UNIVERSITY OF CALIFORNIA
RIVERSIDE

Electric Vehicle Route Planning in the Presence of Stochasticity

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Payas Rajan

March 2022

Dissertation Committee:

Dr. Chinya V. Ravishankar, Chairperson
Dr. Vassilis J. Tsotras
Dr. Ahmed Eldawy
Dr. Amr Magdy

The Dissertation of Payas Rajan is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

This work, like all other works, is a result of inputs from several people, and I'd like to thank everyone who helped me along the way. The first order of thanks, of course, goes to my advisor, Dr. Ravishankar. Thank you, for your patience, for being so generous with your time, and teaching me how to think clearly. Big thanks also to Daniel Delling at Apple, for giving me the opportunity to work with and learn from so many other people, Moritz, Michael, Dennis, Christian, Casey, Sofia. Chapter 3 was written during the internship, and provided a boost when it was most needed.

Thanks also to Megan Danielson and Mikel Maron at Mapbox for allowing us access to the dataset used in chapters 4 and 5, and to Guoyuan Wu and Matthew Barth for the dataset used in chapter 2. Finally, thanks to Abhishek, Ravdeep, Amrita, and my parents for being such amazing pillars of personal support.

To everyone I've ever learned from.

ABSTRACT OF THE DISSERTATION

Electric Vehicle Route Planning in the Presence of Stochasticity

by

Payas Rajan

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, March 2022
Dr. Chinya V. Ravishankar, Chairperson

Electric Vehicle (EV) route planning is an important but hard problem, since battery capacity is limited, charging times are long, and charging stations are sparsely distributed. It is nonetheless critical to improving the time and energy efficiency of future transportation systems, and to promoting EV adoption, by reducing driver range anxiety. EV routing is usually modeled as a Shortest Feasible Path (SFP) problem, which ensures a non-negative State of Charge along the route, taking both travel time and energy consumption to be deterministic and known in advance. In practice, however, travel time and energy consumption are both stochastic variables, which can be hard to estimate accurately.

This dissertation presents a set of techniques to advance our abilities to address such stochasticity. First, we show how to accurately predict energy consumption and travel times along routes using *phases*, a new structuring abstraction for vehicle speed profiles. Contrary to conventional wisdom, using phases outperforms even microscopic energy estimation models. We show that using phases to generate synthetic trips preserves the real-world variance in travel times and energy consumptions of real-world trips.

Next, we show how to efficiently encapsulate the tradeoffs between travel times and robustness of feasible routes against deviations in energy consumptions using the *Starting Charge Map* and *Buffer Map* constructs. Further, we generalize the SFP problem to permit stochastic travel times and energy consumptions using two different probabilistic definitions of route feasibility. These definitions allow drivers to maintain route feasibility either in expectation, or by setting explicit lower bounds on stranding probability.

We also study how to effectively apply well-known speedup techniques, such as Contraction and Edge Hierarchies, for route planning with stochastic edge weights. We show that the choice of weight representations has a significant impact on the routing query runtimes, and introduce the *tiering* technique, which significantly improves query times for three different stochastic routing objectives. We evaluate all presented methods on realistic routing instances.

Lastly, we generalize the problem of identifying dwell regions for trajectory sets to that of finding shared dwell regions, and present two novel approaches to the problem. We show that our solutions outperform the state-of-the-art by nearly a factor of three.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Route planning has been extensively studied over the last two decades, and several methods for fast routing on large road networks are now commonly used [13, 149]. However, Electric Vehicles (EVs) have limited battery capacities and much longer charging times, while the charging infrastructure remains sparse and fragmented among different providers. EV route planning, therefore, must consider energy consumption along routes in order to increase the effectiveness of transportation systems and to minimize the risk of the drivers getting stranded [133, 140].

Much prior work on EV routing attempts to find *feasible* routes, on which a non-negative State of Charge (SoC) can be maintained on the EV [18, 19]. However, they also assume that the energy consumptions and travel times of vehicles along the road network are deterministic values that are known exactly [18, 19, 155, 151, 154]. In practice, though, these parameters depend significantly on transient factors such as driver aggressiveness, weather, and the state of wear of the battery, and may only be estimated as random variables [53].

In this thesis, we provide a set of techniques that enable EV route planning to be effective in presence of such stochasticity. Our contributions span several subproblems, which we discuss in the remainder of this section.

1.1 Our contributions

Our first contribution is in the domain of EV modeling and energy estimation. Several kinds of energy consumption models exist in literature, and can be classified as microscopic, mesoscopic or macroscopic, depending on the temporal granularity at which they operate. In chapter 2, we present *phases*, a new structuring abstraction for EV speed profiles, which we use to derive an accurate and mesoscopic energy consumption model that is tunable with only one parameter. We evaluate the accuracy of our model using real-world data collected from 52 hours of trips on a Nissan Leaf 2013. Further, we also show that phases can be used to build a simple, yet effective, synthetic speed profile generation model that preserves the variance in energy consumption and travel times of real-world trips.

Our next two contributions are in the domain of EV route planning algorithms. EV routing is often modeled as the NP-hard Shortest Feasible Path (SFP) problem, which minimizes the total travel time while maintaining a non-negative SoC for the EV at all points along the route. SFPP admits an EXPTIME solution called the Charging Function Propagation (CFP) [18, 19]. However, using CFP for real-world routes presents two major problems: the SFP model presumes all energy consumption estimates are *perfect*, and the problem returns only the *shortest* feasible paths, which may not suffice for a wide variety of drivers considering the different levels of acceptable risk for different driving scenarios.

In chapter 3, we first introduce the *Starting Charge Map (SCM)* and *Buffer Map (BM)* constructs which encapsulate the trade off between *robustness* of feasible EV routes and travel times. Then, we present two generalizations of SFPP which allow us to compute SCM and BM exactly and efficiently.

Next, in chapter 4, we generalize the SFP problem to explicitly allow stochastic edge weights. However, doing so requires us to revisit the traditional definitions of *feasibility* of routes to accommodate probabilistic values of energy consumption and travel times. Therefore, we define $\mathbb{E}$- and p- feasibility, which allow the user to maintain a non-negative SoC along the route either in *expectation*, or provide a lower bound on the probability of not getting stranded along the route. We then generalize CFP to accept stochastic edge weights and return $\mathbb{E}$- or p- feasible routes. In order to speed up our queries, we extend the multicriteria Contraction Hierarchies to allow stochastic edge weights, and evaluate our methods on a real-world dataset containing speeds on road network edges in the Los Angeles area over a span of four and a half months.

In chapter 5, we study the performance of Contraction Hierarchies [64, 120] and Edge Hierarchies [73] when the edge weights are probability distributions, for three stochastic routing objectives: the *mean-risk routes* [115], *probabilistic budget routing* and *non-dominated routes* [120]. Similar to [84], we observe that most time for answering such queries is spent computing convolutions of edge weight distributions. Therefore, we introduce the *tiering* technique of using various edge weights representations at different tiers of shortcut hierarchies. We apply tiering to both Contraction and Edge Hierarchies, and show that it leads to a reduction in both preprocessing and query times.

Chapter 6 is a departure from the route planning methods discussed in the earlier chapters, and generalizes the dwell regions problem of [160] to the case of multiple moving objects. We define the *shared* dwell regions and N/S-shared dwell region queries, and present two novel methods to answer the queries by casting the problem as a Pointwise Dense Region query of [110]. Our experiments on real-world trajectory datasets show that the proposed methods outperform the baseline methods by up to a factor of 3.3x.

Chapter 2

The Phase Abstraction

2.1 Introduction

Many spatiotemporal applications need route planning. Traditional route planning methods find minimum-time routes, assuming motion at the maximum speed allowed by traffic or law [13, 149]. However, this assumption is invalid for Electric Vehicles (EVs), since EV battery capacity is limited, and energy losses from wind resistance grow quadratically with speed. Higher speeds shorten travel time, but may cause unsustainable battery drain. Therefore, EV route planning requires finding a suitable trade-off between travel time and energy consumed over the set of paths between a source and a destination vertex in the road network.

Given its importance to spatiotemporal applications, the database community has begun to work on energy estimation models, both for internal combustion vehicles [70, 71], as well as EVs [87, 94, 162]. Historically, such work has been studied by traffic engineers as energy estimation [54, 171, 165, 43, 42, 128], and speed profile estimation [117, 118].

Addressing these problems independently is not useful for EV route planning, where explicit time-energy tradeoffs must be made. Current methods differ widely in how they approach this tradeoff. Some use energy consumption as the sole route-selection criterion [23, 50, 139], while others attempt bi-criteria search on time and energy [20, 26, 67, 94].

2.1.1 EV-Specific Issues in Estimation Models

EV estimation models are complicated by EV-specific features, such as regenerative braking. EVs typically use regenerative braking at low decelerations and mechanical braking at high decelerations, say, when avoiding collisions. Decelerations may occur in three modes: regenerative braking, mechanical braking, or by stepping off the pedal (cruising). Energy recovery may be significant only in the first case. Besides, the deceleration threshold that triggers different braking modes can be context-dependent. Often, the EV may forego energy recovery when the battery status of charge is high [157]. It is not straightforward for an EV energy estimation model to determine whether or not a deceleration event recovers energy. Static models designed for internal combustion vehicles like [32] cannot handle such additional EV-specific complexities.

The effectiveness of any EV route planning method depends on the accuracy of energy and time estimates. All energy consumption models represent vehicle trips as sequences of smaller movement units. Thus, *microscopic* energy estimation models [54, 171, 165] work with instantaneous (or per-second) vehicle movement, while *mesoscopic* models [42, 36] use coarser units, such as transits over network edges. Understandably, models using different abstractions of movement differ significantly in accuracy and computational costs. Limited battery capacity and inadequate charging infrastructure also cause *range anxiety*

[33, 48, 60, 135], though current EV range may suffice for most trips [109]. Psychological studies suggest that people treat machines as if they were human [134], and that greater user trust in EVs may help counter range anxiety. Improved route planning and range estimation are useful in helping build such trust [80, 108].

2.1.2 Our Contributions

Our **first** contribution is the definition and development of the ***phase*** abstraction of EV movement, using which we derive a *tunable* mesoscopic EV energy consumption model (Section 2.3.2). Our estimation model requires only a single parameter, τ, which suffices to capture the complexity induced by regenerative braking.

Second, in Section 2.4, we develop a stochastic model to generate *realistic* trips along a path, given a set of past trips along the same path. Our model is able to generate a set of speed profiles which exhibit the real-world variance in energy consumption and travel times observed in past trips. In practice, one would not expect any two trips, even by the same vehicle, at same times of the day, and on the same route, to take the same time or consume the same amount energy. Model-generated speed profiles are currently used in traffic simulators like POLARIS [10] to perform regional energy use analysis [11], and it is essential that this natural variance in these parameters be preserved.

To generate realistic speed profiles, our approach first models the set of past trips as sequences of phases and learns relevant parameters using Markov chains and Kernel Density Estimation (KDE). Then, it generates speed profiles using a random walk on the Markov chain to model phase transitions, sampling the PDF functions obtained from KDE to derive the average acceleration and duration of each phase. Energy and time estimates

for route planning can then be obtained from the generated speed profiles by applying our phase-based EV energy consumption model.

We evaluate our energy and speed profile generation models using 52 hours of real-world data from a 2013 Nissan Leaf. Though current literature assumes that models at a lower temporal granularity yield better energy estimates [1, 158], our phase-based mesoscopic energy estimation model outperforms even analytic microscopic energy consumption models. Our speed profile model, tunable with τ, generates speed profiles up to 1250 seconds long that are a close match real-world data for distance and energy consumed over time.

This paper augments prior work by the authors [131] in three ways: first, it presents detailed experiments explaining why the proposed phase-based model outperforms the analytic instantaneous energy estimation models. Second, it elaborates how τ acts as a smoothing parameter and shows the effect of different values of τ on estimation accuracy. Lastly, it presents a detailed description and experimental results from using our speed profile generation model, demonstrating how the variances in energy consumption and distance travelled over time in synthetic trips matches those observed in real-world trips.

2.2 Background and related work

Many route planning algorithms were developed in the last fifteen years, following the 9th DIMACS challenge [13, 149]. Route planning for conventional vehicles is usually treated as a shortest-path problem, with edge weights modeling time or distance. EV route planning is more complex, and requires three components: estimating EV behaviour along

a path, an energy model to map EV behaviour to energy consumption, and a route planning algorithm that can trade-off energy for time or distance. These components have typically been addressed in isolation.

Estimating vehicle behaviour along a path: Traffic engineers routinely monitor, model, and predict aggregated traffic parameters such as flow, speed, and travel time for routes [86, 117, 145, 163]. Such models use statistical methods such as ARIMA [163] and neural networks [86, 117, 145] to learn vehicle speed and flow at different times of the day using data collected from sensors on roads. Other work [118] tries to estimate individual vehicle speed profiles from aggregated traffic parameters at different points along a path.

In [82, 147], each step in a random walk on a Markov chain gives the speed estimate for one second. In contrast, our trip generation model (Section 2.4) models vehicle behaviour as a series of phases. We use a phase-centric, rather than an edge-centric approach, since per-edge energy consumption estimates lose information about state transitions within edges, losing accuracy for longer trips, as in [168, 40].

Energy estimation models: Energy estimation models vary widely in complexity, speed, resource requirements and accuracy. They may derive their estimates from historical trips [117, 128], analytically [54], or a combination of both [171, 94]. In terms of temporal granularity, energy estimation models can be *microscopic, mesoscopic* or *macroscopic*. We discuss these three classes of energy estimation models below-

Microscopic models produce an energy estimate per second of the trip, and therefore must make second-by-second calculations, which makes them slower and more computationally expensive than models belonging to the other two categories. They also require an accurate per-second model of vehicle behaviour as input [171, 165, 90, 128, 54]. However, they are useful when high per-estimate accuracy is required. Sophisticated powertrain simulators like Autonomie [7] and FASTSim [107] may also be used as microscopic models.

Mesoscopic models are temporally coarser, and provide energy estimates for aggregated representations of vehicle behaviour such as micro-trips [143], kinematic parameters such as speed and distance [42], transits on network edges [41, 158], or spatial features such as road segments [143, 98, 81, 128]. When static edge energy costs are to be applied to each edge of the road network, generalized frameworks such as EcoMark 2.0 [71] can be be used to compare the accuracy of edge weights obtained from different energy estimation models.

However, finding the right aggregated representation (or 'abstraction') of movement for EVs is a harder problem than such models acknowledge. It is known [92] that the movement abstraction used affects both accuracy and computational cost. The use of both mechanical and regenerative braking complicates matters. Low decelerations at high speeds recover no energy. Neither do very high decelerations, where braking is mechanical. Moreover, a time lag typically precedes any regenerative energy recovery, since the electrical system must switch from battery drain mode to charge mode. Even worse, both the range of accelerations in which energy recovery occurs, and the switch delay are highly dependent on the EV-specific hardware. Therefore, *tunability* is critical for mesoscopic models.

SIDRA-4Mode [32], a mesoscopic model for internal combustion vehicles, defines a set of four modes, namely, idle, cruise, acceleration and deceleration. Trips are modeled the vehicle having followed a sequence of modes, and energy consumption is estimated for each mode analytically. While this usage of four modes to represent vehicle movement appears analogous to our idea of phases, it should be noted that models like SIDRA-4Mode, due to being designed for internal combustion vehicles, are not *tunable*, and thereby ignore the complexity induced in energy estimation models due to EV-specific issues. We make the phases abstraction effective through proper parameterization. We show that using a single tuning parameter τ suffices for an accurate energy consumption model for EVs.

Macroscopic models are the coarsest, and model energy consumption using aggregated parameters like average speed, total elevation gain [12, 158], or road type [170]. Macroscopic models are fast, but likely to be the least accurate in estimating energy consumption for individual vehicles. Some models rely on clustering or machine-learning [94, 143, 43, 41] to map the vehicle behaviour along a path to the energy consumption and hence require large training datasets. Such datasets are often hard to obtain, so these models may be especially unsuitable when estimates are required for several different paths on the underlying road network or under varying traffic conditions. Our estimation model (Section 2.3.2), is analytic, and does not depend on historical trip data.

EV route planning algorithms: Some recent route-planning and speedup methods seek battery-optimal paths [8, 23, 139, 50]. Others are based on the constrained shortest path problem, and add time as another weight on the edges and seek minimum-time paths while maintaining battery feasibility [151, 18]. Other approaches provide the full set of pareto-

optimal paths accounting for both time and energy consumption [20]. Some works also model the time-energy consumption tradeoff as per-edge tradeoff functions [26, 25]. Another degree of freedom in these models is the presence and the type of recharging stations that they account for. Early work on electric vehicle routing tends not to consider recharging stations [8, 139, 50]. Some recent work accounts for specific types of recharging stations (say, battery swapping stations) [154, 67], or for variable recharge times at such stations [18, 99].

2.3 EV modeling and energy estimation

We present the standard instantaneous model, define phases, and develop a phase-based energy estimation model.

2.3.1 An instantaneous EV model

We consider vehicle paths on the X, Y plane. We use the following definitions in the rest of the paper: A *path* P is a sequence of points $\{x_1 x_2 ... x_{n-1} x_n\}$, where $x_i \in \mathbb{R}^2$. A *speed profile* $S = \{v_1 v_2 \dots v_{n-1} v_n\}$ of a moving vehicle is a sequence of instantaneous speeds $v_i \in \mathbb{R}$. A *trip* $T = \langle S, P \rangle$ consists of a speed profile S over a path P. v_i is the speed of vehicle at point x_i.

Problem definition: An EV with a given set of vehicle-specific parameters starts from rest at source $s \in \mathbb{R}^2$ and travels to a destination $t \in \mathbb{R}^2$ along a path P with a speed profile S. Our goal is to accurately estimate the total energy consumed by EV while travelling along the trip $T = \langle S, P \rangle$.

Table 2.1: Our notations

Symbol	**Parameter**	**Value**
M	Mass of EV	1961 kg
A_f	Frontal area	2.3316 m^2
ρ	Air density	1.2256 kg/m^3
c_1	Rolling resistance constant	5.74×10^{-5}
c_2	Rolling resistance constant	0.0080
β	gc_1	5.62×10^{-4}
γ	gc_2	0.0784
C_D	Drag coefficient	0.28
D	$(\rho A_f C_D)/(2M)$	2.0401×10^{-4}
η_1	Drivetrain loss factor	1.1944
η_2	Recovery efficiency	0.62
$v(t)$	Speed at time t	
$a(t)$	Acceleration at time t	
$\theta(t)$	Road's angle	
τ	phase threshold	
(S, P)	(Speed profile S, path P)	
$E_I(S)$	Instantaneous model's energy estimate	
$E_\Phi(S)$	Phase-based model's energy estimate	
$E_M(S)$	Measured energy consumption	

Estimating energy consumption

To move the EV along P, the vehicle powerplant must do enough work to maintain the vehicle at speed $v(t)$, working against gravity on inclines and overcoming dissipative losses due to rolling friction and wind resistance. The standard approach models these three dissipative forces as follows (see [54], for example).

The drag force is $\left(\frac{\rho A_f C_D}{2}\right) v^2(t)$, where ρ is the air density, A_f is the vehicle's aerodynamic cross section and C_D is the drag coefficient. To simplify our equations, we define $D = \frac{\rho A_f C_D}{2M}$, where M is the mass of the EV.

The rolling resistance is $N(c_1 v(t) + c_2)$, where c_1 and c_2 are constants and $N = Mg\cos\theta(t)$ is the normal reaction of the road surface inclined at angle $\theta(t)$.

The gravitational resistance to motion is $Mg\sin\theta(t)$.

We assume that the EV is fitted with regenerative brakes, and account for drivetrain and regenerative braking inefficiencies as follows. Let P_0 be the power that the powerplant must produce if the drivetrain were 100% efficient, and P be the power that it must actually produce to have the same effect on the vehicle. Let Q_0 be the energy recovered when the regenerative brakes are 100% efficient, and Q be the actual energy recovered. We define $\eta_1 = \frac{P}{P_0}$ and $\eta_2 = \frac{Q}{Q_0}$. Since the direction of current is reversed in the two cases, we use $\eta_1 > 0$ and $\eta_2 < 0$.

Let $a(t)$ be the (signed) acceleration at time t. Let

$$\alpha(t) = a(t) + Dv^2(t) + g\cos\theta(t)(c_1 v(t) + c_2) + g\sin\theta(t) \tag{2.1}$$

Assuming the EV starts from rest, let $v(x)$ be the speed, $a(x)$ be the acceleration and $\theta(x)$ be the road's angle at a distance x from the start. Then, $v(x) = \sqrt{2a(x)x}$. Transforming $\alpha(t)$ to make distance traveled x the independent variable,

$$\alpha'(x) = a(x) + 2Dxa(x) + g\cos\theta(x)[c_1\sqrt{2a(x)x} + c_2] + g\sin\theta(x) \tag{2.2}$$

Since drivetrain and recovery efficiencies are modeled separately, the force to be overcome by the powerplant during acceleration at position x is

$$G(x) = M\alpha'(x) \tag{2.3}$$

The energy consumed (or recovered) over a trip segment of length d is given by

$$E_I(T) = \eta \int_0^d G(x)dx, \quad \eta = \begin{cases} \eta_1 \text{ if accelerating,} \\ \\ \eta_2 \text{ if decelerating.} \end{cases} \tag{2.4}$$

Microscopic energy estimation models, such as the one above, estimate energy consumption for each instant and aggregate to get the total energy consumption. Such models can be computationally expensive, but have generally been regarded as the most accurate class of models [1, 158].

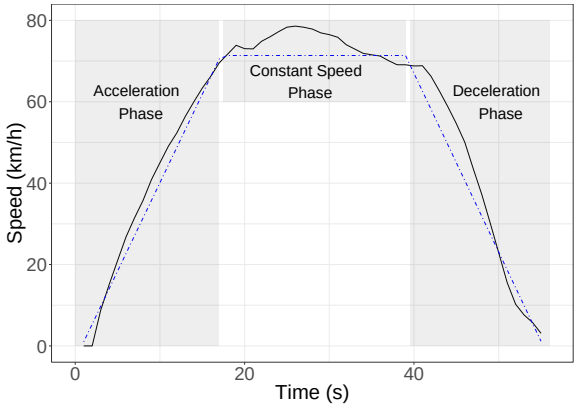


Figure 2.1: The solid line is a speed profile segment for a 2013 Nissan Leaf. For $\tau = 0.5m/s^2$, we see three *phases* of average accelerations 1.20, 0 and -1.21 m/s^2 respectively. The dotted blue line is the modeled velocity in each *phase*.

2.3.2 Mesoscopic EV modeling with phases

Intuitively, a *phase* is a trip segment defined by an acceleration profile. Let τ be a *phase threshold* and let a be the average acceleration of the vehicle during time interval (t_1, t_2). The vehicle is in an *acceleration phase* during (t_1, t_2) if $a > \tau$, a *constant speed phase* if $-\tau \leq a \leq \tau$, or a *deceleration phase* if $a < -\tau$. An example appears in Figure 2.1.

Let $S = \{v_1 v_2 \dots v_N\}$ be a speed profile given as a series of speeds indexed by time instants. At instant k, the EV is accelerating if $v_{k+1} \geq v_k$, and decelerating if $v_{k+1} < v_k$. Without loss of generality, we classify $v_{k+1} = v_k$ as acceleration. We can now assign one of the labels 'A', 'D' or 'R' to each instant, according to whether the vehicle is accelerating, decelerating or at rest. Let the speed profile S generate the *label sequence* $\lambda_S = \{L_1 L_2 \dots L_M\}$, where $L_i \in \{A, D, R\}$.

Next, we define a *run* as a series of consecutive of 'A's or of 'D's in a label sequence. Given an acceleration threshold τ, we define four types of *phases* as follows:

Acceleration Phase: A maximal run of 'A' labels during which the average acceleration exceeds τ.

Deceleration Phase: A maximal run of 'D' labels during which the average acceleration is lower than $-\tau$.

Constant Speed Phase: A maximal run of 'A' or 'D' labels where the average acceleration is between $-\tau$ and τ inclusive.

Rest Phase: A maximal run of 'R' labels.

Algorithm 1 gives pseudocode to identify phases. Given a speed profile S, the label sequence λ_S of S, and a *phase threshold* τ, it finds the runs for which the average acceleration lies in the ranges given in the definitions above. It also partitions the speed profile S into a set $\Phi_A(S)$ of acceleration phases, a set $\Phi_D(S)$ of deceleration phases, a set $\Phi_C(S)$ of constant speed phases and a set $\Phi_R(S)$ of rest phases. Let $\Phi(S) = \Phi_A(S) \cup \Phi_D(S) \cup \Phi_C(S) \cup \Phi_R(S)$ be the set of phases for S.

In practice, the EV's instantaneous speed suffers transient fluctuations due to factors such as driver behavior, road conditions, traffic, and wind speed. Applying the definition of phases as continuous label runs directly to a real-world speed profile is likely to produce many short constant speed phases of only a few seconds each. However, our experiments show that modeling the speed profile at this level of detail can be counterproductive. Therefore, we use Algorithm 2 to aggregate consecutive constant speed phases into longer, but fewer constant speed phases.

Algorithm 1 Identify *phases* in a speed profile

procedure ExtractPhases(S, λ_S, τ)
  $\Phi_A, \Phi_C, \Phi_D, \Phi_R \leftarrow \phi$
  for $i \leftarrow 1$ *to* $|S|$ **do**
    if $\lambda_S^i =$ 'R' **then**
      $\phi_C \leftarrow$ maximal *run* of 'R'
      $\Phi_R \leftarrow \Phi_R \cup \{\phi_C\}$
    else if $\lambda_S^i =$ 'A' **then**
      $\phi_A \leftarrow$ maximal *run* of 'A'
      if average acceleration in $\phi_A > \tau$ **then**
        $\Phi_A \leftarrow \Phi_A \cup \{\phi_A\}$
      else
        $\Phi_C \leftarrow \Phi_C \cup \{\phi_A\}$
      end if
    else if $\lambda_S^i =$ 'D' **then**
      $\phi_D \leftarrow$ maximal *run* of 'D'
      if average acceleration in $\phi_D < -\tau$ **then**
        $\Phi_D \leftarrow \Phi_D \cup \{\phi_D\}$
      else
        $\Phi_C \leftarrow \Phi_C \cup \{\phi_D\}$
      end if
    end if
    increment i by the length of added phase
  end for
  return $\Phi_A, \Phi_C, \Phi_D, \Phi_R$
end procedure

2.3.3 Energy Consumption over a Phase

Equation 2.4 yields the energy consumed if the force experienced at a distance x from the start is $G(x)$ in some phase. Let $E(p)$ denote the energy consumption in phase p. Let a_p be the average acceleration (or deceleration), d_p be the distance traveled, and u_p be the velocity at the start of phase p. For a flat road, $\theta \equiv 0$. We define $\beta = gc_1, \gamma = gc_2$. Let the drivetrain and recovery efficiency factors be η_1 and η_2. Given speed profile S, we get upon integrating Equation 2.4,

$$E(p) = \begin{cases} M\eta_1[Du_p^2 d_p + \beta u_p d_p + \gamma d_p], & p \in \Phi_C(S) \\ M\eta_1 E, & p \in \Phi_A(S) \\ M\eta_2 E, & p \in \Phi_D(S) \\ 0, & p \in \Phi_R(S) \end{cases}, \tag{2.5}$$

where

$$E = a_p d_p + D d_p(u_p^2 + a_p d_p) + \frac{2\beta}{3a_p}(u_p^2 + 2a_p d_p)^{\frac{3}{2}} + \gamma d_p.$$

When the road slopes, $\theta \not\equiv 0$, and the $\sin\theta$ and $\cos\theta$ terms are relevant. We find, however, that it is unnecessary to model the effects of gravity in a microscopic manner for each instant, as other models do. Instead, if the end points of a phase p are q_i and q_j, we simply correct for the work $W_p = Mg(h(q_j) - h(q_i))$ done to raise the EV between q_i and q_j.

Algorithm 2 Aggregate consecutive phases in Φ_C

procedure ConstantSpeedPhaseAggregation(Φ_C)
  for $i \leftarrow 1$ *to* $|\Phi_C|$ **do**
    $t \leftarrow i$
    while ϕ_t and $\phi_{(t+1)}$ are consecutive segments of trip **do**
      $\phi_i \leftarrow \{\phi_i \cup \phi_t\}$ ▷ ϕ_i is the i^{th} phase in Φ_C
      $\Phi_C \leftarrow \Phi_C - \phi_t$
      $t \leftarrow t + 1$
    end while
    $\Phi_C \leftarrow \{\Phi_C \cup \phi_i\}$
  end for
  return Φ_C
end procedure

The energy consumption $E_\Phi(T)$ for a trip $T = \langle S, P \rangle$ is

$$E_\Phi(T) = \sum_{p \in \Phi(S)} (E(p) + W_p) \tag{2.6}$$

2.4 Generating realistic speed profiles

Synthetic speed profiles are needed for *prospectively* estimating energy and time along a path P, for route planning, or in models like POLARIS [10] to estimate region-wide energy use. However, we must also generate realistic variations in the speed profiles of different trips along the same P. We show how to use the phase abstraction to capture enough information from historical trips to generate synthetic trips that model the time taken and energy consumed accurately.

Our objective is not to produce spatially accurate instantaneous speed profiles, as [117, 118] seek to do. Rather, it is to generate *realistic* speed profiles sufficiently close to historical trips to permit accurate distance and energy estimates. Thus, our generated speed profiles need not have spatial fidelity over the path P. We could generate equivalent

vehicle behaviour at a point different from that in input speed profiles, and still produce accurate time and energy estimates.

2.4.1 Trip modeling with Markov Chains & KDE

Let $\Lambda = \{\lambda_1, \lambda, \ldots, \lambda_N\}$ and $\Phi = \{\Phi_1, \Phi_2, \ldots, \Phi_N\}$ be the set of label sequences and set of phases, respectively, for historical trips $T_1, T_2, \ldots, T_N$. We learn the *phase transition probabilities* and *phase duration and accelerations* from $T_1, T_2, \ldots, T_N$. We use a four-state Markov chain to simulate phase transitions and a set of 2-D kernel density estimates for accelerations and time durations in each phase. We show that we can produce realistic speed profiles from these parameters.

Definition 1. *An EV Behaviour Model is* $\Gamma = \langle M, K \rangle$*, where* M *is the Markov phase transition model and* K *is the set of phase parameter PDFs, extracted from historical trips.*

Markov modeling of phase transitions

A Markov chain is a stochastic model comprising *states* and state transition probabilities. The probability of switching to state X_{i+i} from state X_i at time step i depends only on the transition probability $P_{X,Y} = \frac{N_{X \to Y}}{N_X}$, where $N_{X \to Y}$ is the number of times the chain transitions from state X to Y and N_X is the total number of times the chain is in state X.

We use a Markov chain having four states, one corresponding to each type of phase. If state $X \in \{\text{`}A\text{'}, \text{`}D\text{'}, \text{`}C\text{'}, \text{`}R\text{'}\}$, the transition matrix M for the Markov chain becomes

$$M = \begin{bmatrix} P_{A,A} & P_{A,C} & P_{A,D} & P_{A,R} \\ P_{D,A} & P_{D,C} & P_{D,D} & P_{D,R} \\ P_{C,A} & P_{C,C} & P_{C,D} & P_{C,R} \\ P_{R,A} & P_{R,C} & P_{R,D} & P_{R,R} \end{bmatrix},$$

where $P_{X,Y} = \frac{N_{X \to Y}}{N_X}$. Naturally, M is right-stochastic.

KDE modeling of phases

We extract phase durations and average accelerations from Λ and Φ as a 2-D histogram of the observed time duration and accelerations for the 'A', 'D', and 'C' phases. For 'R' phases, a 1-D histogram of time duration suffices. Sampling these histograms directly yields poor estimates when N is small. Hence, we use kernel density estimation with a truncated Gaussian kernel to fit a 2-D PDF of the duration and accelerations of the 'A', 'D', and 'C' phases. For 'R' phases, a 1-D Gaussian kernel is used.

Regimes

We would expect EV behavior on city roads to differ significantly from that on highways. A *regime* is a trip section with consistent behavior. We learn a regime-specific behavior model $\Gamma_R = \langle M_R, K_R \rangle$ for each regime R. The set of regimes observed may vary by trip. For example, trips that also include unpaved or graveled roads may need more regimes than trips that only use paved roads and highways.

