ERCA*: A New Approach for the Resource Constrained Shortest Path Problem

Zhongqiang Ren[1], Zachary B. Rubinstein[1], Stephen F. Smith[1], Sivakumar Rathinam[2] and Howie Choset[1]

***Abstract*—The Resource Constrained Shortest Path Problem (RCSPP) seeks to determine a minimum-cost path between a start and a goal location while ensuring that one or multiple types of resource consumed along the path do not exceed their limits. This problem is often solved on a graph where a path is incrementally built from the start towards the goal during the search. RCSPP is computationally challenging as comparing these partial solution paths is based on multiple criteria (i.e., the accumulated cost and resource along the path), and in general, there does not exist a single path that optimizes all criteria simultaneously. Consequently, the search needs to maintain and explore a large number of partial paths in order to find an optimal solution. While a variety of algorithms have been developed to solve RCSPP, they either have little consideration about efficiently comparing and maintaining the partial paths, which reduces their overall runtime efficiency, or are restricted to handle only one resource constraint as opposed to multiple resource constraints. This paper develops Enhanced Resource Constrained A* (ERCA*), a fast A*-based algorithm that can find an optimal solution while satisfying multiple resource constraints. ERCA* leverages both the recent advances in multi-objective path planning to efficiently compare and maintain partial paths, and techniques from the existing RCSPP literature. Furthermore, ERCA* has a functional parameter to broker a trade-off between solution quality and runtime efficiency. The results show ERCA* often runs several orders of magnitude faster than an existing leading algorithm for RCSPP.**

***Index Terms*—Path Planning, Heuristic Search, Shortest Path**

I. Introduction

THE Resource Constrained Shortest Path Problem (RCSPP) seeks to find a minimum cost path between a start and goal location without depleting a set of resources. This problem arises in many applications such as air cargo transportation [7], railway management [11] and airline crew scheduling [30]. RCSPP is often formulated as a graph search problem (Fig. 1), where each edge in the graph is associated with both a scalar cost value and a resource vector, which indicate the cost incurred and the resources consumed respectively if the edge is included into the solution path. RCSPP is an NP-Hard problem to solve to optimality (i.e., finding a minimum cost path from the start to goal while satisfying all resource constraints), even when considering only a single resource constraint [12]. To solve RCSPP, most of the recent approaches [1], [5], [16], [33] search the graph to incrementally build a solution path, and a fundamental challenge is to efficiently maintain a large number of partial solution paths during the search. Specifically, solving RCSPP to optimality requires comparing partial paths with respect to both the accumulated cost and the consumed resource vector along the partial paths during the search. As a result, the notion of dominance from multi-objective optimization [9] arises and is used to check if one partial path is better than another. There are often a large number of incomparable (i.e., non-dominated) partial solution paths from the start vertex to any other vertex in the graph. To find an optimal solution for RCSPP, an algorithm needs to maintain and explore these non-dominated partial paths, and conduct a lot of dominance checks, which makes RCSPP computationally burdensome.

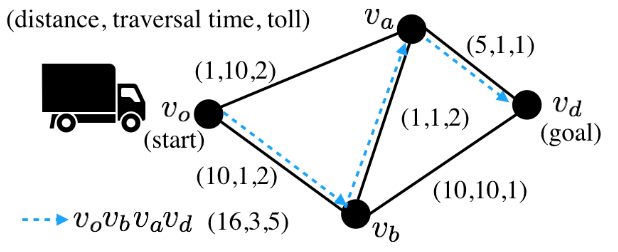


Fig. 1. A toy example of RCSPP. Given a graph where each edge is associated with three non-negative values: distance, traversal time and toll, the problem seeks to find a minimum distance path from the start to the goal while satisfying two resource constraints, i.e, the arrival time must be less than three and the accumulated toll must be less than six. The blue dotted lines show an optimal solution path to the problem.

RCSPP has been studied for decades [12], [16], [33] and remains an active research topic [1], [5]. On the one hand, many existing methods focus on developing either pruning techniques to eliminate infeasible partial paths as early as possible during the search, or techniques that can quickly lower the primal bound (i.e., the cost of the best feasible solution found thus far during the computation). However, they have little consideration about efficiently comparing and maintaining the non-dominated partial solution paths, which reduces their runtime efficiency. On the other hand, some recent approaches [1] leverage fast dominance check techniques and are able to expedite the computation for several orders of magnitude. However, these methods are restricted to handling a single resource constraint. This paper develops Enhanced Resource Constrained A* (ERCA*), a fast planner that can solve

Manuscript received March, 25, 2023; Revised June, 9, 2023; Accepted June 30, 2023. This work was supported by National Science Foundation under Grant No. 2120219 and 2120529. The Associate Editor for this article was TODO. *(Corresponding author: Zhongqiang Ren.)*

Zhongqiang Ren, Zachary B. Rubinstein, Stephen F. Smith and Howie Choset are with Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA. (email: zhongqir@andrew.cmu.edu; zbr@cs.cmu.edu; sfs@cs.cmu.edu; choset@andrew.cmu.edu).

Sivakumar Rathinam is with Texas A&M University, College Station, TX 77843-3123. (email: srathinam@tamu.edu).

Digital Object Identifier (DOI):

RCSPP to optimality subject to multiple resource constraints.

The development of ERCA* leverages both the recent advances in multi-objective A* [14], [23], [27] and the existing techniques in the RCSPP literature [5], [16]. Specifically, ERCA* leverages the recent advances in multi-objective A* [14], [23], [27] to efficiently maintain the non-dominated partial paths to achieve fast dominance check of partial paths. The ability to do fast dominance check in the presence of multiple resource constraints distinguishes ERCA* from the existing methods for RCSPP. Additionally, ERCA* inherits some popular pre-processing and partial path pruning techniques developed in the RCSPP literature [5], [16] to remove the partial paths, which are predicted to exceed the resource limits before reaching the goal vertex, from the search as early as possible. Furthermore, we also develop a variant of ERCA* that can trade-off solution quality for runtime efficiency, by quickly finding a bounded sub-optimal solution without violating any resource constraints.

We compare ERCA* against BiPulse [5], an existing leading algorithm for RCSPP, with two and three resource constraints in city-like road networks from a online data set. Our experimental results show that ERCA* often runs several orders of magnitude faster than BiPulse, and the bounded sub-optimal variant of ERCA* can quickly find a 10% sub-optimal solution in graphs with more than a million vertices subject to three resource constraints. We open source our software to facilitate the practitioners and researchers in the community.[1]

We summarize the contributions of this article as follows.

- This article develops a new algorithm called ERCA* that solves RCSPP to optimality. ERCA* is able to expedite the computation by bringing together the existing RCSPP techniques and the recent multi-objective search techniques within the same framework.
- We also develop a variant of ERCA* that can trade-off solution quality for runtime efficiency, by quickly finding a bounded sub-optimal solution without violating any resource constraints.
- We test ERCA* on city-like maps from a public dataset. ERCA* runs faster than the existing BiPulse for up to several orders of magnitude. The variant of ERCA* can find a bounded sub-optimal path in graphs with more than a million vertices subject to three resource constraints.

The rest of this paper is organized as follows. Sec. II reviews the related work and Sec. III describes the problem. We then present ERCA* in Sec. IV and discuss its relationship to the existing algorithms in Sec. V. The experimental results are shown in Sec. VI, and the conclusion is presented in Sec. VII.

II. Related Work

A. Resource Constrained Shortest Path Problem

Methods for RCSPP include both exact approaches [5], [16], [22], [33] (i.e., approaches that can solve RCSPP to optimality) and approximation algorithms [13], [18], [34], and this article focuses on the exact approaches. Conventional exact algorithms for RCSPP include dynamic programming based algorithms [8], Lagrangian relaxation [3], [28], path ranking approaches [29], etc., and a survey of these early effort can be found in [22]. Subsequently, an algorithm called Pulse [16] was developed, which takes a depth-first search (DFS) strategy to enumerate the paths from the start to any other vertices in the graph while employing several rules to prune partial paths during the search. With DFS, Pulse attempts to quickly find a feasible solution whose cost provides a primal bound (i.e., upper bound) of the optimum, and then keeps refining the bound during the search until the optimum is found. After the development of Pulse, an algorithm called Resource Constrained Bi-Directional A* (RCBDA*) [33] was developed, which outperforms Pulse. RCBDA* simultaneously runs a forward (from the start to goal) and a backward (from the goal to start) search while ensuring both searches do not exceed 50% of a user-selected resource limit so that both searches "meet in the middle". Upon the success of RCBDA*, the idea of bi-directional search is then combined with Pulse and the resulting Bidirectional Pulse (BiPulse) [5] shows faster computational speed than RCBDA* in many instances. For RCSPP with an arbitrary number of resource limits, BiPulse remains a leading algorithm and is selected as a baseline in this work for comparison. Different from these existing methods for RCSPP, the prominent feature of ERCA* is its ability to employ several fast dominance check techniques from the recent advances in multi-objective A*, which can expedite the search. Additionally, ERCA* does not leverage the idea of bi-directional search, and how to combine bi-directional search with ERCA* remains an open question and is left as our future work.

Weight Constrained Shortest Path Problem (WCSPP), as a special case of RCSPP, considers a single resource constraint, which is often referred to as the weight (of a path) [22]. Any algorithm that can solve RCSPP is also applicable to WCSPP. Recently, several new algorithms [1] were developed to solve WCSPP by leveraging the bi-criteria nature of WCSPP (i.e., dealing with the cost and weight of the paths). These algorithms leverage the techniques from both WCSPP literature and bi-objective A* search [14] (such as the fast dominance check technique but limited to bi-criteria), and is thus able to achieve several orders of magnitude shorter runtime in comparison with its predecessors such as BiPulse [5] and RCBDA* [33]. However, these algorithms are limited to handle a single resource constraint.

