Heuristic Search for Path Finding with Refuelling

Shizhe Zhao[1*], Anushtup Nandy[2*], Howie Choset[2], Sivakumar Rathinam[3] and Zhongqiang Ren[4]

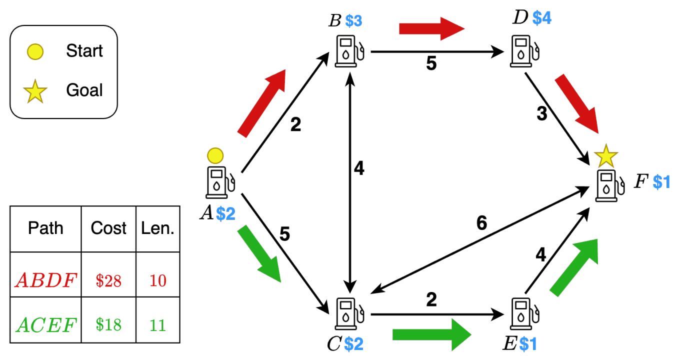
***Abstract*—This paper considers a generalization of the Path Finding (PF) problem with refuelling constraints referred to as the Gas Station Problem (GSP). Similar to PF, given a graph where vertices are gas stations with known fuel prices, and edge costs are the gas consumption between the two vertices, GSP seeks a minimum-cost path from the start to the goal vertex for a robot with a limited gas tank and a limited number of refuelling stops. While GSP is polynomial-time solvable, it remains a challenge to quickly compute an optimal solution in practice since it requires simultaneously determine the path, where to make the stops, and the amount to refuel at each stop. This paper develops a heuristic search algorithm called Refuel A* (RF-A*) that iteratively constructs partial solution paths from the start to the goal guided by a heuristic while leveraging dominance rules for pruning during planning. RF-A* is guaranteed to find an optimal solution and often runs 2 to 8 times faster than the existing approaches in large city maps with several hundreds of gas stations.**

***Index Terms*—Motion and Path Planning, Scheduling and Coordination, Robotics in Under-Resourced Settings**

I. Introduction

GIVEN a graph with non-negative edge costs, the Path Finding problem seeks a minimum-cost path from the given start vertex to a goal vertex. This paper considers a Gas Station Problem (GSP), where the vertices represent gas stations with known fuel prices, and the edge costs indicate the gas consumption when moving between vertices. The fuel prices can be different at vertices and are fixed over time at each vertex. GSP seeks a start-goal path subject to a limited gas tank and a limited number of refuelling stops while minimizing the total fuel cost along the path (Fig. 1).

GSP was studied [8], [11], [17], [26], and arises in applications such as path finding for electric vehicles between cities [2], [3], [18] and package delivery using an unmanned vehicle [10], [12], where a robot needs to move over long distances when refuelling becomes necessary. While GSP is polynomial time solvable [8], [11], [17], it remains a challenge to quickly compute an optimal solution in practice since the robot needs to simultaneously determine the path, where to make the stops, and the amount of refuelling at each stop, possibly in real-time with limited on-board computation.



Path	Cost	Len.
ABDF	$28	10
ACEF	$18	11

Fig. 1: An illustrative example of GSP. This graph consists of six vertices representing gas stations, each associated with a gas price, and each edge with its fuel expenditure. The objective is to find a minimum-cost path from start to goal, assuming the tank capacity is 5 and the refuelling stop limit is 3. The figure shows the minimum-cost path (*ACEF*), using green arrows, and the minimum fuel consumption path, (*ABDF*) using red arrows. Note that the minimum fuel consumption path does not incur the lowest fuel cost. Along the optimal solution *ACEF*, the cost of refuelling at each vertex is: $10 at A, $14 at C, $18 at E.

This paper focuses on exact algorithms that can solve GSP to optimality. In [11], a dynamic programming (DP) approach is developed to solve GSP to optimality, which has been recently further improved in terms of its theoretic runtime complexity [17]. This approach identifies a principle regarding the amount of refuelling the robot should take at each stop along an optimal path. This principle allows the decomposition of GSP into a finite number of sub-problems, and DP can be leveraged to find an optimal solution by iteratively solving all these sub-problems.

To expedite computation, this paper develops a new heuristic search algorithm called RF-A* (Refuelling A*), which iteratively constructs partial solution paths from the start vertex to the goal guided by a heuristic function. RF-A* gains computational benefits over DP in the following aspects. First, RF-A* never explicitly explores all sub-problems as DP does and only explores the sub-problems that are needed for the search. Second, RF-A* uses a heuristic to guide the search, reducing the number of sub-problems to be explored before an optimal solution is found. Third, taking advantage of our prior work in multi-objective search [21], [24], [25], RF-A* introduces a dominance rule to prune partial solutions during

Manuscript received: October 15, 2024; Revised January 4, 2025; Accepted January 31, 2025

This paper was recommended for publication by Editor Bera Aniket upon evaluation of the Associate Editor and Reviewers' comments.

This work was supported by the National Science Foundation under Grant Nos. 2120219 and 2120529. The authors at Shanghai Jiao Tong University are supported by the Natural Science Foundation of China under Grant No. 62403313. (Corresponding author: Zhongqiang Ren.)

[1] Shizhe Zhao is with UM-SJTU Joint Institute, Shanghai Jiao Tong University, China. Email: shizhe.zhao@sjtu.edu.cn

[2] Anushtup Nandy and Howie Choset are at Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA. Emails: {anandy, choset}@andrew.cmu.edu

[3] Sivakumar Rathinam is with the Department of Mechanical Engineering and the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843-3123. Email: srathinam@tamu.edu

[4] Zhongqiang Ren is with UM-SJTU Joint Institute and the Department of Automation, Shanghai Jiao Tong University, China. Email: zhongqiang.ren@sjtu.edu.cn

* Equal contribution and co-first authors.

Digital Object Identifier (DOI): see top of this page.

the search, which saves computation. RF-A* is guaranteed to find an optimal solution.

We compare RF-A* against DP [11] and a method based on mixed-integer programming (MIP) [14] in real-world city maps of various sizes from the OpenStreetMap dataset. Our results show that RF-A* is orders of magnitude faster than the MIP, and is often 2 to 8 times faster than DP in the city maps with hundreds of gas stations. In addition, RF-A* is up to 64 times faster than DP, when the heuristic can be pre-computed and cached for planning. These results demonstrate the scalability of RF-A*, enabling it to plan for a robot with a limited tank in large urban areas.

