A Fast and Simple Algorithm for the Resource Constrained Shortest Path Problem

Saman Ahmadi
School of Engineering, RMIT University, Melbourne, Australia

Andrea Raith
Department of Engineering Science, University of Auckland, New Zealand

Mahdi Jalili
School of Engineering, RMIT University, Melbourne, Australia

Abstract

Constrained pathfinding is a classic yet challenging network optimization problem with broad applicability across many real-world domains. The Resource-Constrained Shortest Path (RCSP) problem focuses on finding cost-optimal paths that satisfy multiple resource constraints. In this paper, we propose a novel heuristic-guided search framework that accelerates constrained search in large-scale networks, including those with negative costs and resources, by leveraging efficient queuing and pruning strategies. Experimental results on real-world benchmark maps show that our framework achieves up to two orders of magnitude speedup over state-of-the-art methods, demonstrating its effectiveness in solving challenging RCSP instances within limited time.

2012 ACM Subject Classification Computing methodologies → Search methodologies; Theory of computation → Shortest paths

Keywords and phrases constrained pathfinding, shortest path problem, heuristic search

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.97

Related Version *arXiv version*: `https://doi.org/10.48550/arXiv.2503.11037`

Funding 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.

1 Introduction

The Resource Constrained Shortest Path (RCSP) problem is a fundamental yet challenging network optimization problem. It focuses on finding cost-optimal paths between two nodes in a graph while adhering to constraints on the utilization of limited, potentially multidimensional resources. Many real-world problems can be modeled as an RCSP instance. Examples are planning time-constrained explorable non-shortest routes in roaming navigation applications [24], and finding energy efficient paths with detour limits for electric vehicles [4]. As a subproblem, RCSP has been utilized to solve orienteering problems [33], and the vehicle routing problem [12]. RCSP is an NP-Hard problem even for single resource constraint [16].

RCSP and its single-constraint variant, the Weight Constrained Shortest Path (WCSP) problem, have been studied in the literature for decades. Traditional solutions to RCSP are generally based on path ranking, dynamic programming (labeling) or branch-and-bound (B&B) approaches [25, 13], but recent studies have demonstrated that the labeling method is more effective for solving large-scale RCSP instances. The recent RCBDA* algorithm [31] is a labeling method that utilizes the dynamic programming approach presented in [29] to perform a bidirectional A* search guided by cost heuristics, using rigorous yet costly pruning rules to avoid exploring unpromising paths. Complete paths in this algorithm are obtained by matching backward and forward partial paths, and the amount of a critical resource available to each search is half of the total budget. The authors concluded that RCBDA* can perform faster than two other existing RCSP approaches on many instances, namely the Pulse algorithm (B&B method) [22] and the CSP (path ranking) approach in [30]. A parallel bidirectional search framework on the basis of Pulse was later developed in [10], known as BiPulse. This algorithm utilizes the the queuing mechanism developed in [9] to limit the depth of the Pulse search and postpone the exploration of deep branches of the search tree, where halted partial paths are explored in a breadth-first search manner. The authors reported better performance and more solved cases with BiPulse compared to RCBDA*. Both BiPulse and RCBDA* received awards for their contribution to RCSP [15]. WCEBBA* [3], WCA* [6] and WCBA* [5] are three other A*-based methods that leverage fast pruning rules in recent bi-objective search literature [32, 2] to tackle constrained pathfinding more efficiently. Although all of the algorithms have been shown to outperform RCBDA* and BiPulse on large graphs, their application is limited to instances with a single resource constraint. Another recent A*-based RCSP method, called ERCA* [27], takes advantage of recent advancements in multi-objective search with A* [26, 28] to improve the efficiency of its pruning rules through binary search trees. Like other A*-based methods, the search in ERCA* is guided by cost heuristics and prunes unpromising paths. Its pruning involves rigorously discarding any path that exceeds the resource budget or is dominated (i.e., no better in cost and resource usage) by another previously extended path to the same node, both before and during path extension. The authors reported several orders of magnitude faster runtime for ERCA* when compared with BiPulse. However, the performance of ERCA* against RCBDA* remains unknown. All of the above algorithms are designed and evaluated for problem instances with non-negative cost and resources.

Contributions. This paper presents a novel and straightforward label-setting framework built upon A*, called NWRCA*, which incorporates effective search and lazy pruning strategies to accelerate constrained pathfinding in large-scale networks with negative costs and resources. We evaluate the framework in two variants that differ slightly in how they maintain search labels, and we investigate a faster method for computing A*'s heuristics that can potentially replace the conventional reweighting approach when dealing with negative-weight graphs. Extensive experiments demonstrate the success of the framework in solving difficult RCSP instances several orders of magnitude faster than ERCA* and RCBDA*.

2 Problem Definition

Consider an RCSP problem with d resource limits $\{R_1, R_2, \ldots, R_d\}$ is provided as a directed graph G with a set of vertices V and a set of edges $E \subseteq V \times V$. Every edge $e \in E$ of the graph has $d+1 \in \mathbb{N}$ attributes, represented as $(cost, \mathbf{resources})$ where $\mathbf{resources} = \{resource_1, \ldots, resource_d\}$ is a form of vector (denoted boldface). An acyclic path π consists of n distinct vertices $\{v_1, v_2, v_3, \ldots, v_n\}$, where $n \leq |V|$ and $(v_i, v_{i+1}) \in E$ for $i \in \{1, \ldots, n-1\}$. The cost and resource usage of a path π are determined by summing the attributes of all edges that make up the path. The RCSP problem aims to find (at least) one *cost*-optimal paths π^* from $v_s \in V$ (start or origin) to $v_g \in V$ (goal or destination) such that the resource usages of the path are within the given limits, that is, we must have $resource_k(\pi^*) \leq R_k$ for each resource $k \in \{1, \ldots, d\}$.

In line with the notation in the RCSP literature, we refer to our search objects as *labels*. Each label l encapsulates the key information about a partial path from v_s and is associated with an end vertex $v \in V$ (last vertex of the path). Label l traditionally stores value pairs

g_l and $\mathbf{r}_l$ that measure the *cost* and **resources** of the path, respectively. Additionally, l includes p_l, a reference to the parent label of l, and f_l, an estimate of the *cost* of a complete path to v_g by extending l. Specifically, for each search label l associated with vertex v, we have $f_l = g_l + h_c(v)$ where $h_c(v)$ is a consistent *cost* heuristic function [17].

