Resource Constrained Pathfinding with Enhanced Bidirectional A* Search

Saman Ahmadi[1], **Andrea Raith**[2], **Guido Tack**[3], **Mahdi Jalili**[1]

[1] School of Engineering, RMIT University, Australia
[2] Department of Engineering Science, University of Auckland, New Zealand
[3] Department of Data Science and AI, Monash University, Australia
saman.ahmadi@rmit.edu.au, a.raith@auckland.ac.nz, guido.tack@monash.edu, mahdi.jalili@rmit.edu.au

Abstract

The classic Resource Constrained Shortest Path (RCSP) problem aims to find a cost optimal path between a pair of nodes in a network such that the resources used in the path are within a given limit. Having been studied for over a decade, RCSP has seen recent solutions that utilize heuristic-guided search to solve the constrained problem faster. Building upon the bidirectional A* search paradigm, this paper introduces a novel constrained search framework that uses efficient pruning strategies to allow for accelerated and effective RCSP search in large-scale networks. Results show that, compared to the state of the art, our enhanced framework can significantly reduce the constrained search time, achieving speed-ups of over to two orders of magnitude.

Introduction

Often studied in the context of network optimization, the Resource Constrained Shortest Path (RCSP) problem seeks to find a cost-optimal path between two nodes in a network such that the resource usage of the path is within a given limit. Its applications span a wide range of domains. As a core problem, RCSP can be modelled to find minimum time paths for an unmanned aerial vehicle such that the total cumulative noise disturbance created by the vehicle along its trajectory is within an allowable noise disturbance level (Cortez 2022). As a subroutine, RCSP appears as a form of pricing problem in column generation (a solving technique in linear programming) for routing and scheduling applications (Parmentier, Martinelli, and Vidal 2023). The problem has been shown to be NP-hard (Handler and Zang 1980).

RCSP is a well-studied topic in the AI and mathematical optimization literature. A summary of conventional solutions to RCSP (methods that solve the problem to optimality) can be found in Pugliese and Guerriero (2013) and Ferone et al. (2020). Among the recent solutions, the RCBDA* algorithm of Thomas, Calogiuri, and Hewitt (2019) is a label setting method that explores the network bidirectionally using heuristic-guided A* search. Built on the bidirectional dynamic programming algorithm of Righini and Salani (2006), RCBDA* defines an explicit perimeter for the search of each direction by evenly distributing the resource budget between the directions. More precisely, each direction explores only those labels whose resource consumption is not more than half of the resource budget, while enabling the construction of complete paths by joining forward and backward labels. The authors reported that the guided search of A* helps RCBDA* perform faster in large graphs when compared to the conventional RSCP methods of Lozano and Medaglia (2013) and Sedeño-Noda and Alonso-Rodríguez (2015). Nevertheless, it still leaves several instances with a single resource constraint unsolved, even after a five-hour timeout. BiPulse (Cabrera et al. 2020) is another bidirectional search framework that parallelizes the branch-and-bound Pulse procedure of Lozano and Medaglia (2013). BiPulse utilizes a queueing approach to explore branches longer than a specific depth limit separately in a breadth-first search manner. The authors reported better performance with BiPulse compared to RCBDA*, though no direct comparison was made as RCBDA*'s runtimes were scaled from the original paper. Both RCBDA* and BiPulse were recognized with awards (Golden and Shier 2021).

Quite recently, the unidirectional A*-based constrained search proposed in Ren et al. (2023), called ERCA*, has demonstrated superior performance compared to BiPulse. Ren et al. adapted their multi-objective search algorithm EMOA* (Ren et al. 2022) to RCSP, and proposed a solution that can reduce the constrained search effort by integrating binary search trees into pruning strategies. Although not compared against RCBDA*, ERCA* was reported to outperform BiPulse by several orders of magnitude. The two recent WCBA* (Ahmadi et al. 2022) and WCA* (Ahmadi et al. 2024c) algorithms can also be seen as heuristic-guided approaches to RCSP, but their application is limited to instances with only a single constraint (the weight constrained variant). Both algorithms were reported to perform faster than RCBDA* and BiPulse.

This paper introduces an enhanced bidirectional A* search algorithm for RCSP, called RCEBDA*. Built on the basis of perimeter search with bidirectional A*, our proposed algorithm leverages the recent advancements in both constrained and multi-objective search, and adopts optimized dominance pruning and path matching procedures that help reduce the search effort substantially. Experimental results on benchmark instances reveal that RCEBDA* can effectively solve more instances than the state of the art, achieving speed-ups above two orders of magnitude.

Problem Definition

Consider a directed graph G with a set of states S and a set of edges $E \subseteq S \times S$ where every edge has $k \in \mathbb{N}_{\geq 2}$ attributes, represented as $(cost, resource_1, \ldots, resource_{k-1})$. A path on G is a sequence of n states $u_i \in S$ with $i \in \{1, \ldots, n\}$ and $(u_i, u_{i+1}) \in E$ for $i \in \{1, \ldots, n-1\}$. The cost and resource consumption of path $\pi = \{u_1, \ldots, u_n\}$ is then the sum of corresponding attributes on all the edges along the path. The objective in the RCSP problem is to find a $cost$-optimal path π^* between a given pair of $(start, goal)$ states such that the consumption of resources on the path respects a set of limits $\mathbf{R} = \{R_1, \ldots, R_{k-1}\}$, that is, we must have $resource_i(\pi^*) \leq R_i$ for every $i \in \{1, \ldots, k-1\}$.

Our notation generalizes the two possible *forward* and *backward* search directions by searching in direction d from an *initial* state to a *target* state. A forward search explores the forward graph, denoted by $G^f = (S, E^f)$, from the $start$ state, while a backward search begins from the $goal$ state on the reversed graph (i.e., G^f with all its edges reversed), denoted by $G^b = (S, E^b)$. Further, $Succ^d(u)$ denotes all successor states of u in the graph of search direction d.

To simplify our algorithmic description, we define edge attributes as a form of cost vector, namely $\mathbf{cost} = (cost_1, \ldots, cost_k)$, with $cost_1$ denoting the primary cost of the problem and $(cost_2, \ldots, cost_k)$ representing the resources. Edge costs in the search graph of direction d can be accessed through the function $\mathbf{cost}^d : E^d \to \mathbb{R}^k_{\geq 0}$. The search objects in our notation are defined as *nodes*, which can be seen as partial paths obtained through exploring the graph from the $initial$ state. A node x conventionally contains some key information on the partial path extended from the $initial$ state to state $s(x) \in S$. The function $s(x)$ here returns the state associated with x. Node x traditionally stores $\mathbf{cost}$ of the partial path, accessible via the value pair $\mathbf{g}(x)$. It also stores a value pair $\mathbf{f}(x)$, which estimates the $\mathbf{cost}$ of a complete $initial$-$target$ path via x; and also a reference $parent(x)$ which indicates the parent node of x.