While this article limits its focus to graphs with non-negative edge cost and resources, some recent variants of RCSPP consider negative edge cost and resources. These variants arise when planning paths for vehicles with the possibility to recharge or refuel the vehicle along the paths [6], [19], [21], [32], which is part of our future work.

B. Multi-Objective Path Finding

Of close relevance to RCSPP, Multi-Objective Path Finding (MO-PF) considers a graph, where each edge is associated with a cost vector (as opposed to a scalar cost value), and each component of the vector corresponds to an objective to be optimized. MO-PF seeks to find paths that minimize

[1]https://github.com/wonderren/public_erca

the accumulated cost vectors along the paths. With multiple objectives, there is in general no single solution path that can simultaneously optimize all the objectives [9], [31], and MO-PF thus seeks to find a set of so-called Pareto-optimal solutions. Recently, several efficient algorithms [14], [27] have been developed for MO-PF as well as its variants [24], [26].

A common computational challenge in MO-PF and RCSPP is to address a large number of non-dominated partial solution paths during the search, which has been shown to be a key computational bottleneck in MO-PF [14], [23], [27]. To address this challenge, several techniques have been developed in the literature, which are briefly summarized here and revisited in detail in Sec. IV-C. The so-called dimensionality reduction technique [23] is able to ignore the first component of the cost vectors during the dominance check which alleviates the computational burden. With a focus on bi-objective problems, Bi-Objective A* [14] inherits the dimensionality reduction technique and further develop the idea of lazy check, which re-organizes the workflow of the search algorithm and can further speed up all dominance checks during the search. Subsequently, our prior work Enhanced Multi-Objective A* (EMOA*) [27] is able to generalize the fast dominance check techniques in Bi-Objective A* to an arbitrary number of objectives by employing balanced binary search tree data structure to expedite the dominance check.

Recently, Bi-Objective A* [14] has been combined with the idea of bi-directional search and results in the algorithm Bi-Objective Bi-directional A* [2], which then lead to WCBA* [1], a leading algorithm for WCSPP that runs several orders of magnitude faster than the previous methods for WCSPP. However, WCBA* is limited to handle a single resource constraint.

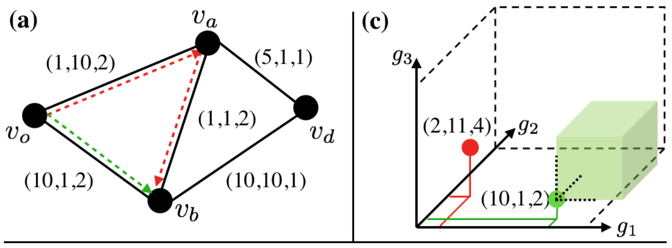
III. Problem Definition

Let $G = (V, E)$ denote a finite undirected graph with the vertex set V representing the possible locations, and the edge set $E \subseteq V \times V$ denoting the transition between any two locations. Each edge in the graph is associated with a scalar non-negative *cost value* $c(e) \in \mathbb{R}^+$, and a non-negative *resource vector* $\vec{r}(e) \in (\mathbb{R}^+)^M$ with M being a positive integer denoting the number of different resources. Let v_o and v_d denote the initial vertex (i.e., origin) and the destination vertex respectively. Let $\pi(v_1, v_\ell) := \{v_1, v_2, \cdots, v_\ell\}$ denote a path that consists of a list of vertices with each pair of adjacent vertices $v_k, v_{k+1}, k \in \{1, 2, \cdots, \ell-1\}$ connected by an edge $(v_k, v_{k+1}) \in E$. We refer to $\pi(v_1, v_\ell)$ simply as π when there is no confusion. For a path $\pi(v_1, v_\ell)$, let $c(\pi) := \sum_{k=0}^{k=\ell-1} c(v_k, v_{k+1})$ represent the *path cost*, which is the accumulative cost of the edges that are present in the path π. Similarly, let $\vec{r}(\pi) := \sum_{k=0}^{k=\ell-1} \vec{r}(v_k, v_{k+1})$ denote the *path resource vector*, which describes the total amount of resource consumed when moving from v_1 to v_ℓ along π. Finally, let $\vec{r}_{limit} \in (\mathbb{R}^+)^M$ denote the *resource limits*, a non-negative M-dimensional vector.

The goal of the Resource Constrained Shortest Path Problem (RCSPP) is to find a path π from v_o to v_d such that (i) $\vec{r}(\pi) \leq \vec{r}_{limit}$, which means every component in $\vec{r}(\pi)$ is no larger than

Notation	Meaning
$\pi(u,v)$	A path between u and v with $u, v \in V$.
$c(\cdot)$	A non-negative scalar cost value.
$\vec{r}(\cdot)$	A M-d non-negative resource.
$\vec{g}(\cdot)$	A $(M+1)$-d augmented cost vector.
$\vec{h}(\cdot)$	A $(M+1)$-d heuristic vector.
l	A label.
$\vec{f}(l)$	A $(M+1)$-d vector that is equal to $\vec{g}(l) + \vec{h}(l)$.
Trunc$(\cdot)$	The truncation function.
$\preceq$	Dominance.
$<_{lex}$	Lexicographically smaller than.
$\mathcal{F}(u)$	The frontier set at vertex $u \in V$.
$\mathcal{ND}(u)$	The non-dominated subset of Trunc$(\mathcal{F}(u))$.
$\mathcal{T}_{\mathcal{ND}}(u)$	The balance binary search tree of $\mathcal{ND}(u)$.

TABLE I
Frequently used notations.



(b) $\mathcal{F}(v_b) = \{l_1, l_2\}$

Non-dominated paths to v_b	Labels	Augmented Cost Vector $\vec{g}$
$v_o v_a v_b$	l_1	🔴 (2,11,4)
$v_o v_b$	l_2	🟢 (10,1,2)

Fig. 2. Visualization of related concepts. (a) shows a RCSPP example with two resources ($M = 2$) and the augmented cost vector $\vec{g}(e)$ for each edge e. Two non-dominated paths from v_o to v_b are shown in red and green dashed lines. (b) shows the detail about the two non-dominated paths from v_o to v_b. The labels representing these two paths are l_1 and l_2, and they are both in the frontier set at v_b. (c) shows the space of augmented cost vectors and the two augmented cost vectors mentioned in (b) using the red and green dots. The green cubic volume illustrates the set of vectors that are dominated by vector $(10, 1, 2)$.

the corresponding component in $\vec{r}_{limit}$; and (ii) $c(\pi)$ reaches the minimum.

Remark 1. *Weight Constrained Shortest Path Problem (WCSPP) is a special case of RCSPP with $M = 1$. In WCSPP, $\vec{r}_{limit}$ is a vector of length one, and there is only one resource limit, which is called the weight limit.*

IV. Method

A. Concepts and Notations

Table I summarizes the frequently used notations and Fig. 2 visualizes some of the concepts. We refer to a path π from v_o to any other vertex $v \in V$ that satisfies the resource constraints as a *feasible* path. A feasible path connecting v_o and v_d is called a *solution* path. Let Trunc : $\mathbb{R}^K \to \mathbb{R}^{K-1}$ (with K being an integer that is no less than two) denote the *truncation function* that removes the first component from the input vector. Let $\vec{g}(\pi) := (c(\pi), \vec{r}(\pi)) \in (\mathbb{R}^+)^{M+1}$ denote an *augmented cost vector*, whose first component is the path cost and Trunc$(\vec{g}(\pi))$ is the path resource vector $\vec{r}(\pi)$. Without confusion, let $\vec{g}(e) := (c(e), \vec{r}(e)) \in (\mathbb{R}^+)^{M+1}$ denote the augmented cost vector of an edge e in the graph G. In this paper, we use subscript to denote a specific component of a vector (e.g. $g_k(e), k = 1, 2, \cdots, M+1$ denotes the k-th component of $\vec{g}(e)$).

In RCSPP, there can be multiple "incomparable" feasible paths from v_o to any other vertex $v \in V$. To differentiate between these incomparable paths, the notion of label is introduced. Let $l = (v, \vec{g})$ denote a *label*[2], which is a tuple of a vertex $v \in V$ and an augmented cost vector $\vec{g}$. Intuitively, a label represents a path from v_o to v with the augmented cost vector $\vec{g}$ (Fig. 2(a) and 2(b)). To simplify notations, given a label l, let $v(l)$ and $\vec{g}(l)$ denote the vertex and the augmented cost vector related to label l, respectively. To compare two labels, we compare the augmented cost vectors related to the labels using the notion of dominance and lexicographic order.

Definition 1 (Dominance)**.** *Given two vectors $\vec{a}$ and $\vec{b}$ of length K ($K \geq 2$), $\vec{a}$ dominates $\vec{b}$ (denoted as $\vec{a} \preceq \vec{b}$)[3] if and only if $\forall m \in \{1, 2, \ldots, M\}$, $a_m \leq b_m$, and $\exists m \in \{1, 2, \ldots, M\}$, $a_m < b_m$.*

Fig. 2(c) visualizes the notion of dominance. If $\vec{a}$ does not dominate $\vec{b}$, this non-dominance is denoted as $\vec{a} \not\preceq \vec{b}$. Any two paths $\pi_1(v_1, v_\ell), \pi_2(v_1, v_\ell)$, for two vertices $v_1, v_\ell \in V$, are non-dominated (with respect to each other) if the corresponding augmented cost vectors $\vec{g}(\pi_1)$ and $\vec{g}(\pi_2)$ do not dominate each other.

