Battery Modelling

An MMSC Case Study on Mathematical Modelling

Candidate Number: 1072462

Abstract

This work will attempt to

Contents

1 Introduction

Clearly, electric batteries are largely important for various industries today and demand for them is ever-growing. This includes, especially, the renewable energy sector due to the unpredictability of energy supplies such as wind and solar power where short-term storage is a necessary evil. Similar relevance may be found in the car industry where one aims for highly (space-)efficient mobile storage of energy. In many countries and/or regions, Electric Vehicles (EVs) still lack a well-enlarged network of charging stations, for various reasons including incompatibilties between charging station suppliers. Using methods introduced in the present report, we aim to optimise the customer experience of an electric vehicle owner through a routing application that takes electric battery peculiarities and battery life-time into account.

In this report, we will consider how to model a Lithium-Ion battery, more specifically the Panasonic 18650PF for which there is some characteristic data available in the public domain (Kollmeyer 2018). To do this, we will consider an Equivalent Circuit Model (ECM), namely the Thevenin ECM, which may be found in Figure 2. The key idea here is to model the voltage output $V(t)$ given a current profile $I(t)$. This model not only depends on current and time, but also keeps track of internal quantities like the state of charge (SOC, $s \in [0,1]$) and state of health (SOH, $h \in [0,1]$). It can be reduced to a system of ordinary differential equations, for which, using given profiles, we find parameters R_0, R_1, C_1 and more by a fitting procedure. Later on, we also considered aging effects of the battery through usage over time, which resulted in an additional differential equation in the ODE system.

Based on the model we obtained, verified and validated on new pulse test data, the next step was to apply the model to a real-world application, namely that of electric vehicle routing. For this purpose, we obtain graph data of the road network of choice (users may enter new localitys as a text input) from OpenStreetMap (OpenStreetMap contributors 2023). In order to find the best route from A to B, the algorithm then performs a classical routing algorithm called *A-Star* to find the (geometrically) shortest path between A and B, which may not yet be a feasible or optimal path to the destination but a good start. A good initial condition is all that the remaining algorithm requires, which is a Monte-Carlo iterative method that repeatedly applies small, well-defined, perturbations to the graph route and then obtains a time or cost estimate of the modified trip using the battery (and car) model we introduced. If the

modified route yields an improvement in the metric of choice, it is accepted as the new state and the process repeats until a satisfactory route was found.

This report will focus on explicit formulation of the problem and model, numerical simulation of an electric car on a given real-world route and path optimisation through the Metropolis-Hastings method.

2 Problem and Model Formulation

2.1 The Isolated Battery

In order to model a Lithium-Ion battery in and of itself, we consider the following (physical) quantities: Let $s \in [0,1]$ denote the *state of charge* (SOC) of the battery, $h \in [0,1]$ the *state of health* (SOH), $Q \in \mathbb{R}^+$ the charge, $Q_{00} \in \mathbb{R}^+$ the maximum possible charge at the time of production (in Coulombs), $V \in \mathbb{R}$ the voltage across the battery (in Volts) with $I \in \mathbb{R}$ the corresponding current (in Amperes) where $I > 0$ corresponds to discharging the battery. Then, per common definition, $s := \frac{Q}{Q_0}$ is the amount of charge currently present in the battery as compared to $Q_0 \in \mathbb{R}^+$ the current maximum capacity, which itself is dependent on the state of health, as given by $Q_0 := hQ_{00}$. Further let $T \in [-273.15, \infty)$ denote the temperature of the battery (in degrees Celsius) and let $t \in \mathbb{R}$ represent time (in seconds).

From the definition of current $I := \frac{\mathrm{d}Q}{\mathrm{d}t}$, we further have that for a single cycle,

$$s = 1 - \frac{1}{Q_0} \int_0^t I(\tau) \mathrm{d}\tau \,,$$

under the assumption that Q_0, and therefore h, stays constant during that cycle.

