge
    let edgeDuration = 0;
    // and 6: calculate the change in power for this edge
    let edgePower = 0;

    // calculations are based on the edge's end node type
    if (['a', 'i', 'b', 'd'].includes(node.data.type)) {
      // duration generated by OSRM
      edgeDuration = edge.duration ? 0;
      // and 6: calculate the change in power for this edge
      edgePower = 0;
    }
    else if (node.data.type === 'c') {
      edgeDuration = calculateChargingDuration({
        soc, capacity: node.data.current, targetSoc: node.data.chargeLevel,
      });
      edgePower = edge.power;
    }
    // power from previous calculations based on road/vehicle data
  });
}
batteryCapacity,
});

// charge up (negative power)
if (edgeDuration > 0) {
    edgePower = -1_000 * ((node.data.chargeLevel / 100) * batteryCapacity
      - soc);
    newLabel.chargingDuration += edgeDuration;
    newLabel.chargingKw += -edgePower / 1_000;
    newLabel.chargingStops += 1;

    // calculate the financial cost of charging
    newLabel.cumulativeFinancialCost =
      lCurrent.cumulativeFinancialCost +
      (edgeDuration * (node.data.costMin ? 0) / 60 -
       (edgePower * (node.data.costKwh ? 0) / 1_000;

    if (node.data.costKwh === 0) console.log('costKwh is 0', node.data);
} else if (node.data.type === 'o') {
    // duration is fixed as overhead in graph construction
    if (!edge.duration)
        throw new Error(
            'Edge duration is undefined and should have been added in graph
            construction as overhead amount: ' + JSON.stringify(
                edge,
            ));

    edgeDuration = edge.duration ? 0;
}

newLabel.cumulativeDuration = lCurrent.cumulativeDuration + edgeDuration;
newLabel.cumulativePower = lCurrent.cumulativePower + edgePower / 1_000;

// line 8
if (initialSoC - newLabel.cumulativePower >= minSoC) {
    if (fastMode) {
        // Equation box B
        const temp = lTemp.toArray();
        const tempMatches = temp.filter((l) => l.currentNode ===
          newLabel.currentNode);
        if (tempMatches.every((l) => l[type] > newLabel[type])) {
            const permMatches = lPerm.toArray();
            const permMatches = temp.filter((l) => l.currentNode ===
              newLabel.currentNode);
            if (permMatches.every((l) => l[type] > newLabel[type])) {
                // lines 10 and 11
                const newTemp = temp.filter((l) => l.currentNode !==
                  newLabel.currentNode);
                lTemp.init(newTemp);
                lTemp.add(newLabel);
            }
        }
    } else {
        // line 9
        lTemp.add(newLabel);
    }
}
true, // only outgoing edges
); // line 11, end of for loop
} // line 12, end of while loop
console.log();

// line 13
if (lCurrent ? .currentNode === d) {
    const perm = lPerm.toArray();
    const path = [];
    path.unshift(lCurrent);
    let current = lCurrent;
    while (current ? .precedingNode) {
        const prev = perm.find(
            p =>
                p.currentNode === current ? .precedingNode &&
                p.currentLabelIndex === current ? .prevLabelIndex,
        );
        if (!prev) throw new Error('prev is undefined');
        path.unshift(prev);
        current = prev;
    }
    console.log({
        path,
        d: path[path.length - 1]
    });
    lTemp.clear();
    lPerm.clear();
    return path;
} else {
    const perm = lPerm.toArray().slice(-10);
    console.log({
        perm
    });
    lTemp.clear();
    lPerm.clear();
    // line 14: "No feasible solution found."
    console.error('No feasible solution found. ');
    return null;
}
frontal_area: 2.19, // in cubic meters
drag_coefficient: 0.29,
rolling_resistance_coefficient: 0.008,
mass_correction_factor: 0.05,
accessory_power_draw: 300.0, // in watts

