The Pit Stop Problem: How to Plan Your Next Road Trip

Sreenivas Gollapudi
Google Research
sgollapu@google.com

Kostas Kollias
Google Research
kostaskollias@google.com

Debmalya Panigrahi
Duke University
debmalya@cs.duke.edu

ABSTRACT

Many online trip planning and navigation software need to routinely solve the problem of deciding where to take stops during a journey for various services such as refueling (or EV charging), rest stops, food, etc. The goal is to minimize the overhead of these stops while ensuring that the traveller is not starved of any essential resource (such as fuel, rest, or food) during the journey. In this paper, we formally model this problem and call it the *pit stop* problem. We design algorithms for this problem under various settings: single vs multiple types of stops, and offline vs online optimization (i.e., in advance of or during the trip). Our algorithms achieve provable guarantees in terms of approximating the optimal solution. We then extensively evaluate our algorithms on real world data and demonstrate that they significantly outperform baseline solutions.

CCS CONCEPTS

• **Theory of computation → Online algorithms**; • **Information systems → Geographic information systems**.

KEYWORDS

Online algorithms, electric vehicle routing

ACM Reference Format:
Sreenivas Gollapudi, Kostas Kollias, and Debmalya Panigrahi. 2022. The Pit Stop Problem: How to Plan Your Next Road Trip. In *The 30th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '22), November 1–4, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/https://doi.org/10.1145/3557915.3560982

1 INTRODUCTION

Trip planning and navigation software are an important component of the bouquet of services offered to online users today. Perhaps the most visible use of such software is in providing driving plans for personal mobility such as in planning road trips (e.g., Google maps, Apple maps, Waze, etc.). But, they have also played a crucial role in the rapid growth of many other services on the Internet. For instance, ride sharing services (such as Uber, Lyft, etc.) rely extensively on dynamic trip planning to optimize QoS and revenue, autonomous vehicles increasingly used in logistics rely on automated decision making for optimizing trip times, the online travel and tourism industry uses navigation and planning software to design vacation plans, increased adoption of electric vehicles (EVs) has been facilitated by software support for charging schedules during long trips, etc. This increasing variety of application domains have given rise to many more uses of online maps beyond the traditional point-to-point navigation and guidance. Further, these applications enable different levels of interactions with the navigation software. Examples include mobile devices used for advance or dynamic trip planning (e.g., Google Maps, Tom Tom, Apple Maps, Bing Maps), in-vehicle interaction both using software built into vehicles (particularly by high end car manufacturers like Tesla) and using third party software provided via mobile devices (e.g., Android Auto, Apple CarPlay), and the burgeoning autonomous vehicles in the logistics industry that are controlled by the routing software in a tightly integrated setting [11].

A central problem for these services is to plan stops along a trip route to avail services such as refueling, food, rest stops, facilities breaks, truck weighing stations, etc. We call the problem of planning these stops in an optimal manner the *pit stop problem*. In some cases, these pit stops can be decided in advance of the trip, and are incorporated into route planning itself. In practice, however, most travellers do not plan out a trip at a level of detail that includes the precise schedule of stops beforehand. Indeed, such planning might even be impossible because of the uncertainties associated with traffic delays, rest area or weighing station closings, facilities hours, etc. that are difficult to predict in advance. In such cases, one needs a more adaptive strategy where the navigation software plans pit stops along the route during the journey itself, while dynamically adapting to the current road conditions, facility closures, etc. Whether planned in advance or during the journey, the goal is to minimize the overhead incurred by these pit stops, while ensuring that the traveller is not starved of any essential resource such as food, fuel, or rest during the journey.

In this paper, we model and study the pit stop problem from an algorithmic perspective. We consider both the *offline* setting, where the stops are planned before the journey commences, and the *online* problem where the stops are decided during the journey. In the offline case, we provide an **optimal** algorithm using a dynamic programming approach. In the online setting, we design a novel algorithm that only uses *local* information about resource availability in close proximity to the current location of the traveller to decide her next pit stop. We give provable guarantees on the performance of this algorithm, namely that it achieves a **constant approximation** of an (offline) optimal solution. In addition to the theoretical guarantees, we evaluate our algorithms on real data sets for EV charging and food stops, and show that our algorithms improve on benchmark greedy solutions for these applications.

1.1 Our Contributions

Our first contribution is in formally modeling the pit stop problem as an algorithmic question in both the offline and online settings. Recall that the offline setting refers to planning the stops in advance before the commencement of the journey. In contrast, the online

SIGSPATIAL '22, November 1–4, 2022, Seattle, WA, USA

ACM ISBN 978-1-4503-9529-8/22/11.
https://doi.org/https://doi.org/10.1145/3557915.3560982

setting is more flexible and plans stops during the journey itself, allowing it to adapt to dynamic variables such as traffic conditions and facilities closings. In addition to offline vs online, a different dimension for categorizing the pit stop problem is whether the stops are homogeneous (i.e., provide the same resource) or heterogeneous (i.e. provide a variety of resources). Some navigation software provide planning for individual services separately; e.g., many electric vehicles (EV) have inbuilt software to plan charging stops along a route (e.g., Tesla, Volvo). On the other hand, some services provide generic tools for trip planning involving a heterogeneous set of pit stops to avail facilities such as food, fuel, and others[1]. To distinguish between these cases, we consider two versions of the pit stop problem, a single resource and a multi-resource variant.

Our algorithmic results are the following:

- For the offline pit stop problem, we give an **optimal** algorithm. Interestingly, we show that this algorithm can also be incorporated into route planning. I.e., given a graph representing the road network, a source-destination pair, and the map of resource locations, the algorithm finds a route from the source to the destination along with pit stops on this route that minimizes the total length of the journey.
- For the online pit stop problem, we give an algorithm that achieves a **constant approximation** of the optimal solution for a single resource. Unfortunately, this algorithm does not generalize to multiple resources; indeed, we give a lower bound showing that a bounded approximation factor cannot be obtained for the online problem even with just two resources. Therefore, we design heuristics for solving the online pit stop problem in the multi-resource setting.

Next, we empirically evaluate our algorithms on real world maps, and compare them to baseline greedy solutions. We focus on road trips with an electric vehicle (EV) and consider both a single resource (battery charging) and multiple resources (battery and food) in our experiments. Our experiments show that our algorithms consistently outperform greedy baseline solutions for both settings.