2.2 The Equivalent Circuit Model

As mentioned earlier, we want to model the battery as an electrical component which exerts specific behaviour in an electrical circuit. In electrical engineering, such models of more complicated components are frequently represented by equivalent circuits which only consist of basic components, mostly resistors, diodes, transistors, capacitors and inductors.

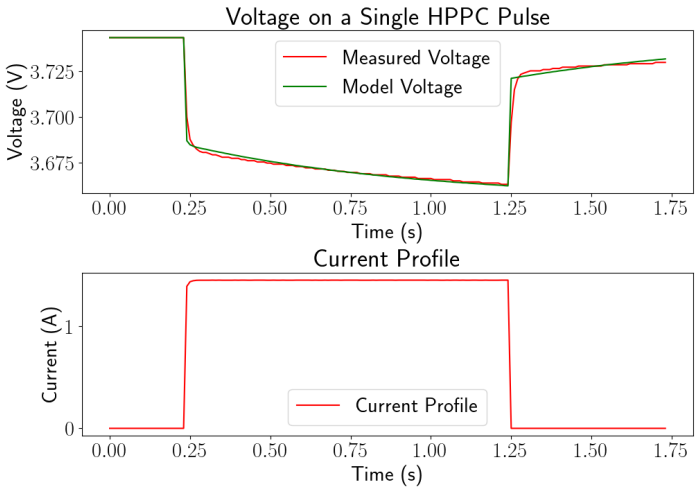


Figure 1: One of the sample HPPC (Hybrid Pulse Power Characterization) pulses we used for parameter finding. We want to model the output voltage $V(t)$ given a current input profile $I(t)$. Each sample is characterised at a specific state of charge s, state of health h and temperature T, this one was taken at $s \approx 1$, $h \approx 1$ and $T = 10$ °C.

Empirically, we know that a battery's voltage output looks roughly as given in Figure 1, which is part of a standardised cycle characterisation procedure taking the battery from "fully charged" ($s = 1$) to "completely empty" ($s = 0$). Data is due to Kollmeyer 2018. As one can see from the voltage curve, the battery exerts a "time-relaxation" behaviour on high-frequency changes in the current. One way of modelling this is through an RC circuit, confer Figure 2: the parallel circuit of R_1 and C_1 is responsible for an exponential-looking behaviour in the voltage curve given (near-)jumps in the current. Normally, RC circuits also Additional ohmic impedance is modelled through R_0. The "heart" of the equivalent circuit model is the open-circuit voltage V_{OC} which behaves like a voltage source but strongly depends on parameters s, h and T. We modelled it accordingly, using a polynomial ansatz including cross-terms

$$V_{OC}(s,h,T) = c_1 s + c_2 h + c_3 T + c_4 sh + c_5 sT + c_6 hT + c_7 shT + \mathcal{O}(s^2, h^2, T^2, \ldots)\,,$$

and similar polynomials are chosen for R_0, R_1 and C_1, completing the model.

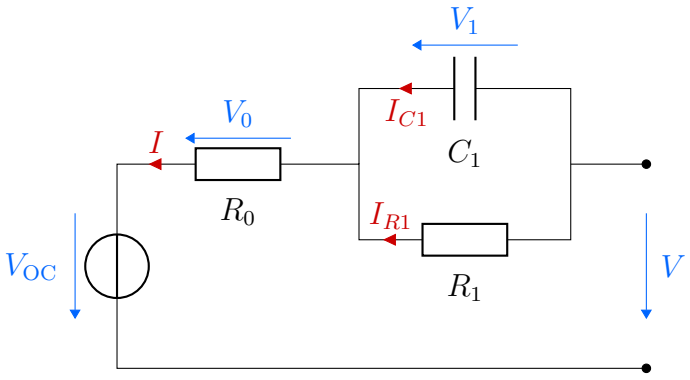


Figure 2: The Thevenin equivalent circuit model (ECM) with parameters $R_0 \in \mathbb{R}^+$, $R_1 \in \mathbb{R}^+$ and $C_1 \in \mathbb{R}^+$ and $V_{OC} \in \mathbb{R}^+$ the *open circuit voltage* which behaves according to a function $V_{OC}(s, h, T)$ dependent on s, h and T.