II. Related Work

Path planning with refueling constraints (GSP) involves determining a path and a refueling schedule simultaneously. Dynamic programming methods were introduced by [11], [17] for the GSP either from a given start to a goal or for all-pair vertices in the graph. Besides planning start-goal paths, another related problem generalizes travelling salesman problem and vehicle routing problem with refuelling constraints [1], [9], [11], [14], [28]. These problems seek a tour that visits multiple vertices subject to a limited fuel tank. This paper only considers finding a start-goal path.

Recently, due to the prevalence of electric vehicles (EV) and unmanned aerial vehicles (UAV), several variants of GSP were proposed to address fuel constraints using various methods. Mathematical programming models were proposed by [19], which generalized the refuelling cost in EV applications. [13] addresses the refuelling constraints for an UAV cruise system under an online setting using a greedy approach. [5] considers the fairness of resource utilization and employs GSP as a cost function. [27] considers the uncertainty of the waiting time for refuelling and proposed a policy that can be solved by dynamic programming. Other methods also appear in literature to deal with refuelling constraints, such as constraint programming [20] and learning [13], [16].

Among these methods, both mathematical programming and dynamic programming can guarantee solution optimality for GSP. Mathematical programming models the GSP as a mixed integer program and then invokes an off-the-shelf solver to handle the problem [19]. Dynamic programming (DP) decomposes the GSP to a finite number of sub-problems, then exploits the relation between these sub-problems to find the optimal solution by solving all sub-problems iteratively [11]. However, DP often performs redundant work. This paper aims to reduce the redundancy of DP in GSP by heuristic search.

III. Problem Statement

Let $G = (V, E)$ denote a directed graph, where each vertex $v \in V$ represents a gas station, and each edge $(u, v) \in E$ denotes an action that transits the robot from vertex u to v. Each edge $(u, v) \in E$ is associated with a non-negative real value $d(u, v) \in \mathbb{R}^+$, called edge cost, which represents the amount of fuel needed to traverse the edge from u to v. The robot has a fuel capacity $q_{max} \in \mathbb{R}^+$ representing the maximum amount of fuel it can store in its tank. Let $c : V \to [0, \infty]$ denote the refuelling price per unit of fuel at each vertex in $v \in V$.[1]

Let a path $\pi(v_1, v_\ell) = (v_1, v_2, \ldots, v_\ell)$ be an ordered list of vertices in G such that every pair of adjacent vertices in $\pi(v_1, v_\ell)$ is connected by an edge in G, i.e., $(v_i, v_{i+1}) \in E, i = 1, 2, \ldots, \ell-1$. Let $g(\pi)$ denote total fuel cost along the path; specifically, let a non-negative real number $a(v) \in \mathbb{R}^+$ denote the amount of refuelling taken by the robot at vertex v, then $g(\pi) = \sum_{i=1,2,\ldots,\ell} a(v_k) c(v_i)$.

In practice, the robot often has to stop to refuel, which slows down the entire path execution time. Therefore, let $k_{max} \in \mathbb{Z}^+, k_{max} > 1$ denote the maximum number of refuelling stops the robot is allowed to make along its path.

Definition 1 (Gas Station Problem (GSP) [11]) *Given a pair of start and goal vertex $v_o, v_g \in V$, the robot has zero amount of fuel at v_o and must refuel to travel. GSP seeks a path π from v_o to v_g and the amount of refueling $a(v), v \in \pi$ along the path, such that $g(\pi)$ is minimized, while the number of refuelling stops along π is no larger than k_{max}, and the amount of fuel in the tank is not greater than q_{max} after each refuelling.*

Remark 1 *In the literature [11], [14], the graph is often assumed to be a fully connected graph. For a graph G that is not fully connected, one can convert it to a fully connected graph G' by finding a minimum edge cost path $\pi(u, v)$ (without any limit on the number of refuel stops) for each pair of vertices (u, v) in G and using an edge $e' \in G'$ to indicate path $\pi(u, v)$. Although such a conversion is straightforward, it may require extra runtime when deploying the algorithm on a robot. For this reason, this paper does not assume the graph G is fully connected and presents GSP on general graphs.*

IV. Method

This section introduces RF-A*, a heuristic search approach to find an optimal solution for GSP. It initiates at start vertex v_o and systematically explores potential paths from v_o towards goal vertex v_g while minimizing the overall fuel cost. The heuristics help estimate the remaining cost to the goal and guide the search. RF-A* also considers the fuel tank limit q_{max} and the refuelling stop limit k_{max}, by comparing two paths that reach the same vertex using multiple criteria. During the search, RF-A* maintains an open set of candidate paths that are to be expanded, similar to A*. It continues searching until finding an optimal path satisfying the constraints. A toy example of the search process is provided in Fig. 2.

A. Notations and Background

1) Basic Concepts: In GSP, there can be multiple paths from v_o to a vertex v, and to differentiate them, we use the notion of labels. Intuitively, a label $l = (v, g, q, k)$ consists of a vertex $v \in V$, a non-negative real number $g \in \mathbb{R}^+$ that represents the cost-to-come from v_o to v, a non-negative real number $q \in \mathbb{R}^+$ that represents the amount of fuel remaining at v before refuelling, and an integer $0 \leq k < k_{max}$

[1] For vertices v in G where the robot cannot refuel, let $c(v) = \infty$.

indicates the number of refuelling stops before v. We use $v(l), g(l), q(l), k(l)$ to denote the respective component of a label. To compare labels, we use the following notion of label dominance.

Definition 2 *Given two labels l, l' with $v(l) = v(l')$, label l dominates l' if the following three inequalities hold: (i) $g(l) \leq g(l')$, (ii) $q(l) \geq q(l')$ and (iii) $k(l) \leq k(l')$.*

If l dominates l', then l' can be discarded during the search, since for any path from v_o via l' to v_g, there must be a corresponding path from v_o via l to v_g with the same or smaller cost. Otherwise, both l and l' are non-dominated by each other. For a vertex $v \in V$, let $\mathcal{F}(v)$ denote a set of labels that reach v and are non-dominated by each other. $\mathcal{F}(v)$ is also called the *frontier set* at v. Additionally, the procedure *CheckForPrune*(l) compares a label l against all existing labels in $\mathcal{F}(v(l))$ to check if l is dominated and should be discarded.