All operations on the vectors (shown in boldface) are assumed to be done element-wise. We use the symbol $\preceq$ in direct comparisons of cost vectors. To compare vectors for their resource component only, we use the symbol $\preceq^{\mathrm{Tr}}$ to denote the truncation of the first cost in the comparison, e.g., $\mathbf{g}(x) \preceq^{\mathrm{Tr}} \mathbf{g}(y)$ denotes $g_i(x) \leq g_i(y)$ for all $i \in \{2, \ldots, k\}$.

Definition Let $\overline{\mathbf{f}} = (\overline{f_1}, \ldots, \overline{f_k})$ be a global upper bound on $\mathbf{cost}$ of any $start$-$goal$ paths. x is out of bounds if $\mathbf{f}(x) \not\preceq \overline{\mathbf{f}}$.

Definition Node x weakly dominates node y if we have $\mathbf{g}(x) \preceq \mathbf{g}(y)$; x (strongly) dominates y if $\mathbf{g}(x) \preceq \mathbf{g}(y)$ and $\mathbf{g}(x) \neq \mathbf{g}(y)$; y is not dominated by x if $\mathbf{g}(x) \not\preceq \mathbf{g}(y)$.

In the context of bidirectional A*, the search in direction d is guided by cost estimates or $\mathbf{f}$-values, which are traditionally established based on a heuristic function $\mathbf{h}^d : S \to \mathbb{R}^k_{\geq 0}$ (Hart, Nilsson, and Raphael 1968). In other words, for every search node x in direction d, we have $\mathbf{f}(x) = \mathbf{g}(x) + \mathbf{h}^d(s(x))$ where $\mathbf{h}^d(s(x))$ returns lower bounds on the $\mathbf{cost}$ of paths from $s(x)$ to the $target$ state.

Definition The function $\mathbf{h}^d : S \to \mathbb{R}^k_{\geq 0}$ is consistent if $\mathbf{h}^d(u) \leq \mathbf{cost}^d(u, v) + \mathbf{h}^d(v)$ for every edge $(u, v) \in E^d$. $\mathbf{h}^d$ is admissible if it is consistent and $\mathbf{h}^d(initial) = \mathbf{0}$.

Bidirectional Constrained A* Search

Originally introduced by Pohl (1971) for single-objective pathfinding, bidirectional A* search strategies have since evolved, with newer approaches offering promising advancements in search efficiency (Alcázar 2021; Siag et al. 2023). In the RCSP context, the classic front-to-end bidirectional A* search of RCBDA* has proven effective in guiding constrained searches to optimal solutions (Thomas, Calogiuri, and Hewitt 2019; Ahmadi et al. 2021). Analogous to classic bidirectional A* search, the RCBDA* algorithm employs an interleaved strategy in which the search is guided by the direction that exhibits the lowest f_1-value. Let f_1^* be the optimal cost of a valid solution path. RCBDA* enumerates in best-first order all partial paths with f_1-value smaller than f_1^* in both directions sequentially. A complete $start$-$goal$ path in this framework is built by joining forward and backward partial paths arriving to the same state. However, since the number of paths to each state can grow exponentially during the search, partial path matching can become a demanding task if each direction is allowed to explore the entire search space.

To mitigate the bottleneck above, RCBDA* utilizes the perimeter bounding strategy proposed by Righini and Salani (2006), and equally distributes one of the resource budgets between the directions. The selected resource is referred to as the *critical* resource. Consequently, the algorithm explores, in each direction, those partial paths that consume no more than half of the provided critical resource budget. In addition, to prevent the search from exploring unpromising paths, RCBDA* does not process nodes that have already exhausted (any of) the resource budgets. Thomas, Calogiuri, and Hewitt experimented RCBDA* with the pruning rules presented in Lozano and Medaglia (2013), namely: pruning by dominance and infeasibility. The former allows the search to prune nodes dominated by a previously explored node, while the latter ensures that nodes with estimated resource usage larger than the given limits are not explored. Although the authors did not discuss the pruning rules as part of their label-setting approach, we have provided in Algorithm 1 a refined pseudocode closest to their description, including both pruning rules.

The algorithm starts with initializing a global search upper bound $\overline{\mathbf{f}}$ using the resource limits provided. It then establishes the heuristic functions of the forward and backward direction $\mathbf{h}^f$ and $\mathbf{h}^\mathbf{b}$, respectively. These heuristic functions are conventionally obtained through k rounds of single-objective one-to-all searches, e.g., using Dijkstra's algorithm (Johnson 1973), in the reverse direction. The algorithm also initializes for each direction d two types of data structures: a priority queue, called $Open^d$, that maintains unexplored nodes generated in direction d; and a node list $\mathcal{X}^d(u)$ that stores nodes explored with state $u \in S$ in direction d. $\mathcal{X}^d(u)$ is used to fulfil the pruning by dominance rule, as well as path matching with nodes explored in the opposite direction. To commence the search, the algorithm generates two initial nodes, one for each direction, and inserts them into the corresponding priority queue.

The main constrained search starts at line 9 and contin-

Algorithm 1: RCBDA* (Improved)

Input: An RCSP Problem $(G, start, goal, R_1, \ldots, R_{k-1})$
Output: Optimal cost of the problem