Kirchhoff's law then tells us that the currents $I_{R1} \in \mathbb{R}$ and $I_{C1} \in \mathbb{R}$ add up to the total current $I = I_{R1} + I_{C1}$, and that the voltages $V_0 \in \mathbb{R}$, $V_1 \in \mathbb{R}$ and V_{OC} sum up to $V = V_0 + V_1 + V_{OC}$. The capacitor behaves according to $I_{C1} = C_1 \frac{dV_1}{dt}$, while the resistors follow Ohm's law $V_0 = R_0 I$ and $V_1 = R_1 I_{R1}$. The resulting circuit then behaves approximately as a real Lithium-Ion battery would in many relevant situations. Let us consider an application of this model in an electric vehicle next.

2.3 Battery in an Electric Vehicle (EV)

In order to model a road network, let us first consider the definition of an undirected graph $G = (V_G, E)$. We take an undirected graph for simplicity, assuming that cars may go in arbitrary direction along each edge.

2.1 Definition: Undirected Graph

A graph $G = (V_G, E)$ with vertices V_G and edges $E \subseteq V_G \times V_G$ is undirected if and only if $(v_i, v_j) \in E \Rightarrow (v_j, v_i) \in E \quad \forall\, v_i, v_j \in V_G$.

Modelling a vehicle on a road network requires a few more definitions. On a graph (V_G, E) with edges $E = \{AB, AC, ...\} \subseteq V_G \times V_G$ and vertices $V_G = \{A, B, ...\}$, let $d_{AB} \in \mathbb{R}^+$ denote the distance between two vertices $A \in V_G$ and $B \in V_G$ (in meters), $x = x_{AB} \in [0, d_{AB}]$ the progress (current location) on the route from vertex A to B (in meters), $v := \frac{dx}{dt}$ denote the current velocity with $v_{\max,AB} \in \mathbb{R}^+$ the maximum allowed velocity on AB (in meters per second). Then let $T_{env}(x) \in [-273.15, \infty)$ denote the temperature of the environment (in degrees Celsius) at location x. Note that this

graph-based approach to the problem does not require any more geometric information about the network than edge lengths $\{d_{ij}\}_{i,j}$, while it may of course be motivated by spherical coordinates on the earth. Consider for example Figure 3.

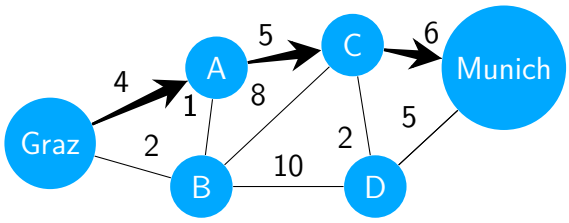


Figure 3: Exemplary route from Graz to Munich on a road network (V_G, E), where edge weights correspond to distances in between the physical places represented by nodes.

The model of our car still lacks a component realising electrical into mechanical power (a motor), which we model very simplistically. Let $P \in \mathbb{R}$, $P := I \cdot V$ denote the (electrical) power the car draws from the battery (in Watts) so $P > 0$ corresponds to discharging the battery. This power is to be realised into a mechanical component $P_{\text{motor}} \in \mathbb{R}^+$ driving the car forwards, heating for the battery $P_{\text{heat}} \in \mathbb{R}^+$, $P_{\text{heat}} = c(T - T_{\text{env}})$, with $c \in \mathbb{R}^+$ the heat conduction constant describing the relation between the heater and battery, and power dissipation $P_{\text{diss}} \in \mathbb{R}^+$. While driving, $P = P_{\text{motor}} + P_{\text{heat}} + P_{\text{diss}}$. The acceleration of the car $a \in \mathbb{R}$ (in meters per second squared), where $a := \frac{\mathrm{d}v}{\mathrm{d}t} = \frac{\mathrm{d}^2x}{\mathrm{d}t}$ is decomposed into $a_m \in \mathbb{R}$, which directly impacts $P_{\text{motor}}(a_m)$, and the deceleration due to friction (air, etc.) $a_f(v) \in \mathbb{R}^-$, so that in total $a = a_m + a_f$.