Similarly to A*, let $h(l)$ denote the h-value of label l that estimates the cost-to-go from v to v_g. We further explain the heuristic in Sec. IV-B2. Let $f(l) = g(l) + h(l)$ be the f-value of label l. Let OPEN denote a priority queue of labels, where labels are prioritized based on their f-values from the minimum to the maximum.

A major difficulty in GSP is determining the amount of refuelling at each vertex during the search, a continuous variable that can take any value in $[0, q_{max}]$. To handle this difficulty, we borrow the following lemma from [11], which provides an optimal strategy for refuelling at any vertex.

Lemma 1 (Optimal Refuelling Strategy) *Given refuelling stops $v_1, \ldots, v_n$ along an optimal path using at most k_{max} stops in a complete graph. At v_{g-1}, which is the stop right before the goal vertex v_g, refuel enough to reach v_g with an empty tank. Then, an optimal strategy to decide how much to refuel at each stop for any $n < g - 1$:*

- *if $c(v_n) < c(v_{n+1})$, then fill up entirely at v_n.*
- *if $c(v_n) \geq c(v_{n+1})$, then fill up enough to reach v_{n+1}.*

The intuition behind Lemma 1 is that the robot either fills up the tank if the next stop has a higher fuel price, or fills just enough amount of fuel to reach the next stop if the next stop has a lower price. By doing so, the robot minimizes its accumulative fuel cost. A detailed proof is given in [11]. Note that Lemma 1 assumes the graph is complete. To adapt it to a general graph, we need to identify all possible transitions from one vertex to any other vertices in G (going through one or multiple edges) without refuelling, which will be described in *ComputeReachableSets* in the next section.

Algorithm 1 RF-A*

1: *ComputeReachableSets*()
2: *ComputeHeuristic*(v_g)
3: $l_o \leftarrow (v_o, g = 0, q = 0, k = 0),\ f(l_o) \leftarrow 0 + h(l_o)$
4: $parent(l_o) \leftarrow NULL$
5: Add l_o to OPEN
6: $\mathcal{F}(v) \leftarrow \emptyset, \forall v \in V$
7: **while** OPEN $\neq \emptyset$ **do**
8: pop $l = (v, g, q, k)$ from OPEN
9: **if** *CheckForPrune* $(l, \mathcal{F}(v(l)))$ **then**
10: **continue**
11: add l to $\mathcal{F}(v(l))$
12: **if** $v(l) = v_g$ **then**
13: **continue**
14: **if** $k = k_{max}$ **then**
15: **continue**
16: **for all** $v' \in$ *GetReachableSet*$(v(l))$ **do**
17: **if** $c(v') > c(v)$ **then**
18: $g' \leftarrow g(l) + (q_{max} - q(l))c(v)$
19: $q' \leftarrow q_{max} - d(v, v')$
20: $k' \leftarrow k + 1$
21: **else**
22: **if** $d(v', v) \geq q(l)$ **then**
23: $g' \leftarrow g(l) + (d(v', v) - q(l))c(v)$
24: $q' \leftarrow 0$
25: $k' \leftarrow k + 1$
26: **else**
27: **continue** ▷ No need to refuel
28: $l' \leftarrow (v', g', q', k')$
29: $g(l') \leftarrow g'$
30: **if** *CheckForPrune* $(l', \mathcal{F}(v(l)))$ **then**
31: **continue**
32: $f(l') \leftarrow g(l') + h(v(l'))$
33: $parent(l') \leftarrow l$
34: add l' to OPEN
35: **return** *Reconstruct*(v_d)

Algorithm 2 *ComputeReachableSets*

$Reach(v) \leftarrow \emptyset, \forall v \in V$
for $v \in V$ **do**
$d^*(u) \leftarrow \infty, \forall u \in V$
$d^*(v) \leftarrow 0$
Add v to OPEN$_v$
while OPEN$_v \neq \emptyset$ **do**
pop u from OPEN$_v$
if $d^*(u) > q_{max}$ **then**
continue
else
add u to $Reach(v)$
for $u' \in$ *GetSucc*(u) **do**
if $u' \in Reach(v)$ **then**
continue
if $d^*(u') > d^*(u) + d(u, u')$ **then**
$d^*(u') \leftarrow d^*(u) + d(u, u')$
add u' to OPEN$_v$

B. Refuel A Algorithm*

RF-A* (Alg. 1) takes a graph G, tank capacity q_{max}, v_o, v_g, and max refuelling stops k_{max} as the inputs. It begins by calling Alg. 2 *ComputeReachableSets* (IV-B3) to compute the set of all vertices that the robot can travel to from any vertex u given a full tank. Subsequently, to compute the heuristic which gives the amount of fuel needed to reach v_g from any other vertex, it runs an exhaustive backward Dijkstra search from v_g. We present the heuristic computation in Sec. IV-B2. After the Dijkstra, RF-A* initiates the label $l_o = (v_o, g = 0, q = 0, k = 0)$ at vertex v_o with the f-value, $f(l_o) = h(l_o)$, and inserts it into OPEN. The frontier set $\mathcal{F}(v)$ at each $v \in V$ is initialized as an empty set $\emptyset$.

During the search (Lines 8-34), in each iteration, the label with the lowest f-value is popped from OPEN for further processing. This label is checked for dominance against existing labels in $\mathcal{F}(v(l))$ using *CheckForPrune*. This procedure

Algorithm 3 *CheckForPrune*$(l, \mathcal{F}(v(l)))$

```
INPUT: A label l and F(v(l)), the frontier set at vertex v(l).
for all l' ∈ F(v(l)) do
    if g(l') ≤ g(l) and q(l') ≥ q(l) and k(l') ≤ k(l) then
        return true                              ▷ l should be pruned.
return false                                     ▷ l should not be pruned.
```

employs Def. 2 to compare the g, q and k values of the popped label against other labels in $\mathcal{F}(v(l))$.