Definition 2 (Lexicographic Order)**.** *Given two vectors $\vec{a}$ and $\vec{b}$ of length K ($K \geq 2$), $\vec{a}$ is lexicographically smaller (or larger) than $\vec{b}$, which is denoted as $\vec{a} <_{lex} \vec{b}$ (or $\vec{a} >_{lex} \vec{b}$) if there exists a $k \in \{1, 2, \cdots, K\}$, $a_k < b_k$ (or $a_k > b_k$), and $\forall m \in \{1, 2, \ldots, k-1\}$, $a_m = b_m$.*

As its name suggests, the lexicographic (hereafter abbreviated as lex.) order begins by comparing the first component of the vectors $\vec{a}$ and $\vec{b}$ to determine if $\vec{a}$ is lex. smaller or larger than $\vec{b}$, and it only needs to compare the next component if the current component of $\vec{a}$ and $\vec{b}$ are the same.

Correspondingly, a label l is said to be dominated by (or is equal to) another label l' if $v(l) = v(l')$ and $\vec{g}(l) \preceq \vec{g}(l')$ (or $\vec{g}(l) = \vec{g}(l')$). A label l is lex. smaller (or larger) than another label l' if $v(l) = v(l')$ and $\vec{g}(l) <_{lex} \vec{g}(l')$ (or $\vec{g}(l) >_{lex} \vec{g}(l')$).

Let $\vec{h}(v) \in (\mathbb{R}^+)^{M+1}, v \in V$ denote a heuristic vector of vertex v that estimates the "cost-to-go" and "resource-to-go" from v to v_d. Specifically, the first component in $\vec{h}(v)$ is an estimate of the path cost from v to v_d while $\text{Trunc}(\vec{h}(v))$ is an estimate of the path resource vector from v to v_d. If $\vec{h}(v), v \in V$ is component-wise no larger than the augmented cost vector of any possible paths from v to v_d, $\vec{h}$ is then an *admissible* heuristic. If a heuristic satisfies $\vec{h}(v) \leq \vec{h}(u) + \vec{c}(u, v), \forall u, v \in V$, then $\vec{h}$ is a *consistent* heuristic. A consistent heuristic is always admissible. The algorithm developed in this paper requires that the heuristic is consistent, which will be revisited later.

[2]To identify a path, different names such as nodes [14], states [25] and labels [5], have been used in the multi-objective path-planning literature. This work uses "labels" to identify paths, and reserves "nodes" for the tree nodes in the balanced binary search tree in the ensuing section.

[3]In the literature, another symbol $\succeq$ is also widely used to denote the dominance relation between two vectors. We choose to use $\preceq$ in this work since (i) the goal here is to minimize costs, and (ii) $\preceq$ is visually similar to $\leq$ (no larger than) and is more intuitive to read.

Algorithm 1 ERCA*

```
1:  Backwards Dijkstra search to compute $\vec{h}(v), \forall v \in V$.
2:  $l_o \leftarrow (v_o, \vec{0})$, $\vec{f}(l_o) \leftarrow \vec{0} + \vec{h}(v_o)$
3:  Add $l_o$ to OPEN
4:  $\mathcal{F}(v) \leftarrow \emptyset, \forall v \in V$
5:  while OPEN $\neq \emptyset$ do
6:      $l \leftarrow$ OPEN.pop()                          ▷ Label extracted
7:      if IsPrunByFront($l$) or IsPrunByResour($l$) then
8:          continue                              ▷ Current iteration ends
9:      FilterAndAddFront($l$)
10:     if $v(l) = v_d$ then
11:         break                                ▷ The while loop ends
12:     for all $v' \in$ GetNgh($v(l)$) do              ▷ Label expanded
13:         $l' \leftarrow (v', \vec{g}(l) + \vec{c}(v, v'))$
14:         $parent(l') \leftarrow l$
15:         $\vec{f}(l') \leftarrow \vec{g}(l') + \vec{h}(v(l'))$
16:         if IsPrunByFront($l'$) or IsPrunByResour($l'$) then
17:             continue                ▷ Move to the next neighbor.
18:         OPEN.insert($l'$)
19: return Reconstruct($v_d$)
```

Additionally, let $\vec{f}(l) := \vec{g}(l) + \vec{h}(v(l))$ denote the f-vector of a label l. Intuitively, $\vec{f}(l)$ provides a lower bound of the path cost and resources to reach v_d from v_o by extending the path represented by l. Let OPEN denote a priority queue of labels, where labels are prioritized by their corresponding $\vec{f}$-vectors in the lex. order from the minimum to the maximum. Finally, let $\mathcal{F}(u), u \in V$ denote the *frontier* set *at* vertex u, which stores all non-dominated labels l *at* vertex u (i.e., $v(l) = u$). Intuitively, each label $l \in \mathcal{F}(u), u \in V$ identifies a non-dominated path from v_o to u (Fig. 2(b)).

B. ERCA Algorithm Overview*

Our algorithm ERCA* is shown in Alg. 1. It begins with a pre-processing step which runs exhaustive backwards Dijkstra search from v_d to all other vertices in the graph to compute the heuristic. Specifically, a new graph $G'_k, k = 1, 2, \cdots, M+1$ with the same set of vertices and edges as G is created. For each edge e' (or vertex v') in G'_k, let e (or v) denote the corresponding edge (or vertex) in G. Then, e' is associated with a scalar cost value $c'(e') := g_k(e)$, i.e., the k-th component of the augmented cost vector of that edge. A Dijkstra search is conducted from v'_d to all other vertices v' in G'_k so that an optimal path π' from v' to v'_d is computed. The cost value of π' in G' provides the heuristic value $h_k(v)$, which is the k-th component of the heuristic vector $\vec{h}(v)$. After these Dijkstra searches, the heuristic vector of all vertices in G are known.

After computing the heuristic, ERCA* creates an initial label $l_o = (v_o, \vec{0})$ at vertex v_o with $\vec{f}(l_o) = \vec{h}(v(l_o))$, and inserts l_o into OPEN. Additionally, the frontier set $\mathcal{F}(v)$ at any vertex $v \in V$ is initialized as an empty set. In each iteration of the search (Lines 6-18), a label with the lex. minimum f-vector is *extracted* from OPEN for processing.

The extracted label l is first checked for dominance against the existing labels in the frontier set (Line 7) using the procedure *IsPrunByFront*, where $\vec{g}(l)$ is checked for dominance against the g-vector of the existing labels in $\mathcal{F}(v(l))$. The realization of *IsPrunByFront* is elaborated in Sec. IV-C3, which is an important procedure that affects the computational efficiency of the search. If the label l is not pruned by dominance in *IsPrunByFront*, l is then checked against the resource limits in *IsPrunByResour* in Line 7. Specifically, *IsPrunByResour* compares the $\text{Trunc}(\vec{f}(l))$ against the resource limits $\vec{r}_{limit}$ (both vectors are of length M). Since $\text{Trunc}(\vec{f}(l))$ is a lower bound of the required resources to reach v_d by extending the path represented by label l, if any component of $\text{Trunc}(\vec{f}(l))$ is greater than the corresponding component of $\vec{r}_{limit}$, label l cannot lead to a feasible path to reach v_d and is thus pruned.

If the extracted label l from OPEN is not pruned in *IsPrunByFront* and *IsPrunByResour*, l is then added to the frontier set $\mathcal{F}(v(l))$ in *FilterAndAddFront* in Line 9. Specifically, *FilterAndAddFront* first removes any existing labels in $\mathcal{F}(v(l))$ that are dominated by l, and then adds l to $\mathcal{F}(v(l))$. The actual realization of *FilterAndAddFront* is detailed in Sec. IV-C3. After *FilterAndAddFront*, ERCA* verifies if $v(l)$ reaches v_d. If $v(l) = v_d$, label l is guaranteed to represent a minimum cost solution path and the entire search terminates (Line 11). Otherwise (i.e., $v(l) \neq v_d$), label l is *expanded* by generating new labels at all neighboring vertices of $v(l)$ in G. Specifically, for each adjacent vertex v' of v (Line 12), a new label $l' \leftarrow (v', \vec{g}(l) + \vec{c}(v, v'))$ is created and then checked for pruning using *IsPrunByFront* and *IsPrunByResour*. If l' is not pruned, l' is added to OPEN for future expansion.

At the end of the search, the procedure *Reconstruct* either returns a solution path by iteratively tracking the parent pointers of labels from v_d to v_o, or returns failure if v_d is never reached by any label during the search. When failure is returned, it indicates that the given RCSPP instance is infeasible (i.e., unsolvable).

C. Fast Dominance Check

The frontier set $\mathcal{F}(v)$ at any vertex $v \in V$ may contain a large number of non-dominated labels, especially when there are multiple resource limits (i.e., M is large). Therefore, the realization of *IsPrunByFront* and *FilterAndAddFront* can affect the overall runtime of Alg. 1 since both procedures are repetitively invoked during the search. To expedite the search, ERCA* employs several techniques from the multi-objective path planning literature.

1) Lazy Dominance Check: The first technique employed in ERCA* is referred to as *lazy dominance check* [14]. When a new label l' is generated and added to OPEN (Line 18 in Alg. 1), the algorithm can either conduct "eager" dominance check by immediately running dominance comparison to find and remove any existing label l'' in OPEN that is at the same vertex as $v(l')$ (i.e., $v(l'') = v(l')$) and is dominated by l', or conduct "lazy" dominance check by deferring the check and the removal of l'' until l'' is extracted from OPEN (Line 7 in Alg. 1). It has been shown that running lazy check is more runtime efficient for the overall search than running eager check [14]. ERCA* thus chooses to run lazy check: at Line 7 in Alg. 1, a label is checked for dominance and discarded after it is extracted from OPEN and before being expanded.