On the graph (V_G, E) there exists a set of EV charging stations $V_{\text{charge}} \subseteq V_G$ where $P_{C,\text{charge}}$ denotes the possible charging power (in Watts) at the charging station vertex $C \in V_{\text{charge}}$ with $K_C \in \mathbb{R}^+$ the occuring costs per energy unit (in Euros per Watt second) and $t_C \in \mathbb{R}^+$ the charging time per charging station B (in seconds).

2.4 A Variational Optimisation Problem

Given source and destination vertices $A, Z \in V_G$ on the graph (V_G, E), which connected set of edges $E_R \subseteq E$ connecting A to Z, set of visited charging stations $V_C \subseteq V_{\text{charge}}$ and charging times $\{t_C\}_{C \in V_C}$ visited on the route E_R, and driving behaviour $a_m(x, v, t, s, h, T_{\text{env}}, ...)$, $a_m \in \mathcal{C}^1(\mathbb{P})$ with $\mathbb{P}$ the parameter space [1] minimises

[1] to be defined.. TODO

1. the total travel time $t_{\text{total}} := \int_{V_R} \frac{1}{v} \mathrm{d}x + \sum_{C \in V_C} t_C$,
2. the total cost of travel $K := \sum_{C \in V_C} P_{C,\text{charge}} t_C K_C$,
3. $-N$ where N is the highest possible number of repetitions (commutes from A to Z) with the same battery (requiring $h > 0$).

Formulated differently, we aim to minimise the functional $F \in \mathcal{C}(\mathbb{P})^*, F : \mathcal{C}(\mathbb{P}) \mapsto \mathbb{R}$ where either $F[\chi] = t_{\text{total}}$ or $F[\chi] = K$.

Charging station + street data could be obtained from OpenStreetMap by calling `osmfilter england-latest.o5m keep="amenity=charging_station"` [2].

2.5 Battery Aging

$$Q(t, c, s, I) = Q_0 - \frac{F_{acycle}(c)}{F_{current}(I)} - F_{cal}$$

By $F_{acycle}(c)$ we denote the *Current Agnostic Cycle Degradation Factor* models cycle aging at a single current. Also $F_{current}(I)$ represents the *Current Scaling Factor*, a value $0 < F_{current} \leq 1$ that incorporates behaviour where higher currents are worse for cycle aging. Finally, $F_{cal}(s, t)$, the *Calendar Degradation Factor*, ages the battery over time and accounts for suboptimal storage in terms of state of charge.