If the selected label is non-dominated (and thus unpruned), it is added to the frontier set. Subsequently, the algorithm checks if the vertex of the label $v(l)$ is v_g, which means that the label l represents a solution with the minimum cost, and the search terminates. It is also confirmed whether the k_{max} stops limit has been reached. In cases where $v(l) \neq v_g$ and $k' \neq k_{max}$, the label is expanded, which generates new labels for all reachable vertices from $v(l)$ in G. This involves a loop that iterates each reachable vertex and creates a new label l' with new g', q', k'. The amount of refuelling is determined using Lemma 1, and the corresponding accumulative fuel cost g' is computed. Note that l' is generated only if a refuelling stop at $v(l)$ is required. v' at Line 27 can thus be skipped. Finally, the algorithm uses *CheckForPrune* to check for dominance. If l' is not pruned, l' is added to OPEN for future expansion.

1) ComputeReachableSets: This procedure aims to find all successor vertices that RF-A* needs to consider when expanding a label. To achieve this, it identifies all vertices $v' \in V$ that the robot with a full tank can reach from vertex v without refuelling. Specifically, Alg. 2, initializes Reach(v) as an empty set for each $v \in V$. In each iteration, it designates a vertex v and runs a Dijkstra search from v to all other vertices in G to find vertices that are reachable from v without refuelling. Lines 3-17 show this Dijkstra search process starting from a specific vertex v. This algorithm involves iterating through vertices in V and traversing their successors to update the distances.

2) Heuristic Computation: A possible way to compute the heuristic is to first run an exhaustive backward Dijkstra search on G from v_g to any other vertices in G using fuel consumption $d(u, v)$ (ignoring the fuel tank limit of the robot and the refuelling cost). After this Dijkstra search, let $d_{v_g}(v)$ denote the minimum fuel consumption path from v to v_g. Let $c_{min} := \min_{v \in V - \{v_g\}}(c(v))$ denote the minimum fuel price in G. Then, let $h(l) = \max\{(d_{v_g}(v(l)) - q(l))c_{min},\ 0\}$ be the h-value of label l. When computing this heuristic, the tank limit of the robot is ignored and the fuel price at any station is a lower bound of the true price at that station. As a result, $h(v)$ provides a lower bound of the total fuel cost to reach v_g from v. We therefore have the following lemma.

Lemma 2 (Admissible Heuristic) *The heuristic,* $h(l) = max\{(d_{v_g}(v(l)) - q(l))c_{min}, 0\}$, *is admissible.*

3) CheckForPrune$(l, v(l))$: As shown in Alg. 3, this procedure checks for dominance for each label l against all labels in $\mathcal{F}(v(l))$, the frontier set at vertex $v(l)$, by using Def. 2. Alg. 3 iterates the frontier set $\mathcal{F}$, with its efficiency determined by the size $|\mathcal{F}(v(l))|$, leading to a complexity of $O(|\mathcal{F}(v(l))|)$. This dominance check can be expedited by the techniques in [24], [25], which is left for our future work.

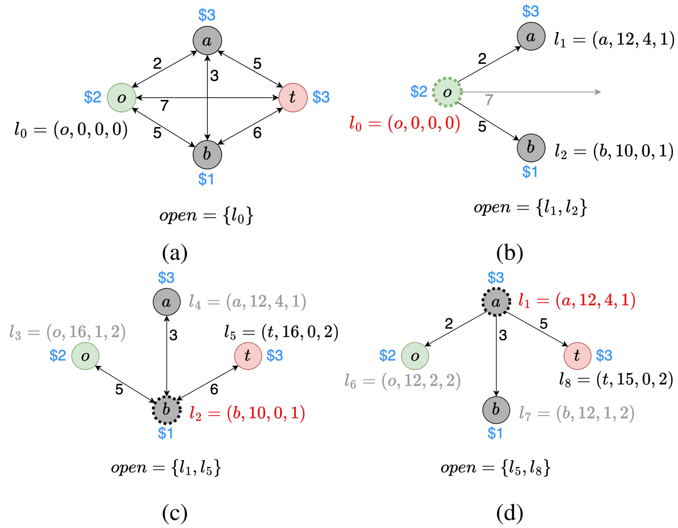


Fig. 2: A toy example showing the Refuel A* with four vertices. The graph G has four vertices (o, a, b and t), with fuel consumption in black, fuel price in blue, $q_{max} = 6$ and $k_{max} = 2$. The i-th label associated with a vertex v is $l_i = (v, g_i, q_i, k_i)$. The optimal path is shown to be $o \to b \to t$, with a cost of 15.

Remark 2 *We use the same definition of GSP as in [11], which assumes the robot always starts with an empty tank. For the cases where the initial tank at v_o is not empty (i.e, $q_o > 0$), one can construct a new problem that starts with an empty tank at a pseudo vertex v_p, and v_p is only connected with v_o such that, by following the Lemma 1, the robot can only fill up the tank at v_p and then reaches v_o with q_o amount of remaining fuel. More details about constructing the new problem is described in [11].*

C. RF-A* Example

An example of Alg. 1 RF-A* is shown in Fig. 2. It considers a graph G with four vertices $o, a, b,$ and t. The source is o, and goal t. The label $l_0 = (o, 0, 0, 0)$ is initiated and inserted into OPEN. In the first search iteration, as seen in Fig. 2(b), l_0 is popped from OPEN. l_0 does not reach the goal vertex. Its reachable vertices are computed, which leads to labels $l_1 = \{a, 12, 4, 1\}$ and $l_2 = \{b, 10, 0, 1\}$. To compute l_1, we can see that, since $c(a) > c(o)$, the robot fills up the tank and then moves to a which causes $g(l_1) = 12$, $q(l_1) = 6 - 2 = 4$ and $k(l_1) = 1$. Alternately, $c(b) < c(o)$, so the robot only fills up the amount needed to go to b, leading to label l_2 with $g(l_2) = 10$, $q(l_2) = 5 - 5 = 0$ and $k(l_2) = 1$. Both l_1 and l_2 are added to the OPEN as they are non-dominated. Next (Fig. 2(c)), l_2, which has the lowest f-value, is popped from OPEN, and follows the same process for generating labels $l_3, l_4$and l_5 corresponding to vertices o, a and t with $k(l_3) = k(l_4) = k(l_5) = 2$. Subsequently, l_3 and l_4 are pruned by l_0

and l_1 respectively, using *CheckForPrune*. Hence, only l_5 is added into the OPEN, that is, OPEN = $\{l_1, l_5\}$. In the next iteration, as shown in Fig. 2 (d), l_1 is popped and generates l_8. The OPEN becomes $\{l_8, l_5\}$. l_8 is the next popped label. It reaches the goal and represents an optimal solution. The path and the minimum cost are $o \to b \to t$ and 15 respectively.