// motor
motor_type: 'induction_motor',
p_motor_rated,
norm_factor,

// battery
battery_capacity: 24, // 24 kWh
rte: 0.95,

// regenerative speed bounds
u1: 1.39, // 1.39 m/s = 5 km/h
u2: 4.72, // 4.72 m/s = 17 km/h

// transmission
gear_efficiency: 0.97,

};

Appendix 4: Functions for force and power at the wheels

/**
 * force of aerodynamic drag in Newtons
 * @param rho density of air in kg/m^3
 * @param c_d drag coefficient
 * @param area frontal area in m^2
 * @param v velocity in m/s
 * @returns force of aerodynamic drag in Newtons
 */
export const calc_f_ad = ({
  rho,
  c_d,
  area,
  v,
}): { number => {
  return 0.5 * rho * c_d * area * v ** 2;
}};

/**
 * force of rolling resistance in Newtons
 * @param mu_rr rolling resistance coefficient
 * @param m mass in kg
 * @param theta slope angle in radians
 * @param g gravitational acceleration in m/s^2
 * @returns force of rolling resistance in Newtons
 */
export const calc_f_rr = ({


mu_rr, m, theta, g = 9.81, { 
  mu_rr: number; // rolling resistance coefficient 
  m: number; // mass in kg 
  theta: number; // slope angle in radians 
  g?: number; // gravitational acceleration in m/s^2 
}): number => { 
  return mu_rr * m * g * Math.cos(theta); 
};

/**
 * hill climbing force in Newtons
 * @param m mass in kg
 * @param theta slope angle in radians
 * @param g gravitational acceleration in m/s^2
 * @returns hill climbing force in Newtons
 */
export const calc_f_hc = ({ m, theta, g = 9.81, }): { 
  m: number; // mass in kg 
  theta: number; // slope angle in radians 
  g?: number; // gravitational acceleration in m/s^2 
}): number => { 
  return m * g * Math.sin(theta); 
};

/**
 * force of linear acceleration in Newtons
 * @param m mass in kg
 * @param a acceleration in m/s^2
 * @returns force of linear acceleration in Newtons
 */
export const calc_f_la = ({ m, a, }): { 
  m: number; // mass in kg 
  a: number; // acceleration in m/s^2 
}): number => { 
  return m * a; 
};

/**
 * inertial force in Newtons
 * @param c_i mass correction factor for rotational inertia acceleration
 * @param m mass in kg
 * @param a acceleration in m/s^2
 * @returns inertial force in Newtons
 */
export const calc_f_omega_a = ({ c_i,}
m,
a,
}: {
c_i: number; // mass correction factor for rotational inertia acceleration
m: number; // mass in kg
a: number; // acceleration in m/s^2
}): number => {
  return c_i * m * a;
};

/**
 * total tractive effort at the wheels in Newtons
 * @param f_ad force of aerodynamic drag in Newtons
 * @param f_rr force of rolling resistance in Newtons
 * @param f_HC hill climbing force in Newtons
 * @param f_lA force of linear acceleration in Newtons
 * @param f_omega_a inertial force in Newtons
 * @returns total tractive effort at the wheels in Newtons
 */
extport const calc_f_te = ({
f_ad,
f_rr,
f_HC,
f_lA,
f_omega_a,
}): {
  f_ad: number; // force of aerodynamic drag in Newtons
  f_rr: number; // force of rolling resistance in Newtons
  f_HC: number; // hill climbing force in Newtons
  f_lA: number; // force of linear acceleration in Newtons
  f_omega_a: number; // inertial force in Newtons
}): number => {
  return f_ad + f_rr + f_HC + f_lA + f_omega_a;
};

/**
 * The traction power to drive the vehicle at speed u in Watts
 * @param f_te total tractive effort at the wheels in Newtons
 * @param u car velocity in m/s
 * @returns traction power at the wheels in Watts
 */
extport const calc_p_te = ({
f_te,
u,
}): {
  f_te: number; // traction power at the wheels in Newtons
  u: number; // car velocity in m/s
}): number => {
  return f_te * u;
};