1.2 Related Work

Our problem can be broadly classified as a vehicle routing problem (VRP), an area that has been widely studied in the operations research and approximation algorithms communities (see [15, 32] for books on VRP in operations research; in approximation algorithms, some of the well-studied variants include orienteering [3, 5, 8], dial-a-ride [7, 16, 17], and capacitated VRP [6, 18, 25]). In the offline setting, our algorithms are inspired by classical DP solutions to shortest path problems, such as Dijkstra's and Bellman-Ford's algorithms (e.g., [10]). The work in [21] solves the *gas station problem*, which is a special case of the pit stop problem, specifically offline with a single resource. In the online setting, perhaps the closest to our problem is the so called *Canadian Traveler Problem* (CTP) where the goal is to find shortest path solutions in a graph that is revealed in parts. The complexity of algorithms for CTP have been studied in [4, 28, 29] and branch and bound type solutions explored in [1, 13]. The main point of difference between these problems and the pit stop problem is the constraint that the route must ensure the traveller does not run out of any resource during the journey. We are not aware of previous work in the shortest path literature under such "periodic replenishment" constraints.

The online pit stop problem also falls under the general umbrella of decision-making under uncertainty. Indeed, our problem can be thought of as a complement to the classic secretary problem (e.g., [12, 14, 22, 27]) and prophet inequalities (e.g. [2, 19, 20, 23, 24, 31]). In fact, while the latter problems put an upper bound on the number of selected items with the values of future items being uncertain, the online pit stop problem does exactly the opposite, i.e., puts a lower bound on the number of stops where the cost of future stops is uncertain. Nevertheless, what again distinguishes the pit stop problem is the periodicity of the constraint, i.e., that the number of stops is not simply lower bounded by a minimum threshold but that there is a limit to the gap between two consecutive stops. This makes our problem unique compared to the existing literature, and requires new tools that we explore in this paper.

On the more practical side, there has been work that studies the problem of minimizing charging or refuelling cost of electric vehicles or vehicles in general. Lin et. al. in [9, 26] considered the problem of minimizing energy and travel cost for a fleet of electric vehicles. Other works [30, 33] focused on optimizing the use of electric charging stations via routing vehicles to the appropriate station. Unlike our work, the above papers do not provide provable guarantees for the algorithms they propose, and do not consider online versions of their problems.

2 MODEL

We first consider the single resource variant of the pit stop problem. We use the refueling application to describe the problem, although it applies equally well for any other service being availed during the trip. The goal of the problem is to select refueling stops along a prescribed route. We denote the start of the route by location 0 and the destination by location t. (The route may be defined in units of time rather than distance, or in terms of some other parameter. This detail is not relevant to the problem formulation itself, and is hence omitted.) There are n candidate stops along the route. Each stop v is at location $v \in (0, t)$ along the route. The cost of the stop is denoted c_v, and represents the overhead of making this stop. We denote by l_v the fuel level of the vehicle when it returns to the route, i.e., the vehicle can travel till location $(v + l_v)$ without refueling again. We also define the fuel capacity of the vehicle by R, which is an upper bound on the fuel level l_v for any stop v.

In the offline problem, the entire set of n candidate stops, their locations, costs, and fuel levels are available offline to the algorithm. In contrast, the online problem is defined using a *look-ahead parameter* d, and the details of a candidate stop are revealed only when the stop is within distance d of the vehicle's location. We will be evaluating our online algorithms using the standard benchmark of *competitive ratio*, which is the worst-case (over all instances) ratio of the cost suffered by the algorithm versus that of an optimal (offline) solution. For simplicity, we also assume that the vehicle starts with fuel level R, i.e., at full capacity.

In the more general *multi-resource pit stop problem*, there are k types of resources that deplete over time. For simplicity, we assume that a unit of travel depletes a unit of each resource type, but we also note that our results easily generalize to the setting where resources

[1] e.g., https://trips.furkot.com/

deplete at arbitrary and non-uniform rates for each portion of the trip and at each stop. Let R_q be the capacity with respect to resource q and let l_{vq} be the level of resource q after making stop v. When we make stop v with entry level l_q for a given resource q, we may decide to replenish it to l_{vq} or leave it as is and suffer the c_v cost on it, depending on what is more beneficial. (Note that this is not a concern for the single resource case since we would not stop at the location where the replenishment level is lower than the incoming resource level. But, in the multi-resource case, we might need to stop because of some other resource, which forces us to decide whether to replenish a different resource or not.) Hence, the exit level of resource q will be $\max\{l_{vq}, l_q - c_v\}$. If l is a vector of resource levels, we write $u_v(l)$ for the exit levels after stopping at v, which are defined as explained above. Again, the objective is to plan a minimum cost schedule of stops subject to not running out of any resource.

In the offline setting, we also briefly discuss the multi-resource pit stop problem *on a graph*, i.e., when the trip route is decided in conjunction with the stops before the commencement of the journey. This incorporates the pit stop problem in route planning itself, which might be relevant in situations that involve controlled environments with limited uncertainty (for instance, when planning a route using dedicated High Occupancy Vehicle (HOV) lanes). In this case, the algorithm returns a route from the source to the destination along with the stops to minimize cost subject to not running out of any resource during the trip. We note that this more general problem is less natural in the online setting since route planning is typically done prior to the trip. In any case, it is easy to show strong lower bounds that rule out any competitive algorithm for the online problem on a general graph instead of a fixed route.

3 OFFLINE PIT STOP PROBLEM

In this section, we consider the offline version of the pit stop problem, i.e., where the entire instance with all potential stops are available to the algorithm.

3.1 Single Resource

Even though this particular version is solved in [21], we present a dynamic program (DP) to optimally solve the offline pit stop problem as it facilitates the discussion of the more general case. We use $C(v, l)$ to denote the optimal cost of reaching t from v with fuel $l \in L_v$ at v. The DP is based on the following recurrence, where v' is the stop that immediately follows v (note that $v' - v$ is the distance between them):

$$C(v, l) = \min\left\{C(v', l - (v' - v)), c_v + C(v', l_v - (v' - v))\right\} \quad (1)$$

with the following initial conditions:

$$C(v, l) = \infty,\ \forall v, l < 0 \text{ and } C(t, l) = 0,\ \forall l \geq 0. \quad (2)$$

Intuitively, this recurrence chooses between the two options of stopping and refueling at v versus skipping v. Let L_v be the set of possible fuel levels at location v. Note that the cardinality of this set is bounded by the number of stops n, since there are at most as many possible fuel levels at v as potential stops to refuel at before reaching it. For every v, we only need to compute $C(v, l)$ for $l \in L_v$.