▶ **Definition 1.** *The heuristic function $h_c : V \to \mathbb{R}$ is admissible iff $h_c(v) \leq cost(\lambda^*)$ for every $v \in V$, where λ^* is a cost-optimal path from v to v_g. h_c is consistent if we have $h_c(v) \leq cost(v, u) + h_c(u)$ for every $(v, u) \in E$.*

We assume all operations on the resource vectors are performed element-wise. In addition, we use the $\preceq$ symbol for direct comparisons of resource vectors, e.g., $\mathbf{r}_l \preceq \mathbf{r}_{l'}$ denotes $resource_k$ of label l is no greater than that of label l' for all $k \in \{1, \ldots, d\}$. Analogously, we use the $\npreceq$ symbol if the relation cannot be satisfied. We now define *dominance* over labels.

▶ **Definition 2.** *Label l' is weakly dominated by label l if we have $g_l \leq g_{l'}$ and $\mathbf{r}_l \preceq \mathbf{r}_{l'}$; l' is strongly dominated by l if $g_l \leq g_{l'}$ and $\mathbf{r}_l \preceq \mathbf{r}_{l'}$ but $g_l \neq g_{l'}$ or $\mathbf{r}_l \neq \mathbf{r}_{l'}$; l' is not dominated by l if $g_l \nleq g_{l'}$ or $\mathbf{r}_l \npreceq \mathbf{r}_{l'}$.*

3 Resource Constrained Pathfinding with A*

Constrained pathfinding with A* involves a systematic search by exploring labels in best-first order, meaning that the search is guided by the partial path with the smallest *cost* estimate or f-value. Each iteration involves three main steps:
i) Extraction: remove one least-*cost* label from a priority queue $\mathcal{Q}$, also known as *Open* list.
ii) Dominance pruning: skip exploring labels that are weakly dominated by at least one other label (with the same vertex);
iii) Expansion: generate new descendant labels, check them for feasibility (against resource limits), and, if not pruned, store them in $\mathcal{Q}$ for further expansion.
The search in this framework generally terminates once an optimal solution is found, or there is no label in $\mathcal{Q}$ to explore, where the latter means there is no feasible solution. If there is only one resource, it is shown that dominance pruning can be done in constant time when performing best-first search [5, 6]. Nonetheless, this is not the case in RCSP with multiple resources and the dominance pruning remains a costly task.

RCSP with negative weights. Many real-world RCSP problems deal with attributes that are negative in nature, such as energy recuperation in electric vehicles, or attributes that may exhibit negative values in specific circumstances, such as negative reduced costs in column generation or pricing problems [19]. Although recent A*-based RCSP solutions, such as ERCA* and RCBDA*, are primarily designed and evaluated for problem instances with non-negative edge attributes, they can be adapted to be used for instances with negative weights (but no negative cycles). This can be achieved through a graph reformulation phase, where all edge weights are shifted to non-negative values, for example, using Johnson's reweighting technique [21]. However, as we will show in this section, our constrained search framework can be applied directly to graphs with negative edge weights, provided that its *cost* heuristic is consistent. With this introduction, we now describe our new RCSP algorithm.

Our approach. Algorithm 1 presents the pseudocode of NWRCA*, our A*-based RCSP method that employs *lazy* dominance checks and can handle problem instances with negative edge weights (but no negative cycles). The algorithm begins by initializing a priority queue $\mathcal{Q}$ and a global cost upper bound $\overline{f}$, which maintains the best-known cost of any feasible solution found during the search. For each vertex $v \in V$, it also initializes a list $\mathcal{X}(v)$ to store

Algorithm 1 NWRCA*.

```
   Input: An RCSP Problem $(G, v_s, v_g, \mathbf{R})$
   Output: A set of non-dominated cost-optimal solution labels
 1 $\mathcal{Q} \leftarrow \emptyset$ , $\overline{f} \leftarrow \infty$
 2 $\mathcal{X}(v) \leftarrow \emptyset \ \ \forall v \in V$
 3 $h_c(v) \leftarrow$ optimal cost from $v$ to $v_g$ $\forall v \in V$
 4 $\mathbf{h}_r(v) \leftarrow$ lower bound on resources from $v$ to $v_g$ $\forall v \in V$
 5 $l \leftarrow$ new label associated with $v_s$
 6 $g_l \leftarrow 0$ , $\mathbf{r}_l \leftarrow \mathbf{0}_d$ , $f_l \leftarrow h_c(v_s)$, $p_l \leftarrow \varnothing$
 7 add $l$ to $\mathcal{Q}$
 8 while $\mathcal{Q} \neq \emptyset$ do
 9     extract from $\mathcal{Q}$ label $l$ with the smallest $f$-value
10     if $f_l > \overline{f}$ then break
11     $v \leftarrow$ vertex associated with the extracted label $l$
12     dominated $\leftarrow$ false
13     for each $l' \in \mathcal{X}(v)$ do
14         if $\mathbf{r}_{l'} \preceq \mathbf{r}_l$ then
15             dominated $\leftarrow$ true
16             break
17     if dominated = true then continue
18     for each $l' \in \mathcal{X}(v)$ do
19         if $\mathbf{r}_l \preceq \mathbf{r}_{l'}$ then remove $l'$ from $\mathcal{X}(v)$
20     add $l$ to $\mathcal{X}(v)$
21     if $v = v_g$ then
22         $\overline{f} \leftarrow f_l$
23         continue
24     for each successor $u$ of $v$ do
25         $l' \leftarrow$ new label associated with $u$
26         $g_{l'} \leftarrow g_l + cost(v, u)$
27         $\mathbf{r}_{l'} \leftarrow \mathbf{r}_l + \mathbf{resources}(v, u)$
28         $f_{l'} \leftarrow g_{l'} + h_c(u)$
29         $p_{l'} \leftarrow l$
30         if $\mathbf{r}_{l'} + \mathbf{h}_r(u) \not\preceq \mathbf{R}$ then continue
31         add $l'$ to $\mathcal{Q}$
32 return $\mathcal{X}(v_g)$
```

the non-dominated labels resulting from all prior expansions (i.e., closed labels) associated with vertex v. Like other A*-based RCSP algorithms, NWRCA* requires a *cost* heuristic function h_c to establish f-values. To enhance pruning of unpromising paths, we also compute lower bounds on **resources** using an admissible heuristic function $\mathbf{h}_r : V \to \mathbb{R}^d$. These cost and resource heuristics can be computed using $1 + d$ single-objective Dijkstra runs [11], with re-expansions allowed to accommodate negative weights [20], assuming there are no negative cycles on any path from v_s to v_g in any dimension (cost or resources). A similar strategy has been explored in [1] for multi-objective search. As we will demonstrate in the experimental section, this heuristic construction method, despite its exponential worst-case complexity, often outperforms alternatives such as Johnson's reweighting technique [21] in practice. NWRCA* then initializes a label with v_s and inserts it into $\mathcal{Q}$. Each iteration of the main search starts at line 8. Let $\mathcal{Q}$ be a non-empty queue. The algorithm extracts in each iteration a label l with the smallest f-value (line 9) and attempts the following steps.