// Functions related to the transmission and powertrain

/**
 * angular motor speed in rad/s
 * @param g_ratio gear ratio of the transmission system
 * @param uAngular motor speed in rad/s
 */
* @param r_wheel wheel radius in m
* @returns linear motor speed in m/s
*/
export const calc_omega_motor = ({
g_ratio,
u_angular,
r_wheel,
}): {
g_ratio: number;
u_angular: number;
r_wheel: number;
}) => {
const u_linear = u_angular * r_wheel;
const f_ad = g_ratio * u_linear;
return f_ad;
};

/**
* mechanical power from the motor in Watts
* @param traction_power power from the motor in Watts
* @param n_gear gear efficiency of the transmission system
* @returns mechanical power in Watts
*/
export const calc_p_motor_out = ({
traction_power,
n_gear,
}): {
traction_power: number;
n_gear: number;
}) => {
if (traction_power < 0) {
    return traction_power * n_gear;
} else {
    return traction_power / n_gear;
}
};

/**
* motor output torque in Nm
* @param p_motor_out mechanical power from the motor in Watts
* @param omega_motor_speed angular motor speed in rad/s
* @returns motor output torque in Nm
* @throws Error if omega_motor is zero
*/
export const calc_t_motorout = ({
p_motor_out,
omega_motor_speed,
}): {
p_motor_out: number;
omega_motor_speed: number;
}) => {
if (omega_motor_speed === 0) {
    throw new Error('omega_motor_speed is zero');
}
return p_motor_out / omega_motor_speed;
};
// Functions related to efficiency at the motor

/**
 * load efficiency approximation
 * @param p_motor_out current mechanical power of the motor
 * @param p_motor_rated rated power of the motor
 * @param motor_type the type of motor, either 'induction_motor' or 'permanent_magnet_motor'
 * @returns the load efficiency of the motor
 */
export const calc_efficiency = ({
  p_motor_out,
  p_motor_rated,
  motor_type,
}: {
  p_motor_out: number;
  p_motor_rated: number;
  motor_type: MotorType;
}): number => {
  const x = (0.001 * Math.abs(p_motor_out)) / p_motor_rated;
  if (x < 0) {
    throw new Error('x is negative');
  }
  let cout1: number;
  let cout2: number;
  let cout3: number;
  let dout1: number;
  let dout2: number;
  let eout1: number;
  let eout2: number;
  if (p_motor_out > 0) {
    if (motor_type === 'induction_motor') {
      cout1 = 0.9243;
      cout2 = 0.000127;
      cout3 = 0.01273;
      dout1 = 0.08;
      dout2 = 0.86;
      eout1 = -0.0736;
      eout2 = 0.9752;
    } else if (motor_type === 'permanent_magnet_motor') {
      cout1 = 0.942269;
      cout2 = 0.000061;
      cout3 = 0.006118;
      dout1 = 0.06;
      dout2 = 0.905;
      eout1 = -0.076;
      eout2 = 1.007;
    } else {
      throw new Error('unknown motor type');
    }
  } else {
    if (motor_type === 'induction_motor') {
      cout1 = 0.925473;
      cout2 = 0.000148;
    } else {
      throw new Error('unknown motor type');
    }
  }
}
cout3 = 0.014849;
dout1 = 0.075312;
dout2 = 0.858605;
eout1 = -0.062602;
eout2 = 0.971034;
}
else if (motor_type === 'permanent_magnet_motor') {
  cout1 = 0.942545;
cout2 = 0.000067;
cout3 = 0.006732;
dout1 = 0.057945;
dout2 = 0.904254;
eout1 = -0.066751;
eout2 = 1.002698;
} else {
  throw new Error('unknown motor type');
}
}
if (x < 0.25) {
  return (cout1 * x + cout2) / (x + cout3);
} else if (x < 0.75) {
  return dout1 * x + dout2;
} else {
  return eout1 * x + eout2;
};