Generating synthetic speed profiles

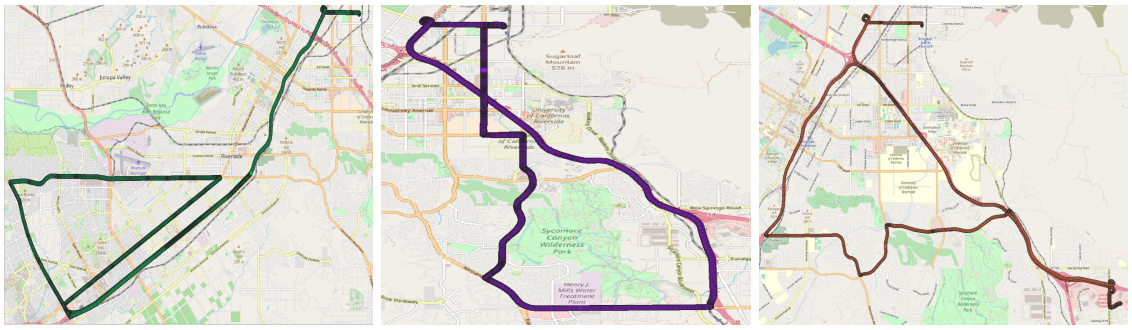
Let x denote the distance along the route. If regime R_k applies to road segment (x_k, x_{k+1}), we use Γ_{R_k} to generate a trip section for that segment. We expect some variation in x_k and x_{k+1} over trips, and use their mean observed starting and ending locations. Say the EV enters regime R_k in phase p_1. The next trip phase p_2 is obtained from a random walk on M_{R_k} starting from state p_1, and a sample drawn from the estimated PDF in K_{R_k} for p_2 to get the average acceleration and duration of p_2. This random walk terminates when the EV has arrived at x_{k+1}. The full generated speed profile over path P is the concatenation of the speed profiles for all the regimes along P.

2.5 Experiments

We now validate the phase-based energy estimation model and speed profile generation method of Sections 2.3 and 2.4.

2.5.1 Validating the phase-based energy estimation model

For a description of data post-processing, see [128]. We compare the energy estimates of our model (Section 2.3.1) with those of the analytic microscopic estimation models of [54], and observed real-world energy consumption. the Center forEnvironmental Research & Technology at UC Riverside. We are grateful to the authors of [171, 128] for this data.



(a) Route 1 (b) Route 2 (c) Route 3

Figure 2.2: EV routes for data collection in Riverside, CA.

Data collection and processing

The real-world dataset was collected from a 2013 Nissan Leaf over three routes in (location elided for anonymity) for 52 hours (Figure 2.2). The EV was fitted with a Trimble Lassen iQ GPS receiver and a CONSULT III plus kit to log vehicle parameters such as latitude and longitude, battery voltage and current, and auxiliary power consumption for air conditioning, etc. Data was collected for 10 trips along Route 1, 16 trips along Route 2 and 26 trips along Route 3. Each trip lasted an hour. The EV always started with a battery charge state greater than 60%. The raw data from the vehicle sensors was processed as follows:

Frequency synchronization: GPS and CONSULT III kit data collection frequencies were synchronized, and output at 1Hz.

Trip data synchronization: the GPS and CONSULT III kit time reference bases were coordinated. Cross-correlation was applied using vehicle speed to synchronize their data streams.

Map matching: Map matching was used align the GPS traces with locations on the underlying road network.

Elevation data fusion: Elevation data from a high-resolution Trimble R8 device was added to the the map-matched traces.

Phase-based energy estimates

We compared the energy consumption estimates $E_I(T)$ of the standard instantaneous model (Eqn. 2.4) and the estimates $E_\Phi(T)$ of our model (Eqn. 2.5, 2.6) against the ground truth $E_M(T)$, the actual measured energy consumption of the EV. Figure 2.3 shows the percentage errors.[1] The EV parameter values are as in [54], and shown in Table 2.1.

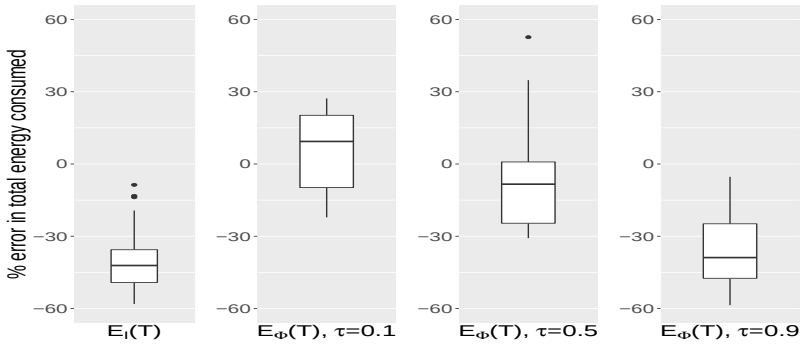


Figure 2.3: Instantaneous model $E_I(T)$ vs phase-based model $E_\Phi(T)$. Negative errors are underestimates by model.

The box-and-whisker plots in Figure 2.3 show the percentage error distribution of estimates by the instantaneous model and our model with $\tau \in \{0.1, 0.5, 0.9\}$. Our model outperforms the instantaneous model, producing percentage errors much closer to zero. We discuss our choice of τ in Section 2.5.1.

The model of [43] tries to augment the simple instantaneous estimation model of Section 2.3.1 by trying to account for factors such as the weather, temperature, road

[1] Percentage error is $100 \times \left(\frac{\text{estimate}-\text{measured}}{\text{measured}}\right)$.

types and traffic lights. Since each of these factors is highly variable and can cause speed profiles to vary significantly, a mesoscopic model accounting for these factors would need to consider a number of abstractions exponential in the number of factors considered. Such approaches yield little real gain, yet add unnecessary complexity and reduce the generality of the instantaneous model. Systems such as Autonomie [7] and FastSIM [107] try to work with more precise vehicle characteristics such as power and efficiency maps for the EV motor. However, they still lack the detailed engine models for the EV used in our experiments, a Nissan Leaf 2013.

MAPE (Mean Absolute Percentage Error)[2] is a performance metric widely used for estimation models [66, 105]. Figure 2.4 shows that our mesoscopic model achieves per-estimate MAPE very close to that of the microscopic instantaneous model. This is an unexpected result, and a revealing insight: microscopic models may not always be the best option for EV energy estimation, even for simple metrics, such as total energy consumed over an entire trip. They do offer better per-estimate results, but are much more expensive, accumulate error quickly, and are prone to drift.

Understanding τ

We can see τ as a *smoothing parameter*, not just as a way to aggregate microscopic behaviour into mesoscopic phases. Speed fluctuates constantly, due to factors such as driver action or the environment, but vehicle behavior is broadly consistent within phases,

[2] If we make n estimates over a trip, and the percentage errors for these estimates are $e_1, e_2, \ldots, e_n$, then MAPE $= \frac{|e_1|+|e_2|+\cdots+|e_n|}{n}$.

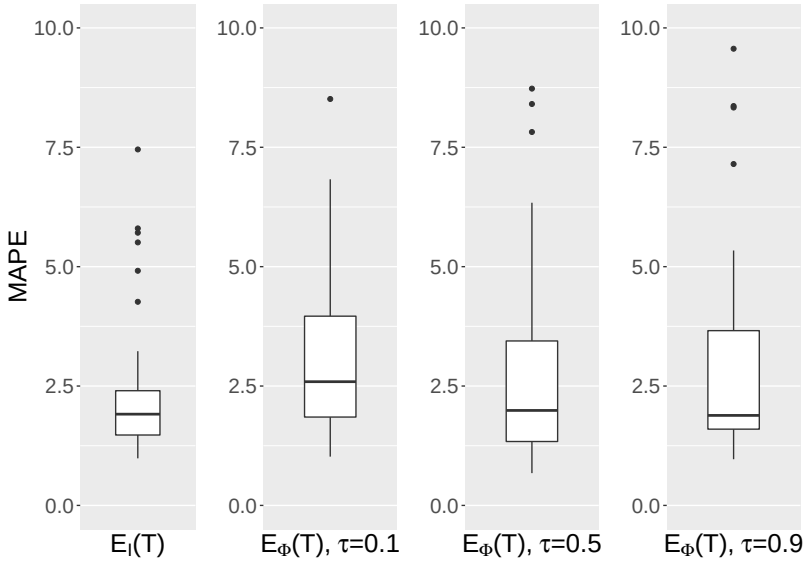


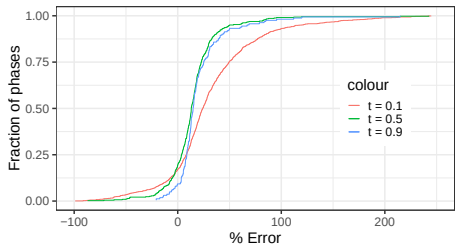
Figure 2.4: Mean Absolute Percentage Error (MAPE) for $E_I(T)$ and $E_\Phi(T)$, for various τ.

despite any fluctuations. Hence, it is overkill to model instantaneous speed and acceleration, applying η_1 or η_2 to compute energy expenditure or recovery, as microscopic models do. Thresholding the average acceleration over a whole run using τ smoothes out local variations, and captures the high-level behavior well.

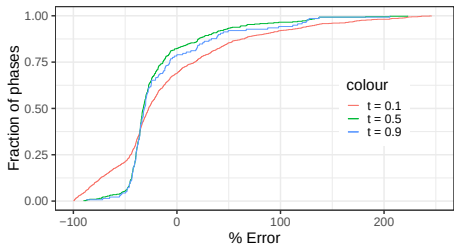
This approach also allows us to tune our model's *sensitivity* to transient behaviour. Lower τ increases sensitivity to local fluctuations, and higher τ has the opposite effect. At $\tau = 0$, our model becomes the instantaneous energy consumption model. Table 2.2 gives the mean percentage of time spent in different phases for different τ values. Clearly, small changes in τ can significantly affect how a trip is modeled as phases.

Effect of τ on estimation error

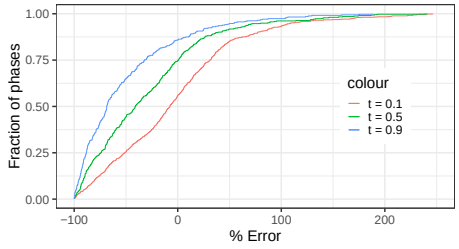
Figure 2.5 shows the CDF of the percentage deviations from real-world measurements for each phases type for various τ. Figure 2.5(a) and (b) show that lowering τ



(a) Errors in acceleration phases



(b) Errors in deceleration phases



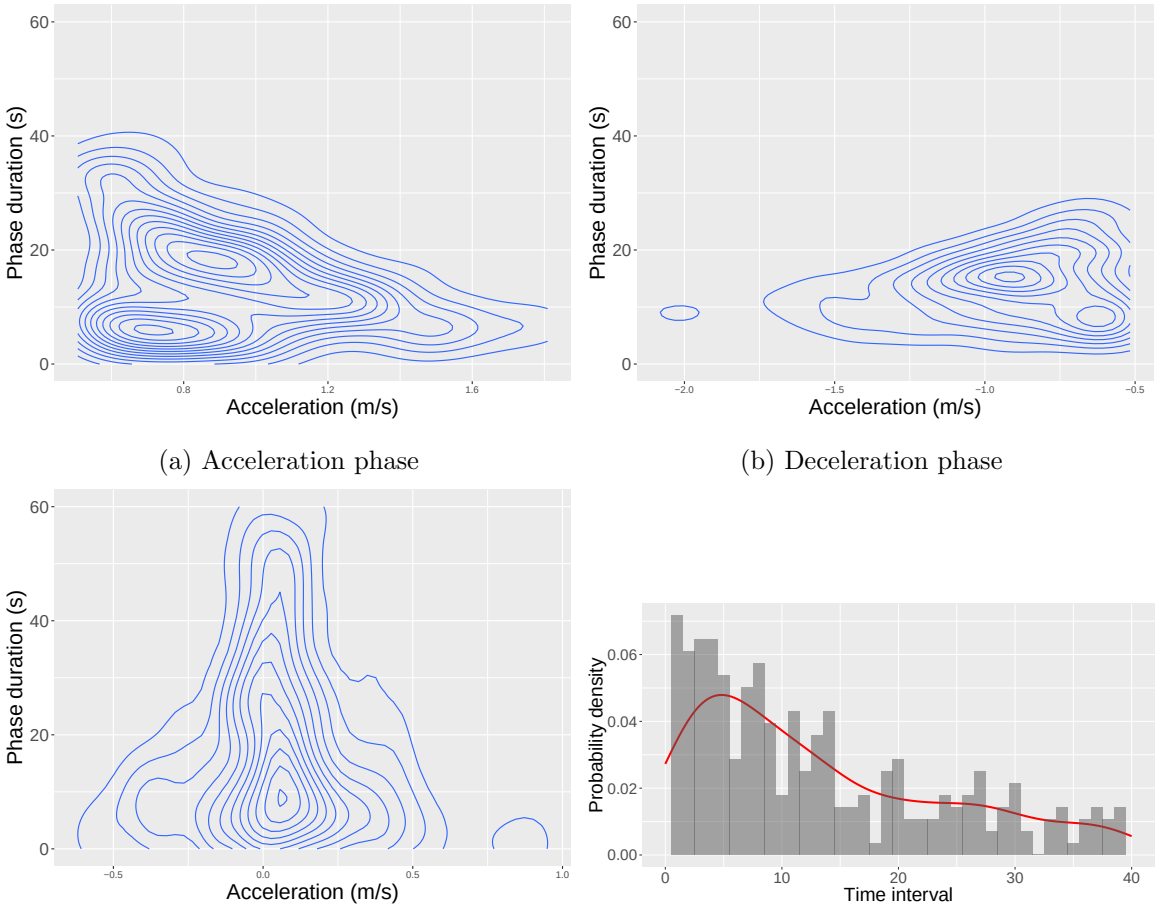
(c) Errors in constant speed phases

Figure 2.5: CDF of % deviation from real-world measurements, for different phase types. In acceleration and deceleration phases, higher τ values yield more accurate estimates, while in constant speed phases, the reverse is true. See Section 2.5.1.

Table 2.2: Effect of τ on mean % of time spent in each *phase*.

Phase Type	$\tau = 0.1$	$\tau = 0.5$	$\tau = 0.9$
Acceleration	45.3%	18.8%	4.5%
Deceleration	35.0%	15.9%	4.8%
Constant Speed	19.6%	65.2%	90.6%

increases errors in acceleration and deceleration phases, since it is now easier for a local transient to falsely qualify as an acceleration or deceleration phase. This holds up to a point; the CDF of errors for $\tau = 0.5$ and $\tau = 0.9$ are similar. Conversely, Figure 2.5(c) shows that higher τ raises the error in constant speed phases, since it is harder for a phase to qualify as an acceleration or deceleration phase, raising the probability of misclassification as a constant speed phase.



(a) Acceleration phase

(b) Deceleration phase

(c) Constant speed phase

(d) Rest phases

Figure 2.6: Using KDE to learn the PDF of the acceleration and time spent in each type of phase. The contour plots show the 2-D PDF for trips on Route 1, with $\tau = 0.5$. Part (d) shows the KDE estimates for the time interval PDF for rest phases.

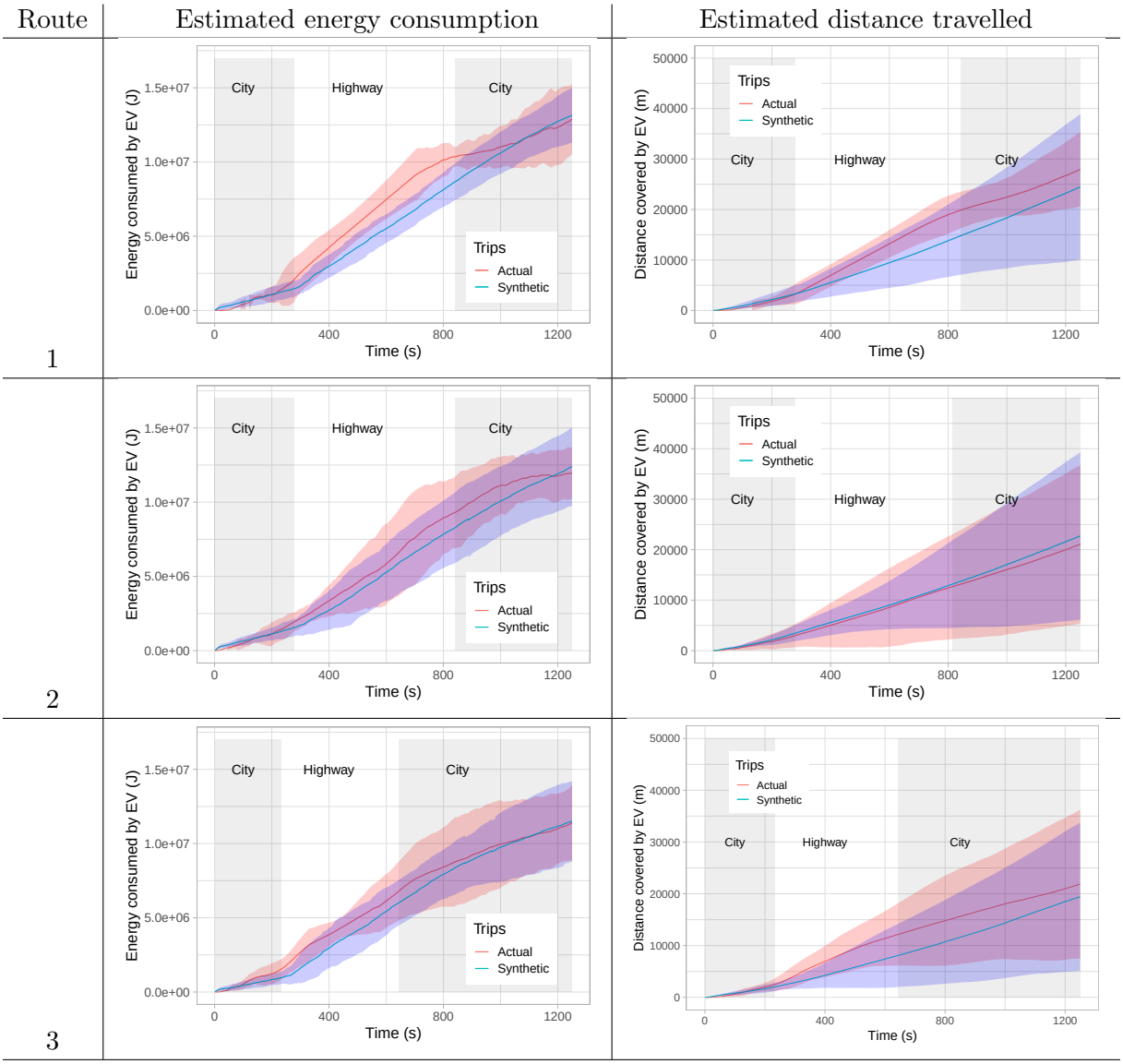


Figure 2.7: Energy consumed and distance traveled versus time for actual trips versus 100 trips generated by our model. Solid lines show mean values, and error bands show 95% confidence intervals. The mean distance traveled and energy consumption over time for observed and synthetic trips are close, with significant overlap in the error bands. This shows that our model generates trips that match the natural variance in energy consumption and distance traveled over time.

$\tau = \{0.1, 0.5, 0.9\}$ give the best results for our dataset and for the metrics we use. However, the optimal τ depends on EV specifics, the metric used, and on factors such as traffic and weather. We offer the following heuristic: a lower τ value is better for trips where accelerations and decelerations are frequent, such as travel along roads with traffic and traffic lights. A higher τ is recommended for trips with longer sections of uninterrupted high speeds, say, on highways. Despite its simplicity and use of only one tuning parameter, our model produces high-quality energy consumption estimates which are close to real-world values.

2.5.2 Evaluating the speed profile generation model

Table 2.3: Regime parameters used to generate synthetic trips

Route	Regime	start, end time (s)	τ
1	City	(0, 280)	0.5
1	Highway	(281, 842)	0.9
1	City	(843, 1250)	0.5
2	City	(0, 280)	0.5
2	Highway	(281, 815)	0.9
2	City	(816, 1250)	0.5
3	City	(0, 232)	0.5
3	Highway	(233, 643)	0.9
3	City	(644, 1250)	0.5

Two regimes, 'City' and 'Highway' suffice to model our dataset. Figure 2.6 shows the contour plots for PDFs estimated for the 'City' regime from trips along Route 1.

We generated a hundred synthetic speed profiles for each of the three routes in our dataset, using historical trips between 10–11 AM. Figures 2.7 plots the energy consumed and distance traveled versus time for the generated and the real-world trips over Routes

1–3. It also shows the regimes along the route and the mean and spread in distance and energy values. On Route 1, for instance, 'City' regimes correspond to time intervals $(0, 280)$, and $(843, 1250)$, and a 'Highway' regime to interval $(281, 842)$. Coincidentally, the first 1250 seconds of trips along all three routes in our dataset consist of sections with regimes in the same order {City, Highway, City}.

Since our approach learns regime-specific models, different τ can be used in each regime. In our data, for instance, $\tau = 0.5$ is best for the 'City' regime, but $\tau = 0.9$ is better for Highway regimes. The slopes of mean energy consumption in Figure 2.7 differ significantly for the regimes. Table 2.3 shows the starting and ending times for each regime and τ value.

Figure 2.7 shows that the variance in energy consumed and distance travelled over time increases with time even for the real-world data. The mean distances over time in generated trips are very close to real-world values within the parameters of this variance. The 95% confidence intervals overlap substantially, showing the accuracy and generality of our model.

Chapter 3

Robustness Generalizations of the Shortest Feasible Path Problem

3.1 Introduction

Several factors can cause an Electric Vehicle (EV) to get stranded along a route: They often have shorter ranges than internal combustion (IC) vehicles, charging stations can be sparse and fragmented among different providers. For drivers, this *stranding risk* manifests as *range anxiety* and *range stress* [49, 61, 124, 125, 133, 140]. To alleviate range anxiety, route planning for EVs must consider battery constraints while selecting routes [50, 93, 94, 152, 154].

Previous work [18, 19] models EV routing with charging stops as the NP-hard Shortest Feasible Path Problem (SFPP): Given a road network modeled as a weighted, directed graph with energy consumptions and travel times on each edge; charging stations

[0]This work was done while the author was an intern at Apple Inc.

on a subset of vertices and their respective concave charging functions; a source vertex, a destination vertex and a starting battery SoC, find a path that minimizes the total travel time including charging time while maintaining a non-zero battery SoC at all points along the route. The Charging Function Propagation (CFP) algorithm solves SFPP in exponential time and space.

In practice, however, the shortest feasible path might not be sufficient. First, the energy consumptions on edges are derived from estimation models that are not perfectly accurate [43, 54, 129, 131]. Second, the energy consumption of an EV depends on several factors that are difficult to even estimate: driver aggressiveness, age of the battery, wear and tear of the EV. Each of these factors can affect the energy consumption significantly [3, 55]. Third, users may have varying risk tolerances, and thus a one-size-fits-all approach is not sufficient to alleviate range anxiety when serving routes for a large number of EV drivers.

In this work, we introduce two generalizations of SFPP which are used to compute two constructs, the *Starting Charge Map (SCM)* and *Buffer Map (BM)*. Both SCM and BM are computed between a source vertex s and target vertex t. Evaluating SCM_{st} for a valid starting SoC β_s gives the corresponding shortest feasible path between s and t, while evaluating BM_{st} for buffer energy $\mathfrak{b}$ returns a shortest feasible path where the SoC is guaranteed to never drop below $\mathfrak{b}$ along the route.

The SCM and BM allow route planning systems to access a larger set of alternative feasible paths than the standard CFP algorithm, which returns only a single feasible path. This variety in paths has several applications–recommending EV drivers alternative

routes, generating suggestions like charging extra at s to save travel time, or letting users choose the degree of acceptable risk for a trip. Both problems can be solved by brute force approaches that run CFP for all possible values of β_s or $\mathfrak{b}$. However, since β_s or $\mathfrak{b}$ can take an infinite number of possible values, such an approach would simply not terminate. In this paper, we make the following contributions:

- We introduce the *Starting Charge Map (SCM)* and *Buffer Map (BM)* that encapsulate a set of alternative routes to help alleviate range anxiety for a wide variety of EV drivers.
- Computing SCM and BM using standard CFP requires several expensive runs of the algorithm. We present fast, exact algorithms that compute the two abstractions with acceptable real-time performance on large graphs.
- We evaluate our algorithms on realistic instances, using real-world road networks of California and Oregon, an energy consumption model taken from a Nissan Leaf 2013 [54], and a dataset of public EV charging stations [5]. Our results show good performance even without the use of preprocessing techniques for shortest path queries.

3.2 Related Work

Most current EVs suffice for a majority of trips that drivers take, as shown in [109]. However, *range anxiety*, defined as an EV driver's fear of getting stranded along a route is often cited as a major hindrance to widespread EV adoption [59, 62]. Prior work shows that perceived range anxiety is inversely related to the degree of drivers' *trust* in the

EVs [80, 133, 61, 140, 83]. Route planning for EVs, therefore, has two objectives: Minimize travel times under battery constraints, and reduce *surprise* for the driver to minimize range anxiety.

Early works on EV route planning like [8, 139] consider the problem of minimizing energy consumption along routes insted of standard route planning formulations that minimize travel time [13]. Since then, many additions have been proposed to the EV routing problem to make it more realistic. Several newer variants consider battery-swapping stations [50] or charging functions [18, 25, 99, 155]. Some works [68, 152, 154] model EV routing as a multicriteria Dijkstra's search [97], which returns a set of pareto-optimal routes that are not dominated in either travel time or energy consumption. Conversely, some other works like [18, 25] present EV routing as an extension of the Constrained Shortest Path problem. These problems seek to minimize total travel time including charging time, while constraining the total energy consumption of paths to levels allowed by realistic battery capacities. Another line of research considers "profile queries", which look for all optimal shortest paths depending on a certain state [141], e.g., the initial state of charge of an EV [22, 27] or the current point in time [24, 47, 56].

Underlying all EV routing algorithms is an assumption that the energy consumptions assigned to graph edges are accurate. In practice, this is difficult to achieve with existing energy consumption models [43, 54, 53, 129, 131, 98]. EV energy consumption is affected by several factors including traffic conditions, driver aggressiveness, battery health and regular wear-and-tear of the vehicle, which are hard to estimate. Recently, [3] showed that each of these factors can impact the energy consumption along short routes by as

much as 40%. Similarly, [131] show a high variance in EV energy consumptions for short trips. To mitigate the effects of inaccurate estimates, [140] recommend holding a *safety margin* between 12 and 23% of battery capacity. Only few EV routing algorithms [55, 76] accommodate buffer energy for variance in energy consumption estimates or provide robust routes.

3.3 Preliminaries

Our setup is similar to the standard shortest feasible path problem [18, 19]. We consider a road network modeled as directed graph $G = \langle V, E \rangle$, with V the set of vertices and $E : V \times V$ the set of edges. We are given two edge weight functions $d : E \to \mathbb{R}_{\geq 0}$ and $c : E \to \mathbb{R}$ that assign the travel time and energy consumption to each $e \in E$. An $s - t$ *path* in G is a sequence of adjacent vertices $P = [s = v_1 v_2 \dots v_n = t]$, such that $\forall 1 \leq i \leq n, (v_i, v_{i+1}) \in E$ holds.

For a path P, the total *driving time* is $d(P) = \Sigma_{i=1}^{n-1} d(v_i, v_{i+1})$. The *consumption profile*, $f_P : [0, M] \to [-M, M] \cup \{-\infty\}$ is a function that maps the starting SoC β_s to residual SoC β_t at t after traversing P. f_P can be negative due to energy recuperation along P, or $-\infty$ if it is not possible to traverse P with starting SoC β_s. $f_P(\beta)$ can be computed using a 3-tuple $\langle \text{in}_P, \text{cost}_P, \text{max}_P \rangle$, where in_P is the minimum SoC required at s to traverse P, $cost_P = \Sigma_{i=1}^{n-1} c(v_i, v_{i+1})$, and out_P is the maximum SoC possible at t after traversing P [50]. Conversely, we define an *inverse consumption profile* $f_P^{-1} : [-M, M] \to [0, M] \cup \{\infty\}$, which maps residual SoC β_t to the starting SoC β_s. We evaluate both functions as:

$$f_P(\beta) = \begin{cases} -\infty & \text{if } \beta < \text{in}_P \\ \text{out}_P & \text{if } \beta - \text{cost}_P > \text{out}_P \\ \beta - \text{cost}_P & \text{otherwise} \end{cases}, \; f_P^{-1}(\beta) = \begin{cases} \infty & \text{if } \beta > \text{out}_P \\ \text{in}_P & \text{if } \beta + \text{cost}_P < \text{in}_P \\ \beta + \text{cost}_P & \text{otherwise} \end{cases}$$

Let $f_\phi(\beta)$ and $f_\phi^{-1}(\beta)$ be identity SoC profiles that always map a given SoC β to itself. Given two paths $P = [v_1 v_2 \dots v_k]$ and $Q = [v_{k+1} \dots v_n]$, we can get the concatenation $P \circ Q = [v_1 \dots v_k v_{k+1} \dots v_n]$ and a linked consumption profile $f_{P \circ Q}$ as $\text{in}_{P \circ Q} = max\{\text{in}_P, \text{cost}_P + \text{in}_Q\}$, $\text{out}_{P \circ Q} = min\{\text{out}_Q, \text{out}_P - \text{cost}_Q\}$, and $\text{cost}_{P \circ Q} = max\{\text{cost}_P + \text{cost}_Q, \text{in}_P - \text{out}_Q\}$, if $\text{out}_P \geq \text{in}_Q$; otherwise, $P \circ Q$ is infeasible and $f_{P \circ Q} \equiv -\infty$. Lastly, an (inverse) SoC profile f_1 is said to *dominate* f_2 if $\forall \beta \in [0, M], f_1(\beta) \geq f_2(\beta)$.

A set $S \subseteq V$ marks the available charging stations on the road network. Each $v \in S$ is assigned a concave, monotonically increasing *charging function* $\text{cf}_v : \mathbb{R}_{\geq 0} \to [0, M]$ that maps the charging time at v to the resultant SoC after charging. Conversely, we also define the inverse charging function $\text{cf}_v^{-1} : [0, M] \to \mathbb{R}_{\geq 0}$. To obtain the time it takes to charge from β_1 to β_2, we compute $\text{cf}^{-1}(\beta_2) - \text{cf}^{-1}(\beta_1)$.

Definition 2. *A* shortest feasible path P *between a source* $s \in V$ *and a target* $t \in V$ *for an EV with a starting SoC* $\beta_s \in [0, M]$ *is one that minimizes the total trip time (travel time + charging time) while maintaining a non-negative battery SoC at all points on* P*.*

For this work, we add two constraints to the original definition of charging functions: First, we require that all charging functions have a minimum initial SoC of 0 and are able to fully charge EVs to M SoC. This constraint is realistic as any real-world charging

station can charge an EV with an empty battery to its full capacity. Second, similar to [18], we require all charging functions to be *piecewise linear*.

3.3.1 Charging Function Propagation (CFP)

CFP [18, 19] is a generalization of the bicriteria Dijkstra's algorithm [97] with two major differences: First, the set of labels at a vertex represent all possible tradeoffs between charging time and the resultant SoC after charging at the last station, and second, the decision how much to charge at a station is taken at the immediately following station the EV visits. This is because the amount of charge needed at $u \in S$ is dependent on energy consumed by the EV between u and the next station $v \in S$. If v_i and v_j are two consecutive charging stations on a path $P = [v_1 \dots v_n]$, we call the subpath $[v_i \dots v_j]$ a *leg* of P.

Assume that we want to find a shortest feasible path between $s, t \in V$ for starting SoC β_s. For all $v \in V$, we maintain sets $L_{uns}(v)$ for unsettled and $L_{set}(v)$ for settled labels. For vertex v, a label of the CFP search is a 4-tuple $\ell = \langle \tau_v, \beta_u, u, f_{[u \dots v]} \rangle$ where τ_v is the total travel time from s to v except the charging time at the last charging station u, β_u is the EV's SoC on arriving at u and $f_{[u \dots v]}$ is the consumption profile of subpath $[u \dots v]$. The CFP search propagates through G as follows:

1. *At s*: A label $\ell = \langle 0, \beta_s, s, f_\phi \rangle$ is added to the travel time ordered min-priority queue PQ.

2. *Search reaches a non-charging vertex $v \neq t$*: Let path $P = [s = v_1 \dots v_k = v]$ and total travel time $\tau_P = \Sigma_{i=1}^{k-1} d(v_i, v_{i+1})$. Create label $\langle \tau_P, \beta_s, s, f_P \rangle$ and add to $L_{uns}(v)$.

3. *Search reaches first charging station vertex* $v \neq t$: Let path $P = [s \dots v]$ and total travel time over P be τ_P. Create label $\langle \tau_P, f_P(\beta_s), v, f_\phi \rangle$ and add to $L_{uns}(v)$.

4. *Search reaches a non-charging vertex* $v \neq t$: Let $\ell = \langle \tau_v, \beta_u, u, f_{[u \dots v]} \rangle$ be the current label extracted from PQ. Since u is the last charging station, let subpath $P = [u \dots v]$ and the total travel time over P be τ_P. Add label $\langle \tau_{[s \dots v]}, f_{[s \dots v]}(\beta_s), u, f_P \rangle$ to $L_{uns}(v)$.