1 $\overline{\mathbf{f}} \leftarrow (\infty, R_1, \ldots, R_{k-1})$
2 $\mathbf{h}^f, \mathbf{h}^b \leftarrow$ forward and backward **cost** lower bounds
3 $Open^f \leftarrow \emptyset$, $Open^b \leftarrow \emptyset$
4 $\mathcal{X}^f(u) \leftarrow \emptyset, \mathcal{X}^b(u) \leftarrow \emptyset \; \forall u \in S$
5 $x \leftarrow$ new node with $s(x) = start$
6 $y \leftarrow$ new node with $s(y) = goal$
7 $\mathbf{g}(x) \leftarrow \mathbf{g}(y) \leftarrow \mathbf{0}, \mathbf{f}(x) \leftarrow \mathbf{h}^f(start), \mathbf{f}(y) \leftarrow \mathbf{h}^b(goal)$
8 add x to $Open^f$ and y to $Open^b$
9 **while** $Open^f \bigcup Open^b \neq \emptyset$ **do**
10 extract from $Open^f \bigcup Open^b$ node x with the smallest f_1-value
11 **if** $f_1(x) \geq \overline{f_1}$ **then break**
12 $d \leftarrow$ the direction from which x was extracted
13 $d' \leftarrow$ the opposite direction of d
14 **if** $g_\kappa(x) \leq \overline{f_\kappa}/2$ **then**
15 **for each** $t \in Succ^d(s(x))$ **do**
16 $y \leftarrow$ new node with $s(y) = t$
17 $\mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{cost}^d(s(x), t)$
18 $\mathbf{f}(y) \leftarrow \mathbf{g}(y) + \mathbf{h}^d(t)$
19 **if** $\mathbf{f}(y) \not\preceq \overline{\mathbf{f}}$ **then continue**
20 **if** IsDominated$(y, \mathcal{X}^d(t))$ **then continue**
21 Add y to $Open^d$
22 **for each** $y \in \mathcal{X}^{d'}(s(x))$ **do**
23 **if** $\mathbf{g}(x) + \mathbf{g}(y) \preceq \overline{\mathbf{f}}$ **then** $\overline{f_1} \leftarrow g_1(x) + g_1(y)$
24 add x to $\mathcal{X}^d(s(x))$
25 **return** $\overline{f_1}$

Algorithm 2: IsDominated

Input: A node x and a set of nodes χ
Output: *true* if x is weakly dominated, *false* otherwise

1 **for each** $y \in \chi$ **do**
2 **if** $\mathbf{g}(y) \preceq^{\text{Tr}} \mathbf{g}(x)$ **then return** *true*
3 **return** *false*

ues until both priority queues are empty. Let $\overline{f_1}$ be the best known upper bound on the primary cost of the problem (initially unknown). In each iteration, the algorithm extracts from one of the queues a node x that has the smallest f_1-value among all unexplored nodes. If there are multiple candidates, it breaks ties arbitrarily. The search can stop immediately if we find $f_1(x) \geq \overline{f_1}$, essentially because it has proven the optimality of $\overline{f_1}$. Note that A* explores nodes in non-decreasing order of their f_1-value. Thus, all unexplored nodes in the priority queues would not improve $\overline{f_1}$ upon fulfillment of the termination criterion of line 11.

Let d be the direction of the priority queue from which x was extracted. Also assume d' is the opposite direction of d. If the search is not terminated, the algorithm expands x if the consumption of its critical resource, denoted by $g_\kappa(x)$ for $\kappa \in \{2, \ldots, k\}$, is not more than 50% of the corresponding budget, or $\overline{f_\kappa}/2$ equivalently (line 14). Expansion of x generates a set of descendant nodes. Each descendant node y will be added into $Open^d$ unless it is deemed out of bounds (line 19), a process known as pruning by infeasibility. The algorithm then checks the descendant node against previously explored nodes (known as pruning by dominance). As shown in Algorithm 2, the dominance check involves comparing $\mathbf{g}(y)$ against **cost** of all previously explored nodes with $s(y)$ in $\mathcal{X}^d(s(y))$. Since nodes with $s(y)$ are explored in non-decreasing order of their g_1-values, we optimize the operation and compare nodes for their resource component only (using the $\preceq^{\text{Tr}}$ operator).

The next step involves constructing complete $start$-$goal$ paths by joining x with nodes explored with $s(x)$ in the opposite direction d', already available in $\mathcal{X}^{d'}(s(x))$. The search updates the best known upper bound on the primary cost, i.e., $\overline{f_1}$, if it finds that the joined path is within the bounds (lines 22-23). After the completion of the path matching step, x will be added to the list of expanded nodes with $s(x)$ in direction d (line 24), to be joined with nodes explored in the opposite direction. Finally, the algorithm returns $\overline{f_1}$ as an optimal cost to the given RCSP instance.

An Enhanced Bidirectional RCSP Search

As explained in the previous section, RCBDA* stores in both directions all explored nodes of each state so it can prune dominated nodes before inserting them into the priority queue. Although this approach helps alleviate the queue load by removing dominated nodes as soon as they are generated, it may still let the search explore some dominated nodes. This is because the algorithm does not check newly generated nodes against unexplored nodes, which can lead to having dominated nodes in the queues. Nonetheless, given that the number of explored nodes grows exponentially during the search, it is likely that RCBDA* spends most of its search time on dominance pruning. This costly pruning procedure can be considered a search bottleneck, yet it is essential to prevent the frontier from growing uncontrollably.

Besides dominance pruning, search initialization can be seen as a crucial task in improving the efficiency of the main search. RCBDA* is conventionally initialized using multiple rounds of one-to-all searches, which yield the A*'s required heuristics. Although the correctness of constrained A* search relies on its heuristic being consistent and admissible, it has been shown that the search performance can be greatly impacted with the quality of heuristics (Ahmadi et al. 2024c), suggesting further opportunities for enhancing the exhaustive resource constrained search with A*.

To address the above shortcomings, this paper presents RCEBDA*, a new RCSP algorithm that enhances the resource constrained search with bidirectional A*. A pseudocode of RCEBDA* is provided in Algorithm 4, with the symbol $*$ next to specific line numbers indicating our proposed changes. We describe each enhancement as follows.

Better informed heuristics: When conducting A* bidirectionally, we need to compute both backward and forward lower bounds. In the context of RCSP, this lower bounding strategy provides us with a great opportunity to feed the

Algorithm 3: `Initialize`

Input: An RCSP instance (G, $start$, $goal$, $\overline{\mathbf{f}}$)
Output: Lower bounds ($\mathbf{h}^f, \mathbf{h}^b$), S updated

$\mathbf{h}^f(u) \leftarrow \mathbf{h}^b(u) \leftarrow \infty^k \; \forall u \in S$
for $i \in \{k, \dots, 1\}$ **do**
  $h_i^f \leftarrow$ Single-objective backward search on $cost_i$
  $h_i^b \leftarrow$ Single-objective forward search on $cost_i$
  $S \leftarrow \{u \mid u \in S \text{ and } h_i^f(u) + h_i^b(u) \leq \overline{f_i}\}$
return $(\mathbf{h}^f, \mathbf{h}^b)$

main search with better informed heuristics, thereby reducing the search effort and node expansions, as shown by Ahmadi et al. (2021, 2024c) for the weight constrained variant. We present in Algorithm 3 our proposed initialization phase for RCEBDA*. After initializing the lower bounds, the algorithm runs k rounds of one-to-all searches in both directions, starting from the last attribute. In each round, we apply a technique called *resource-based network reduction* (Aneja, Aggarwal, and Nair 1983) and use the established lower bounds to prune out-of-bounds states. Given $\overline{f_i}$ as the global upper bound on $cost_i$ of paths, state $u \in S$ can be removed if its least $cost_i$ paths to both ends yield a complete path that violates $\overline{f_i}$ (line 5). This pruning can be skipped in the last round ($i = 1$), as $\overline{f_1}$ is initially unknown. Consequently, there will be $k - 1$ levels of graph reduction, all contributing to improving the quality of the heuristics.