/**
 * calculates the efficiency normalization factor based
 * on rated output power and efficiency requirements
 * @param p_motor_rated rated power of the motor in kW
 * @returns the efficiency normalization factor
 */
export const calc_norm_factor = (p_motor_rated: number): number => {
  if (p_motor_rated <= 0.75) return 0.817;
  if (p_motor_rated <= 1.1) return 0.839;
  if (p_motor_rated <= 1.5) return 0.855;
  if (p_motor_rated <= 2.2) return 0.874;
  if (p_motor_rated <= 3) return 0.889;
  if (p_motor_rated <= 4) return 0.901;
  if (p_motor_rated <= 5.5) return 0.914;
  if (p_motor_rated <= 7.5) return 0.926;
  if (p_motor_rated <= 11) return 0.94;
  if (p_motor_rated <= 15) return 0.949;
  if (p_motor_rated <= 18.5) return 0.956;
  if (p_motor_rated <= 22) return 0.96;
  if (p_motor_rated <= 30) return 0.968;
  if (p_motor_rated <= 37) return 0.973;
  if (p_motor_rated <= 45) return 0.978;
  if (p_motor_rated <= 55) return 0.981;
  if (p_motor_rated <= 75) return 0.987;
  if (p_motor_rated <= 90) return 0.99;
  if (p_motor_rated <= 110) return 0.993;
  if (p_motor_rated <= 132) return 0.996;
  if (p_motor_rated <= 160) return 0.998;
  return 1.0;
```javascript
/**
 * calculates the input power of the motor in W
 * in other words, how much power is drawn from the battery
 * to reach this level of motor output power when in motor mode, or
 * how much power is delivered to the battery when in generator mode
 * @param p_motor_out current mechanical power of the motor in W
 * @param regen_factor the speed-dependent regeneration factor
 * @param efficiency the load efficiency of the motor
 * @param norm_factor the efficiency normalization factor
 * @param p_te traction power in W
 * @returns the input power of the motor in W
 */
export const calc_p_motor_in = ({
  p_motor_out,
  regen_factor,
  efficiency,
  norm_factor,
  p_te,
}) => {
  if (regen_factor < 0 || regen_factor > 1)
    throw new Error('regen_factor must be between 0 and 1');
  if (efficiency < 0 || efficiency > 1)
    throw new Error('efficiency must be between 0 and 1');

  if (p_te <= 0)
    return p_motor_out * regen_factor * efficiency * norm_factor;
  else
    return p_motor_out / (efficiency * norm_factor);
};

/**
 * calculates the regeneration factor based on the current speed,
 * minimum speed for regeneration, and
 * the speed for maximum regeneration (above which additional energy is heat waste)
 * @param u the current speed of the vehicle in m/s
 * @param u1 the minimum speed for regeneration in m/s
 * @param u2 the speed for maximum regeneration in m/s
 * @returns the regeneration factor, between 0 and 1
 */
export const calc_regen_factor = ({
  u,
  u1 = 1.39,
  u2 = 4.72,
}) => {
  u: number;
  u1?: number;
  u2?: number;
  c?: number;
};
```
if (u1 < 0) throw new Error('u1 must be greater than 0');
if (u2 < 0) throw new Error('u2 must be greater than 0');
if (u2 < u1) throw new Error('u2 must be greater than u1');

const c = 1 / (u2 - u1);

if (u <= u1) {
    return 0;
} else if (u <= u2) {
    return c * (u - u1);
} else {
    return 1;
}

/**
 * calculates the total battery change in power in W
 * @param p_motor_in the input power of the motor in W
 * @param p_ac power draw by the accessories in W
 * @returns the total battery change in power in W
 */
export const calc_p_battery_out = ({
    p_motor_in, 
    p_ac,
}: { 
    p_motor_in: number; 
    p_ac: number; 
}): number => {
    return p_motor_in + p_ac;
};