(a)

$$\mathcal{F}(u) = \{l_1, l_2, l_3\}$$
$$l_1 = (3, 9, 6)$$
$$l_2 = (4, 8, 7)$$
$$l_3 = (5, 6, 7)$$
$$\text{Trunc}(\mathcal{F}(u)) = \{(9,6), (8,7), (6,7)\}$$
$$\mathcal{ND}(u) = \{(9,6), (6,7)\}$$

(b)

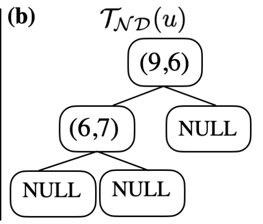


Fig. 3. Visualization of $\mathcal{F}(u)$, $\text{Trunc}(\mathcal{F}(u))$ and $\mathcal{ND}(u)$ for some vertex $u \in V$. (a) shows the frontier set at u containing three labels l_1, l_2 and l_3, and the corresponding $\text{Trunc}(\mathcal{F}(u))$ and $\mathcal{ND}(u)$. (b) shows the AVL-tree corresponding to $\mathcal{ND}(u)$.

2) Dimensionality Reduction: The second technique employed in ERCA* for fast dominance check is the so-called "dimensionality reduction" [23]. With (i) a consistent heuristic and (ii) an OPEN list where labels are prioritized using the lex. order, the first component of the g-vectors corresponding to labels that are extracted from OPEN are guaranteed to be monotonically non-decreasing. The first component of the g-vectors can thus be ignored in the dominance check, and only the corresponding truncated vectors are needed for comparison, which saves computational effort. Formally speaking, let $\mathcal{L} = l_1, l_2, \cdots, l_K$ denote the sequence of labels that are extracted from OPEN during the search. Given the aforementioned condition (i) and (ii), it is guaranteed that the first component of $\vec{g}(l_k), k = 1, 2, \cdots, K$ in $\mathcal{L}$ are monotonically non-decreasing. During the search process of Alg. 1, for any two labels l_j, l_k at the same vertex (i.e., $v(l_k) = v(l_j)$) with l_k being extracted from OPEN after l_j (i.e., $k > j$), $\vec{g}(l_k)$ is dominated (or non-dominated) by $\vec{g}(l_j)$ if and only if $\text{Trunc}(\vec{g}(l_k))$ is dominated (or non-dominated) by $\text{Trunc}(\vec{g}(l_j))$. ERCA* leverages this idea in the procedures *IsPrunByFront* and *FilterAndAddFront*, which is elaborated in the next sub-section.

3) BBST-based IsPrunByFront and FilterAndAddFront: The third technique employed by ERCA* to achieve fast dominance check is using a balanced binary search tree (BBST) to organize the truncated vectors corresponding to labels in $\mathcal{F}(u)$ for each vertex $u \in V$, so that there is no need to iterate every truncated vector for dominance check [27] and only a subset of these truncated vectors need to be considered. This section shows this BBST-based technique and the realization of *IsPrunByFront* and *FilterAndAddFront* in ERCA*.

Given a label l, the goal of *IsPrunByFront* is to determine whether $\vec{g}(l)$ is dominated by $\vec{g}(l')$ for any existing label $l' \in \mathcal{F}(v(l))$. With the aforementioned dimensionality reduction technique, *IsPrunByFront* only needs to compare $\text{Trunc}(\vec{g}(l))$ against $\text{Trunc}(\vec{g}(l'))$ for every existing label $l' \in \mathcal{F}(v(l))$. Let $\text{Trunc}(\mathcal{F}(u)) := \{\text{Trunc}(\vec{g}(l')), l' \in \mathcal{F}(u)\}$ denote the set of truncated vectors of the g-vectors corresponding to labels in $\mathcal{F}(u), u \in V$, and let $\mathcal{ND}(u), u \in V$ denote the non-dominated subset of $\text{Trunc}(\mathcal{F}(u))$, where any two vectors in $\mathcal{ND}(u)$ are non-dominated with respect to each other. Fig. 3 provides an illustration of $\mathcal{F}(u)$, $\text{Trunc}(\mathcal{F}(u))$

Algorithm 2 *IsPrunByFront*(n, l)

```
    INPUT: $n$ is a node in an AVL-tree and $l$ is a label
 1: $a \leftarrow \mathrm{Trunc}(\vec{g}(l))$
 2: if $n = NULL$ then
 3:     return false
 4: if $n \succeq a$ or $n = a$ then
 5:     return true
 6: if $a <_{\mathrm{lex}} n$ then
 7:     return IsPrunByFront($n.left, l$)
 8: else                                          ▷ i.e., $a >_{\mathrm{lex}} n$
 9:     if IsPrunByFront($n.left, l$) then        ▷ Removed if $M = 2$
10:         return true                           ▷ Removed if $M = 2$
11:     return IsPrunByFront($n.right, l$)
```

Algorithm 3 *Filter*(n, l)

```
    INPUT: $n$ is a node in an AVL-tree and $l$ is a label
 1: $a \leftarrow \mathrm{Trunc}(\vec{g}(l))$
 2: if $n = NULL$ then
 3:     return
 4: if $a >_{\mathrm{lex}} n$ then
 5:     $n.right \leftarrow$Filter($n.right, a$)
 6: else
 7:     $n.left \leftarrow$Filter($n.left, a$)
 8:     $n.right \leftarrow$Filter($n.right, a$)
 9: if $a \succeq n$ then
10:     Delete $n$ from the tree
11:     return
    Note: the tree needs to be re-balanced after the entire
    filtering process.
```

and $\mathcal{ND}(u)$. With an input label l, a simple realization of *IsPrunByFront* runs a for-loop to iterate all $\vec{b} \in \mathcal{ND}(u)$ and checks if the given $\mathrm{Trunc}(\vec{g}(l))$ is dominated by or equal to $\vec{b}$, which has $O(M|\mathcal{ND}(v(l))|)$ worst-case runtime complexity. We hereafter refer to this realization simply as the "for-loop" realization, which is a baseline in our tests (Sec. VI).

To expedite the computation, ERCA* uses a BBST (and more specifically an AVL-tree in this paper) $\mathcal{T}_{\mathcal{ND}}(u)$ to represent $\mathcal{ND}(u)$ for each vertex $u \in V$, where (i) each node in $\mathcal{T}_{\mathcal{ND}}(u)$ is a vector in $\mathcal{ND}(u)$, (ii) for any node n in $\mathcal{T}_{\mathcal{ND}}(u)$, its left child node $n.left$ is lex. smaller than n while its right child node $n.right$ is lex. larger than n. As shown in Alg. 2, *IsPrunByFront* is invoked with two input arguments, where n is the root node of $\mathcal{T}_{\mathcal{ND}}(v(l))$ and l is the label to be checked for dominance. Alg. 2 then recursively traverses the $\mathcal{T}_{\mathcal{ND}}(u)$ for dominance check as follows. The truncated vector $\mathrm{Trunc}(\vec{g}(l))$ of the input label l (denoted as a in Alg. 2) is first compared with the current node n, and there are two cases:

- If n is empty (i.e., a $NULL$ pointer), it means that a leaf node of the tree is reached and a is non-dominated (Line 2).
- If n is not empty, n is compared with a and the procedure returns true if a is dominated by or is equal to n.

When none of these two cases hold, then *IsPrunByFront* starts to look at the children based on whether a is lex. smaller or larger than n, which leads to the following two cases.

- If a is lex. smaller than n, then only the left child (i.e., the left sub-tree) of n needs to be further traversed for dominance checks (Line 7), because no node in the right sub-tree can dominate a [27]. This is the case where the computational effort is saved in comparison with the aforementioned for-loop realization, since a subset of the vectors are skipped during the tree traversal.
- If a is lex. larger than n, then both the left and the right sub-tree of n needs to be traversed for dominance checks (Line 9 and 11).

This BBST-based realization has $O(M|\mathcal{ND}(v(l))|)$ runtime complexity, which is the same as the for-loop realization. However, in practice, the BBST-based realization can significantly expedite the dominance check [27].

There are two special cases. When $M = 1$, the key of each node in the BBST becomes a vector of length one (i.e., a scalar value) and the BBST always has a single node (which is the root node). In this case, the dominance check becomes the comparison of two scalars, which can be done in constant time. This case has been extensively studied in the Bi-Objective A* algorithm [14]. Another special case is when $M = 2$, the theoretic runtime complexity of this BBST-based realization can be further reduced from linear to log time, by removing Line 9 and 10 from Alg. 2. Specifically, it is guaranteed that when a is lex. larger than n, no node in the left sub-tree can dominate a due to both the fact that $M = 2$ and the way in which tree is constructed [27]. Consequently, at each node in the tree, Alg. 2 goes to either the left or the right child, which leads to a runtime complexity that is linear with respect to the height of the tree. Since the tree is balanced, the height of tree is $O(\log|\mathcal{ND}(v(l))|)$. Therefore, the overall runtime complexity of Alg. 2 when $M = 2$ is $O(\log|\mathcal{ND}(v(l))|)$.