5. *Search reaches a subsequent charging vertex* $v \neq t$: Let $\ell = \langle \tau_v, \beta_u, u, f_{[u \dots v]} \rangle$ be the current label extracted from PQ. Since u is the last charging station, let *leg* $\mathcal{L} = [u \dots v]$ of path $P = [s \dots v]$, and the total travel time over P be τ_P. Compute the *SoC function* $b_\ell(\tau) := \tau_P + f_{\mathcal{L}}(\mathrm{cf}_u(\beta_u, \tau - \tau_P))$. Since all charging functions are assumed to be piecewise linear, it suffices to create one label per breakpoint of b_ℓ [18]. For breakpoint $B = (\tau_B, SoC_B)$, create a label $\langle \tau_B, SoC_B, v, f_\phi \rangle$ and add to $L_{uns}(v)$.

6. *Search reaches destination* t: Terminate and backtrack to extract a path from s to t.

The label sets for all $v \in V$ are used to minimise the total number of dominance checks among labels for v. L_{uns} is implemented as a min-heap with total feasible travel time as the key, and the following invariant is maintained: The minimum label ℓ in $L_{uns}(v)$ is not dominated by any label in $L_{set}(v)$. Label ℓ dominates ℓ' iff $b_\ell(\tau) \geq b_{\ell'}(\tau)$ when $\tau \geq 0$.

As the number of labels created during CFP search can be exponential, the algorithm belongs to the EXPTIME class. A combination of A* search using potential functions and Contraction Hierarchies [65] can be used to speed up CFP on large graphs in practice. When both speedup techniques are combined, the result is called the CHArge algorithm [18, 19].

3.4 Starting Charge Maps

Definition 3. *For a given source $s \in V$ and target $t \in V$, a* starting charge map $SCM_{st} : [0, M] \rightarrow P$ *is a function that maps a starting charge β_s to the corresponding shortest feasible path P.*

An SCM is a generalization of the shortest feasible path problem where the starting SoC β_s is unknown. First, it can be used to *recommend* users faster routes that they can take if the starting SoC is higher. For example, given an SCM, it is trivial to generate recommendations for EV drivers like "The best path with your current SoC takes 45 minutes, but you might save 10 minutes if you spend 15 more minutes charging at your present location before starting your trip". Such recommendations can be particularly useful to EV drivers for routes with flexible starting times. Second, different trips taken by an EV user might have *different levels of risk aversion*, and an SCM can be used to show users feasible paths that suit the current scenario. As an example, consider two EV trips, the first through an urban area with a high density of charging stations during daytime, and a second trip through a sparsely populated area after nightfall. In the first scenario, most drivers might trade off a higher stranding risk for shorter travel times, while the preferences might be reversed for the second route. SCMs can be used to explore such alternatives and present them to the driver. Asking the driver to charge longer might reduce the risk, while allowing to start with a lower SoC usually increases the risk. Lastly, in most applications, routes are computed on a server and sent to the users on mobile clients. Since battery constraints apply for EV routing, more information about the vehicle needs to be sent to the server than for regular Internal Combustion (IC) vehicles. If instead of individual routes, SCMs

are computed and sent to the client for display, the current SoC no longer needs to be sent to the routing server, which may result in *better privacy* for the drivers.

A brute force approach to compute SCM_{st} is to run the CFP algorithm for all values in $[0, M]$. However, since $[0, M]$ contains an infinite number of values, this is clearly not feasible. Even if we discretize the domain and restrict it to only percentage values that are multiples of a small fixed integer k, running CFP $\frac{100}{k}$ times, once each for $\{0, k, 2k, 3k, ..., 100\}\%$, would still be too slow for interactive routing applications where queries need to be answered quickly. A better approach is to run a series of binary searches in the starting SoC range $[0, M]$ such that on iteration i, the search returns a breakpoint starting SoC $\beta \in [0, M]$, where the shortest feasible paths for starting SoC β and $(\beta + \epsilon)$ differ by at least one edge. However, if $|SCM_{st}| = N$, such an approach would take $N \log N$ runs of the CFP algorithm. In the next section, we present an algorithm that computes SCM_{st} in N runs.

3.4.1 Reverse Charging Function Propagation

First, we introduce the following intermediate problem:

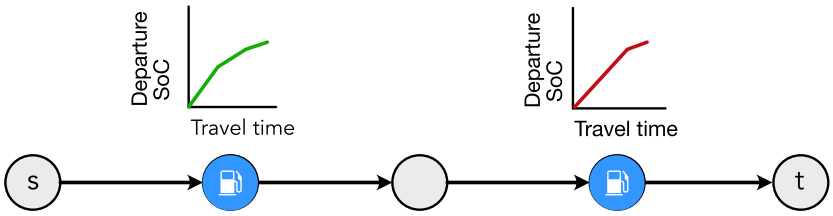
Definition 4. *The Reverse Shortest Feasible Path (RSFP) Problem:*
Given a graph $G = \langle V, E \rangle$, edge weight functions $d : E \to \mathbb{R}_{\geq 0}$ and $c : E \to \mathbb{R}$ that represent travel time and energy consumption on edges respectively, a source $s \in V$ and a target $t \in V$, a set $S \subseteq V$ marked as charging stations, and an SoC β_t, find a shortest path P such that SoC never drops below 0 along P and has a residual SoC at least β_t at t.

As RSFP is closely related to the regular shortest feasible path problem, it can be solved with a *reverse* variant of the CFP algorithm. Note that several operations needed for CFP are not symmetric, e.g., $f(P \circ Q) \neq f(Q \circ P)$. Following, we detail the Reverse Charging Function Propagation (RCFP) algorithm and extend it to compute Starting Charge Maps.

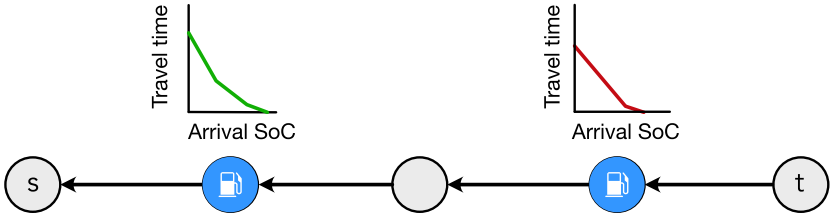
The Reverse CFP works on a backward graph G', obtained by reversing the directions of all edges in G. The RCFP search starts at t with residual SoC β_t and propagates towards s. At $v \in V$, a label ℓ' is defined as $\langle \tau_t, \beta'_u, u, f_{[v \dots u]} \rangle$, with τ_t the total travel time on subpath $[t \dots v]$, u the last charging station encountered in the search, β'_u the SoC *after* charging at u, and $f_{[v \dots u]}$ the consumption profile of subpath $[v \dots u]$.

A key difference between forward and reverse CFP search labels is that while a label ℓ for the forward search contains β_u, the SoC *before* charging at the last charging station u, ℓ' stores β'_u, the SoC *after* charging at u. Computing β'_u is *only* possible in reverse CFP search, because of the following: As forward CFP search reaches v, only the exact energy consumption on $[u \dots v]$ is known, and therefore CFP needs to keep track of all possible charging scenarios at *previous* charging station u until the search reaches t or the next charging station. However, in RCFP search, the exact energy consumption between v and the target or *next* charging station u is known. Thus, as RCFP search reaches a charging station or origin, we know *exactly* how much charge is needed to travel from v to u, and have residual SoC β'_u. Both forward and reverse CFP maintain two label sets for each $v \in V$: $L_{uns}(v)$ for unsettled and $L_{set}(v)$ for settled labels.

Further, for RCFP, we transform all cf_v to inverse charging functions cf_v^{-1}. At $v \in S$, cf_v^{-1} returns the time required to charge an empty battery to resultant SoC β'. Note



(a) CFP runs on G with piecewise linear, concave charging functions.



(b) RCFP runs on backward graph G' with inverse charging functions.

Figure 3.1: Comparing SFP and RSFP problem setups. While charging functions map the time spent charging to an EV's SoC at *departure*, inverted charging functions map the EV's SoC at *arrival* at the charging station to the least possible charging time required to reach target.

that under our assumptions, the inverse charging functions are piecewise linear, convex and monotonically decreasing. RCFP propagates through G' as follows:

1. *At* t: A label $\ell' = \langle 0, \beta_t, t, f_\phi^{-1} \rangle$ is added to the travel time ordered min-priority queue.

2. *Search reaches a non-charging vertex* $v \neq s$: Let path $P = [v = v_1 \dots v_k = t]$ and total travel time be $\tau_P = \Sigma_1^k d(v_i, v_{i+1})$. Create label $\langle \tau_P, \beta_t, t, f_P^{-1} \rangle$ and add to $L_{uns}(v)$.

3. *Search reaches first charging station vertex* $v \neq s$: Let path $P = [v \dots t]$, total travel time over P be τ_P. Create label $\langle \tau_P, f_P^{-1}(\beta_t), v, f_\phi^{-1} \rangle$ and add to $L_{uns}(v)$.

4. *Search reaches a non-charging vertex* $v \neq t$: Let $\ell = \langle \tau_v, \beta_u, u, f_{[u \dots v]} \rangle$ be the current label extracted from PQ. Since u is the last charging station, let subpath $P = [u \dots v]$ and the total travel time over P be τ_P. Add label $\langle \tau_{[s \dots v]}, f_{[s \dots v]}^{-1}(\beta_s), u, f_P^{-1} \rangle$ to $L_{uns}(v)$.

5. *Search reaches a subsequent charging vertex $v \neq s$*: Let $\ell' = \langle \tau_t, \beta'_u, u, f^{-1}_{[u\ldots v]} \rangle$ be the current label extracted from PQ. Since u is the last charging station, let *leg* $\mathcal{L} = [u \ldots v]$ and path $P = [v \ldots t]$. Let the total travel time over P be τ_P. Next, compute the *Starting SoC function* $b'_{\ell'}(\beta) := \tau_P + max(0, \mathrm{cf}^{-1}_u(f^{-1}_\ell(\beta)) - \mathrm{cf}^{-1}_u(\beta'_u))$. Again, since we assume that all inverted charging functions are piecewise linear, it suffices to create one label per breakpoint of $b'_{\ell'}$. For breakpoint $B = (\tau_B, SoC_B)$, create a label $\langle b'_{\ell'}(SoC_B), SoC_B, v, f^{-1}_P \rangle$ and add to $L_{uns}(v)$.

6. *Search reaches destination s*: Terminate and backtrack to extract a path from t to s.

A label ℓ'_1 is said to dominate ℓ'_2 iff $b'_{\ell'_1}(\beta) \leq b'_{\ell'_1}(\beta)$ for $\beta \geq 0$.

Lemma 5. *If a shortest feasible $s-t$ path exists, running the RCFP algorithm from t to s with $\beta_t = 0$ finds it.*

Proof. Let P be a shortest feasible $s-t$ path in G. Now, we show that RCFP computes the correct solution (travel time and starting SoC) for P. We distinguish three cases:

- *P contains no charging stop:* The linking operation on (inverse) consumption profiles is associative [22]. Further, the order in which labels are added to $L_{uns}(v)$ does not affect the correctness of the algorithms. Therefore, a shortest feasible path is found regardless of search direction and the correctness of RCFP follows from that of CFP [19].

- *P contains a single charging stop:* Let u be the charging stop on P, which divides P into subpaths $[s \ldots u]$ and $[u \ldots t]$. As the RCFP search starts from t and reaches u, the departure SoC at u is set to $\mathrm{in}_{[u\ldots t]}$, the minimum SoC required to ensure feasibility

of P. Charging more at u only increases the charging time without any corresponding decrease in travel time, which in turn increases the total travel time along P, violating the assumption that P is the *shortest* feasible path. On subpath $[s \dots u]$, the RCFP search proceeds as in case (1).

- *P contains multiple charging stops:* Let u and u' be two consecutive charging stations on P, which divide P into subpaths $[s \dots u], [u \dots u']$ and $[u' \dots t]$. Lemma 2 in [19] shows that for CFP, the *optimal* departure time at u always corresponds to charging to either $\text{in}_{[u \dots u']}$ or to a breakpoint of cf_u. Similarly, after the RCFP search reaches u, the departure time at u' always corresponds to charging to either $\text{in}_{[u \dots u']}$, or to a breakpoint of $\text{cf}_{u'}^{-1}$, which is optimal.

Next, we show that the minimum time label in RCFP search is not dominated by other labels and reaches s the first. The first claim follows from the dominance criterion for RCFP, which is symmetric to that of CFP: A label ℓ_v is dominated if it results in a higher total travel time for every possible initial SoC at v. This implies that a dominated label can not result in a unique optimal solution, since replacing the sub-path to the target it represents with the sub-path of the label dominating it would result in a better or equal solution. Lastly, since labels are ordered by travel time at all $L_{uns}(v)$, the label with minimum total travel time reaches s first. □

Computing SCM with Reverse CFP

If the Reverse CFP algorithm does not terminate when the search reaches s and is instead allowed to continue to run until PQ is empty, we would have the set of all pareto-

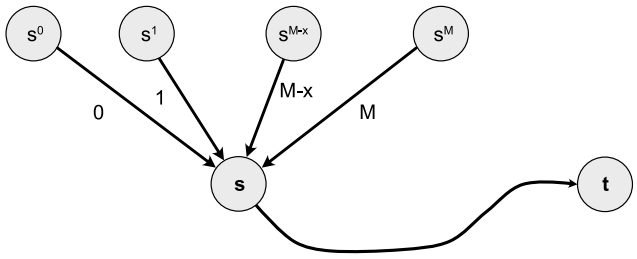


Figure 3.2: 'Virtual' vertices added to the graph.

optimal feasible paths from s to t at s. This set of pareto-optimal feasible paths forms the Starting Charge Map between vertices s and t.

Theorem 6. *If the RCFP algorithm is run from $t \in V$ with $\beta_t = 0$ until the priority queue is empty, the Pareto-set of labels at every $s \in V$ is equivalent to SCM_{st}.*

Proof. We prove Theorem 6 by showing that after running the RCFP from t, a starting SoC β_s, the label set at s contains a label that corresponds to $SCM_{st}(\beta_s)$. For this, we add temporary *virtual* vertices $s^{(M-x)}$ and an edge from $s^{(M-x)}$ to s with energy consumption x to the network, as depicted in Figure 3.2. From Lemma 5, we know that RCFP can compute a shortest feasible path P from $s^{(M-x)}$ to t. Note that by construction, P must contain s and $\text{in}_{[s\ldots t]} \leq x$, since $(M - x)$ energy is consumed on the edge from $s^{(M-x)}$ to s. Thus, a label ℓ must exist at s that represents the shortest feasible path from s to t and requires an initial SoC of at most $x.\ell$ corresponds to $SCM_{st}(x)$. Since the computation of RCFP in the network without $s^{(M-x)}$ is independent of the existence of $s^{(M-x)}$, the RCFP algorithm has to compute the label ℓ before the priority queue runs empty even if $s^{(M-x)}$ is not part of the network. □

3.5 Buffer Maps

Like Starting Charge Maps, a Buffer Map is a generalization of the Shortest Feasible Path Problem; albeit instead of unknown starting charge β_s, the lower bound of minimum allowed SoC along the path is raised from 0 to an arbitrary $\mathfrak{b} \in [0, M]$. Formally,

Definition 7. *A* buffer map $BM_{st} : [0, M] \rightarrow P$ *between a source* $s \in V$ *and target* $t \in V$ *is a function that maps a given* buffer SoC $\mathfrak{b} \in [0, M]$ *to the corresponding shortest feasible path* P *such that the EV maintains at least* $\mathfrak{b}$ *SoC at all points in* P*.*

Further, like SCMs, Buffer Maps can be used to show alternative routes to EV drivers who can decide upon the degree of acceptable stranding risk along the route. However, a key difference between the two abstractions and their usage is that while SCMs are used to get alternative routes depending on the *starting state* of the EV, alternative routes in buffer maps differ on the basis of *projected EV behaviour along the route*. In this way, alternative routes in BMs offer strong guarantees against stranding risk for EV drivers; not surprisingly, they are also more expensive to compute. Note that this problem would also qualify as what is referred to as "profile query" in the literature, since we ask for an optimal solution for arbitrary initial SoC [22, 27]. However, unlike [22, 27], we consider a multi-criteria variant of this problem and also allow intermediate charging stops.

For each distinct $\mathfrak{b}$, a run of the CFP algorithm can yield a shortest feasible path with the minimum SoC equal to $\mathfrak{b}$. A brute force approach to computing a Buffer Map is to run CFP several times, setting $\mathfrak{b}$ to each value in $[0, M]$. However, this approach is not feasible since the interval $[0, M]$ contains infinite values. In the next subsection, we present an exact, practical algorithm to compute a buffer map.

3.5.1 Iterative Charging Function Propagation

Each EV path consists of a sequence of legs. We define:

Definition 8. *Given an SoC* $\mathfrak{b} \in [0, M]$*, a* critical leg *of a shortest feasible path* P *is one on which the SoC drops to* $\mathfrak{b}$*.*

CFP computes the exact amount of charge that an EV charges at every station along a feasible route P in order to minimize total travel time. However, to ensure that the minimum SoC of the EV along P never drops below a given $\mathfrak{b} \in [0, M]$, the EV must charge extra on the charging stations adjacent to critical legs along P. The amount of extra energy to charge at such stations is exactly equal to that required to maintain at least $\mathfrak{b}$ SoC along the route, and is called the *buffer energy*.

Our approach to computing a Buffer Map BM_{st} for a given source $s \in V$ and target $t \in V$ works in *iterations*. Every iteration starts with choosing a value $\mathfrak{b}' \in [0, M]$. An augmented variant of CFP is run that returns a shortest feasible path P' such that the minimum SoC of the EV along P' is equal to $\mathfrak{b}'$. A collection of all such P' constitutes the set of paths in BM_{st}. Therefore, our approach has two components: first, an augmented variant of the CFP that respects the buffer SoC $\mathfrak{b}'$, and second, an algorithm that computes the increase in $\mathfrak{b}'$ on every iteration.

Augmenting CFP

The first iteration of our algorithm starts with $\mathfrak{b}' = 0$. The augmented CFP search starts from s with an SoC β_s and propagates towards t. Assume that the search requires charging at consecutive stations u' and u, and reaches $v \in V$. Let the breakpoints of $\mathrm{cf}_{u'}$

be $[B^1_{u'}B^2_{u'}\dots B^m_{u'}]$, where $B^i_{u'} = (\tau^i_{u'}, SoC^i_{u'}), 1 \le i \le m$ where SoC'_i is EV's resultant SoC after charging for time $\tau^i_{u'}$. Similarly, the breakpoints of cf_u are $[B^1_u B^2_u ... B^n_u]$.

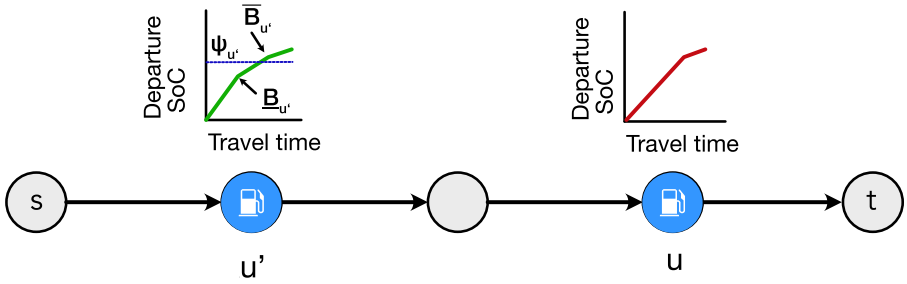


Figure 3.3: Augmented CFP setup.

Recall that CFP sets the amount of charge added to the EV at a station only after the search reaches the next charging station. Let the EV's SoC be $\psi_{u'}$ at departure after charging at station u'. Also, let $\underline{B}_{u'} = (\underline{\tau_{u'}}, \underline{SoC_{u'}})$ be the breakpoint of $\text{cf}_{u'}$ with SoC immediately lesser or equal to $\psi_{u'}$, and $\overline{B}_{u'} = (\overline{\tau_{u'}}, \overline{SoC_{u'}})$ be the next breakpoint after $\underline{B}_{u'}$. Therefore, $\underline{SoC}_{u'} \le \psi_{u'} < \overline{SoC}_{u'}$. Figure 3.3 shows an example of the $\underline{B}_{u'}$ and $\overline{B}_{u'}$ corresponding to a given $\psi_{u'}$. Similarly, given cf_u and ψ_u, $\underline{SoC}_u \le \psi_u \le \overline{SoC}_u$.

At $v \in V$, a label of the search is given by $\mathfrak{l} = \langle \tau_v, \beta_u, u, f_{[u...v]}, \rho_v, \delta_v \rangle$, where τ_t, β_u, u, and $f_{[u...v]}$ are analogous to regular CFP, ρ_v is the time required to add unit buffer energy to the EV on the current path, and δ_v is the maximum SoC up to which it can be charged without a loss in charging rate (due to concavity of charging functions).

Lemma 9. *Let P be a shortest feasible $s-t$ path with k charging stops on P and $\mathfrak{b}$ be the minimum allowed SoC along P. Assume that the EV arrives at i^{th} charging station with SoC α_i, charges for t_i time, and departs with SoC ψ_i. Further, let C be the charging stations at the beginning of critical legs in P. To increase the buffer energy along P by ϵ,*

increasing departure SoC ψ_i to $(\psi_i + \epsilon)$ on all stations in C is an optimal solution if:

1. *On charging stations in C, $f(\psi_i + \epsilon) - f(\psi_i) = \epsilon$, i.e. charging ϵ more increases the residual SoC at t by ϵ.*

2. *On all non-critical legs, the minimum allowed SoC is at least $\mathfrak{b} + \epsilon$.*

3. *For all charging stations in C, the charging function is differentiable and does not have breakpoints with SoCs in range $[\psi_i, (\psi_i + \epsilon)]$.*

4. *For all charging stations at the end of a critical leg, the charging function is differentiable and does not have a breakpoint in SoC range $[\alpha_i, (\alpha_i + \epsilon)]$.*

Proof. First, note that increasing ψ_i at all charging stations in C by ϵ is sufficient to increase the total buffer by ϵ —this follows immediately from conditions (1) and (2).

Let a *solution* S be the set of charging stops and charging times along path P, resulting from an Augmented CFP run between vertices s to t. We will show that no other solution can result in a lower total travel time along path P without changing at least one edge in P. Assume for contradiction, a solution S' has a lower total travel time than S along same path P. In order to increase the buffer energy for S by ϵ, ψ_i for each charging station in C must be increased by at least $(\psi_i + \epsilon)$. This can only be achieved by charging additional energy at a station on P.

Let j be the charging stop closest to t at which charging time differs between S and S'. We claim that there must be a critical leg after departing from j and that its departure SoC is $(\psi_j + \epsilon)$—if this were not the case, we could decrease the departure SoC at j to $(\psi_j + \epsilon)$, which would be sufficient to increase buffer energy by ϵ, giving us a faster solution

and contradicting the assumption that S is optimal. This implies that we can decrease the departure SoC at j to $(\psi_j + \epsilon)$, which is sufficient to increase buffer energy by ϵ, which gives us a faster solution contradicting the assumption that S is optimal. Since the departure SoC on j is equal to $(\psi_j + \epsilon)$, the arrival SoC at j must be greater. In other words, we charge more at some other stop i so we can charge less at j. But then, we can create a faster solution for buffer energy $\mathfrak{b}$ as follows: there exists a $\delta > 0$ such that we can charge δ more at i and charge δ less at j (since the charging function is concave and differentiable around ψ_j, the charging rate remains the same as for $(\psi_j + \epsilon)$). This contradicts the fact the solution S for buffer SoC $\mathfrak{b}$ was optimal. □

The CFP search starts from s with $\rho_s = 0$ and $\delta_s = M$. Assume that the search reaches charging station u after charging at a prior station u'. Let $\mathfrak{l}_u = \langle \tau_v, \beta_u, u, f_{[u\ldots v]}, \rho_v, \delta_v \rangle$ be the current label extracted from priority queue. If $\psi_{u'} = \mathfrak{b}'$, i.e. the leg $[u' \ldots u]$ is a critical leg, we set $\delta_u = \min(\delta_{u'}, SoC_{B_2} - SoC_{B_1}, \overline{SoC}_u - \mathrm{cf}_u(f_{[u'\ldots u]}(\psi_{u'})))$, where SoC_{B_1} and SoC_{B_2} are the SoC of the first and second breakpoints of the SoC function of $\mathfrak{l}_u$. We also set $\rho_u = \rho_{u'} + \frac{(\overline{\tau}_{u'} - \underline{\tau}_{u'})}{(\overline{SoC}_{u'} - \underline{SoC}_{u'})} - \frac{(\overline{\tau}_u - \underline{\tau}_u)}{(\overline{SoC}_u - \underline{SoC}_u)}$. If $[u' \ldots u]$ is not a critical leg, the $\delta_u = \min(\delta_{u'}, \mathrm{in}_{[u'\ldots u]} - \mathfrak{b}')$, and $\rho_u = \rho_{u'}$.

A label $\mathfrak{l}_u$ dominates $\mathfrak{l}'_u$ if the SoC function of $\mathfrak{l}_u$ dominates the SoC function of $\mathfrak{l}_{u'}$, and $\rho_u \leq \rho_{u'}$. In other words, a label $\mathfrak{l}$ dominates $\mathfrak{l}'$ if it represents a faster path to which the buffer energy can be added at a faster rate.

Computing $\mathfrak{b}'$ for the next iteration

On an iteration, we let the augmented CFP run and collect the complete set of non-dominated labels at target t. Let the set of labels collected at t be $\mathfrak{L}$. The Augmented CFP search guarantees that every $\mathfrak{l}_i \in \mathfrak{L}$ represents a feasible path where the SoC along the path does not drop below $\mathfrak{b}$. Let the label $\mathfrak{l}_{min}$ have the minimum total travel time of all $\mathfrak{l} \in \mathfrak{L}$. Next, we need to determine the maximum buffer energy that can be added at stations adjacent to critical legs of the shortest feasible path P found in the current iteration, while ensuring that no other feasible path becomes a better (faster) choice than P. We can solve this problem geometrically on an X-Y plane, where X and Y axis represent the buffer SoC and the total travel time of the EV respectively.

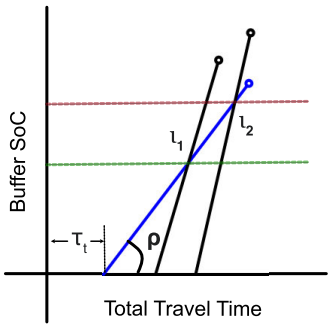


Figure 3.4: Each line on the X-Y plane represents a label $\mathfrak{l}_i \in \mathfrak{L}$ for iteration N. Highlighted blue *label line segment* represents the minimum time label $\mathfrak{l}_{min}$. The slope of blue label line segment is $\rho \in \mathfrak{l}_{min}$, and the X-intercept is equal to travel time $\tau_t \in \mathfrak{l}_{min}$. It intersects with two other label line segments at ι_1 and ι_2. Similarly, let intersection points be $\{\iota_1, \iota_2, ..., \iota_n\}$ if $\mathfrak{L}$ contains more labels. b' for the next iteration is equal to the minimum buffer SoC in $\{\iota_1, \iota_2, ..., \iota_n\}$ (SoC of shown green line).

For a label $\mathfrak{l}_t$, we draw a *label line segment* with slope $\rho_t \in \mathfrak{l}_t$, and the X-intercept equal to the total travel time τ_t of $\mathfrak{l}$. Further, the maximum ordinate of the line segment is

given by $\delta_t \in \mathfrak{l}_t$. Figure 3.4 shows an example where $\mathfrak{L}$ contains three labels. The next step is to find the *globally minimum buffer SoC*, δ_{min} to which the EV can be charged the fastest among all labels in $\mathfrak{L}$. To find such a value, we start with the label line segment for $\mathfrak{l}_{min}$, and find its intersections with all other label line segments on the plane. Let the set of such intersections be $\{\iota_1, \iota_2, ..., \iota_n\}$. Since Figure 3.4 has only three label line segments, it shows two intersection points ι_1 and ι_2. Thus, δ_{min} is given by the buffer SoC of the intersection point that lowest on the Y-axis in the plane. For the next iteration, we set $\mathfrak{b}' = \delta_{min}$ and add the feasible path represented by $\mathfrak{l}_t$ to the buffer map BM.

Lemma 10. *The global delta selection algorithm is correct, i.e. no feasible path has a lower total travel time and can add buffer energy faster than the chosen route given by the algorithm.*

Proof. We prove geometrically. Since all charging functions are convex with a positive slope, the slopes of all label line segments in the X-Y plane are positive. Further, since $\mathfrak{l}_{min}$ has the smallest X-intercept, in buffer SoC interval $[0, \text{SoC of } \iota_1]$, no other label in $\mathfrak{L}$ can charge the EV to a higher buffer SoC in lesser time. □

As we increase $\mathfrak{b}'$ on each iteration, the augmented CFP search becomes more selective and the number of feasible paths from s to t decreases, since only on fewer paths would an EV be able to maintain a higher minimum SoC. The iterations terminate when $\mathfrak{b}'$ becomes high enough so the CFP search does not return any feasible paths.

Theorem 11. *The Iterative CFP algorithm terminates and computes BM_{st} correctly.*

Proof. We have already argued that we compute ρ and δ correctly for labels propagated by the Augmented CFP search, and that for label $\mathfrak{l}$ it gives us the minimum additional required

charging time in order to increase the buffer energy by any value in $[0, \delta]$ on the feasible path represented by $\mathfrak{l}$. We now show that the solutions added to the buffer map are indeed optimal and there is no remaining path with a shorter time for some value of buffer energy. Assume for contradiction, that we add a label $\mathfrak{l}$ to the buffer map, for which there exists a label $\mathfrak{l}'$ that offers a faster solution for some buffer energy. Observe that this implies that it is not a part of the Pareto set at the target, since the global delta computation finds the best label in that set by lemma 10. We can now distinguish two cases:

1. $\mathfrak{l}'$ *represents a feasible path with at least one critical leg:* Since $\mathfrak{l}'$ can not have a faster (minimum) traversal time than $\mathfrak{l}$ by construction (the algorithm selected $\mathfrak{l}$ and added it to the buffer map because it is the label with minimum travel time), it can only become the better solution after adding additional charge so it yields shorter total travel time for higher buffer energy. In other words, $\mathfrak{l}'$ offers a better charging rate and therefore is not dominated by $\mathfrak{l}$, which implies that it (or another dominating label) must be a member of the Pareto set. This must result in a lower intersection point on the Y-axis than δ during the global delta computation, which contradicts our assumption.

2. $\mathfrak{l}'$ *represents a feasible path with no critical leg:* This implies it has no charging stop (if there was a label with a charging stop but no critical leg, we could always charge less to obtain a faster solution). This means it cannot be dominated by $\mathfrak{l}$ because it has $\rho = 0$, and therefore it or another single-leg path must be a part of the Pareto set, which implies that it is taken into account when computing the global value of δ, again leading to a contradiction.

□

Several factors can affect the total number of iterations required to compute BM: the distance between s and t, the total number of charging stations required to reach from s to t, which in turn depends on the parameters of the EV under consideration. The number of iterations further depends on the number of breakpoints in charging functions along the feasible paths from s to t. However, in practice, the number of iterations remains small for the following reasons: First, $\text{cf}_u, u \in S$ are usually simple, linear functions up to 80% charge and only have a small number of breakpoints in the $80 - 100\%$ range. Next, most EV trips tend to not have a large number of charging stops along the way, and as EV ranges increase, this number would further decrease.

3.6 Experiments

We implemented our algorithms in C++ using Apple clang version 10.0.1 with $-O3$ optimizations. All experiments were run on macOS 10.14.6 using a Mac Pro 6,1 with a quad-core Intel Xeon E5 (3.7 GHz base clock). The processor has 256 KB of per-core L2 and 10 MB of shared L3 cache. The machine has 64 GBs of DDR3-ECC memory clocked at 1866 MHz.

3.6.1 Preparing a realistic EV Routing instance

We extract the road networks of Oregon and California from OpenStreetMap (OSM)[1] and label each edge with travel time equal to geographic distance divided by the

[1] `https://openstreetmap.org/`

Table 3.1: Our road network is taken from OpenStreetMap, public charging stations data from the Alternative Fuel Data Center [5], elevations from NASADEM [106] and an energy consumption model from a Nissan Leaf 2013 [54].

Dataset	**Vertices**	**Edges**	**Charging stations**
Oregon (contracted)	502327	710107	323
California (contracted)	2547618	3741891	1406

maximum allowed speed for the road segment type. We contract all vertices with degrees ≤ 2 for our experiments, keeping only the largest connected component of the network. Table 3.1 shows the size of road networks after contraction.

Next, we add the elevation to each vertex of the network, taken by sampling the NASADEM elevation dataset at $30m$ resolution [106]. The elevation is required to compute the energy consumption on every edge of the network, which we derive from a microscopic EV energy consumption model for a Nissan Leaf 2013 [54].

Lastly, we extract the locations of public EV charging stations in Oregon and California from the Alternative Fuels Data Center [5]. For each charging station in the dataset, we mark the vertex geographically closest to it as the charging station. We assign each charging station vertex one of three charging functions: i) a *slow* linear function that charges the EV to full battery in 120 minutes; ii) a *fast* charging function that charges the EV to 80% in 30 minutes and to full capacity in 60 minutes, and iii) a *fastest* charging function that charges to 80% capacity in 20 minutes, and to full in 40 minutes. We arbitrarily assign 60% of all charging stations the *slow* charging function, another 30% stations the *fast*, and the remaining 10% the *fastest* charging functions.