/**
 * calculates the total power flow for the battery in W
 * @param p_battery_out current power of the battery in W
 * @param rte the round trip efficiency factor for the battery
 * @returns the total power of the motor in W
 */
export const p_total = ({
    p_battery_out, 
    rte = 0.95,
}: { 
    p_battery_out: number; 
    rte: number; 
}): number => {
    if (rte <= 0 || rte > 1) throw new Error('rte must be between 0 and 1');

    if (p_battery_out <= 0) {
        return p_battery_out * Math.sqrt(rte);
    } else {
        return p_battery_out / Math.sqrt(rte);
    }
};

/**
 * calculate the energy consumption in Wh in one second
 * @param p_te the current traction power in W
@param n_gear gear efficiency
@param efficiency motor efficiency
@param p_motor_rated rated power of the motor in kW
@param regen_factor the speed-dependent regeneration factor
@param norm_factor the efficiency normalization factor
@param p_ac power draw by the accessories in W
@returns rte the round trip efficiency factor for the battery

export const calc_energy_consumption = ({
  p_te,
  n_gear,
  efficiency,
  p_motor_rated,
  regen_factor,
  norm_factor,
  p_ac,
  rte,
}): {
  p_te: number;
  n_gear: number;
  efficiency: number;
  p_motor_rated: number;
  regen_factor: number;
  norm_factor: number;
  p_ac: number;
  rte: number;
}: number => {
  const motor_out = calc_p_motor_out({
    traction_power: p_te,
    n_gear,
  });

  const battery_out = calc_p_battery_out({
    p_motor_in: calc_p_motor_in({
      p_motor_out: motor_out,
      regen_factor,
      efficiency,
      norm_factor,
      p_te,
    }),
    p_ac,
  });

  if (p_te <= 0) {
    // regeneration from the wheels
    if (battery_out <= 0) {
      // battery is charging
      return {
        p_te *=
        n_gear *
        efficiency *
        ((0.001 * Math.abs(motor_out)) / p_motor_rated) *
        regen_factor *
        norm_factor +
        p_ac) *
        (1 / 3600) *
// Function for total power flow for a route segment

/**
 * calculate the energy consumption for a route segment
 * @param distance the distance of the segment in meters
 * @param duration the duration of the segment in seconds
 * @param elevation_start the elevation at the start of the segment in meters
 * @param elevation_end the elevation at the end of the segment in meters
 * @param vehicle the vehicle to calculate the energy consumption for
 * @param density_of_air the density of air in kg/m^3
 * @returns the energy consumption for the segment in Wh
 */
export const calc_route_segment_battery_power_flow = ({
    distance,
    duration,
    elevation_start,
    elevation_end,
    vehicle,
    density_of_air,
  }):
  number => {
    if (duration === 0) return 0;

    Math.sqrt(rte)
  };
} else {
  // battery is discharging because accessories draw exceeds regen
  return {
    ((p_te *
        n_gear *
        efficiency *
        ((0.001 * Math.abs(motor_out)) / p_motor_rated) *
        regen_factor *
        norm_factor +
        p_ac) *
        (1 / 3600)) /
    Math.sqrt(rte)
  };
} else {
  return {
    (p_te /
    (n_gear * efficiency * ((0.001 * Math.abs(motor_out)) / p_motor_rated) *
    norm_factor) +
    p_ac) *
    (1 / 3600)) /
    Math.sqrt(rte)
  };
}
// calculate the average speed for the segment
const v = distance / duration;

// calculate the average slope angle for the segment in radians
const slope_angle = Math.atan((elevation_end - elevation_start) / distance);