We now present the BBST-based *FilterAndAddFront*. Given a label l, *FilterAndAddFront* first uses l to filter any existing nodes in the tree $\mathcal{T}_{\mathcal{ND}}(v(l))$ and then adds $\mathrm{Trunc}(\vec{g}(l))$ into the tree. Adding a node into a BBST is a standard operation, where the tree is rotated if needed in order to maintain the balance of the tree. We now focus on the filtering step which is presented in Alg. 3.

To filter the tree with a non-dominated label l, Alg. 3 is invoked with two input arguments, where n is the root node of $\mathcal{T}_{\mathcal{ND}}(v(l))$ and l is the label to be used to filter the tree. Note that since Alg. 3 is always invoked after Alg. 2 in ERCA*, the truncated vector $\mathrm{Trunc}(\vec{g}(l))$ (denoted as a in Alg. 3) is thus guaranteed to be non-dominated by any existing vectors in the tree. If the input node is $NULL$ (Line 2), the algorithm terminates and returns. When the input node n is not $NULL$, the algorithm verifies whether $a >_{\mathrm{lex}} n$.

- If $a >_{\mathrm{lex}} n$, there is no need to consider the left sub-tree of n for filtering, since any node in the left sub-tree of n must be non-dominated by a. The algorithm thus recursively invokes itself to traverse only the right sub-tree for filtering. Skipping the left sub-tree is the reason that computational effort is saved in comparison with the aforementioned for-loop realization.
- Otherwise (i.e., $a <_{\mathrm{lex}} n$), both the left and the right

sub-trees need to be filtered (Line 7 and 8).

At the end (Line 10), n is checked for dominance against a, and n is removed from the tree if n is dominated by a. After invoking Alg. 3, if nodes are deleted, the tree can become "highly unbalanced", i.e., the height difference between the left sub-tree and right sub-tree of a node n can be greater than two. In this case, a single rotation operation related to n cannot re-balance the tree, and one possible implementation to re-balance the tree is to first mark the node to be deleted during the filtering process, and then conduct an in-order traversal of the tree while skipping the marked nodes. During this in-order traversal, a new BBST is built. This implementation to re-balance the tree takes $O(M|\mathcal{ND}(v(l))|)$ time. For the filtering part (Alg. 3), in the worst case, the entire tree is traversed and all nodes in the tree are to be deleted (from the leaves to the root), which also takes $O(M|\mathcal{ND}(v(l))|)$ time. Although this BBST-based *FilterAndAddFront* has the same worst-case runtime complexity as the for-loop realization, as we will see in the test results, BBST-based method can obviously expedite the search in practice.

D. Bounded Sub-optimal ERCA*

ERCA* can leverage the heuristic inflation technique [20] to trade off solution quality for runtime efficiency. Specifically, in Lines 2 and 15 in Alg. 1, the f-vector of a label l can be computed as $\vec{f}(l) := \vec{g}(l) + \vec{w} * \vec{h}(l)$ (as opposed to $\vec{f}(l) := \vec{g}(l) + \vec{h}(l)$), where $\vec{w} = (1+\epsilon, 1, \cdots, 1)$ is a vector of length $M{+}1$ with $\epsilon \geq 0$, and $*$ stands for the component-wise product of two vectors of the same length. The inflation parameter ϵ only takes effect on the first component of the augmented cost vector. By doing so, ERCA* tends to greedily extract and expand labels that are closer to v_d, especially when ϵ is large. As a result, this heuristic inflation technique often expedites the A*-like search in practice while guaranteeing that the cost $c(\pi)$ of the computed solution path π is at most $(1+\epsilon)c(\pi_*)$ where π_* denotes a true optimal solution path. Finally, for any label l, its truncated f-vector $\mathrm{Trunc}(\vec{f}(l))$ remains unchanged, when inflating the heuristic, to make sure $\mathrm{Trunc}(\vec{f}(l))$ is still a lower bound of the resources needed to reach v_d, which guarantees that the computed solution path does not violate the resource limits.

E. Properties of ERCA*

Theorem 1 (Completeness)**.** *Both ERCA* and the heuristic inflated ERCA* either return a feasible solution or terminate in finite time and returns failure if the given problem instance is infeasible.*

Proof. First, given a label l, to expand l, all possible neighboring vertices of $v(l)$ are considered when generating new labels. Second, at any time during the search, a label l is discarded if one of the following two cases happens.

- (i) Label l is dominated by some existing label l' with $v(l') = v(l)$ (i.e., pruned by *IsPrunByFront*). In this case, l can be discarded since any future path from l can be cut-and-paste to l' without increasing the cost or the resources of the path.
- (ii) There is no feasible path to reach v_d by using l (i.e., pruned by *IsPrunByResour*). In this case, l can be discarded since l cannot lead any feasible solution path to reach v_d.

Therefore, the search enumerates all paths from v_o to any other vertices and discards the infeasible and dominated paths. The search terminates if a solution is found. Since G is finite, there is a finite number of paths from v_o to any other vertices. If the search terminates without returning any solution path, it means the given problem instance is infeasible. $\square$

Theorem 2 (Optimality)**.** *For a feasible problem instance, the solution returned by ERCA* is a minimum cost solution path.*

Proof. In ERCA*, OPEN prioritizes labels in the lex. order. Therefore, every label l that is extracted from OPEN must have the minimum f_1 value among the remaining candidate labels $l' \in$ OPEN. Since the heuristic is consistent,[4] $f_1(l'), l' \in$ OPEN is no larger than the true path cost to reach v_d from l'. As a result, the first label l that is extracted from OPEN and reaches v_d (i.e., $v(l) = v_d$) is guaranteed to have the minimum f_1 value among all feasible paths from v_o to v_d. $\square$

Furthermore, if the heuristic is inflated as described in Sec. IV-D, ERCA* is guaranteed to return a bounded sub-optimal solution path.

Theorem 3 (Bounded Sub-optimality)**.** *For any label l, if $\vec{f}(l) := \vec{g}(l) + \vec{w} * \vec{h}(l)$ with $\vec{w} = (1+\epsilon, 1, \cdots, 1)$ and $\epsilon \geq 0$, then for a feasible problem instance, the heuristic inflated ERCA* returns a bounded sub-optimal solution path π, whose cost satisfies $c(\pi) \leq (1{+}\epsilon)c(\pi_*)$, where π_* is a minimum cost solution path for that problem instance.*

Proof. Let π_* denote an optimal solution of the given instance, and let l_* denote the corresponding label that identifies this optimal solution during the search. Since OPEN prioritizes labels in lex. order based on their f-vectors and $\vec{f}(l) := \vec{g}(l) + \vec{w} * \vec{h}(l)$, for an extracted label l, it is guaranteed that $f_1(l) \leq (1+\epsilon)f_1(l_*) = (1+\epsilon)c(\pi_*)$ (note that $\vec{f}(l_*) = \vec{g}(l_*)$ and $\vec{h}(l_*) = 0$). Therefore, for any label l that reaches v_d (i.e., $v(l) = v_d$), we have $f_1(l) = g_1(l) \leq (1+\epsilon)c(\pi_*)$. $\square$

V. Discussion

A. Relationship to EMOA*

As aforementioned, EMOA* is an algorithm that can compute the entire Pareto-optimal front for a Multi-Objective Path Finding problem. EMOA* conducts A*-like search in the graph to iteratively construct paths from the start vertex towards the goal vertex, while maintaining only non-dominated paths from the start to any other vertex in the graph. Although the problems solved by EMOA* and ERCA* are different, these two algorithms are closely related and their relationship is summarized as follows.

[4]Note that the consistency of the heuristic is required by the dimensionality reduction technique for fast dominance check. In Alg. 1, if no dimensionality reduction is used in *IsPrunByFront*, the heuristic only needs to be admissible.

1) Common features: Both ERCA* and EMOA* [27] employ the three aforementioned fast dominance check techniques to efficiently maintain a frontier set at each vertex v in the graph to memorize non-dominated paths from v_o to v.

2) Without pruning by solutions: ERCA* seeks to find a single optimal solution while EMOA* aims to find a set of non-dominated paths from v_o to v_d. Correspondingly, EMOA* has the notion of the solution set $\mathcal{S}$, which contains the set of non-dominated solution paths that have been found thus far during the search, and an additional pruning procedure, which compares the f-vector of a label l against the f-vector of each solution in $\mathcal{S}$ and discard l if l cannot lead to a non-dominated solution path to reach v_d. In contrast, ERCA* has neither the notion of a solution set $\mathcal{S}$ nor the pruning using existing solutions in $\mathcal{S}$.

3) With pruning by resource limits: ERCA* has *IsPrunByResour*, a pruning rule based on the resource limits, while EMOA* does not. Pruning by resource limits is common in algorithms for RCSPP. During the search of ERCA*, the truncated f-vector of a label $\text{Trunc}(\vec{f}(l))$ provides an underestimate of the resource needed to reach v_d by extending the path represented by that label, and ERCA* can compare $\text{Trunc}(\vec{f}(l))$ against the resource limit to prune a candidate label as early as possible during the search.

4) Termination condition: ERCA* terminates when the first label l with $v_l = v_d$ is extracted from OPEN (Line 11), which guarantees that an optimal solution is found. In contrast, EMOA* terminates when OPEN depletes, i,e., all labels in OPEN are extracted and pruned since they are dominated by some existing solution in the solution set of EMOA*.

B. Relationship to *BiPulse*

1) Best-first search: ERCA* searches G in a best-first manner by iteratively expanding a label with the lex. minimum f-vector in OPEN. In contrast, BiPulse [5] employs a depth-first search (DFS) strategy to quickly find a feasible solution, whose cost provides a primal bound (i.e., upper bound), and then keeps tightening the primal bound. Note that BiPulse also leverages the idea of combining best-first search strategy with DFS [4] by introducing a depth limit on each DFS search branch and by using a priority queue to store the DFS branches that reach this depth limit. By doing so, BiPulse is able to avoid diving too deep into unpromising DFS branches.