Theorem 3.1. *Solving the offline pit stop problem recurrence backwards from t computes the optimal cost $C(0, R)$ in $O(n^2)$ time.*

Proof. First we explain correctness. The initial conditions (2) ensure the optimal cost $C(t, l)$ is correct for every l. Now suppose we have the correct optimal cost for every $l \in L_{v'}$ for some given $v' > 0$. We will argue that recurrence (1) computes the correct optimal cost $C(l, v)$ for every $l \in L_v$, for the stop v that immediately precedes v'. There are two options given any fuel level l: either charge or not. The DP considers both of these options, the first term in the min function is the option to skip the stop and move on to the next location with fuel levels depleted by $v' - v$. The second term is to refuel at v, which suffer a cost of c_v but updates the fuel level at v to l_v, followed by the same depletion of $v' - v$ to reach v' from v. Using the inductive hypothesis at v', we can now claim that the DP identifies the smaller of these two optimal as the optimal cost at v with level l.

The time complexity of the DP is $O(n^2)$. There are n stops and for each stop v there are at most n possible levels in L_v. Hence, there are $O(n^2)$ values $C(l, v)$ to be computed and each such computation takes constant time. □

3.2 Multiple Resources

The DP can be generalized to the case of multiple resources as we explain in the rest of the section. We omit details on optimality, which follows similarly to the fuel only case. Let L_{vq} be the set of possible levels of resource q at location v. Let $L_v = \times_q L_{vq}$ be all the possible vectors of resource levels at v. We write $C(v, l)$ for the optimal cost of reaching t from v with resource levels $l \in L_v$ at v. Below, we write $\mathbf{e}$ for the vector of k ones, where k is the number of resources. The DP is based on the following recurrence, where v' is the stop that immediately follows v:

$$C(v, l) = \min\left\{C(v', l - (v' - v)\mathbf{e}), c_v + C(v', u_v(l) - (v' - v)\mathbf{e})\right\}.$$

and the initial conditions:

$$C(v, l) = \infty,\ \forall v, l \in L_v \text{ such that some } l_j < 0.$$
$$C(t, l) = 0,\ \forall l \in L_v \text{ such that } l \geq 0.$$

Here $u_v(l)$ is the function that updates resource levels at stop v, as described in Section 2. The number of values to compute in the DP table is at most $\sum_v |L_v| \leq n^{k+1}$ and each one is computed in constant time, thus giving an overall run time of $O(n^{k+1})$. Typically, the number of resource types k is much smaller than the number of candidate stops n and can be treated as a constant. The algorithm's run time then is polynomial in n.

3.3 General Graph and Multiple Resources

In this section we consider the graph multi-resource pit stop problem. Our input graph is G and there are k types of resources. Let L denote the maximum number of different levels of each resource. We now set up a shortest path problem whose solution yields the optimal route including stops for the pit stop problem.

The nodes of the transformed graph G' are constructed as follows. For each node v in G, the transformed graph G' has a set of L^k nodes labeled (v_{in}, l) and another L^k nodes labeled (v_{out}, l) where l is a vector of resource levels. Node (v_{in}, l) corresponds to arriving

at stop v with resource levels l and node (v_{in}, l) corresponds to departing from stop v with resource levels l.

The edges of G' are added as follows. Let c be the distance from v to v' in G. Then for every $l \geq c\mathbf{e}$ (recall that $\mathbf{e}$ denotes the vector of k ones), we add an edge from (v_{out}, l) to $(v'_{in}, l - c\mathbf{e})$. This edge is assigned a length of 0 in G'. We also add an edge from each (v_{in}, l) to $(v_{out}, u_v(l))$, where $u_v(\cdot)$ is the resource update function for stop v. The length of this edge is equal to the cost of the stop, c_v.

It is not difficult to see that the edges of the first type correspond to the feasible trips between stops, whereas edges of the second type correspond to actually making stop v. Resource updates and costs are tracked exactly during trips and stops. If we want to get from node s to node t starting with resource levels l, we may run a shortest path algorithm on the transformed graph from node (s, l) and stop when we reach the first node of the form (t, l'), for any l'. This recovers the minimum cost schedule of stops on the graph. Since the transformed graph G' has $2L^k n$ nodes and at most as many edges (since every vertex in G' has out-degree at most 1), the running time of Dijkstra's shortest path algorithm on G' is $O(L^k n(\log n + k \log L))$.

4 ONLINE PIT STOP PROBLEM

In this section, we consider the pit stop problem in the online setting, i.e., where stops are only visible within a lookahead distance of d from the current location of the vehicle.

4.1 Single Resource

First, we consider the version of the problem where fuel is the only resource. We first prove that when $d < R$ then any algorithm is arbitrarily bad when compared to the optimal solution.

Theorem 4.1. *If $d < R$, then any algorithm for the online pit stop problem has an arbitrarily large competitive ratio.*

Proof. We construct an instance with two candidate stops. Stop 1 is at location 1 and Stop R at location R. The destination t is at location $R + 1$. We have $c_1 = 1$, $l_1 = R$ and $l_R = 1$. The cost c_R is unknown as it is beyond the look-ahead distance d. In effect the algorithm has to pick one of the two stops in order to get to the destination, without knowing the cost of Stop R. An adversary can then easily set the cost of Stop R to be 0 or ∞ depending on the choice that the algorithm makes at Stop 1 and make the algorithm's choice arbitrarily worse than that of the optimal solution. □

Given the above lower bound, we assume that $d \geq R$ for the remainder of this section. Specifically, we prove that we can get constant competitive ratio when $d = R$. Note that the optimal solution does not depend on the value of d, and hence this immediately implies a constant competitive ratio for any value of $d \geq R$. For the remainder of the section we will assume that t is a multiple of R. This is an assumption that simplifies our analysis and algorithm description and is in effect without loss of generality due to the following simple transformation: We may turn the true destination into a stop of zero cost which offers fuel level R and place a dummy destination at the next location that is a multiple of R.

Before presenting our full algorithm, we discuss some special cases of the problem and the algorithms that solve them within a constant approximation. Our algorithm for the general case will then be a conceptual generalization of these tailored solutions.