To allow for tests with reasonable running times, we make it easier for a label to dominate another in the (Reverse) CFP search. We do this by adding a constant *slack*

energy consumption ϵ to the dominance criterion in all three algorithms. Given labels ℓ_1 and ℓ_2, ℓ_1 dominates ℓ_2 iff all breakpoints of ℓ_1's SoC function have a higher energy than breakpoints of ℓ_2's SoC function after decreasing each breakpoint by ϵ energy. We set ϵ to 1% of the total battery capacity of the EV. Similar modifications to the dominance criteria have been proposed in earlier work, e.g. see [14, 25].

3.6.2 Reverse Shortest Feasible Path Queries & Starting Charge Maps

Table 3.2: Average performance of 1000 queries running RCFP vs. variants of standard CFP. The EV is always assumed to start with 100% SoC at source. CFP with stopping criterion terminates after finding only one feasible route, and is therefore much faster than regular CFP which returns all feasible routes. RCFP can be seen to perform at par with CFP without stopping criterion. Time shown in seconds, also shown—no. of labels extracted from priority queue, alternative routes to t, and the no. of times search reached target. Targets found differ between RCFP and CFP because of the difference in dominance criteria.

		16 kWh				**32 kWh**			
	Alg.	Time	kLabels	Routes	Targets	Time	kLabels	Routes	Targets
California Oregon	CFP (Stp)	1.767	933	0.709	709	1.510	873	0.895	895
	CFP	3.853	1758	4.962	709	3.629	1973	5.634	895
	RCFP	4.861	2477	7.703	710	3.502	2370	7.176	895
	CFP (Stp)	34.847	10141	0.722	722	21.805	8645	1.0	1000
	CFP	70.076	19684	8.88	722	61.171	21596	9.837	1000
	RCFP	66.096	22571	11.467	724	46.617	22191	11.645	1000

Table 3.3: Continuation of RCFP results for 64 and 128 kWh.

		64 kWh				**128 kWh**			
		Time	kLabels	Routes	Targets	Time	kLabels	Routes	Targets
California Oregon	CFP (Stp)	0.877	730	1.0	1000	0.678	621	1.0	1000
	CFP	2.648	1859	5.205	1000	2.521	1751	5.04	1000
	RCFP	2.641	2071	6.599	1000	2.241	1877	5.76	1000
	CFP (Stp)	13.919	6631	1.0	1000	7.221	5006	1.0	1000
	CFP	47.197	18273	8.429	1000	25.182	13752	6.304	1000
	RCFP	36.568	19079	9.897	1000	16.255	12220	6.563	1000

Tables 3.2 and 3.3 shows the results of running 1000 SFP and RSFP queries with several standard EV battery capacities (16, 32, 64, and 128 kWh) between random vertices in the road networks of Oregon and California. The table compares the performance of three algorithms–forward CFP with *stopping criterion*, which makes the search terminates as soon as it reaches t; *full* forward CFP that runs till all pareto-optimal feasible paths from s to t are found; and the Reverse CFP algorithm as presented in Section 3.4.

We find that the CFP with stopping criterion performs at least a factor of two faster than full CFP that computes the pareto-optimal set of feasible paths. This is hardly surprising as the full CFP offers a richer set of routes which planners can use, in lieu of more computational overhead. However, if faster queries are desirable at the cost of alternative routes, the same technique can be applied to the reverse CFP algorithm with little effort. Target pruning [21] is another closely related technique that can achieve the same goal.

We observe that for both networks, SFP and RSFP query times generally decrease with increase in range of the EV, with a notable exception of capacity increase from 16 to 32 kWh, in which case the reverse search query times increase for the Oregon network and full CFP query times for the California network. This can be explained as follows: As the battery capacity increases, the (R)CFP search is able to reach vertices farther away. However, with increase in range, the slack energy ϵ also increases, making it easier for a label to dominate another, so fewer labels are settled in the search. The net effect of the two opposing factors, in this case, is that the total query time increases.

Table 3.4: Average performance of Iterative CFP to answer 1000 Buffer Map queries (with 50 and 100% starting SoC) between random vertices on the Oregon road network.

	Range	Time (s)	*k*Labels	Iterations	Avg. $\lvert BM \rvert$	Targets
50%	16 kWh	67.405	30754	7.03	6.103	640
	32 kWh	99.310	45685	9.987	9.061	878
	64 kWh	32.571	25441	9.37	8.538	1000
	128 kWh	17.440	15316	7.013	6.359	1000
100%	16 kWh	198.395	57545	10.573	9.57	709
	32 kWh	85.484	47106	14.285	13.29	895
	64 kWh	53.706	37930	14.225	14.22	1000
	128 kWh	14.240	16183	10.611	9.635	1000

3.6.3 Iterative CFP and Buffer Maps

Table 3.4 shows the results of 1000 Iterative CFP queries between random vertices in the Oregon network. We do not report the running times for California, since they were found to be impractical with some queries running for more than 3 hours.

The total running time of the Iterative CFP algorithm has two components: The cost of Augmented CFP runs and the cost of computing the minimum global δ energy in each round. The cost of global delta computation is negligible in practice, since the number of Augmented CFP labels reaching the target vertex is often low. In Table 3.4, note that an Augmented CFP run takes longer than full CFP. This is caused due to inclusion of an additional parameter (charging rate) in the dominance criteria of the Augmented CFP.

The Iterative CFP is slower than the other algorithms discussed. This is expected as the algorithm involves running several iterations of an exponential-time shortest path computation. Our networks do not use standard speedup techniques like Contraction Hierarchies (CHs) [65], their multicriteria variant [63], or CRP [45, 46], though. Applying any of these techniques can significantly reduce query times by reducing the number of vertices

explored to find shortest paths. A combination of speedup techniques such as CHs and A* search could be further applied for even greater speedups [15] at the cost of additional complexity.

Chapter 4

Stochastic Route Planning for Electric Vehicles

4.1 Introduction

Routing methods for Electric Vehicles (EVs) cannot just minimize travel time, but must also address driver *range anxiety*. EVs have limited battery capacity, charging times are long, and the charging infrastructure remains relatively sparse, so a major concern is *stranding*, which occurs when the battery's State of Charge (SoC) reaches zero en route. A route for an EV is hence considered *feasible* only if the SoC along the route never reaches zero.

Merely trying to minimize travel time greatly increases the risk of stranding, since energy consumption is typically quadratic in vehicle speed. Standard formulations such as [19, 50, 99] model EV routing as a generalization of the NP-hard Constrained Shortest Path problem [79, 164], and seek to minimize travel time while maintaining a non-zero SoC along

the route. Some recent work [17, 112] even tries to exercise direct control over travel time and route feasibility, by pre-determining and assigning optimal EV travel speeds for each road segment.

Most existing problem formulations also assume that travel times and energy consumption values on road network edges are deterministic. In practice, both travel time and energy consumption are stochastic, and difficult to estimate reliably [3, 53, 131]. In such a context, even routing algorithms offering strong feasibility guarantees are of limited value. Approaches that pre-determine and assign vehicle speeds for each edge are not practical, since speed is not always a variable under driver control, but rather a result of prevalent traffic conditions.

Consequently, travel times and route feasibility may only be defined *probabilistically*. Stochastic routing algorithms [37, 58, 116, 115, 114, 120, 121, 122, 84, 167], model travel times along network edges as random variables with given probability distributions, and allow richer query semantics, such as finding paths to maximize the probability of arrival before a deadline [52], or finding the latest departure time and path to guarantee a certain probability of arrival before a deadline [111]. Despite recent improvements [132], stochastic routing is typically several orders of magnitude slower than deterministic routing, since obtaining travel time distributions along a path requires very expensive convolutions of its edge distributions.

Limited work exists on stochastic route planning for EVs. [39] assumes lognormal travel-time and Gaussian energy-consumption distributions, and uses bicriteria search to find the Pareto-optimal routes optimizing energy consumption and travel time reliability.

[77] allows arbitrary distributions of travel times on edges and charging stations, and uses multicriteria search to minimize the cost of charging and travel time, subject to a minimum reliability threshold, on small synthetic graphs with randomly generated edge weights and charging station placements. [144] allow correlated travel time distributions between edges, and use bicriteria search on travel times and energy consumptions. They assume deterministic energy consumptions, and run experiments on a network of only a few hundred vertices.

4.1.1 Our Contributions

We study EV routing when both travel times and energy consumptions are stochastic. The travel time on each edge $e \in E$ of a road network $G = \langle V, E \rangle$ is always a random variable $\boldsymbol{T}_e$ with a known distribution (estimated from data, say). The energy consumption ε_e along e is a function of EV speed and distance. We introduce two probabilistic definitions of route feasibility: We say that a route is $\mathbb{E}$-feasible if the SoC of the EV is always maintained above zero *in expectation*, and p-feasible if the probability of route feasibility is *at least* p. We show how to enhance stochastic routing queries for travel times with these feasibility criteria to find non-dominated feasible routes and probabilistic budget feasible routes. Our work addresses the four types of stochastic routing queries in the cells of the following table:

Table 4.1: Stochastic EV routing queries considered in this chapter.

	$\mathbb{E}$-Feasibility	p-Feasibility
Non-Dominated Routes	✓	✓
Probabilistic Budget Routes	✓	✓

We address these queries by generalizing the Charging Function Propagation algorithm of [18, 19] to accommodate stochastic edge weights. We evaluate our methods experimentally using a realistic road network instance with travel time distributions derived from traffic speeds observed over four and a half months in the Los Angeles area, and real-world elevations and charging station locations. Further, we apply an uncertain variant of Contraction Hierarchies [64] to speed up our queries and present results. Our results indicate that in general, $\mathbb{E}$-feasible routing queries can be computed much faster than p-feasible queries, and produce similar routes for longer routes with higher time budgets.

4.2 Related Work

EV routing has been typically modeled as energy-aware routing, with objective functions ranging from minimizing the total energy consumption [50, 139], to minimizing travel time while maintaining route *feasibility* [8, 19, 112, 152], to multicriteria search on both travel time and energy consumption [68]. In contrast, most prior work on routing Internal Combustion-based vehicles merely minimizes the total travel time [13, 149].

More attention is now being paid to real-world issues. Examples include allowing battery-swapping stations [154], partial recharges at stations [19, 99, 18, 155] and maintaining battery buffer to relieve range anxiety [130, 76, 55]. Many challenges remain, however. The energy consumption models are imperfect, and factors such as battery wear, driver aggressiveness [100], or traffic conditions are hard to model, but can have a significant impact. Data also suggests that drivers may prefer familiar paths to shortest paths [178, 96].

4.2.1 Stochastic Route Planning

Stochastic route planning goes back to [58], which attempted an exact solution for the shortest path problem in stochastic graphs, using Monte Carlo simulations to derive path weights. It is now known that driver behavior changes if travel time is stochastic [57, 148], so problem variants have been explored. Existing work can be categorized in three ways: by objective function, the forms assumed for edge probability distributions, and by targeted outcome. For conciseness, we discuss our categorization here, and show references in Figure 4.1.

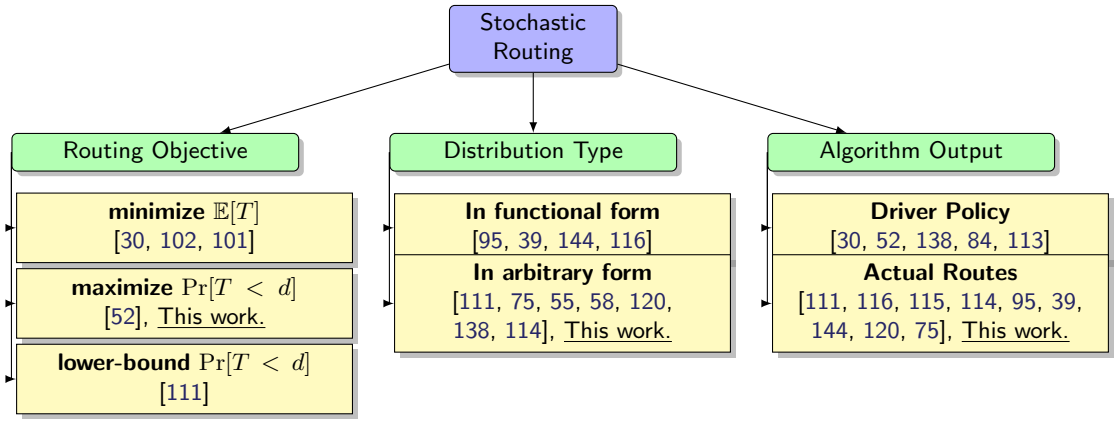


Figure 4.1: Stochastic route planning, classified by routing objective, edge distribution, and result. Our work finds energy-feasible routes that maximize probability of arrival before deadline.

By routing objective: Routing objectives can be quite varied, such as minimizing expected time [30, 102, 101], maximizing the on-time-arrival probability [52], maintaining on-time-arrival probability above a given threshold [111]. Some works [144, 39, 77] apply stochastic routing algorithms to EVs, while others [2] route multiple EVs collaboratively, on-line.

By distribution: The edge distributions assumed can have a functional form, or be arbitrary without a closed form. This choice also affects the edge weight representations used. For functional forms, storing the distributional parameters suffices, but arbitrary distributions require more space-intensive representations such as histograms. Further, with functional forms, a small number of observations can suffice to capture real-world behaviour, but histograms require much more data. Edge weight representations have been shown to significantly affect the runtime performance of stochastic shortest path queries [122, 132].

Output: Adaptive methods [119, 111] output policies for drivers to make routing decisions on-line, as they reach vertices or edges during the drive. In contrast, a-priori approaches produce routes before travel begins. [111] showed that adaptive approaches can produce strictly better solutions than the a-priori approaches, but are much more computationally expensive. Recently, performance improvements to policy-based approaches, such as the Stochastic On-Time Arrival problem have also been proposed [138, 113, 84].

4.3 Problem Setup

A road network is a directed graph $G = \langle V, E \rangle$ where V is the set of vertices and $E : V \times V$ is the set of edges. An s-t path $P = [s = v_1, v_2 \cdots, v_n = t]$ is a sequence of adjacent vertices in the road network G. A set $C \subseteq V$ is marked as *charging stations*.

Definition 12 (State of Charge)**.** *The State of Charge (SoC) of an EV is the charge status of the EV's battery, lying between 0 and the battery capacity M. We denote the SoC on arrival at a vertex v by ${}_v\beta$ and the SoC at departure from v by β_v. We have $\beta_v \geq {}_v\beta$ if the EV charges its batteries at node v, and $\beta_v = {}_v\beta$ otherwise.*

Each $c \in C$ has a monotonically increasing, piecewise-linear charging function Φ_c such that $\Phi_c({}_c\beta, t_c) \mapsto \beta_c$ where t_c is charging time. We require ${}_c\beta \geq 0$, and $\beta_c \leq M$ [130].

Definition 13 (Leg and Prefix)**.** *A subpath* $L = [c_1, \ldots, v, \ldots, c_2]$ *is a* leg *of path* P *iff* c_1, c_2 *are successive charging stations along* P*. Each* $\lambda_v = [c_1, \ldots, v]$*,* $v \neq c_2$ *is a* prefix *of* L*.*

4.3.1 Travel Times and Energy Depletion

The travel time along each edge e is a random variable $\boldsymbol{T}_e$ with a known probability distribution. For problem tractability, we assume that the EV travels on e at a uniform speed drawn from the distribution $\boldsymbol{T}_e$. This is reasonable, since variable travel time on an edge can be easily modeled by splitting an edge into several smaller edges.

Let $e_1, e_2, \ldots, e_{n-1}$ be the edges along path P, and let e_k have travel time distribution T_k. The aggregate travel time distribution for the path P is $\boldsymbol{T}_P = \boldsymbol{T}_1 * \boldsymbol{T}_2 * \cdots * \boldsymbol{T}_{n-1}$, where $*$ denotes linear convolution. Let $\boldsymbol{T}_\emptyset$ be the Dirac "delta" distribution defined so that $\boldsymbol{T}_\emptyset(0) = 1$ and $\boldsymbol{T}_\emptyset(x) = 0$ at $x \neq 0$. Now, $\boldsymbol{T}_\emptyset$ functions as a convolution identity, so $\boldsymbol{T}_\emptyset * \boldsymbol{T}_P = \boldsymbol{T}_P$.

We assign to each edge e a function $\varepsilon_e : \mathbb{R}^+ \to \mathbb{R}$, which maps a travel time to the battery energy depleted by travel along e. The total energy depletion is the sum of the work done along e by the EV against air resistance, rolling resistance, and against gravity. The wind resistance grows quadratically with speed. If t is the travel time along edge e,

these three terms cause $\varepsilon_e(t)$ to assume the form

$$\varepsilon_e(t) = \frac{a_e}{t^2 - b_e} + \frac{c_e}{t} + d_e. \tag{4.1}$$

where a_e, b_e, c_e, d_e are fixed coefficients for each edge e. We can derive the edge *energy depletion distribution* $\boldsymbol{D}_e$ from the travel time distribution $\boldsymbol{T}_e$ using Equation 4.1, thereby associating probabilities with energy depletions. A path may have negative energy depletion; EVs have regenerative brakes, and can accumulate charge, say, when going down a slope.

We can also aggregate energy depletion distributions using convolutions. If $e_1, e_2, \ldots, e_{n-1}$ are the edges along a path P, and edge e_i has the depletion distribution $\boldsymbol{D}_i$, the aggregate energy depletion distribution for P is $\boldsymbol{D}_P = \boldsymbol{D}_1 * \boldsymbol{D}_2 * \cdots * \boldsymbol{D}_{n-1}$. By analogy with $\boldsymbol{T}_\emptyset$, we define $\boldsymbol{D}_\emptyset$ to be the Dirac "delta" function corresponding to energy depletion, so that $\boldsymbol{D}_\emptyset * \boldsymbol{D}_P = \boldsymbol{D}_P$. Sometimes, as with expected-feasibility queries, it suffices to add expectations directly, since $\mathbb{E}[\boldsymbol{D}_1 * \boldsymbol{D}_2] = \mathbb{E}[\boldsymbol{D}_1] + \mathbb{E}[\boldsymbol{D}_2]$.

4.3.2 $\mathbb{E}$-Feasible Routing

In this class of queries, we assume that the travel times are stochastic, but define feasibility in terms of the expectations for the energy depletion distributions. Say that an EV starts from vertex s with State of Charge (SoC) $\beta_s \in [0, M]$ and wishes to travel to vertex t along the s-t path P. Let leg $L = [c, \ldots, c']$ of P lie between charging stations c and c' along P.

Definition 14 ($\mathbb{E}$-Feasible Path)**.** *Leg L is* expected-feasible *(or* $\mathbb{E}$-feasible*) iff $\mathbb{E}[\boldsymbol{D}_\lambda] \leq \beta_c$, where $\boldsymbol{D}_\lambda$ is the depletion distribution for all prefixes λ of L, and β_c is the EV's SoC when*

it departs c. A path $P = [L_1, L_2, \ldots, L_n]$ is $\mathbb{E}$-feasible iff each of its legs L_i is $\mathbb{E}$-feasible.

We consider two $\mathbb{E}$-feasible queries:

Query 1 (Non-Dominated $\mathbb{E}$-feasible Paths)**.** *Find the set of $\mathbb{E}$-feasible s-t paths such that their travel time distributions are not dominated by any other path.*

Query 2 (Probabilistic Budget $\mathbb{E}$-feasible Path)**.** *Find an $\mathbb{E}$-feasible s-t path that maximizes the probability of reaching t before a given deadline d.*

4.3.3 p-Feasible Routing

Definition 15 (p-Feasible Path)**.** *A leg L of a path is* p-feasible *if the probability of traversing L without being stranded is at least p. A route P with legs $L_1, L_2, \ldots, L_n$ is p-feasible iff each leg L_i is p-feasible. We call p the non-stranding probability.*

Analogous to the $\mathbb{E}$-Feasible case, now, we consider the following two p-feasible queries, that we seek to answer.

Query 3 (Non-Dominated p-Feasible Paths)**.** *Find the set of s-t paths whose travel time distributions are not dominated by any other path, and which ensure that probability not being stranded is at least p.*

Query 4 (Probabilistic Budget p-Feasible Paths)**.** *Find an s-t path which maximizes the probability of reaching t before a given deadline d, while keeping the probability of not being stranded is at least p.*

Sym	Meaning	Sym	Meaning
$\boldsymbol{T}_P$	Travel time distribution on path P	$\boldsymbol{T}_\emptyset$	Convolution identity for $\boldsymbol{T}$
$\boldsymbol{D}_P$	Energy depletion distribution on path P	$\boldsymbol{D}_\emptyset$	Convolution identity for $\boldsymbol{D}$
δ_λ	Depletion function for leg prefix λ	$\delta_\emptyset$	Depletion function for null path
${}_u\beta$	SoC on arrival at vertex u	β_u	SoC at departure from vertex u
Φ_c	Charging function at charging station c	ε_e	Energy depletion function on edge e

Table 4.2: Symbols used in this chapter.

4.4 Charging Function Propagation for $\mathbb{E}$-Feasible Routing

The CFP algorithm of [19] uses only deterministic edge weights, but we show how to extend it to answer expected-feasible stochastic shortest path queries. As in [19], we ensure that the SoC on departing a charging station suffices to complete the ensuing leg. Our Dijkstra's search labels maintain the set of all possible tradeoffs between charging time and the resulting SoC.

However, complexities arise since we use stochastic travel times. The deterministic case can simply use a min-priority queue ordered by travel times, but distributions can be ordered in different ways. For simplicity, we will use *usual stochastic ordering* [142] to order the travel time distributions in the priority queue, under which two random variables X and Y obey $X \preceq Y$ iff $\Pr[X > x] \leq \Pr[Y > x], \forall x \in (-\infty, \infty)$. Other stochastic orderings, such as the hazard rate or likelihood ratio ordering, may result in interesting tradeoffs for the EV, but are beyond the scope of this paper. Also, deterministic travel times can be simply added along a path, but travel time distributions must be convolved to aggregate travel time distributions.

For expected-feasible routes, we will use stochastic travel times, but expected values for energy depletion. That is, let $e_1, e_2, \ldots, e_n$ be the edges comprising a path

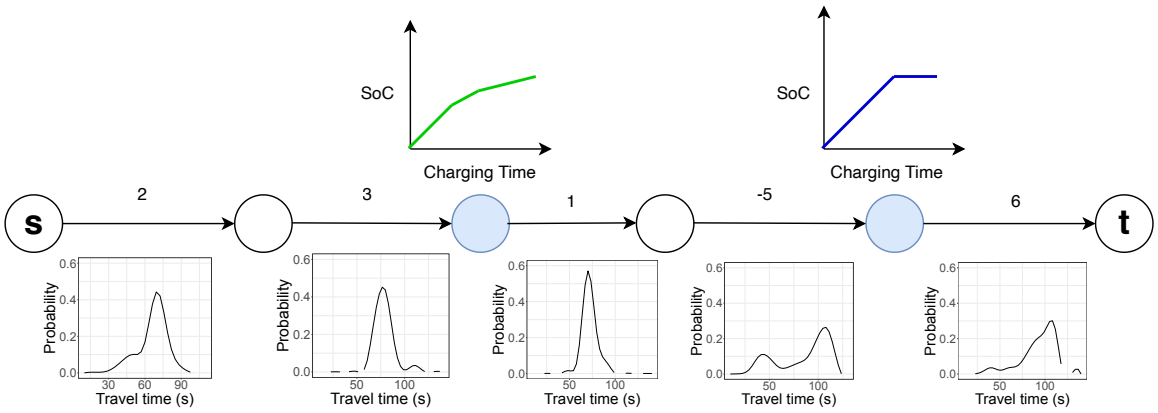


Figure 4.2: $\mathbb{E}$-feasible queries. Edges have two weights: a travel time distribution (below), and an expected energy depletion (above). Shaded nodes are charging stations, with piecewise-linear charging functions. The CFP search propagates travel time distributions using convolutions.

P, and let edge e_i have travel time and energy depletion distributions $\boldsymbol{T}_i$ and $\boldsymbol{D}_i$. For expected-feasible routing, the aggregate travel time distribution $\boldsymbol{T}_P = \boldsymbol{T}_1 * \boldsymbol{T}_2 * \cdots * \boldsymbol{T}_n$, and the aggregate energy depletion value is $\mathbb{E}[\boldsymbol{D}_P] = \mathbb{E}[\boldsymbol{D}_1] + \mathbb{E}[\boldsymbol{D}_2] + \ldots + \mathbb{E}[\boldsymbol{D}_n]$.

4.4.1 The Depletion Function Along Route Legs

Even if the energy depletion over leg $L = [c_1, \ldots, v, \ldots, c_2]$ is deterministic with value E_L, departing c_1 with an SoC of $\beta_{c_1} = E_L$ may not suffice to complete L. For instance, L may go up a hill, climbing which requires more energy than β_{c_1}. Similarly, ${}_{c_2}\beta$, the arrival SoC at c_2, may not equal $\beta_{c_1} + E_L$ when $E_L < 0$, since the SoC can never exceed M.

Consider a prefix $\lambda = [c, \ldots, v]$ of some leg that starts with charging station c. Let S_λ be the minimum starting SoC required to traverse λ, E_λ be the maximum ending SoC possible at v, and let $\mathrm{C}_\lambda = \mathbb{E}[\boldsymbol{D}_\lambda]$. The *depletion function* δ_λ (similar to *SoC profiles* in [18, 19]) for prefix λ maps the SoC at the start of λ to the SoC at the end of λ, and is

defined as

$$\delta_\lambda(\beta_c) = {}_v\beta = \begin{cases} -\infty, & \text{if } \beta_c < \mathrm{S}_\lambda, \\ \mathrm{E}_\lambda, & \text{if } \beta_c - \mathrm{C}_\lambda > \mathrm{E}_\lambda, \\ \beta_c - \mathrm{C}_\lambda, & \text{otherwise.} \end{cases} \tag{4.2}$$

The depletion function for a null path comprising a single vertex s is the *identity* depletion function $\delta_\emptyset : \beta_s. \mapsto \beta_s$, Let $P_1 = [v_i, v_{i+1}, \ldots, v_j]$ and $P_2 = [v_{j+1}, v_{j+2}, \ldots, v_k]$, be contiguous segments, and $P = P_1P_2 = [v_i, \ldots, v_k]$ be their concatenation. In this case, we have $\mathrm{S}_P = \max\{\mathrm{S}_{P_1}, \mathrm{C}_{P_1} + \mathrm{S}_{P_2}\}$, $\mathrm{E}_P = \min\{\mathrm{E}_{P_2}, \mathrm{E}_{P_1} - \mathrm{C}_{P_2}\}$ and $\mathrm{C}_P = \mathrm{C}_{P_1} + \mathrm{C}_{P_2}$.

4.4.2 Dijkstra Search for $\mathbb{E}$-feasible Routes

We find expected-feasible paths via Dijkstra search using two types of priority queues: the global queue Q_G holds the travel time distributions from s to all other vertices in the road network G, and per-vertex queues $L_u(v)$ and $L_s(v)$. $L_u(v)$ and $L_s(v)$ hold the unsettled and settled search labels at vertex v respectively. All priority queues are ordered by $\boldsymbol{T}_{[s\ldots v]}$ using the usual stochastic ordering $\preceq$ defined above. Each label in $L_s(v)$ corresponds to an s-v path already known to be feasible, and gives the required charging time at the last charging station. Consequently, as in [19], we maintain the invariant that the minimum element in $L_u(v)$ is not dominated by any label in $L_s(v)$.

The EV leaves s having acquired an SoC of β_s at s, so we treat s as a charging station, by default. One of our major challenges in the search will be to determine at which stations to charge, and for how long. Our search hence remembers the last charging station c along the route in the search labels, since dropping to an SoC below a permissible threshold

signals the need to include a charging time at c, and update route times accordingly.

The Search Algorithm

When the search reaches vertex v, the label at v is a four-tuple $\langle \boldsymbol{T}_{[s\ldots v]}, {}_c\beta, c, \delta_{[c\ldots v]} \rangle$, where $\boldsymbol{T}_{[s\ldots v]}$ is the travel time distribution for the subpath $[s \ldots v]$, c is the last charging station enroute from s to v, ${}_c\beta$ is the arrival SoC at c, and $\delta_{[c\ldots v]}$ is the depletion function of the subpath $[c \ldots v]$. We note that the charging times at some charging stations may be zero.

A label is extracted from $L_u(v)$ on each search iteration, where v is the minimum-travel travel time vertex in Q_G. It is then settled, and added to $L_s(v)$. A label in $L_s(v)$ represents a path from s to v that we know to be feasible, along with the exact charging time at the last charging station. A label in $L_u(v)$ represents a potentially feasible path that we haven't checked for feasibility. If an unsettled label in $L_u(v)$ is dominated by a label in $L_s(v)$, we can discontinue search along that path and discard that label, because we already know a better feasible path. The search proceeds as follows:

1. *At s*: Mark s as a charging station. Add the label $\langle \boldsymbol{T}_{\emptyset}, {}_s\beta, s, \delta_{\emptyset} \rangle$ to $L_u(s)$.

2. *At a non-charging vertex v*: Let $\ell = \langle \boldsymbol{T}_{[s\ldots v]}, {}_c\beta, c, \delta_{[c\ldots v]} \rangle$ be the label extracted from $L_u(v)$. Since ℓ indicates that c is the last charging station encountered, add label $\langle \boldsymbol{T}_{[s\ldots v]}, \delta_{[s\ldots v]}({}_s\beta), c, \delta_{[c\ldots v]} \rangle$ to $L_u(v)$ and update the travel times for v in Q_G.

3. *At a charging vertex v*: Let label $\ell = \langle \boldsymbol{T}_{[s\ldots v]}, {}_c\beta, c, \delta_{[c\ldots v]} \rangle$ be the minimum element extracted from $L_u(v)$. Let t_c be the charging time at the last charging station c, so that $\beta_c = \Phi_c({}_c\beta, t_c)$ is the SoC when the EV departs c.

The CFP algorithm of [18, 19] uses only deterministic travel times, but our travel times are distributions. As [18] shows, however, the charging times corresponding to the breakpoints of the charging function $\Phi_c(\cdot)$ capture the information required to make the required tradeoffs between charging times and travel times. To see how we approach the problem, let τ represent some value for the travel time from s to v, and compute

$$b_\ell(t_c, \tau) := \begin{cases} \delta_{[c\ldots v]}(\beta_c) & \text{if } t_c > 0 \text{ and } \boldsymbol{T}_{[s\ldots v]}(\tau) > 0 \\ -\infty & \text{otherwise} \end{cases}$$

Since the charging function $\Phi_c(\cdot)$ is assumed to be piecewise linear, its breakpoints induce breakpoints for b_ℓ. For a given value of τ we need to create one label per breakpoint of b_ℓ [18]. For a fixed τ and each breakpoint $B = (t_B, \mathrm{SoC}_B)$ of b_ℓ, we add to $L_u(v)$ the label $\langle t_B, \mathrm{SoC}_B, v, \boldsymbol{T}_\emptyset \rangle$, and update the travel times to v in Q_G.

In principle, τ can take an infinite number of values. We handle this difficulty by discretizing the domain of $\boldsymbol{T}_e$. We use histograms to represent $\boldsymbol{T}_e$ in our implementation, and generate one set of breakpoints per histogram bin (see section 4.7.1 for details).

4. *At the destination t*: End search, backtrack using parent pointers to extract an s-t path.

A label ℓ is said to dominate another label ℓ' if $b_\ell(t, \tau) \geq b_{\ell'}(t, \tau)$ for all $t > 0$ and all $\tau > 0$.

If we end the search only when Q_G is empty, not simply when t is reached, we obtain the $\mathbb{E}$-feasible non-dominated paths. For $\mathbb{E}$-feasible probabilistic budget paths, we end the search when $\boldsymbol{T}_P(d) = 0$, i.e. the probability of reaching t within the time budget d

drops to 0.

