Fastest-Path Algorithm for Electrical Vehicles

A. B. Eriksen
Institute of Computer Science
Aalborg University
xxx@xxx.dk

M. A. Madsen
Institute of Computer Science
Aalborg University
xxx@xxx.dk

M. M. Andersen
Institute of Computer Science
Aalborg University
xxx@xxx.dk

S. B. Jensen
Institute of Computer Science
Aalborg University
xxx@xxx.dk

ABSTRACT

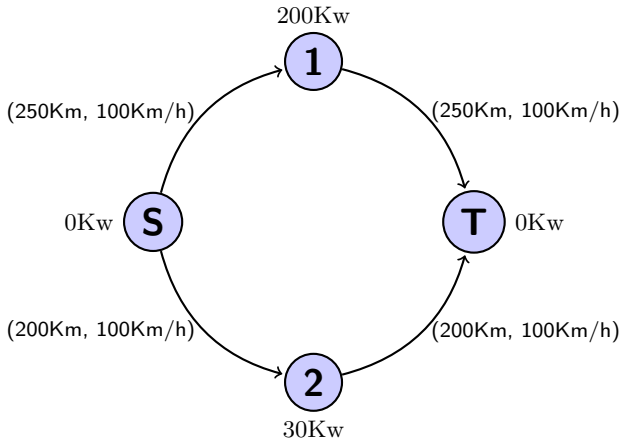
Keywords
Keyword1, Keyword2, Keyword3, Keyword4

1. INTRODUCTION

As the adoption rate of battery powered cars increases, the need for a better and more complex route planing solution appears. Battery powered vehicles generally have a shorter range as well as a significantly slower refuel time, compared to traditional gasoline cars. As the refuel time have become a significant factor in the combined travel time, the time spend refueling should be taken into account by a route plan, to accommodate for an otherwise large margin of error. Time spend recharging batteries is inherently variable, as the charge rate is significantly affected by both the charge stationâĂŹs charge rate, as well as the batteryâĂŹs current charge. When planing a route for a modern electric vehicle, it is therefore paramount to be able to incorporate charge stations, charge rates and battery charge into the plan. Another important aspect is resource consumption itself. With the large amount of time spend refueling, a high rate of energy consumption while driving fast, can in turn result in a significant time spend on refueling. It should be clear that no traditional shortest path algorithm is able to accommodate for the variables and relationships between travel speed, charge rate and route planing. In this article we present two different approaches to solving the problem of optimizing a route plan for electiv verhicles. One is a linear program..... The other is a more naive solution which uses heuritics..... Both approaches are analysed on accuracy and running time, based on a real world road network as input.

2. THE BODY

2.1 Shorter Not Equal to Faster



A shorter path does not necessarily mean a faster path for electrical vehicles. This is partly due to the fact that an electrical vehicles uses exponentially more energy as its speed increases but also due to the fact that charge times on charge stations varies. Chosing a longer path, where a faster charge station exist can turn out to be the ultimately fastest choice. This is illustrated in **fig. above**. In this example we want to go from vertice `S` to vertice `T`, assume our car has a battery capacity of 100kWh and an consumtion rate of 0,4kWh/km, at 100Km/h. Path 1 consist of two edges connecting three vertices. Edges represent the road segment, in this example path 1 has distance = 250km and speed limit = 100km/t. All road segments are connected by vertices, which can be intersections, end of a roads, charging station and so on. In figure **figure above** we also have a charge station on path 1 with a charge rate of 200Kwh/h. On path 2 we again have two edges with distance = 200 km and speed limit 100 km/h and a charge station with a charge speed of 30 kWh/h. The total time of each path:

Path 1: route time = ((250km + 250km) / 100km/h) + (100kWh / 200kWh/h) = 5,5h

Path 2: route time = ((200 km + 200 km) / 100 km/h) + (60 kWh / 30 kWh/h) = 6 h

We see that chosing path 1, with a powerful charge station, will save us half an hour, even thought the road segment alone is 100km longer than path 2.