4.1.1 $c_v = 1$ for all stops. First, we consider a special case of our problem where all stops have a unit cost but might offer different fuel levels. One simple approach for this setting is to pick the last stop that the vehicle can reach as the next refueling spot. This algorithm however falls short if this last stop offers a small amount of fuel. A better choice, therefore, is to pick the stop that will offer a fuel level such that we can travel the furthest, namely maximize $v + l_v$. Indeed, this turns out to the optimal strategy in this case.

4.1.2 $l_v = R$ for all stops. Our second special case is one where all stops offer the maximum amount of fuel but their costs differ. Intuitively, in this setting we would like to pick the cheapest stop within our look-ahead range. However, this strategy perform poorly in the following setting: Consider a dense sequence of stops where the first one has unit cost and each stop costs an arbitrarily small ϵ more than the previous one. The above algorithm will pick every single one of these stops, whereas the optimal solution picks only one stop every R steps. Correcting for this leads to the following algorithm: Find the cheapest stop in our look-ahead range. Suppose it has cost c. Scale that cost by a constant parameter α and pick the latest stop in our range that has a cost at most αc. Setting, for example, $\alpha = 2$ gives an algorithm with constant competitive ratio. We omit the details of the argument since the proof of our main result that we describe next subsumes it. The main takeaway from this special case is that we need to round costs up to constants to ensure that the algorithm is not too sensitive to small cost differences between stops.

4.1.3 General Case. We now present our algorithm for the general case, Algorithm 1. The algorithm sets milestones every R steps. At all times, the algorithm computes, for every fuel level l, optimal solutions (using the offline DP) from the current location (say v) to the next milestone (say x) and from x to $v + R$ such that the fuel level at x is at least l (lines 4-7). Note that these optimal solutions can be obtained, since the algorithm has visibility on all stops up to $x + d \geq x + R$. Among these solutions for different fuel levels, the algorithm chooses (lines 8-9) the one that has maximum fuel level at x, under the constraint that its cost is within α times the minimum cost ($\alpha > 2$ is a parameter that we will optimize later.) This solution is now used to upgrade the plan from v to x as follows. The algorithm, "purchases" this upgraded schedule of stops up to the milestone, by which we mean that the algorithm commits to making each one of these stops when reaching it. This allows the algorithm to reset the cost of these purchased stops to 0 (lines 10-11). When we eventually get to the milestone, the algorithm also purchases the stops that are after it (lines 13-16).

We now prove that this algorithm has a constant competitive ratio. First, we introduce some notation. Let $p_0 = 0, p_1 = R, p_2 = 2R, \ldots, p_{t/R} = t$ be the milestones of the instance. Now, let c_i^* be the cost incurred by the optimal set of stops, OPT, between p_{i-1} and p_i. Also, let $S_1^i, S_2^i, \ldots, S_{k_i}^i$, be the sets of stops $S_L^{l^*}$ (in line 8) when moving in $[p_{i-1}, p_i)$ and in iterations of the while loop in which the algorithm actually purchased new stops. Similarly, let $\hat{S}_1^i, \hat{S}_2^i, \ldots, \hat{S}_{k_i}^i$ be the upgraded sets of stops that the algorithm actually purchased

Algorithm 1: Online Pit Stop Algorithm

Input: Capacity R, stop info $N = \{c_v, l_v\}_v$, destination t, parameter α
Output: Set of stops S

1 $S \leftarrow \emptyset$; $f \leftarrow R$; $v \leftarrow 0$; $x \leftarrow R$ // Variables: stops set, current fuel level, location, and milestone
2 **while** $v \neq t$ **do**
3 $h \leftarrow v + R$ // Planning horizon
4 **for** *all possible fuel levels* $l \in L_v$ **do**
  // Get optimal to the milestone and from the milestone to the horizon for l
5 $S_L^l \leftarrow \text{GetOpt}(v, f, x, l, N)$
6 $S_R^l \leftarrow \text{GetOpt}(x, l, h, 0, N)$
7 **end**
8 $l^* \leftarrow \arg\min_{l=0,1,\ldots,R} \left\{c(S_L^l \cup S_R^l)\right\}$
9 $\hat{l} \leftarrow \max_{l=0,1,\ldots,R} \left\{l : c(S_L^l) \leq \alpha \cdot c(S_L^{l^*})\right\}$ // Get highest fuel solution within α of the best
10 $S \leftarrow S \cup S_L^{\hat{l}}$ // Add stops until the milestone
11 $c_u \leftarrow 0$ for all $u \in S_L^{\hat{l}}$ // Update "purchased" stops
12 $f \leftarrow u_v(f)$ // Refuel
13 **if** $v = x - 1$ **then**
14 $S \leftarrow S \cup S_R^{\hat{l}}$ // Add all stops at milestone
15 $x \leftarrow x + R$ // Update the milestone
16 **end**
17 $f \leftarrow f - (next(v) - v)$ // Fuel at the next stop
18 $v \leftarrow next(v)$ // Move to the next stop
19 **end**
20 **return** S

in line 10 and (for the last set only) line 14. Finally, let T^i be the set added by line 14 when on $p_i - 1$.

Main result proof plan. We will first show that, as the vehicle moves towards any given milestone, the costs of successive solutions computed by the DP of line 5 increase by a multiplicative factor $\alpha - 1$. This will suggest that the cost of the last such computed solution dominates all previous ones (within a constant approximation). We prove this in Lemma 4.2. Next we will show that the cost of any such solution of line 5 is bounded by a constant multiple of the cost that an optimal solution suffers until the *next* milestone. We prove this in Lemma 4.3. Combining these two lemmas shows that the sum of all solutions computed at line 5 is bounded by a constant multiple of the cost suffered by the optimal solution in the same area. We will finally combine with the fact that, due to line 9, the costs of purchased stop sets are always within a constant of the DP solutions of line 5, to get that the algorithm achieves a constant competitive ratio.

Lemma 4.2. *For every i, j, it holds that $c(S_j^i) \geq (\alpha - 1) \cdot c(S_{j-1}^i)$.*

Proof. We begin by tracing how and why the algorithm has chosen these sets. The algorithm at some point computed set S_j^i as part of the best solution to go from the current location v to $v + R$. Specifically, that was the best set of stops to buy in order to reach the milestone p_i. Suppose the corresponding cost at the time of calculation was c. The algorithm possibly upgraded this solution to $\hat{S}_j^i$, which it actually purchased at cost at most αc. Before reaching the milestone, the algorithm computed S_{j+1}^i as the best way to reach the milestone on the way from $v' > v$ to $v' + R$. We note that at that point, the algorithm had already purchased stops $\hat{S}_j^i$ to reach the milestone at a given fuel level f. This implies that the algorithm switched to S_{j+1}^i in order to have a higher fuel level $f' > f$ at the milestone.