Lazy dominance check. NWRCA* borrows a lazy label pruning technique from the multi-objective search literature and delays dominance check of newly generated labels until they are extracted from $\mathcal{Q}$, as in [18] and [1]. Let l be a label extracted from $\mathcal{Q}$ with f-value within the global upper bound. Also let v be the end vertex associated with l. Because the first dimension is always explored in sorted f order (guaranteed in A* with a consistent heuristic), we observe that l is already weakly dominated by previous expansions in terms of *cost*. Note that all expansions with v use $h_c(v)$ as lower bound. As a result, all prior expansions of v must have exhibited g-value no smaller than g_l. This means that the dominance test can only be done by comparing the resource vector of l, that is, $\mathbf{r}_l$, with that of (potentially all) previous expansions stored in $\mathcal{X}(v)$. This process can be further improved by prioritizing the most recent expansion of v as the first (potentially more informed) candidate for a quick dominance check, prior to a more rigorous dominance check against the labels of $\mathcal{X}(v)$ [1]. The expansion of label l can be skipped if it is found to be dominated by any previously expanded label (lines 13-16). If l is determined to be non-dominated, NWRCA* removes from the $\mathcal{X}(v)$ list any label l' whose resource vector is weakly dominated by $\mathbf{r}_l$ (lines 18-19), and then adds l to $\mathcal{X}(v)$ for future dominance checks at vertex v (line 20). This removal is correct because any future label that would be dominated by l' will also be dominated by l, making it unnecessary to keep the now-redundant label in $\mathcal{X}(v)$.

Capturing solutions. A tentative solution path is found if the search extracts a non-dominated label associated with the goal vertex v_g (line 21). Expansion of solution labels is not necessary. However, in the absence of tie-breaking in $\mathcal{Q}$, new solutions may offer better resources than previous ones. NWRCA* takes care of this situation by maintaining only non-dominated labels associated with the goal vertex in $\mathcal{X}(v_g)$.

Expansion. l's expansion involves generating new descendant labels through v's successors (adjacent vertices). Given $\mathbf{h}_r$ as an admissible heuristic function, descendant labels with estimated resource usage exceeding the given limit $\mathbf{R}$ cannot lead to a feasible solution and can be pruned (line 30). Note that in the case of negative resources, new labels whose usage of **resources** exceeds the budget (i.e., $\mathbf{r}_{l'} \not\preceq \mathbf{R}$) can be pruned, provided that all partial paths are also required to satisfy the resource constraints (which is not assumed in this work). As part of NWRCA*'s lazy dominance pruning rule, newly generated labels are not checked for dominance against previously expanded labels or those still in $\mathcal{Q}$; instead, they are only checked when extracted from the queue. Thus, each descendant label will be added to $\mathcal{Q}$ if its estimated usage of **resources** to v_g is within $\mathbf{R}$.

Termination. NWRCA* explores labels based on their f-value in ascending order. Given $\overline{f}$ as global *cost* upper bound, the search can terminate safely once it extracts a label with f-value greater than $\overline{f}$ (line 10). Although this upper bound is initially unknown, it will get updated as soon as a feasible solution is discovered (line 22). The algorithm returns $\mathcal{X}(v_g)$, a set containing *cost*-optimal labels with non-dominated **resources** (line 32). If there is no solution to a given instance (i.e., v_g is never reached), this set would remain empty.

Example. We further elaborate on the key steps of NWRCA* by solving a sample RCSP instance with three edge attributes (two resources), depicted in Figure 1. We assume resource limits are $(R_1, R_2) = (3, 3)$. The vertices v_s and v_g denote *start* and *goal*, respectively. The graph is free of negative cycles, and the values inside each vertex denote h_c and (h_{r1}, h_{r2}), calculated prior to the main search via three single-objective backward searches. We briefly

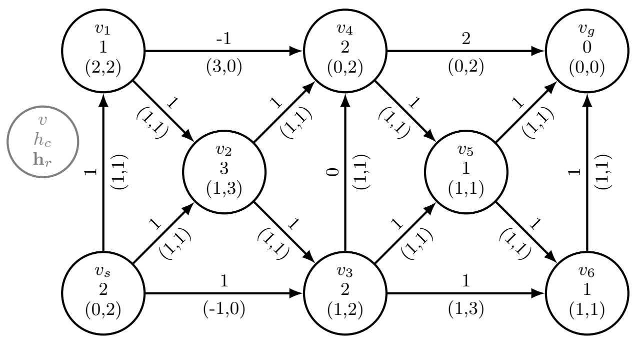


Figure 1 An RCSP instance with *cost* above and **resources** below edges ($d = 2$). Values inside the nodes denote vertex identifier, h_c and $\mathbf{h}_r$ of each vertex, listed from top to bottom.

explain all iterations (It.) of NWRCA* for the given instance, with the trace of $\mathcal{Q}$ and $\mathcal{X}$ sets provided in Table 1. Pruned labels of each iteration are shown in the third column. For label extractions, we adopt the Last-In, First-Out (LIFO) strategy in the event of ties in f-values of labels present in $\mathcal{Q}$. The queue is initialized with the first label l_1 associated with v_s.

- **It.1:** The search starts with expanding the initial label l_1. Since l_1 is the first expansion of v_s and thus non-dominated, it is stored in $\mathcal{X}(v_s)$. Expansion of l_1 generates three new labels: l_2, l_3, and l_4. l_2 and l_4 are added to $\mathcal{Q}$ but l_3 is pruned as its resource estimates are out of bounds, i.e., we have $\mathbf{r}_{l_3} + \mathbf{h}_r(v_2) \not\preceq \mathbf{R}$.
- **It.2:** There are two labels in $\mathcal{Q}$. l_2 has the smallest f-value and is expanded first. l_2 is non-dominated and its expansion generates l_5 and l_6 with v_2 and v_4, respectively. Both labels are pruned since their resource estimates are out of bounds.
- **It.3:** l_4, associated with v_3, is the only label in $\mathcal{Q}$ and is extracted. l_4 will be stored in $\mathcal{X}(v_3)$ as a non-dominated label. l_4's expansion generates three new labels: l_7, l_8, and l_9. However, only two of these are within the resource bounds and can be added to $\mathcal{Q}$. l_9 is out of bounds as we have $3 + 1 \not\leq 3$ for its second resource estimate.
- **It.4:** There are two labels in $\mathcal{Q}$, both with the same f-value. l_8 is extracted based on the LIFO strategy. l_8 is non-dominated, as this is the first expansion with v_4. Expansion of l_8 generates two new labels (l_{10} and l_{11}, both within the bounds) and adds them to $\mathcal{Q}$.
- **It.5:** l_{11} is extracted based on the LIFO strategy. l_{11} is non-dominated and its expansion generates two labels l_{12} and l_{13}, but only l_{12} is added into $\mathcal{Q}$ as the second resource estimate for l_{13} is not within the bound, i.e., we have $3 + 1 \not\leq 3$.
- **It.6:** There are three labels in $\mathcal{Q}$, all with the same f-value. The recent insertion l_{12} is extracted with v_g. This yields the first solution path, followed by updating $\overline{f} \leftarrow 3$. l_{12} does not get expanded.
- **It.7:** l_{10} is extracted, and it appears as a non-dominated solution. However, l_{10}'s resource vector $(0, 3)$ dominates that of the previous solution. Thus, l_{10} replaces l_{12} in $\mathcal{X}(v_g)$.
- **It.8:** l_7 is extracted. l_7 is a non-dominated label but dominates and replaces l_{11} in $\mathcal{X}(v_5)$. Expansion of l_7 generates two new labels (l_{14} and l_{15}). Both labels are within the resource bounds and thus added into $\mathcal{Q}$.
- **It.9:** l_{14} is extracted with v_g. l_{14} is non-dominated and is added to $\mathcal{X}(v_g)$. l_{14} does not dominate the previous solution.
- **It.10:** l_{15} is extracted. Given the *cost* upper bound updated in **It.6**, the search terminates due to $f_{l_{15}}$ surpassing the upper bound $\overline{f}$, i.e., we have $f_{l_{15}} = 4 > 3$.