2) Without bi-directional search: BiPulse (and many other algorithms for RCSPP and WCSPP) leverage the idea of bi-directional search, while ERCA* does not. It remains an open question how to combine bi-directional search with ERCA* in the presence of multiple resource limits, which is listed as our future work (Sec. VII).[5]

3) With fast dominance check: Although the idea of using the dominance rule to prune paths during the search has been widely used in the existing algorithms for RCSPP [5], [16], [33], these algorithms have little consideration about addressing the computational burden of the dominance check. In contrast, a key focus of ERCA* is to bypass the computational burden of the dominance check, which is a prominent feature of ERCA*. Note that the technique of BBST-based fast dominance check is also applicable to the existing algorithms for RCSPP (e.g. BiPulse) by representing the frontier set at each vertex as a BBST. Other techniques such as lazy check and dimensionality reduction may not be directly applied to these existing RCSPP algorithms as their search process is different from ERCA*.

[5]Some recent fast algorithms for WCSPP leverages the fact that there are only two criteria (cost and one resource called weight) and let the forward and backward search use different orders of criteria to expedite the computation: e.g. the forward search considers (cost, weight) as the g, h, f-vectors of paths while backward search considers (weight, cost). It is worthwhile to investigate how to generalize this technique to more than one resource.

VI. Numerical Results

A. Test Settings and Implementation

We introduce two baselines for comparison. The first baseline is BiPulse [5], an existing leading algorithm for RCSPP that can address any number of resource limits. The second baseline is ERCA* with the aforementioned for-loop realization of dominance checks (as described in Sec. IV-C3) instead of using BBSTs, and this baseline is hereafter referred to as ERCA*-Naive. We implement both baselines and our ERCA* in C++, and test them on a Ubuntu 20.04 laptop with an Intel Core i7-11800H 2.40GHz CPU and 16GB RAM with compiler optimization flag -o3. The BiPulse implementation has two threads, one thread for the forward search and another thread for the backward search, while the implementation of ERCA* and ERCA*-Naive are both single-threaded. BiPulse has a parameter δ that controls the depth limit of each depth-first search branch. According to [5], BiPulse performs the best when $\delta = 2$ and we thus set $\delta = 2$ in our tests.

We test the algorithms using three city road networks from a online data set.[6] We visualize the three networks used in this article in Fig. 4. This data set is widely used in the RCSPP and WCSPP literature, but limited to one cost and one resource. For each road network, this data set provides distance (c_1) and travel time (c_2) for each edge, and c_1 is often used as the cost to be minimized and c_2 is used as the resource in the literature [5]. As this article focuses on more than one resource constraints, we introduce two more properties of edges as follows, which are deterministic and reproducible. Let $deg(v)$ denote the degree (number of adjacent vertices) of $v \in V$, and let $deg(e) := \frac{deg(u)+deg(v)}{2}, e = (u, v) \in E$. If $deg(e) \geq 4$, $c_3(e) = 2$, otherwise $c_3(e) = 1$. The design of c_3 is motivated by hazardous material transportation [10], where the transportation in busy areas in a city can lead to higher risk if leakage happens, and $deg(e)$ is an indicator about how busy an edge is. The fourth property of edges c_4 is simply one for all edges. In other words, given a path π, $c_4(\pi)$ is the number of edges present in π. The intuition behind the design of c_4 is that, the number of edges in a path indicates the number of traffic intersections or stop signs where the vehicle often needs to slow down or stop, which can reduce the comfort of the driver and the passengers [15].

For all the tests, we use c_1 as the cost to be minimized and use the other properties as the resource. We test with both

[6]http://www.diag.uniroma1.it//~challenge9/download.shtml

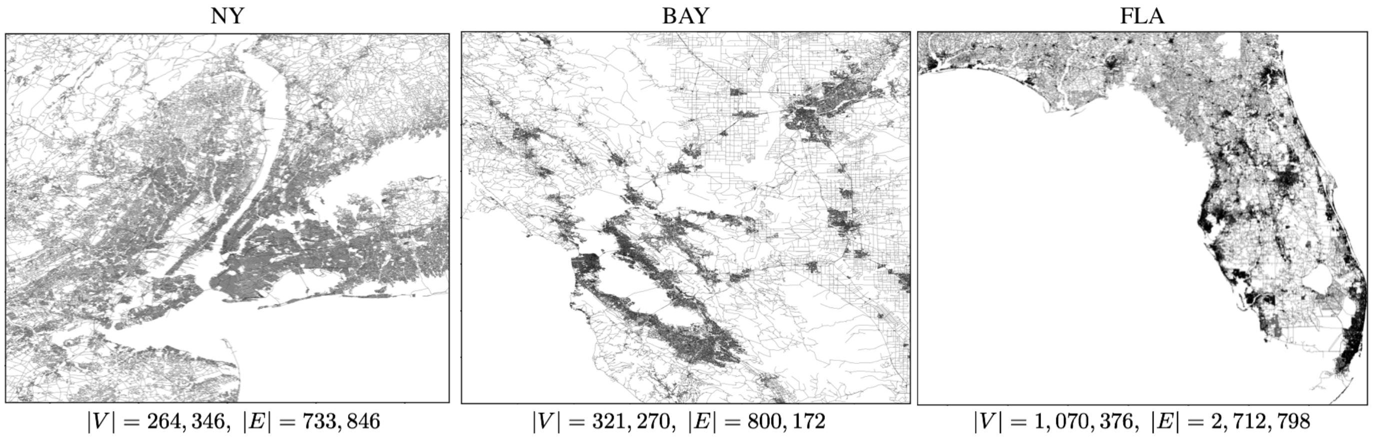

NY
BAY
FLA
$|V| = 264,346,\ |E| = 733,846$
$|V| = 321,270,\ |E| = 800,172$
$|V| = 1,070,376,\ |E| = 2,712,798$

Fig. 4. Visualization of the graphs $G = (V, E)$ used in the tests. The graphs are from a online data set and represent city road networks. The number of vertices and edges in the graph are shown in each sub-figures. More details about these graphs are provided in Sec. VI-A.

two types of resources (c_2, c_3) and three types of resources (c_2, c_3, c_4). To set the resource limits $\vec{r}_{limit}$, we follow the convention in the RCSPP literature [28], which can be summarized as follows. For each edge property $c_k, k \in \{1, 2, 3, 4\}$, an optimal path π_k is computed to minimize the accumulated c_k along the path. Let $c_i(\pi_j), i, j \in \{1, 2, 3, 4\}$ denote the accumulated c_i along path π_j. Then, the resource limit $r_{limit,k-1}, k = 2, 3, 4$ corresponding to c_k is defined as $(c_k(\pi_1) - c_k(\pi_k))p + c_k(\pi_k)$ where $p \in [0, 1]$ is a parameter that controls the *tightness* of the limits. In our tests, we always use the same p for all resources $c_k, k = 2, 3, 4$.

For each city road network, there are 50 start-goal pairs provided in [17], and we use these start-goal pairs in our tests. Each start-goal pair is referred to as an *instance*, and we set a *five* minutes (i.e., 300 seconds) runtime limit for each instance. Both BiPulse and ERCA* require the backwards Dijkstra search at the beginning of the algorithm, and the time needed for the backwards Dijkstra search are excluded from the runtime of the algorithm reported next.

B. Two Resource Limits with Different Tightness

We begin by testing the algorithms in the NY road network with two resource limits and vary the tightness parameter $p \in \{0.2, 0.5, 0.8\}$. The results are shown in Fig. 5. For visualization purposes, we order the instances based on the runtime taken by ERCA* to solve them. Note that when there are more than one resource limit and $p < 1$, an instance can be infeasible, since there may not exist any solution path that simultaneously satisfies all resource constraints. We eliminate from the figure the data points corresponding to both the infeasible instances and the instances where all algorithms time out.

As shown in Fig. 5, first, both ERCA* and ERCA*-Naive often run several orders of magnitude faster than BiPulse, which shows the advantage of the proposed A*-based search in ERCA* and ERCA*-Naive over the depth-first search in BiPulse. Second, we add a green dotted curve in the figure to show the runtime ratio of ERCA* over ERCA*-Naive. It can be observed that, for the instances with small indices in Fig. 5, both ERCA* and ERCA*-Naive can solve them quickly (i.e, in less than 0.1 seconds). For those instances, the runtime ratio is around or above one, which means, the advantage of using BBST in ERCA* is not obvious.

However, for instances that take longer time to solve, ERCA* runs up to an order of magnitude faster than ERCA*-Naive, which verifies the advantage of using BBST in ERCA* for fast dominance checks. Additionally, note that, the green dotted curve goes up at the rightmost side of the figure, and the reason is that ERCA*-Naive times out for these instances (and the counted runtime is 300 seconds), while ERCA* can still solve them with an increased runtime, which makes the ratio higher. Finally, when p decreases, the resource limits become tighter and leads to fewer feasible instances as shown in Fig. 5(c). For the rest of the tests, we fix $p = 0.8$ to ensure that most of the instances are feasible.

C. Internal Running Status of ERCA*

We then test ERCA* and the two baseline methods with two resource constraints in BAY, a different and larger graph, and similar results can be observed in Fig. 6(a). To better understand the running status of ERCA*, we visualize some hardware independent indicators about the internal running status of the algorithm. In all three sub-figures in Fig. 6, the instances are organized with the same indices.