Suppose, for the sake of contradiction, that $c(S_j^i) < (\alpha - 1) \cdot c(S_{j-1}^i)$. We argue that the algorithm should have purchased $S_j^i \cup S_{j+1}^i$ instead of $\hat{S}_j^i$ in the previous round of purchases. We get:

$$c(S_j^i \cup S_{j+1}^i) \leq c(S_j^i) + c(S_{j+1}^i) < \alpha \cdot c(S_j^i) = \alpha c.$$

We observe that the cost of these stops is also at most αc and the induced fuel level at the milestone is $f' > f$, which means that line 9 would not have picked $\hat{S}_j^i$. This gives a contradiction. □

Lemma 4.3. *The following hold:*

(1) *For every $i \in [1, t/R - 1]$ and every $j = 1, 2, \ldots, k_i$, we get $c(S_j^i) \leq c_i^* + c_{i+1}^*$.*
(2) *For every $i \in [1, t/R - 1]$, we get $c(T^i) \leq c_i^* + c_{i+1}^*$.*
(3) *For $i = t/R$, we get $c(S_j^{t/R}) \leq c_{n/R}^*$.*

Proof. We first consider $i \in [1, t/R - 1]$. We argue that, at the beginning of every interval $[p_{i-1}, p_i)$, we have a high enough fuel level or have purchased enough stops to reach any location where the optimal solution *OPT* refuels in this window. Note that at least one such stop exists since there must be at least one stop every R steps. There are two cases for each stop: either it is on p_{i-1}, in which case our claim holds trivially since the vehicle is already there, or it is somewhere in $[p_{i-1} + 1, p_i - 1]$. In this case, we distinguish between the first window $i = 1$ and any other window after that. For the first window, the fact that we start with the same fuel level as the optimal solution, ensures our claim holds and we can reach the stop under consideration. For subsequent windows, the property is given by the declaration of Algorithm 1. At location $p_{i-1} - 1$ and because of line 14, the algorithm purchases enough stops to be able to reach $p_i - 1$, which proves our claim.

Since we can reach the stops where *OPT* refuels in this window and since we can follow the schedule of *OPT* after it all the way to p_{i+1}, the cost of any solution computed in line 8, will be at most $c_i^* + c_{i+1}^*$, as long as the vehicle has not driven past the last of *OPT*'s stops in the window. We will now consider the case when the vehicle has driven past the last such stop v. Consider the moment the vehicle reached v. The optimal solution follows a path from v to p_{i+1} that costs at most $c_i^* + c_{i+1}^*$, hence, we have an option from v to $v + R \leq p_{i+1}$ that costs at most that much. The algorithm will either add such a set of stop P or upgrade the stops before p_i so that the fuel level at p_i is larger than what P achieves (line 9). In both cases the algorithm purchases any stops necessary to reach the first stop of P after p_i. Hence, as we move between v and p_i, there will always exist an option that costs at most $c_i^* + c_{i+1}^*$ and carries the vehicle all the way to p_{i+1}, which is always more than

distance R away from any location in $[v, p_i]$. This proves that in this scenario as well, line 8 will find a set that costs at most $c_i^* + c_{i+1}^*$. This completes item 1 of the lemma.

It is not hard to see that the above reasoning is sufficient to prove item 2 of the lemma as well. Upon reaching $p_i - 1$, the algorithm can still reach the first stop of P after p_i and hence can achieve a cost of at most $c_i^* + c_{i+1}^*$ for T^i.

Finally we prove the claim for $i = n/R$, i.e., item 3 of the lemma. Similarly to the $i \in [1, t/R - 1]$ case, when at $p_{t/R-1}$, we have a high enough fuel level or have purchased enough stops to reach the stops where OPT refuels in this window. This follows identically as in the first paragraph of this proof. Hence, it follows that there exists a plan that takes us to location t and costs at most $c_{t/R}^*$. The algorithm will add the corresponding stops and terminate. □

Theorem 4.4. *Algorithm 1 is $(8 + 4\sqrt{2})$-competitive.*

Proof. The competitive ratio of the algorithm now follows from the above lemmas using a sequence of inequalities. Let ALG be the solution returned by the algorithm and OPT an optimal solution. We get:

$$c(ALG) = \sum_{i=1}^{t/R} \sum_{j=1}^{k_i} c(\hat{S}_j^i) + c(T^i) \leq \sum_{i=1}^{t/R} \sum_{j=1}^{k_i} \alpha c(S_j^i) + c(T^i),$$

where the inequality follows from line 9 of Algorithm 1. By Lemma 4.2 we get that for every j, $c(S_{j-1}^i) \leq c(S_j^i)/(\alpha - 1)$. Applying multiple times we get that for every j, $c(S_j^i) \leq c(S_{k_i}^i)/(\alpha - 1)^j$. Hence, we get:

$$\begin{aligned} c(ALG) &\leq \sum_{i=1}^{t/R} \alpha \sum_{j=1}^{k_i} \frac{c(S_{k_i}^i)}{(\alpha-1)^j} + c(T^i) \\ &\leq \sum_{i=1}^{t/R} \alpha \cdot \frac{\alpha - 1}{\alpha - 2} \cdot c(S_{k_i}^i) + c(T^i). \end{aligned}$$

Lemma 4.3 shows that every $c(S_{k_i}^i)$ and $c(T^i)$ are at most $c_i^* + c_{i+1}^*$. Then, we get:

$$c(ALG) \leq \sum_{i=1}^{t/R} (\alpha \cdot \frac{\alpha - 1}{\alpha - 2} + 1)(c_i^* + c_{i+1}^*) \leq 2(\alpha \cdot \frac{\alpha - 1}{\alpha - 2} + 1) c(OPT).$$

Optimizing for the value of α, we get $\alpha = 2 + \sqrt{2}$ and a competitive ratio of $8 + 4\sqrt{2}$. □

Run time. The time complexity of the algorithm is dominated by the number of calls to the DP (i.e., to GetOpt), which computes an optimal solution time quadratic in n, as explained in Section 3.1. The while loop of line 2 runs n times and the for loop of line 4, at most $|L_v| \leq n$ times per each iteration of the while loop, for a total of $O(n^2)$ calls to the DP and a run time of $O(n^4)$. We note that our experimental evaluation shows that the algorithm is much faster in practice and always completes in a few tens of milliseconds on a single core for realistic instances.