// calculate the total tractive effort at the wheels in Newtons
const aerodynamic_drag_force = calc_f_ad({
    rho: density_of_air,
    c_d: vehicle.drag_coefficient,
    area: vehicle.frontal_area,
    v,
});

const rolling_resistance_force = calc_f_rr({
    mu_rr: vehicle.rolling_resistance_coefficient,
    m: vehicle.mass,
    theta: slope_angle,
});

const hill_climbing_force = calc_f_hc({
    m: vehicle.mass,
    theta: slope_angle,
});

const linear_acceleration_force = calc_f_la({
    m: vehicle.mass,
    a: 0,
});

const inertial_force = calc_f_omega_a({
    c_i: vehicle.mass_correction_factor,
    m: vehicle.mass,
    a: 0,
});

const traction_force = calc_f_te({
    f_ad: aerodynamic_drag_force,
    f_rr: rolling_resistance_force,
    f_hc: hill_climbing_force,
    f_la: linear_acceleration_force,
    f_omega_a: inertial_force,
});

const traction_power = calc_p_te({
    f_te: traction_force,
    u: v,
});

// calculate the motor output power in Watts
const motor_out_power = calc_p_motor_out({
    traction_power,
    n_gear: vehicle.gear_efficiency,
});

const regen_factor = calc_regen_factor({
    u: v,
});
const efficiency = calc_efficiency({
  p_motor_out: motor_out_power,
  p_motor_rated: vehicle.p_motor_rated,
  motor_type: vehicle.motor_type,
});

const norm_factor = calc_norm_factor(vehicle.p_motor_rated);

const motor_in_power = calc_p_motor_in({
  p_motor_out: motor_out_power,
  regen_factor,
  efficiency,
  norm_factor,
  p_te: traction_power,
});

const p_battery_out = calc_p_battery_out({
  p_motor_in: motor_in_power,
  p_ac: vehicle.accessory_power_draw,
});

const battery_power_flow_in_wH = p_total({
  p_battery_out,
  rte: vehicle.rte,
});

return (duration * battery_power_flow_in_wH) / 3600;

Appendix 5: Experiment

let db = new Database('results.db', {
  verbose: console.log
});

dbpragma('journal_mode = WAL');
db.exec('CREATE TABLE IF NOT EXISTS routes {
  id TEXT PRIMARY KEY,
  startTime DATETIME,
  origin BLOB,
  destination BLOB,
  route BLOB,
  totalPower FLOAT,
  chargingStations BLOB,
  graph BLOB,
  minimumCapacity INTEGER,
  chargeLevelInterval INTEGER,
  financialCostPath BLOB,
  optimizedCost FLOAT,
  optimizedCostDuration FLOAT,
  durationPath BLOB,
  optimizedDuration FLOAT,
  optimizedDurationFinancialCost FLOAT,
  endTime DATETIME,
  error BLOB
}'});
export const experimentForRoute = async(data: {
  origin: {
    latitude: number;
    longitude: number
  },
  destination: {
    latitude: number;
    longitude: number
  }
}) => {
  if (!db) throw new Error('db not defined');

  const init = db.prepare(`
    INSERT INTO routes (id, startTime, origin, destination, route, totalPower)
    VALUES (?, ?, ?, ?, ?, ?)
  `);

  const addStations = db.prepare(`
    UPDATE routes set chargingStations = ? WHERE id = ?`
  );
  const addGraphSettings = db.prepare(`
    UPDATE routes set minimumCapacity = ?, chargeLevelInterval = ? WHERE id = ?`
  );
  const addFinancialCostData = db.prepare(`
    UPDATE routes set financialCostPath = ?, optimizedCost = ?,
    optimizedCostDuration = ? WHERE id = ?`
  );
  const addDurationData = db.prepare(`
    UPDATE routes set durationPath = ?, optimizedDuration = ?,
    optimizedDurationFinancialCost = ? WHERE id = ?`
  );
  const addEndTime = db.prepare(`
    UPDATE routes set endTime = ? WHERE id = ?`
  );
  const addError = db.prepare(`
    UPDATE routes set error = ? WHERE id = ?`
  );

  const jobId = uid();
  try {
    const startTime = Date.now();
    const origin,
    destination
  } = data;