D. Analysis

In the worst case, RF-A* has the same run-time complexity $O(k_{max}n^3)$ as the basic version of the DP method [11]. This scenario for RF-A* may occur when the heuristic is absent (e.g. $h(v) = 0$ for any $v \in V$), dominance pruning does not occur, and all possible labels are expanded. However, as shown in Sec. V, in practice, RF-A* is often much faster than the DP method. The following theorem summarizes this property.

Theorem 1 *RF-A* has polynomial worst-case run-time complexity.*

In Alg. 1, due to Line 14, RF-A* never expands a label l with $k(l) = k_{max}$. As a result, RF-A* never generates labels with more than k_{max} refuelling stops, and the path returned by RF-A* is feasible, i.e., does not exceed the limit on the number of stops k_{max}. The following lemma thus holds.

Lemma 3 (Path Feasibility) *The path returned by RF-A* is feasible.*

To expand a label, RF-A* considers all reachable neighboring vertices as described in Sec. IV-B1 and determine the amount of refuelling via Lines 17-25. With Lemma 1, the expansion of a label l in RF-A* is complete, in a sense that, all possible actions of the robot, which may lead to an optimal solution, are considered during the expansion. The following lemma summarizes this property.

Lemma 4 (Complete Expansion) *The expansion of a label in RF-A* is complete.*

During the search, if a label is pruned by dominance in *CheckForPrune*, then this label cannot lead to an optimal solution for the following reasons. If a label $l = (v, g, q, k)$ is dominated by any existing label $l' = (v, g', q', k') \in \mathcal{F}(v(l))$. This means that l has a higher cost $g \geq g'$, lower remaining fuel $q \leq q'$ and has made more stops $k \geq k'$ than l'. Assume that expanding l leads to an optimal solution π_*, and let $\pi_*(v, v_g)$ denote the sub-path within π_* from v to v_g. Then, another path π' can be constructed by concatenating the path represented by l' from v_o to v, and $\pi_*(v, v_g)$ from v to v_g. Path π' is feasible and its cost $g(\pi') \leq g(\pi_*)$. So, π' is a better path than π_*, which contradicts with the assumption. We summarize this property with the following lemma.

Lemma 5 (Dominance Pruning) *Any label that is pruned by dominance cannot lead to an optimal solution.*

We now show that RF-A* is complete and returns an optimal solution for solvable instances.

Theorem 2 (Completeness) *For unsolvable instances, RF-A* terminates in finite time. For solvable instance, RF-A* returns a feasible solution in finite time.*

Proof: Due to Lemma 1 and that the graph G is finite, only a finite number of possible labels can be generated during the search. With Lemma 4, the expansion of a label is complete, which means, RF-A* eventually enumerates all possible labels. For an unsolvable instance, RF-A* terminates in finite time after enumerating all these labels. For solvable instances, due to Lemma 3, RF-A* terminates in finite time and finds a label that represents a feasible path.

Theorem 3 (Solution Optimality) *For solvable instances, the path returned by RF-A* is an optimal solution.*

Proof: When a label l is popped from OPEN and claimed as a solution by RF-A*, due to Lemma 2, any other labels in OPEN and their successor labels cannot lead to a cheaper solution than $g(l)$. With Lemma 5, the pruned labels cannot lead to a cheaper solution than l.

V. Experimental Results

This section compares the performance of RF-A* with DP [11], and a Mixed-Integer Programming (MIP) model, which are described in Sec. V-A. Both DP and RF-A* require a preprocessing to compute the reachable set for all vertices in the graph (Alg. 2). Although this preprocessing is time consuming and takes most of the runtime of both methods, it only needs to run once to support arbitrary number of invocation on the planner, allowing the runtime to be amortized. So we exclude the preprocessing time from the runtime, readers can find it from Table I. In the remaining sections, all runtime values are solely for the search after the preprocessing.

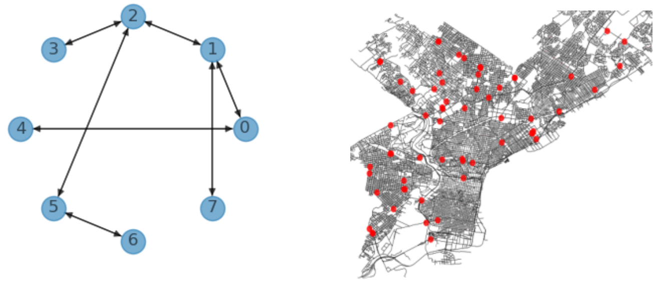
In the experiments, we use both a synthetic and a real-world dataset. Fig. 3 shows an example of each dataset. The synthetic dataset includes three small (*Synth-S*) and three large (*Synth-L*) random graphs. Each random graph is a binomial graph that has a single connected component, with a probability of 0.3 for edge creation [7]. The vertex numbers are 8, 16 and 32 for *Synth-S*, and 256, 512, 1024 for *Synth-L*. For the synthetic maps, the refuelling cost c at vertices are random integers from 1 to 10.

The real-world dataset consists of five road networks from OpenStreetMap. For each city map, the refuelling cost c at vertices are randomly sampled from 2.5 to 4.5[2]. Table I presents a summary of the graphs.

Table II shows the default parameters, where the tank capacity q_{max}, and the maximum refuelling stop k_{max} are chosen based on Table I in the following way. For small maps (*Synth-S*), we set q_{max} and k_{max} to ensure that there is always a solution. For large maps (*Synth-L* and *City*), we set q_{max} to approximately three times the average edge cost. A pair of start and goal vertex in a map defines a GSP instance. For all instances, we confirmed that both DP and RF-A* always have the same cost, which is not surprising since they both guarantee solution optimality. Each GSP instance has a 30-seconds runtime limit. All methods are implemented in C++ and tested on Ubuntu 22.0 Desktop with a 13th Gen Intel i7-13700 and 32GB RAM. [3]

[2]Values are referenced from https://gasprices.aaa.com

[3]Our software and dataset is available at https://github.com/rap-lab-org/public_refuelastar



(a) (b)

Fig. 3: (a) Visualization of a random map. (b) Visualization of the map for Philadelphia, USA. It showcases the road network of the city, where the red dots represent the gas stations.