4.2 Multiple Resources

First, we show that there is no online algorithm with a provable guarantee even with only two resource types.

Theorem 4.5. *The competitive ratio of any algorithm for the online pit stop problem with more than one resource types is unbounded.*

We defer the proof to the full version due to space limitations.

Algorithm 2: Online Multi-Resource Pit Stop Algorithm

Input: Look-ahead d, capacities vector R, stop info $N = \{c_v, l_v\}_v$, destination t
Output: Set of stops S

1 $S \leftarrow \emptyset$; $f \leftarrow R$; $v \leftarrow 0$; $x \leftarrow R$ // Variables: stops set, current fuel level, location, and milestone
2 **while** $v \neq t$ **do**
3 $\quad h \leftarrow v + d$ // Planning horizon
4 $\quad S' \leftarrow \text{GetOpt}(v, f, h, 0, N)$ // Get the best set of stops from v at level f to x at level l
5 $\quad S \leftarrow S \cup S'$ // Add stops to the solution
6 $\quad c_u \leftarrow 0$ for all $u \in S_L^{\hat{l}}$ // Update "purchased" stops
7 $\quad f \leftarrow u_v(f)$ // Replenish resource levels
8 $\quad f \leftarrow f - (next(v) - v)e$ // Spend resources
9 $\quad v \leftarrow next(v)$ // Move to the next stop
10 **end**
11 **return** S

Given the above impossibility result, we design a heuristic for the multiple resource case, Algorithm 2. Our heuristic resembles a simplified version of Algorithm 1, in the sense that it computes an optimal solution on the set of visible stops and follows it until something better is recomputed once further stops become visible. Specifically, at every location v, the algorithm computes an optimal solution to reach $h = d + v$ (where d is the look-ahead parameter). This is done by invoking the multi-resource offline algorithm from the previous section and computing the corresponding optimal set of stops S'. This set S' is added to the set of "purchased" stops S, i.e., the algorithm commits to stop at these locations and their costs are set to 0. This computation is repeated at each location and the overall solution is the collection of all stops added by the individual solutions.

Run time. As in the case of the single resource algorithm, the multi-resource algorithm's run time is also dominated by calls to the DP subroutine (GetOpt), which has a $O(n^{k+1})$ run time (since it invokes the multi-resource variant of the offline algorithm). The while loop is repeated n times, which gives an overall run time of $O(n^{k+2})$. As in the case of a single resource, our experimental evaluation shows that the algorithm is much faster than this worst case run time bound in practice and always completes in a few tens of milliseconds on a single core for realistic instances.

5 EXPERIMENTAL EVALUATION

In this section we evaluate the performance of our algorithms using real data. We focus on road trips with an electric vehicle (EV). Initially we consider the pit stop problem with a single resource, the battery level of the EV, and subsequently we add the consumption of food by the passengers as a second consideration. We compare the performance of our algorithms against the following baselines.

Baseline 1 (Greedy). Pick a stop if and only if it is the last stop for battery/food before we run out of battery/food.

Baseline 2 (Cheap Greedy). Pick a stop if and only if it is the cheapest stop for battery/food before we run out of battery/food.

These are natural baselines for the problem and, moreover, they are the algorithmic solutions for the special cases discussed in Sections 4.1.1 and 4.1.2. To provide intuition on the behavior of the baselines, consider a single-resource instance where we begin at location 0 and want to reach location $2R$. There are three candidate stops: one at location $R/2$, which costs 1 and provides fuel R, one at location R which costs 3 and provides fuel R, and one at location $3R/2$ which costs 1 and provides fuel $R/2$.

The Greedy baseline skips past stop $R/2$ since there is enough fuel to reach stop R. It then selects to charge at R, before driving all the way to the destination.

Cheap Greedy refuels at $R/2$, given that the stops in range are $R/2$ and R, and the former is the cheaper one. Then Cheap Greedy skips stop R, since the stops in range at that point are R and $3R/2$ and the latter is cheaper. Cheap Greedy ends up picking R and $3R/2$.

Note that Greedy suffers a cost of 3 whereas Cheap Greedy suffers 2. If we were to modify the cost of stops $R/2$ and $3R/2$ from 1 to 2, the algorithms would make the exact same decisions and Greedy will have a cost of 3 vs 4 for Cheap Greedy.

For each family of instances (single-resource and two-resource), we conduct two types of experiments. For the first one, we assign costs to the stations that are given by characteristics provided in the data (detour cost, charging speed). We call these the *static* costs of the stations. For the second type, we add a random cost to each station that models uncertain road congestion and service waiting times. This is an important aspect of our model, as it tests how the different algorithms can handle uncertainty in costs and, hence, unforeseeable conditions with respect to traffic, service queues, etc. For this second type of experiment we consider the following third baseline of interest.

Baseline 3 (Static Cost Optimal). Compute an optimal solution on the static costs of the stations.

The Static Cost Optimal baseline solves the problem assuming knowledge of all station static costs. In this sense it is very powerful compared to our algorithm and the other baselines, which can only observe the stations within the look-ahead range. However, this baseline cannot observe the random costs that are added to the stations. Note that when the random costs are relatively small, this baseline will perform very close to optimal and in the absence of random costs it will be exactly optimal.

The performance comparison between algorithms is done in terms of the average competitive ratio over 1000 instances for each experiment. The optimal solutions are computed by the offline DPs we describe in Sections 3.1 and 3.2.

5.1 Experimental Set Up

We extract locations and kW values of EV charging stations in Europe and distances between them from a popular Maps API. We also extract information about which of these stations include restaurants on site. We construct instances for the pit stop problem by randomly selecting endpoints from this set of stations and computing the shortest route between them that has a station in proximity at least once every 100 km. The resulting instances have from very few up to approximately 200 candidate stations. There were no instances with mode than 200 candidate stations.

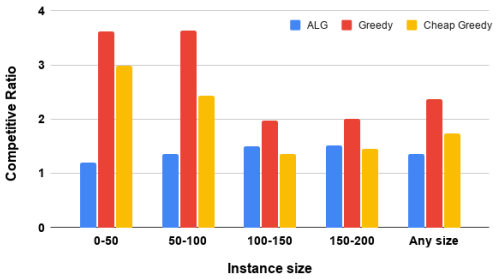

Figure 1: The competitive ratio of our algorithm and the baselines for different instance sizes.