Efficient dominance pruning: The fact that nodes in A* are explored in non-decreasing order of their f_1-value enables us to reduce the dimension of cost vectors by one in all dominance checks. In other words, a node x is weakly dominated by a previously explored node y, $s(x) = s(y)$, if $\mathbf{g}(y) \preceq^{\text{Tr}} \mathbf{g}(x)$. Otherwise, x will be stored in $\mathcal{X}^d(s(x))$ if it is deemed non-dominated. In the latter case, if we observe $\mathbf{g}(x) \preceq^{\text{Tr}} \mathbf{g}(y)$, we can guarantee that any future node with $s(x)$ and weakly dominated by y will similarly be weakly dominated by x, rendering the storage of y in $\mathcal{X}^d(s(x))$ redundant. Previously studied in Pulido, Mandow, and Pérez-de-la-Cruz (2015), this technique has been utilized in both constrained and multi-objective search to reduce the dominance check effort by removing dominated (truncated) cost vectors from the expanded lists (Ren et al. 2023; Hernández et al. 2023). In our bidirectional framework, however, such nodes cannot be fully removed from the search, as every non-dominated node must be available for potential path matching. To this end, we allow each state $u \in S$ to store its previously explored nodes in two lists: $\mathcal{X}^d(u)$ and $\mathcal{X}^d_{Dom}(u)$. The latter maintains (previously non-dominated) nodes whose truncated cost vector is dominated by that of one node in $\mathcal{X}^d(u)$, while the former stores non-dominated ones. Note that neither $\mathcal{X}^d(u)$ nor $\mathcal{X}^d_{Dom}(u)$ contains out-of-bounds nodes. In addition, to ensure nodes are checked for dominance before expansion, we lazily delay the dominance check of nodes until they are extracted from the queue (line 18). Although this approach incurs extra queueing effort due to queues containing more dominated nodes, it will lead to reducing node expansions.

Algorithm 4: RCEBDA*

Input: An RCSP problem (G, $start$, $goal$, $R_1, \dots, R_{k-1}$)
Output: A set of node pairs representing optimal paths

1 $\overline{\mathbf{f}} \leftarrow (\infty, R_1, \dots, R_{k-1})$
2* $\mathbf{h}^f, \mathbf{h}^b \leftarrow$ `Initialize`(G, $start$, $goal$, $\overline{\mathbf{f}}$)
3* $Sols \leftarrow \emptyset$
4 $Open^f \leftarrow \emptyset$, $Open^b \leftarrow \emptyset$
5 $\mathcal{X}^f(u) \leftarrow \emptyset, \mathcal{X}^b(u) \leftarrow \emptyset \; \forall u \in S$
6* $\mathcal{X}^f_{Dom}(u) \leftarrow \emptyset, \mathcal{X}^b_{Dom}(u) \leftarrow \emptyset \; \forall u \in S$
7 $x \leftarrow$ new nodes with $s(x) = start$
8 $y \leftarrow$ new node with $s(y) = goal$
9 $\mathbf{g}(x) \leftarrow \mathbf{g}(y) \leftarrow \mathbf{0}$, $\mathbf{f}(x) \leftarrow \mathbf{h}^f(start)$, $\mathbf{f}(y) \leftarrow \mathbf{h}^b(goal)$
10 add x to $Open^f$ and y to $Open^b$
11 **while** $Open^f \bigcup Open^b \neq \emptyset$ **do**
12 extract from $Open^f \bigcup Open^b$ node x with the smallest f_1-value
13* **if** $f_1(x) > \overline{f_1}$ **then break**
14 $d \leftarrow$ direction from which x was extracted
15 $d' \leftarrow$ opposite direction of d
16* $z \leftarrow$ last node in $\mathcal{X}^d(s(x))$
17* **if** $\mathbf{g}(z) \preceq^{\text{Tr}} \mathbf{g}(x)$ **then continue**
18* **if** `IsDominated`$(x, \mathcal{X}^d(s(x)))$ **then continue**
19* **for each** $y \in \mathcal{X}^d(s(x))$ **do**
20* **if** $\mathbf{g}(x) \preceq^{\text{Tr}} \mathbf{g}(y)$ **then**
21* move y from $\mathcal{X}^d(s(x))$ to $\mathcal{X}^d_{Dom}(s(x))$
22 add x to $\mathcal{X}^d(s(x))$
23* `Match`$(x, \mathcal{X}^{d'}(s(x)), \mathcal{X}^{d'}_{Dom}(s(x)))$
24 **if** $g_\kappa(x) \leq \overline{f_\kappa}/2$ **then**
25 **for each** $t \in Succ^d(s(x))$ **do**
26 $y \leftarrow$ new node with $s(y) = t$
27 $\mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{cost}^d(s(x), t)$
28 $\mathbf{f}(y) \leftarrow \mathbf{g}(y) + \mathbf{h}^d(t)$
29 $parent(y) \leftarrow x$
30* $z \leftarrow$ last node in $\mathcal{X}^d(t)$
31* **if** $\mathbf{f}(y) \not\preceq \overline{\mathbf{f}}$ **or** $\mathbf{g}(z) \preceq^{\text{Tr}} \mathbf{g}(y)$ **then continue**
32 add y to $Open^d$
33 **return** $Sols$

Let x be a non-dominated node just extracted from $Open^d$. This node will eventually be added to $\mathcal{X}^d(s(x))$, but to keep the truncated cost vector of nodes in $\mathcal{X}^d(s(x))$ non-dominated, we first check other nodes of the list against x. As shown in lines 19-21 of Algorithm 4, all nodes whose truncated cost vector is dominated by that of x are removed from $\mathcal{X}^d(s(x))$ and added into $\mathcal{X}^d_{Dom}(s(x))$ subsequently.

Quick dominance pruning: We also utilize a quick pruning rule that checks x against the most recent node explored with $s(x)$ before the rigorous dominance check, and also during expansion for descendants of x (see lines 16-17 and 30-31). This technique has proven to be effective in substantially reducing the number of dominance checks during exhaustive multi-objective A* search (Ahmadi et al. 2024b).