Every time when Alg. 1 reaches Line 12 to get neighbors for a label, we say a label is expanded and the number of expansion increases by one. Similar notion of expansion can also be found in BiPulse, and Fig. 6(b) shows the number of expansion (#Exp.) for both ERCA* and BiPulse. Note that the computational burden of each iteration in BiPulse and ERCA* are different since both algorithms are different. Therefore the number of expansion of both BiPulse and ERCA* cannot be directly compared, and the reported data in Fig. 6(b) can only serve as a reference about the computational burden. Fig. 6(c) counts the size of the frontier sets at all vertices and reports the average of $\{|\mathcal{F}(v)|, \forall v \in V\}$ and $\max_{\forall v \in V} |\mathcal{F}(v)|$. Fig. 6(c) also shows the length of the solution path. We can observe that, for instances whose solution path length is less

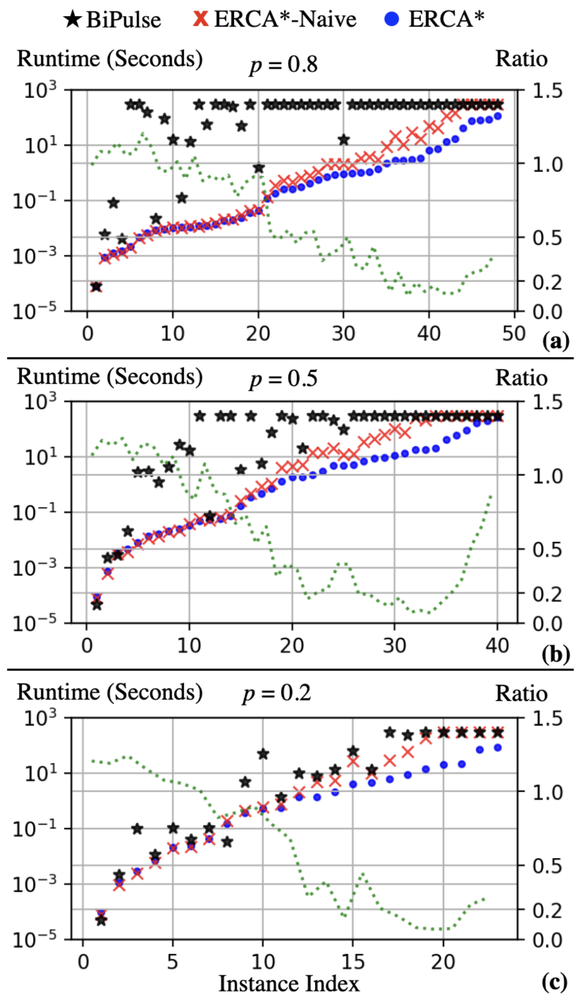


Fig. 5. Numerical results of BiPulse (baseline) ERCA*-Naive (another baseline) and ERCA* (ours) with two resource limits in NY road network. The horizontal axis shows the indices of the test instances. The star, cross and circle markers are against the left vertical axis to show the runtime of each instance. The green dotted curve is against the right vertical axis to show the runtime ratio of ERCA* over ERCA*-Naive. Both ERCA* and ERCA*-Naive are often several orders of magnitude faster than BiPulse, and ERCA* is up to an order of magnitude faster than ERCA*-Naive.

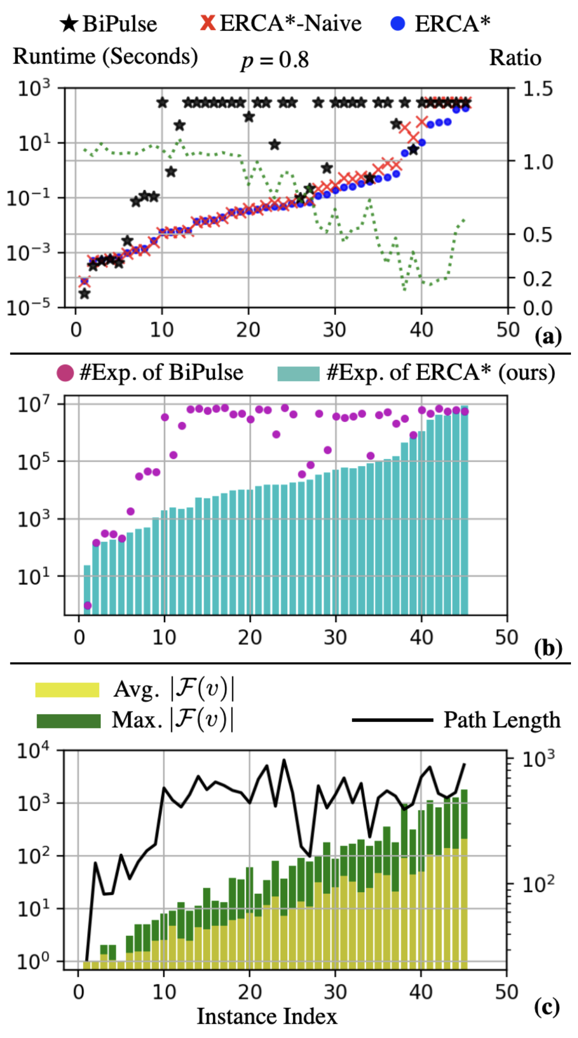


Fig. 6. Results of BiPulse, ERCA*-Naive and ERCA* with two resource limits in BAY road network. The horizontal axis shows the indices of the test instances and all three plots share the same horizontal axis. Different from Fig. 5, where all three sub-plots share the same legends, this figure shows different data and uses different legends in the three sub-plots. In (a), the runtime of the algorithms is shown. In (b), #Exp. stands for the number of expansions. In (c), the green bar shows the size of the largest frontier set at any vertex in the graph when the ERCA* search terminates (i.e., $\max_{\forall v \in V} |\mathcal{F}(v)|$), while the yellow bar shows the average size of the frontier set at any vertex. The black curve shows the solution path length (i.e., the number of vertices in the path) against the right vertical axis. ERCA* is advantageous than ERCA*-Naive for instances larger frontier sets, i.e., more non-dominated paths from v_o to other vertices in the graph.

than 200, the advantage of ERCA* is not obvious and the existing BiPulse can also solve these instances quickly. As the path length increases, ERCA* runs faster than BiPulse. Finally, it can be observed from the figure that, instances that take longer runtime to solve often require a larger number of expansions, and have more non-dominated paths from v_o to other vertices in the graph on average.

D. Three Resource Limits with Different Heuristic Inflation

Finally, we test ERCA* and its bounded sub-optimal variant with three resource limits in FLA, a larger graph with more than a million vertices. We test $\epsilon = 0$ (i.e., the original ERCA*), $\epsilon = 0.1, 0.2$ and report the success rates, runtime, and cost ratio statistics in Table II. The cost ratio is the solution cost computed by the heuristic inflated ERCA* divided by the solution cost computed by ERCA*. We observe that, having a small ϵ (e.g. 0.1) can increase the success rates and reduce the runtime in comparison with ERCA* without any heuristic inflation ($\epsilon = 0$). Additionally, the cost ratios offered by the heuristic inflated ERCA* in practice is often much smaller than the theoretic bound $(1 + \epsilon)$.

However, no further improvement is observed by keeping increasing ϵ (i.e., from 0.1 to 0.2). Additionally, we observe from our tests that, there are four instances where ERCA* ($\epsilon = 0$) succeeds while ERCA* ($\epsilon = 0.1$) fails. The possible reason is that, with an inflated heuristic, ERCA* tends to greedily search towards v_d, which may lead to infeasible paths due to the resource limits, and the search effort is thus wasted. We leave the further investigation of sub-optimal algorithms for RCSPP as our future work.

ϵ	Succ. / Total Inst.	Avg. R.T.	Max. R.T.	Avg. C.R.	Max. C.R.
0	28/50	28.34	205.16	-	-
0.1	34/50	3.02	23.56	1.012	1.055
0.2	34/50	2.97	24.27	1.022	1.115

TABLE II
NUMERICAL RESULTS OF ERCA* WITH DIFFERENT HEURISTIC INFLATION RATES ϵ IN THE FLA MAP WITH THREE RESOURCE LIMITS. THE SECOND COLUMN "SUCC. / TOTAL INST." SHOWS THE NUMBER OF INSTANCES THAT ARE SUCCESSFULLY SOLVED BY THE ALGORITHM WITHIN THE RUNTIME LIMIT AND THE TOTAL NUMBER OF INSTANCES. FOR THE LAST THREE COLUMNS: "R.T." STANDS FOR RUNTIME; "C.R." STANDS FOR SOLUTION COST RATIO; "AVG." AND "MAX." STANDS FOR THE AVERAGE AND MAXIMUM RESPECTIVELY. THESE NUMBERS ARE CALCULATED BASED ON THE INSTANCES THAT ARE SOLVED BY ALL THREE ALGORITHMS WITHIN THE RUNTIME LIMIT.

Finally, we discuss the computational effort of ERCA* as the number of resource constraints varies. From our test results, we do not observe clear trends and there are two possible factors that affect the search efficiency. First, when the number of resource constraints increases, the $\vec{r}, \vec{g}, \vec{f}$ vectors have longer length and the dominance checks become computationally more expensive, which can slow down the ERCA* search. Second, as the number of resource constraints increases, a path is pruned in the *IsPrunByResour* procedure if that path violates any one of the resource constraints. As a result, it may reduce the number of paths to be maintained in the frontier set at a vertex, and therefore expedite the search. For the future work, one can further investigate the efficiency of ERCA* as the number of resources vary in a specific application domain.