4.5 Charging Function Propagation for p-Feasible Routing

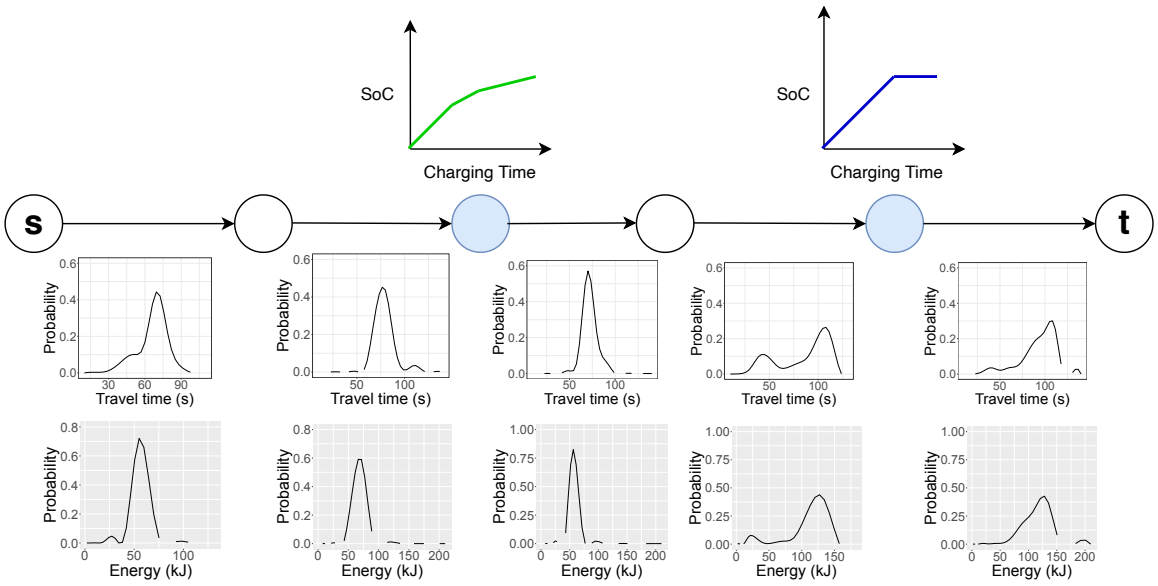


Figure 4.3: p-Feasible queries. Travel time and energy depletion are both distributions., propagated by the CFP search using convolutions. While the non-dominated search stops only when Q_G becomes empty, the probabilistic budget route search can stop when $\boldsymbol{T}_P(t)$ drops to 0.

For p-feasible routing, we must consider the actual depletion distribution $\boldsymbol{D}_P$ for a path P, not simply $\mathbb{E}[\boldsymbol{D}_P]$, which sufficed for expected-feasible paths. As with expected-feasible paths, we must also deal with the travel time distribution $\boldsymbol{T}_P$. If path P has the edges $e_1, e_2, \ldots, e_n$, then $\boldsymbol{T}_P = \boldsymbol{T}_1 * \boldsymbol{T}_2 * \cdots * \boldsymbol{T}_n$ and $\boldsymbol{D}_P = \boldsymbol{D}_1 * \boldsymbol{D}_2 * \cdots * \boldsymbol{D}_n$.

We say that a leg L of a route is p-feasible if the probability of successfully traversing L without being stranded is at least a query-specific threshold p. A route P with legs $L_1, L_2, \ldots, L_n$ is p-feasible iff each leg L_i is p-feasible.

We will call p the *non-stranding probability.* We can use p to place a bound on the maximum energy depletion we can accommodate over a path P. Let

$$\mathrm{C}_P(p) = \arg\max_x \{\boldsymbol{D}_P(x) \leq p\},$$

so that $\mathrm{C}_P(p)$ is the highest energy depletion that could occur along P with a probability of no more than p, that is, to ensure a non-stranding probability of p.

We define the *stochastic depletion function* analogously to Equation 4.2. Let $\mathrm{S}_P(p)$ be the minimum starting SoC at s required to traverse P with non-stranding probability p. Similarly, let $\mathrm{E}_P(p)$ be the maximum SoC possible on arriving at vertex t with at least probability p, and $\mathrm{C}_P = \boldsymbol{D}_P$. The *stochastic depletion function* for P is

$$\sigma_P(\beta_s, p) = {}_t\beta = \begin{cases} -\infty, & \text{if } \beta_s < \mathrm{S}_P(p), \\ \mathrm{E}_P(p), & \text{if } \beta_s - \mathrm{C}_P(p) > \mathrm{E}_P(p), \\ \beta_s - \mathrm{C}_P(p), & \text{otherwise.} \end{cases} \tag{4.3}$$

Let $\sigma_\emptyset$ be the *identity* stochastic depletion profile for a null path, so that $\sigma_\emptyset(\beta_s, p) = \beta_s$. If $P_1 = [v_i, v_{i+1}, \ldots, v_j]$ and $P_2 = [v_{j+1}, v_{j+2}, \ldots, v_k]$, the depletion profile of the concatenated path $P = P_1P_2 = [v_i \ldots v_k]$ is given by $\mathrm{S}_P(p) = \max\{\mathrm{S}_{P_1}(p), \mathrm{C}_{P_1}(p) + \mathrm{S}_{P_2}(p)\}$, $\mathrm{E}_P(p) = \min\{\mathrm{E}_{P_2}(p), \mathrm{E}_{P_1}(p) - \mathrm{c}_{P_2}(p)\}$ and $\mathrm{C}_P = \mathrm{C}_{P_1} * \mathrm{C}_{P_2}$.

4.5.1 Dijkstra Search for p-feasible Routes

The label at vertex v is a four-tuple $\langle \boldsymbol{T}_{[s\ldots v]}, {}_c\beta, c, \sigma_{[c\ldots v]} \rangle$, where $\boldsymbol{T}_{[s\ldots v]}$ is the travel time distribution for the subpath $[s \ldots v]$, c is the last charging station enroute from s to v, ${}_c\beta$ is the arrival SoC at c, and $\sigma_{[c\ldots v]}$ is the stochastic depletion function of the subpath $[c \ldots v]$.

As for $\mathbb{E}$-feasible routes, we maintain a global priority queue Q_G storing the travel time distributions from s, and queues $L_u(v)$ and $L_s(v)$ to store the unsettled and settled labels at vertex v respectively. All queues use the usual stochastic ordering. On each search iteration, a label is extracted from $L_u(v)$, where v is the minimum-travel time vertex in Q_G, settled, and added to $L_s(v)$. Each label in $L_s(v)$ represents a feasible path from s to v, including the charging time at the last charging station. Each label in $L_u(v)$ represents a potentially feasible path whose feasibility is yet unverified. If a label $\ell \in L_u(v)$ is dominated by $\ell' \in L_s(v)$, we can prune the search along that path and discard ℓ, because a faster feasible path is already known. p-feasible queries have four parameters: the source vertex s, the destination vertex t, the β_s, and the given p. The search proceeds as follows:

1. *At s*: Mark s as a charging station. Add the label $\langle \boldsymbol{T}_{\emptyset}, {}_s\beta, s, \sigma_{\emptyset} \rangle$ to $L_u(s)$.

2. *At a non-charging vertex v*: Let $\ell = \langle \boldsymbol{T}_{[s\ldots v]}, {}_c\beta, c, \sigma_{[c\ldots v]} \rangle$ be the label extracted from $L_u(v)$. Since c is the last charging station encountered on the route represented by ℓ, add label $\langle \boldsymbol{T}_{[s\ldots v]}, \sigma_{[s\ldots v]}({}_s\beta, p), c, \sigma_{[c\ldots v]} \rangle$ to $L_u(v)$ and update the travel times for v in Q_G.

3. *At a charging vertex v*: Let $\ell = \langle \boldsymbol{T}_{[s\ldots v]}, {}_c\beta, c, \sigma_{[c\ldots v]} \rangle$ be the label extracted from $L_u(v)$.

Let t_c be the charging time at the last charging station c, so $\beta_c = \Phi_c({}_c\beta, t_c)$. As with $\mathbb{E}$-feasible routes, the charging times corresponding to breakpoints of $\Phi_c(\cdot)$ suffice to make the required tradeoff between charging and travel times. Let τ represent some value for travel time from s to v, and compute

$$b'_\ell(t_c, \tau, p) := \begin{cases} \sigma_{[c\ldots v]}(\beta_c, p) & \text{if } t_c > 0 \text{ and } \boldsymbol{T}_{[s\ldots v]}(\tau) > 0 \\ -\infty & \text{otherwise} \end{cases}$$

Since $\Phi_c(\cdot)$ is piecewise linear, its breakpoints induce breakpoints for b_ℓ. Moreover, p is already known at query time, so for a given value of τ, we only need to create one label per breakpoint of b'_ℓ [18]. For a fixed τ and each breakpoint $B = (t_B, \mathrm{SoC}_B)$ of b'_ℓ, we add to $L_u(v)$ the label $\langle t_B, \mathrm{SoC}_B, v, \boldsymbol{T}_\emptyset \rangle$, and update the travel times to v in Q_G.

As with $\mathbb{E}$-feasible routes, τ can take an infinite number of values, but we use histograms to represent $\boldsymbol{T}_e$, and we need to generate only one set of breakpoints per histogram bin.

4. *At the destination t*: End search, backtrack using parent pointers to extract an s-t path.

For a given p, a label ℓ dominates another label ℓ' if $b'_\ell(t, \tau, p) \geq b'_\ell(t, \tau, p)$ for all $t > 0$ and $\tau > 0$. If the search terminates only when Q_G is empty, the resulting s-t paths are the p-feasible Non-Dominated Paths. For Probabilistic Budget queries, we end the search only when it reaches far enough for the probability of reaching t within the time budget d is 0.

4.6 Stochastic Contraction Hierarchies

For deterministic queries, Contraction Hierarchies (CHs) [64] are widely used for speed up. Graph vertices are ranked, and 'contracted' in ranked order. If u-v-w is a shortest path from u to w, vertex v is "contracted" by adding an edge u-w, and removing v from the graph. Such shortcuts significantly speed up the query-time Dijkstra search. The vertex ranks and the edge-weight hierarchy significantly affect preprocessing and query times [16]. A multicriteria CH variant is used in [153] for constrained shortest paths with positive weights. The CHArge algorithm [19, 18] combines a partial multicriteria CH with A* search. It contracts most graph vertices, creating a partial multicriteria CH but keeps an uncontracted *core* with charging stations. A* search using potential functions is used in the core to find routes at query time.

CHs have also been applied recently to stochastic route planning [120, 132]. However, we are interested in finding *feasible* routes that satisfy the energy bounds on EVs. Our queries are stochastic, and in fact doubly so. Travel time is always stochastic, and energy depletion is also stochastic for p-feasible queries. The stochastic dominance criterion is known to be too restrictive in practice [179], so it is hard to find dominating paths for most shortest paths in the network. Since the added shortcuts in CHs must not violate correctness, we can only avoid adding a shortcut covering a shortest path P only if we can find another witness path that dominates P [64, 153].

We solve this problem by relaxing our definition of dominance as follows. For distributions $\boldsymbol{T}_P$ and $\boldsymbol{D}_P$, we use the restricted-dominance criterion of [28], which checks if the CDF of one distribution is greater than that of the other within a fixed interval I, which

we set to two standard deviations on each side of $\mathbb{E}[\boldsymbol{T}_P]$ or $\mathbb{E}[\boldsymbol{D}_P]$. For search labels, we use a definition of ϵ-dominance similar to that of [14, 130]. We say that a label ℓ_1 dominates another label ℓ_2 if all breakpoints of b_{ℓ_1} or b'_{ℓ_1} have SoC_B values within ϵ of b_{ℓ_2} or b'_{ℓ_2}. We set $\epsilon = 2\%$ of battery capacity in our experiments.

4.7 Experiments

Our algorithms were implemented in Rust 1.60.0-nightly with full optimizations and run on an Intel core i5-8600K processor with 3.6GHz base clock, 192KB of L1, 1.5 MB of L2, and 9 MB of L3 cache and equipped with 64GB of dual-channel 3200MHz DDR4 RAM.

4.7.1 Preparing a realistic routing instance

We extracted traffic speeds from Mapbox Traffic Data[1] for Tile 0230123,[2] between 15$^{\text{th}}$ July and 30$^{\text{th}}$ November, 2019. Tile 0230123 covers Los Angeles county between Long Beach and Oxnard, and yielded a graph with 559,271 vertices and 1,058,450 edges. The dataset contained speed updates for an edge subset at 5-minute intervals, which we aggregated into weekday and weekend speed histograms. We discarded the weekend histograms due to sparsity, and and used only the weekday speeds for our experiments. We added latitudes and longitudes for each vertex from the OSM dataset taken from GeoFabrik,[3] contracted the degree-2 vertices, and extracted the largest connected component. This step resulted in the final routing graph of 244,728 vertices and 453,942 edges.

[1] `https://www.mapbox.com/traffic-data`
[2] `https://labs.mapbox.com/what-the-tile`
[3] `https://download.geofabrik.de/north-america/us/california/socal.html`

We added elevation data from the NASADEM dataset [106] at 30M resolution to each vertex, using bilinear interpolation to estimate elevations at vertex locations. Lastly, we obtained charging stations from the Alternative Fuels Data Center,[4] marking the vertex closest to each charging station as the charging vertex. The charging function Φ_c on each vertex c was linear, and either (1) a *slow*, charging to 100% in 100 minutes, or (2) *fast*, charging up to 80% in 30 minutes, and up to 100% in 60 minutes. We randomly assigned the slow charging function to 70% of charging stations, the fast charging function to the rest.

Energy consumption parameters for ε_e on all edges e were derived using the vertex elevations and the values a_e, b_e, c_e, d_e used for Nissan Leaf 2013 in [54]. To force the search to require charging en route for feasibility, we assumed that the EV had a 12 kWh battery.

Choice of edge weight representation: Histograms capture arbitrary $\boldsymbol{T}_e$ and $\boldsymbol{D}_e$ distributions, but take more space. Functions may be less faithful to real-world distributions, but are compact and may lead to faster queries in some cases [132]. We used histograms to represent the travel time and energy consumption distributions on edges since it allowed us to represent arbitrary distributions while keeping the implementation simple.

Applying Contraction Hierarchies: Building a full CH by contracting all vertices of the graph can be prohibitively expensive due to the high cost of contracting the highest ranked vertices. So, we build a only partial CH by contracting 97% of the vertices, keeping an uncontracted *core* containing all the charging stations on the network. Queries are run in three stages—from s to a vertex in the core restricted to using only (upward) edges from lower to higher ranked vertices, backward search from t to a vertex in the core using

[4] `https://afdc.energy.gov/fuels/electricity_locations.html`

downward edges, and a simple bidirectional search within the core of the network.

4.7.2 Results

Using stochastic edge weights raises many challenges that do not arise for deterministic weights. Two obvious issues are maintaining route feasibility, and aggregating edge distributions $\boldsymbol{T}_e$ and $\boldsymbol{D}_e$ into path distributions $\boldsymbol{T}_P$ or $\boldsymbol{D}_P$, which requires expensive convolutions. Several other issues also arise, two of which we will discuss.

Number of histogram bins: The time and energy value ranges in the path distributions $\boldsymbol{T}_P, \boldsymbol{D}_P$ increases linearly with the number of edges in P, so more histogram bins are needed to maintain accuracy. As in the deterministic case, the Dijkstra search labels track the travel time-charging time tradeoff. The labels represent histograms, so the label sizes increase with the number bins used for $\boldsymbol{T}_P$ and $\boldsymbol{D}_P$. Labels become progressively heavier for longer routes, raising the cost of all operations on the distributions, (convolution, dominance checks, etc.).

At charging stations, moreover, we must create a set of breakpoints per bin of the energy depletion histogram. More breakpoints are created for charging stations further along the route, increasing costs and making label dominance checks labels more difficult.

We also note that the CH shortcuts represent longer routes, whose histograms have more bins than the original graph edges. Shortcut edges are hence more expensive to handle than original graph edges, decreasing the utility of shortcuts in speeding up route planning queries.

Ensuring stochastic feasibility: Standard probabilistic budget routes use a single criterion, such as travel time [120, 132]. In contrast, our queries must handle search

with two criteria to maintain feasibility. Further, the number of breakpoints in the charging functions along a route determines the number of labels generated.

Table 4.3: Single-criterion probabilistic budget routing queries [132] vs. our $\mathbb{E}$-feasible and p-feasible queries on the Tile 0230123 graph. Times (seconds) are averages over 100 random vertex pairs. The energy consumption model is for a Nissan Leaf 2013 with 12 kWh battery and 50% starting SoC.

	Ignoring	$\mathbb{E}$-feasible	p-feasibleRoutes		
d	Feasibility	Routes	$p = 0.8$	$p = 0.85$	$p = 0.9$
5 min.	6.192	10.662	13.313	12.222	11.41
15 min.	19.999	24.711	39.977	40.24	37.11
25 min.	45.384	38.123	79.875	77.8	76.34

Table 4.3 quantifies the overhead of maintaining feasibility of routes in stochastic settings, and compares the query times for single-criteria probabilistic budget routes (time only, feasibility ignored) with those of our two-criteria feasible probabilistic budget routes. Single-criteria routing is fastest, followed by $\mathbb{E}$-feasible routing, and p-feasible routing. The anomaly for $d = 25$ minutes can be understood as follows. Multicriteria search must explore a larger set of routes from the source than single-criteria queries, because it needs to return the pareto frontier of routes, rather than a single route. The $\mathbb{E}$-feasible and p-feasible queries must also carry and update per-vertex labels, and maintain more information in each label to capture the travel time-charging time tradeoffs. However, we use the restricted dominance criterion for $\mathbb{E}$-feasible and p-feasible routes but not for the single-criteria routes, making the cost per convolution is slightly lower for the two feasible-path queries. This suffices to make $\mathbb{E}$-feasible routing slightly faster for longer routes than even single-criterion queries.

Table 4.4 compares $\mathbb{E}$-feasible and p-feasible queries, for longer deadlines. $\mathbb{E}$-

Table 4.4: $\mathbb{E}$-feasible and p-feasible query performance on the Tile 0230123 graph, with real-world charging station and elevation data. Times (seconds) are over 500 random vertex pairs. Energy consumption model is for a Nissan Leaf 2013 fitted with a 12 kWh battery and 50% starting SoC.

Query Type	**Feasibility Threshold**	**Time Budget (d)**			
		10 min.	**20 min.**	**30 min.**	**40 min.**
$\mathbb{E}$-feasible	—	18.01	34.975	48.662	72.198
p-feasible	$p = 0.8$	32.221	53.001	86.479	96.98
	$p = 0.85$	28.112	51.556	85.55	95.77
	$p = 0.9$	27.863	51.003	84.62	96.01

feasible queries are generally faster than p-feasible queries because they must convolve only $\boldsymbol{T}_P$, but p-feasible queries convolve both $\boldsymbol{T}_P$ and $\boldsymbol{D}_P$. p-feasible queries with higher p thresholds tend to run slightly faster, as they can prune the search quicker than searches run with lower p.

Table 4.5: Average Jaccard Index for 500 random $\mathbb{E}$-feasible and p-feasible routes, with $p = 0.85$. The index is 0 when the routes are edge-disjoint, and 1 when they are identical.

Queries Compared	d (min.)	Avg. Jaccard Index
$\mathbb{E}$-feasible and p-feasible, for $p = 0.85$	10 min	0.71
	20 min	0.74
	30 min	0.88
	40 min	0.96

Table 4.5 shows how similar the $\mathbb{E}$-feasible and p-feasible routes are, using the average Jaccard Similarity between the set edges of a route chosen by each of them. The Jaccard similarity for two routes P_1 and P_2 is the number of edges common to both divided

by the number of edges in their union. That is,

$$J(P_1, P_2) = \frac{|\{e \in P_1\} \cap \{e \in P_2\}|}{|\{e \in P_1\} \cup \{e \in P_2\}|}$$

The Jaccard index clearly increases with the time budget, so the $\mathbb{E}$-feasible and p-feasible routes are more similar when the routes are longer. This is because longer routes require more convolutions, making $\boldsymbol{D}_P$ closer to the Gaussian, which is more concentrated near its mean. In such cases, the pruning of edges forced by the feasibility criterion brings the set of edges of $\mathbb{E}$-feasible routes closer to the set of edges for p-feasible routing. For shorter routes, however, the difference between the two types of queries is higher. Hence, if stronger feasibility guarantees are desired for shorter routes, p-feasible queries may be better.

Chapter 5

The Tiering Technique for Stochastic Contraction & Edge Hierarchies

5.1 Introduction

For route planning, road networks are traditionally modeled as graphs with static edge weights representing travel times. In practice, however, no two vehicles travelling along the same road segment can be expected to take the *exactly* same amount of time. A more accurate model would represent edge weights as stochastic quantities, drawn from discrete or continuous travel time distributions. Fixed edge weights yield unique shortest paths, but with stochastic edge weights, one can define shortest paths only in a probabilistic sense. The shortest path computation is also more complicated, since edge weight distributions

cannot be directly compared. Route planning with stochastic edge weights allows for several types of shortest path queries [4, 6, 34, 37, 38, 74, 113, 116, 114, 115, 95, 102, 119, 120, 137, 169, 167, 180, 176]. Since stochastic route planning is inherently a harder problem than its deterministic counterpart, speedup techniques have been explored to make queries more practical [95, 120].

5.1.1 Contraction and Edge Hierarchies

Contraction Hiearchies (CHs) [64, 65] and Edge Hierarchies (EHs) [73] are speedup techniques originally developed for deterministic route planning, and find shortest paths in two stages. In the preprocessing stage, *shortcut edges* are added to the graph. In the query stage, these shortcut edges help answer shortest path queries quickly. CHs and EHs are similar, and work as follows: given a road network modeled as a graph with travel times as edge weights, each vertex (or edge) is assigned a *rank*, and the *contraction* operation is applied to all vertices (or edges) in increasing order of rank. Contracting a vertex adds a shortcut edge to the graph if it lies on the shortest path between its two of its neighbour vertices (or edges). The query stage runs a bidirectional Dijkstra's algorithm from the source and target vertices, settling only vertices (relaxing only edges) that have higher ranks than the source or target.

5.1.2 Handling Uncertain Edge Weights

CHs have been applied in uncertain settings [120], but no prior work exists on applying EHs in uncertain contexts. EHs have higher preprocessing costs since they preprocess more edges, and offer a finer grained hierarchy. They are also known to have worse

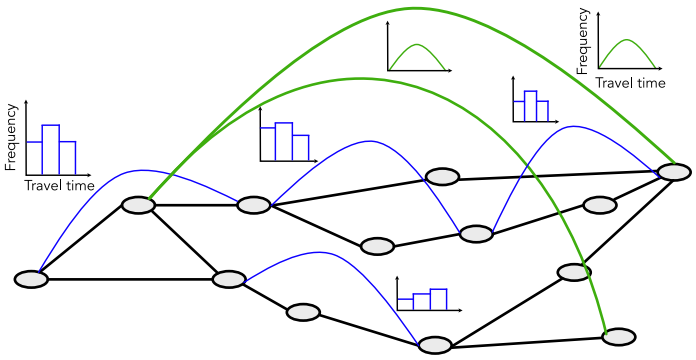


Figure 5.1: Histograms are better lower levels of a hierarchy (blue edges), as they can represent arbitrary distributions. At higher levels (green edges), shortcuts connect vertices farther away, and distributions such as Gaussians offer compactness and fast convolutions, while losing little accuracy.

query times for scalar edge weights. Nevertheless, EHs are more selective in relaxing edges in the query phase, and hold the the promise of better performance when relaxing edges is expensive. Finding stochastic shortest paths is one such application.

When edge weights are modeled as probability distributions, a major cost of finding shortest paths is computing convolutions of edge weights. Edge weight distributions can be represented in different ways, including histograms and continuous functions [169, 115]. The choice of edge weight representation can significantly affect shortest path query times, since each makes a different tradeoff between error, convolution costs, and space usage.

The Tiering Idea

In this work, we present the *tiering* technique for CHs and EHs, which intuitively works as follows: The set of shortcut edges in a CH or EH with uncertain weights is divided into a series of *tiers*, each tier using an edge weight representation suitable for that tier.

For instance, lower-ranked shortcut edges are likely to connect local vertices, with travel time distributions that are unusual and non-standard. Histograms, which can represent arbitrary distributions well, are likely to be better representations here. However, shortcuts ranking higher in the CH or EH connect far-away vertices, and represent travel over many graph edges. Their edge weight distributions are aggregations over multiple distributions, and likely to converge to stable distributions, such as the Gaussian. For most high-ranked shortcuts, using a simple Gaussian distribution offers fast convolutions and is compact and accurate.

We apply *tiering* to both EHs and CHs, and present Uncertain Contraction Hierarchies (UCHs) and Uncertain Edge Hierarchies (UEHs), whose edge weights represent stochastic travel times. Given a graph $G = \langle V, E \rangle$, a source $s \in V$ and destination $t \in V$, we study UCHs and UEHs for the following types of stochastic queries:

1. *Non-dominated routes*: Find routes between s and t that have edge weight distributions not dominated by other routes.

2. *Probabilistic budget routes*: Find a route that maximises the probability of travel cost being within a given budget b.

3. *Mean-risk routes*: Given a risk aversion coefficient $c \geq 0$, find routes minimizing (mean travel time $+ c * \sqrt{\text{variance}}$).

We show that tiered UCHs and UEHs offer much faster stochastic routing query times than their non-tiered variants for all three query types, when coupled with proper heuristics.

Current literature suggests that EHs have inferior query performance to CHs. However, we show that for all query types, UEHs can have query times comparable to UCHs,

under the proper optimizing heuristics. UEHs, however, still require higher preprocessing times. This is because while the query algorithms for both EH and CH are very similar, EHs relax far fewer edges due to their finer grained hierarchy, but CHs offer better stalling performance.

5.2 Related Work

Stochastic route planning dates to as far back as 1968, when [58] presented a Monte Carlo method to estimate the joint probability distribution of the shortest path in a graph with edge weights assumed to be probability distributions. Recent work on stochastic routes can be categorized in several ways: i) path versus edge-based algorithms, ii) whether edge weights are known when the vehicle reaches an edge iii) histogram versus continuous distributions used as edge weights. We look at these categorizations in this section.

Static vs adaptive edge weights: Two routing scenarios are possible here. First, given a graph and a set of edge weight distributions, we are to find the shortest path for a given definition of 'shortness' [120, 115, 114]. Second, the exact edge weight is 'revealed' when the search reaches a vertex, such as in the Stochastic On-Time Arrival (SOTA) and SPOTAR problems [113, 137, 6, 84], we are required to find the optimal *policy* for the driver to follow in order to have the highest probability of reaching the destination before deadline.

Path vs. edge centric routing: Most work on stochastic and deterministic routing is edge-centric. Edge weights or distributions are considered to be independent and the smallest unit of a path in a graph. Some authors [6, 40, 167, 169] argue that stochastic route planning should use paths and not edges as the smallest unit of routing, since travel

times between edges of a network can be correlated.

Histograms vs. distributions: Edge weight distributions may be modeled as functions [116, 115, 114, 95], or as histograms [6, 120, 169]. Histograms discretize time, and are easy to create from spatiotemporal probe data, but perform well only with sufficient data. Functions are difficult to obtain, but do not depend on the availability of data. A very recent preprint [78] attempts to bridge this divide by combining the advantages of the two representations.

5.3 Background

We are given a graph $G = \langle V, E \rangle$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges, and a stochastic edge weight function $W : E \rightarrow \boldsymbol{R}$ mapping each edge e to a random variable $R_e \geq 0$. A *path* is a sequence of vertices $[v_0 v_1 v_2 ... v_n]$ where $(v_i, v_{i+1}) \in E$. An *s-t path* is a path $[s = v_0 v_1 v_2 ... v_n = t]$ with s and t as the first and last vertices respectively.

The cost of path P is the sum of all stochastic edge weights along P, and is denoted $\text{cost}(P)$. If we have edges $e_1, e_2, \ldots, e_k$ along P, with weight distributions $W(e_1), W(e_2), \ldots, W(e_k)$, we denote the aggregate distribution along P through the convolution $\sum_{i=1}^{k} W(e_i)$.

5.3.1 Stochastic Route Planning

We are interested in three types of stochastic routing queries: probabilistic budget routing, non-dominated routes, and routes that optimize the mean-risk objective.

Probabilistic budget and non-dominated routing

In *probabilistic budget routing* [120], we are given a source $s \in V$, a target $t \in V$ and a cost budget $b \geq 0$. We are to find an s-t path P that maximizes the probability that $\text{cost}(P) \leq b$. A simple example would be a driver trying to reach an airport before a deadline b, which is the budget. We would want a route that maximises the probability of reaching the airport before the deadline.

The goal of *non-dominated routing* or *pareto-optimal routing* is to find to find the full set of paths between source and destination vertices that are not *dominated* by other paths. A path P is said to dominate another path P' if the travel times for P are always lower than that for P'. Edge weights in these problems can be required to satisfy the *First-In-First-Out (FIFO) property.* This 'no overtake' rule guarantees that vehicles using the same path will complete the trip in the same order that they started it. Stochastic routing has been studied both for cases when the edge weights satisfy the FIFO property [102, 180] and they do not [120].

The Mean-risk model

In the mean-risk model of paths, the objective is to minimize a linear combination of the mean and variance of edge weights along the path [114]. A typical metric to minimize is travel time *delays* along a path. Formally, we have a graph $G = \langle V, E \rangle$ and an edge weight function $W : E \to \boldsymbol{R}$ that assigns a Gaussian travel time delay distribution to each $e \in E$, a source $s \in V$, a target $t \in V$, and a risk-aversion coefficient $c \geq 0$. Then, if $\mu : E \to \mathbb{R}_+$ is a function that maps each edge e to its mean μ_e of travel time delays and $\tau : E \to \mathbb{R}_+$

maps every edge e to the variance τ_e of its travel time delay, our objective is to find a path P from s to t that minimizes $\sum_{e \in P}(\mu_e + c\sqrt{\tau_e})$.

The work in [95] showed how to make routing practical under the mean-risk model by dividing routing into two stages. In the *preprocessing stage*, a set of *distance oracles* [159] are created from the inputs. In the *query stage*, the distance oracles are used to speed up the queries. However, a drawback of this method is that c must lie within a pre-specified range.

5.3.2 Edge Hierarchies

Edge Hierarchies were introduced in [73] as a speedup technique for deterministic (non-stochastic) routing, and are closely related to Contraction Hierarchies [64]. Both techniques add *shortcut edges* to the graph in the preprocessing stage. The query phase consists of running a slightly modified bidirectional Dijkstra's search, where both forward and backward searches only settle vertices (or relax edges) that are ranked higher than the source in forward search and the target in the backward search. We detail the two stages in the remainder of this section.

The preprocessing stage

Let $G = \langle V, E \rangle$ be a graph and $W : E \to \mathbb{R}_+$ be a function that assigns static non-negative weights to each $e \in E$. Each edge in E is first given a *rank* according to some heuristic. Let $r(u, v)$ be the rank of the edge (u, v).

Initially, all edges are unranked. In each iteration, an unranked edge (u, v) is picked, and the *contraction* operation applied to it as follows: a Dijkstra's run is used to

determine if (u, v) lies on the shortest path between any unranked edges (u', u) and (v, v'). If it does, shortcut edges (u, v') and (u', v) are added to a set S. Next, the algorithm computes a minimum vertex cover of the bipartite graph in S. The edges that remain after the minimum vertex cover are then added to G and (u, v) removed.

The edge ranking heuristic can make a significant difference in both preprocessing and query times. In [73], the edges are ranked in *rounds*. At the beginning of each round, a subset of remaining unranked edges is picked. Edges are picked in increasing order of the number of shortcuts that would be added if they were to be contracted. The next round begins after all edges in the current round have been ranked. While the optimal order of edges can be found for Contraction Hierarchies [103, 16], finding the optimal ordering of edges to contract remains an open problem.

The query stage

In the query stage, rank 0 is assigned to the source $s \in V$ and target $t \in V$. Bidirectional Dijkstra's is now run so that in every iteration, only edges (u, v) with $r(u, v) > r(u)$ are relaxed. Also, the distance of a vertex v from source is updated in the priority queue while relaxing edge (u, v), $r(u) = r(u, v)$. The Dijkstra's search yields the shortest path between s and t.

The query stages of Contraction and Edge Hierarchies use *stalling* techniques to terminate search early, significantly lowering query times. CHs are known to perform well with *stall on demand*, where forward search terminates at $v \in V$ when a shortest path can not be found via the incoming edges in the backward search [73]. Similarly, the backward search terminates at $v \in V$ when a shortest path cannot be found via outgoing edges in

the forward search. However, stalling on demand can be potentially wasteful as it may relax every edge twice: once for settling, and once for stalling. Therefore, EHs use *stall in advance*, where the search relaxes every edge at most once. This is achieved as follows: when search reaches vertex v all edges (v, v') that are both higher and lower ranked than vertex v in the search are relaxed. The updated distance for lower ranked edges is then stored in a separate label, which can be used to check the distance in the stalling check.

EHs have been explored in some depth in [73], with results showing that CHs outperform EHs significantly. Our work shows however (Section 5.5), that this large performance disparity is not inherent to EHs. Our implementation of EHs, which uses adjacency lists, shows significant performance gains over that of [73].

5.4 Uncertain Hierarchies

Three major factors affect the performance of speedup techniques, beyond low-level optimizations: the structure of the road network, the available 'hierarchy' in edge weights that can be exploited by adding shortcuts to the graph, and the runtime cost of basic operations required to compute edge weights. For instance, CHs are known to perform well for graphs with a low road or skeleton dimension, and perform much better if travel times are used as edge weights, rather than physical distances between the vertices [31, 85]. Further, the cost of operations required on edge weights is a significant component of the algorithm engineering required for data structures such as the Time-dependent Uncertain Contraction Hierarchies [120]. Our goal in this section is to develop Uncertain Contraction and Edge Hierarchies for stochastic routing.