Table 1 Tracking $\mathcal{Q}$ and $\mathcal{X}$ lists in each iteration (**It.**) of NWRCA* for Figure 1. The symbol $^{\uparrow}$ denotes the extracted label, and the last column shows pruned labels during the expansion phase.

It.	$\mathcal{Q}:\quad [f_l, g_l, \mathbf{r}_l, v_l, p_l]$	$\mathcal{X}$	Generated but Pruned
1	$^{\uparrow}l_1 = [2, 0, (0, 0), v_s, \varnothing]$	$\mathcal{X}(v_s) = \{l_1\}$	$l_3 = [4, 1, (1, 1), v_2, l_1]$
2	$^{\uparrow}l_2 = [2, 1, (1, 1), v_1, l_1]$	$\mathcal{X}(v_1) = \{l_2\}$	$l_5 = [5, 2, (2, 2), v_2, l_2]$
	$l_4 = [3, 1, (-1, 0), v_3, l_1]$		$l_6 = [2, 0, (4, 1), v_4, l_2]$
3	$^{\uparrow}l_4 = [3, 1, (-1, 0), v_3, l_1]$	$\mathcal{X}(v_3) = \{l_4\}$	$l_9 = [3, 2, (0, 3), v_6, l_4]$
4	$l_7 = [3, 2, (0, 1), v_5, l_4]$		
	$^{\uparrow}l_8 = [3, 1, (0, 1), v_4, l_4]$	$\mathcal{X}(v_4) = \{l_8\}$	
5	$l_7 = [3, 2, (0, 1), v_5, l_4]$		
	$l_{10} = [3, 3, (0, 3), v_g, l_8]$		
	$^{\uparrow}l_{11} = [3, 2, (1, 2), v_5, l_8]$	$\mathcal{X}(v_5) = \{l_{11}\}$	
6	$l_7 = [3, 2, (0, 1), v_5, l_4]$		
	$l_{10} = [3, 3, (0, 3), v_g, l_8]$		
	$^{\uparrow}l_{12} = [3, 3, (2, 3), v_g, l_{11}]$	$\mathcal{X}(v_g) = \{l_{12}\}$	
7	$l_7 = [3, 2, (0, 1), v_5, l_4]$		
	$^{\uparrow}l_{10} = [3, 3, (0, 3), v_g, l_8]$	$\mathcal{X}(v_g) = \{l_{10}\}$	
8	$^{\uparrow}l_7 = [3, 2, (0, 1), v_5, l_4]$	$\mathcal{X}(v_5) = \{l_7\}$	
9	$^{\uparrow}l_{14} = [3, 3, (1, 2), v_g, l_7]$	$\mathcal{X}(v_g) = \{l_{10}, l_{14}\}$	
	$l_{15} = [4, 3, (1, 2), v_6, l_7]$		
10	$^{\uparrow}l_{15} = [4, 3, (2, 3), v_6, l_{11}]$		

The example above shows how NWRCA* processes labels in the order of their f-value. As we observed in **It.6**, NWRCA* may capture a dominated solution in the absence of tie-breaking, but we discussed in **It.7** how the search refines the set $\mathcal{X}(v_g)$ in such circumstances to keep **resources** of solution paths non-dominated. We now briefly discuss the key differences of NWRCA* with two recent RCSP approaches, namely ERCA* and RCBDA*.

NWRCA* vs. ERCA*. Although both algorithms utilize best-first search to guide their constrained search, they differ in four key aspects regarding how labels are handled: (1) ERCA* checks labels for both dominance and resource usage once during expansion and again before expansion, whereas NWRCA* performs upper bound pruning only during expansion and dominance check only before expansion; (2) ERCA* explores labels in lexicographical order of their **resources** in case of ties between f-values in the queue (i.e., it compares labels based on resources in order, starting from the first resource), whereas NWRCA* does not perform tie-breaking; (3) ERCA* uses a specialized and relatively complex data structure (balanced binary search tree) to organize and dominance check labels during the search, whereas expanded labels in NWRCA* are stored in simple lists; (4) ERCA* immediately terminates upon finding a feasible solution, whereas NWRCA* terminates with all non-dominated solutions.

NWRCA* vs. RCBDA*. The algorithms differ in four key aspects: (1) RCBDA* conducts a bidirectional A* search, requiring a specialized and time-intensive procedure to handle frontier collisions (joining bidirectional labels), whereas NWRCA* performs a simple unidirectional search. (2) RCBDA* checks new labels for dominance against all previous expansions of the vertex, whereas NWRCA* only checks against labels with non-dominated **resources**; (3) RCBDA* checks labels for dominance only during expansion, which can lead to the expansion of dominated labels, whereas NWRCA* performs dominance checks before expansion; (4) Similar to ERCA*, RCBDA* does not compute all non-dominated solutions.

4 Theoretical Results

This section provides a formal proof for the correctness of constrained search of NWRCA* and presents theoretical results on why the algorithm can solve RCSP instances with negative weights but no negative cycles. Throughout this section, we assume there is no negative cycle on any path from start vertex (v_s) to goal vertex (v_g) in any dimension, and thus consistent and admissible heuristic functions h_c and $\mathbf{h}_r$ can be computed for the problem instance.