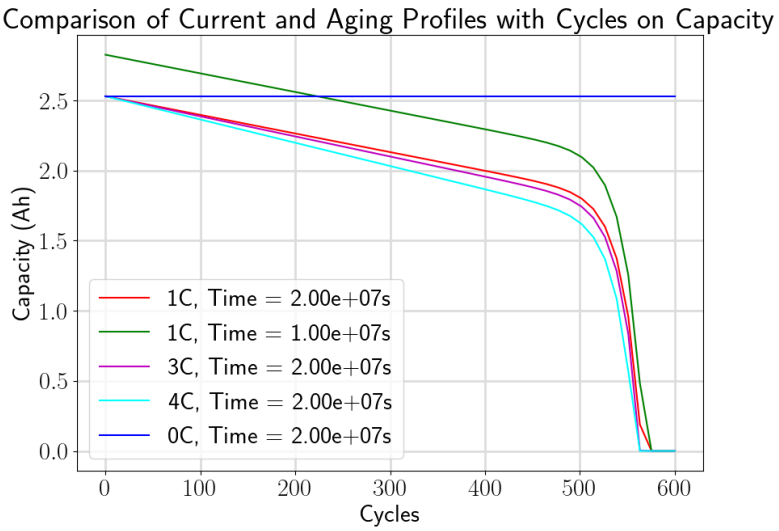


Figure 4: Aging

[2] OSM's public map data may be obtained from https://download.geofabrik.de/

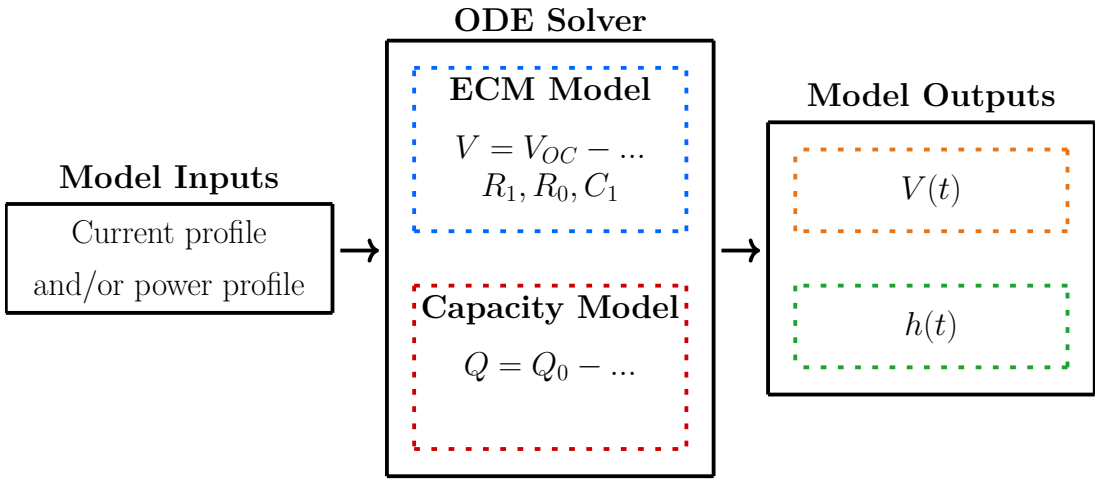


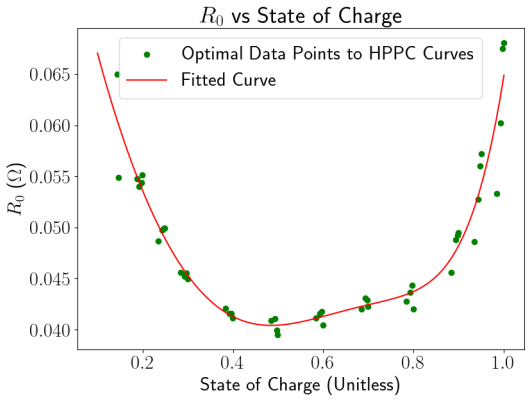
Figure 5: Overview

Perez et al. 2018.

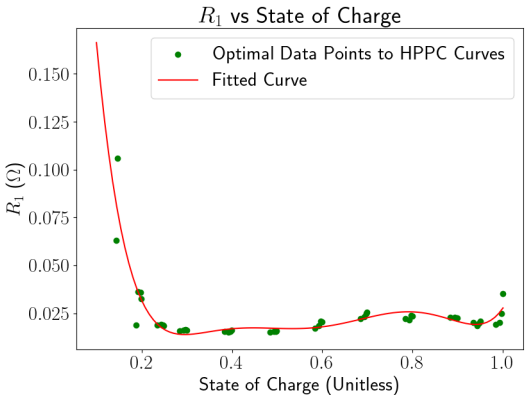
3 Numerical Simulation of a Car

For simplification, we neglected temperature changes in the environment and a respective battery heating strategy $P_{\text{heat}}(x, v, t, s, h, T_{\text{env}}, ...)$, $P_{\text{heat}} \in \mathcal{C}(\mathbb{P})$.