Problem Definition: Given a road network modeled as a graph $G = \langle V, E \rangle$ where V is the set of vertices and $E : V \times V$ is the set of edges, and an edge weight function $W : E \to \boldsymbol{R}$ that assigns to all $e \in E$ a random variable in $\boldsymbol{R}$ with range $\mathbb{R}_+$ representing uncertain travel times. Given a source $s \in V$ and target $t \in V$, we are to answer the following three queries:

1. *Probabilistic budget routes*: Given a budget b, find an s-t path P that maximizes the probability of $\text{cost}(P) \leq b$.

2. *Non-dominated routes*: Find the complete set of paths between s and t such that no route is 'dominated' by another.

3. *Mean-risk routes*: Given a risk-aversion coefficient $c \in \mathbb{R}_+$, find an s-t path P to minimize $\sum_{e \in P}(\mu_e + c\sqrt{\tau_e})$, where μ_e and τ_e are the mean and variance of the travel time on $e \in E$.

5.4.1 Tiering in Hierarchies

Travel-time distributions are often derived from collected trajectory data [169, 120, 35, 95] or other traffic sensors [9]. Broadly, there are two ways to represent uncertain edge weights: using histograms [120, 6, 169] or using continuous functions [115, 74, 167, 95]. Speed-up techniques for graphs with uncertain edge weights benefit greatly if the edge weight representations have the following properties:

1. *Accuracy*: The representation should capture all the information about the edge cost distribution without errors.

2. *Cheap convolutions*: Convolution is a basic operation in finding shortest paths, so representations that offer cheaper convolutions can improve query performance.

3. *Space efficiency*: Compact edge weight representations can have improve cache performance, reducing query times.

Histograms and continuous distributions make different tradeoffs between these properties. Since most real-world data is collected by sampling periodically, rather than continuously, histograms are usually the most accurate representations of available information. However, the source distributions can be arbitrary, so convolving two histograms can be expensive. In contrast, convolutions are much faster when the edge weights resemble stable distributions such as the Gaussian.

When using continuous functions to represent edge weight distributions, compactness and the convolution costs depends on the properties of the edge weight distributions. If all functions are stable distributions such as the Gaussian [115, 95], convolution is just the two additions, since for two random variables $F = \mathcal{N}(\mu_1, \sigma_1^2)$ and $G = \mathcal{N}(\mu_2, \sigma_2^2)$, $F * G = \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. However, using more complex representations, like Gaussian Mixture Models, can be less compact and have high convolution costs [167].

Definition 16. *A* tier T *in a CH or EH is the set of shortcut edges with ranks* $\tau^T_{\min} \leq r(e) < \tau^T_{\max}$ *for given thresholds* $\tau^T_{\min}$ *and* $\tau^T_{\max}$.

A tier T is marked as a *histogram tier* or a *function tier* depending on whether histograms or continuous functions are used to represent the edge weights for edges in T. For edges in a histogram tier, a histogram with a fixed bucket width w and number of

buckets b is used to represent edge weights. Similarly, edge weights in a function tier are represented by a mixture of one or more stable probability distributions.[1].

Definition 17. *A* tiered *contraction or edge hierarchy is a series of tiers* $[T_1, T_2, ...T_N]$ *such that* $\tau_{\min}^{T_{(i+1)}} = \tau_{\max}^{T_i} + 1,\ i = 1, \ldots, N$.

The next problem is to choose the number and type of tiers for the hierarchy. Here, the Central Limit Theorem suggests a useful heuristic: edges with weights derived from a large number of convolutions are likely to have distributions that approximate the Gaussian. Using this heuristic, we use a *two-tiered* contraction or edge hierarchy, which contains a histogram tier H with thresholds $\tau_{\min}^{H}$ and $\tau_{\max}^{H}$, and a Gaussian tier G with thresholds $\tau_{\min}^{G}$ and $\tau_{\max}^{G}$ that uses a Gaussian distribution for weights of all edges in G.

Note that [120] uses a similar heuristic for pruning Dijkstra's search. However, an important difference is that we use the central limit theorem to alter the *structure* of the contraction or edge hierarchy in the preprocessing stage. Moreover, we do not consider time-varying edge weights.

5.4.2 Uncertain Edge Hierarchies

We model Uncertain Edge Hierarchies (UEHs) as two-tiered hierarchies. We now show their construction and use to answer the three types of stochastic routing queries under consideration.

[1] A stable probability distribution is one such that the linear combination of two or more random variables with the distribution results in the same distribution.

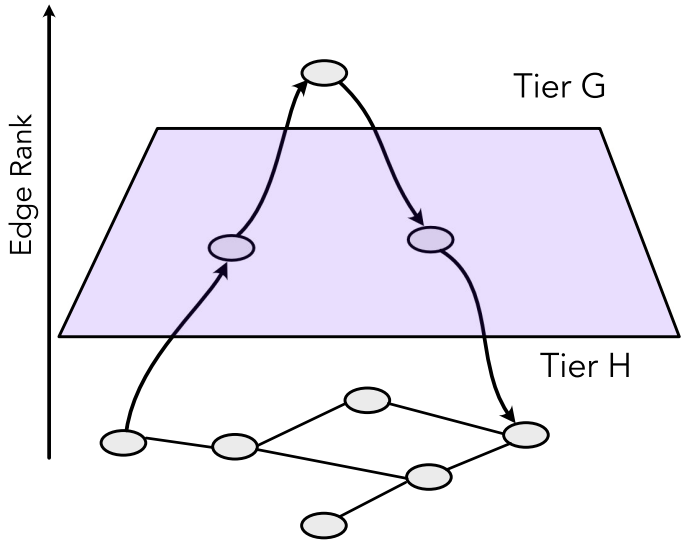


Figure 5.2: Tiering in shortcut hierarchies. Distributions are represented as histograms in Tier H, and as Gaussian approximations in Tier G.

Preprocessing

The edges in E are ranked in rounds. In each round, for every unranked edge, we find the number of edges that would be added to G if they were ranked in this round. Then, a set of edges that would add the minimum number of edges among their neighbor edges is picked and added to the ranking set S in increasing order. We then rank edges in S in the current round.

To rank an edge (u, v), we first run the *witness search* to find shortest paths from u' to v' such that $(u', u) \in E$ and $(v, v') \in E$. If (u, v) is on the shortest path from u' to v', we add (u, v) to S. Then, we find the bipartite minimum vertex cover MVC over S, and add all edges in MVC to G.

Determining the Tier Thresholds

We must next find suitable thresholds for the histogram tier H and the Gaussian tier G of the UEH. Since H is the lower tier, $\tau^H_{\min} = 1$.

We find $\tau^G_{\min}$ as follows. Since the number of edges is very large, we use sampling, and examine only one in λ edges. Let e be a sampled edge with edge weight histogram $W(e)$. We first construct a Gaussian distribution $\mathcal{N}(\mu_e, \sigma_e^2)$ with mean and variance equal to that of $W(e)$. We next sample this Gaussian and construct a histogram $G(e)$. Finally, we compute the KL-divergence [88] between $G(e)$ and $W(e)$ for the shortcut edges e being added. Given two discrete distributions G, W on a probability space χ,

$$D_{KL}(W\|G) = \sum_{x\in\chi} W(x) \log \frac{W(x)}{G(x)}.$$

If the KL-divergence is less than a predetermined similarity threshold σ_T for *all* the edges to be added, we set $\tau^G_{\min} = \lambda$ and $\tau^H_{\max} = \lambda - 1$. Algorithm 3 shows the pseudocode for computing UEHs.

The sampling frequency λ and similarity threshold σ_T are configuration parameters for the UEH. Since λ depends on the number of edges in the EH, $1 \leq \lambda \leq |E|$. The paramaeter λ presents a tradeoff between preprocessing times and query times. Similarly, σ_T trades off the *accuracy* for the routing query times. Setting a low λ means that we check for threshold of tiers H and G more often while building the UEH, resulting in faster query times but a much higher preprocessing cost. Conversely, setting λ too high would make tier H contain more edges than strictly required, and the queries would no longer benefit from

Algorithm 3 Building the two-level Uncertain Edge Hierarchy

procedure BUILDEDGEHIERARCHY
  $currentRank \leftarrow 0, T \leftarrow$ 'H'
  while unranked edges remain in G **do**
    Pick unranked edge (u, v); $r(u, v) \leftarrow currentRank$
    for unranked (u', u) **do**
      for unranked (v, v') **do**
        if (u, v) lies on shortest path from u' to v' **then**
          $S \leftarrow S \cup \{(u', u), (v, v')\}$
        end if
      end for
    end for
    MVC $\leftarrow$ Bipartite Minimum Vertex Cover over S
    if $currentRank$ is a multiple of λ **then**
      for $e \in MVC$ **do**
        $\mu_e \leftarrow$ mean, $\nu_e \leftarrow$ variance of $W(e)$
        $KL \leftarrow KL \cup \{D_{KL}[W(e) \parallel \mathcal{N}(\mu_e, \nu_e^2)]\}$
      end for
      if all edges in KL have similarity $< \sigma_T$ **then**
        $T \leftarrow$ 'G'
      end if
    end if
    Add edges in MVC to tier T
    $currentRank \leftarrow currentRank + 1$
  end while
end procedure

the cheaper cost of convolutions in tier G. On the other hand, setting a high σ_T means we approximate edge weight histograms with Gaussian distributions even when the two differ significantly, reducing query accuracy. Finally, setting σ_T too low slows down queries as the UEH contains mostly edge weight histograms that have a high computational cost for convolutions.

5.4.3 Stochastic Query Processing

Let $s \in V$ be the source and $t \in V$ be the destination. Since each edge is assigned an integer rank as in the deterministic EH, there exists an *up-down* path between s and t [73]. The search algorithm is a bidirectional Dijkstra's run from s and t, which first sets the vertex ranks $r(s) = 0$ and $r(t) = 0$. Then, search in both directions expands only edges with rank greater than $r(v)$ after reaching vertex v. If the distance of a vertex v from source is updated in the priority queue while relaxing edge (u, v), $r(u)$ is set to $r(u, v)$. The key difference UEHs and standard deterministic EHs, however, is that UEHs have edge weights represented as either histograms or continuous functions. Therefore, as the search progresses, it may relax edges from one or both tiers H and G. Whenever a Dijkstra's search starting from tier $T \in \{H, G\}$ reaches some edge $e \notin T$, we call e a *boundary edge*.

Lemma 18. *A shortest path between any two vertices s and t in a UEH can contain at most two boundary edges.*

Proof. Since all shortest paths in a UEH are up-down paths, let $P = [s = v_1, v_2, ..., v_m, ..., v_n = t]$ be a shortest path such that $r((v_i, v_{i+1})) < r((v_{i+1}, v_{i+2}))$ for $1 \leq i < m$, and $r((v_j, v_{j+1})) > r((v_{j+1}, v_{j+2}))$ for $m \leq j \leq n$. Only one of two cases can arise.

First, if rank $r((v_{m-1}, v_m)), < \tau^G_{min}$, all edges in P lie in tier H, and P has no boundary edges. Second, if $r((v_{m-1}, v_m)) \geq \tau^G_{min}$, we must have $i, j \in [1, n]$ such that $r((v_i, v_{i+1})) \leq \tau^G_{min} \leq r((v_{i+1}, v_{i+2}))$, and $r((v_j, v_{j+1})) \geq \tau^G_{min} \geq r((v_{j+1}, v_{j+2}))$. Then, (v_{i+1}, v_{i+2}) and (v_{i+1}, v_{i+2}) are the two boundary edges in P. □

Error Bounds for KL Divergence

In a UEH, the weight of a shortcut edge e is the convolution of all edge weight distributions that the shortcut replaces. However, if e lies in tier G, we store only a Gaussian *approximation* of the exact distribution such that the maximum KL Divergence between the two is σ_T. The maximum error this approximation induces for an edge weight is given by Pinsker's inequality [126]:

$$\|W(e) - \mathcal{N}(\mu_e, \nu_e^2)\|_1 \leq \sqrt{2\sigma_T}, \text{ where} \tag{5.1}$$

$$\|W(e) - \mathcal{N}(\mu_e, \nu_e^2)\|_1 = \sup\{|W(e)(x) - \mathcal{N}(\mu_e, \nu_e^2)(x)|, x \in \mathbb{R}\}$$

Here, $\|W(e) - \mathcal{N}(\mu_e, \nu_e^2)\|_1$ is the L_1 distance between the exact edge weight distribution of edge e, $W(e)$ and $\mathcal{N}(\mu_e, \nu_e^2)$, its Gaussian approximation. The L_1 distance is the maximum difference between the two values for any observation $x \in \mathbb{R}$ for which they are defined.

As longer shortcuts are added to the UEH at higher levels, we can determine the rate at which the edge weight distributions converge to Gaussian distributions. Using Lemma 18, an s-t shortest path P can be divided into a sequence of three subpaths $\{P_H, P_G, P_{H'}\}$ where P_H and $P_{H'}$ lie in tier H and P_G lies in tier G. The edge weights in P_H and $P_{H'}$ are exact histograms that induce no error, while those in P_G use approximations

with Gaussian distributions. Let $s(e)$ represent the edges in G that each shortcut edge $e \in P_G$ replaces. Then, the total number of "unpacked" edges in P_G is $\Sigma_{e \in P_G} s(e)$. Finally, the rate at which convolutions of edge weights in P_G converge to a Gaussian distribution is given by the Berry-Esseen theorem [29, 51],

$$\sup|S_n(x) - \Phi(x)| \leq \frac{C\Sigma_{i=1}^{n}\rho_i}{\sqrt{(\Sigma_{i=1}^{n}\sigma_i)^3}}, \text{ where } x \in \mathbb{R}. \tag{5.2}$$

Here, $S_n(x)$ is the CDF of $\Sigma_{e \in P_G} W(e)$, $\Phi(x)$ represents CDF of the standard Gaussian distribution, $C = 0.56$ [146], ρ_i is the expected value of third moment of i^{th} edge weight $< \infty$, σ_i is the variance of i^{th} edge and n is the total number of unpacked edges $\Sigma_{e \in P_G} s(e)$.

Equation 5.2 can be used to get the rate of convergence and error in a shortest path query only after all the edges along the shortest path from s to t are known. Further, every query can then be as inaccurate as the bound given by Equation 5.2.

Using the Hellinger Distance

An alternative approach to the UEH would be to use the Hellinger Distance (HD) [72] instead of KL-Divergence for constructing the hierarchy. For discrete probability distributions $P = \{P_1, P_2, ..., P_k\}$ and $Q = \{Q_1, Q_2, ..., Q_k\}$, the Hellinger distance is $H(P, Q) = \frac{1}{\sqrt{2}}\sqrt{\Sigma_{i=1}^{k}\left(\sqrt{P_i} - \sqrt{Q_i}\right)^2}$. Unlike KL Divergence, HD satisfies the triangle inequality, which can be used to reduce the approximation error by storing the HD in each edge weight in tier G, and including it in the dominance criterion in the query phase. Each stochastic routing query then returns the pareto-optimal set of routes with two objectives: to minimize the distance from s to t, and to minimize the approximation error. We call this

new variant of UEHs the *HD-UEH*. The original version of UEH is called the *KLD-UEH*, since it uses the KL Divergence to construct the hierarchy.

Query Processing Details

We now consider the three types of queries described in Section 5.4.

(1) Non-dominated routes: To find non-dominated routes, we maintain two label sets at all vertices. At $v \in V$, $L_{un}(v)$ and $L_{set}(v)$ respectively store the unsettled and settled labels for a Dijkstra search. Before each query, all sets are emptied. A bidirectional Dijkstra search starts from source s and target t, setting $r(s) = r(t) = 0$. On reaching vertex v, a Dijkstra label $\ell_v = \langle \text{dist}(v), \xi_v \rangle$ is created and added to $L_{un}(v)$, where dist(v) represents the convolution of all edge weights on the current path from s to v, and ξ_v is the error term. In KLD-UEH using KL-Divergence, ξ_v is set to a null value. In the HD-variant of UEH, when the search reaches a vertex u, on relaxing edge (u, v), we set $\xi_v = \xi_u + HD(u, v)$. Since HD satisfies the triangle inequality, $HD(s, v) < HD(s, u) + HD(u, v)$.

We bound the approximation error in an HD-UEH as follows. Assume that $P = [s = v_1, ..., v_b, v_{b+1}..., v_{b'}, v_{b'+1}, ..., v_n = t]$ is a shortest path, where (v_b, v_{b+1}) and $(v_{b'}, v_{b'+1})$ are the boundary edges. P comprises the three subpaths $S = [v_1, ..., v_b]$, $S' = [v_{b+1}, ..., v'_b]$ and $S'' = [v_{b'+1}, ..., v_n]$, where S and S'' lie in tier H, and S' lies entirely in tier G. Only S' can cause an approximation error in P.

We quantify the error from S' as follows. Assume that a Dijkstra search starts from v_b and reaches $v_{b'}$, spawning a label ℓ' at $v_{b'}$. Let $\xi_{b'} \in \ell'$ be the error term in label ℓ'. Let $C_{S'}$ be the convolution of the edge-weight histograms along the path, so $C_{S'} = \Sigma_{e \in S'} W(e)$.

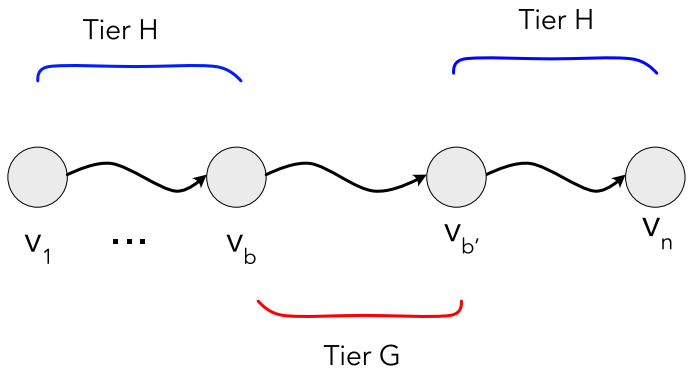


Figure 5.3: Computing approximation error on a path in UEH. Gaussian approximations are used only on the tier-G subpath. When this subpath is known, KLD-UEH uses Pinsker's inequality (Equation 5.1) to find the total error, while HD-UEH propagates an error term in Dijkstra's search labels.

Let $\text{cost}_{S'}$ be the convolution of the Gaussian approximations along the path. Then, $\xi_{b'}$ is an upper bound on the Hellinger Distance between $\text{cost}_{S'}$ and $C_{S'}$. By the definition of Hellinger Distance,

$$\xi_{b'} > \frac{1}{\sqrt{2}} \|\text{cost}_{S'} - C_{S'}\|_2$$

To compute the non-dominated routes, we do not terminate the Dijkstra search on reaching vertex t. The search terminates only when the priority queue contains no more vertices.

(2) Probabilistic budget routes A Probabilistic Budget Route query runs exactly like a non-dominated route query, with an additional pruning criterion: on relaxing an edge (u, v), we compare the distance from the s to v and compare it with b. If the search reaches a vertex with distance from source greater than b, it is terminated.

(3) Mean-risk objective Our method is similar to [95], which creates a set K of distance oracles [159] with deterministic edge weights, to $\epsilon-$*approximately* answer shortest path

queries that minimize the mean-risk objective. In their settings, $\forall e \in E$, $W(e) = \mathcal{N}(\mu_e, \tau_e^2)$. Then, given an ϵ, they set $\xi = \sqrt{\frac{\epsilon}{1+\epsilon}}$, $L = \min(\tau_e)$ and U equal to the maximum variance along any path in G. Finally, they show that it suffices to build distance oracles with edge lengths $l_k = k\mu_e + \tau_e$ for each $k \in \{L, (1+\xi)L, (1+\xi)^2 L, ..., U\}$, and collected in set K.

In [95], all the edge weights distributions are Gaussian. In a UEH, in contrast, only edges weights in tier G are Gaussian. Therefore, we build a set K of deterministic EHs for the subset of complete graph in tier G. Further, we set ϵ to a very low value, in order to avoid have approximation errors from both our approximations and their method. Therefore, we get ξ close to 1 and derive L and U empirically from the dataset.

A query for a risk aversion coefficient c runs like one for Non-Dominated Routes, with one change: after a Dijkstra search from $s \in V$ reaches a boundary edge b_e, the search in tier G runs a shortest path query on all EHs in set K. The path with minimum $(\mu + c\sqrt{\tau})$ among all shortest path queries is the shortest path in tier G. After the second boundary edge $b_{e'}$ along the search is reached, it progresses to target $t \in V$ by convolving histograms on the path.

5.4.4 Uncertain Contraction Hierarchies

We also model Uncertain Contraction Hierarchies (UCHs) as two-tiered hierarchies. The a lower tier H holds low-ranked local edges using histograms, and an upper tier G, storing only Gaussian approximations of edge weights. The thresholds of tier H are given by τ_{min}^H and τ_{max}^H, and those of tier G are given by τ_{min}^G and τ_{max}^G. We now describe the preprocessing and query algorithms for UCHs.

Preprocessing

We proceed as with UEHs. First, all vertices are ranked using a heuristic or an exact method. Then, the vertices are contracted in order of rank, and on every λ^{th} vertex contracted, we compute the KL-Divergence (or Hellinger Distance) between the edge weight of the shortcut being added and a Gaussian with same mean and variance. If on contracting $v \in V$, the KLD or HD of all shortcuts being added to the graph is less than σ_T, we set the threshold τ^G_{min} to the rank of vertex v.

Query Processing

All three query types can be handled with algorithms similar to those for UEHs, with two differences. First, while building UCHs for minimizing the mean-risk objective, we create CHs instead of EHs for the shortcut edges in tier G. Second, as with their deterministic versions, we use *stall on demand* for UCHs and *stall in advance* for UEHs.

5.4.5 Stable Distributions and Limitations

Our heuristic for tiering in a hierarchy uses edge weight representations at the lower levels more faithful to real-world data, but approximates with stable distributions at higher tiers, so convolutions are cheaper. This heuristic may not be universal, however. For example, some works use the log-normal and beta distributions as edge weights [84]. Our current heuristic also relies on convergence of edge weight distributions to stable distributions, such as Gaussians, for longer routes. We have found that this works well for travel times, but further work is needed to validate this for more general distributions. There may be statistical limit theorems that may be helpful for constructing suitable heuristics.

5.5 Experiments

To evaluate our methods, we implemented our algorithms in Rust and compiled them with rustc 1.54.0−nightly with full optimizations. All experiments are then run on an Ubuntu Linux machine running kernel 5.4.0 equipped with an Intel i5-8600K 3.6 GHz processor with 1.5 MB of L2 and 9 MB of L3 cache. The machine has 64 GBs of DDR4-2133 Mhz RAM.

5.5.1 Baselines for Deterministic Routing

The work in [73] suggests that CHs outperforms EHs significantly. However, their implementation uses adjacency arrays for CH, and adjacency lists for EH, confounding the effects of this difference with possible factors inherent to EHs. Our experiments show that the implementation makes a big difference, and any inherent performance disparities are smaller than previously thought. These are fairer comparisons between CH and EH.

First, we compare our implementation against reference implementations of Contraction and Edge Hierarchies, using static edge weights. We use the CH implementation from the RoutingKit library[2] and the EH [3] made available by the authors. Both reference implementations recommend using the GCC compiler toolchain, so they were compiled with GCC 10.2 with full optimizations. All benchmarks were run on the same machine. The graph datasets used are from the 9^{th} DIMACS challenge [4] and an instance of the road network of Los Angeles area taken from OpenStreetMaps (OSM). The edge weights in DIMACS instances are pre-populated. For the OSM dataset, we use the Vincenty's distance

[2] https://github.com/RoutingKit/RoutingKit
[3] https://github.com/Hespian/EdgeHierarchies
[4] http://www.diag.uniroma1.it//~challenge9/

between coordinates of adjacent vertices, and divide the distance by the maximum speed allowed on the road type to obtain travel times.

Dataset	Source	Vertices	Edges
New York	DIMACS	264346	733846
Bay Area	DIMACS	321270	800172
California & Nevada	DIMACS	1890815	4657742
USA West	DIMACS	6262104	15248146
Los Angeles Area	OSM	2549286	1666283
Tile 0230123 (contracted)	OSM	244728	453942

Table 5.1: The DIMACS graphs and the LA area OSM graph used to evaluate our CH and EH implementation, for fixed edge weights. Tile 0230123 is a part of the LA area OSM graph between Long Beach and Oxnard, covering most of LA city. We contract all vertices with degree < 2 for Tile 0230123.

Our implementation uses a different language and compiler toolchain than both reference implementations. Moreover, both our CH and EH implementations represent graphs as adjacency lists. This is in contrast with RoutingKit, which uses adjacency arrays. The effects of this choice are clearly visible in Table 5.3. Our EH implementation has much better preprocessing *and* query performance than the implementation in [73]. This is because our implementation shows much better cache utilization than that of [73]. For CHs, our implementation has somewhat better preprocessing times with travel times as edge weights, but worse preprocessing times with the distance metric. The query times of our CH implementation are worse than RoutingKit for both metrics due to the difference in graph representations.

5.5.2 Stochastic Routing

Our CH and EH implementations are generic, and support scalar or composite edge weights such as histograms and continuous functions. We benchmark the three kinds of stochastic queries on Tileset 0230123, a subset of the Los Angeles OSM graph.[5].

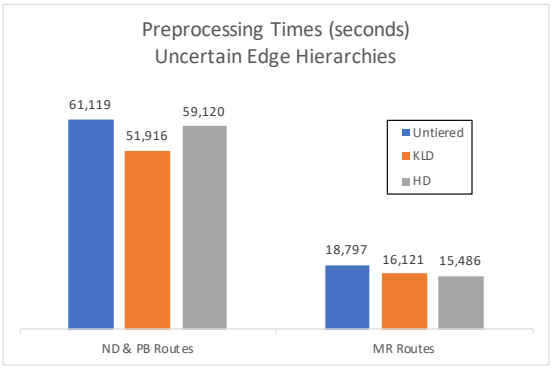
The travel time distributions are obtained from the Mapbox traffic data API [6] for Tile ID 0230123, which updates the edge travel times at 5-minute intervals. We collected the data for four and a half months between 15$^{\text{th}}$ July and 30$^{\text{th}}$ November 2019, giving us 42,299 travel time updates of the underlying graph edges. These updates were grouped into 30-minute intervals over the 24 hours in a day, then histograms extracted separately for weekdays and weekends. The weekend travel time histograms are much sparser than those for weekdays, and are therefore not used for our experiments. This is because both UCHs and UEHs use histograms to represent edge weights at lower levels of the hierarchy, which are known to be inaccurate when number of observations is low [78].

To ensure reasonable query processing times, we contracted the vertices in the road network with degree ¡ 2. This reduces the number of graph vertices and edges significantly, speeds up preprocessing and query times without losing accuracy. This method is used for maps derived from OpenStreetMaps in prior works [13].

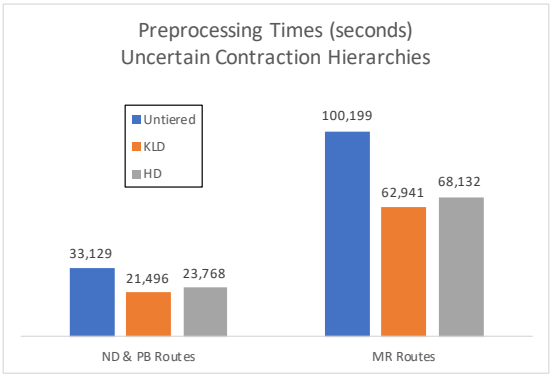
Preprocessing Times: Figure 5.4 shows preprocessing times for all three query types. Non-Dominated Routes and Probabilistic Budget Routes can both be computed on the same underlying graph, so we construct only one UEH and UCH of each type for these query types. The preprocessing times for Mean-risk routes, however, are much larger than

[5] https://labs.mapbox.com/what-the-tile/

[6] https://www.mapbox.com/traffic-data



(a) Uncertain Edge Hierarchies



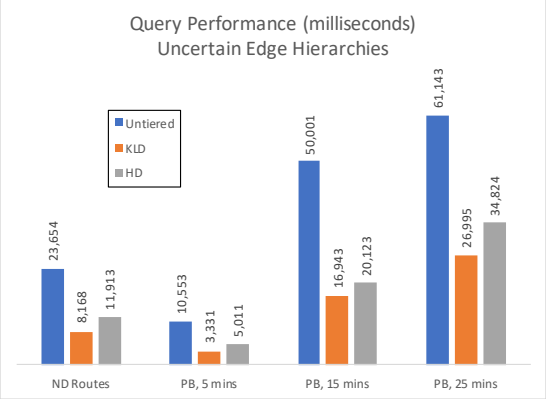
(b) Uncertain Contraction Hierarchies

Figure 5.4: Preprocessing times for the tiered and untiered uncertain EHs and CHs on the contracted Tile 0230123 road network.

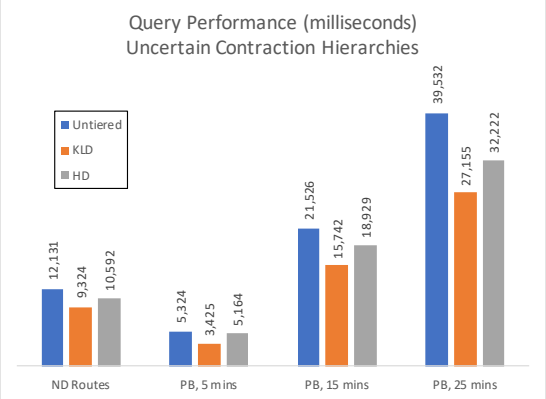
for the other queries. This is because we must construct a *set* of EHs or CHs for a subset of the graph to answer mean-risk queries. For all hierarchies, we set $\sigma_T = 0.2$ and $\lambda = 100$.

We see that the Untiered hierarchies take the longest time to build, followed by the HD and KLD variants of UEH, and the HD and KLD variants of UCH. This is because of the convolution on histograms is expensive. The Untiered UCH and UEH lack tier G which offers cheap convolutions, and incur a high cost for witness searches, increasing the preprocessing time. UCH has lower preprocessing time than UEH because of the same reasons as in deterministic routing; it offers a coarser hierarchy, needs to to preprocess far fewer vertices, and can avoid the costly Minimum Vertex Cover computation on each vertex contraction.

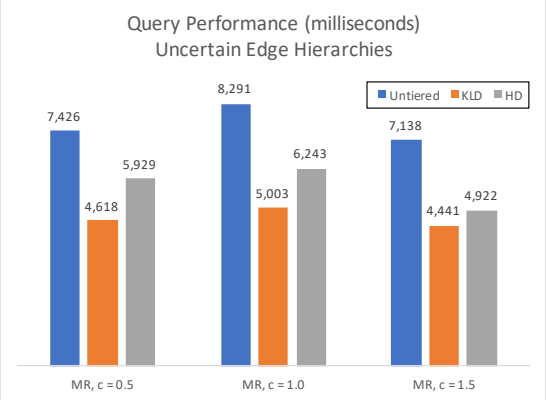
Query times The query times are shown in Figure 5.5. Routes minimizing the mean-risk objective tend to be the fastest to compute. This is due to having a set of deterministic EHs and CHs for shortcuts in tier G. The very low cost of finding routes in a deterministic hierarchy outweighs having to run the shortest path queries for each



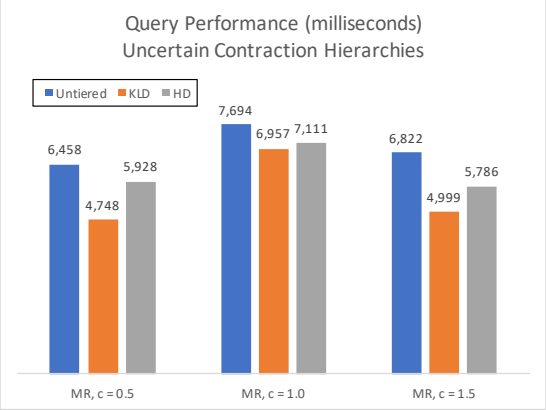
(a) Non-Dominated and Probabilistic Budget routes with $b \in \{5, 15, 25\}$ minutes.



(b) Non-dominated and Probabilistic Budget routes with $b \in \{5, 15, 25\}$ minutes.



(c) Mean-risk routes with $c \in \{0.5, 1.0, 1.5\}$



(d) Mean-risk routes with $c \in \{0.5, 1.0, 1.5\}$

Figure 5.5: Query times for all three query types, using the tiered and untiered uncertain CHs and EHs on the contracted road network for Tile 0230123.

EH or CH in K. For Probabilistic Budget Routes, the query time increases as the budget increases, because the search can reach vertices farther from the source. However, for mean-risk routes, the correlation between c and query times is not well-defined, and can depend on the locations of source and target vertices, variance in edge weights in the vicinity, etc.

Next, we see that for all three query types, Untiered hierarchies are the slowest, followed by Hellinger Distance variants of the UEH and UCH. The KLD variants tend to be the fastest. This is due to three reasons: first, HD-UEH and HD-UCH store the Hellinger Distance on each edge of the road graph, and on running queries, include it in the dominance criterion for Dijkstra's labels. In other words, in KLD-UCH or UEH, a Dijkstra's search label at vertex v, ℓ_v *dominates* ℓ'_v iff the route represented by ℓ_v always takes more time to traverse than one represented by ℓ'_v. For the HD variants presented, in addition to travel time dominance, ℓ_v must have a smaller Hellinger Distance as compared to ℓ'_v. This makes it hard for a label to dominate another, and fewer searches can be pruned, which results in slower query times.