Path matching: RCEBDA* stores all explored but non-dominated nodes of each state in two lists, so it can still join

Algorithm 5: `Match`

Input: A node x, and sets (χ, χ_{Dom}) corresponding to $s(x)$
Output: $Sols$ updated

1 **for each** $y \in \chi \bigcup \chi_{Dom}$ **do**
2 **if** $\mathbf{g}(x) + \mathbf{g}(y) \preceq \overline{\mathbf{f}}$ **then**
3 **if** $g_1(x) + g_1(y) < \overline{f_1}$ **then**
4 $\overline{f_1} \leftarrow g_1(x) + g_1(y)$
5 $Sols \leftarrow \emptyset$
6 $dominated \leftarrow$ false
7 **for each** $(x', y') \in Sols$ **do**
8 **if** $\mathbf{g}(x') + \mathbf{g}(y') \preceq^{\text{Tr}} \mathbf{g}(x) + \mathbf{g}(y)$ **then**
9 $dominated \leftarrow$ true; **break**
10 **if** $\mathbf{g}(x) + \mathbf{g}(y) \preceq^{\text{Tr}} \mathbf{g}(x') + \mathbf{g}(y')$ **then**
11 remove (x', y') from $Sols$
12 **if** $dominated =$ false **then** add (x, y) to $Sols$

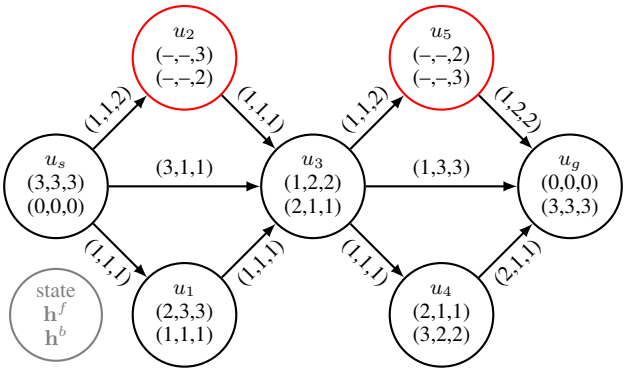

Figure 1: An example graph with $k = 3$ and $R = \{4, 4\}$. Triples inside the states denote $\mathbf{h}^f$ and $\mathbf{h}^b$. u_2 and u_5 (in red) can be seen out of bounds for their last resource ($3+2 \not\leq 4$).

bidirectional paths. Let x be a non-dominated node explored in direction d. As shown in Algorithm 5, RCEBDA* joins x with all non-dominated nodes of the opposite direction d', stored in $\mathcal{X}^{d'}(s(x))$ and $\mathcal{X}^{d'}_{Dom}(s(x))$. Contrary to RCBDA* where it captures the optimal cost only, we allow the algorithm to store all node pairs yielding a non-dominated $cost_1$-optimal solution path. For every candidate node y of opposite direction, the algorithm checks whether joining x with y constructs a feasible path (line 2). If the joined path improves the best known optimal cost $\overline{f_1}$, we update the global upper bound and remove all previous (suboptimal) solutions (line 3-5). The algorithm then explicitly checks the joined path against all existing solutions for dominance (lines 6-11). The node pair representing the resulting path will be added to $Sols$ if the algorithm finds the joined path non-dominated (line 12).To keep the solution set only containing nodes that form non-dominated solution paths, the algorithm removes from $Sols$ all node pairs weakly dominated by the joined path. Note that the algorithm allows successive updates to the upper bound when a better solution is found, and all node pairs in $Sols$ always have the same f_1-value.

Although the path matching procedure of Algorithm 5 considers all non-dominated nodes explored in the opposite direction, we can optimize the procedure by prioritizing nodes in the $\mathcal{X}^{d'}(s(x))$ list. In this optimized strategy, if we find all joined paths out of bounds for their resource usage, that is, if we have $\mathbf{g}(x) + \mathbf{g}(y) \not\preceq^{\text{Tr}} \overline{\mathbf{f}}$ for all $y \in \mathcal{X}^{d'}(s(x))$, we can skip matching x with candidates from $\mathcal{X}^{d'}_{Dom}(s(x))$ because they are already dominated by a node in $\mathcal{X}^{d'}(s(x))$ for their resources, rendering their matching unnecessary.

A formal proof of the correctness of RCEBDA* is provided in Ahmadi et al. (2024a). The example below further elaborates on the key steps involved in the algorithm.

Example: Consider the sample RCSP instance of Figure 1 with three edge attributes (two resources) with resource limits $R = \{4, 4\}$. We aim to find an optimal feasible path from state u_s to state u_g. RCEBDA* first sets the upper bound $\overline{\mathbf{f}} \leftarrow (\infty, 4, 4)$. For every state, we have provided the forward and backward lower bounds as triples within the states. Note that each h^d_i-value represents a $cost_i$-optimal path for $i \in \{1, 2, 3\}$. Checking the states for their bidirectional lower bounds, we realize that states u_2 and u_5 (shown in red) are out of bounds for their last cost, as we have $3 + 2 \not\leq 4$. Thus, they can be removed from the search graph in the initialization phase. Let the last (third) attribute be the critical resource for the perimeter search of RCEBDA*, i.e., $\kappa = 3$. Thus, each direction is given half of the critical resource budget, or equivalently $\overline{f_\kappa}/2 = 2$, meaning that nodes with g_3-value greater than 2 are not expanded. We now briefly explain iterations (It.) of RCEBDA* that lead to finding the first solution path. The iterations are shown in Table 1. For each direction d, we show the trace of its priority queue $Open^d$, and also, in separate columns, changes on $\mathcal{X}^d$ and $\mathcal{X}^d_{Dom}$ of expanded states during the iteration. Expanded nodes are denoted by an asterisk next to them (*). The last column shows changes on $Sols$. Before the first iteration starts, the algorithm inserts one initial node to the priority queue of each direction. If both queues exhibit the same smallest f_1-value, we prioritize the forward search.

It.1: Node x_1, with u_s, is extracted from the forward direction. x_1 is not dominated, thus it will be stored in $\mathcal{X}^f(u_s)$. Its critical resource usage is also within the predefined perimeter. Thus, x_1 will be expanded, yielding two new nodes x_3 and x_4 with states u_1 and u_3, respectively. Note that u_2 is already removed from the search graph. Both nodes will be added to $Open^f$ since they are within the bounds.