3.1 Finding ECM Parameters



(a) Fit of R_0 as a function of the SOC (s).



(b) Fit of R_1 as a function of the SOC (s).

3.2 Forward Euler Simulation

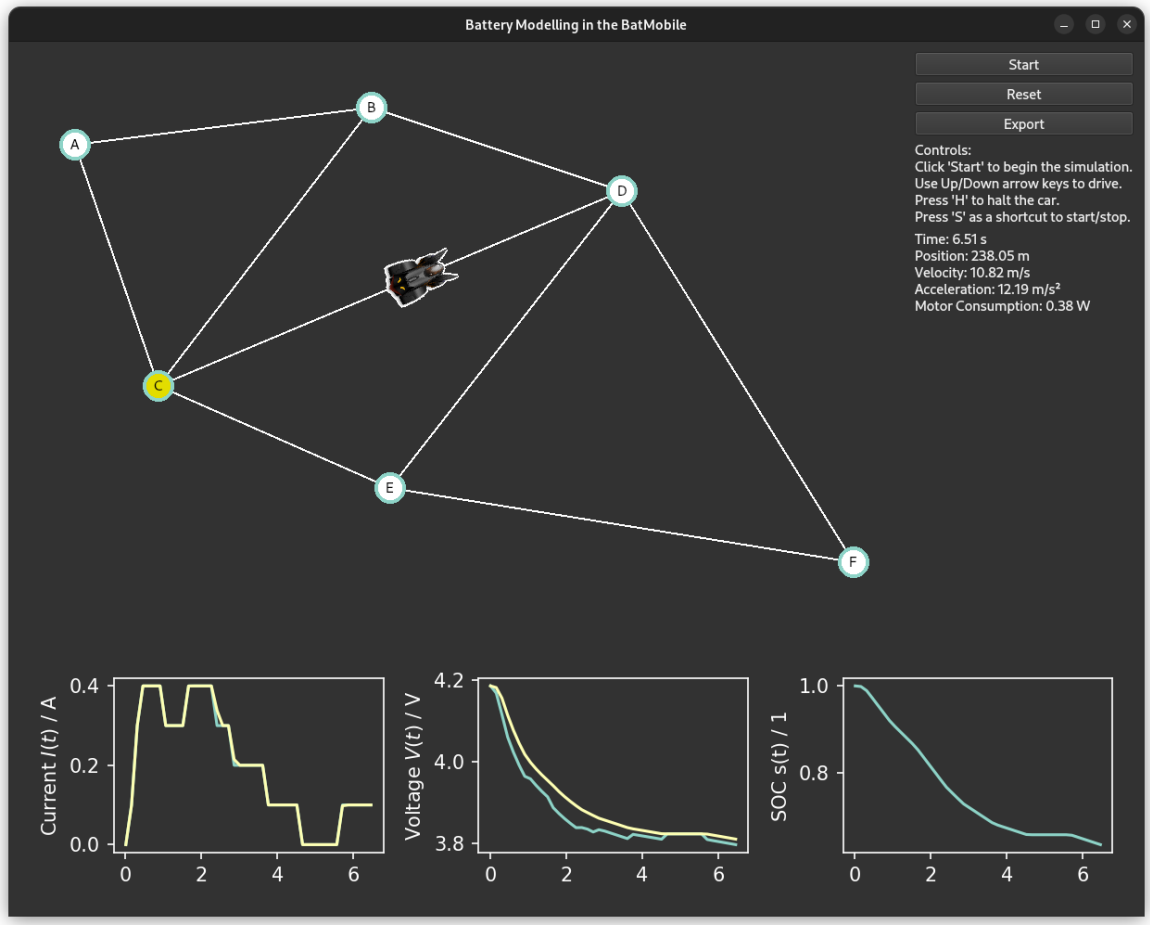


Figure 7: User Interface

4 Metropolis-Hastings and A-Star

4.1 Shortest Path Finding

The graph data was retrieved using OSMNX (Boeing 2017) which itself is built on NetworkX (Hagberg, Schult and Swart 2008).