Effect of approximation error on routes Since tiered hierarchies approximate edge weight histograms in tier G, they trade off some accuracy for better query processing times. Table 5.4 shows the percentage change in mean travel times for a 100 random stochastic routing queries running on untiered versus tiered hierarchies.

We note that generally, the travel time increases when approximate edge weights are used. This is because when Gaussian approximations of histograms are computed in tier G, the Gaussians approximations typically tend to overestimate the histograms.

		Contraction Hierarchies						Edge Hierarchies					
	Dataset	Preprocessing (ms)			Query (μs)			Preprocessing (ms)			Query (μs)		
		RtKt	Ours	% Gain	RtKt	Ours	%Gain	[73]	Ours	%Gain	[73]	Ours	%Gain
Travel Time	New York	7183	7130	0.7%	15	23	-53.3%	427456	285471	33.2%	41	35	14.6%
	Bay Area	4329	4220	2.5%	10	15	-50.0%	267031	241691	9.5%	27	22	18.5%
	CA & NV	28065	27297	2.7%	16	29	-81.3%	1592972	1258829	21.0%	46	39	15.2%
	USA West	96388	90301	6.3%	22	51	-131.8%	4783188	3772306	21.1%	62	53	14.5%
	L.A. Area	6158	5543	10.0%	13	20	-53.8%	384892	318524	17.2%	33	27	18.2%
	T. 0230123	3954	4523	14.4%	9	16	-77.8%	297123	279923	5.8%	26	20	23.1%

Table 5.2: Deterministic routing using the distance metric: Our CH and EH implementation uses an adjacency list representation for both speedup techniques, and performs better than the original EH implementation but is slower than RoutingKit. The performance gap between CH and EH techniques when using the same underlying graph representation is lesser than originally reported.

		Contraction Hierarchies						Edge Hierarchies					
	Dataset	Preprocessing (ms)			Query (μs)			Preprocessing (ms)			Query (μs)		
		RtKt	Ours	% Gain	RtKt	Ours	%Gain	[73]	Ours	%Gain	[73]	Ours	%Gain
Distance	New York	11728	12948	-10.4%	31	45	-45.2%	750250	685268	8.7%	82	71	13.4%
	Bay Area	6574	6930	-5.4%	19	27	-42.1%	448575	329455	26.6%	52	44	15.4%
	CA & NV	46366	50537	-9.0%	39	71	-82.1%	3061041	268851	91.2%	117	102	12.8%
	USA West	156994	167487	-6.7%	59	108	-83.1%	9150456	7752139	15.3%	186	152	18.3%
	L.A. Area	8853	9412	-6.3%	26	44	-69.2%	661573	543212	17.9%	72	63	12.5%
	T. 0230123	5423	8121	-49.8%	13	22	-69.2%	450032	392191	12.9%	43	32	25.6%

Table 5.3: Deterministic routing using the travel time metric: Our CH and EH implementation uses an adjacency list representation for both speedup techniques, and performs better than the original EH implementation but is slower than RoutingKit. The performance gap between CH and EH techniques when using the same underlying graph representation is lesser than originally reported.

Change in mean travel times (%)	**Uncertain Contraction Hierarchies**			**Uncertain Edge Hierarchies**		
	Untiered	KLD	HD	Untiered	KLD	HD
Non-dominated Routes	0	+5.9%	+3.3%	0	+7.1%	+5.4%
Probabilistic Budget Routes						
$b = 5$ minutes	0	-13.6%	-8.6%	0	+4.6%	+4.7%
$b = 15$ minutes	0	+6.2%	+5.3%	0	+9.5%	+7.7%
$b = 25$ minutes	0	+8.1%	+6.7%	0	+5.9%	+5.9%

Table 5.4: Effect of approximation error on route travel times in UEH and UCH: Routes generally take slightly more time when edge weight approximations are used as compared to Untiered hierarchies.

Chapter 6

Shared Dwell Regions

6.1 Introduction

A region R is called a *dwell region* for a moving object O if, given a threshold distance d_q and duration t_q, every point of R remains within distance d_q of O for at least time t_q [160]. Dwell region queries can be considered a generalization of the stay points problem [91], which finds applications in several different domains: wildlife exploration, finding intergalactic habitable zones [69, 104], trajectory summarization [175, 123], etc.

In this work, we are interested in *shared* dwell regions between different trajectories. In particular, we seek to answer the following queries efficiently:

Query 5 (**Shared Dwell Region**). *Given a set of trajectories S, and a query (W, t_q, d_q), where W is a time window, and t_q is a duration and d_q is a duration, find the set of regions $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ such that all points in each R_i remain within distance d_q of* all trajectories in S, *for some duration $t \geq t_q$ within the time window W.*

That is, within time window W, each R_i is an intersection of dwell regions corresponding to a given threshold distance d_q and duration t_q for all trajectories in S.

Query 6 (N/S **Shared Dwell Region**). *Given a set of trajectories S, and a query (W, N, t_q, d_q), where W is a time window, $N < S$ is a count, d_q is a duration, and t_q is a duration, find the set of regions $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ such that all points in each R_i remain within distance d_q of* at least N trajectories in S*, for some duration $t \geq t_q$ within the time window W.*

In [160], the authors propose an online algorithm to approximate the dwell region to arbitrary precision, by maintaining the projections of each point in the trajectory in k directions in $\mathbb{R}^2$. A naïve approach to solving the shared dwell region queries would be to simply find the dwell regions of all trajectories in the given dataset, and then to find their intersections. However, finding the dwell regions of each trajectory individually can be prohibitively expensive when the number of trajectories is large. Since [160] uses k heaps to answer an online dwell region query for a single trajectory, a total of $k|S|$ heaps must be maintained as point updates are received for any trajectory. Moreover, once the dwell regions for all trajectories are found, finding the *precise* shared dwell region requires additional work, since each dwell region can be of arbitrary shape and size.

In this work, we present an alternative method of finding approximate shared dwell regions by casting the problem as a variant of the pointwise dense region problem of [110]. At time t, a point $p \in \mathbb{R}^2$ is said to be ρ-dense if the a square of side l with p as its center contains at least n moving objects, so that the density $d_t(p) = \frac{n}{l^2} \geq \rho$. A pointwise-dense region is a collection of contiguous ρ-dense points at time t. In [110], the authors present a

filtering-refinement approach to finding all ρ-dense regions at t. We show that this approach can be extended to answer the Shared N/S dwell regions queries efficiently, for both online and offline settings.

6.1.1 Our contributions

- We show that the shared and N/S shared dwell region queries can be formulated as variants of the pointwise dense region queries, yielding efficient approximate algorithms that can answer queries quickly.

- We propose two methods: an online method that we call the *replay method*, which can be used to determine shared dwell regions as location updates are received from points, and an offline method, which can be used for archived sets of trajectories. Our methods are simple to implement, easily parallelizable, and make efficient use of hardware to achieve fast query times.

- We evaluate our methods on the T-Drive trajectory dataset containing 17 million GPS points collected from 10,000 real-world taxis over the span of a week in Beijing, China.

6.2 Preliminaries

In this section, we review the existing work on Dwell Region and the Pointwise Dense Region queries, and briefly describe the existing solutions proposed in literature.

6.2.1 Dwell Regions

Definition 19 (Dwell Regions)**.** *A region R is a* dwell region *for a moving object O if, given a threshold distance d_q and duration t_q, every point of R remains within distance d_q of O for at least time t_q.*

[160] present an online method to compute the dwell regions for a moving object, which is based on the following observation: The smallest dwell region that can exist is a single point, which is also the center of the smallest enclosing circle, SEC_S, of points in S. This case occurs when the objects are equally angularly spaced around SEC_S, and d_q is equal to the radius of SEC_S. Further, the center of SEC_S is a part of all dwell regions, if they exist, for any values of t_q and d_q. Therefore, the method maintains an approximation of SEC_S by a k-sided polygon, which can be updated quickly as more points are received. A detailed description of the method follows:

Point updates: For a moving object, two data structures are maintained: a window of location updates S of fixed size, and a set of k max-heaps. Each max-heap stores the projection of points in S on k unit vectors at the origin, spaced so they divide the unit circle into equal sectors. The heaps are used to maintain MBP_S, a k-sided Minimum Bounding Polygon of points in S. Using MBP_S, we can derive a lower bound $\lfloor SEC_S \rfloor$, and upper bound $\lceil SEC_S \rceil$ on the radius of SEC_S.

Query processing: When a query (d_q, t_q) is received, if $d_q < \lfloor SEC_S \rfloor$, no dwell region can exist, and no further processing is required. If $d_q > \lceil SEC_S \rceil$, a dwell region definitely exists, and can be derived by computing the intersections of circles with radii r_q and centers at *frontier points* in the k directions. Similarly, if $\lfloor SEC_S \rfloor < d_q < \lceil SEC_S \rceil$, the radius of

SEC_S can be determined exactly using the convex hull of frontier points, which can then be used to determine whether a dwell region exists. Since the number of frontier points is shown to be much smaller than the total number of points in a trajectory, the method scales to a large number of trajectories.

However, this method is not a good fit for the shared and N/S dwell region queries we are interested in. This approach requires us to maintain k heaps for each of the S trajectories from which point updates are being received, in order to answer the queries online. This can be prohibitively expensive in both memory and time. Moreover, the algorithm assumes that position updates received from different objects are known with perfect accuracy, which in practice, is unlikely to be the case.

6.2.2 Pointwise Dense Regions

Definition 20 (**l-square** neighbourhood)**.** *An l-square neighbourhood S_p^l of a point p is the square centered around p with edge length l, including the right and top edges, but excluding the left and bottom edges.*

Definition 21 (**point density**)**.** *The point density $d_t(p)$ at a time t of a point p is the value $n_t(S_p^l)/l^2$, where $n_t(S_p^l)$ is the number of moving objects located within S_p^l at time t.*

Definition 22 (**ρ-density**)**.** *Given $\rho \geq 0$, point p is considered ρ-dense with respect to an l-square neighbourhood at time t, if $d_t(p) \geq \rho$. A region is ρ-dense with respect to l-square neighbourhoods at time t if every point inside the region is ρ-dense at time t.*

[110] offer a filtering-refinement approach to answer the pointwise dense region queries exactly, which works as follows: The approach maintains two data structures to

store the point updates of moving objects: a density histogram DH, and a TPR-tree containing all positions received from moving objects. The temporal extent of trajectories is partitioned into bins of fixed duration H. Then, for each timestamp $\tau \in [t, t+H]$, the spatial extent of moving objects is divided into a grid of size $G = m \times m$. m is chosen such that the cell edge length $l_c = L/m \leq l_{min}/2$, where $L \times L$ is the extent in which the objects are moving on the plane. Then, any pointwise dense region queries can be answered for l-square neighourhoods with $l \geq l_{min}$ in two stages- filtering and refinement.

In the filtering stage, using DH, the cells in G are classified as either guaranteed dense, guaranteed not dense, or *candidate* cells for further processing. In the refinement stage, a band sweep is carried out to identify dense areas within each candidate cell. The sweep uses a band of width l and computes pointwise densities only when a vertical line at center of the band touches points within the cell (tracked using the TPR-tree). [110] shows that the filtering-refinement approach scales well to large areas and number of points, and also provides approximation algorithms using Chebyshev polynomials that can answer Pointwise Dense Region queries quickly.

6.3 Approximate dwell region processing

In this section, we outline two methods to determine approximate dwell region queries. Our methods are based on the following observation: each location reading from objects is not perfectly accurate, and depends on the quality of the sensors, the immediate environment etc. So, it is wasteful to represent the location of a moving object exactly as it is received, as only a small number of significant digits of the received location may be

actually correct. Therefore, we divide the spatial area of trajectories into a grid with cell size approximately equal to the accuracy to which each object's location may be known in the dataset. For instance, a grid size of 25000 $\times$ 25000 cells suffices to divide the extent of Beijing into cells of side approximately 11 meters, which roughly matches the accuracy of most consumer-grade GPS sensors. This transformation allows us to represent the position in a point update of a trajectory as an ID of a grid cell. Our methods then use this property to build efficient methods to find shared dwell regions between trajectories. Since our methods essentially quantize the spatial extent of considered trajectories, we can only answer the shared dwell regions queries approximately, with a minimum resolution set by cells of the considered spatial grid. The resulting algorithms are less general than that of [160], but we show that they result in faster query times.

6.3.1 The Replay (Online) Method

Intuitively, our method works as follows: since we assume that the spatial extent of trajectories is divided into a grid of a given size, now we can track the movement of objects in the real world on this spatial grid, and extract the regions that are common to at least N of d_q-square neighbourhoods of trajectories.

We track the movement of objects on the spatial grid in terms of the *movement events* they generate. We say that a movement event occurs for an object if its location changes between two consecutive point updates. To track the set of movement events, we create a segment tree ST over a 2-D plane, with one axis being the IDs of the moving objects in the dataset, and the other axis being timestamps. Figure 6.1 illustrates our method.

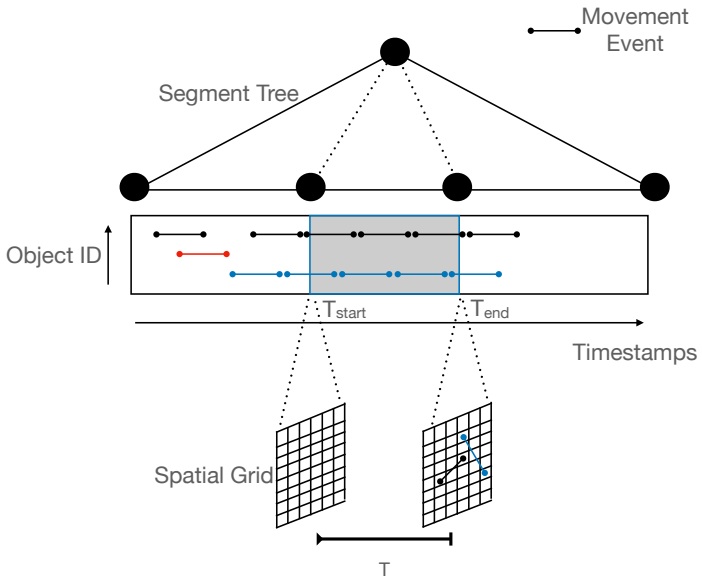


Figure 6.1: The online shared dwell region method using a segment tree of movement event time intervals. For query $\langle T_{start}, T_{end}, d_q, t_q \rangle$, we extract all events in the time interval $\langle W = T_{start}, T_{end} \rangle$ from the segment tree, find corresponding object locations, and increase the cell counts within a radius d_q of the object. Each cell c_{ij} holds a list of $\langle \tau_s, \tau_e \rangle$ pairs, where τ_s is the time when c_{ij} first came within a radius d_q of an object O, and τ_e is the time when c_{ij} ceased to be within a radius d_q from O. We extract the entries where $(\tau_e - \tau_s) \geq t_q$. The resulting set of cells form the shared dwell regions.

Point Update: A position update $U_{i,t} = \langle i, x, y, t \rangle$ indicates that moving object O_i has coordinates x, y at time t. From $U_{i,t}$, we generate a movement event, which is a 3-tuple $\langle t_{prev}, t, i \rangle$, where t_{prev} is the timestamp of last position update received from O_i before $U_{i,t}$. A movement event e is a horizontal line on the considered plane, indexed by the ST.

Query: When a query $\langle T_{start}, T_{end}, t_q, d_q \rangle$ is received, we first extract all movement events in the time interval $\langle T_{start}, T_{end} \rangle$ into a set M. Then, for each event in $e \in M$, we retrieve the object's positions $p_1 = \langle x_1, y_1 \rangle$ at time t_{prev} and $p_2 = \langle x_2, y_2 \rangle$ at time t. Next, we iterate over all cells in the grid whose centers are within $(2 * d_q)$-square neighbourhoods of p_1 and p_2, increasing the counter for each cell so encountered. We call this operation the *application*

of event e on the grid G. Each cell c_{ij} in G also holds a list of tuples $\langle \tau_s, \tau_e \rangle$, where τ_s is the timestamp at which c_{ij} entered the d_q-square neighbourhood of a moving object O, and τ_e is the timestamp at which c_{ij} stopped being a part of the d_q-square neighbourhood of O.

After all events in M have been applied to G, we filter the cells with counts greater than or equal to N. Then, for each filtered cell, we find intervals where $(\tau_e - \tau_s) \geq t_q$. If such an interval exists for a cell c_{ij}, the cell is marked. The set of all marked cells is finally returned as the result of the shared dwell region query. Table 6.1 compares the algorithmic complexity of our method with the original SEC-tracking method of [160].

Table 6.1: Comparing the cost of maintaining a window of W position updates, adding E new point updates, and the approximation error for queries in our method vs. the online method presented in [160]. Here, l_G is the side of a cell in the considered grid, k is the number of vectors maintained around the origin, d_q is the query distance.

Method	Maintaining W	E point updates	Approximation Error
Heaps-based [160]	$O(kW \log(W))$	$kE \log(W)$	Error in $SEC_S = O(1/k^2)$
Pointwise	$(\frac{d_q}{l_G})^2 W$	$O(\frac{d_q}{l_G})^2 E$	Inv. proportional to l_G^2

6.3.2 The Offline Method

We now seek to build an index for large datasets of archived trajectories, in order to answer shared dwell region queries quickly. The replay method applies all movement events in a specified time window to the spatial grid at query time. While this method requires minimal preprocessing, i.e. only building the segment tree, which can be done for several million movement events in only a few seconds (In our experiments, building a segment tree for all movement events in the T-Drive dataset of 17 million points takes

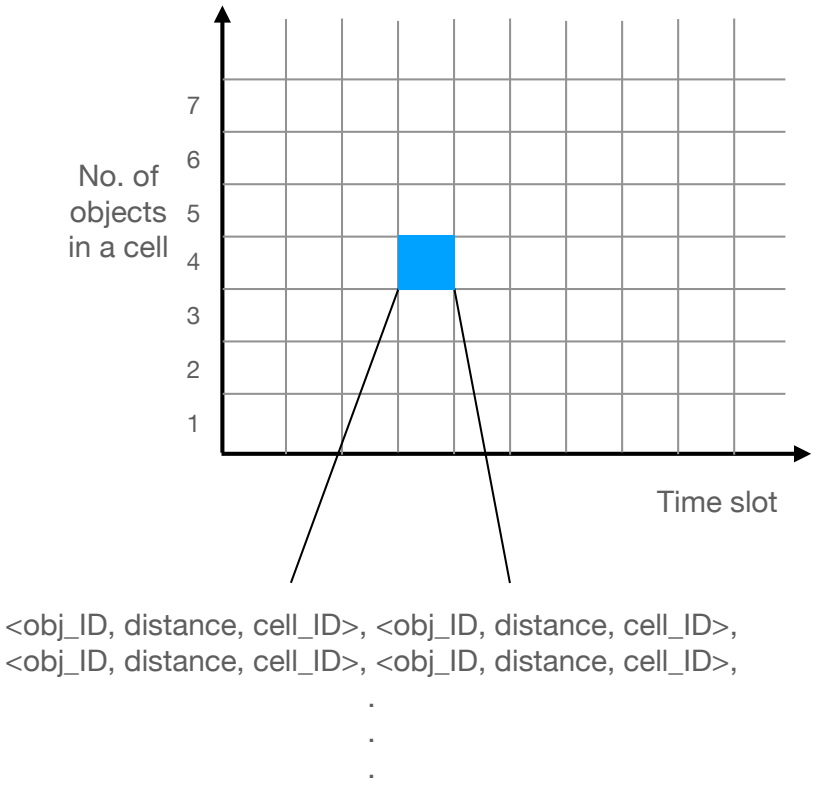


Figure 6.2: The offline method to compute shared dwell regions: In the preprocessing step, we create the shown 2-D grid. The x-axis shows the snapshots at which we maintain the state of object movements, the y-axis indexes the grid by number of objects present inside a cell. Each cell, in the grid stores a list of $\langle i, distance, c_{jk} \rangle$ tuples, where i is the ID of the object O_i in the trajectory dataset, *distance* is the distance of O_i from the center of c_{jk}.

only 7.3 seconds). However, archived trajectory datasets can be large, and as the number of movement events increases, the cost of applying events to the grid at query time can become prohibitively expensive.

To keep the problem tractable, we make the following simplifying assumptions- first, we place two explicit bounds on the dwell region queries we shall answer: we say that $D_{min} \leq d_q \leq D_{max}$, and that $T_{min} \leq t_q \leq T_{max}$, and that t_q is always a multiple of T'. These bounds are reasonable as they can be tuned according to the context and

the application that dwell region queries are being used for. Say, for a dataset containing human trajectories, T_{min} values of less than the order of a few seconds or T_{max} values of larger than a few hours are not likely to be of much practical use. Bounds on the d_q, the query radius for the shared dwell region, may be similarly justified. We also assume that the number of objects allowed to be within the same cell of the spatial grid is bounded by a constant C.

Preprocessing: We set T_{min} and T_{max} to the temporal extent of the considered dataset, i.e. let the first point update in the dataset be at time T_{min}, and the last update be at time T_{max}. We divide the time interval $[T_{min}, T_{max}]$ into $T+1$ time slots, and maintain the state of moving objects at T *snapshots* of time. We then set $T' = \frac{T_{max}-T_{min}}{T+1}$.

For each snapshot T, we maintain an array of size C, each cell of the array containing a list of 3-tuples $\langle i, distance, c_{jk} \rangle$, where i is the index of the moving object O_i, and *distance* is the euclidean distance from the center of cell c_{jk} to O_i. The collection of all such arrays gives us an index that can now be queried to determine the shared dwell regions.

To preprocess the trajectories in a given dataset, we iterate over all points in the given dataset, and for each point p, select the cells within the $2 * D_{max}$ neighborhood of p that are not in the $2 * D_{min}$ neighborhood. For all selected cells, we then- i) increase the counter ii) determine and add the $\langle i, distance, c_{jk} \rangle$ tuple to cell c_{jk}. Figure 6.2 illustrates the index built by the preprocessing method.

Query: When a query $\langle T_{start}, T_{end}, d_q, t_q, N \rangle$ is received, we can then answer it as follows: Since we know that t_q is always a multiple of T', we need to only aggregate the results from $\frac{t_q}{T'}$ snapshots. We start with the snapshot at T_{start}, and filter cells with counts of

objects $\geq N$. Then, we iterate over the lists maintained in all filtered cells, extracting the tuples with $distance \leq d_q$, and store them in a set F_{start}. Similarly, we extract the sets $F_{start+T'}, F_{start+2T'}, \ldots F_{end}$, and collect them into a set F'. Lastly, we extract the cell IDs in the tuples that are common to all sets in F', and return them as a result to the shared dwell region query.

6.4 Experiments

We implemented our algorithms in C++, and compiled using GCC 11.2 with all optimizations enabled. Our parallel segment tree implementation is taken from the Parallel Augmented Maps library [1]. We benchmarked our code on a machine with the 6-core Intel i5-8600K with 3.7 GHz base clock, and 192 KB of L1, 1.5 MB of L2 and 9 MB of L3 caches. The machine is equipped with 64 GBs of DDR4-2133 MHz, dual-channel RAM.

6.4.1 The Replay Method

To compare our methods with those of [160], we test our queries on the GeoLife dataset, containing 24 million locations of 8,970 trajectories collected from 182 users over a period of three years (April 2007 to August 2012) in Beijing, China [177].

Table 6.2 shows the mean running times of 3 shared dwell region queries with t_q= 30 minutes, $d_q = 1.2$ miles and randomly chosen $T = 24$ hours between the online method of [160] and our method. Note that the running times for the baseline method include only the time taken to return the dwell regions of individual trajectories, which we must then post-process to find the intersections between each pair of dwell regions. However, the time

[1] https://github.com/cmuparlay/pam

required to find such intersections is likely to be small, if for parity with our method, we assume that the dwell regions are found as the intersections of squares of sides $2 * d_q$, rather than circles of radius d_q.

6.4.2 The Offline Method

To compare our methods with the τ-index [161], we use the T-Drive dataset [173, 172]. The dataset contains trajectories of 10,357 taxis over the span of a week from February 2 to February 8, 2008 in Beijing, China; a total of 15 million GPS locations.

Table 6.3 compares the preprocessing times for τ-index built with different number of temporal partitions using linear partitioning with that of our method. We see that our preprocessing time increases with an increase in D_{max}. This is because for all point updates, each cell in the $2 * D_{max}$ neighborhood needs to be updated.

Finally, table 6.4 compares the query times of our method for different values of N against the τ-index. Note that using the τ-index does not return the exact dwell region, but only the center and radius of the smallest enclosing circle SEC_S, and the subsequence of points in the trajectory that form a dwell region. To determine the approximate dwell region, we must then draw the $2 * d_q$ neighborhoods of each point, and find the intersections of squares. However, the cost of such an operation is linear in the number of points that form the dwell region, which should be a small fraction of all points in the trajectory.

Table 6.2: Query times (in seconds) for the dwell regions with different values of k vs. the pointwise dwell regions queries on the GeoLife trajectory dataset. Note that our query times do not change, since k affects the quality of approximation only in the original algorithm [160]. The quality of our approximation depends only on the number of cells in the accumulation grid AG, which we set to the maximum value, 25000 × 25000 cells here. Each cell in the grid covers a 11.11m × 11.11m area, which suffices to capture the accuracy of GPS traces (5 ∼ 10 meters) in the GeoLife dataset.

T (hrs)	k	d_q (miles)	t_q (min)	Time (s)			Multithreaded Time (s)		
				Dwell Region	Pointwise	**Speedup**	Dwell Region	Pointwise	**Speedup**
24	4	1.2	30	302.67	311.06	**0.97**	58.94	69.04	**0.85**
24	6	1.2	30	428.96	311.06	**1.38**	84.24	69.04	**1.22**
24	8	1.2	30	631.18	311.06	**2.03**	123.565	69.04	**1.79**
24	10	1.2	30	704.673	311.06	**2.25**	137.582	69.04	**1.99**
24	12	1.2	30	901.41	311.06	**2.88**	176.160	69.04	**2.55**
24	14	1.2	30	1027.31	311.06	**3.30**	203.4	69.04	**2.94**

Table 6.3: Comparing the preprocessing times of our offline shared dwell regions method with the τ index on the T-Drive trajectory dataset containing 10,357 trajectories and 15 million points. For the τ index, we set $k = 8$, and used a linear partitioning scheme for the time dimension.

No. Partitions	Time(s)	D_{max}(m)	Time(s)
3	3300.590	275	4111.299
4	3336.064	550	5596.71
5	3544.470	825	6211.223
6	3674.585	1000	6998.672
7	3789.073	1100	7598.429

Table 6.4: Comparing the query times of our offline shared dwell regions method with the τ index on the T-Drive trajectory dataset. For the τ index, we set $k = 8$, and used a linear partitioning scheme for the time dimension. The query times shown are the averages for $d_q = \{250, 500, 1000, 1500, 1900\}$ meters and $t_q = \{10, 20, 30\}$ minutes.

No. Partitions	Query Time(s)	N	Query Time(s)
3	0.119	2	0.156
4	0.154	3	0.114
5	0.168	4	0.103
6	0.193		
7	0.196		

Chapter 7

Conclusions

This dissertation presented a set of techniques to address stochasticity in the energy consumptions and travel times for Electric Vehicle route planning.

7.1 Summary

In chapter 2, we presented *phases*, a simple new abstraction for modeling EV movement, and derived a mesoscopic energy estimation model that is tunable using a single parameter, τ. We showed that our model captures EV-specific complexities such as regenerative braking, and has an accuracy comparable to per-second models, despite operating at a lower temporal resolution. Further, we used Markov chains and kernel density estimation to learn the patterns of movement in historical data, using it to generate synthetic trips that capture the real-world variance in distance and energy consumption. Our models were evaluated using 52 hours of real-world driving data collected on a Nissan Leaf 2013 in Riverside, California.

Next, in chapter 3 we showed how to capture the tradeoffs between travel times and robustness of feasible routes against deviations in energy consumptions using the *Starting Charge Map* and *Buffer Map* constructs. The two constructs can be helpful in preventing EV users' range anxiety and enabling other use cases, such as minimizing trip time by charging more at the starting point. Computing both constructs involved extending the Shortest Feasible Path problem [19, 18] in two different ways, essentially increasing its output by another dimension. We proposed a simple approach, largely symmetric to the standard Charging Function Propagation (CFP) algorithm to compute the Starting Charge Maps. However, computing Buffer Maps required a more sophisticated approach, involving multiple runs of the CFP algorithm. We tested our methods on real-world road networks of Oregon and California, showing that while our approach for Starting Charge Maps could easily scale to large graphs, computing Buffer Maps required significantly more time.

In chapter 4, we studied EV routing when both energy consumptions and travel times are stochastic. The standard definition of *feasibility* of routes, however, does not apply to stochastic edge weights. Therefore, we defined the $\mathbb{E}$- and $p-$ feasibility criteria for stochastic EV route planning, and extended the CFP algorithm to permit stochastic edge weights. Our criteria allow drivers to maintain feasibility either *in expectation*, or setting explicit lower bounds on stranding probability. We also extend the multicriteria Contraction Hierarchies to accept stochastic edge weights, and evaluated our methods on a real-world road network of the Los Angeles Area. Our results showed that adding the feasibility criteria to standard stochastic route planning methods only results in a 30-40% slowdown, while offering strong guarantees against stranding, which can significantly improve experience

and reduce range anxiety for EV drivers. Further, our results indicate that the $\mathbb{E}$- and p- feasible routes tend to become more similar as the length of routes increases, while $\mathbb{E}$-feasible routes can be computed much faster than p-feasible routes. This result indicates that using *expected* values of energy consumptions might be sufficient for ensuring a high degree of confidence in route feasibility.

In chapter 5, we studied the effective application of well-known speedup techniques such as Contraction and Edge Hierarchies to routing with stochastic edge weights. We showed that the tradeoff between the accuracy, compactness and cost of convolution offered by different edge representations must be carefully managed to achieve fast query times for stochastic route planning. Then, we introduced the *tiering* technique that divides hierarchies into tiers of distinct edge weight representations to improve query times. Our experiments using a real-world traffic speeds dataset of the Los Angeles road network showed that the tiering technique works well for three different stochastic routing objectives, offering a speedup of as much as 3x for some queries.

Lastly, in chapter 6, we generalized the problem of identifying dwell regions for individual trajectories [160] to that of finding *shared* dwell regions between a set of trajectories. Prior methods of computing dwell regions maintain k heaps per object in the trajectory dataset, which can get quickly expensive. Therefore, we present two novel methods- an online, and an offline method, of answering the shared dwell region queries based on the Pointwise-Dense Regions approach of [110]. Our methods simulate the movement of trajectories in $\mathbb{R}^2$ on a grid of fixed size, keeping track of the cells in the grid that are in proximity of multiple objects, in order to answer queries quickly. Our methods are simple

to implement and easy to parallelize, and our results show that they can achieve a speedup of up to 3.3x over the baseline method on real world datasets.

7.2 Outlook

There are several directions for future research that may build upon the work presented in this thesis. For instance, all presented algorithms focus on point-to-point routes between a given source and destination. However, other applications such as routing electric trucks or delivery vehicles may seek to maximize other metrics such as goods delivered while maintaining $\mathbb{E}$- or p- feasibility along routes. For such applications, generalizations of the standard Vehicle Routing Problem with E- or p- feasibility may be developed, which may find several important real-world applications.

Standard EV routing algorithms such as the CFP have been shown to scale well to large graphs. Still, the number of labels generated for such methods remains exponential, making the memory consumption the major bottleneck in adapting such algorithms in real-world systems. More research may look into the problem of reducing the number of generated labels by relaxing the strict dominance criteria prevalent in current literature. However, relaxing the dominance criteria may result in a larger set of routes and slower queries, so the tradeoff between stringency of dominance criteria and time complexity and the memory requirements of EV route planning algorithms needs to better understood.

Lastly, while this thesis worked with the standard measure-theoretic probabilities, applying non-additive probabilities and Dempster-Shafer belief functions to capture the *lack* of available information about edge weights may be an interesting avenue for research.

Bibliography

[1] Kyoungho Ahn and Hesham Rakha. The effects of route choice decisions on vehicle energy consumption and emissions. *Transp. Res. Part D: Trans. Environ.*, 13(3):151–167, May 2008.

[2] Niklas Akerblöm, Yuxin Chen, and Morteza Haghir Chehreghani. An online learning framework for Energy-Efficient navigation of electric vehicles. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, pages 2051–2057, 2020.

[3] Yazan Al-Wreikat, Clara Serrano, and José Ricardo Sodré. Driving behaviour and trip condition effects on the energy consumption of an electric vehicle under real-world driving. *Appl. Energy*, 297:117096, September 2021.

[4] Saad Aljubayrin, Bin Yang, Christian S Jensen, and Rui Zhang. Finding non-dominated paths in uncertain road networks. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, number Article 15 in SIGSPATIAL '16, pages 1–10, New York, NY, USA, October 2016. Association for Computing Machinery.