It.2: x_3 is extracted (with u_1). This node is non-dominated, and thus can be stored in $\mathcal{X}^f(u_1)$. Since x_3's critical resource usage is smaller than the allocated budget, it will be expanded. Thus, a new node x_5 will be added to the queue.

It.3: x_5 is extracted (with u_3). x_5 is non-dominated and can be expanded. Two new nodes x_6 (with u_g) and x_7 (with u_4) are generated. The first node x_6 appears out of bounds as we obtain $\mathbf{f}(x_6) = (3, 5, 5)$ during the expansion. Node x_7, however, is within the bounds and will be added to $Open^f$.

It.4: Now x_2 is extracted from the *backward* direction. Expansion of x_2 adds two new nodes x_8 and x_9 into $Open^b$.

It.5: x_8 from the backward direction is extracted. x_8 is non-

It.	$Open^f : [\mathbf{f}(x), \mathbf{g}(x), s(x)]$	$\mathcal{X}^f$	$\mathcal{X}^f_{Dom}$	$Open^b : [\mathbf{f}(x), \mathbf{g}(x), s(x)]$	$\mathcal{X}^b$	$\mathcal{X}^b_{Dom}$	$Sols$
1	$^*x_1$=[(3,3,3), (0,0,0), u_s]	$\mathcal{X}^f(u_s)$={x_1}		x_2=[(3,3,3), (0,0,0), u_g]			
2	$^*x_3$=[(3,4,4), (1,1,1), u_1] x_4=[(4,3,3), (3,1,1), u_3]	$\mathcal{X}^f(u_1)$={x_3}		x_2=[(3,3,3), (0,0,0), u_g]			
3	$^*x_5$=[(3,4,4), (2,2,2), u_3] x_4=[(4,3,3), (3,1,1), u_3]	$\mathcal{X}^f(u_3)$={x_5}		x_2=[(3,3,3), (0,0,0), u_g]			
4	x_4=[(4,3,3), (3,1,1), u_3] x_7=[(5,4,4), (3,3,3), u_4]			$^*x_2$=[(3,3,3), (0,0,0), u_g]	$\mathcal{X}^b(u_g)$={x_2}		
5	x_4=[(4,3,3), (3,1,1), u_3] x_7=[(5,4,4), (3,3,3), u_4]			$^*x_8$=[(3,4,4), (1,3,3), u_3] x_9=[(5,3,3), (2,1,1), u_4]	$\mathcal{X}^b(u_3)$={x_8}		
6	$^*x_4$=[(4,3,3), (3,1,1), u_3] x_7=[(5,4,4), (3,3,3), u_4]	$\mathcal{X}^f(u_3)$={x_4}	$\mathcal{X}^f_{Dom}(u_3)$={x_5}	x_9=[(5,3,3), (2,1,1), u_4]			{(x_4, x_8)}

Table 1: Trace of $Open^d$ and $Sols$ in the first 6 iterations (It.) of RCEBDA*. The node extracted in each iteration is marked with symbol *. We also show changes on the $\mathcal{X}^d$ and $\mathcal{X}^d_{Dom}$ lists of the explored states in separate columns for both directions.

dominated and thus will be stored in $\mathcal{X}^b(u_3)$. However, x_8 is not expanded as its critical resource usage is not within the allocated budget, i.e, we have $g_3(x_8) \nleq 2$.

It.6: x_4 from the *forward* direction is extracted. There is already one node, x_5, in the $\mathcal{X}^f(u_3)$ list, which cannot dominate x_4. However, we observe that the truncated cost vector of x_5 is dominated by that of x_4. Thus, x_5 will be moved to $\mathcal{X}^f_{Dom}(u_3)$ before adding x_4 to $\mathcal{X}^f(u_3)$. Checking the nodes explored with u_3 in the backward direction, we find x_8 as a candidate for path matching. Joining x_4 with x_8 yields a complete path that is within the bounds, i.e., $\mathbf{g}(x_4) + \mathbf{g}(x_8) = (4, 4, 4) \preceq \overline{\mathbf{f}}$. Consequently, the upper bound on primary cost can be updated, and (x_4, x_8) can be captured as a solution node pair.

Empirical Analysis

This section evaluates the performance of our proposed algorithm and compares it against the state of the art. For the baseline, we consider the recent ERCA* algorithm (Ren et al. 2023) and the improved version of RCBDA* (Thomas, Calogiuri, and Hewitt 2019) we presented in Algorithm 1.

Benchmark: We use the instances of Ahmadi et al. (2024c), which are defined over the road networks from the 9th DIMACS Implementation Challenge (DIMACS 2005). We choose 20 ($start$,$goal$) pairs over three maps, with the largest map containing over 400K nodes and 1M edges. We study two RCSP scenarios: $k = 3$ (two resources) and $k = 4$ (three resources). The first and second edge costs are *distance* and *time*, respectively. Following Ren et al. (2023), we choose the third cost to be the average (out)degree of the link, that is, the number of adjacent vertices of each end point. We choose the fourth cost of each edge to be one. Each ($start$,$goal$) pair is evaluated on five tightness levels, yielding 100 instances per map per scenario. Following the literature, we define for each resource $i \in \{1, \ldots, k-1\}$ the budget R_i based on the tightness of the constraint δ as:

$$\delta = \frac{R_i - cost^{min}_{i+1}}{cost^{max}_{i+1} - cost^{min}_{i+1}} \quad \text{for } \delta \in \{10\%, 30\%, \ldots, 90\%\}$$

where $cost^{min}_{i+1}$ and $cost^{max}_{i+1}$ are lower and upper bounds on $cost_{i+1}$ of $start$-$goal$ paths, respectively. Each $cost^{max}_{i+1}$ is calculated using the non-constrained $cost_1$-optimal path.

Implementation: We implemented our RCEBDA* and the improved version of RCBDA* in C++ and used the publicly available C++ implementations of ERCA*. All C++ code was compiled using the GCC7.5 compiler with O3 optimization settings. In addition, we implemented a variant of our RCEBDA* where the bidirectional searches are conducted concurrently, denoted RCEBDA*$_{\text{par}}$. Similar to the extended parallel weight constrained search in Ahmadi et al. (2024c), we allocate one thread to each direction and allow the forward (resp. backward) search to work on $Open^f$ (resp. $Open^b$) independently. The overall search in this framework terminates when both concurrent searches have exited (see Ahmadi et al. (2024c) for more details). We choose the last edge attribute as the critical resource in both RCBDA* and RCEBDA*. In addition, following Ahmadi et al. (2024b), we store non-dominated nodes of each state in a dynamic array, in which nodes are maintained in lexicographical order of their truncated cost vector. We ran all experiments once on a single core of an Intel Xeon Platinum 8488C processor running at 2.4 GHz, with 32 GB of RAM and a one-hour timeout. Our code is publicly available[1].