▶ **Lemma 3.** *Assume the constrained A* search is guided by smallest (potentially negative) f-values. Let l_i and l_{i+1} be labels extracted from $\mathcal{Q}$ in two consecutive iterations of the search. We have $f_{l_i} \leq f_{l_{i+1}}$ if h_c is consistent.*

Proof. We distinguish two cases: i) if l_{i+1} was available in $\mathcal{Q}$ at the time l_i was extracted, the lemma is trivially true. ii) otherwise, l_{i+1} is the descendant label of l_i. For an edge (v, u) linking l_i to its descendant l_{i+1}, h_c's consistency ensures $h_c(v) \leq h_c(u) + cost(v, u)$. Adding the cost g_{l_i} to both sides of the inequality yields $f_{l_i} \leq f_{l_{i+1}}$. ◀

▶ **Corollary 4.** *Consider the sequence of labels $(l_1, l_2, ..., l_t)$ extracted from $\mathcal{Q}$. If h_c is consistent, then according to the conditions of Lemma 3, $i \leq j$ guarantees that $f_{l_i} \leq f_{l_j}$. In other words, the f-values of the extracted labels do not decrease throughout the process.*

▶ **Lemma 5.** *Suppose l' is extracted after l, both with the same vertex. l weakly dominates l' if $\mathbf{r}_l \preceq \mathbf{r}_{l'}$.*

Proof. Since l is extracted before l', we have $f_l \leq f_{l'}$ according to Corollary 4, and consequently $g_l \leq g_{l'}$. The other condition $\mathbf{r}_l \preceq \mathbf{r}_{l'}$ means that l' is no smaller than l in all resources, verifying l' is weakly dominated by l. ◀

▶ **Lemma 6.** *Let l' be a label weakly dominated by a previously explored label l, both associated with the same vertex. The expansion of l' is not necessary.*

Proof. We prove this by contradiction, assuming that l's expansion is necessary to obtain a *cost*-optimal and non-dominated solution path. As l weakly dominates l', it holds that $g_l \leq g_{l'}$ and $\mathbf{r}_l \preceq \mathbf{r}_{l'}$. In this scenario, the partial path represented by l' can be replaced with the one represented by l, resulting in a path that is better than or equal to the optimal solution path obtained via l'. If the resulting path is better, it contradicts the assumption of l''s expansion leading to a non-dominated or optimal solution. If both paths are identical in terms of *cost* and **resources**, it becomes clear that the expansion of l has been sufficient, and thus expanding l' is unnecessary, again contradicting the assumption. Therefore, we conclude that expanding the weakly dominated label l' is not necessary. ◀

▶ **Lemma 7.** *The expansion of label l is not necessary if $f_l \npreceq \overline{f}$ or $\mathbf{r}_l + \mathbf{h}_r \npreceq \mathbf{R}$.*

Proof. Given that $\mathbf{h}_r$ is admissible, the second condition ensures that expanding l towards v_g cannot yield a solution within the resource limits. Similarly, since h_c is admissible, the first condition guarantees that l's expansion cannot produce a solution with a *cost* better than the best-known upper bound $\overline{f}$. Therefore, expanding l cannot contribute to any optimal solution path, and thus is not necessary. ◀

▶ **Theorem 8.** *NWRCA* produces all cost-optimal solution paths with non-dominated resources for the RCSP problem.*

Proof. The algorithm explores all feasible search labels from v_s towards v_g in best-first order, in search of all optimal solutions. Labels are checked for dominance (Lemma 5) before expansion, and weakly dominated ones can be safely pruned, as they offer no improvement over previously expanded labels (with the same vertex) in either *cost* or **resources** (Lemma 6). Labels violating the upper bounds can similarly be pruned safely (Lemma 7). The algorithm also retains only non-dominated labels in the $\mathcal{X}$ lists. This procedure is correct because if a recently extracted label l' weakly dominates a previously explored label l, then any future label l'' that would have been weakly dominated by l is already weakly dominated by l', rendering the storage of l redundant. Building on the above, we just need to show that the algorithm terminates with returning non-dominated labels. NWRCA* prunes all weakly dominated labels, but captures all non-dominated labels with v_g. However, since it does not process labels in any specific order of their **resources**, some tentative solutions may later appear dominated. Let l' be a new non-dominated solution extracted after solution l. We must have $f_l = f_{l'}$, otherwise the algorithm was terminated if $f_{l'} > \overline{f} = f_l$. In this case, the dominance check in lines 18-19 of Algorithm 1 removes l from $\mathcal{X}(v_g)$ if it is deemed to be weakly dominated by new solution l' (in terms of **resources**). Once the search exceeds $\overline{f}$, Corollary 4 ensures that expanding the remaining labels in $\mathcal{Q}$ cannot produce solutions with a primary cost better than $\overline{f}$. Thus, the termination criterion is correct, and NWRCA* returns $\mathcal{X}(v_g)$ as the set of all *cost*-optimal, non-dominated solution labels. ◀

5 Empirical Analysis

We evaluate the constrained search performance of NWRCA* against two recent A*-based RCSP approaches: the award-winning RCBDA* algorithm [31] and ERCA* [27]. We did not consider BiPulse [10] as prior work [27] has demonstrated that it is outperformed by ERCA*. The algorithms were evaluated on 600 instances across three maps (NY, BAY, and COL) from the 9th DIMACS Implementation Challenge[1], featuring road networks with 264K to 435K vertices and 733K to 1M edges. For each map, we generated 25 random (v_s,v_g) pairs to be evaluated on four levels of tightness, resulting in 100 test cases. We study each test case in scenarios with two and three resources ($d = 2, 3$). Following the RCSP literature, we define each resource limit R_k for $k \in \{1, \dots, d\}$ based on the tightness of the constraint τ as:

$$\tau = \frac{R_k - h_{rk}}{ub_{rk} - h_{rk}} \quad \text{for } \tau \in \{20\%, 40\%, 60\%, 80\%\}$$

where ub_{rk} and h_{rk} represent the upper and lower bounds on *resource*$_k$ for paths from v_s to v_g, respectively. The upper bound is set by the (non-constrained) *cost*-optimal path. We choose the same τ for all resources.

To evaluate the algorithms on graphs with non-correlated but negative weights, following our previous work [1], we first enriched each map with Shuttle Radar Topography Mission[2] height data, and set the edge weights of each map with four new attributes as follows. The *cost* of each edge is the energy requirement of an electric vehicle with three passengers on board, using the energy model in [7]. The calculated energy can be negative in some downhill links due to energy recuperation. We deliberately did not impose a bound on

[1] `http://www.diag.uniroma1.it/challenge9`
[2] `https://www2.jpl.nasa.gov/srtm/`

battery capacity, enabling the use of plain constrained search for computing energy-efficient paths. We chose a penalized height function for $resource_1$. Given $height(v)$ as elevation of vertex v, the second attribute of link $(v, u) \in E$ is set to $2 \times (height(u) - height(v))$ for uphill links, and $height(u) - height(v)$ for downhill links (negative). For $resource_2$ and $resource_3$, we employed the (reversed) Johnson's technique to generate cycle-free but random negative weights in two steps: i) we first assigned a random integer in the [-100,0] range, known as potential, to each vertex; ii) we then set the third cost of each link $(v, u) \in E$ to be the potential difference between v and u, which was then added by a random integer in the [0, 10] range. The result is two sets of randomized negative weights in the [-100, 110] range with no negative cycle, which ensures the applicability of the selected algorithms over the instances. Note that the first and second dimensions cannot contain negative cycles due to the nature of the chosen metrics. The average percentage of negative weights observed across the maps for $[cost, resrource_1, resrource_2, resrource_3]$ is [10, 42, 45, 45]%, respectively.