Dataset	Map	$\lvert V \rvert$	$\lvert E \rvert$	E_{median}	Preproc.(s)
Synth-S	8	8	15	5.0	0.0
	16	16	51	4.0	0.0
	32	32	269	4.0	0.0
Synth-L	256	256	19473	4.0	0.2
	512	512	78783	4.0	2.8
	1024	1024	314529	4.0	33.7
City	Phil	61	3661	9920.4	0.0
	Austin	87	7483	9842.9	0.0
	Phoenix	178	31507	19021.7	0.2
	London	258	66307	22930.7	0.7
	Moscow	423	178507	20706.2	3.6

TABLE I: Summary of the two datasets. E_{median} is the median edge cost. *Preproc.* is the preprocessing time in second.

A. Baselines

1) Dynamic Programming (DP): We provide details of DP for GSP from [11]. This DP defines a set of sub-problems where each one is (v, k, q) with $v \in V$, k denoting the number of stops and q denoting the gas level. For each sub-problem, let $A(v, k, q)$ represent the minimal cost to traverse from vertex v to the goal v_g, within k refuelling stops, and starting with q units of fuel. With the help of Lemma 1, for each $v \in V$, there is only a finite number of possible values that q can take, which is bounded by $|V|$. As a result, the set of sub-problems is finite since each of v, k, q can take a finite number of possible values. The base case of the DP method is $A(v_o, 0, 0) = 0$, and the default value for all other sub-problems are ∞. Then, the DP method iteratively solve all sub-problems $A(v, k, q)$ based on the base case or previously solved sub-problems. The optimal solution can be obtained from $A(v_g, k_{max}, 0)$. In [11], two methods are presented: a naive method with a time complexity of $O(k_{max}n^3)$ and an advanced method reduces the complexity to $O(k_{max}n^2 \log(n))$. We use the advanced method for comparison in our experiments.

Parameters\Datasets	Synth-S	Synth-L	City
q_{max}	8	15	60000
k_{max}	3	10	10

TABLE II: Default input parameters of each dataset, where *Synth-S* and *Synth-L* are synthetic random maps with small size (8, 16, 32) and large size (256, 512, 1024) respectively

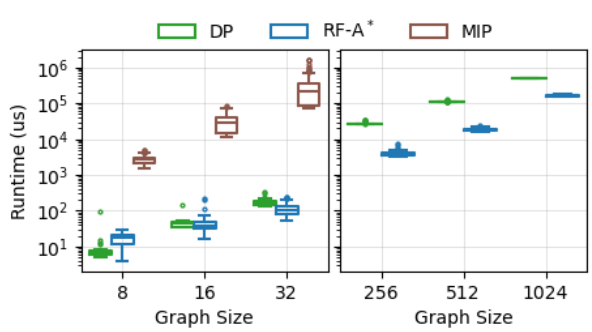


Fig. 4: Runtime on the synthetic dataset.

2) Mixed Integer Programming Formulation: We introduce a simple Mixed Integer Programming formulation as an alternative baseline to solve the GSP. The solver for the MIP model is Gurobi 11. A binary variable $x(u, v)$ indicates if a path passes through the edge from u to v, and binary variable $y(u)$ indicates whether the robot refuels at the vertex u. Decision variables $a(u), q(u)$ are introduced in Sec. III. The objective function in Eq.1 is the total fuel cost along the path (borrowed from Sec. III), and is to be minimized. Eq. (2) and Eq. (3) define the domain of decision variables. Eq. (4) defines the limit on the refuel stops, and Eq. (5) defines the path from v_o to v_g. Eq. (6) indicates that if an edge (u, v) is on the path, i.e., $x(u, v) = 1$, then the remaining fuel at v must be equal to the remaining fuel at u plus the amount of refuelling at u, minus the consumption on the edge. Eq. (7) expresses the optimal refuelling strategy in Lemma 1.

$$\min_{a(u),q(u),x(u,v)} \left(\sum c(u)a(u)\right) \tag{1}$$

$$\begin{aligned} x(u,v) &\in \{0,1\} \\ q(u), a(u) &\geq 0 \\ q(o) &= 0 \\ q(u) + a(u) &\leq q_{max} \end{aligned} \tag{2}$$

$$y(u) = \begin{cases} 1, & a(u) > 0 \\ 0, & a(u) = 0 \end{cases} \tag{3}$$

$$\sum_{u \in V} y(u) \leq k_{\max} \tag{4}$$

$$\sum_{v \in V} x(u,v) - \sum_{v \in V} x(v,u) = \begin{cases} 1, & u = v_o; \\ -1, & u = v_g; \\ 0, & u \in V/\{v_o, v_g\} \end{cases} \tag{5}$$

$$(q(u) + a(u) - d(u,v) - q(v))x(u,v) = 0, \quad (u,v) \in E \tag{6}$$

$$\begin{gathered} \forall x(u,v) = 1\,, (u,v) \in E, \\ \begin{cases} a(u) = q_{max}, & c(u) < c(v) \\ a(u) + q(u) \geq d(u,v), & c(u) \geq c(v) \end{cases} \end{gathered} \tag{7}$$

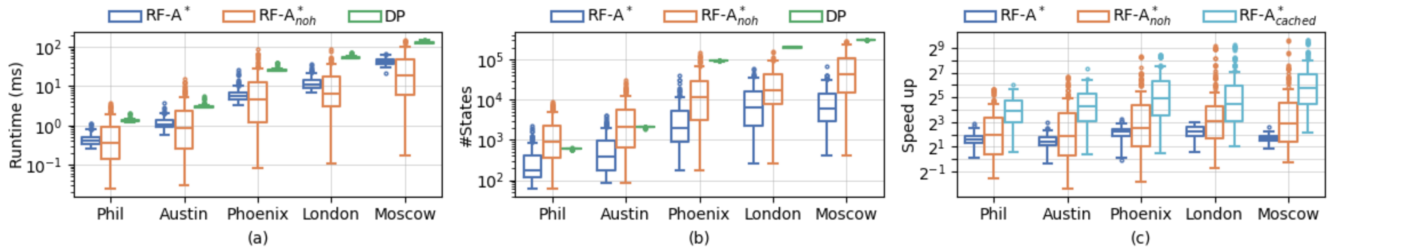


Fig. 5: (a) and (b) show runtime and memory cost of all methods. (c) shows the speed-up of RF-A* variants compare to DP.