We assume we are routing an electric vehicle vehicle with a range of 300 km and that passengers need to feed at least once every 500 km. In the first round of experiments we ignore the food resource and solve instances of the single-resource pit stop problem. In the second round of experiments, we consider both resources and solve the two-resource pit stop problem.

Before we present our evaluation results, we give a short note on the run time of our algorithms, which proved to be extremely fast in practice. All instances in our experiments completed within 100 ms, with the highest times appearing for two-resource instances with more than 100 candidate stops. Typical two-resource instances and all single-resource instances were solved in at most 20 ms. The experiments were conducted on a single 3.5 GHz core.

5.2 EV Charging Stops

In all of our experiments we consider the costs as unknown outside the look-ahead range, which for our battery-only experiments, is set to 300 km. We repeat the EV charging experiment twice. We initially assign the (unknown) costs to the stations based only on characteristics extracted from the data, i.e., the total cost of a station is the sum of the detour required to reach it and a charging time that is inversely proportional to the station's kW.

Recall that Algorithm 1 is parameterized by the cost scaling parameter α. Recall that this parameter controls the amount by which we allow ourselves to over-spend until the next milestone (compared to the minimum possible cost), so we can have a higher battery level at it. We run 1000 instances of the pit stop problem and find that a value around 1.5 seems to be the best-performing for our dataset. This is the value that we will use for our algorithm in the comparisons against the baselines that follow.

Our second experiment is the first comparison between our algorithms and the baselines. We let the algorithms run for 1000 instances and compute the average competitive ratio for each one of them. Over all instances, the results yield 1.35 for our algorithm, 2.37 for Greedy and 1.74 for Cheap Greedy. We also group the instances by size (i.e., number of total candidate stations) and present the

Figure 2: The competitive ratio of our algorithm and the baselines for different levels of uncertainty in costs.

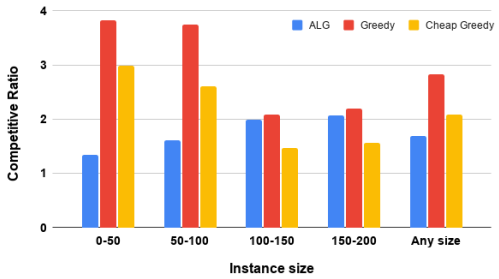

Figure 3: Competitive ratios of our algorithm and the baselines against the instance size with two resources.

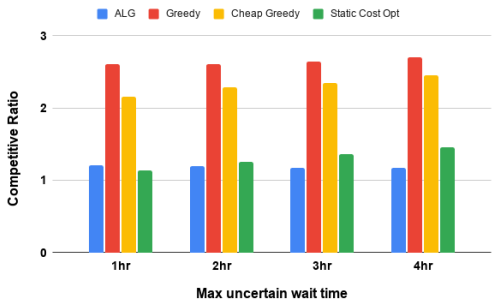

Figure 4: Competitive ratios of our algorithm and the baselines for different levels of uncertainty with two resources.

average competitive ratio achieved by each algorithm against the size of the instance in Figure 1. Looking at the trends, we observe that Algorithm 1 has a very strong performance on small instances, due to the fact that the planning done by the DP subroutine is expected to be very close to the optimal solution for small instances. Performance slightly degrades as the instances become larger and finally stabilizes. The greedy algorithms do their worst on small instances since myopic mistakes are costly when the number of total decisions made is small. The greedy algorithms' performance improves on larger instances before it stabilizes, and in fact we see that Cheap Greedy matches and sometimes slightly beats Algorithm 1 on the rare very large instances. As we will see, this is not the case once uncertainty is introduced. The fraction of smaller instances in the data is higher, it is very rare to encounter a trip during which the vehicle passes near 100 or more stations. Specifically, approximately 90% of the instances are in the 0-100 candidate stations range.

Our third experiment adds synthetic costs to the stops that correspond to unforeseen traffic congestion on the detours and waiting times at the stations. Each stop draws a uniformly random cost in $[0, x]$, which is added to its initial cost. The extent of uncertainty is controlled by parameter x. In Figure 2, we present the average competitive ratio of the algorithms against the value of x which we let vary from 1 hour to 4 hours. We observe that the performance of the algorithms is not impacted too much by the extent of uncertainty (at least for these realistic values of x) and that Algorithm 1 significantly outperforms the two greedy baseline algorithms.

The Static Cost Optimal baseline has a strong performance, as expected, when the uncertainty is small. As uncertainty grows, and in contrast to the other baselines, the performance of this baseline degrades. Overall, it is worse than Algorithm 1, even though it uses knowledge of the static costs of future stations outside the look-ahead parameter. The fact that Algorithm 1 beats this baseline exhibits the need for an online algorithm that works with uncertain costs. Relying on an offline solution is not enough and, in fact, the performance of the Static Cost Optimal baseline will keep degrading as the random costs become large compared to the static ones (e.g., in applications where the static costs are close to 0 such an algorithm is in effect useless).

5.3 EV Charging Stops and Restaurants

In this section we add a second resource type to our experiments, specifically we extract information about the location of restaurants in the vicinity of the charging stations. We label the subset of charging stations that have a restaurant onsite as food stations as well. We repeat the experiments of the previous section, using the multi-resource version of our algorithm (Algorithm 2) in place of the single-resource one (Algorithm 1), requesting that there is a charging stop every 300 km and a food stop every 500 km.

Over all instances, the average competitive ratios were 1.68 for our algorithm, 2.83 for Greedy, and 2.08 for cheap Greedy. Figure 3 is similar to the findings of the single-resource case for different instance sizes. We do observe larger competitive ratios for all algorithms compared to the single-resource case, as was expected. Again we observe the expected behavior of the DP-based algorithm slightly degrading in performance as the instance size increase, while the opposite is true for the greedy baselines.

Figure 4, where we plot the competitive ratios of the algorithms with different levels of cost uncertainties, also verifies the findings of the single-resource case. Again the performance of the algorithms is only mildly dependent on changes to the level of uncertainty, and our algorithm outperforms the baselines.