Implementation. We used the publicly available C++ implementations of ERCA* and developed an improved version of RCBDA* in C++ based on its provided description. In doing so, we addressed a potential inefficiency in the design related to the label matching strategy of RCBDA*: rather than storing all expanded labels in a large pool and searching for complimentary labels (which adds unnecessary overhead in identifying the same matching vertex), we allocated a separate set for each vertex, optimizing the matching of partial paths during the bidirectional search. Additionally, the original description of RCBDA* does not include any dominance or infeasibility pruning rules. Nonetheless, our implementation incorporates these strategies, thereby improving its search efficiency. Both RCBDA* and ERCA* were provided with reweighted graphs (and budgets), ensuring that instances can be handled correctly in the presence of negative weights. We implemented our NWRCA* algorithm in C++, providing two variants that differ slightly in their label ordering mechanisms:

- NWRCA*$_{\text{v1}}$: (i) labels in $\mathcal{X}$ lists are maintained in lexicographical order based on their $\mathbf{r}$-value, (ii) unexplored labels are ordered in a bucket-based queue and extracted using a LIFO strategy in the event of ties in their f-values (as in WCA* [6]).
- NWRCA*$_{\text{v2}}$: (i) labels in $\mathcal{X}$ lists are not maintained in any specific order, (ii) unexplored labels are ordered in a binary heap queue and extracted in lexicographical order of their resource estimates (i.e., $\mathbf{r} + \mathbf{h}_r$) in the event of ties in their f-values.

These variants allow us to investigate the impact of lexicographical label ordering in both $\mathcal{X}$ and $\mathcal{Q}$ lists on search performance. Note that the correctness of NWRCA* does not depend on labels being ordered lexicographically by *cost* and **resources** in $\mathcal{Q}$. Keeping non-dominated labels in lexicographical order within each $\mathcal{X}$ list in the first variant allows for partial traversal over labels of the list for both dominance checking and removal of dominated labels, but it incurs an additional sorting overhead (see [1] for more details). In contrast, the second variant provides faster insertion/removal of labels to/from the list (with minimal ordering overhead), but requires two linear scans over all labels in $\mathcal{X}$, one to ensure that a newly extracted label is non-dominated and another to remove any dominated labels from the list.

We compiled all C++ code using the GCC7.5 compiler and optimization level O3. The experiments were conducted on a single core of an Intel Xeon Platinum 8175 processor, clocked at 2.5GHz, with 16GB of RAM and a one-hour time limit per run. It is worth mentioning that RCBDA* and both variants of NWRCA* were implemented within the same framework, using the same data structures to manage labels. Therefore, the performance comparisons between the algorithms are head-to-head. Our code is publicly available[3].

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

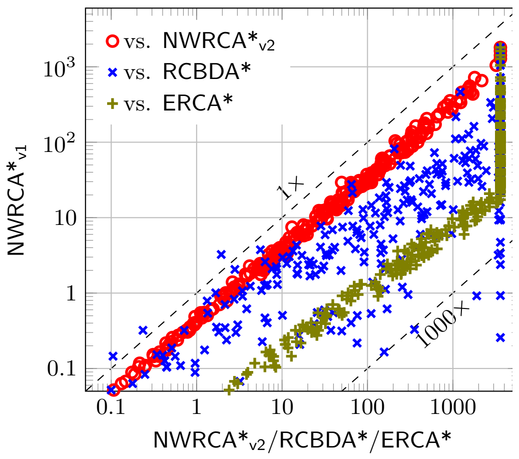
Table 2 Average reweighting and lower-bounding time in each map with three resources ($d = 3$).

	Reweighting time (s)			Lower bounding (s)		
Approach	NY	BAY	COL	NY	BAY	COL
Graph Reformulation (Johnson)	1.64	2.92	4.02	0.41	0.50	0.54
Modified Dijkstra	-	-	-	0.52	1.00	1.02

Table 3 The runtime statistics of the algorithms (in seconds) for $d = 2, 3$. For unsolved instances, the runtime is considered to be one hour. $|\mathcal{P}|$ denotes the number of instances solved within the timeout, and ϕ is the average slowdown factor of the mutually solved instances w.r.t. NWRCA*$_{\text{v1}}$.

			Search time (s)					Search time (s)			
Map	Algorithm	$\|\mathcal{P}\|$	Mean$_A$	Mean$_G$	Max.	ϕ	$\|\mathcal{P}\|$	Mean$_A$	Mean$_G$	Max.	ϕ
NY	NWRCA*$_{\text{v1}}$	100	**1.01**	**0.24**	**11.0**	-	100	**33.29**	**2.89**	**461.0**	-
	NWRCA*$_{\text{v2}}$	100	3.24	0.59	38.1	2.5	100	108.21	7.73	1607.3	2.5
	ERCA*	100	66.04	11.46	765.5	50.5	67	1513.00	217.35	3600.0	93.7
	RCBDA*	100	18.91	1.74	31.1	12.1	86	803.41	44.62	3600.0	44.8
BAY	NWRCA*$_{\text{v1}}$	100	**3.31**	**0.54**	**31.1**	-	100	**125.21**	**8.65**	**1803.5**	-
	NWRCA*$_{\text{v2}}$	100	10.20	1.31	124.6	2.4	96	335.69	22.84	3600.0	2.5
	ERCA*	95	306.88	27.28	3600.0	50.0	54	1922.49	479.12	3600.0	98.9
	RCBDA*	97	208.72	6.41	3600.0	28.8	73	1320.69	136.40	3600.0	122.0
COL	NWRCA*$_{\text{v1}}$	100	**2.14**	**0.50**	**21.9**	-	100	**67.92**	**7.70**	**720.7**	-
	NWRCA*$_{\text{v2}}$	100	6.18	1.22	61.3	2.4	100	201.75	19.63	1797.5	2.4
	ERCA*	97	204.77	26.46	3600.0	52.3	60	1779.38	468.34	3600.0	96.0
	RCBDA*	96	297.38	9.68	3600.0	379.0	72	1346.47	169.16	3600.0	112.4
		Instances with two resources $d = 2$					Instances with three resources $d = 3$				