2.2 The graph model

2.3 Optimization algorithm

The main goal of this paper is to design an algorithm to find the fastest path for an EV, considering both the time spend driving and the time spend charging. The hard thing about finding the fastest path is and discussed in section 2.1 there is no optimal substructure to this problem. The solution model we purpose consistes of continuous pruning and then solving the problem for the path with the lowest T_p, until we end up with just a single path which is the fastest path. An initial way to prune is to find a path P from s to t, and solve it optimally. Solving P optimally will return the time it will take to follow this path, we call this value T_p. We then observe that if another path P' driven with max speed, ignoring the energy constraint, will take time $T_{p'}$ in time. If $T_p \leq T_{p'}$ we can simply ignore the path P', since the actual time used to drive P' will be greater then or equal to $T_{p'}$. If we find a path which can be driven faster then T_p we update P and $T_{p'}$. When it comes to determining how a path is optimally solved will be discussed in section 2.4. Using pruning and a way to find out how a path is optimally solved, we can produce an algorithm with findes the fastest path as follows.

function FASTESTPATH(G, s, t)
  $p \leftarrow$ initialpath(G, s, t)
  $T_p \leftarrow$ solve(p)
  $paths \leftarrow$ getpaths(G, s, t, T_p)
  repeat
    $T_{p1} \leftarrow$ solve(paths[1])
    if $T_{p1} < T_p$ **then**
      $T_p \leftarrow T_{p1}$
      $p \leftarrow paths[1]$
    end if
    $paths.remove(paths[1])$
    for all $p \in paths$ **do**
      if $T_p \leq p.min$ **then**
        $paths.remove(p)$
      end if
    end for
  until $paths.size = 0$
  return (p, T_p)
end function

Where $initialpath(G, s, t)$ is the initial path we choose to solve e.g. shortest path. $solve(p)$ is a function which estimates the optimal solution to a path i.e. optimal charging times for each charging station and driving speed for each road segment in a path. $getpaths(G, s, t, T_p)$ is a function which finds all paths in G between s and t with a max travling time of t. Having the algorithm we now investigate how to estimate an optimal solution for a path. s

2.4 Optimizing with a linear program

To solve the route planning problem with a linear program solver, we are going to get approximate solutions. Linear program solvers work on linear equations, and we ultimately want to use non-linear expressions for the energy consumption as a function of speed, as well as charge rate as a function of battery charge. These functions will each be represented by by a set of linear equations. The precision of these peicewise linearized expressions of course depend on the amount of linear expressions used. In section (experiment), you can see the effect of this varying degree of approximation on a route plan.

minimizing the sum of time spend driving and the sum of time spend charging.

$$\text{minimize} \quad \sum_{i=1}^{n} \frac{Charge\ rate[i]}{Velocity[i]} + \sum_{i=1}^{n} Charge\ station[i] \tag{1}$$

subject to:

$$\forall_{i \in 1...n} \ : \ 0 \leq \sum_{j=1}^{m} Selected\ lines[i,j] \leq 1 \tag{2}$$

$$\forall_{i \in 1...n, j \in 1...m} \ : \ Selected\ lines[i,j] = 1 \tag{3}$$

$$\forall_{i \in 1...n, j \in 1...m} \ : \ Velocity[i,j] \leq Selected\ points[i,j] * Points1[i,j] \tag{4}$$

$$\forall_{i \in 1...n, j \in 1...m} \ : \ Velocity[i,j] \geq Selected\ points[i,j] * Points2[i,j] \tag{5}$$

$$\forall_{k \in 1...n} \ : \ 0 \leq \sum_{i=1}^{k} Charge\ rate[i] * Charge\ time - \sum_{i=1}^{k} Edge\ distance[i] (\sum_{j=1}^{m} LinesA[i,j] * Velocity[i,j] + \sum_{j=1}^{m} Selected\ edges[i,j] * LinesB[i,j]) \leq Battery\ capacity \tag{6}$$

Vis pseudokode for proceduren

2.5 Optimizing with heuristics

3. EXPERIMENTS

In this section we will describe the experiments being conducted.

3.1 Setup

To facilitate the experiments, we've had to find a real world road network graph dataset which has road distances, speed limits and charging stations of varying power output. Such dataset does not openly exist to our knowledge, instead we have used OpenStreetMaps, henceforth *OSM*. One can read more about it at `http://www.openstreetmap.org/about`. OSM has a concept of *Ways* and *Nodes*. Ways represent geographical planar objects, e.g. roads. A node is a geographical point (latitude,longitude). A way is then constituted of several nodes, and some tags which describe meta-information such as the name of the way, and what type of construct it is. As such, it follows that if way A and B intersect in node x, they will share the node x, known as an intersection in the road network.

These objects can easily be converted into a road network structure, by filtering ways of type *road* and use these ways as edges in the road network. To get speed limits we can derive general speed limits from the type of the roads, OSM carries such information as motorway,residential,tertiary for roads.