  const route = await getRoute({
    origin: [origin.latitude, origin.longitude],
    destination: [destination.latitude, destination.longitude],
  });
  const {
    route: routeWithPower,
    totalPower
  } = calcPowerForRouteWithVehicle(route);

  let info = init.run(
    jobId,
    startTime,
    JSON.stringify(origin),
    JSON.stringify(destination),
    JSON.stringify(route),
    totalPower,
  );

  const chargingStations = await getChargingStationsAlongRoute({
    origin: [origin.latitude, origin.longitude],
  });
destination: [destination.latitude, destination.longitude],
getPricing: true,
);
const stations: ChargingStationAPIStation[] =
    chargingStations.stations as unknown as ChargingStationAPIStation[];

info = addStations.run(JSON.stringify(chargingStations), jobId);

const minimumCapacity = 43;
const chargeLevelInterval = 50;
const g = await createGraphFromRouteAndChargingStations({
    intersections: convertRouteFromStepsToIntersections(routeWithPower),
    stations: chargingStations.stations,
    minimumCapacity,
    chargeLevelInterval,
});

info = addGraphSettings.run(minimumCapacity, chargeLevelInterval, jobId);

const nodeCount = g.getNodesCount();
const edgeCount = g.getLinksCount();
const originalOutletCount = stations.reduce((acc, s) => acc + s.outletList.reduce((l) => l + 1, 0),
0,
);
const outletCount = stations.reduce((acc, s) => acc + s.outletList.reduce((l, o) => l + (o.capacity >= minimumCapacity && o.costKwh || o.costMin) ?
1 : 0),
0,
);

let type = 'cumulativeFinancialCost'
as 'cumulativeDuration' | 'cumulativeFinancialCost';
const financialCostId = uid();
const financialCostPath = findPathInGraphWithCostFunction({
    g,
    type,
    initialSoC: TestVehicle.battery_capacity * 0.95,
});
info = addFinancialCostData.run{
    JSON.stringify(financialCostPath),
    financialCostPath ?
    JSON.stringify(financialCostPath[financialCostPath.length - 1].cumulativeFinancialCost) :
    null,
    financialCostPath ?
    JSON.stringify(financialCostPath[financialCostPath.length - 1].cumulativeDuration) :
    null,
    jobId,
};

type = 'cumulativeDuration';
const durationId = uid();
const durationPath = findPathInGraphWithCostFunction({
    g,
    type,
    initialSoC: TestVehicle.battery_capacity * 0.95,
});
info = addDurationData.run(
    JSON.stringify(durationPath),
    durationPath ?
        JSON.stringify(durationPath[durationPath.length - 1].cumulativeDuration) :
        null,
    durationPath ?
        JSON.stringify(durationPath[durationPath.length - 1].cumulativeFinancialCost) :
        null,
    jobId,
);
const endTime = Date.now();
info = addEndTime.run(endTime, jobId);

const results = {
    jobId,
    startTime,
    origin,
    destination,
    route,
    totalPower,
    chargingStations,
    graph: g,
    financialCostPath,
    optimizedCost: financialCostPath ?
        financialCostPath[financialCostPath.length - 1].cumulativeFinancialCost :
        null,
    optimizedCostDuration: financialCostPath ?
        financialCostPath[financialCostPath.length - 1].cumulativeDuration : null,
    durationPath,
    optimizedDuration: durationPath ?
        durationPath[durationPath.length - 1].cumulativeDuration : null,
    optimizedDurationFinancialCost: durationPath ?
        durationPath[durationPath.length - 1].cumulativeFinancialCost : null,
    endTime,
};
return results;
} catch (error: unknown) {
    if (error instanceof Error) {
        addError.run(JSON.stringify(error), jobId);
    }
    const endTime = Date.now();
    const info = addEndTime.run(endTime, jobId);
}

export const runExperiment = async (routes: number) => {
    for (let i = 0; i < routes; i++) {
        const origin = getLatLonInSweden();
        const destination = getLatLonInSweden();
        await experimentForRoute({origin, destination});
    }
}