Computation of heuristics. Our first analysis explores two approaches for computing heuristic functions for instances with three resources: (i) graph reformulation using Johnson's technique (as in ERCA*), and (ii) a modified version of Dijkstra's algorithm that allows for re-expansions. For the first approach, we used the improved Bellman-Ford algorithm [23, 8, 14] to reweight edge costs, followed by Dijkstra's to compute lower bounds. In case of RCBDA*, two rounds of lower-bound computation are required (one for the forward search and another for the backward search). Table 2 compares the average computation time for both approaches, assuming unidirectional heuristics are needed (as in ERCA* and NWRCA*). We observe that, the approach (i) requires a considerable amount of time to preprocess the graphs but enables faster computation of lower bounds, whereas the second approach, using modified Dijkstra's algorithm, delivers 3.5 to 4 times faster *overall* processing time without the need for reweighting in our (sparse) road networks. While approach (ii) can be advantageous in scenarios where the graph configuration changes between queries, it may be up to twice as slow compared to approach (i) when the search graph is static and already reweighted, though the difference is less than 0.5 seconds. Nonetheless, computing heuristic functions for NWRCA* using approach (ii) offers several benefits: i) it provides a simple yet efficient setup for constrained A* search with negative weights; ii) it eliminates the need for a time-intensive graph reformulation step, which is advantageous in dynamic scenarios; iii) it performs comparably to conventional methods when weights are non-negative (would be equivalent to standard Dijkstra).



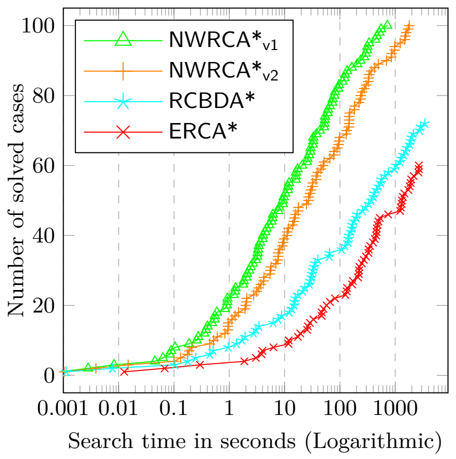


Figure 2 (Left) Runtime distribution of NWRCA*$_{v1}$ versus NWRCA*$_{v2}$ and the two other algorithms over all instances with $d = 3$. (Right) Algorithms' performance over the instances of the COL map with $d = 3$ (solved cases sorted based on runtime).

Algorithmic performance. Table 3 presents the experimental results for both scenarios ($d = 2, 3$). It displays the number of test cases successfully solved ($|\mathcal{P}|$), the arithmetic and geometric mean (Mean$_A$ and Mean$_G$) and maximum constrained runtime observed (in seconds) by each algorithm. To enable a fair comparison of actual search performance across algorithms, and since all algorithms use same heuristic functions, we exclude preprocessing times (graph reformulation and lower bounding) from our runtime analyses. Therefore, the runtimes reported in this table refer solely to resource-constrained search time. All algorithms solved their easiest instances in less than 0.1 seconds. For unsolved cases, the search time is recorded as the timeout. It can be observed that both variants of NWRCA* have successfully solved almost all instances for each map in both scenarios, except for the BAY map where NWRCA*$_{v2}$ leaves four instances unsolved. The outperformance is even more pronounced in the BAY and COL maps with $d = 3$, where ERCA* and RCBDA* could solve less than 75% of cases. In terms of computation time, both variants of NWRCA* deliver significantly better performance than ERCA* and RCBDA* in all metrics. The detailed results reveal that NWRCA*$_{v1}$ consistently outperforms other algorithms, including NWRCA*$_{v2}$, across nearly all instances, successfully solving its most challenging case in under 31 minutes.

The parameter ϕ in the last column of Table 3 presents the (arithmetic) average slowdown factor for each algorithm relative to the top-performing algorithm, NWRCA*$_{v1}$. This factor is calculated by comparing the search time of each mutually solved instance with the corresponding search time of NWRCA*$_{v1}$, highlighting the relative slowdown. We also show in Figure 2 (Left) the runtime distribution of NWRCA*$_{v1}$ against other algorithms over instances of all maps with three resources ($d = 3$). Based on the results, NWRCA*$_{v2}$ performs nearly 2.5 times slower than NWRCA*$_{v1}$. This difference in performance is primarily due to NWRCA*$_{v2}$'s costly queuing process, which requires lexicographical ordering of labels in $\mathcal{Q}$ (tie-breaking with **resources**), and NWRCA*$_{v1}$'s more efficient label pruning mechanism. This improved pruning efficiency, in particular, stems from the partial traversal of the $\mathcal{X}$ lists, where non-dominated labels are maintained in lexicographical order of their **resources**.

Larger slowdown factors are observed for ERCA* and RCBDA*, with both algorithms being outperformed by up to two orders of magnitude on average. The more than one order of magnitude performance gap between ERCA* and NWRCA* is primarily due to NWRCA*'s more efficient lazy pruning strategy and its simpler yet effective data structure for dominance checks, both contributing to significantly faster search times. In contrast, ERCA* suffers from a more rigid dominance pruning strategy that attempts to eliminate dominated and infeasible labels both at generation time and again upon extraction from the queue. While early dominance checks can help reduce queue size, they add computational overhead when applied to non-dominated labels. Furthermore, rechecking resource feasibility after extraction is redundant, as resource limits remain unchanged throughout the search, resulting in wasted computation and reduced efficiency.

Figure 2 (Right) also shows the number of solved cases for each algorithm, sorted by runtime, on the COL map with three resources. We observe that both variants of NWRCA* solve 60% of instances within 50 seconds, whereas ERCA* and RCBDA* require above 17 minutes to achieve the same success rate. In summary, both variants of NWRCA* demonstrate significantly faster performance than the state of the art in terms of computation time, with NWRCA*$_{\text{v1}}$ being approximately twice as fast as NWRCA*$_{\text{v2}}$.

ERCA* vs. RCBDA*. No performance comparison was provided between ERCA* and RCBDA* by the authors of ERCA*. Our results show that, while the improved RCBDA* algorithm solves more instances and achieves better average runtimes than ERCA* on certain maps, it does not consistently outperform ERCA*. The strong performance of RCBDA* in many instances is due to its bidirectional framework, which combines perimeter and alternating search strategies. In terms of worst-case runtime performance (Figure 2 (Left)), ERCA* generally performs better than RCBDA* but tends to be consistently around two orders of magnitude slower than NWRCA*$_{\text{v1}}$, whereas RCBDA* can be over three orders of magnitude slower than NWRCA*$_{\text{v1}}$ in some cases.