6 CONCLUSIONS

In this paper, we considered the problem of planning stops during a journey for availing services such as fuel/battery charge, food, or

even rest. This is a problem routinely solved by online navigation and trip planning software that are ubiquitous today. We abstracted its key features and modeled it as an optimization problem that we called the pit stop problem. We gave offline and online algorithms for this problem. Our offline algorithms are optimal, while the online algorithms obtain a constant competitive ratio. We also evaluated our algorithms on real world data and showed that they significantly outperform natural greedy baselines. Perhaps the most interesting direction of future work would be to incorporate some of the heterogeneity between stops of difference types (say, between a refueling stop and a detour to a scenic point of interest, one being mandatory while the other is optional) in the optimization problem.

7 ACKNOWLEDGEMENTS

Debmalya Panigrahi was supported in part by NSF grants CCF-1750140 and CCF-1955703 and ARO grant W911NF2110230.

REFERENCES

[1] Vural Aksakalli, O. Furkan Sahin, and Ibrahim Ari. 2016. An AO* Based Exact Algorithm for the Canadian Traveler Problem. *INFORMS J. Comput.* 28, 1 (2016), 96–111.
[2] Saeed Alaei. 2014. Bayesian combinatorial auctions: Expanding single buyer mechanisms to many buyers. *SIAM J. Comput.* 43, 2 (2014), 930–972.
[3] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. 2004. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In *STOC*. 166–174.
[4] Amotz Bar-Noy and Baruch Schieber. 1991. The Canadian Traveller Problem. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms*. 261–270.
[5] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff. 2007. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM J. Comput.* 37, 2 (2007), 653–670.
[6] M. Charikar, S. Khuller, and B. Raghavachari. 2001. Algorithms for capacitated vehicle routing. *SIAM J. Comput.* 31, 3 (2001), 665–682.
[7] M. Charikar and B. Raghavachari. 1998. The finite capacity dial-a-ride problem. In *FOCS*. 458–467.
[8] C. Chekuri, N. Korula, and M. Pál. 2012. Improved algorithms for orienteering and related problems. *ACM Trans. Algorithms* 8, 3 (2012), 23:1–23:27.
[9] T. Chen, B. Zhang, H. Pourbabak, A. Kavousi-Fard, and W. Su. 2018. Optimal Routing and Charging of an Electric Vehicle Fleet for High-Efficiency Dynamic Transit Systems. *IEEE Transactions on Smart Grid* 9, 4 (2018), 3563–3572.
[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2009. *Introduction to Algorithms (3. ed.)*. MIT Press.
[11] DHL. [n.d.]. Self Driving Vehicles in Logistics. http://https://www.dhl.com/content/dam/downloads/g0/about_us/logistics_insights/dhl_self_driving_vehicles.pdf.
[12] E. B. Dynkin. 1963. Optimal choice of the stopping moment of a Markov process. *Dokl. Akad. Nauk SSSR* 150 (1963), 238–240. Issue 2.
[13] Patrick Eyerich, Thomas Keller, and Malte Helmert. 2010. High-Quality Policies for the Canadian Traveler's Problem. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.
[14] P. R. Freeman. 1983. The Secretary Problem and Its Extensions: A Review. *International Statistical Review / Revue Internationale de Statistique* 51, 2 (1983), 189–206.
[15] B. L. Golden and A. A. Assad (Eds.). 1988. *Vehicle Routing: Methods and Studies*. Studies in Management Science and Systems, Vol. 16. North-Holland, Amsterdam.
[16] I. L. Gørtz, V. Nagarajan, and R. Ravi. 2015. Minimum makespan multi-vehicle dial-a-ride. *ACM Trans. Algorithms* 11, 3 (2015), 23:1–23:29.
[17] A. Gupta, M. T. Hajiaghayi, V. Nagarajan, and R. Ravi. 2010. Dial a ride from k-forest. *ACM Trans. Algorithms* 6, 2 (2010), 41:1–41:21.
[18] M. Haimovich and A. H. G. Rinnooy Kan. 1985. Bounds and heuristics for capacitated routing problems. *Math. Oper. Res.* 10, 4 (1985), 527–542.
[19] Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, and Tuomas Sandholm. 2007. Automated Online Mechanism Design and Prophet Inequalities. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*. 58–65.
[20] DP Kennedy. 1987. Prophet-type inequalities for multi-choice optimal stopping. *Stochastic Processes and their Applications* 24, 1 (1987), 77–88.
[21] Samir Khuller, Azarakhsh Malekian, and Julián Mestre. 2011. To fill or not to fill: The gas station problem. *ACM Trans. Algorithms* 7, 3 (2011), 36:1–36:16.
[22] Robert Kleinberg. 2005. A multiple-choice secretary algorithm with applications to online auctions. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 630–631.
[23] Ulrich Krengel and Louis Sucheston. 1977. Semiamarts and finite values. *Bull. Amer. Math. Soc.* 83, 4 (1977), 745–747.
[24] Ulrich Krengel and Louis Sucheston. 1978. On semiamarts, amarts, and processes with finite value. *Probability on Banach spaces* 4 (1978), 197–266.
[25] C-L. Li and D. Simchi-Levi. 1990. Worst-case analysis of heuristics for multidepot capacitated vehicle routing problems. *INFORMS Journal on Computing* 2, 1 (1990), 64–73.
[26] Jane Lin, Wei Zhou, and Ouri Wolfson. 2016. Electric vehicle routing problem. *Transportation Research Procedia* 12, Supplement C (2016), 508–521.
[27] D. V. Lindley. 1961. Dynamic Programming and Decision Theory. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 10, 1 (1961), 39–51.
[28] Evdokia Nikolova and David Karger. 2008. Route Planning under Uncertainty: The Canadian Traveller Problem. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI*. 969–974.
[29] Christos Papadimitriou and Mihalis Yannakakis. 1991. Shortest paths without a map. *Theoretical Computer Science* 84 (1991), 127–150.
[30] Martin Sachenbacher, Martin Leucker, Andreas Artmeier, and Julian Haselmayr. 2011. Efficient energy-optimal routing for electric vehicles. In *Twenty-fifth AAAI conference on artificial intelligence*.
[31] Ester Samuel-Cahn. 1984. Comparison of Threshold Stop Rules and Maximum for Independent Nonnegative Random Variables. *The Annals of Probability* 12, 4 (1984), 1213–1216.
[32] P. Toth and D. Vigo (Eds.). 2001. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.
[33] H. Yang, Y. Deng, J. Qiu, M. Li, M. Lai, and Z. Y. Dong. 2017. Electric Vehicle Route Selection and Charging Navigation Strategy Based on Crowd Sensing. *IEEE Transactions on Industrial Informatics* 13, 5 (2017), 2214–2226.