Table 2 presents the results for both scenarios. We report the number of solved cases $|\mathcal{S}|$ and runtime statistics (in seconds, including the initialization time). We also design a virtual best oracle, which, for each instance, provides the best runtime achieved by either of the algorithms. The last column of the table shows the average slowdown factor ϕ of each algorithm compared to the virtual best oracle over mutually solved instances of each map. We observe that, while RCEBDA* and RCEBDA*$_{\text{par}}$ successfully solved 100% of instances within 20 minutes, RCBDA* and ERCA* were unable to solve several instances of each map in both scenarios within the one-hour timeout. Interestingly, the (improved) RCBDA* algorithm has consistently performed better than ERCA* in all maps in terms of solved cases and average runtime. Comparing the average slowdown factors, we find both RCEBDA* and RCEBDA*$_{\text{par}}$ performing close to the virtual best oracle, with the parallel variant showing slightly better performance. Our detailed results show that this superior performance is primarily driven by the proposed search

[1] https://bitbucket.org/s-ahmadi/multiobj

Map	Algorithm	$\lvert\mathcal{S}\rvert$	$t_{\min}$	t_{avg}	$t_{\max}$	ϕ
First scenario with two resources ($k = 3$)						
NY	RCBDA*	98	0.37	153.76	3600	19.4
	ERCA*	97	1.46	162.10	3600	40.0
	RCEBDA*	100	0.20	1.82	29.8	1.2
	RCEBDA*$_{\text{par}}$	100	0.21	1.74	30.2	1.0
BAY	RCBDA*	100	0.47	67.82	1434.3	8.8
	ERCA*	93	1.62	290.02	3600	43.0
	RCEBDA*	100	0.24	3.12	51.3	1.1
	RCEBDA*$_{\text{par}}$	100	0.24	2.43	33.0	1.0
COL	RCBDA*	90	0.63	541.77	3600	196.1
	ERCA*	79	2.23	818.82	3600	95.0
	RCEBDA*	100	0.33	15.57	247.9	1.1
	RCEBDA*$_{\text{par}}$	100	0.33	13.45	191.8	1.0
Second scenario with three resources ($k = 4$)						
NY	RCBDA*	94	0.50	368.90	3600	16.6
	ERCA*	88	2.07	493.89	3600	51.4
	RCEBDA*	100	0.24	7.30	139.5	1.1
	RCEBDA*$_{\text{par}}$	100	0.24	7.71	156.7	1.0
BAY	RCBDA*	94	0.63	335.70	3600	14.6
	ERCA*	86	2.38	557.47	3600	111.6
	RCEBDA*	100	0.29	11.68	190.6	1.1
	RCEBDA*$_{\text{par}}$	100	0.27	9.55	138.4	1.0
COL	RCBDA*	80	0.83	851.59	3600	159.4
	ERCA*	73	3.22	1094.83	3600	81.9
	RCEBDA*	100	0.41	76.70	1252.7	1.1
	RCEBDA*$_{\text{par}}$	100	0.40	72.35	1095.4	1.1

Table 2: Runtime statistics of the algorithms (t in seconds) with two and three resources. $\lvert\mathcal{S}\rvert$ is the number of solved cases (out of 100), and ϕ shows the average slowdown factor of mutually solved cases with respect to a virtual best oracle. Runtime of unsolved instances is considered to be one hour.

engine enhancements, especially efficient dominance pruning, followed by the optimized two-stage path-matching strategy and, to a lesser extent, quick dominance pruning. The other two algorithms, RCBDA* and ERCA*, perform up to two orders of magnitude slower than the virtual best oracle on average, highlighting the success of our proposed algorithm in solving challenging RCSP instances faster. To better compare the algorithms for their performance, Figure 2 displays the runtimes of RCEBDA* and RCEBDA*$_{\text{par}}$ versus RCBDA* across all instances with three resources. Despite the benchmark including numerous easy instances, we observe that RCEBDA* and RCEBDA*$_{\text{par}}$ demonstrate superior performance in the more challenging instances, outperforming RCBDA* by over two orders of magnitude. Note that the comparison of the three algorithms here is head-to-head, as they are implemented within the same framework.

To study the impact of initialization, we experimented a variant of RCEBDA* in which the upper bound pruning of states is disabled (line 5 of Algorithm 3). Figure 3 depicts, for each tightness level, the distribution of slowdown factors obtained with this variant with respect to the standard variant across all instances with $k = 4$. We observe that, while disabling state upper bounding during initialization results in slower performance across all tightness levels, the impact is more pronounced in tight constraints (10% and 30%), where it leads to an average slowdown exceeding a factor of two.

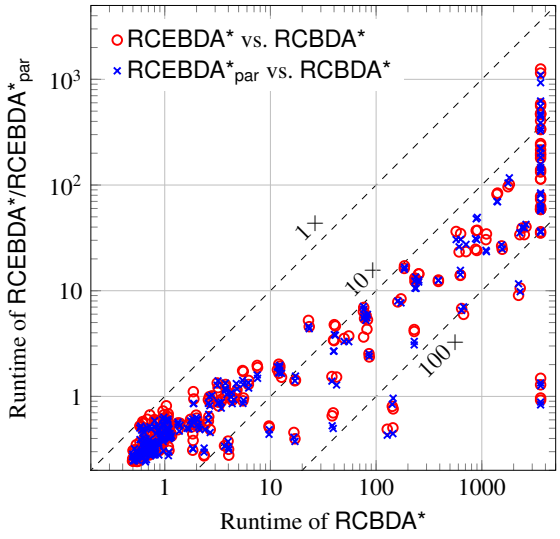

Figure 2: Runtime distribution of RCEBDA*$_{\text{par}}$ and RCEBDA* versus RCBDA* over all instances with $k = 4$.

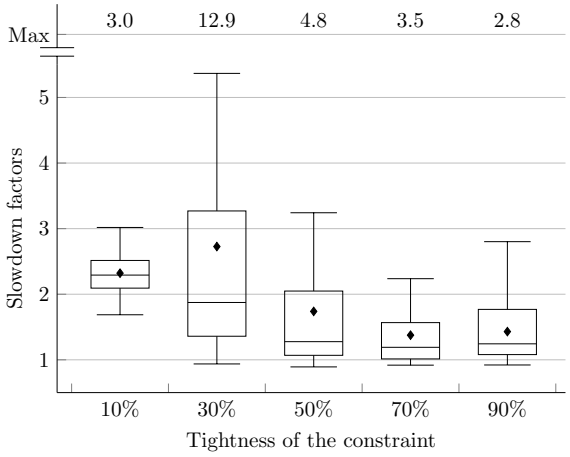

Figure 3: Distribution of slowdown factors for RCEBDA* initialized conventionally. Each plot shows first quartile (25% data), median, third quartile (75% of data), mean (♦), minimum and maximum within the 1.5× interquartile range, and absolute maximum slowdown. Outliers are not shown.