4.2 Monte-Carlo Optimisation

Optimise a large problem (huge state-space). Metric: Time! Could be anything. ⇒ Use Monte-Carlo Markov Chain Methods! Slightly perturb the route using a specific

alteration technique. Metropolis-Hastings updates the state (route) based on

$$p_{\text{accept}} = \min\left(1, \mathrm{e}^{-\beta(T_{\text{next}} - T_{\text{current}})}\right), \qquad \text{with } \beta \in \mathbb{R}^{+} \text{ a transition factor}.$$

Does a full numerical simulation of the drive. Stop to charge? Explore the state-space to some extent, and return the best route! Larger scales / maps (e.g. England) are not a problem!

```
newRoute = self.simulator.batgraph.perturbRoute(self.route)
delta = self.measureRoute(newRoute) - self.testedRoutes[self.route]
acceptanceProbability = min(1, math.exp(-delta / self.temperature))
if random.random() < acceptanceProbability:
    self.route = newRoute
    print(f"Accepted new route {newRoute} with delta: {delta:.2f}. Total:
    ↪ {self.testedRoutes[newRoute]:.2f}.")
else:
    print(f"Rejected route {newRoute} with delta: {delta:.2f}.")
```

4.3 Special Case: Granny's House

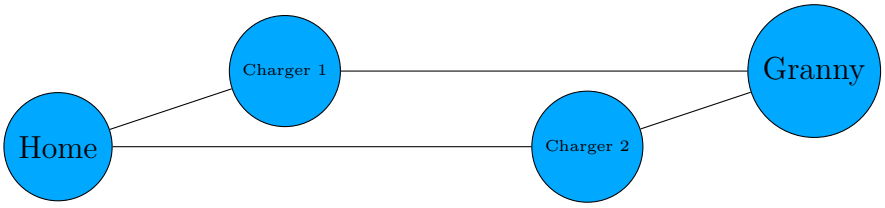


Figure 8: The exemplary problem "Granny's House", a special case of Problem (TODO).

4.4 Routing in Jericho

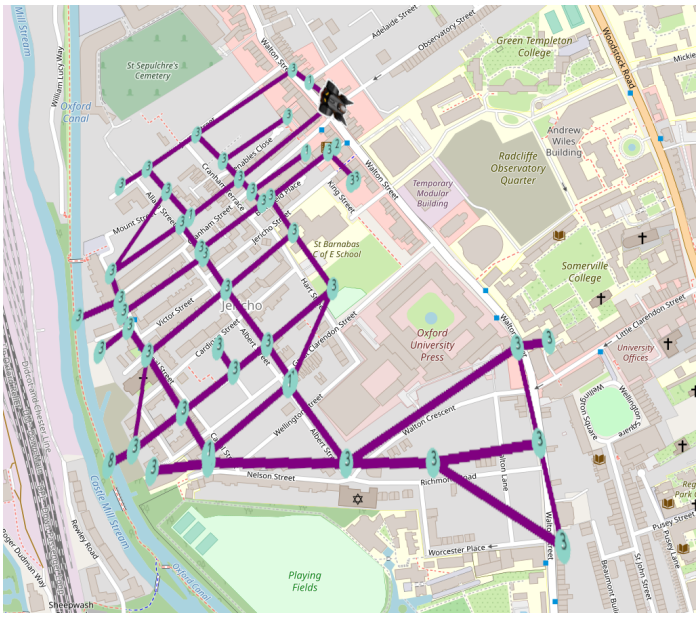


Figure 9: Overlay of the routing graph on a map of Jericho (OpenStreetMap contributors 2023), without adjusting for the Merkator projection, which leads to a slightly skewed appearance. The underlying data is exactly the same.