6 Conclusion

This paper introduced a novel, fast heuristic-guided label-setting approach with effective pruning mechanisms for efficiently solving resource-constrained shortest path (RCSP) problems in large-scale graphs. Unlike existing methods that aim to find a feasible solution, our framework identifies all non-dominated optimal solutions for a given RCSP instance. It also naturally handles instances with negative costs and resources, provided the cost heuristic is consistent and the search graph contains no negative cycles. We explored different configurations of the framework, distinguished by how heuristics are derived and how search labels are stored and explored during the search process. Experimental results highlight the importance of efficient ordering of labels in constrained search over large graphs and demonstrate the superiority of the proposed framework in solving challenging RCSP instances within tight time limits, outperforming state-of-the-art methods, including an enhanced version of the award-winning RCBDA* algorithm, by up to two orders of magnitude on average.

References

1 Saman Ahmadi, Nathan R. Sturtevant, Daniel Harabor, and Mahdi Jalili. Exact multi-objective path finding with negative weights. In Sara Bernardini and Christian Muise, editors, *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*, pages 11–19. AAAI Press, 2024. doi:10.1609/ICAPS.V34I1.31455.

2 Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. Bi-objective search with bi-directional A*. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 3:1–3:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.3.

3 Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. 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*, pages 12217–12224. AAAI Press, 2021. doi:10.1609/AAAI.V35I14.17450.

4 Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. Vehicle dynamics in pickup-and-delivery problems using electric vehicles. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CP.2021.11.

5 Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. Weight constrained path finding with bidirectional A*. In Lukás Chrpa and Alessandro Saetti, editors, *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria, July 21-23, 2022*, pages 2–10. AAAI Press, 2022. doi:10.1609/SOCS.V15I1.21746.

6 Saman Ahmadi, Guido Tack, Daniel Harabor, Philip Kilby, and Mahdi Jalili. Enhanced methods for the weight constrained shortest path problem. *Networks*, 84(1):3–30, 2024. doi:10.1002/net.22210.

7 Saman Ahmadi, Guido Tack, Daniel Harabor, Philip Kilby, and Mahdi Jalili. Real-time energy-optimal path planning for electric vehicles. *arXiv preprint arXiv:2411.12964*, 2024. doi:10.48550/arXiv.2411.12964.

8 Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

9 Manuel A. Bolívar, Leonardo Lozano, and Andrés L. Medaglia. Acceleration strategies for the weight constrained shortest path problem with replenishment. *Optim. Lett.*, 8(8):2155–2172, 2014. doi:10.1007/s11590-014-0742-x.

10 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. doi:10.1002/net.21960.

11 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.

12 Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004. doi:10.1002/NET.20033.

13 Daniele Ferone, Paola Festa, Serena Fugaro, and Tommaso Pastore. On the shortest path problems with edge constraints. In *22nd International Conference on Transparent Optical Networks, ICTON 2020, Bari, Italy, July 19-23, 2020*, pages 1–4. IEEE, 2020. doi:10.1109/ICTON51198.2020.9203378.

14 Lester R Ford Jr. Network flow theory. Technical report, Rand Corp Santa Monica Ca, 1956.

15 Bruce L. Golden and Douglas R. Shier. 2019–2020 glover-klingman prize winners. *Networks*, 2021. doi:10.1002/net.22072.

16 Gabriel Y. Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980. doi:10.1002/net.3230100403.

17 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.

18 Carlos Hernández, William Yeoh, Jorge A Baier, Ariel Felner, Oren Salzman, Han Zhang, Shao-Hung Chan, and Sven Koenig. Multi-objective search via lazy and efficient dominance checks. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 7223–7230, 2023.

19 Gerhard Hiermann, Jakob Puchinger, Stefan Ropke, and Richard F. Hartl. The electric fleet size and mix vehicle routing problem with time windows and recharging stations. *Eur. J. Oper. Res.*, 252(3):995–1018, 2016. doi:10.1016/J.EJOR.2016.01.038.

20 Donald B Johnson. A note on dijkstra's shortest path algorithm. *Journal of the ACM (JACM)*, 20(3):385–388, 1973. doi:10.1145/321765.321768.

21 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. `doi:10.1145/321992.321993`.

22 Leonardo Lozano and Andrés L. Medaglia. On an exact method for the constrained shortest path problem. *Comput. Oper. Res.*, 40(1):378–384, 2013. `doi:10.1016/j.cor.2012.07.008`.

23 Edward F. Moore. The shortest path through a maze. In *Proc. of the International Symposium on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.

24 Keisuke Otaki, Tomosuke Maeda, Takayoshi Yoshimura, and Hiroyuki Sakai. Roaming navigation: Diverse constrained paths using heuristic search. *IEEE Access*, 11:75617–75627, 2023. `doi:10.1109/ACCESS.2023.3295830`.

25 Luigi Di Puglia Pugliese and Francesca Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013. `doi:10.1002/net.21511`.

26 Francisco Javier Pulido, Lawrence Mandow, and José-Luis Pérez-de-la-Cruz. Dimensionality reduction in multiobjective shortest path search. *Comput. Oper. Res.*, 64:60–70, 2015. `doi:10.1016/j.cor.2015.05.007`.

27 Zhongqiang Ren, Zachary B Rubinstein, Stephen F Smith, Sivakumar Rathinam, and Howie Choset. ERCA*: A new approach for the resource constrained shortest path problem. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

28 Zhongqiang Ren, Richard Zhan, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Enhanced multi-objective a* using balanced binary search trees. In Lukás Chrpa and Alessandro Saetti, editors, *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria, July 21-23, 2022*, pages 162–170. AAAI Press, 2022. `doi:10.1609/SOCS.V15I1.21764`.

29 Giovanni Righini and Matteo Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discret. Optim.*, 3(3):255–273, 2006. `doi:10.1016/j.disopt.2006.05.007`.

30 Antonio Sedeño-Noda and Sergio Alonso-Rodríguez. An enhanced K-SP algorithm with pruning strategies to solve the constrained shortest path problem. *Appl. Math. Comput.*, 265:602–618, 2015. `doi:10.1016/j.amc.2015.05.109`.

31 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. `doi:10.1002/net.21856`.

32 Carlos Hernández Ulloa, William Yeoh, Jorge A. Baier, Han Zhang, Luis Suazo, and Sven Koenig. A simple and fast bi-objective search algorithm. In J. Christopher Beck, Olivier Buffet, Jörg Hoffmann, Erez Karpas, and Shirin Sohrabi, editors, *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pages 143–151. AAAI Press, 2020. URL: `https://aaai.org/ojs/index.php/ICAPS/article/view/6655`.

33 Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *Eur. J. Oper. Res.*, 209(1):1–10, 2011. `doi:10.1016/J.EJOR.2010.03.045`.