Conclusion

This paper presented RCEBDA*, an enhanced bidirectional A* search algorithm for RCSP. Built on the half-way bidirectional search scheme in the literature, our proposed method introduces several enhancements to the bidirectional constrained A* search, including more efficient initialization, dominance pruning and partial path matching strategies. Our experiments over benchmark instances from the literature demonstrate the success of RCEBDA* in substantially reducing the computation time of RCSP in large-scale graphs, achieving speed-ups of several orders of magnitude compared to the state of the art.

Acknowledgements

This research was supported by the Department of Climate Change, Energy, the Environment and Water under the International Clean Innovation Researcher Networks (ICIRN) program grant number ICIRN000077. Mahdi Jalili is supported by Australian Research Council through projects DP240100963, DP240100830, LP230100439 and IM240100042.

References

Ahmadi, S.; Raith, A.; Tack, G.; and Jalili, M. 2024a. Resource Constrained Pathfinding with Enhanced Bidirectional A* Search. *arXiv preprint arXiv:2412.13888*.

Ahmadi, S.; Sturtevant, N. R.; Harabor, D.; and Jalili, M. 2024b. Exact Multi-objective Path Finding with Negative Weights. In Bernardini, S.; and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*, 11–19. AAAI Press.

Ahmadi, S.; Tack, G.; Harabor, D.; and Kilby, P. 2022. Weight Constrained Path Finding with Bidirectional A*. In Chrpa, L.; and Saetti, A., eds., *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria, July 21-23, 2022*, 2–10. AAAI Press.

Ahmadi, S.; Tack, G.; Harabor, D.; Kilby, P.; and Jalili, M. 2024c. Enhanced methods for the weight constrained shortest path problem. *Networks*, 84(1): 3–30.

Ahmadi, S.; Tack, G.; Harabor, D. D.; and Kilby, P. 2021. A Fast Exact Algorithm for the Resource Constrained Shortest Path Problem. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Event, February 2-9, 2021*, 12217–12224. AAAI Press.

Alcázar, V. 2021. The Consistent Case in Bidirectional Search and a Bucket-to-Bucket Algorithm as a Middle Ground between Front-to-End and Front-to-Front. In Biundo, S.; Do, M.; Goldman, R.; Katz, M.; Yang, Q.; and Zhuo, H. H., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*, 7–15. AAAI Press.

Aneja, Y. P.; Aggarwal, V.; and Nair, K. P. K. 1983. Shortest chain subject to side constraints. *Networks*, 13(2): 295–302.

Cabrera, N.; Medaglia, A. L.; Lozano, L.; and Duque, D. 2020. An exact bidirectional pulse algorithm for the constrained shortest path. *Networks*, 76(2): 128–146.

Cortez, A. C. 2022. *Path Planning with Dynamic Obstacles and Resource Constraints*. Master's thesis, The Ohio State University.

DIMACS. 2005. 9th DIMACS Implementation Challenge - Shortest Paths. http://www.diag.uniroma1.it/challenge9. Accessed: 2025-01-20.

Ferone, D.; Festa, P.; Fugaro, S.; and Pastore, T. 2020. On the Shortest Path Problems with Edge Constraints. In *22nd International Conference on Transparent Optical Networks, ICTON 2020, Bari, Italy, July 19-23, 2020*, 1–4. IEEE.

Golden, B. L.; and Shier, D. R. 2021. 2019–2020 Glover-Klingman Prize Winners. *Networks*.

Handler, G. Y.; and Zang, I. 1980. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4): 293–309.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2): 100–107.

Hernández, C.; Yeoh, W.; Baier, J. A.; Felner, A.; Salzman, O.; Zhang, H.; Chan, S.-H.; and Koenig, S. 2023. Multi-objective search via lazy and efficient dominance checks. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 7223–7230.

Johnson, D. B. 1973. A note on Dijkstra's shortest path algorithm. *Journal of the ACM (JACM)*, 20(3): 385–388.

Lozano, L.; and Medaglia, A. L. 2013. On an exact method for the constrained shortest path problem. *Comput. Oper. Res.*, 40(1): 378–384.

Parmentier, A.; Martinelli, R.; and Vidal, T. 2023. Electric Vehicle Fleets: Scalable Route and Recharge Scheduling Through Column Generation. *Transp. Sci.*, 57(3): 631–646.

Pohl, I. 1971. Bi-directional search. *Machine intelligence*, 6: 127–140.

Pugliese, L. D. P.; and Guerriero, F. 2013. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3): 183–200.

Pulido, F. J.; Mandow, L.; and Pérez-de-la-Cruz, J. 2015. Dimensionality reduction in multiobjective shortest path search. *Comput. Oper. Res.*, 64: 60–70.

Ren, Z.; Rubinstein, Z. B.; Smith, S. F.; Rathinam, S.; and Choset, H. 2023. ERCA*: A New Approach for the Resource Constrained Shortest Path Problem. *IEEE Transactions on Intelligent Transportation Systems*.

Ren, Z.; Zhan, R.; Rathinam, S.; Likhachev, M.; and Choset, H. 2022. Enhanced Multi-Objective A* Using Balanced Binary Search Trees. In Chrpa, L.; and Saetti, A., eds., *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria, July 21-23, 2022*, 162–170. AAAI Press.

Righini, G.; and Salani, M. 2006. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discret. Optim.*, 3(3): 255–273.

Sedeño-Noda, A.; and Alonso-Rodríguez, S. 2015. An enhanced K-SP algorithm with pruning strategies to solve the constrained shortest path problem. *Appl. Math. Comput.*, 265: 602–618.

Siag, L.; Shperberg, S. S.; Felner, A.; and Sturtevant, N. R. 2023. Front-to-End Bidirectional Heuristic Search with Consistent Heuristics: Enumerating and Evaluating Algorithms and Bounds. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, 5631–5638. ijcai.org.

Thomas, B. W.; Calogiuri, T.; and Hewitt, M. 2019. An exact bidirectional A* approach for solving resource-constrained shortest path problems. *Networks*, 73(2): 187–205.