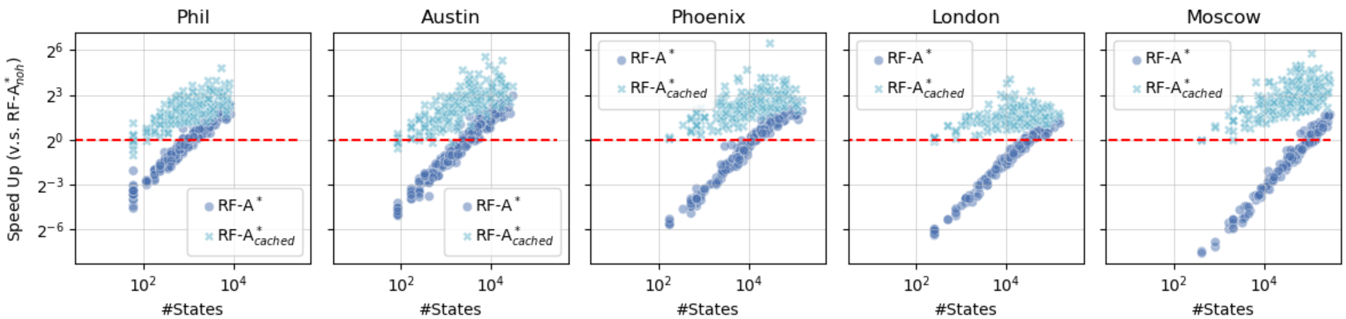


Fig. 6: Speed-up compared to RF-A$^*_{noh}$, with the increase in its explored states. Points below the red dash lines indicate performance worse than RF-A$^*_{noh}$.

B. Synthetic Dataset Results

For each map from *Synth-S*, we create an instance for each possible pair of start-goal vertices, and for each map from *Synth-L*, we randomly select 100 vertex pairs. Fig. 4 presents the results.

In *Synth-S*, we can see that both DP and RF-A* can find optimal solutions within tens of microseconds. Although DP initially performs slightly faster than RF-A* when handling a small graph, this advantage diminishes as the size of the graph increases. Eventually, in *Synth-L* RF-A* becomes faster than DP by several factors. Conversely, MIP is slower than both DP and RF-A* by orders of magnitude, and this gap increases as the size of the graph grows. We therefore remove the MIP from the subsequent experiments.

C. City Dataset Results

The runtime of RF-A* includes heuristic computation and search process. The former requires a backward Dijkstra from the goal location, and the computed heuristic can reduce the runtime of the search by generating fewer labels. In practice, we can reuse the computed heuristic as long as the goal location remains the same, namely, by caching the heuristic. The overall influence of the heuristic on the search process are two-folds. On the one hand, it can slow down the search due to the overhead on computing the heuristic. On the other hand, it can accelerate the search if the speed-up on search outweighs the overhead, or if no overhead exists when the cached heuristic can be used.

To show the effectiveness of the heuristic in RF-A*, we introduce additional baselines. The first, RF-A$^*_{noh}$, conducts a search without a heuristic. The second, RF-A$^*_{cached}$, excludes the runtime to compute the heuristic before the search starts and only counts the runtime for search, representing an ideal situation where a cached heuristic is always available.

In this dataset, instances are 100 randomly selected vertex pairs from the graph. To compare the number of sub-problems have been explored by RF-A*, RF-A$^*_{noh}$, and DP, we define the term *state*. The #States for RF-A* and RF-A$^*_{noh}$ refers to the number of labels that are generated during the search. For DP, #States refers to the number of $A(v, k, q)$ whose cost values are computed during the DP iterations. As shown in Fig. 5(a) and 5(b), DP has a similar #States and has similar runtime across various instances. Compared to DP, RF-A* explores fewer states and needs less runtime, while RF-A$^*_{noh}$ needs less runtime but explores more states in the case of small graphs (e.g., Phil and Austin).

We use the word "the speed-up of an algorithm" to denote the ratio of the runtime of DP divided by the runtime of that algorithm. Fig. 5(c) reveals that most results of RF-A* have a speed-up between 2 to 8 times, compared to DP. The results of RF-A$^*_{noh}$ are distributed across a wider range, which means without the guidance of the heuristic, the search may expand many states that are useless to find an optimal solution. For the median values, RF-A$^*_{noh}$ shows a better performance than RF-A*, particularly with larger cities, e.g., London and Moscow, which suggests that the major contributor to RF-A*'s runtime is the heuristic computation, making RF-A* slower than RF-A$^*_{noh}$ for some instances. Finally, RF-A$^*_{cached}$ combines the benefit of having a heuristic and eliminates the overhead of computing a heuristic, which leads to the fastest approach and the speed-up of RF-A$^*_{cached}$ rises towards a range of 8 to 64.

To better understand the effectiveness of heuristic across different instances, we plot the speed-up factor of RF-A* and RF-A$^*_{cached}$, compared to RF-A$^*_{noh}$ with the increasing number of generated state of RF-A$^*_{noh}$, shown in Fig. 6. We can see that, with the guidance provided by the heuristic, RF-A* runs faster than RF-A$^*_{noh}$ in instances that require a lot of state generation, despite the overhead of computing the heuristic. This result indicates that the existing pre-processing techniques on a road network (e.g., [6], [15]) can be potentially applied to reduce the overhead of computing the heuristic to further expedite the search.

VI. Conclusion and Future Work

This paper investigates the GSP problem introduced in [11] and develops RF-A*, a fast A*-based algorithm that leverages the heuristic search and dominance pruning rules. Numerical results verify the advantage of RF-A* over the existing dynamic programming approach.

The limitation of our approach is that it relies on the assumption that all information about the graph and fuel cost are known in advance and remains unchanged, which may not hold in practice, e.g., the fuel price changes over time. For future work, one can introduce an additional time dimension [23] to RF-A* if the time-vary information is available. One can also integrate RF-A* into a predict-then-optimize framework [4] if the environment is not fully observable. Finally, one can also consider the multi-agent version of the problem [22].