[5] Alternative Fuels Data Center. Electric vehicle charging station locations. `https://afdc.energy.gov/fuels/electricity_locations.html`, 2021. Accessed: 2021-6-9.

[6] Georgi Andonov and Bin Yang. Stochastic shortest path finding in Path-Centric uncertain road networks. In *2018 19th IEEE International Conference on Mobile Data Management (MDM)*, pages 40–45, June 2018.

[7] Argonne National Labs. Autonomie. `https://www.autonomie.net/`. Accessed: 2019-10-15.

[8] Andreas Artmeier, Julian Haselmayr, Martin Leucker, and Martin Sachenbacher. The shortest path problem revisited: Optimal routing for electric vehicles. In *KI 2010: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 309–316. Springer, Berlin, Heidelberg, September 2010.

[9] Mohammad Asghari, Tobias Emrich, Ugur Demiryurek, and Cyrus Shahabi. Probabilistic estimation of link travel times in dynamic road networks. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '15, pages 47:1–47:10, New York, NY, USA, 2015. ACM.

[10] Joshua Auld, Michael Hope, Hubert Ley, Vadim Sokolov, Bo Xu, and Kuilin Zhang. POLARIS: Agent-based modeling framework development and implementation for integrated travel demand and network and operations simulations. *Transp. Res. Part C: Emerg. Technol.*, 64:101–116, March 2016.

[11] Joshua Auld, Omer Verbas, Mahmoud Javanmardi, and Aymeric Rousseau. Impact of Privately-Owned level 4 CAV technologies on travel demand and energy. *Procedia Comput. Sci.*, 130:914–919, 2018.

[12] Fouad Baouche, Rochdi Trigui, Nour Eddin El Faouzi, and Romain Billot. Energy consumption assessment for electric vehicles. In *International symposium on recent advances in transport modeling*, page 5 p., April 2013.

[13] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. In *Algorithm Engineering*, Lecture Notes in Computer Science, pages 19–80. Springer, Cham, 2016.

[14] Lucas S Batista, Felipe Campelo, Frederico G Guimarães, and Jaime A Ramírez. A comparison of dominance criteria in many-objective optimization problems. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 2359–2366, June 2011.

[15] R Bauer, D Delling, P Sanders, D Schieferdecker, and others. Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. *Algorithms*, 2008.

[16] Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. *Theor. Comput. Sci.*, 645:112–127, September 2016.

[17] Moritz Baum. *Engineering Route Planning Algorithms for Battery Electric Vehicles*. PhD thesis, KIT, 2018.

[18] Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf. Shortest feasible paths with charging stops for battery electric vehicles. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 44. ACM, November 2015.

[19] Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf. Shortest feasible paths with charging stops for battery electric vehicles. *Transportation Science*, 53(6):1627–1655, November 2019.

[20] Moritz Baum, Julian Dibbelt, Lorenz Hübschle-Schneider, Thomas Pajor, and Dorothea Wagner. Speed-Consumption Tradeoff for Electric Vehicle Route Planning. In Stefan Funke and Matúš Mihalák, editors, *14th Workshop on Algorithmic*

Approaches for Transportation Modelling, Optimization, and Systems, volume 42 of *OpenAccess Series in Informatics (OASIcs)*, pages 138–151, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[21] Moritz Baum, Julian Dibbelt, Lorenz Hübschle-Schneider, Thomas Pajor, and Dorothea Wagner. Speed-Consumption tradeoff for electric vehicle route planning. In \'u, editor, *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, OpenAccess Series in Informatics (OASIcs), pages 138–151. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014.

[22] Moritz Baum, Julian Dibbelt, Thomas Pajor, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Energy-Optimal routes for battery electric vehicles. *Algorithmica*, December 2019.

[23] Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Energy-optimal routes for electric vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 54–63. ACM, 5 November 2013.

[24] Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner, editors. *Dynamic Time-Dependent Route Planning in Road Networks with User Preferences*, volume 9685 of *Lecture Notes in Computer Science*. Springer International Publishing, Cham, 2016.

[25] Moritz Baum, Julian Dibbelt, Dorothea Wagner, and Tobias Zündorf. Modeling and engineering constrained shortest path algorithms for battery electric vehicles. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 87, 2017.

[26] Moritz Baum, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Consumption profiles in route planning for electric vehicles: Theory and applications. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 75, 2017.

[27] Moritz Baum, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Consumption profiles in route planning for electric vehicles: Theory and applications. In *16th International Symposium on Experimental Algorithms (SEA 2017)*. drops.dagstuhl.de, 2017.

[28] Roger L Berger. A nonparametric, intersection-union test for stochastic order. Technical report, North Carolina State University. Dept. of Statistics, 1986.

[29] Andrew C Berry. The accuracy of the gaussian approximation to the sum of independent variates. *Trans. Amer. Math. Soc.*, 49(1):122–122, jan 1941.

[30] Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16(3):580–595, August 1991.

[31] Johannes Blum and Sabine Storandt. Lower bounds and approximation algorithms for search space sizes in contraction hierarchies. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, August 2020.

[32] D P Bowyer, R Akçelik, and D C Biggs. Guide to fuel consumption analysis for urban traffic management. Technical report, Australian Road Research Board, 1985.

[33] Gail Helen Broadbent, Danielle Drozdzewski, and Graciela Metternicht. Electric vehicle adoption: An analysis of best practice and pitfalls for policy making from experiences of europe and the US. *Geography Compass*, 12(2):e12358, February 2018.

[34] Zhiguang Cao, Hongliang Guo, Jie Zhang, Dusit Niyato, and Ulrich Fastenrath. A data-driven method for stochastic shortest path problem. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1045–1052, October 2014.

[35] Lily Chai and Morgan Herlocker. Generating accurate speed estimations using aggregated telemetry data. (20190063939:A1), February 2019.

[36] N Chang, D Baek, and J Hong. Power consumption characterization, modeling and estimation of electric vehicles. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 175–182, November 2014.

[37] Anthony Chen and Zhaowang Ji. Path finding under uncertainty. *J. Adv. Transp.*, 39(1):19–37, September 2005.

[38] Bi Yu Chen, William H K Lam, Agachai Sumalee, Qingquan Li, and Mei Lam Tam. Reliable shortest path problems in stochastic Time-Dependent networks. *J. Intell. Transp. Syst.*, 18(2):177–189, April 2014.

[39] Xiao-Wei Chen, Bi Yu Chen, William H K Lam, Mei Lam Tam, and Wei Ma. A bi-objective reliable path-finding algorithm for battery electric vehicle routing. *Expert Syst. Appl.*, 182:115228, November 2021.

[40] Jian Dai, Bin Yang, Chenjuan Guo, Christian S Jensen, and Jilin Hu. Path cost distribution estimation using trajectory data. *Proceedings VLDB Endowment*, 10(3):85–96, November 2016.

[41] C De Cauwer, W Verbeke, J Van Mierlo, and T Coosemans. A model for range estimation and Energy-Efficient routing of electric vehicles in Real-World conditions. *IEEE Trans. Intell. Transp. Syst.*, pages 1–14, 2019.

[42] Cedric De Cauwer, Joeri Van Mierlo, and Thierry Coosemans. Energy consumption prediction for electric vehicles based on Real-World data. *Energies*, 8(8):8573–8593, August 2015.

[43] Cedric De Cauwer, Wouter Verbeke, Thierry Coosemans, Saphir Faid, and Joeri Van Mierlo. A Data-Driven method for energy consumption prediction and Energy-Efficient routing of electric vehicles in Real-World conditions. *Energies*, 10(5):608, May 2017.

[44] Daniel Delling, Andrew V Goldberg, Andreas Nowatzyk, and Renato F Werneck. PHAST: Hardware-accelerated shortest path trees. *J. Parallel Distrib. Comput.*, 73(7):940–952, July 2013.

[45] Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. Customizable route planning. In *Experimental Algorithms*, pages 376–387. Springer, Berlin, Heidelberg, May 2011.

[46] Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. Customizable route planning in road networks. *Transportation Science*, 51(2):566–591, May 2015.

[47] Daniel Delling and Dorothea Wagner. Time-Dependent route planning. In *Robust and Online Large-Scale Optimization*, Lecture Notes in Computer Science, pages 207–230. Springer, Berlin, Heidelberg, 2009.

[48] Ona Egbue and Suzanna Long. Barriers to widespread adoption of electric vehicles: An analysis of consumer attitudes and perceptions. *Energy Policy*, 48:717–729, September 2012.

[49] Matthias Eisel, Ilja Nastjuk, and Lutz M Kolbe. Understanding the influence of in-vehicle information systems on range stress – insights from an electric vehicle field experiment. *Transp. Res. Part F Traffic Psychol. Behav.*, 43:199–211, November 2016.

[50] Jochen Eisner, Stefan Funke, and Sabine Storandt. Optimal route planning for electric vehicles in large networks. In *AAAI*, 2011.

[51] Carl-Gustav Esseen. On the liapunoff limit of error in the theory of probability. *Ark. Mat. Astron. Fys.*, A28(9):1–19, 1942.

[52] Y Y Fan, R E Kalaba, and J E Moore. Arriving on time. *J. Optim. Theory Appl.*, 127(3):497–513, December 2005.

[53] C Fiori, V Marzano, V Punzo, and M Montanino. Energy consumption modeling in presence of uncertainty. *IEEE Trans. Intell. Transp. Syst.*, pages 1–12, 2020.

[54] Chiara Fiori, Kyoungho Ahn, and Hesham A Rakha. Power-based electric vehicle energy consumption model: Model development and validation. *Appl. Energy*, 168:257–268, April 2016.

[55] Matthew William Fontana. *Optimal routes for electric vehicles facing uncertainty, congestion, and energy constraints*. PhD thesis, Massachusetts Institute of Technology, 2013.

[56] Luca Foschini, John Hershberger, and Subhash Suri. On the complexity of Time-Dependent shortest paths. *Algorithmica*, 68(4):1075–1097, April 2014.

[57] Mogens Fosgerau. The valuation of travel time variability. Technical report, International Transport Forum, 2016.

[58] H Frank. Shortest paths in probabilistic graphs. *Oper. Res.*, 17(4):583–599, 1969.

[59] Thomas Franke and Josef F Krems. Interacting with limited mobility resources: Psychological range levels in electric vehicle use. *Transp. Res. Part A: Policy Pract.*, 48:109–122, February 2013.

[60] Thomas Franke and Josef F Krems. What drives range preferences in electric vehicle users? *Transp. Policy*, 30:56–62, November 2013.

[61] Thomas Franke, Isabel Neumann, Franziska Bühler, Peter Cocron, and Josef F Krems. Experiencing range in an electric vehicle: Understanding psychological barriers: Experiencing range. *Appl. Psychol.*, 61(3):368–391, July 2012.

[62] Thomas Franke, Nadine Rauh, Madlen Günther, Maria Trantow, and Josef F Krems. Which factors can protect against range stress in everyday usage of battery electric vehicles? toward enhancing sustainability of electric mobility systems. *Hum. Factors*, 58(1):13–26, February 2016.

[63] Stefan Funke and Sabine Storandt. Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pages 41–54, Philadelphia, PA, USA, 2013. Society for Industrial and Applied Mathematics.

[64] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Experimental Algorithms*, pages 319–333. Springer, Berlin, Heidelberg, May 2008.

[65] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

[66] Konstantinos N Genikomsakis and Georgios Mitrentsis. A computationally efficient simulation model for estimating energy consumption of electric vehicles in the context of route planning applications. *Transp. Res. Part D: Trans. Environ.*, 50(Supplement C):98–118, January 2017.

[67] Michael T Goodrich and Paweł Pszona. Two-Phase bicriterion search for finding fast and efficient electric vehicle routes. 10 September 2014.

[68] Michael T Goodrich and Paweł Pszona. Two-phase bicriterion search for finding fast and efficient electric vehicle routes. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 193–202. ACM, November 2014.

[69] Michael Gowanlock and Henri Casanova. In-Memory distance threshold similarity searches on moving object trajectories. *International Journal on Advances in Software*, 2014.

[70] Chenjuan Guo, Yu Ma, Bin Yang, Christian S Jensen, and Manohar Kaul. EcoMark: Evaluating models of vehicular environmental impact. In *Proc. 20th Intl. Conference*

on Advances in Geographic Information Systems, SIGSPATIAL'12, pages 269–278, New York, NY, USA, 2012. ACM.

[71] Chenjuan Guo, Bin Yang, Ove Andersen, Christian S Jensen, and Kristian Torp. EcoMark 2.0: empowering eco-routing with vehicular environmental models and actual vehicle fuel consumption data. *Geoinformatica*, 19(3):567–599, July 2015.

[72] E Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *J. Reine Angew. Math.*, 1909(136):210–271, July 1909.

[73] Demian Hespe and Peter Sanders. More hierarchy in route planning using edge hierarchies. In *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, OpenAccess Series in Informatics (OASIcs), pages 10:1–10:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany.

[74] Jilin Hu, Bin Yang, Chenjuan Guo, and Christian S Jensen. Risk-aware path selection with time-varying, uncertain travel costs: a time series approach. *VLDB J.*, 27(2):179–200, April 2018.

[75] Ming Hua and Jian Pei. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 347–358, New York, NY, USA, March 2010. Association for Computing Machinery.

[76] G Huber, K Bogenberger, and H van Lint. Optimization of charging strategies for battery electric vehicles under uncertainty. *IEEE Trans. Intell. Transp. Syst.*, pages 1–17, 2020.

[77] Ehsan Jafari and Stephen D Boyles. Multicriteria stochastic shortest path problem for electric vehicles. *Networks Spat. Econ.*, 17(3):1043–1070, September 2017.

[78] Tobias Skovgaard Jepsen, Christian S Jensen, and Thomas Dyhre Nielsen. UniTE – the best of both worlds: Unifying Function-Fitting and Aggregation-Based approaches to travel time and travel speed estimation. April 2021.

[79] H C Joksch. The shortest route problem with constraints. *J. Math. Anal. Appl.*, 14(2):191–197, May 1966.

[80] Malte F Jung, David Sirkin, Turgut M Gür, and Martin Steinert. Displayed uncertainty improves driving experience and behavior: The case of range anxiety in an electric car. In *Proc. 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2201–2210. ACM, April 2015.

[81] J Kang, T Ma, F Ma, and J Huang. Link-based emission model for eco routing. In *2011 11th International Conference on ITS Telecommunications*, pages 207–212, August 2011.

[82] Dominik Karbowski, Aymeric Rousseau, Vivien Smis-Michel, and Valentin Vermeulen. Trip prediction using GIS for vehicle energy efficiency. 2014.

[83] Johannes Kester. Security in transition(s): The low-level security politics of electric vehicle range anxiety. *Security Dialogue*, 50(6):547–563, December 2019.

[84] Moritz Kobitzsch, Samitha Samaranayake, and Dennis Schieferdecker. Pruning techniques for the stochastic on-time arrival problem - an experimental study. July 2014.

[85] Adrian Kosowski and Laurent Viennot. Beyond highway dimension: Small distance labels using tree skeletons. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1462–1478, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics.

[86] Johannes Geir Kristinsson, Ryan Abraham McGee, Anthony Mark Phillips, Ming Lang Kuang, Wenduo Wang, Jungme Park, Yi Murphey, and Chen Fang. Vehicle speed profile prediction using neural networks. (20150344036:A1), December 2015.

[87] Benjamin Krogh, Ove Andersen, and Kristian Torp. Analyzing electric vehicle energy consumption using very large data sets. In *Database Systems for Advanced Applications*, pages 471–487. Springer International Publishing, 2015.

[88] S Kullback and R A Leibler. On information and sufficiency. *aoms*, 22(1):79–86, March 1951.

[89] Solomon Kullback. *Information Theory and Statistics*. Wiley, 1959.

[90] James Larminie and John Lowry. *Electric Vehicle Technology Explained*. John Wiley & Sons, September 2012.

[91] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, number Article 34 in GIS '08, pages 1–10, New York, NY, USA, November 2008. Association for Computing Machinery.

[92] Weixia Li, Guoyuan Wu, Yi Zhang, and Matthew J Barth. A comparative study on data segregation for mesoscopic energy modeling. *Transp. Res. Part D: Trans. Environ.*, 50:70–82, January 2017.

[93] Yan Li, Pratik Kotwal, Pengyue Wang, Yiqun Xie, Shashi Shekhar, and William Northrop. Physics-guided energy-efficient path selection using on-board diagnostics data. *ACM/IMS Trans. Data Sci.*, 1(3):1–28, September 2020.

[94] Yan Li, Shashi Shekhar, Pengyue Wang, and William Northrop. Physics-guided energy-efficient path selection: a summary of results. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 99–108. ACM, November 2018.

[95] S Lim, C Sommer, E Nikolova, and others. Practical route planning under delay uncertainty: Stochastic shortest path queries. *Robot. Sci. Syst.*, 2013.

[96] Antonio Lima, Rade Stanojevic, Dina Papagiannaki, Pablo Rodriguez, and Marta C González. Understanding individual routing behaviour. *J. R. Soc. Interface*, 13(116), March 2016.

[97] Ernesto Queirós Vieira Martins. On a multicriteria shortest path problem. *Eur. J. Oper. Res.*, 16(2):236–245, May 1984.

[98] Michail Masikos, Konstantinos Demestichas, Evgenia Adamopoulou, and Michael Theologou. Energy-efficient routing based on vehicular consumption predictions of a mesoscopic learning model. *Appl. Soft Comput.*, 28:114–124, March 2015.

[99] Sören Merting, Christian Schwan, and Martin Strehler. Routing of electric vehicles: Constrained shortest path problems with resource recovering nodes. In *OASIcs-OpenAccess Series in Informatics*, volume 48. Schloss Dagstuhl - Leibniz-Zentrum für Informatik GmbH, Wadern/Saarbrücken, Germany, 2015.

[100] Sebastiano Milardo, Punit Rathore, Marco Amorim, Umberto Fugiglando, Paolo Santi, and Carlo Ratti. Understanding drivers' stress and interactions with vehicle systems through naturalistic data analysis. *IEEE Trans. Intell. Transp. Syst.*, pages 1–12, 2021.

[101] Elise D Miller-Hooks and Hani S Mahmassani. Least expected time paths in stochastic, Time-Varying transportation networks. *Transportation Science*, 34(2):198–215, May 2000.

[102] Elise Deborah Miller-Hooks. *Optimal routing in time-varying, stochastic networks: Algorithms and implementations*. PhD thesis, The University of Texas at Austin, 1997.

[103] Nikola Milosavljević. On optimal preprocessing for contraction hierarchies. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '12, pages 33–38, New York, NY, USA, November 2012. Association for Computing Machinery.

[104] Ian S Morrison and Michael G Gowanlock. Extending galactic habitable zone modeling to include the emergence of intelligent life. *Astrobiology*, 15(8):683–696, August 2015.

[105] Ladan Mozaffari, Ahmad Mozaffari, and Nasser L Azad. Vehicle speed prediction via a sliding-window time series analysis and an evolutionary least learning machine: A case study on san francisco urban roads. *Engineering Science and Technology, an International Journal*, 18(2):150–162, June 2015.

[106] J P L Nasa. NASADEM merged DEM global 1 arc second V001, 2020.

[107] National Renewable Energy Labs. FASTSim: Future automotive systems technology simulator. `https://www.nrel.gov/transportation/fastsim.html`. Accessed: 2019-10-15.

[108] Myriam Neaimeh, Graeme A Hill, Yvonne Hübner, and Phil T Blythe. Routing systems to extend the driving range of electric vehicles. *IET Intel. Transport Syst.*, 7(3):327–336, September 2013.

[109] Zachary A Needell, James McNerney, Michael T Chang, and Jessika E Trancik. Potential for widespread electrification of personal vehicle travel in the united states. *Nature Energy*, 1:16112, August 2016.

[110] Jinfeng Ni and Chinya V Ravishankar. Pointwise-Dense region queries in spatio-temporal databases. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 1066–1075, April 2007.

[111] Yu (marco) Nie and Xing Wu. Shortest path problem considering on-time arrival probability. *Trans. Res. Part B: Methodol.*, 43(6):597–613, July 2009.

[112] Patrick Niklaus. A unified framework for electric vehicle routing. Master's thesis, KIT, November 2017.

[113] Mehrdad Niknami and Samitha Samaranayake. Tractable pathfinding for the stochastic On-Time arrival problem. In *Experimental Algorithms*, pages 231–245. Springer International Publishing, 2016.

[114] Evdokia Nikolova. Approximation algorithms for reliable stochastic combinatorial optimization. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 338–351. Springer Berlin Heidelberg, 2010.

[115] Evdokia Nikolova, M Brand, and David R Karger. Optimal route planning under uncertainty. *ICAPS*, 2006.

[116] Evdokia Nikolova, Jonathan A Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *Algorithms – ESA 2006*, pages 552–563. Springer Berlin Heidelberg, 2006.

[117] J Park, D Li, Y L Murphey, J Kristinsson, R McGee, M Kuang, and T Phillips. Real time vehicle speed prediction using a neural network traffic model. In *The 2011 International Joint Conference on Neural Networks*, pages 2991–2996, July 2011.

[118] Jungme Park, Yi Lu Murphey, Ryan McGee, Jóhannes G Kristinsson, Ming L Kuang, and Anthony M Phillips. Intelligent trip modeling for the prediction of an origin–destination traveling speed profile. *IEEE Trans. Intell. Transp. Syst.*, 15(3):1039–1053, 2014.

[119] Axel Parmentier and Frédéric Meunier. Stochastic shortest paths and risk measures. August 2014.

[120] Simon Aagaard Pedersen, Bin Yang, and Christian S Jensen. Fast stochastic routing under time-varying uncertainty. *VLDB J.*, October 2019.

[121] Simon Aagaard Pedersen, Bin Yang, and Christian S Jensen. Anytime stochastic routing with hybrid learning. *Proceedings VLDB Endowment*, 13(9):1555–1567, May 2020.

[122] Simon Aagaard Pedersen, Bin Yang, and Christian S Jensen. A hybrid learning approach to stochastic routing. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1910–1913, April 2020.

[123] Rafael Pérez-Torres, César Torres-Huitzil, and Hiram Galeana-Zapién. Full On-Device stay points detection in smartphones for Location-Based mobile applications. *Sensors*, 16(10), October 2016.

[124] D Pevec, J Babic, A Carvalho, Y Ghiassi-Farrokhfal, W Ketter, and V Podobnik. Electric vehicle range anxiety: An obstacle for the personal transportation (r)evolution? In *2019 4th International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–8, June 2019.

[125] Dario Pevec, Jurica Babic, Arthur Carvalho, Yashar Ghiassi-Farrokhfal, Wolfgang Ketter, and Vedran Podobnik. A survey-based assessment of how existing and potential electric vehicle owners perceive range anxiety. *J. Clean. Prod.*, 276:122779, December 2020.

[126] M S Pinsker. *Information and Information Stability of Random Variables and Processes*. Holden-Day, 1964.

[127] Claudius Proissl and Tobias Rupp. On the difference between search space size and query complexity in contraction hierarchies. In *Proceedings of the 2021 SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*, Proceedings, pages 77–87. Society for Industrial and Applied Mathematics, January 2021.

[128] Xuewei Qi, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew J Barth. Data-driven decomposition analysis and estimation of link-level electric vehicle energy consumption under real-world traffic conditions. *Transp. Res. Part D: Trans. Environ.*, August 2017.

[129] Xuewei Qi, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew J Barth. Data-driven decomposition analysis and estimation of link-level electric vehicle energy consumption under real-world traffic conditions. *Transp. Res. Part D: Trans. Environ.*, 64:36–52, October 2018.

[130] Payas Rajan, Moritz Baum, Michael Wegner, Tobias Zündorf, Christian J West, Dennis Schieferdecker, and Daniel Delling. Robustness generalizations of the shortest feasible path problem for electric vehicles. In Matthias And Federico, Müller-Hannemann, editor, *Proceedings of 21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021)*, Open Access Series in Informatics (OASIcs), pages 11:1–11:18, Dagstuhl, Germany, September 2021. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[131] Payas Rajan and Chinya V Ravishankar. The phase abstraction for estimating energy consumption and travel times for electric vehicle route planning. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '19, pages 556–559, New York, NY, USA, 2019. ACM.

[132] Payas Rajan and Chinya V Ravishankar. Tiering in contraction and edge hierarchies for stochastic route planning. In *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '21, pages 616–625, New York, NY, USA, November 2021. Association for Computing Machinery.

[133] Nadine Rauh, Thomas Franke, and Josef F Krems. User experience with electric vehicles while driving in a critical range situation – a qualitative approach. *IET Intel. Transport Syst.*, 9(7):734–739, July 2015.

[134] Byron Reeves and Clifford Ivar Nass. *The media equation: How people treat computers, television, and new media like real people and places*. Cambridge University Press New York, NY, USA, 1996.

[135] Zeinab Rezvani, Johan Jansson, and Jan Bodin. Advances in consumer electric vehicle adoption research: A review and research agenda. *Transp. Res. Part D: Trans. Environ.*, 34:122–136, January 2015.

[136] Matthias Ruß, Gunther Gust, and Dirk Neumann. The constrained reliable shortest path problem in stochastic Time-Dependent networks. *Oper. Res.*, 69(3):709–726, May 2021.

[137] G Sabran, S Samaranayake, and A Bayen. Precomputation techniques for the stochastic on-time arrival problem. In *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, Proceedings, pages 138–146. Society for Industrial and Applied Mathematics, December 2013.

[138] Guillaume Sabran, Samitha Samaranayake, and Alexandre Bayen. Precomputation techniques for the stochastic on-time arrival problem. In *2014 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, Proceedings, pages 138–146. Society for Industrial and Applied Mathematics, December 2013.

[139] M Sachenbacher, M Leucker, A Artmeier, and J Haselmayr. Efficient Energy-Optimal routing for electric vehicles. *AAAI*, 2011.

[140] S Sautermeister, M Falk, B Bäker, F Gauterin, and M Vaillant. Influence of measurement and prediction uncertainties on range estimation for electric vehicles. *IEEE Trans. Intell. Transp. Syst.*, 19(8):2615–2626, August 2018.

[141] René Schönfelder and Martin Leucker. Abstract routing models and abstractions in the context of vehicle routing. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, June 2015.

[142] Moshe Shaked and J George Shanthikumar. *Stochastic Orders*. Springer New York, October 2006.

[143] R Shankar and J Marco. Method for estimating the energy consumption of electric vehicles and plug-in hybrid electric vehicles under real-world driving conditions. *IET Intel. Transport Syst.*, 7(1):138–150, March 2013.

[144] Liang Shen, Hu Shao, Ting Wu, William H K Lam, and Emily C Zhu. An energy-efficient reliable path finding algorithm for stochastic road networks with electric vehicles. *Transp. Res. Part C: Emerg. Technol.*, 102:450–473, May 2019.

[145] Luou Shen. *Freeway Travel Time Estimation and Prediction Using Dynamic Neural Networks*. PhD thesis, Florida International University, July 2008.

[146] I G Shevtsova. An improvement of convergence rate estimates in the lyapunov theorem. *Dokl. Math.*, 82(3):862–864, December 2010.

[147] Jaewook Shin and Myoungho Sunwoo. Vehicle speed prediction using a markov chain with speed constraints. *IEEE Trans. on Intell. Transp. Syst.*, November 2018.

[148] Kenneth A Small, Clifford Winston, and Jia Yan. Uncovering the distribution of motorists' preferences for travel time and reliability. *Econometrica*, 73(4):1367–1382, July 2005.

[149] Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys (CSUR)*, 46(4):45, April 2014.

[150] S Storandt and S Funke. Enabling E-Mobility: Facility location for battery loading stations. *AAAI*, 2013.

[151] Sabine Storandt. Quick and energy-efficient routes: Computing constrained shortest paths for electric vehicles. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '12, pages 20–25, New York, NY, USA, 2012. ACM.

[152] Sabine Storandt. Quick and energy-efficient routes: Computing constrained shortest paths for electric vehicles. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '12, pages 20–25, New York, NY, USA, 2012. ACM.

[153] Sabine Storandt. Route planning for bicycles — exact constrained shortest paths made practical via contraction hierarchy. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[154] Sabine Storandt and Stefan Funke. Cruising with a Battery-Powered vehicle and not getting stranded. In *AAAI*, volume 3, page 46, 2012.

[155] Martin Strehler, Sören Merting, and Christian Schwan. Energy-efficient shortest routes for electric and hybrid vehicles. *Trans. Res. Part B: Methodol.*, 103(Supplement C):111–135, 1 September 2017.

[156] Yihan Sun, Daniel Ferizovic, and Guy E Belloch. PAM: parallel augmented maps. *SIGPLAN Notices*, 53(1):290–304, February 2018.

[157] Tesla. *Model S owner's manual*, October 2019.

[158] L Thibault, G De Nunzio, and A Sciarretta. A unified approach for electric vehicles range maximization via eco-routing, eco-driving, and energy consumption prediction. *IEEE Transactions on Intelligent Vehicles*, pages 1–1, 2018.

[159] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, November 2004.

[160] M R Uddin, C V Ravishankar, and V J Tsotras. Online identification of dwell regions for moving objects. In *2012 IEEE 13th International Conference on Mobile Data Management*, pages 248–257, July 2012.

[161] Reaz Uddin, Mehnaz Tabassum Mahin, Payas Rajan, Chinya V Ravishankar, and Vassilis J Tsotras. Dwell regions: Generalized stay regions for streaming and archival trajectory data. Under Review.

[162] Toshiaki Uemura. Pre-Estimation of electric vehicle energy consumption on unfamiliar roads and actual driving experiments. In *Proceedings of the VLDB 2019 PhD Workshop*, 2019.

[163] Billy M Williams and Lester A Hoel. Modeling and forecasting vehicular traffic flow as a seasonal ARIMA process: Theoretical basis and empirical results. *J. Transp. Eng.*, 129(6):664–672, November 2003.

[164] Christoph Witzgall and Alan J Goldman. Most profitable routing before maintenance. In *OPERATIONS RESEARCH*, page B82. . . . RD, STE 400, LINTHICUM HTS, MD . . . , 1965.

[165] Xinkai Wu, David Freese, Alfredo Cabrera, and William A Kitch. Electric vehicles' energy consumption measurement and estimation. *Transp. Res. Part D: Trans. Environ.*, 34(Supplement C):52–67, January 2015.

[166] Chi Xie and Nan Jiang. Relay requirement and traffic assignment of electric vehicles. *Computer-Aided Civil and Infrastructure Engineering*, 31(8):580–598, August 2016.

[167] B Yang, C Guo, C S Jensen, M Kaul, and S Shang. Stochastic skyline route planning under time-varying uncertainty. In *2014 IEEE 30th International Conference on Data Engineering*, pages 136–147, March 2014.

[168] Bin Yang, Jian Dai, Chenjuan Guo, Christian S Jensen, and Jilin Hu. PACE: a PAth-CEntric paradigm for stochastic path finding. *VLDB J.*, 27(2):153–178, April 2018.

[169] Bin Yang, Jian Dai, Chenjuan Guo, Christian S Jensen, and Jilin Hu. PACE: a PAth-CEntric paradigm for stochastic path finding. *VLDB J.*, 27(2):153–178, April 2018.

[170] Enjian Yao, Zhiqiang Yang, Yuanyuan Song, and Ting Zuo. Comparison of electric vehicle's energy consumption factors for different road types. *Discrete Dyn. Nat. Soc.*, 2013, December 2013.

[171] Fei Ye, Guoyuan Wu, K Boriboonsomsin, and M J Barth. A hybrid approach to estimating electric vehicle energy consumption for ecodriving applications. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 719–724, November 2016.

[172] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 316–324, New York, NY, USA, August 2011. Association for Computing Machinery.

[173] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 99–108, New York, NY, USA, November 2010. Association for Computing Machinery.

[174] Quan Yuan, Wei Hao, Haotian Su, Guanwen Bing, Xinyuan Gui, and Abolfazl Safikhani. Investigation on range anxiety and safety buffer of battery electric vehicle drivers. *Journal of Advanced Transportation*, 2018, June 2018.

[175] Mingxuan Yue, Yaguang Li, Haoze Yang, Ritesh Ahuja, Yao-Yi Chiang, and Cyrus Shahabi. DETECT: Deep trajectory clustering for Mobility-Behavior analysis. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 988–997, December 2019.

[176] Jie Zheng, Ling Wang, Shengyao Wang, Yile Liang, and Jize Pan. Solving two-stage stochastic route-planning problem in milliseconds via end-to-end deep learning. *Complex & Intelligent Systems*, 7(3):1207–1222, June 2021.

[177] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 791–800, New York, NY, USA, April 2009. Association for Computing Machinery.

[178] Shanjiang Zhu and David Levinson. Do people use the shortest path? an empirical test of wardrop's first principle. *PLoS One*, 10(8):e0134322, August 2015.

[179] Weiwei Zhuang, Yadong Li, and Guoxin Qiu. Statistical inference for a relaxation index of stochastic dominance under density ratio model. *J. Appl. Stat.*, pages 1–19, August 2021.

[180] A K Ziliaskopoulos and H S Mahmassani. A Time-Dependent shortest path algorithm for Real-Time intelligent Vehicle/Highway system. *Transportation Research Record Journal of the Transportation Research Board*, (1408):94–100, January 1993.