5 Conclusion

References

Boeing, Geoff (2017). 'OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks'. In: *Computers, Environment and Urban Systems* 65, pp. 126–139. ISSN: 0198-9715. DOI: 10.1016/j.compenvurbsys.2017.05.004.

Hagberg, Aric A., Daniel A. Schult and Pieter J. Swart (2008). 'Exploring Network Structure, Dynamics, and Function using NetworkX'. In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught and Jarrod Millman. Pasadena, CA USA, pp. 11–15.

Kollmeyer, Phillip (21st June 2018). 'Panasonic 18650PF Li-ion Battery Data'. In: 1. Publisher: Mendeley Data. DOI: 10.17632/wykht8y7tg.1. URL: https://data.mendeley.com/datasets/wykht8y7tg/1 (visited on 02/03/2023).

OpenStreetMap contributors (2023). *Planet dump retrieved from planet.osm.org*. URL: https://www.openstreetmap.org (visited on 08/03/2023).

Perez, Aramis, Vanessa Quintero, Francisco Jaramillo, Heraldo Rozas, Diego Jimenez, Marcos Orchard and Rodrigo Moreno (2018). 'Characterization of the degradation process of lithium-ion batteries when discharged at different current rates'. In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 232.8, pp. 1075–1089. DOI: 10.1177/0959651818774481.

A Simulation Code

```python
import math
import random

from simulator.simulation import Simulation


class Optimiser:
    def __init__(self, graph) -> None:
        self.simulator = Simulation(graph)
        self.testedRoutes = {}
        self.temperature = 1.0

    def initialise(self, source, destination):
        """Simulates (measures) the shortest possible route, ignoring charging stations, etc."""
        if not self.simulator.batgraph.has_node(source):
            source = int(source)
        if not self.simulator.batgraph.has_node(destination):
            destination = int(destination)
        self.route = self.simulator.batgraph.findShortestPath(source, destination)
        self.measureRoute(self.route)

    def metric(self):
        """Metric to minimise for."""
        return self.simulator.totalTimeElapsed

    def measureRoute(self, route):
        """Test out a given route in the simulator."""
        self.simulator.reset()
        self.simulator.runOnPath(route, lambda t, soc: 1)
        self.testedRoutes[route] = self.metric()
        return self.testedRoutes[route]

    def mcmcStep(self):
        """Perform a Monte-Carlo Markov-Chain, Metropolis-Hastings iteration step."""
        perturbationAttempt = 0
        while True:
            try:
                # print("Perturbing...")
                newRoute = self.simulator.batgraph.perturbRoute(self.route)
                if newRoute != self.route:
                    break
            except (KeyError, StopIteration):  # perturbRoute() was unsuccessful
                pass
            perturbationAttempt += 1
            if perturbationAttempt > 20:
                print("Giving up perturbation.")
                return
```

```
48
49          # a negative delta is good!!
50          newEnergy = self.measureRoute(newRoute) if newRoute not in
            ↪  self.testedRoutes else self.testedRoutes[newRoute]
51          delta = newEnergy - self.testedRoutes[self.route]
52          acceptanceProbability = min(1, math.exp(-delta / self.temperature))
53          if random.random() < acceptanceProbability:
54              self.route = newRoute
55              print(f"Accepted new route {newRoute} with delta: {delta:.2f}. Total:
                ↪ {self.testedRoutes[newRoute]:.2f}.")
56          else:
57              print(f"Rejected route {newRoute} with delta: {delta:.2f}.")
```