References

[1] Rashid Alyassi, Majid Khonji, Areg Karapetyan, Sid Chi-Kin Chau, Khaled M. Elbassioni, and Chien-Ming Tseng. Autonomous Recharging and Flight Mission Planning for Battery-Operated Autonomous Drones. *IEEE Trans Autom. Sci. Eng.*, 20(2):1034–1046, 2023.

[2] Cedric De Cauwer, Wouter Verbeke, Joeri Van Mierlo, and Thierry Coosemans. A Model for Range Estimation and Energy-Efficient Routing of Electric Vehicles in Real-World Conditions. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[3] Giovanni De Nunzio, Ibtihel Ben Gharbia, and Antonio Sciarretta. A General Constrained Optimization Framework for the Eco-Routing Problem: Comparison and Analysis of Solution Strategies for Hybrid Electric Vehicles. *Transportation Research Part C: Emerging Technologies*, 123:102935, February 2021.

[4] Emir Demirovic, Peter J. Stuckey, Tias Guns, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Jeffrey Chan. Dynamic Programming for Predict+Optimise. In *The 34th AAAI*, 2020.

[5] Seyed Sajjad Fazeli, Saravanan Venkatachalam, and Jonathon M Smereka. Efficient Algorithms for Electric Vehicles' Min-Max Routing Problem. *Sustainable Operations and Computers*, 2024.

[6] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 2012.

[7] Edgar N Gilbert. Random Graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

[8] Shieu Hong Lin, Nate Gertsch, and Jennifer R. Russell. A Linear-Time Algorithm for Finding Optimal Vehicle Refueling Policies. *Operations Research Letters*, 35(3):290–296, May 2007.

[9] Panagiotis Karakostas and Angelo Sifaleras. The Pollution Traveling Salesman Problem with Refueling. *Comput. Oper. Res.*, 167:106661.

[10] Mohammadjavad Khosravi and Hossein Pishro-Nik. Unmanned Aerial Vehicles for Package Delivery and Network Coverage. In *2020 IEEE 91st Vehicular Technology Conference*, 2020.

[11] Samir Khuller, Azarakhsh Malekian, and Julián Mestre. To Fill or Not to Fill: The Gas Station Problem. *ACM Transactions on Algorithms*, 7(3):1–16, July 2011.

[12] Woojin Lee, Balsam Alkouz, Babar Shahzaad, and Athman Bouguettaya. Package Delivery Using Autonomous Drones in Skyways. In *International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the ACM International Symposium on Wearable Computers*. ACM, 2021.

[13] Longjiang Li, Haoyang Liang, Jie Wang, Jianjun Yang, and Yonggang Li. Online Routing for Autonomous Vehicle Cruise Systems with Fuel Constraints. *Journal of Intelligent & Robotic Systems*, 2022.

[14] Chung-Shou Liao, Shang-Hung Lu, and Zuo-Jun Max Shen. The Electric Vehicle Touring Problem. *Transportation Research Part B: Methodological*, 86:163–180, April 2016.

[15] Arthur Mahéo, Shizhe Zhao, Afzaal Hassan, Daniel D Harabor, Peter J Stuckey, and Mark Wallace. Customised Shortest Paths Using a Distributed Reverse Oracle. In *SOCS*, 2021.

[16] André L. C. Ottoni, Erivelton G. Nepomuceno, Marcos S. De Oliveira, and Daniela C. R. De Oliveira. Reinforcement Learning for the Traveling Salesman Problem with Refueling. *Complex & Intelligent Systems*, 2022.

[17] Kleitos Papadopoulos and Demetres Christofides. A Fast Algorithm for the Gas Station Problem. *Information Processing Letters*, 2018.

[18] Sepideh Pourazarm and Christos G. Cassandras. Optimal Routing of Energy-Aware Vehicles in Transportation Networks With Inhomogeneous Charging Nodes. *IEEE Transactions on Intelligent Transportation Systems*, 19(8):2515–2527, August 2018.

[19] Haripriya Pulyassary, Kostas Kollias, Aaron Schild, David B. Shmoys, and Manxi Wu. Network Flow Problems with Electric Vehicles. In Jens Vygen and Jaroslaw Byrka, editors, *IPCO 2024*, volume 14679 of *Lecture Notes in Computer Science*, pages 365–378. Springer, 2024.

[20] Subramanian Ramasamy, Jean-Paul Reddinger, James Dotterweich, Marshal A. Childers, and Pranav A. Bhounsule. Coordinated Route Planning of Multiple Fuel-constrained Unmanned Aerial Systems with Recharging on an Unmanned Ground Vehicle for Mission Coverage. *J. Intell. Robotic Syst.*, 2022.

[21] Zhongqiang Ren, Carlos Hernández, Maxim Likhachev, Ariel Felner, Sven Koenig, Oren Salzman, Sivakumar Rathinam, and Howie Choset. Emoa*: A framework for search-based multi-objective path planning. *Artificial Intelligence*, 339:104260, 2025.

[22] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. CBSS: A new approach for multiagent combinatorial path finding. *IEEE Transactions on Robotics*, 39(4):2669–2683, 2023.

[23] Zhongqiang Ren, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Multi-objective safe-interval path planning with dynamic obstacles. *IEEE Robotics and Automation Letters*, 7(3):8154–8161, 2022.

[24] Zhongqiang Ren, Zachary B. Rubinstein, Stephen F. Smith, Sivakumar Rathinam, and Howie Choset. ERCA*: A New Approach for the Resource Constrained Shortest Path Problem. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–12, 2023.

[25] Zhongqiang Ren, Richard Zhan, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Enhanced Multi-Objective A* Using Balanced Binary Search Trees. In *Proceedings of the International Symposium on Combinatorial Search*, 2022.

[26] Yoshinori Suzuki. A Generic Model of Motor-Carrier Fuel Optimization. *Naval Research Logistics (NRL)*, 55(8):737–746, 2008.

[27] Timothy M. Sweda, Irina S. Dolinskaya, and Diego Klabjan. Adaptive Routing and Recharging Policies for Electric Vehicles. *Transp. Sci.*, 51(4):1326–1348, 2017.

[28] Timothy M. Sweda, Irina S. Dolinskaya, and Diego Klabjan. Optimal Recharging Policies for Electric Vehicles. *Transportation Science*, 51(2):457–479, May 2017.