VII. CONCLUSION AND FUTURE WORK

This article developed ERCA*, a fast A*-based algorithm that can handle multiple resource constraints for RCSPP. ERCA* is able to expedite the computation by bringing together the existing RCSPP techniques and the recent multi-objective search techniques within the same framework. We also developed a variant of ERCA* that can trade-off solution quality for runtime efficiency, by quickly finding a bounded sub-optimal solution without violating any resource constraints. We tested ERCA* on city-like maps from a public dataset. The results verified the importance of fast dominance check when solving RCSPP, and showed that ERCA* runs several orders of magnitude faster than BiPulse, an existing leading algorithm for RCSPP.

For future work, one can further investigate how to leverage the idea of bi-directional search [1], [33] to expedite ERCA* in the presence of multiple resource limits. Additionally, ERCA* as well as its predecessors mainly consider graphs with non-negative costs and resources defined over the edges. It remains an open question about how to extend these fast A*-based algorithms to handle RCSPP with negative cost and resources defined over the vertices or edges in the graph, which arise in path planning problems for vehicles with recharging or refueling along the paths [6], [19], [21], [32]. One possibility to extend ERCA* to handle negative costs is to introduce a mechanism that can detect negative cycles/loops in the graph and modify the expansion step in the search when negative cycles are detected. Finally, one can compare and combine the heuristic inflated ERCA* with the approximation algorithms [13], [18], [34] for RCSPP to quickly find bounded sub-optimal solutions for challenging instances.

ACKNOWLEDGMENTS

We acknowledge Professor Medaglia, an author of [5], and Professor Thomas, an author of [33], for their timely response and kind help.

REFERENCES

[1] Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. Weight constrained path finding with bidirectional A*. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, pages 2–10, 2022.

[2] Saman Ahmadi, Guido Tack, Daniel D Harabor, and Philip Kilby. Bi-objective search with bi-directional A*. In *Proceedings of the International Symposium on Combinatorial Search*, volume 12, pages 142–144, 2021.

[3] John E Beasley and Nicos Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19(4):379–394, 1989.

[4] Manuel A Bolívar, Leonardo Lozano, and Andrés L Medaglia. Acceleration strategies for the weight constrained shortest path problem with replenishment. *Optimization Letters*, 8(8):2155–2172, 2014.

[5] Nicolás Cabrera, Andrés L Medaglia, Leonardo Lozano, and Daniel Duque. An exact bidirectional pulse algorithm for the constrained shortest path. *Networks*, 76(2):128–146, 2020.

[6] 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, 2021.

[7] Ulrich Derigs, Stefan Friederichs, and Simon Schäfer. A new approach for air cargo network planning. *Transportation Science*, 43(3):370–380, 2009.

[8] Irina Dumitrescu and Natashia Boland. Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks: An International Journal*, 42(3):135–153, 2003.

[9] Matthias Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

[10] Erhan Erkut, Stevanus A Tjandra, and Vedat Verter. Hazardous materials transportation. *Handbooks in operations research and management science*, 14:539–621, 2007.

[11] Jonathan Halpern and I Priess. Shortest path with time constraints on movement and parking. *Networks*, 4(3):241–253, 1974.

[12] Gabriel Y Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980.

[13] Refael Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations research*, 17(1):36–42, 1992.

[14] Carlos Hernández, William Yeoh, Jorge A. Baier, Han Zhang, Luis Suazo, Sven Koenig, and Oren Salzman. Simple and efficient bi-objective search algorithms via fast dominance checks. *Artif. Intell.*, 314:103807, 2023.

[15] Mouna Karoui, Gerard Chalhoub, and Antonio Freitas. An efficient path planning glosa-based approach over large scale and realistic traffic scenario. *Internet Technology Letters*, 4(4):e194, 2021.

[16] Leonardo Lozano and Andrés L Medaglia. On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40(1):378–384, 2013.

[17] Enrique Machuca and Lawrence Mandow. Multiobjective heuristic search in road maps. *Expert Systems with Applications*, 39(7):6435–6445, 2012.

[18] Kurt Mehlhorn and Mark Ziegelmann. Resource constrained shortest paths. In *Algorithms-ESA 2000: 8th Annual European Symposium Saarbrücken, Germany, September 5–8, 2000 Proceedings*, pages 326–337. Springer, 2003.

[19] Florian Morlock, Bernhard Rolle, Michel Bauer, and Oliver Sawodny. Time optimal routing of electric vehicles under consideration of available charging infrastructure and a detailed consumption model. *IEEE Transactions on Intelligent Transportation Systems*, 21(12):5123–5135, 2019.

[20] J. Pearl and J. H. Kim. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(4):392–399, 1982.

[21] Sepideh Pourazarm, Christos G Cassandras, and Tao Wang. Optimal routing and charging of energy-limited vehicles in traffic networks. *International Journal of Robust and Nonlinear Control*, 26(6):1325–1350, 2016.

[22] Luigi Di Puglia Pugliese and Francesca Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013.

[23] Francisco-Javier Pulido, Lawrence Mandow, and José-Luis Pérez-de-la Cruz. Dimensionality reduction in multiobjective shortest path search. *Computers & Operations Research*, 64:60–70, 2015.

[24] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. A conflict-based search framework for multiobjective multiagent path finding. *IEEE Transactions on Automation Science and Engineering*, pages 1–13, 2022.

[25] Zhongqiang Ren, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Multi-objective path-based D* lite. *IEEE Robotics and Automation Letters*, 7(2):3318–3325, 2022.

[26] 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.

[27] 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*, volume 15, pages 162–170, 2022.

[28] Luis Santos, Joao Coutinho-Rodrigues, and John R Current. An improved solution algorithm for the constrained shortest path problem. *Transportation Research Part B: Methodological*, 41(7):756–771, 2007.

[29] Antonio Sedeño-Noda and Sergio Alonso-Rodríguez. An enhanced k-sp algorithm with pruning strategies to solve the constrained shortest path problem. *Applied Mathematics and Computation*, 265:602–618, 2015.

[30] Olivia J Smith, Natashia Boland, and Hamish Waterer. Solving shortest path problems with a weight constraint and replenishment arcs. *Computers & Operations Research*, 39(5):964–984, 2012.

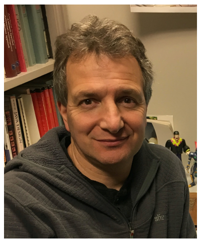
[31] Bradley S. Stewart and Chelsea C. White. Multiobjective A*. *J. ACM*, 38(4):775–814, October 1991.

[32] Martin Strehler, Sören Merting, and Christian Schwan. Energy-efficient shortest routes for electric and hybrid vehicles. *Transportation Research Part B: Methodological*, 103:111–135, 2017.

[33] Barrett W Thomas, Tobia Calogiuri, and Mike Hewitt. An exact bidirectional A* approach for solving resource-constrained shortest path problems. *Networks*, 73(2):187–205, 2019.

[34] Arthur Warburton. Approximation of pareto optima in multiple-objective, shortest-path problems. *Operations research*, 35(1):70–79, 1987.

Zhongqiang (Richard) Ren (Student Member, IEEE) received the dual B.E. degree in mechatronics from Tongji University, Shanghai, China, in 2015, and FH Aachen University of Applied Sciences, Aachen, Germany, and the M.S. and Ph.D. degrees in mechanical engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2017 and 2022, respectively. He is currently a Postdoctoral Fellow with Carnegie Mellon University.



Zachary B. Rubinstein is a Principal Project Scientist in the Robotics Institute at Carnegie Mellon University. He received a BA degree in Philosophy from Brandeis University in 1983, and MS and PhD Degrees in Computer Science from the University of Massachusetts at Amherst in 1993 and 2002. Prior to being at CMU, Dr. Rubinstein was an assistant professor in the Computer Science Department at the University of New Hampshire. Dr. Rubinstein's research has focused on developing intelligent solutions to real-world problems, especially in the area of effective resource allocation in dynamic and uncertain environments.

Stephen F. Smith is a Research Professor of Robotics at Carnegie Mellon University, where he heads the Intelligent Coordination and Logistics Laboratory. Dr. Smith's research focuses broadly on the theory and practice of next-generation technologies for automated planning, scheduling, and control of large multi-actor systems. He pioneered the development and use of constraint-based search and optimization models for planning and scheduling, and has successfully fielded AI-based planning and scheduling systems in a range of application domains. Dr. Smith has published over 300 technical papers in AI, automated planning and scheduling, machine learning and related areas. He is a Fellow of the Association for the Advancement of Artificial Intelligence (AAAI), and in June 2022 he became President Elect of AAAI. His current research interests include collaborative multi-agent decision-making, resilient planning and scheduling systems, stochastic optimization, and smart infrastructure for urban mobility.

Sivakumar Rathinam (Senior Member, IEEE) received the Ph.D. degree in civil systems engineering from the University of California at Berkeley, Berkeley, CA, USA, in 2007. He is currently a Professor with the Department of Mechanical Engineering, Texas A&M University, College Station, TX, USA. His research interests include motion planning and control of autonomous vehicles, collaborative decision making, combinatorial optimization, vision-based control, and air traffic control.

Howie Choset (Fellow, IEEE) received the bachelor's degree in computer science and business from the University of Pennsylvania, Philadelphia, PA, USA, in 1990, and the M.S. and Ph.D. degrees in mechanical engineering from the California Institute of Technology, Pasadena, CA, USA, in 1991 and 1996, respectively. He is currently a Professor with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA.